



# Red Hat Enterprise Linux 7

## 仮想化の導入および管理ガイド

Red Hat Enterprise Linux 物理マシン上での仮想マシンのインストール、設定および  
管理



# Red Hat Enterprise Linux 7 仮想化の導入および管理ガイド

---

Red Hat Enterprise Linux 物理マシン上での仮想マシンのインストール、設定および管理

Jiri Herrmann

Red Hat Customer Content Services

[jherrman@redhat.com](mailto:jherrman@redhat.com)

Yehuda Zimmerman

Red Hat Customer Content Services

[yzimmerm@redhat.com](mailto:yzimmerm@redhat.com)

Laura Novich

Red Hat Customer Content Services

Dayle Parker

Red Hat Customer Content Services

Scott Radvan

Red Hat Customer Content Services

Tahlia Richardson

Red Hat Customer Content Services

## 法律上の通知

Copyright © 2017 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

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, 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 Software Collections 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

本書では、Red Hat Enterprise Linux 7 ホスト物理マシンを設定する方法、および KVM ハイパーバイザーを使ってゲスト仮想マシンをインストールし、設定する方法について説明します。さらに、PCI デバイス設定、SR-IOV、ネットワーク設定、ストレージ、デバイスおよびゲスト仮想管理、さらにトラブルシューティング、互換性および制限について説明します。本書で説明されているすべての手順は、特に指定がない限りホストマシンで実行されることが想定されています。ゲスト仮想マシンで実行される必要のある手順についてその旨が明確に表示されます。Red Hat が提供する仮想化ソリューションの概説については、『Red Hat Enterprise Linux 7 仮想化スタートガイド』を参照してください。この点についてさらにお知りになりたい場合は、Red Hat Virtualization (RH318) トレーニングコースをご利用いただけます。



## 目次

パート I. デプロイメント .....	8
第1章 システム要件 .....	9
1.1. ホストのシステム要件 .....	9
1.2. KVM ハイパーバイザーの要件 .....	10
1.3. KVM ゲスト仮想マシンの互換性 .....	11
1.4. ゲストのサポート対象 CPU モデル .....	11
第2章 仮想化パッケージのインストール .....	13
2.1. RED HAT ENTERPRISE LINUX インストール時の仮想化パッケージのインストール .....	13
2.2. 既存の RED HAT ENTERPRISE LINUX システム上への仮想化パッケージのインストール .....	16
第3章 仮想マシンの作成 .....	19
3.1. ゲスト仮想マシンのデプロイに関する留意事項 .....	19
3.2. VIRT-INSTALL を使用したゲストの作成 .....	19
3.3. VIRT-MANAGER を使用したゲストの作成 .....	24
3.4. VIRT-INSTALL および VIRT-MANAGER のインストールオプションの比較 .....	35
第4章 仮想マシンのクローン作成 .....	37
4.1. 仮想マシンのクローン作成準備 .....	37
4.2. 仮想マシンのクローン作成 .....	40
第5章 KVM 準仮想化 (VIRTIO) ドライバー .....	45
5.1. 既存デバイスでの KVM VIRTIO ドライバーの使用 .....	45
5.2. KVM VIRTIO ドライバーを使用した新規デバイスの作成 .....	46
5.3. GPU デバイスに KVM VIRTIO ドライバーを使用 .....	50
第6章 ネットワーク設定 .....	52
6.1. LIBVIRT を使用した NETWORK ADDRESS TRANSLATION (NAT) .....	52
6.2. VHOST-NET の無効化 .....	53
6.3. VHOST-NET ZERO-COPY の有効化 .....	54
6.4. ブリッジネットワーク .....	54
第7章 KVM でのオーバーコミット .....	59
7.1. はじめに .....	59
7.2. メモリーのオーバーコミット .....	59
7.3. 仮想化 CPU のオーバーコミット .....	60
第8章 KVM ゲストのタイミング管理 .....	61
8.1. RED HAT ENTERPRISE LINUX ゲストに必要な時刻管理パラメーター .....	62
8.2. スチールタイムアカウンティング .....	64
第9章 LIBVIRT を使用したネットワークブート .....	65
9.1. ブートサーバーの準備 .....	65
9.2. PXE を使用したゲストの起動 .....	66
第10章 ハイパーバイザーおよび仮想マシンの登録 .....	68
10.1. ホスト物理マシンへの VIRT-WHO のインストール .....	68
10.2. 新規ゲスト仮想マシンの登録 .....	71
10.3. ゲスト仮想マシンのエントリーの削除 .....	71
10.4. VIRT-WHO の手動インストール .....	71
10.5. VIRT-WHO のトラブルシューティング .....	72
第11章 QEMU ゲストエージェントおよび SPICE エージェントによる仮想化の強化 .....	74
11.1. QEMU ゲストエージェント .....	74

11.2. LIBVIRT による QEMU ゲストエージェントの使用	78
11.3. SPICE エージェント	80
<b>第12章 仮想化のネスト</b>	<b>84</b>
12.1. 概要	84
12.2. 設定	84
12.3. 制約および制限事項	86
<b>パート II. 管理</b>	<b>87</b>
<b>第13章 ストレージプール</b>	<b>88</b>
13.1. ディスクベースのストレージプール	90
13.2. パーティションベースのストレージプール	94
13.3. ディレクトリーベースのストレージプール	101
13.4. LVM ベースのストレージプール	108
13.5. ISCSI ベースのストレージプール	116
13.6. NFS ベースのストレージプール	129
13.7. SCSI デバイスでの NPIV 仮想アダプター (VHBA) の使用	136
13.8. GLUSTERFS ストレージプール	141
<b>第14章 ストレージボリューム</b>	<b>144</b>
14.1. はじめに	144
14.2. ボリュームの作成	145
14.3. ボリュームのクローン作成	146
14.4. ボリュームの削除と消去	147
14.5. ストレージデバイスのゲストへの追加	147
<b>第15章 QEMU-IMG の使用</b>	<b>159</b>
15.1. ディスクイメージの検査	159
15.2. 変更のイメージへのコミット	159
15.3. イメージの比較	159
15.4. イメージのマッピング	160
15.5. イメージの変更	161
15.6. 既存イメージの別形式への変換	161
15.7. 新規イメージまたはデバイスの作成およびフォーマット	161
15.8. イメージ情報の表示	162
15.9. イメージのバックアップファイルの再設定	162
15.10. ディスクイメージのサイズ変更	163
15.11. スナップショットの一覧表示、作成、適用および削除	163
15.12. QEMU-IMG でサポートされる形式	164
<b>第16章 KVM の移行</b>	<b>166</b>
16.1. 移行の定義と利点	166
16.2. 移行の要件と制限事項	166
16.3. ライブマイグレーションと RED HAT ENTERPRISE LINUX バージョンの互換性	168
16.4. 共有ストレージの例: 単純な移行に NFS を使用する	169
16.5. VIRSH を使用した KVM のライブマイグレーション	170
16.6. VIRT-MANAGER を使用した移行	175
<b>第17章 ゲスト仮想マシンデバイスの設定</b>	<b>180</b>
17.1. PCI デバイス	180
17.2. PCI デバイス割り当てと SR-IOV デバイス	194
17.3. USB デバイス	206
17.4. デバイスコントローラーの設定	207
17.5. デバイスのアドレス設定	211

17.6. 乱数ジェネレーターデバイス	213
17.7. GPU デバイスの割り当て	215
<b>第18章 仮想ネットワークの構築</b>	<b>221</b>
18.1. 仮想ネットワークのスイッチ	221
18.2. ブリッジモード	221
18.3. NETWORK ADDRESS TRANSLATION	222
18.4. DNS と DHCP	223
18.5. ルーティングモード	224
18.6. 隔離モード	224
18.7. デフォルト設定	225
18.8. 一般的な事例	226
18.9. 仮想ネットワークの管理	227
18.10. 仮想ネットワークの作成	228
18.11. 仮想ネットワークのゲストへの接続	238
18.12. 仮想 NIC を直接物理インターフェースへ接続	240
18.13. 仮想 NIC に接続しているネットワークブリッジまたはホスト物理マシンの動的な変更	244
18.14. ネットワークのフィルター機能の適用	245
18.15. トンネルの作成	277
18.16. VLAN タグの設定	278
18.17. QOS の仮想ネットワークへの適用	279
<b>第19章 ゲストのリモート管理</b>	<b>280</b>
19.1. トランスポートモード	280
19.2. SSH によるリモート管理	283
19.3. TLS と SSL 経由のリモート管理	285
19.4. VNC サーバーの設定	288
19.5. NSS による仮想マシンのリモート管理の強化	288
<b>第20章 仮想マシンマネージャー (VIRT-MANAGER) を使用したゲストの管理</b>	<b>290</b>
20.1. VIRT-MANAGER の起動	290
20.2. 仮想マシンマネージャーのメインウィンドウ	291
20.3. 仮想ハードウェアの詳細ウィンドウ	292
20.4. 仮想マシンのグラフィカルコンソール	297
20.5. リモート接続の追加	299
20.6. ゲストの詳細の表示	300
20.7. スナップショットの管理	307
<b>第21章 VIRSH を使用したゲスト仮想マシンの管理</b>	<b>311</b>
21.1. ゲスト仮想マシンの状態および種類	311
21.2. VIRSH バージョンの表示	312
21.3. ECHO を使用したコマンドの送信	312
21.4. VIRSH CONNECT を使用したハイパーバイザーへの接続	312
21.5. ゲスト仮想マシンとハイパーバイザーについての情報表示	313
21.6. 仮想マシンの起動、再開および復元	314
21.7. 仮想マシン設定の管理	316
21.8. ゲスト仮想マシンのシャットオフ、シャットダウン、再起動および強制終了	319
21.9. 仮想マシンの削除	320
21.10. ゲスト仮想マシンのシリアルコンソールの接続	322
21.11. NMI (マスク不可能な割り込み) の挿入	322
21.12. ゲスト仮想マシン情報の取得	322
21.13. スナップショットの使用	328
21.14. グラフィカル表示に接続するための URI を表示	331
21.15. VNC ディスプレイの IP アドレスとポート番号の表示	332

21.16. 使用されていないブロックの破棄	332
21.17. ゲスト仮想マシンの検索コマンド	332
21.18. QEMU 引数のドメイン XML への変換	335
21.19. VIRSH DUMP を使ったゲスト仮想マシンのコアのダンプファイルの作成	336
21.20. 仮想マシンの XML ダンプの作成 (設定ファイル)	337
21.21. 設定ファイルでのゲスト仮想マシンの作成	338
21.22. ゲスト仮想マシンの XML 設定の編集	338
21.23. 多機能 PCI デバイスの KVM ゲスト仮想マシンへの追加	338
21.24. 指定されたゲスト仮想マシンの CPU 統計の表示	340
21.25. ゲストコンソールのスクリーンショット	340
21.26. キー入力の組み合わせの指定されたゲスト仮想マシンへの送信	341
21.27. ホストマシンの管理	342
21.28. ゲスト仮想マシン情報の取得	350
21.29. ストレージプールコマンド	351
21.30. ストレージボリュームコマンド	359
21.31. ストレージボリュームの削除	361
21.32. ストレージボリュームの内容の削除	362
21.33. ストレージボリューム情報の XML ファイルへのダンプ	363
21.34. ボリューム情報の一覧表示	363
21.35. ストレージボリューム情報の取得	363
21.36. ストレージボリュームのアップロードおよびダウンロード	364
21.37. ストレージボリュームのサイズ変更	364
21.38. ゲスト仮想マシン別の情報の表示	365
21.39. 仮想ネットワークの管理	371
21.40. インターフェースコマンド	377
21.41. スナップショットの管理	379
21.42. ゲスト仮想マシンの CPU モデルの設定	386
21.43. ゲスト仮想マシンの CPU モデルの設定	390
21.44. ゲスト仮想マシンのリソースの管理	391
21.45. スケジュールパラメーターの設定	392
21.46. ディスク I/O スロットリング	393
21.47. ブロック I/O パラメーターの表示または設定	393
21.48. メモリーチューニングの設定	394
<b>第22章 オフラインツールを使用したゲスト仮想マシンディスクへのアクセス .....</b>	<b>395</b>
22.1. はじめに	395
22.2. 用語について	397
22.3. インストール	397
22.4. GUESTFISH シェル	397
22.5. その他のコマンド	402
22.6. VIRT-RESCUE: レスキューシェル	403
22.7. VIRT-DF: ディスク使用量の監視	404
22.8. VIRT-RESIZE: オフラインのゲスト仮想マシンのサイズ変更	405
22.9. VIRT-INSPECTOR: ゲスト仮想マシンの検査	407
22.10. プログラミング言語での API の使用	409
22.11. VIRT-SYSPREP: 仮想マシン設定のリセット	414
22.12. VIRT-CUSTOMIZE: 仮想マシン設定のカスタマイズ	417
22.13. VIRT-DIFF: 仮想マシンファイル間の相違点の一覧表示	421
22.14. VIRT-SPARSIFY: 空のディスク領域の再要求	424
<b>第23章 ゲスト仮想マシン管理の汎用ユーザーインターフェースツール .....</b>	<b>429</b>
23.1. VIRT-VIEWER	429
23.2. REMOTE-VIEWER	431

23.3. GNOME BOXES	433
<b>第24章 ドメイン XML の操作</b>	<b>439</b>
24.1. 一般的な情報およびメタデータ	439
24.2. オペレーティングシステムの起動	440
24.3. SMBIOS システム情報	443
24.4. CPU の割り当て	444
24.5. CPU のチューニング	444
24.6. メモリーのバッキング	446
24.7. メモリーチューニング	447
24.8. メモリーの割り当て	448
24.9. NUMA ノードのチューニング	449
24.10. ブロック I/O チューニング	450
24.11. リソースパーティション作成	451
24.12. CPU モデルおよびトポロジー	451
24.13. イベント設定	458
24.14. 電力管理	460
24.15. ハイパーバイザーの機能	460
24.16. 時間管理	461
24.17. TIMER 要素属性	465
24.18. DEVICES	466
24.19. ストレージプール	521
24.20. ストレージボリューム	527
24.21. セキュリティーラベル	532
24.22. サンプル設定ファイル	534
<b>パート III. 付録</b>	<b>535</b>
<b>付録A トラブルシューティング</b>	<b>536</b>
A.1. デバッグおよびトラブルシューティングのツール	536
A.2. 障害復旧に備える	537
A.3. ダンプファイルの作成	539
A.4. SYSTEMTAP フライトレコーダーを使用して継続的にトレースデータを取得する	540
A.5. KVM_STAT	541
A.6. シリアルコンソールでのトラブルシューティング	545
A.7. 仮想化ログ	546
A.8. ループデバイスエラー	547
A.9. ライブマイグレーションに関するエラー	547
A.10. BIOS での INTEL VT-X と AMD-V の仮想化ハードウェア拡張の有効化	547
A.11. RED HAT ENTERPRISE LINUX 7 ホスト上での RED HAT ENTERPRISE LINUX 6 ゲストのシャットダウン	549
A.12. 正常なシャットダウンに向けたオプションの方法	551
A.13. KVM ネットワークのパフォーマンス	553
A.14. LIBVIRT による外部スナップショットの作成方法	555
A.15. 日本語キーボードのゲストコンソールで欠如している文字	556
A.16. ゲスト仮想マシンのシャットダウンの失敗	556
A.17. ゲスト仮想マシンの SMART ディスクモニタリングの無効化	557
A.18. LIBGUESTFS のトラブルシューティング	557
A.19. SR-IOV のトラブルシューティング	557
A.20. 一般的な LIBVIRT エラーおよびトラブルシューティング	558
<b>付録B 仮想化の制限</b>	<b>586</b>
B.1. サポート制限	586
B.2. KVM の制限	586

B.3. アプリケーションの制限	589
B.4. その他の制限	590
B.5. ストレージ対応	590
B.6. USB 3 / XHCI サポート	590
<b>付録C その他のリソース</b> .....	<b>591</b>
C.1. オンラインリソース	591
C.2. インストールされているドキュメント	591
<b>付録D IOMMU グループの使用[1]</b> .....	<b>592</b>
D.1. IOMMU の概要	592
D.2. IOMMU グループの詳細	593
D.3. IOMMU グループの特定および割り当て方法	594
D.4. IOMMU ストラテジーおよびユースケース	596
<b>付録E 改訂履歴</b> .....	<b>598</b>



## パート I. デプロイメント



# 第1章 システム要件

仮想化は、Intel 64 および AMD64 アーキテクチャーをベースにした Red Hat Enterprise Linux 7 の KVM ハイパーバイザーで利用できます。本章では、実行中の仮想マシン (VM と呼ばれる) のシステム要件の一覧を記載します。

仮想化パッケージのインストールに関する詳細は、「[2章 仮想化パッケージのインストール](#)」を参照してください。

## 1.1. ホストのシステム要件

### 最小ホストシステム要件

- 6 GB の空きディスク領域
- 2 GB の RAM

### 推奨されるシステム要件

- 各仮想化 CPU に 1 つのコアまたはスレッド、およびホストに 1 つのコアまたはスレッド
- 2 GB の RAM と仮想マシン用の追加の RAM
- ホスト用に 6 GB のディスク領域、および仮想マシンに必要なディスク領域

ほとんどのゲストオペレーティングシステムは少なくとも 6GB のディスク領域を必要とします。各ゲストの追加のストレージ領域はワークロードによって異なります。

### swap 領域

Linux の swap 領域は、物理メモリー (RAM) の空き容量がなくなると使用されます。システムがより多くのメモリーリソースを必要とする場合、RAM に空き容量がないとメモリーの非アクティブなページが swap 領域に移されます。RAM の容量が小さいマシンでは swap 領域が役に立ちますが、容量の大きい RAM の代わりとしてはなりません。swap 領域は、物理メモリーよりもアクセスに時間がかかるハードドライブ上にあります。swap パーティションのサイズは、ホストの物理 RAM から計算できます。Red Hat カスタマーポータルには swap パーティションのサイズを安全かつ効果的に判別する方法についての記事が記載されています:

<https://access.redhat.com/ja/solutions/108483>

- raw イメージファイルを使用する場合、必要なディスク領域の合計はイメージファイルで必要な領域、ホストオペレーティングシステムに必要な 6 GB の領域、およびゲストの swap 領域の合計と同等またはそれ以上となります。

#### 式1.1 raw イメージを使用するゲスト仮想マシンに必要な領域の計算

$$\text{total for raw format} = \text{images} + \text{hostspace} + \text{swap}$$

qcow イメージの場合、qcow および qcow2 イメージは必要に応じて拡大することから、ゲストの予測される最大ストレージ要件 (**total for qcow format**) も計算する必要があります。この拡大を可能にするには、まずゲストの予測される最大ストレージ要件 (**expected maximum guest storage**) を 1.01 倍し、これにホスト (**host**) で必要な領域と必要なスワップ領域 (**swap**) を加えます。

#### 式1.2 qcow イメージを使用するゲスト仮想マシンに必要な領域の計算

$\text{total for qcow format} = (\text{expected maximum guest storage} * 1.01) + \text{host} + \text{swap}$

ゲスト仮想マシン要件は、さらに「[7章KVMでのオーバーコミット](#)」で説明されています。

## 1.2. KVM ハイパーバイザーの要件

KVM ハイパーバイザーには、以下が必要です。

- x86 ベースシステム向けの Intel VT-x および Intel 64 仮想化拡張機能を備えた Intel プロセッサー
- AMD-V および AMD64 仮想化拡張機能を備えた AMD プロセッサー

仮想化拡張機能 (Intel VT-x または AMD-V) は完全仮想化に必要です。以下のコマンドを入力し、システムにハードウェアの仮想化拡張機能があるかどうかや、それらが有効にされていることを判別します。

### 手順1.1 仮想化拡張機能の確認

#### 1. 利用できる CPU 仮想化拡張機能を確認します。

次のコマンドを入力して利用できる CPU 仮想化拡張機能を確認します。

```
$ grep -E 'svm|vmx' /proc/cpuinfo
```

#### 2. 出力を分析します。

- 以下のサンプルには **vmx** エントリーが含まれています。これは、Intel VT-x 拡張機能を備えた Intel プロセッサーであることを示します。

```
flags      : fpu tsc msr pae mce cx8 vmx apic mtrr mca cmov pat
pse36 clflush
dts acpi mmx fxsr sse sse2 ss ht tm syscall lm constant_tsc pni
monitor ds_cpl
vmx est tm2 cx16 xtptr lahfh_lm
```

- 次のサンプルには **svm** エントリーが含まれています。これは、AMD-V 拡張機能を備えた AMD プロセッサーであることを示します。

```
flags      : fpu tsc msr pae mce cx8 apic mtrr mca cmov pat pse36
clflush
mmx fxsr sse sse2 ht syscall nx mmxext svm fxsr_opt lm 3dnowext
3dnow pni cx16
lahfh_lm cmp_legacy svm cr8legacy ts fid vid ttp tm stc
```

**grep -E 'svm|vmx' /proc/cpuinfo** コマンドが出力を返す場合、プロセッサーにはハードウェアの仮想化拡張機能が含まれます。場合によってはメーカーが BIOS で仮想化拡張機能を無効にすることがあります。拡張機能が表示されなかったり、完全仮想化が機能しない場合には、BIOS 設定ユーティリティで拡張機能を有効にする方法について、[手順A.4「BIOSでの仮想化拡張の有効化」](#)を参照してください。

#### 3. KVM カーネルモジュールが読み込まれていることを確認します。

追加のチェック事項として、以下のコマンドを使って **kvm** モジュールがカーネルに読み込まれていることを確認します。

■

```
# lsmod | grep kvm
```

出力に **kvm\_intel** または **kvm\_amd** が含まれている場合は、**kvm** ハードウェア仮想化モジュールが読み込まれています。



#### 注記

**virsh** ユーティリティー (**libvirt-client** パッケージで提供される) は、以下のコマンドを使ってシステムの仮想化機能の詳細一覧を出力します。

```
# virsh capabilities
```

## 1.3. KVM ゲスト仮想マシンの互換性

Red Hat Enterprise Linux 7 サーバーには、いくつかのサポート制限があります。

Red Hat Enterprise Linux のプロセッサおよびメモリー容量の制限についての情報は、以下の URL をご覧ください。

- ホストシステムについては、<https://access.redhat.com/articles/rhel-limits> を参照してください。
- KVM ハイパーバイザーについては、<https://access.redhat.com/ja/articles/1520293> を参照してください。

以下の URL は、Red Hat Enterprise Linux KVM ホストでの実行が認定されているゲストオペレーティングシステムの一覧を表示しています。

- <https://access.redhat.com/articles/973133>



#### 注記

KVM ハイパーバイザーの制限およびサポート上の制限についての詳細は、「[付録B 仮想化の制限](#)」を参照してください。

## 1.4. ゲストのサポート対象 CPU モデル

それぞれのハイパーバイザーには、ゲストにデフォルトで表示する CPU 機能に関して独自のポリシーがあります。ハイパーバイザーがゲストに表示する CPU 機能のセットはゲスト仮想マシン設定で選択される CPU モデルによって異なります。

### 1.4.1. ゲスト CPU モデルの一覧表示

アーキテクチャーの種類に対応する CPU モデルの詳細一覧を確認するには、**virsh cpu-models architecture** コマンドを実行します。以下は例になります。

```
$ virsh cpu-models x86_64
486
pentium
pentium2
pentium3
pentiumpro
coreduo
```

```
n270
core2duo
qemu32
kvm32
cpu64-rhel5
cpu64-rhel6
kvm64
qemu64
Conroe
Penryn
Nehalem
Westmere
SandyBridge
Haswell
athlon
phenom
Opteron_G1
Opteron_G2
Opteron_G3
Opteron_G4
Opteron_G5
```

```
$ virsh cpu-models ppc64
POWER7
POWER7_v2.1
POWER7_v2.3
POWER7+_v2.1
POWER8_v1.0
```

サポート対象の CPU モデルおよび機能は、`/usr/share/libvirt/`にある `cpu_map.xml` ファイルに記載されています。

```
# cat /usr/share/libvirt/cpu_map.xml
```

ゲストの CPU モデルおよび機能は、ドメイン XML ファイルの `<cpu>` セクションで変更できます。詳細は、「[CPU モデルおよびトポロジー](#)」を参照してください。

ホストのモデルは、必要に応じて指定された機能を使用するように設定できます。詳細は、「[指定 CPU に設定された機能の変更](#)」を参照してください。

## 第2章 仮想化パッケージのインストール

仮想化を使用するには、コンピューターに Red Hat 仮想化パッケージをインストールする必要があります。仮想化パッケージは、**yum** コマンドおよび **Subscription Manager** を使用して Red Hat Enterprise Linux のインストール時またはインストール後にインストールすることができます。

KVM ハイパーバイザーは、**kvm** カーネルモジュールを使うデフォルトの Red Hat Enterprise Linux カーネルを使用します。

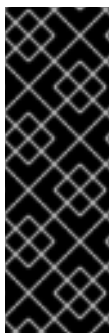
### 2.1. RED HAT ENTERPRISE LINUX インストール時の仮想化パッケージのインストール

このセクションでは、Red Hat Enterprise Linux のインストール時に仮想化パッケージをインストールする方法について説明します。



#### 注記

Red Hat Enterprise Linux のインストールについての詳細は、『[Red Hat Enterprise Linux 7 インストールガイド](#)』を参照してください。



#### 重要

Anaconda インターフェースは、Red Hat Enterprise Linux サーバーのインストール時に Red Hat 仮想化パッケージをインストールする方法を提供します。

Red Hat Enterprise Linux Workstation をインストールする場合、Red Hat 仮想化パッケージは Workstation のインストールの完了後にのみインストールできます。『[既存の Red Hat Enterprise Linux システム上への仮想化パッケージのインストール](#)』を参照してください。

#### 手順2.1 仮想化パッケージのインストール

1. ソフトウェアを選択します。  
インストール手順に従い、インストールの**概要**画面に移動します。

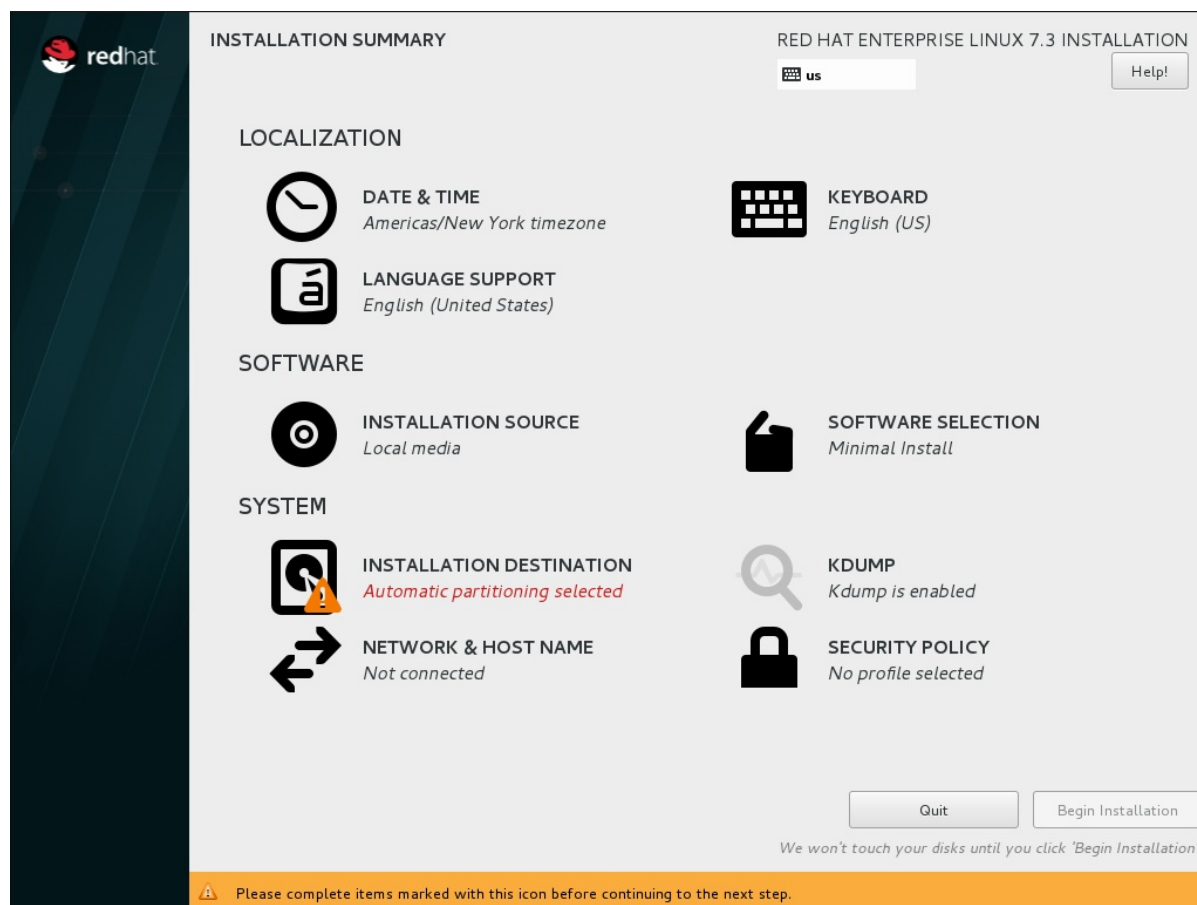


図2.1 「インストールの概要」画面

インストールの概要画面で、ソフトウェアの選択をクリックします。ソフトウェアの選択画面が開かれます。

## 2. サーバタイプとパッケージグループを選択します。

Red Hat Enterprise Linux 7は、基本的な仮想化パッケージまたはグラフィカルユーザーインターフェースでのゲスト管理を可能にするパッケージでインストールできます。以下のいずれかを実行します。

### ○ 最小仮想化ホストのインストール

**ベース環境** ペインの下に**仮想化ホスト** ラジオボタンを選択し、**選択した環境のアドオン** ペインの下に**Virtualization Platform (仮想化プラットフォーム)** チェックボックスを選択します。これにより、**virsh**を使用するか、またはネットワーク経由でリモートで実行できる基本的な仮想化環境がインストールされます。

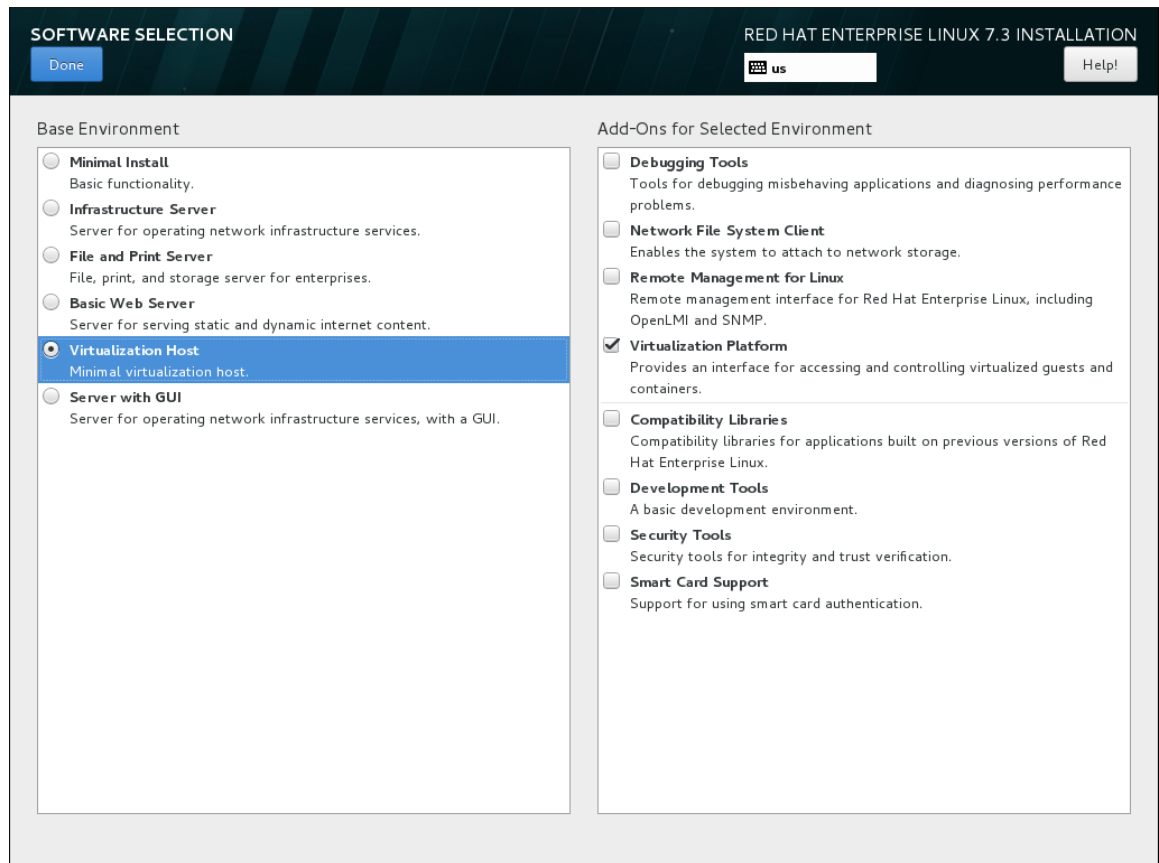


図2.2 「ソフトウェアの選択」画面で選択される仮想化ホスト

。グラフィカルユーザーインターフェースを使った仮想化ホストのインストール

ベース環境 ペインの **Server with GUI** (GUI のあるサーバー) ラジオボタンと、選択した環境のアドオン ペインにある **Virtualization Client** (仮想化クライアント)、**Virtualization Hypervisor** (仮想化ハイパーバイザー)、および **Virtualization Tools** (仮想化ツール) のチェックボックスを選択します。これにより、ゲスト仮想マシンをインストールし、管理するためのグラフィカルツールと共に、仮想化環境がインストールされます。



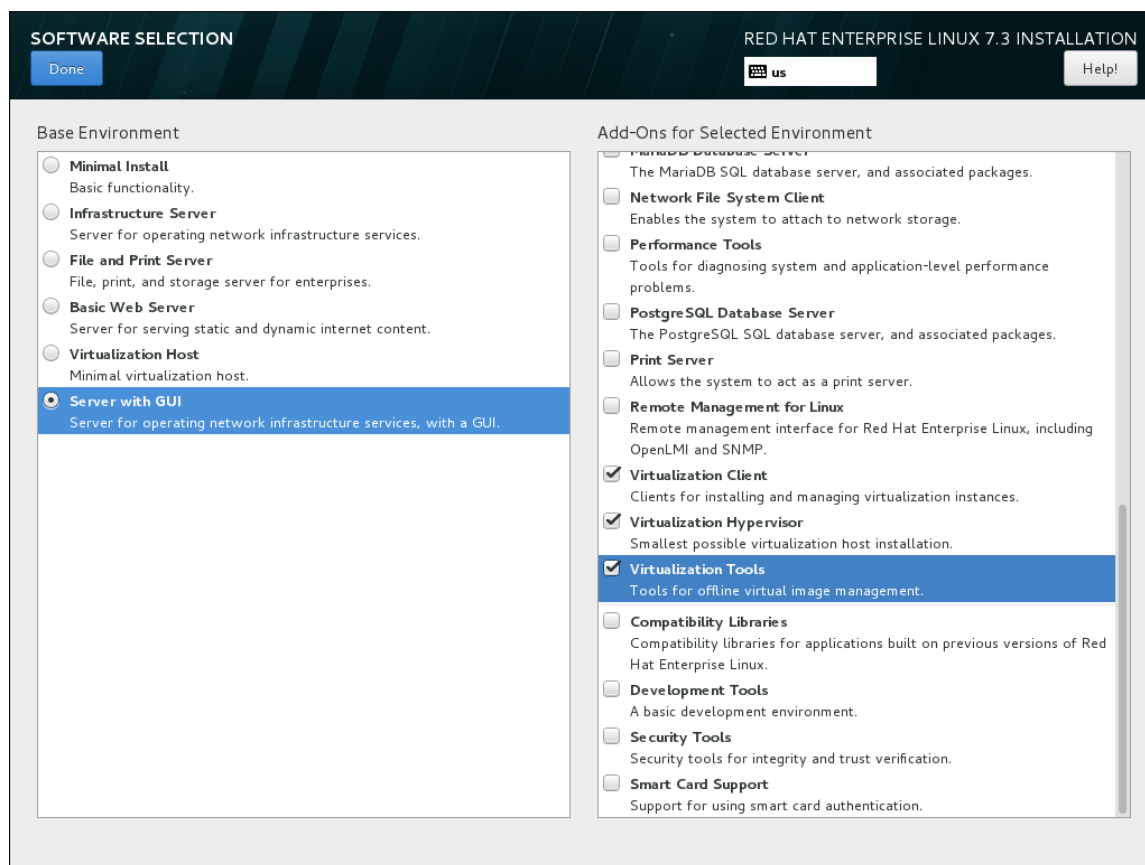


図2.3 「ソフトウェアの選択」画面で選択される Server with GUI

### 3. インストールを終了します。

**Done (完了)** をクリックしてインストールを継続します。



#### 重要

仮想化パッケージの更新を受信するには、有効な Red Hat Enterprise Linux サブスクリプションが必要です。

#### 2.1.1. キックスタータファイルによる KVM パッケージのインストール

キックスタータファイルを使用して、仮想化パッケージと共に Red Hat Enterprise Linux をインストールするには、キックスタータファイルの **%packages** セクションに以下のパッケージグループを追加します。

```
@virtualization-hypervisor
@virtualization-client
@virtualization-platform
@virtualization-tools
```

キックスタータファイルを使用したインストール方法についての詳細は、『[Red Hat Enterprise Linux 7 インストールガイド](#)』を参照してください。

## 2.2. 既存の RED HAT ENTERPRISE LINUX システム上への仮想化パッケージのインストール

このセクションでは、既存の Red Hat Enterprise Linux 7 システムに KVM ハイパーバイザーをインストールするステップを説明します。



パッケージをインストールするには、Red Hat カスタマーポータルにマシンを登録して、サブスクライブすることが必要です。Red Hat Subscription Manager で登録するには、**subscription-manager register** コマンドを実行して、プロンプトに従います。別の方法として、デスクトップのアプリケーション → システムツールから Red Hat Subscription Manager アプリケーションを実行して登録できます。

有効な Red Hat サブスクリプションがない場合には、[Red Hat のオンラインストア](https://access.redhat.com/solutions/253273)に移動してサブスクリプションを取得してください。Red Hat カスタマーポータルへの登録およびサブスクライブに関する詳しい情報は、<https://access.redhat.com/solutions/253273> を参照してください。

### 2.2.1. 仮想化パッケージの手動インストール

Red Hat Enterprise Linux で仮想化を使用するには、最低でも以下のパッケージをインストールする必要があります。

- **qemu-kvm**: このパッケージは、ユーザーレベルの KVM エミュレーターを使用し、ホストとゲスト仮想マシン間の通信を容易にします。
- **qemu-img**: このパッケージは、ゲスト仮想マシンのディスク管理を提供します。



#### 注記

**qemu-img** パッケージは **qemu-kvm** パッケージの依存関係としてインストールされます。

- **libvirt**: このパッケージは、ハイパーバイザーおよびホストシステムとの対話用にサーバーとホスト側のライブラリーを提供します。**libvirtd** デーモンは、ライブラリー呼び出しを処理し、仮想マシンを管理し、ハイパーバイザーを制御します。

これらのパッケージをインストールするには、以下のコマンドを入力します。

```
# yum install qemu-kvm libvirt
```

以下の仮想化管理パッケージも利用可能であり、仮想化の導入時に使用が推奨されています。

- **virt-install**: このパッケージは、コマンドラインからの仮想マシン作成用に**virt-install** コマンドを提供します。
- **libvirt-python**: このパッケージは、Python プログラミング言語で書かれているアプリケーションが libvirt API で供給されるインターフェースを使用できるようにするモジュールが含まれています。
- **virt-manager**: このパッケージは 仮想マシンマネージャーとしても知られる **virt-manager** ツールを提供します。これは、仮想マシンを管理するためのグラフィカルツールです。管理 API として **libvirt-client** ライブラリーを使用します。
- **libvirt-client**: このパッケージは、libvirt サーバーにアクセスするためのクライアント側の API とライブラリーを提供します。**libvirt-client** パッケージには、コマンドラインまたは特殊な仮想化シェルから仮想マシンとハイパーバイザーを管理し、制御する **virsh** コマンドラインツールが含まれます。

これらの推奨される仮想化パッケージすべては、以下のコマンドを使ってインストールしてください。

```
# yum install virt-install libvirt-python virt-manager virt-install
libvirt-client
```

### 2.2.2. 仮想化パッケージグループのインストール

仮想化パッケージは、パッケージグループからもインストールできます。以下の表は、仮想化パッケージグループとその役割について説明しています。

表2.1 仮想化パッケージグループ

パッケージグループ	説明	必須パッケージ	オプションパッケージ
<b>Virtualization Hypervisor</b>	仮想化ホストの最小インストール	libvirt、qemu-kvm、qemu-img	qemu-kvm-tools
<b>Virtualization Client</b>	仮想化インスタンスをインストールし、管理するためのクライアント	gnome-boxes、virt-install、virt-manager、virt-viewer、qemu-img	virt-top、libguestfs-tools、libguestfs-tools-c
<b>Virtualization Platform</b>	仮想マシンおよびコンテナにアクセスしたり制御したりするためのインターフェースを提供	libvirt、libvirt-client、virt-who、qemu-img	fence-virt-d-libvirt、fence-virt-d-multicast、fence-virt-d-serial、libvirt-cim、libvirt-java、libvirt-snmp、perl-Sys-Virt
<b>Virtualization Tools</b>	オフラインの仮想イメージを管理するためのツール	libguestfs、qemu-img	libguestfs-java、libguestfs-tools、libguestfs-tools-c

パッケージグループをインストールするには、**yum groupinstall *package\_group*** コマンドを実行します。**--optional** オプションを使用してパッケージグループのオプションパッケージをインストールします。たとえば、すべてのオプションパッケージを含む **Virtualization Tools** パッケージグループをインストールするには、以下を実行します。

```
# yum groupinstall "Virtualization Tools" --optional
```

## 第3章 仮想マシンの作成

仮想化パッケージを Red Hat Enterprise Linux 7 ホストシステムにインストールすると、**virt-manager** インターフェースを使って仮想マシンの作成やゲストオペレーティングシステムのインストールが可能になります。あるいは、**virt-install** コマンドラインユーティリティを使うこともできます。後者の場合は、パラメーターの一覧またはスクリプトを使用します。本章では、これら 2 つの方法について説明します。

### 3.1. ゲスト仮想マシンのデプロイに関する留意事項

ゲスト仮想マシンを作成する前には、様々な要素を考慮する必要があります。デプロイ前に仮想マシンの役割を評価する必要がありますが、定期的なモニタリングと（負荷やクライアント数などの）様々な要素に基づく評価も実行する必要があります。考慮に入れる要素には、以下が含まれます。

#### パフォーマンス

ゲスト仮想マシンは、意図されるタスクに基づいてデプロイし、設定することをお勧めします。ゲストシステムには、パフォーマンスについての特別な考慮が必要なものもあります（例：データベースサーバーを実行するゲスト）。ゲストの役割や予測されるシステム負荷によっては、ゲストがより多くの割り当て CPU やメモリーを必要とすることもあります。

#### 入出力要件と入出力タイプ

ゲスト仮想マシンの中には、とくに高い入出力要件があるものや、入出力タイプ（例：通常のディスクブロックサイズのアクセスまたはクライアント数）に応じてさらに注意や追加の予測が必要なものもあります。

#### ストレージ

ゲスト仮想マシンの中には、ストレージへの優先的なアクセスやより高速なディスクタイプを必要とするものもあります。また、ストレージ領域への排他的なアクセスを必要とする場合もあります。ゲストが使用するストレージ容量を定期的に監視し、ストレージのデプロイやメンテナンス時にその結果を考慮することをお勧めします。『[Red Hat Enterprise Linux 7 仮想化セキュリティガイド](#)』に説明されているすべての留意事項を確認してください。また、お使いの物理ストレージによって仮想ストレージのオプションが制限される可能性があることについても留意しておくことは重要です。

#### ネットワークおよびネットワークインフラストラクチャー

ゲスト仮想マシンの環境によっては、他のゲストより高速のネットワークリンクを必要とするものもあります。帯域幅や待ち時間はゲストのデプロイおよびメンテナンス時に考慮すべき要素になることが多く、要件や負荷が変化する場合はとくにそう言えます。

#### リクエスト要件

SCSI リクエストは、**virtio** ドライブがディスク全体をベースとし、以下の例のように [ドメイン XML ファイル](#) でディスクデバイスパラメーターが **lun** に設定されている場合にのみ、**virtio** ドライブ上のゲスト仮想マシンに発行されます。

```
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
  <disk type='block' device='lun'>
```

### 3.2. VIRT-INSTALL を使用したゲストの作成

コマンドラインから **virt-install** コマンドを使って、仮想マシンを作成してその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ことができます。**virt-install** は、対話形式で、または仮想マシンを自動作成するスクリプトの一部として使用することができます。対話型のグラフィカルインストールを行う場合は、**virt-install** を実行する前に **virt-viewer** をインストールしておく必要があります。さらに、**virt-install** をキックスタータファイルと共に使用すると、仮想マシンのオペレーティングシステムを無人でインストールすることができます。



### 注記

一部の **virt-install** コマンドでは、正しく完了するのに **root** 権限が必要となる場合があります。

**virt-install** ユーティリティーには、さまざまなコマンドラインオプションが使われます。ただし、ほとんどの **virt-install** オプションは使う必要がありません。

ゲスト仮想マシンのインストールに必要な主なオプションは、以下のとおりです。

#### **--name**

仮想マシンの名前

#### **--memory**

ゲストに割り当てられたメモリー (RAM) の容量 (MiB 単位)

#### ゲストストレージ

以下に示すゲストストレージオプションのどちらかを使います。

- **--disk**

仮想マシンのストレージ設定の詳細を指定します。**--disk none** オプションを使うと、ディスク領域を確保せずに仮想マシンが作成されます

- **--filesystem**

ゲスト仮想マシン用ファイルシステムへのパスを指定します

#### インストール方法

以下に示すインストール方法のいずれかを使います。

- **--location**

インストールメディアの場所を指定します

- **--cdrom**

仮想 CD-ROM デバイスとして使われるファイルまたはデバイスを指定します。ISO イメージへのパスまたは CD-ROM デバイスへのパスのどちらかを指定することができます。最小限の起動用 ISO イメージを取得またはアクセスする URL を指定することもできます

- **--pxe**

ゲストのインストールプロセスを開始する際、初期 **ramdisk** およびカーネルを読み込むのに PXE ブートプロトコルを使います

- **--import**

OS のインストールプロセスをスキップし、既存のディスクイメージを利用してゲストを構築します。ブートに使用するデバイスは、**disk** または **filesystem** オプションで最初に指定するデバイスです

- **--boot**

インストール後の VM ブート設定を指定します。このオプションでは、ブートデバイスの順序を指定し、オプションのカーネル引数を使ってカーネルおよび **initrd** を永続的にブートオフし、BIOS ブートメニューを有効にすることが可能です

オプションの全リストを表示するには、以下のコマンドを入力します。

```
# virt-install --help
```

オプションの属性の全リストを表示するには、以下のコマンドを入力します。

```
# virt install --option=?
```

**virt-install** の **man** ページも、各コマンドのオプション、重要な変数および例について記載しています。

**virt-install** を実行する前に、**qemu-img** を使ってストレージのオプションも設定しなければならない場合があります。**qemu-img** を使用方法については、「[15章qemu-img の使用](#)」を参照してください。

### 3.2.1. ISO イメージから仮想マシンをインストール

以下の例では、ISO イメージから仮想マシンがインストールされます。

```
# virt-install \
  --name guest1-rhel7 \
  --memory 2048 \
  --vcpus 2 \
  --disk size=8 \
  --cdrom /path/to/rhel7.iso \
  --os-variant rhel7
```

**--cdrom /path/to/rhel7.iso** オプションの設定により、指定した場所の CD または DVD イメージから仮想マシンがインストールされます。

### 3.2.2. 仮想マシンイメージのインポート

以下の例では、仮想ディスクイメージから仮想マシンがインポートされます。

```
# virt-install \
  --name guest1-rhel7 \
  --memory 2048 \
  --vcpus 2 \
  --disk /path/to/imported/disk.qcow \
  --import \
  --os-variant rhel7
```

**--import** オプションの設定により、**--disk /path/to/imported/disk.qcow** オプションで指定した仮想ディスクイメージから仮想マシンがインポートされます。

### 3.2.3. ネットワークから仮想マシンをインストール

以下の例では、ネットワークロケーションから仮想マシンがインストールされます。

```
# virt-install \  
  --name guest1-rhel7 \  
  --memory 2048 \  
  --vcpus 2 \  
  --disk size=8 \  
  --location http://example.com/path/to/os \  
  --os-variant rhel7
```

**--location http://example.com/path/to/os** オプションの設定により、特定のネットワークロケーションにあるインストールツリーが指定されます。

### 3.2.4. PXE を使用した仮想マシンのインストール

PXE ブートプロトコルを使って仮想マシンをインストールする場合は、ブリッジネットワークを指定する **--network** オプションおよび **--pxe** オプションの両方を指定する必要があります。

以下の例では、PXE を使って仮想マシンがインストールされます。

```
# virt-install \  
  --name guest1-rhel7 \  
  --memory 2048 \  
  --vcpus 2 \  
  --disk size=8 \  
  --network=bridge:br0 \  
  --pxe \  
  --os-variant rhel7
```

### 3.2.5. キックスタートを使った仮想マシンのインストール

以下の例では、キックスタートファイルを使って仮想マシンがインストールされます。

```
# virt-install \  
  --name guest1-rhel7 \  
  --memory 2048 \  
  --vcpus 2 \  
  --disk size=8 \  
  --location http://example.com/path/to/os \  
  --os-variant rhel7 \  
  --initrd-inject /path/to/ks.cfg \  
  --extra-args="ks=file:/ks.cfg console=tty0 console=ttyS0,115200n8"
```

**initrd-inject** および **extra-args** オプションの設定により、キックスタートファイルを使って仮想マシンがインストールされます。

### 3.2.6. ゲストの作成中にゲスト仮想マシンのネットワークを設定

ゲスト仮想マシンを作成する場合、仮想マシン用のネットワークを指定して設定することができます。このセクションでは、ゲスト仮想マシン用の主なネットワークタイプの各オプションについて説明します。

### NAT を用いるデフォルトのネットワーク

デフォルトのネットワークでは、`libvirtd` の Network Address Translation (NAT) 仮想ネットワークスイッチが使われます。NAT の詳細については、「[libvirt を使用した Network Address Translation \(NAT\)](#)」を参照してください。

NAT を用いるデフォルトのネットワークと共にゲスト仮想マシンを作成する前に、`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てください。

ゲスト仮想マシン用に NAT ネットワークを設定するには、`virt-install` の以下のオプションを使います。

```
--network default
```



#### 注記

`network` オプションを指定しないと、ゲスト仮想マシンは NAT を用いるデフォルトのネットワークと共に設定されます。

### DHCP を用いるブリッジネットワーク

ブリッジネットワーク用に設定された場合、ゲストは外部の DHCP サーバーを使用します。ホストが静的なネットワーク設定で、ゲストがローカルエリアネットワーク (LAN) との完全な受送信接続を必要とする場合は、このオプションを使う必要があります。ゲスト仮想マシンを使ってライブマイグレーションを実施する場合は、このオプションを使う必要があります。ゲスト仮想マシン用に DHCP を用いるブリッジネットワークを設定するには、以下のオプションを使います。

```
--network br0
```



#### 注記

`virt-install` を実行する前に、別途ブリッジを作成しておく必要があります。ネットワークブリッジ作成の詳細については、「[Red Hat Enterprise Linux 7 ホストでのブリッジネットワークの設定](#)」を参照してください。

### 静的な IP アドレスを用いるブリッジネットワーク

ブリッジネットワークを使って、ゲストが静的 IP アドレスを使うように設定することもできます。ゲスト仮想マシン用に静的な IP アドレスを用いるブリッジネットワークを設定するには、以下のオプションを使います。

```
--network br0 \
--extra-args
"ip=192.168.1.2::192.168.1.1:255.255.255.0:test.example.com:eth0:none"
```

ネットワークブートオプションの詳細については、『[Red Hat Enterprise Linux 7 インストールガイド](#)』を参照してください。

### ネットワーク設定なし

ゲスト仮想マシンにネットワークインターフェースを設定しない場合は、以下のオプションを使います。



```
--network=none
```

### 3.3. VIRT-MANAGER を使用したゲストの作成

**virt-manager** としても知られる仮想マシンマネージャーは、ゲスト仮想マシンを作成し、管理するグラフィカルツールです。

このセクションでは、**virt-manager** を使用して Red Hat Enterprise Linux 7 ホスト上で Red Hat Enterprise Linux 7 ゲスト仮想マシンをインストールする方法を説明します。

以下の手順では、KVM ハイパーバイザーや他の必要なパッケージすべてがインストールされており、ホストが仮想化用に設定されていることを前提としてします。仮想化パッケージのインストールについての詳細は、「[2章 仮想化パッケージのインストール](#)」を参照してください。

#### 3.3.1. virt-manager によるインストールの概要

新しい仮想マシンウィザードでは、仮想マシン作成が 5 つのステップで行われます。

1. ハイパーバイザーとインストール方法の選択
2. インストールメディアの場所の選択と設定
3. メモリーと CPU オプションの設定
4. 仮想マシンのストレージの設定
5. 仮想マシンの名前、ネットワーク、アーキテクチャー、他のハードウェアの設定

次に進む前に、**virt-manager** がインストールメディアにアクセスできることを確認してください (ローカルまたはネットワーク経由)。

#### 3.3.2. virt-manager を使用した Red Hat Enterprise Linux 7 ゲストの作成

以下の手順では、ローカルに保存されているインストール用 DVD や DVD イメージを使って Red Hat Enterprise Linux 7 のゲスト仮想マシンを作成する方法を説明します。Red Hat Enterprise Linux 7 の DVD イメージは、[Red Hat カスタマーポータル](#) から入手できます。

#### 手順3.1 virt-manager およびローカルインストールメディアを使用した Red Hat Enterprise Linux 7 ゲスト仮想マシンの作成

##### 1. オプション: 準備

仮想マシン用のストレージ環境を用意します。ストレージの準備に関する詳細は、「[13章 ストレージプール](#)」を参照してください。



#### 重要

ゲスト仮想マシンの保存には、様々な種類のストレージを使用することができます。しかし、仮想マシンで移行機能を使用できるようにするには、ネットワーク接続されたストレージ上に仮想マシンを作成する必要があります。

Red Hat Enterprise Linux 7 には、少なくとも 1GB のストレージ領域が必要です。しかし Red Hat は、Red Hat Enterprise Linux 7 のインストールと本ガイドの手順で 5GB 以上のストレージ領域を使用することを推奨しています。



## 2. virt-manager を開き、ウィザードを開始します。

virt-manager を開くには、root で **virt-manager** コマンドを実行するか、または **アプリケーション → システムツール → 仮想マシンマネージャー** を開きます。または、root で **virt-manager** コマンドを実行します。

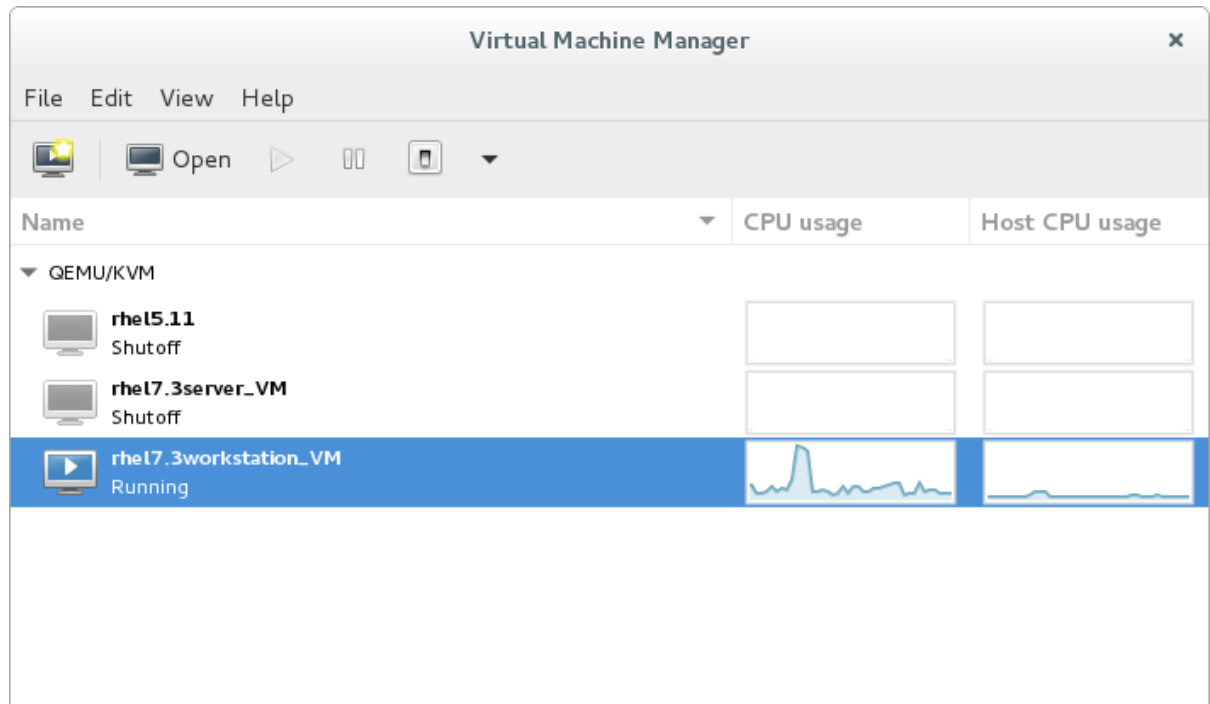


図3.1 仮想マシンマネージャーのウィンドウ

オプションとして、ハイパーバイザーを選択し、**接続 (Connect)** ボタンをクリックしてリモートハイパーバイザーを開きます。



をクリックして、新規の仮想化ゲストウィザードを開始します。

新しい仮想マシン ウィンドウが開きます。

## 3. インストール方法を指定します。

インストールの方法を選択します。

### ローカルのインストールメディア (ISO イメージまたは CD-ROM ドライブ)

この方法では、CD-ROM か DVD、またはインストールディスクのイメージを使用します (例: **.iso**)。

### ネットワークインストール (HTTP、FTP、または NFS)

この方法では、ミラー化された **Red Hat Enterprise Linux** または **Fedora** インストールツリーを使ってゲストをインストールします。インストールツリーには、HTTP または FTP、または NFS のいずれかでアクセスできる必要があります。

**ネットワークインストール** を選択する場合には、インストール URL およびカーネルオプション (必要な場合) を指定します。

### ネットワークブート (PXE)

この方法では、**Preboot eXecution Environment (PXE)** サーバーを使用してゲスト仮想マシンをインストールします。PXE サーバーの設定は、『[Red Hat Enterprise Linux 7 インストールガイド](#)』で説明されています。ネットワークブートでインストールするには、ゲスト

がルーティング可能な IP アドレスまたは共有ネットワークデバイスを備えている必要があります。

ネットワークブートを選択する場合は、ステップ 5 に進みます。すべてのステップが完了すると DHCP 要求は送信され、有効な PXE サーバーが見つかる場合はゲスト仮想マシンのインストールプロセスが開始します。

#### 既存のディスクイメージをインポート

この方法では、新しいゲスト仮想マシンを作成し、そこにディスクイメージ (インストール済みの起動可能なオペレーティングシステムを含む) をインポートできます。

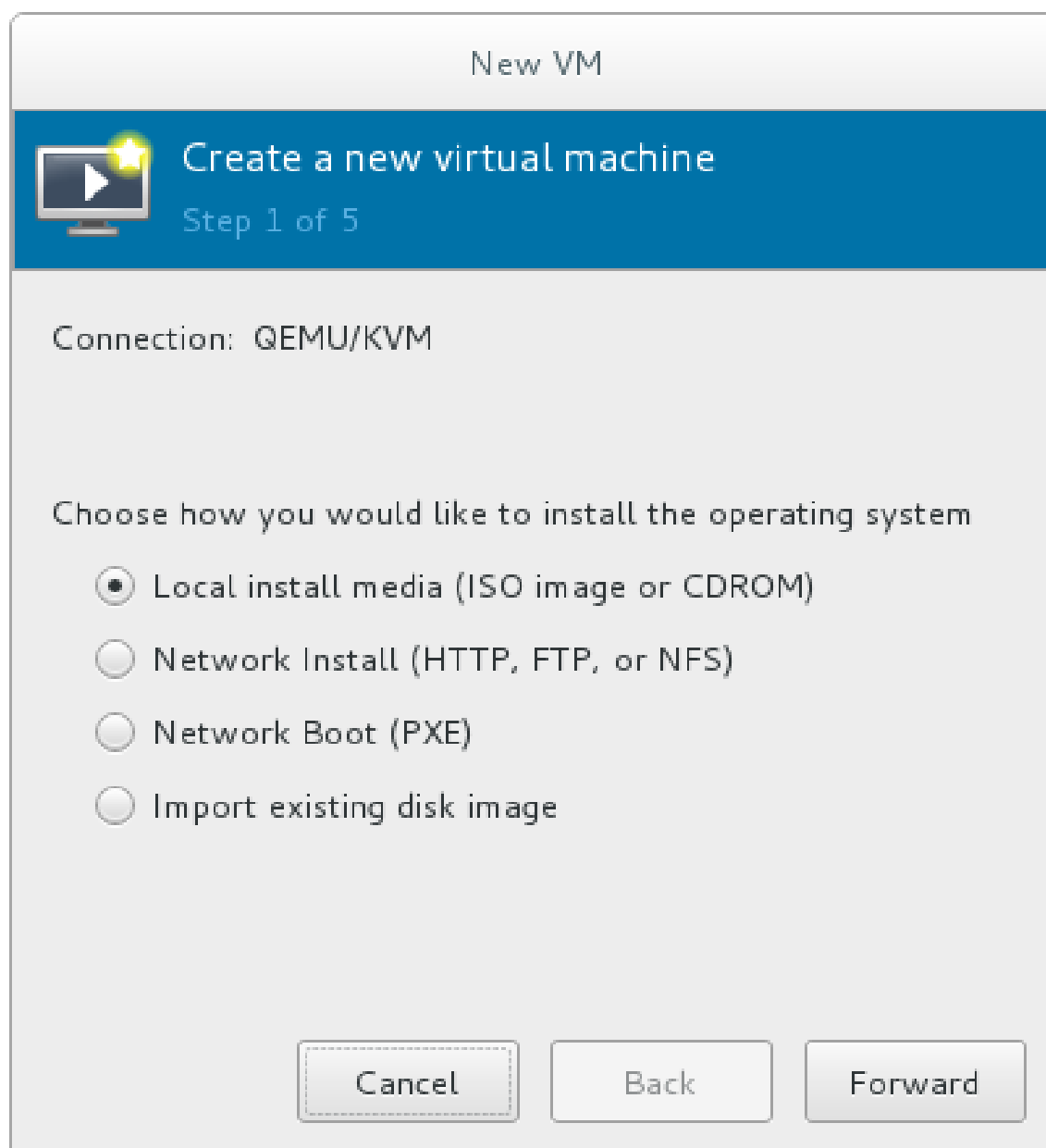


図3.2 仮想マシンのインストール方法

**進む** をクリックして次に進みます。

#### 4. インストールソースを選択します。

- a. ローカルのインストールメディア (ISO イメージまたは CD-ROM ドライブ) を選択した場合、必要なローカルインストールメディアを指定します。

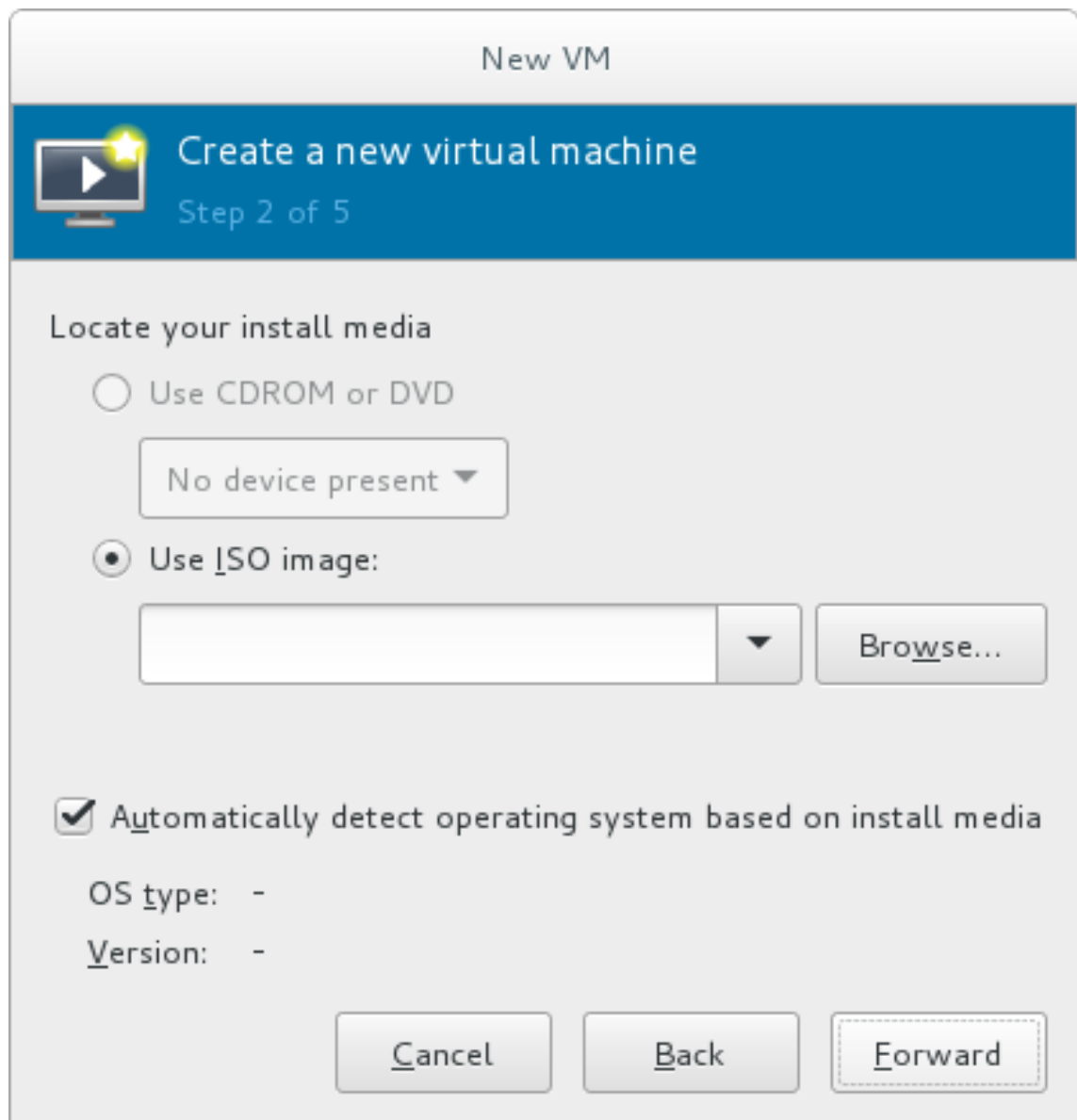


図3.3 ローカル ISO イメージのインストール

- CD-ROM または DVD からインストールする場合は、**CD-ROM または DVD を使用** ラジオボタンを選択し、利用可能なドライブのドロップダウンリストから適切なディスクドライブを選択します。
- ISO イメージからインストールする場合は、**ISO イメージを使用** を選択し、**参照...** ボタンをクリックして **ISO メディアボリュームの検索** ウィンドウを開きます。

使用するインストールイメージを選択し、**ボリュームの選択** をクリックします。

**ISO メディアボリュームの検索** ウィンドウにイメージが表示されない場合、**ローカルを参照** ボタンをクリックしてホストマシンにあるインストールイメージまたはインストールディスクのある DVD ドライブを参照します。インストールイメージまたはインストールディスクのある DVD ドライブを選択して **開く** をクリックします。使用するボリュームが選択され、**新しい仮想マシンを作成** ウィザードに戻ります。



## 重要

ISO イメージファイルまたはゲストストレージファイルの推奨される場所は、`/var/lib/libvirt/images/` です。それ以外の場所を指定する場合、SELinux による追加の設定が必要になる可能性があります。SELinux の設定に関する詳細は、『[Red Hat Enterprise Linux 7 仮想化セキュリティガイド](#)』または『[Red Hat Enterprise Linux 7 SELinux ユーザーおよび管理者のガイド](#)』を参照してください。

- b. ネットワークインストールを選択した場合には、インストールソースの URL および希望するカーネルオプション (必要な場合) を入力します。URL は、インストールツリーのルートディレクトリをポイントしている必要があります。また、インストールツリーには、HTTP、FTP、または NFS のいずれかでアクセスできる必要があります。

キックスタートインストールを実施するには、カーネルのオプションでキックスタートファイルの URL を指定します (`ks=` で始めます)。

**New VM**

Create a new virtual machine  
Step 2 of 5

Provide the operating system install URL

URL:

▼ URL Options

Kernel options:

☐ Automatically detect operating system based on install media

OS type:

Version:

図3.4 ネットワーク経由のキックスタートインストール



### 注記

カーネルオプションの全リストは、『[Red Hat Enterprise Linux 7 インストールガイド](#)』を参照してください。

次に、インストールする **OSの種類** と **バージョン** を選びます。仮想マシンに適したオペレーティングシステムの種類を選択することを確認してください。これは、**インストールメディア** に応じて、**仮想マシン内の OS の種類を自動判別する (Automatically detect operating system based on install media)** チェックボックスを選択して手動で指定することができます。

**進む** をクリックして次に進みます。

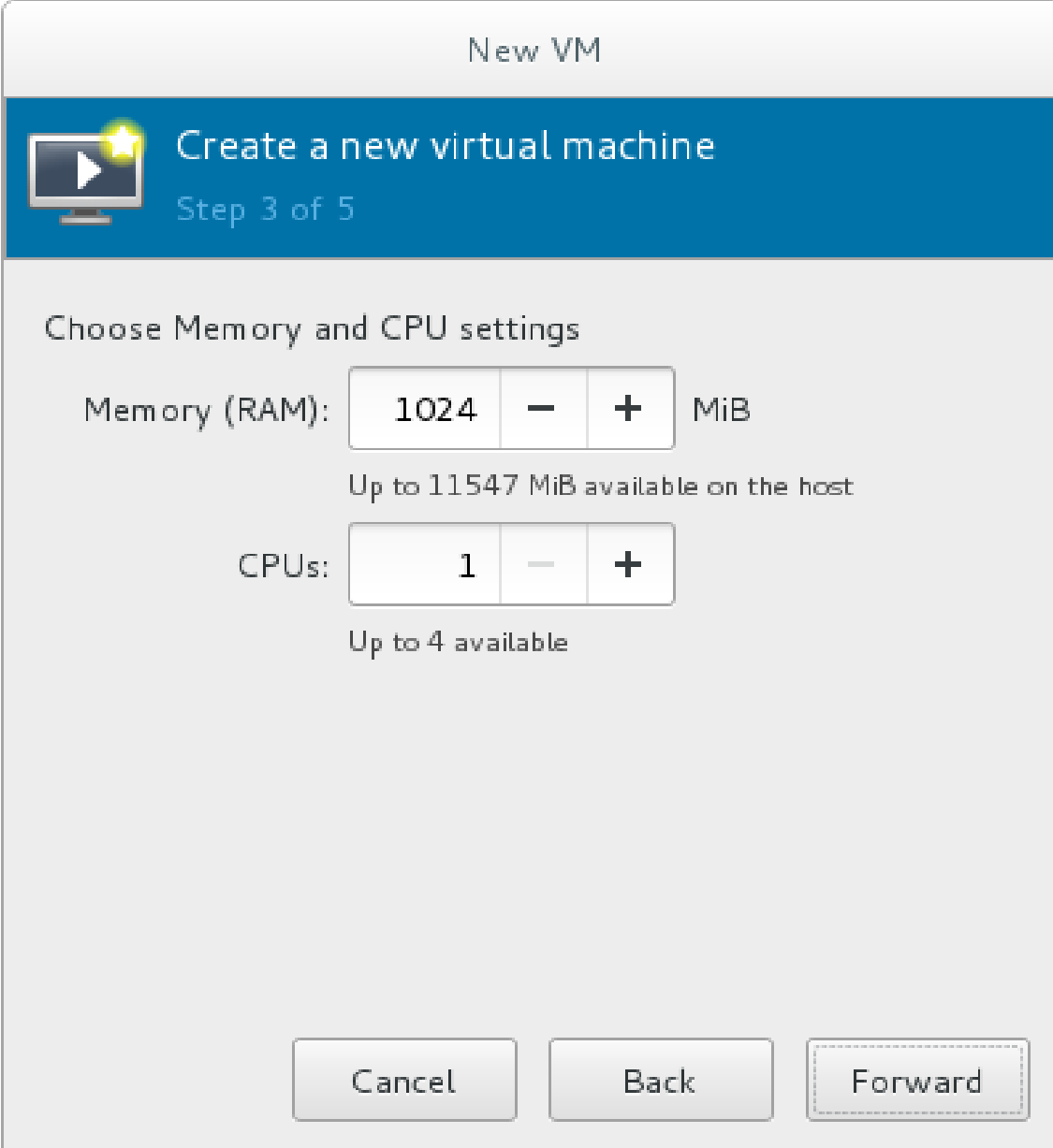
#### 5. メモリー (RAM) および仮想 CPU を設定します。

仮想マシンに割り当てる **CPUの数** と **メモリー (RAM)** の容量を設定します。ウィザードでは、割り当て可能な **CPUの数** と **メモリー量** が表示されます。これらの値はホストとゲストのパフォーマンスに影響を与えます。

仮想マシンの効率的かつ効果的な実行には、十分な物理メモリー (**RAM**) が必要です。Red Hat は、仮想マシンの最小 **512MB** の **RAM** をサポートします。1 論理コアあたりでは最小 **1024MB** の **RAM** を推奨します。

十分な数の仮想 **CPU** を仮想マシンに割り当てます。仮想マシンがマルチスレッドアプリケーションを実行する場合は、ゲスト仮想マシンの効果的な実行に必要な数の仮想 **CPU** を割り当てます。

ホストシステムで利用可能な物理プロセッサ (またはハイパースレッド) よりも多くの仮想 **CPU** を割り当てることはできません。利用可能な仮想 **CPU** 数は、**X** 個まで使用できますフィールドに表示されます。



New VM

Create a new virtual machine  
Step 3 of 5

Choose Memory and CPU settings

Memory (RAM): 1024 — + MiB  
Up to 11547 MiB available on the host

CPUs: 1 — +  
Up to 4 available

Cancel Back Forward

図3.5 メモリーと CPU の設定

メモリーと CPU の設定を行った後に、**進む** をクリックして次に進みます。



#### 注記

メモリーと仮想 CPU はオーバーコミットすることができます。オーバーコミットに関する詳細は、「[7章 KVM でのオーバーコミット](#)」を参照してください。

#### 6. ストレージを設定します。

仮想マシンと必要とされるアプリケーションに十分な領域を有効にし、その割り当てを実行します。デスクトップインストールの場合は少なくとも **5GB** を割り当て、最小インストールの場合は少なくとも **1GB** を割り当てます。

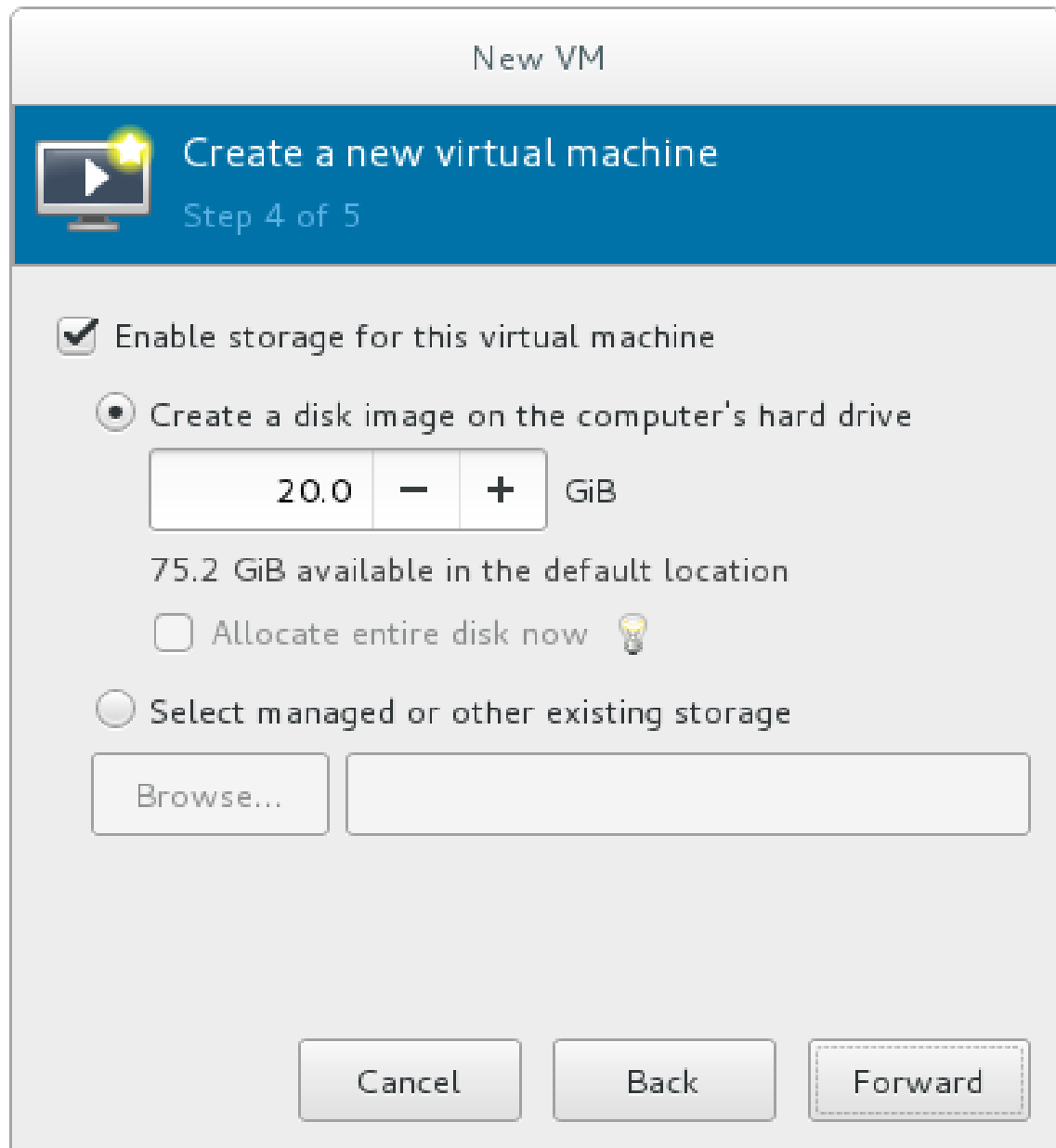


図3.6 仮想マシンのストレージの設定



### 注記

ライブおよびオフラインマイグレーションの場合は、仮想マシンを共有ネットワークストレージにインストールする必要があります。仮想マシン用の共有ストレージの設定に関する詳細は、「[共有ストレージの例: 単純な移行に NFS を使用する](#)」を参照してください。

#### a. デフォルトのローカルストレージを使用

コンピューターのハードディスク上にディスクイメージを作成する ラジオボタンを選択し、デフォルトのストレージプールの `/var/lib/libvirt/images/` ディレクトリーにファイルベースのイメージを作成します。作成するディスクイメージのサイズを入力します。今すぐディスク全体を割り当てる チェックボックスが選択されていると、指定されたサイズのディスクイメージが即時に作成されます。選択されていない場合は、ディスクイメージは要求に応じて大きくなります。

## 注記

ストレージプールは仮想コンテナであるものの、**qemu-kvm**によって許可される最大サイズとホストの物理マシン上のディスクサイズの2つの要素によって制限されます。ストレージプールはホスト物理マシンのディスクサイズを超えることはできません。最大サイズは以下のようになります。

- **virtio-blk** =  $2^{63}$  バイトまたは 8 エクサバイト (raw ファイルまたはディスクを使用)
- **Ext4** = ~16 TB (4 KB ブロックサイズを使用)
- **XFS** = ~8 エクサバイト
- **qcow2** とホストファイルシステムはそれぞれ独自のメタデータを維持します。非常に大きなイメージサイズを使用する場合はスケーラビリティの評価または調整を行う必要があります。**raw** ディスクを使用すると、スケーラビリティや最大サイズに影響を与える可能性のある層の数が少なくなります。

**進む** をクリックしてローカルのハードドライブにディスクイメージを作成します。または、**管理しているストレージか、他の既存のストレージを選択する** を選択してから **参照** を選択し、管理しているストレージを設定します。

### b. ストレージプールの使用

ストレージプールの使用に **管理しているストレージか、他の既存のストレージを選択する** を選択し、**参照** をクリックすると、ストレージボリュームの検索または作成 ウィンドウが開きます。

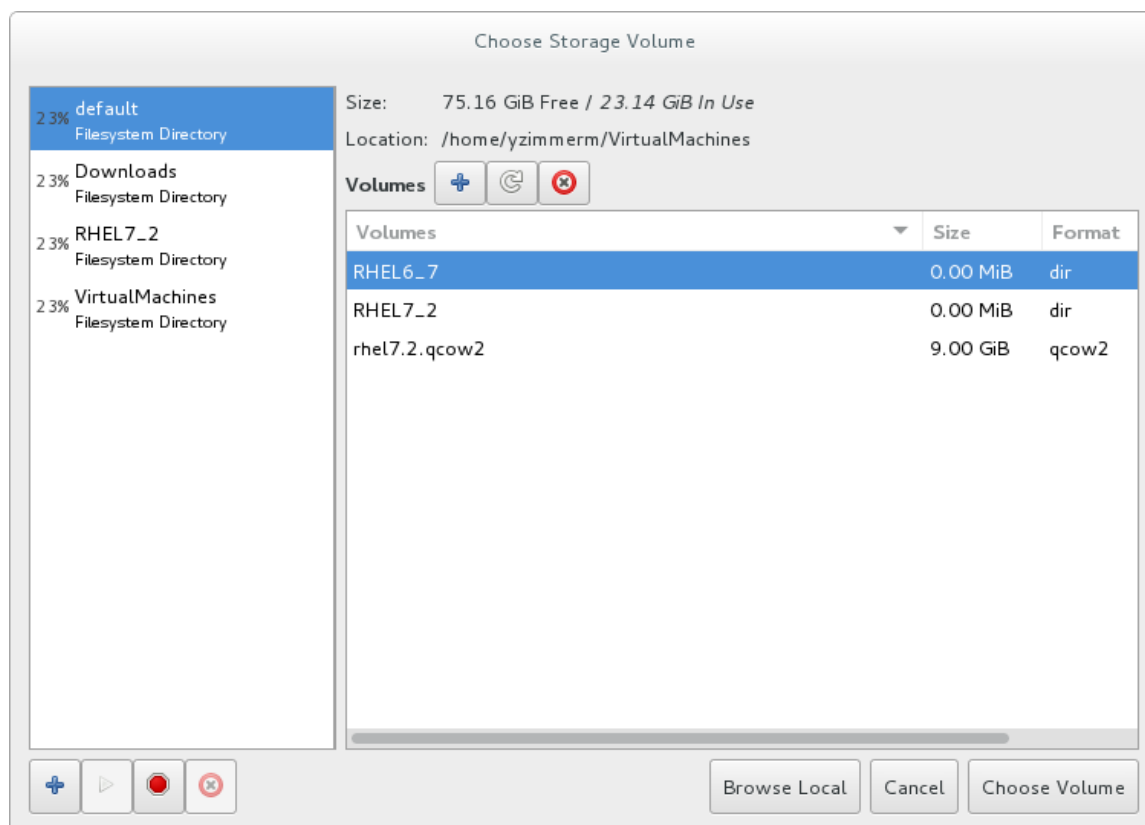


図3.7 「ストレージボリュームの選択」ウィンドウ

- i. ストレージプール一覧からストレージプールを選択します。



- ii. オプション:  をクリックして新しいストレージボリュームを作成します。ストレージボリュームを作成 (Add a Storage Volume) 画面が表示されます。新しいストレージボリュームの名前を入力します。

フォーマット ドロップダウンメニューからフォーマットのオプションを選択します。フォーマットオプションには、raw、qcow2 および qed があります。必要に応じて他のフィールドも調整します。ここで使用されている qcow バージョンはバージョン 3 であることに注意してください。qcow バージョンを変更するには、「[target 要素の設定](#)」を参照してください。

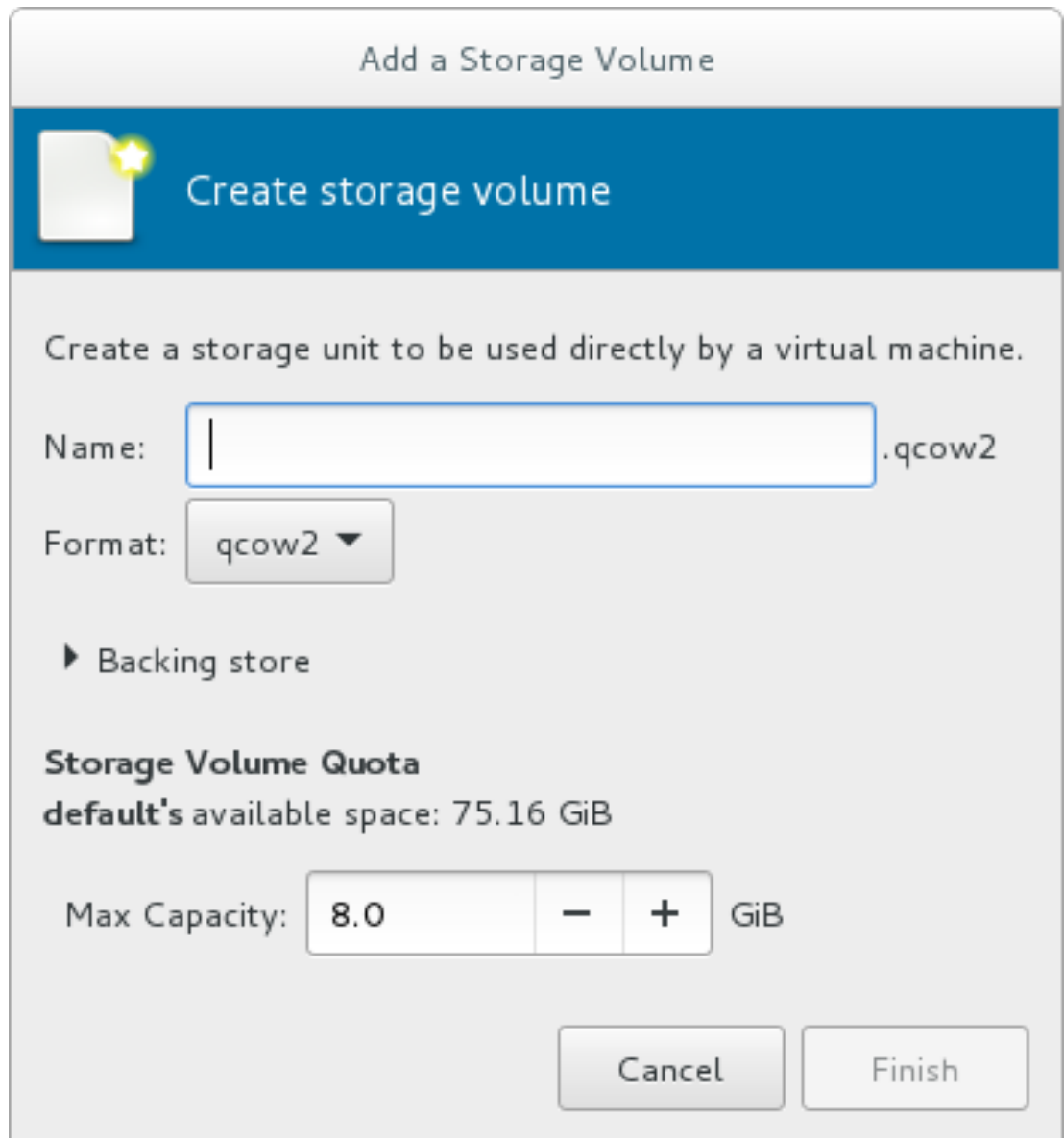


図3.8 「ストレージボリュームを作成 (Add a Storage Volume)」ウィンドウ

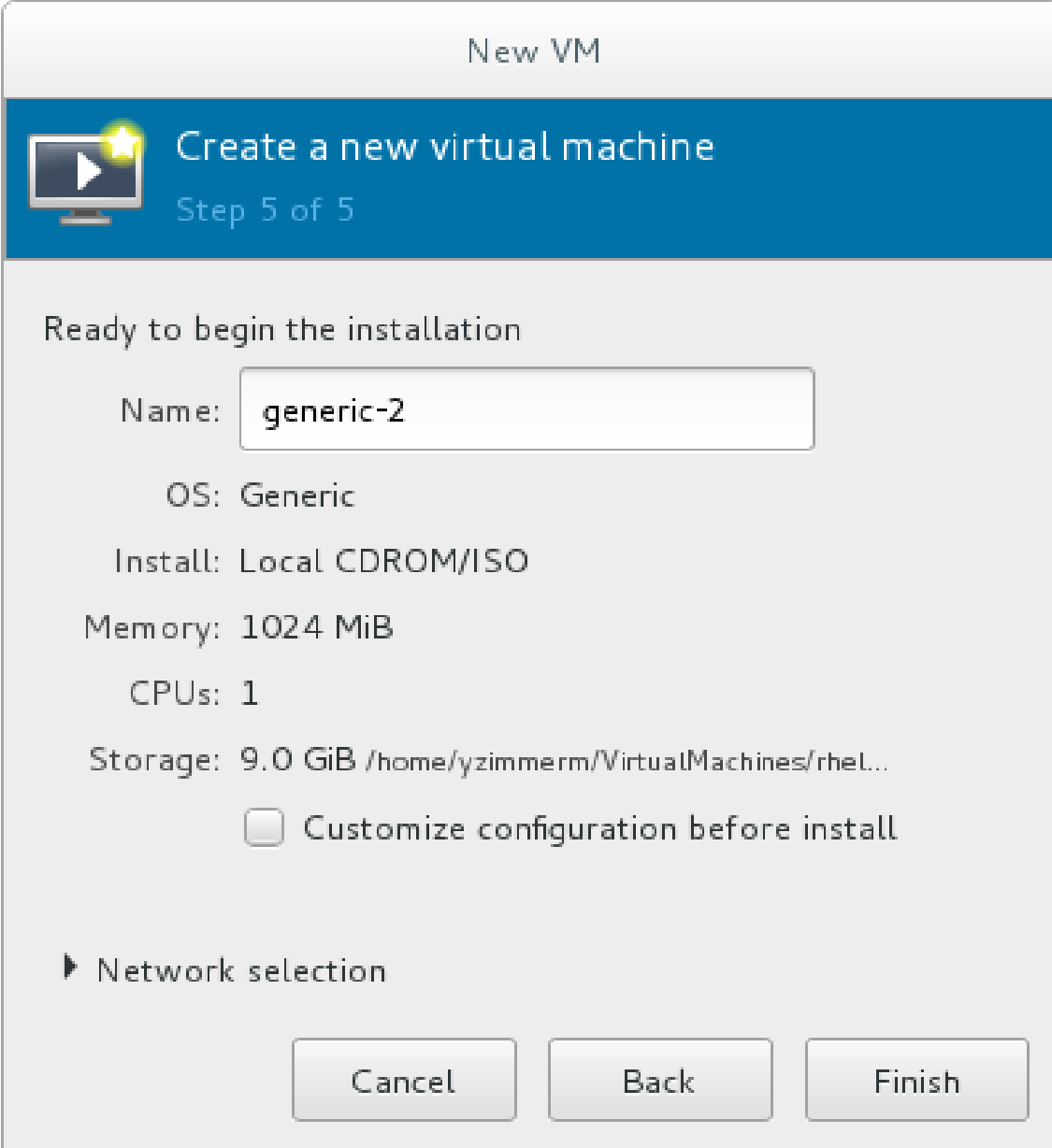
新規ボリュームを選択し、**ボリュームの選択** をクリックします。次に、**完了** をクリックして新しい仮想マシン ウィザードに戻ります。**進む** をクリックして次に進みます。

#### 7. 名前を付けて最終設定を行います。

仮想マシンに名前を付けます。仮想マシンの名前には、文字、数字、およびアンダースコア ( \_ )、ピリオド ( . )、ハイフン ( - ) などの文字を使用することができます。仮想マシンを移行するには、仮想マシンの名前は一意でなければならない、数字のみの名前は使用できません。

デフォルトで、仮想マシンは 'default' というネットワークについて **Network Address Translation (NAT)** を使用して作成されます。ネットワークの選択を変更するには、**ネットワークの選択** をクリックしてホストデバイスとソースモードを選択します。

仮想マシンの設定を確認し、問題がなければ**完了** をクリックします。指定されたネットワーク設定、仮想化の種類およびアーキテクチャーで仮想マシンが作成されます。



The screenshot shows a 'New VM' window titled 'Create a new virtual machine' with a progress indicator 'Step 5 of 5'. The window contains the following configuration details:

- Name: generic-2
- OS: Generic
- Install: Local CDROM/ISO
- Memory: 1024 MiB
- CPU: 1
- Storage: 9.0 GiB /home/yzimmerm/VirtualMachines/rheL...
- ☐ Customize configuration before install
- Network selection (indicated by a right-pointing triangle)

At the bottom, there are three buttons: 'Cancel', 'Back', and 'Finish'.

図3.9 設定の確認

または、仮想マシンのハードウェアをさらに設定する場合は、**インストールの前に設定をカスタマイズする** チェックボックスにチェックを入れます。これによりゲストのストレージまたはネットワークデバイスの変更や、準仮想化 (virtio) ドライバーの使用や、ドライバの追加を実行できます。ここから別のウィザードが開き、仮想マシンのハードウェア設定で追加や削除、設定が可能になります。



## 注記

Red Hat Enterprise Linux 4 または Red Hat Enterprise Linux 5 のゲスト仮想マシンは、グラフィカルモードを使用してインストールすることができません。そのため、ビデオカードには「QXL」ではなく「Cirrus」を選択する必要があります。

仮想マシンのハードウェアを設定した後に **適用** をクリックします。**virt-manager** が指定されたハードウェア設定で仮想マシンを作成します。



## 警告

Red Hat Enterprise Linux 7 のゲスト仮想マシンを、設定済みの TCP/IP 接続なしにリモートメディアからインストールすると、インストールは失敗します。ただし、この状況で Red Hat Enterprise Linux 5 または 6 のゲスト仮想マシンをインストールすると、インストーラーは「Configure TCP/IP (TCP/IP の設定)」インターフェースを開きます。

この違いについての詳細は、[関連するナレッジベース記事](#) を参照してください。

**完了** ボタンをクリックして Red Hat Enterprise Linux インストールを継続します。Red Hat Enterprise Linux 7 のインストール方法の詳細は、『[Red Hat Enterprise Linux 7 インストールガイド](#)』を参照してください。

これで ISO インストールディスクイメージから Red Hat Enterprise Linux 7 ゲスト仮想マシンが作成されました。

## 3.4. VIRT-INSTALL および VIRT-MANAGER のインストールオプションの比較

以下の表は、仮想マシンのインストール時のオプションとして、同等の **virt-install** と **virt-manager** のインストールオプションを比較しています。

ほとんどの **virt-install** オプションは必須ではありません。最小要件として、**--name**、**--memory**、ゲストストレージ (**--disk**、**--filesystem**、または **--disk none**)、およびインストール方法 (**--location**、**--cdrom**、**--pxe**、**--import**、または **boot**) が必要になります。これらのオプションは引数によってさらに詳細化されます。コマンドオプションと関連する引数の詳細の一覧を表示するには、以下のコマンドを入力します。

```
# virt-install --help
```

**virt-manager** では、最小要件として、名前、インストール方法、メモリー (RAM)、仮想 CPU (vCPU)、ストレージが必要になります。

表3.1 ゲストのインストールに関する **virt-install** と **virt-manager** の設定の比較

仮想マシンの設定	virt-install オプション	virt-manager インストールウィザードのラベルとステップ番号
仮想マシン名	--name, -n	名前 (ステップ 5)
割り当てる RAM (MiB)	--ram, -r	メモリー (RAM) (ステップ 3)
ストレージ - ストレージメディアの指定	--disk	この仮想マシンにストレージデバイスを割り当てます。→ コンピューターのハードディスク上にディスクイメージを作成する、または管理しているか、他の既存のストレージを選択する (ステップ 4)
ストレージ - ホストのディレクトリをゲストにエクスポート	--filesystem	この仮想マシンにストレージデバイスを割り当てます。→ 管理しているか、他の既存のストレージを選択する (ステップ 4)
ストレージ - ゲストにローカルのディスクストレージを設定しない	--nodisks	「この仮想マシンにストレージデバイスを割り当てます。」チェックボックスの選択解除 (ステップ 4)
インストールメディアの場所 (ローカルインストール)	--file	ローカルのインストールメディア → インストールメディアの場所 (ステップ 1-2)
ディストリビューションツリーからのインストール (ネットワークインストール)	--location	ネットワークインストール → URL (ステップ 1-2)
PXE を使用したゲストのインストール	--pxe	ネットワークブート (ステップ 1)
仮想 CPU 数	--vcpus	CPU (ステップ 3)
ホストのネットワーク	--network	「詳細なオプション (Advanced options)」ドロップダウンメニュー (ステップ 5)
オペレーティングシステムのバリエーション/バージョン	--os-variant	バージョン (ステップ 2)
グラフィカル表示の方法	--graphics, --nographics	* virt-manager は GUI インストールのみを提供します

## 第4章 仮想マシンのクローン作成

ゲストのコピーを作成する場合、2種類のゲスト仮想マシンのインスタンスが使われます。

- **クローン**は単一の仮想マシンのインスタンスです。クローンは、同一の仮想マシンのネットワークを設定するのに使うことができ、別の場所に分散させることもできます。
- **テンプレート**は、クローン作成元として使われるように作られた仮想マシンのインスタンスです。テンプレートから複数のクローンを作成し、それぞれのクローンに若干の変更を加えることができます。このことは、これらの変更がシステムに及ぼす影響を把握する場合に役立ちます。

クローンおよびテンプレートは、共に仮想マシンのインスタンスです。両者の違いは、その使われ方です。

作成したクローンが正常に機能するためには、クローンを作成する前にクローン作成元の仮想マシンに固有な情報および設定を削除する必要があります。削除すべき情報は、クローンの使われ方によって異なります。

削除すべき情報および設定は、以下に示すレベルのいずれかに属します。

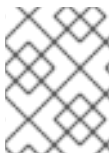
- **プラットフォームレベル**の情報および設定は、仮想化ソリューションによって仮想マシンに割り当てられたものです。ネットワークインターフェースカード (NIC) の番号およびその MAC アドレス等が含まれます。
- **ゲストオペレーティングシステムレベル**の情報および設定は、仮想マシン内で設定されたものです。SSH キー等が含まれます。
- **アプリケーションレベル**の情報および設定は、仮想マシンにインストールされたアプリケーションによって設定されたものです。アクティベーションコードおよび登録情報などが含まれます。



### 注記

本章では、アプリケーションレベルの情報および設定を削除する手順については説明しません。情報および考え方がアプリケーションごとに異なるためです。

したがって、一部の情報および設定は仮想マシン内で削除する必要があり、一方、仮想化環境 (仮想マシンマネージャーまたは VMware 等) を使って仮想マシンから削除しなければならない情報および設定もあります。



### 注記

ストレージボリュームのクローン作成についての情報は、「[ボリュームのクローン作成](#)」を参照してください。

### 4.1. 仮想マシンのクローン作成準備

仮想マシンのクローンを作成する前に、そのディスクイメージの **virt-sysprep** ユーティリティを実行するか、以下の手順を使って準備を行う必要があります。

#### 手順4.1 仮想マシンのクローン作成準備

##### 1. 仮想マシンの設定

- a. クローンまたはテンプレートに使われる仮想マシンを構築します。

- クローンで必要となるソフトウェアをすべてインストールします。
- オペレーティングシステムに固有ではない設定をすべて定義します。
- アプリケーションに固有ではない設定をすべて定義します。

## 2. ネットワーク設定の削除

- a. 以下のコマンドを使って、永続的な **udev** ルールをすべて削除します。

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



### 注記

**udev** ルールが削除されない場合は、1 番目の NIC の名前が **eth0** ではなく **eth1** の可能性があります。

- b. **/etc/sysconfig/network-scripts/ifcfg-eth[x]** を以下のように編集して、**ifcfg** スクリプトから固有のネットワーク情報を削除します。

- i. **HWADDR** および **Static** 行を削除します



### 注記

**HWADDR** と新しいゲストの **MAC** アドレスが一致していないと、**ifcfg** は無視されます。したがって、ファイルから **HWADDR** を削除することが重要です。

```
DEVICE=eth[x]
BOOTPROTO=none
ONBOOT=yes
#NETWORK=10.0.1.0          <- REMOVE
#NETMASK=255.255.255.0    <- REMOVE
#IPADDR=10.0.1.20         <- REMOVE
#HWADDR=xx:xx:xx:xx:xx    <- REMOVE
#USERCTL=no               <- REMOVE
# Remove any other *unique* or non-desired settings, such as
UUID.
```

- ii. **HWADDR** または固有の情報が削除され、**DHCP** 設定が維持されていることを確認します。

```
DEVICE=eth[x]
BOOTPROTO=dhcp
ONBOOT=yes
```

- iii. ファイルに以下の行が含まれていることを確認します。

```
DEVICE=eth[x]
ONBOOT=yes
```

c. 以下のファイルが存在する場合は、内容が同じであることを確認します。

- `/etc/sysconfig/networking/devices/ifcfg-eth[x]`
- `/etc/sysconfig/networking/profiles/default/ifcfg-eth[x]`



#### 注記

仮想マシンで **NetworkManager** または何らかの特殊な設定が使われた場合には、その他すべての固有の情報が `ifcfg` スクリプトから削除されていることを確認します。

### 3. 登録情報の削除

a. 以下のいずれかの方法で、登録情報を削除します。

- **Red Hat Network (RHN)** に登録されたゲスト仮想マシンの場合は、以下のコマンドを実行します。

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

- **Red Hat サブスクリプションマネージャー (RHSM)** に登録されたゲスト仮想マシンの場合:

- クローン元の仮想マシンが使われない場合は、以下のコマンドを実行します。

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

- クローン元の仮想マシンが使われる場合は、以下のコマンドだけを実行します。

```
# subscription-manager clean
```



#### 注記

元の RHSM プロファイルはポータルに残されます。

### 4. その他の固有情報の削除

a. 以下のコマンドを使って、`sshd` パブリック/プライベートキーのペアを削除します。

```
# rm -rf /etc/ssh/ssh_host_*
```



#### 注記

SSH キーを削除することで、SSH クライアントがこれらのホストを信頼しないという問題を防ぐことができます。

b. アプリケーション固有の識別子、または複数のマシンで実行した場合に競合する恐れのある設定をすべて削除します。

## 5. 次回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

- a. 以下のいずれかの方法で、次回起動時に適切な設定ウィザードが実行されるように、仮想マシンを設定します。

- Red Hat Enterprise Linux 6 およびそれ以前のバージョンでは、以下のコマンドを使って、ルートファイルシステムに **.unconfigured** という名前の空ファイルを作成します。

```
# touch /.unconfigured
```

- Red Hat Enterprise Linux 7 では、次のコマンドを実行して **first boot** および **initial-setup** ウィザードを有効にします。

```
# sed -ie 's/RUN_FIRSTBOOT=NO/RUN_FIRSTBOOT=YES/'
/etc/sysconfig/firstboot
# systemctl enable firstboot-graphical
# systemctl enable initial-setup-graphical
```



### 注記

次回起動時に実行されるウィザードは、仮想マシンから削除された設定により異なります。また、クローンの初回起動時に、ホスト名を変更することを推奨します。

## 4.2. 仮想マシンのクローン作成

クローン作成手順を進める前に、仮想マシンをシャットダウンします。**virt-clone** または **virt-manager** を使って仮想マシンのクローンを作成することができます。

### 4.2.1. virt-clone を使用したゲストのクローン作成

**virt-clone** を使って、コマンドラインから仮想マシンのクローンを作成できます。

**virt-clone** を正しく完了するには、**root** 権限が必要となることに注意してください。

**virt-clone** コマンドは、コマンドラインで渡すことのできる数多くのオプションを提供します。これらには、一般的なオプション、ストレージ設定のオプション、ネットワーク設定のオプション、およびその他のオプションが含まれます。必須となるのは **--original** のみです。オプションの詳細の一覧を表示するには、以下のコマンドを入力します。

```
# virt-clone --help
```

**virt-clone** の **man** ページも、各コマンドのオプション、重要な変数および例について記載しています。

以下の例は、デフォルト接続で「**demo**」というゲスト仮想マシンをクローン作成する方法を示しています。新規の名前とディスクのクローンパスが自動生成されます。

#### 例4.1 virt-clone を使用したゲストのクローン作成

```
# virt-clone --original demo --auto-clone
```



以下の例は、複数のディスクと共に「demo」という QEMU ゲスト仮想マシンをクローン作成する方法を示しています。

#### 例4.2 virt-clone を使用したゲストのクローン作成

```
# virt-clone --connect qemu:///system --original demo --name newdemo --  
file /var/lib/xen/images/newdemo.img --file  
/var/lib/xen/images/newdata.img
```

#### 4.2.2. virt-manager を使用したゲストのクローン作成

以下の手順では、**virt-manager** ユーティリティを使用してゲスト仮想マシンのクローンを作成する方法を説明します。

##### 手順4.2 virt-manager を使用した仮想マシンのクローン作成

###### 1. virt-manager を開きます。

**virt-manager** を開始します。アプリケーションメニューを開き、システムツールサブメニューから **仮想マシンマネージャー** アプリケーションを起動します。または、**root** で **virt-manager** コマンドを実行します。

**仮想マシンマネージャー** のゲスト仮想マシンの一覧からクローン作成するゲスト仮想マシンを選択します。

クローン作成するゲスト仮想マシンを右クリックして、**クローン** を選択します。「仮想マシンのクローンを作成」ウィンドウが開きます。

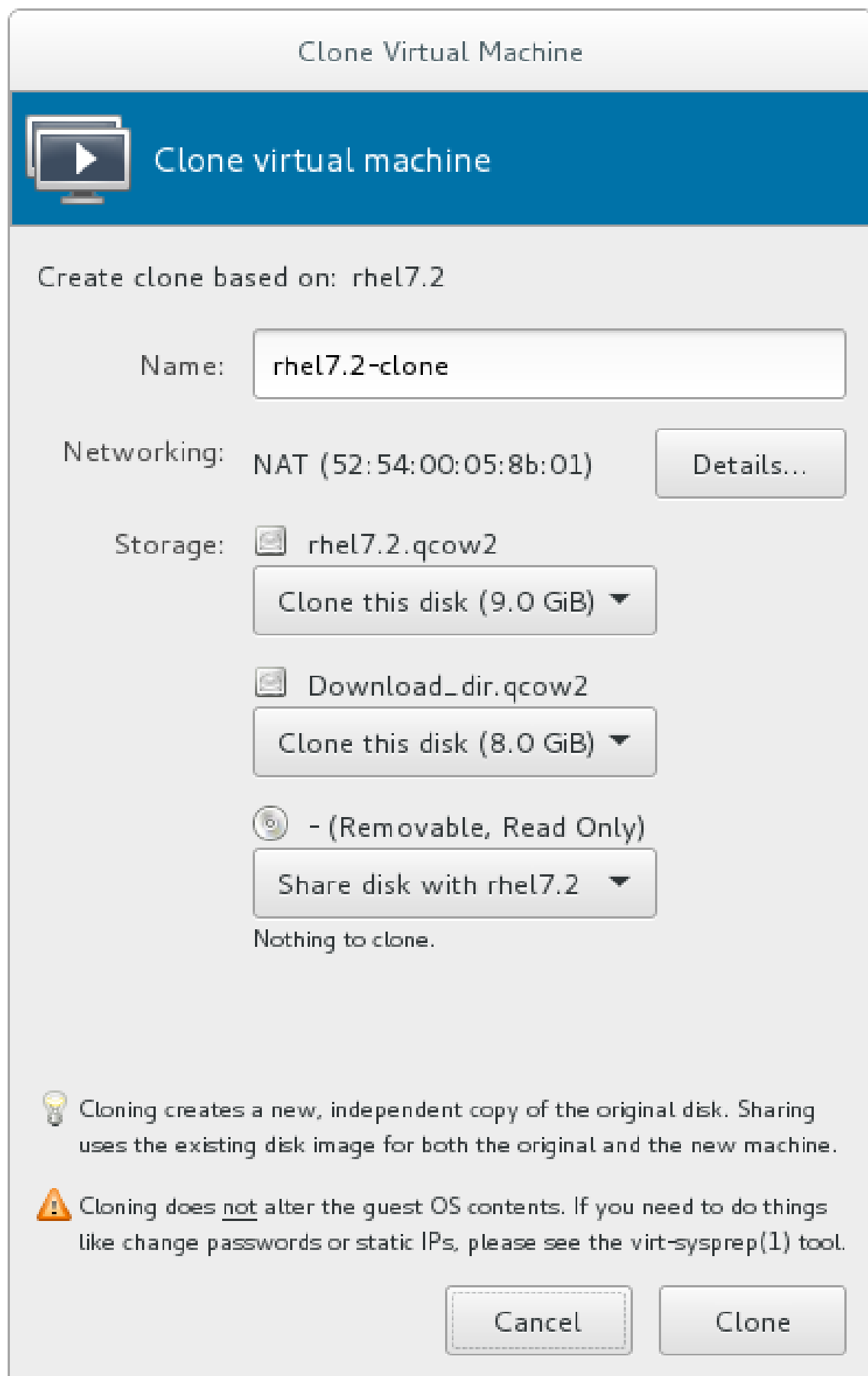


図4.1 「仮想マシンのクローンを作成」ウィンドウ

2. クローンを設定します。

- クローンの名前を変更するには、クローンの新規の名前を入力します。

- ネットワーク設定を変更するには、**詳細** をクリックします。

クローンの新しい MAC アドレスを入力します。

**OK** をクリックします。

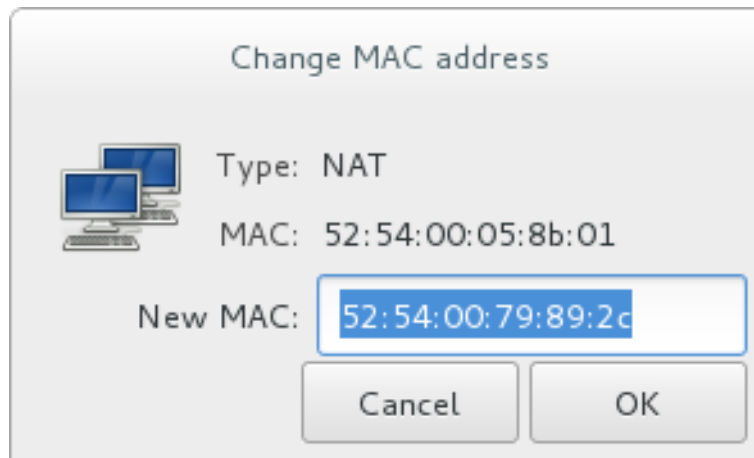


図4.2 「MAC アドレスを変更」ウィンドウ

- クローン作成されたゲスト仮想マシンのそれぞれのディスクについて、以下のオプションのいずれかを選択します。
  - **ディスクをクローン**: ディスクのクローンはクローン作成されたゲスト仮想マシンから作成されます。
  - **ディスクをゲスト仮想マシンの名前と共有**: ディスクは、クローン作成されるゲスト仮想マシンとそのクローンによって共有されます。
  - **詳細**: 「ストレージパスを変更」ウィンドウが開きます。ここからディスクの新規パスを選択できます。

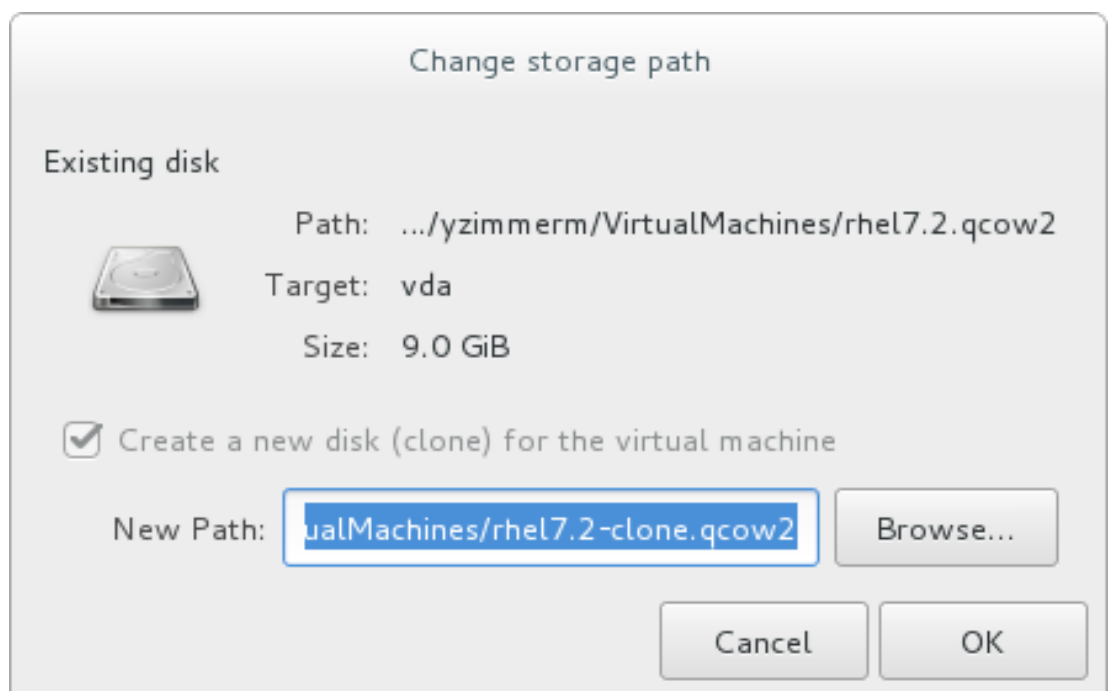


図4.3 「ストレージパスを変更」ウィンドウ

3. ゲスト仮想マシンのクローンを作成します。  
クローン をクリックします。

## 第5章 KVM 準仮想化 (VIRTIO) ドライバー

準仮想化ドライバーはゲストのパフォーマンスを強化し、I/O 待ち時間を短縮すると共に、スループットをベアメタルレベル近くにまで高めます。I/O 負荷の高いタスクとアプリケーションを実行する完全仮想マシンには、準仮想化ドライバーの使用が推奨されます。

virtio ドライバーは、KVM ホスト上で実行されるゲスト仮想マシンに利用できる 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** パッケージに含まれています。**virtio** パッケージは、ブロック (ストレージ) デバイスとネットワークインターフェースコントローラーに対応しています。

KVM virtio ドライバーは、以下に自動的にロードされ、インストールされます。

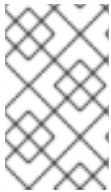
- Red Hat Enterprise Linux 5.3 およびそれ以降
- Red Hat Enterprise Linux 6 およびそれ以降
- Red Hat Enterprise Linux 7 およびそれ以降
- 2.6.27 カーネルまたはそれ以降のカーネルバージョンをベースとした、いくつかの Linux バージョン

上記の Red Hat Enterprise Linux バージョンは、ドライバーを検出し、インストールするので、追加のインストールステップは必要ありません。



### 注記

PCI デバイスは、仮想化システムアーキテクチャーによる制限を受けます。割り当てデバイスの使用時の追加の制限については、「[17章ゲスト仮想マシンデバイスの設定](#)」を参照してください。



### 注記

古いバージョンの virtio ドライバーを新しいバージョンと QEMU と共に使用しようとすると、ネットワーク接続の問題が生じることがあります。そのため、ドライバーを最新の状態に保つことをお勧めします。

## 5.1. 既存デバイスでの KVM VIRTIO ドライバーの使用

ゲストに割り当てられている既存ハードディスクデバイスを変更して、仮想化 IDE ドライバーの代わりに **virtio** ドライバーを使用することができます。このセクションの例では、**libvirt** 設定ファイルを編集します。これらのステップを実行するためにゲスト仮想マシンをシャットダウンする必要はありませんが、変更が適用されるには、ゲストを完全にシャットダウンして再起動する必要があります。

### 手順5.1 既存デバイスでの KVM virtio ドライバーの使用

1. この手順を実行する前に、適切なドライバー (**viostor**) がインストールされていることを確認してください。
2. **root** で **virsh edit guestname** コマンドを実行し、デバイスの XML 設定ファイルを編集します。たとえば、**virsh edit guest1** のようになります。設定ファイルは、**/etc/libvirt/qemu/** ディレクトリにあります。

3. 以下の例は、仮想化 IDE ドライバーを使用したファイルベースのブロックデバイスです。これは、**virtio** ドライバーを使用しない仮想マシンの通常のエンタリーです。

```
<disk type='file' device='disk'>
  ...
  <source file='/var/lib/libvirt/images/disk1.img'/>
  <target dev='hda' bus='ide'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x07'
function='0x0'/>
</disk>
```

4. **virtio** デバイスを使用するために、**bus=** エンタリーを **virtio** に変更します。ディスクが以前に IDE だった場合は、**hda** や **hdb**、**hdc** などと同様のターゲットを持ちます。**bus=virtio** に変更する場合は、ターゲットもそれに応じて **vda** や **vdb**、**vdc** に変更する必要があります。

```
<disk type='file' device='disk'>
  ...
  <source file='/var/lib/libvirt/images/disk1.img'/>
  <target dev='vda' bus='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x07'
function='0x0'/>
</disk>
```

5. **disk** タグ内の **address** タグを削除します。これは、この手順が動作するために必要な作業です。仮想マシンの次回起動時に、**libvirt** が適切に **address** タグを再生成します。

別の方法としては、**virtio** ドライバーを使用して **virt-manager**、**virsh attach-disk**、または **virsh attach-interface** で新規デバイスを追加することもできます。

**Virtio** の使用方法の詳細については、**libvirt** の web サイト <http://www.linux-kvm.org/page/Virtio> を参照してください。

## 5.2. KVM VIRTIO ドライバーを使用した新規デバイスの作成

ここでは、KVM **virtio** ドライバーを使用した **virt-manager** での新規デバイスの作成について説明します。

別の方法としては、**virtio** ドライバーを使用し、**virsh attach-disk** または **virsh attach-interface** コマンドでデバイスを割り当てることもできます。



### 重要

新規デバイスのインストールに進む前に、ゲストにドライバーがインストールされていることを確認してください。ドライバーが利用可能でない場合、デバイスは認識されず、動作しません。

### 手順5.2 virtio ストレージドライバーを使用したストレージデバイスの追加

1. **virt-manager** でゲスト名をダブルクリックしてゲスト仮想マシンを開きます。



2. をクリックして、**仮想マシンの情報を表示** タブを開きます。

3. 仮想マシンの情報を表示 タブで、ハードウェアを追加 ボタンをクリックします。
4. ハードウェアの種類を選択します。  
ハードウェアの種類 で ストレージ を選びます。

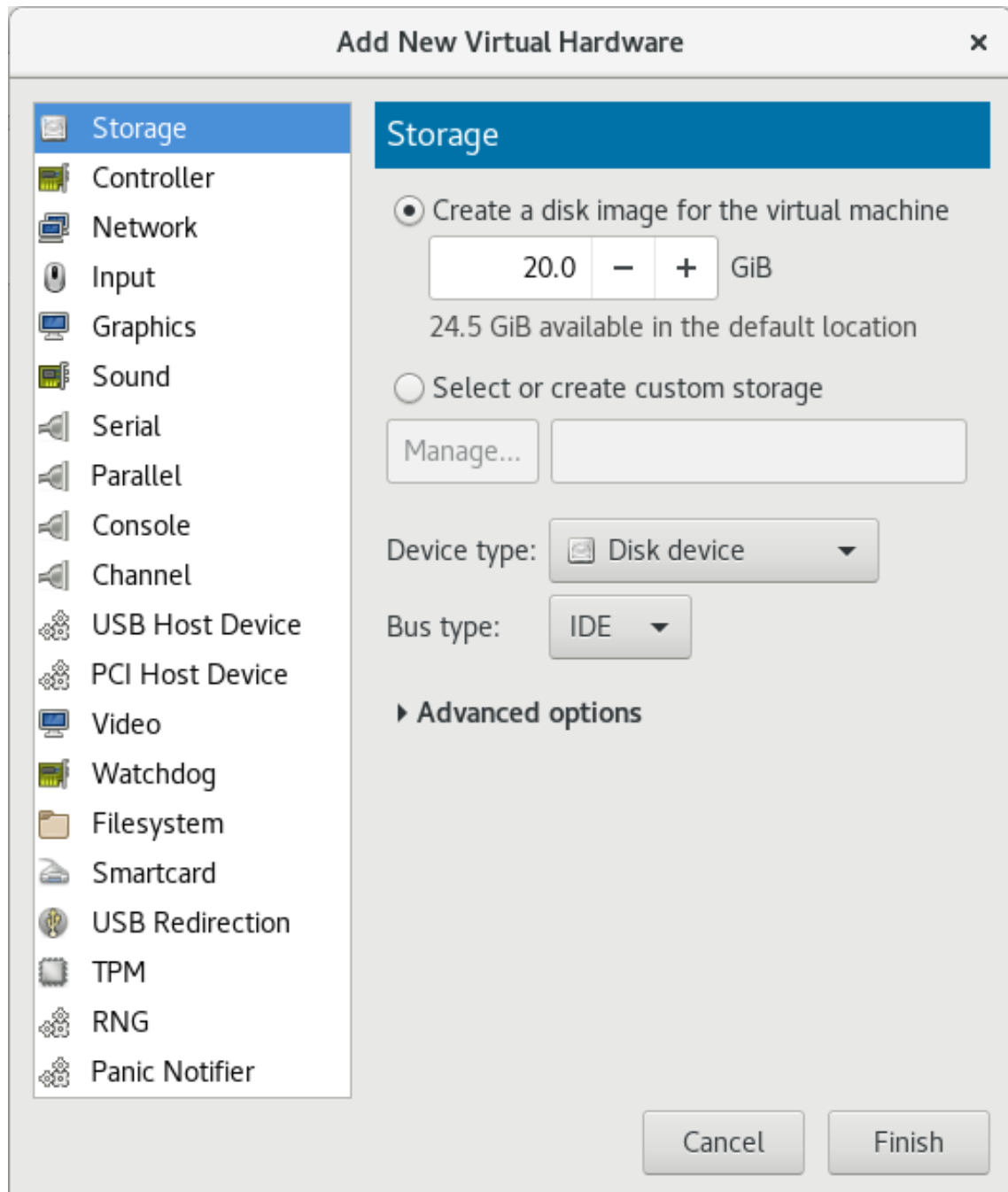


図5.1 新しいハードウェアを追加ウィザード

5. ストレージデバイスとドライバーの選択  
新規ディスクイメージを作成するか、ストレージプールボリュームを選択します。

デバイスの種類 を ディスクデバイス に、バスの種類 を **VirtIO** に設定して virtio ドライバーを使用します。

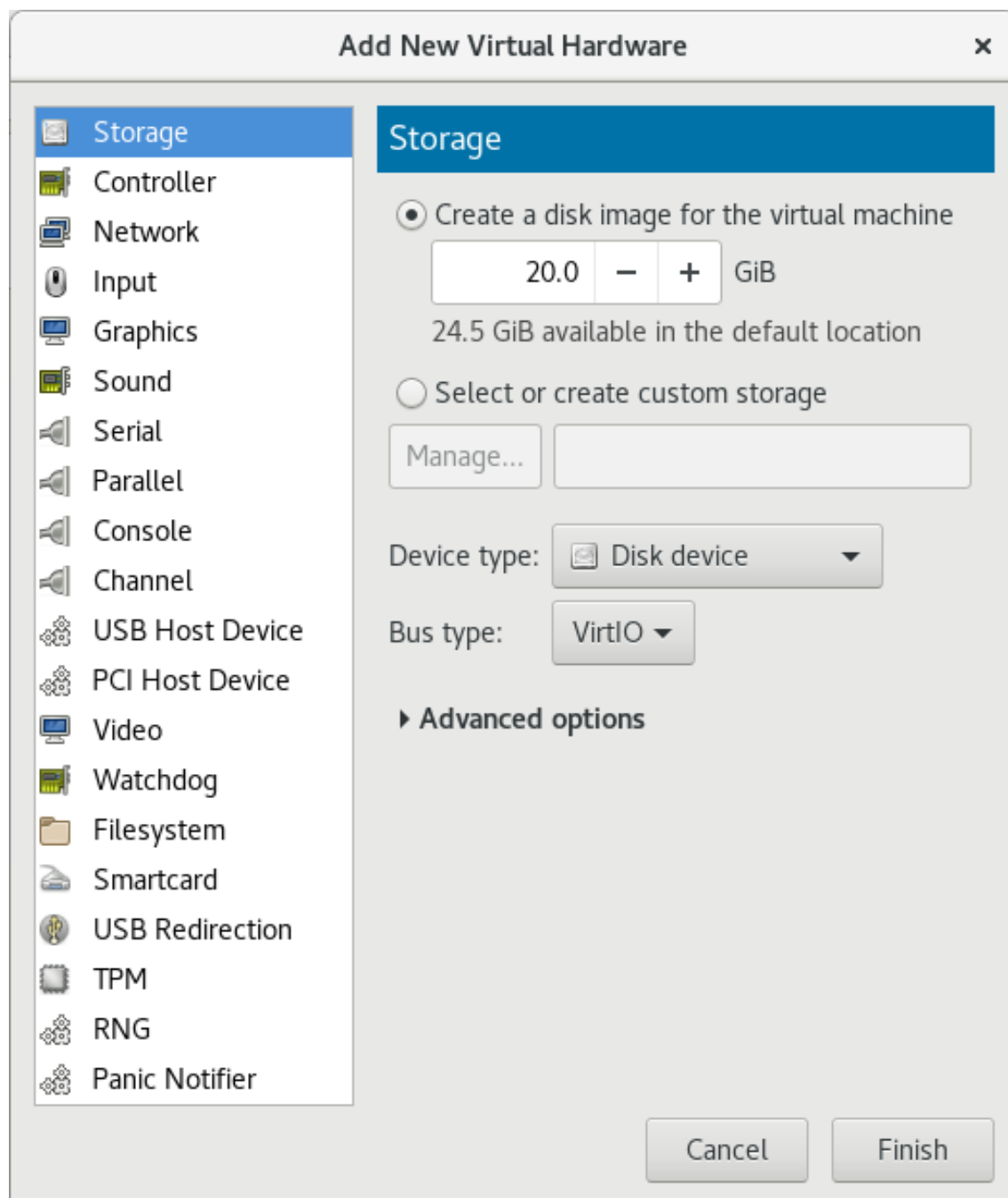



図5.2 新しい仮想ハードウェアを追加ウィザード

完了 をクリックして終了します。

### 手順5.3 virtio ネットワークドライバーを使用したネットワークデバイスの追加

1. **virt-manager** でゲスト名をダブルクリックしてゲスト仮想マシンを開きます。
2.  をクリックして、**仮想マシンの情報を表示** タブを開きます。
3. **仮想マシンの情報を表示** タブで、**ハードウェアを追加** ボタンをクリックします。
4. **ハードウェアの種類** を選択します。  
ハードウェアの種類 で **ネットワーク** を選びます。



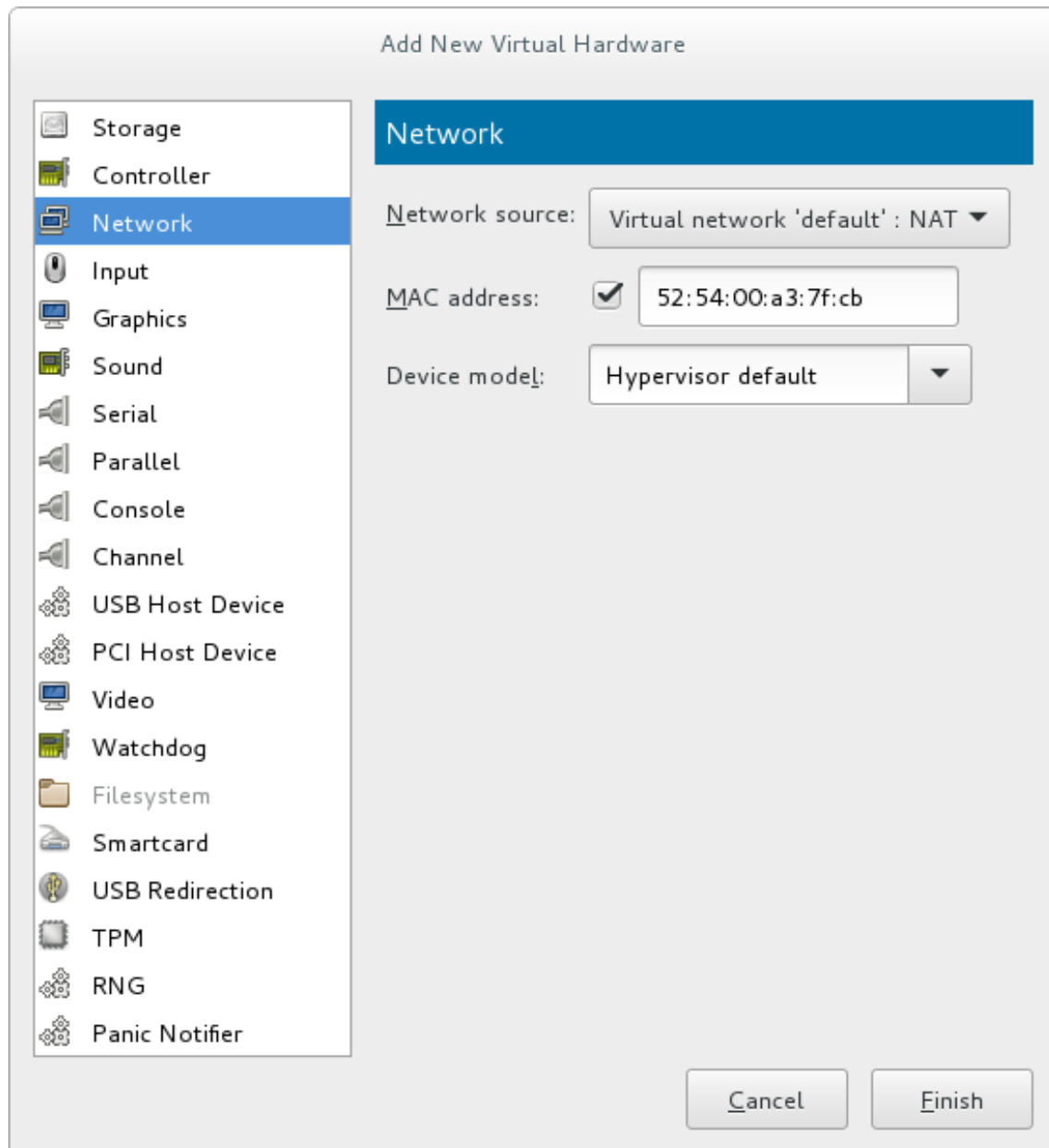


図5.3 新しいハードウェアを追加ウィザード

5. ネットワークデバイスとドライバーを選択します。  
**Device model (デバイスのモデル)** を **virtio** に設定して virtio ドライバーを使用します。  
必要な **ホストデバイス** を選択します。

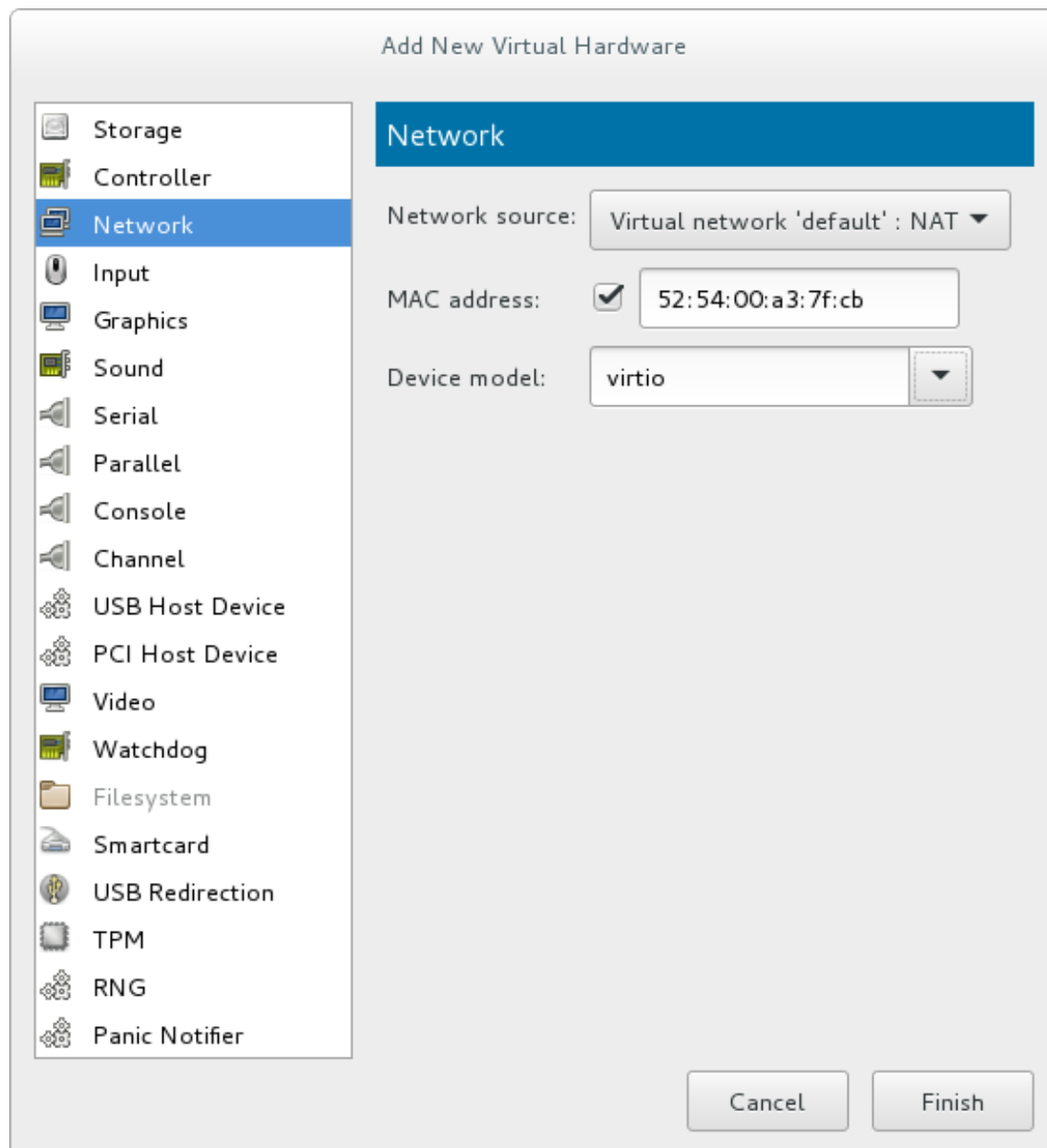


図5.4 新しいハードウェアを追加ウィザード

完了 をクリックして終了します。

すべての新規デバイスが追加されたら、仮想マシンを再起動します。仮想マシンはゲストが再起動するまでデバイスを認識しない可能性があります。

### 5.3. GPU デバイスに KVM VIRTIO ドライバーを使用

**virtio** ドライバーにより、仮想化グラフィックカードデバイスを使うことができます。この機能は **virtio-gpu** と呼ばれ、**virt-viewer**、**remote-viewer**、または **virt-manager** ユーティリティを使ってゲスト仮想マシンのグラフィカルコンソールを表示する際に、パフォーマンスが大幅に向上します。

#### virtio-gpu の有効化

1. システムに以下の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。

- ホストマシン側:

- **qemu**

- virglrenderer
- spice-server
- spice-gtk
- mesa
- libepoxy
- ゲストマシン側:
  - Linux kernel アップストリームバージョン 4.4 またはそれ以降
  - mesa アップストリームバージョン 11.1 またはそれ以降
  - xorg-x11-server-Xorg アップストリームバージョン 1.19 またはそれ以降

2. ゲストの XML ファイルに、以下の行が含まれていることを確認します。

```
<graphics type='spice'>
  <listen type='none' />
  <gl enable='yes' />
</graphics>
<video>
  <model type='virtio' />
</video>
```

3. **virt-viewer** または **remote-viewer** を使ってゲストのコンソールを表示する場合は、**--attach** オプションを使います。このオプションにより、**OpenGL** サービスが機能するのに必要な **unix** ソケット接続が割り当てられます。

```
# virt-viewer --attach {guest-name|id|uuid}
```

**virt-manager** の場合、追加のオプションは必要ありません。

### virtio-gpu アクティベーションの確認

**virtio-gpu** が正常にアクティブ化されたことを確認するには、ゲスト側で以下のコマンドを実行し、同じ表示が出力されるかどうかを確認します。

```
# dmesg | grep '\[drm\]'
...
[drm] virgl 3d acceleration enabled
...

# glxinfo | grep ^OpenGL
...
OpenGL renderer string: Gallium 0.4 on virgl
...
```

## 第6章 ネットワーク設定

この章では、**libvirt** ベースのゲスト仮想マシンが使用する一般的なネットワーク設定について説明します。詳細な情報については、**libvirt** アップストリームの [ネットワークアーキテクチャーに関するドキュメント](#) を参照してください。

Red Hat Enterprise Linux 7 は以下の仮想化ネットワーク設定に対応しています。

- **Network Address Translation (NAT)** を使用した仮想ネットワーク
- **PCI** デバイス割り当てを使用して直接割り当てられた物理デバイス
- **PCIe SR-IOV** を使用して直接割り当てられた仮想機能
- ブリッジネットワーク

ゲスト仮想マシン上のネットワークサービスに外部ホストがアクセスできるようにするには、**NAT** またはネットワークブリッジングを有効にするか、または **PCI** デバイスを直接割り当てる必要があります。

### 6.1. LIBVIRT を使用した NETWORK ADDRESS TRANSLATION (NAT)

ネットワーク接続を共有する最も一般的な方法の1つは、**Network Address Translation (NAT)** 転送 (別名、仮想ネットワーク) の使用です。

#### ホストの設定

標準的な **libvirt** インストールはすべて、デフォルトの仮想ネットワークとして 仮想マシンへの **NAT** ベースの接続を提供します。これが **virsh net-list --all** コマンドで利用可能であることを確認します。

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

これが存在しない場合、以下をゲストの XML 設定ファイル (**/etc/libvirt/qemu/myguest.xml** など) で使用できます。

```
# ll /etc/libvirt/qemu/
total 12
drwx-----. 3 root root 4096 Nov  7 23:02 networks
-rw-----. 1 root root 2205 Nov 20 01:20 r6.4.xml
-rw-----. 1 root root 2208 Nov  8 03:19 r6.xml
```

デフォルトのネットワークは **/etc/libvirt/qemu/networks/default.xml** で定義されています。

デフォルトのネットワークが自動的に開始するようマークします。

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

デフォルトのネットワークを開始します。

■

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

**libvirt** デフォルトのネットワークが実行されると、分離されたブリッジデバイスが表示されます。このデバイスには、物理インターフェースが追加されて **いません**。新規デバイスは、**NAT** および **IP** 転送を使用して物理ネットワークに接続します。新規インターフェースを追加しないでください。

```
# brctl show
bridge name      bridge id                STP enabled    interfaces
virbr0           8000.00000000000000      yes
```

**libvirt** は **iptables** ルールを追加します。このルールは、**INPUT**、**FORWARD**、**OUTPUT**、**POSTROUTING** チェーンで **virbr0** デバイスに割り当てられたゲスト仮想マシンから/へのトラフィックを可能にするものです。次に **libvirt** は、**ip\_forward** パラメーターの有効化を試みます。別のアプリケーションが **ip\_forward** を無効にする場合もあるので、**/etc/sysctl.conf** に以下を追加することが最善の選択肢です。

```
net.ipv4.ip_forward = 1
```

### ゲスト仮想マシンの設定

ホスト設定が完了したら、ゲスト仮想マシンはその名前をベースにした仮想ネットワークに接続できます。ゲストを「デフォルト」の仮想ネットワークに接続するには、ゲストの **XML** 設定ファイル (**/etc/libvirt/qemu/myguest.xml** など) で以下を使用します。

```
<interface type='network'>
  <source network='default' />
</interface>
```

#### 注記

**MAC** アドレスの定義はオプションです。定義されない場合、**MAC** アドレスは自動生成され、ネットワークが使用するブリッジデバイスの **MAC** アドレスとして使用されます。**MAC** アドレスの手動設定は、環境内での一貫性や参照の容易さを保ち、競合の極めて低い可能性を回避するのに役立つことがあります。

```
<interface type='network'>
  <source network='default' />
  <mac address='00:16:3e:1a:b3:4a' />
</interface>
```

## 6.2. VHOST-NET の無効化

**vhost-net** モジュールは **virtio** ネットワーキング用のカーネルレベルのバックエンドで、**virtio** パケット処理タスクをユーザー領域 (**QEMU** プロセス) からカーネル (**vhost-net** ドライバー) に移すことで仮想化のオーバーヘッドを低減します。**vhost-net** は、**virtio** ネットワークインターフェースでのみ利用可能です。**vhost-net** カーネルモジュールがロードされている場合、デフォルトですべての **virtio** インターフェース用に有効化されています。ただし、**vhost-net** の使用時に特定のワークロードのパフォーマンスが低下した場合は、インターフェース設定で無効にできます。

具体的には、**UDP** トラフィックがホストマシンからホスト上のゲスト仮想マシンに送信された場合、ゲスト仮想マシンのデータ処理速度がホストマシンの送信速度より遅いと、パフォーマンスが低下する可能性があります。この状況で **vhost-net** を有効にすると、**UDP** ソケットの受信バッファをより

早くオーバーフローさせることになり、多大なパケットロスにつながります。このため、この状況ではトラフィックを遅らせ、全体のパフォーマンスを上げるために、**vhost-net** を無効にすることが適切です。

**vhost-net** を無効にするには、ゲスト仮想マシンの XML 設定ファイルにある **<interface>** サブ要素を編集し、ネットワークを以下のように定義します。

```
<interface type="network">
  ...
  <model type="virtio"/>
  <driver name="qemu"/>
  ...
</interface>
```

ドライバー名を **qemu** に設定するとパケット処理が **QEMU** ユーザー領域で強制されるため、そのインターフェースでは **vhost-net** が事実上無効になります。

### 6.3. VHOST-NET ZERO-COPY の有効化

Red Hat Enterprise Linux 7 では、**vhost-net zero-copy** はデフォルトで無効にされています。このアクションを永続的に有効にするには、以下の内容を含む新規ファイル **vhost-net.conf** を **/etc/modprobe.d** に追加します。

```
options vhost_net experimental_zcopytx=1
```

これを再び無効にする場合は、以下を実行します。

```
modprobe -r vhost_net
```

```
modprobe vhost_net experimental_zcopytx=0
```

最初のコマンドは古いファイルを削除し、2 目目のファイルは新規ファイル (上記のようなファイル) を作成し、ゼロコピーを無効にします。このコマンドを使って有効にすることもできますが、その場合変更は永続化されません。

有効になったことを確認するには、**cat** **/sys/module/vhost\_net/parameters/experimental\_zcopytx** の出力を確認します。以下のように表示されるはずです。

```
$ cat /sys/module/vhost_net/parameters/experimental_zcopytx
0
```

### 6.4. ブリッジネットワーク

ブリッジネットワーク (ネットワークブリッジングまたは仮想ネットワークスイッチとも呼ばれます) は、仮想マシンのネットワークインターフェースを物理インターフェースと同じネットワーク上に置くために使用されます。ブリッジには最小の設定が必要であり、仮想マシンを既存のネットワーク上に表示させるため、管理オーバーヘッドとネットワークの複雑性が軽減されます。ブリッジにはコンポーネントや設定変数がほとんど含まれていないため、それらは理解しやすく、必要な場合はトラブルシューティングを実行しやすい透過的なセットアップを提供します。

ブリッジは `virt-manager` または `libvirt` などの標準的な Red Hat Enterprise Linux ツールを使用して仮想化環境で設定できます。以下のセクションでこれについて説明します。

ただし、仮想化環境の場合でも、ホストオペレーティングシステムのネットワークツールを使用した方がブリッジをより簡単に作成できる場合があります。このブリッジの作成方法についての詳細は、『[Red Hat Enterprise Linux 7 ネットワークガイド](#)』を参照してください。

### 6.4.1. Red Hat Enterprise Linux 7 ホストでのブリッジネットワークの設定

ブリッジネットワークは、仮想化管理ツールを使わずに Red Hat Enterprise Linux ホスト上の仮想マシンに設定できます。このような設定は主に、仮想化ブリッジがホストの唯一のネットワークインターフェースであるか、またはホストの管理ネットワークインターフェースである場合に推奨されます。

仮想化ツールを使わずにネットワークブリッジングを設定する手順については、『[Red Hat Enterprise Linux 7 ネットワークガイド](#)』を参照してください。

### 6.4.2. 仮想マシンマネージャーを使用したブリッジネットワーク

このセクションでは、`virt-manager` を使用してホストマシンのインターフェースからゲスト仮想マシンへのブリッジを作成する方法を説明します。



#### 注記

ご使用の環境によっては、Red Hat Enterprise Linux 7 で `libvirt` ツールを使用してブリッジをセットアップするためにネットワークマネージャーを無効にする必要がある場合がありますが、これは Red Hat では推奨していません。また、`libvirt` で作成したブリッジでは、ブリッジがネットワークの接続を維持できるよう `libvirtd` が実行中である必要があります。

『[Red Hat Enterprise Linux 7 ネットワークガイド](#)』で説明されているようにブリッジネットワークを Red Hat Enterprise Linux の物理ホストに設定することが推奨されます。ブリッジ作成後に `libvirt` を使用して仮想マシンインターフェースをブリッジに追加します。

#### 手順6.1 `virt-manager` を使用したブリッジの作成

1. `virt-manager` メインメニューから、**編集** ⇒ **接続の詳細** をクリックして **接続の詳細** ウィンドウを開きます。
2. ネットワークインターフェース タブをクリックします。
3. ウィンドウの下部にある **+** をクリックして、新規ネットワークインターフェースを設定します。
4. インターフェースの種類 ドロップダウンメニューで、**Bridge** を選択してから **進む** をクリックして続きます。

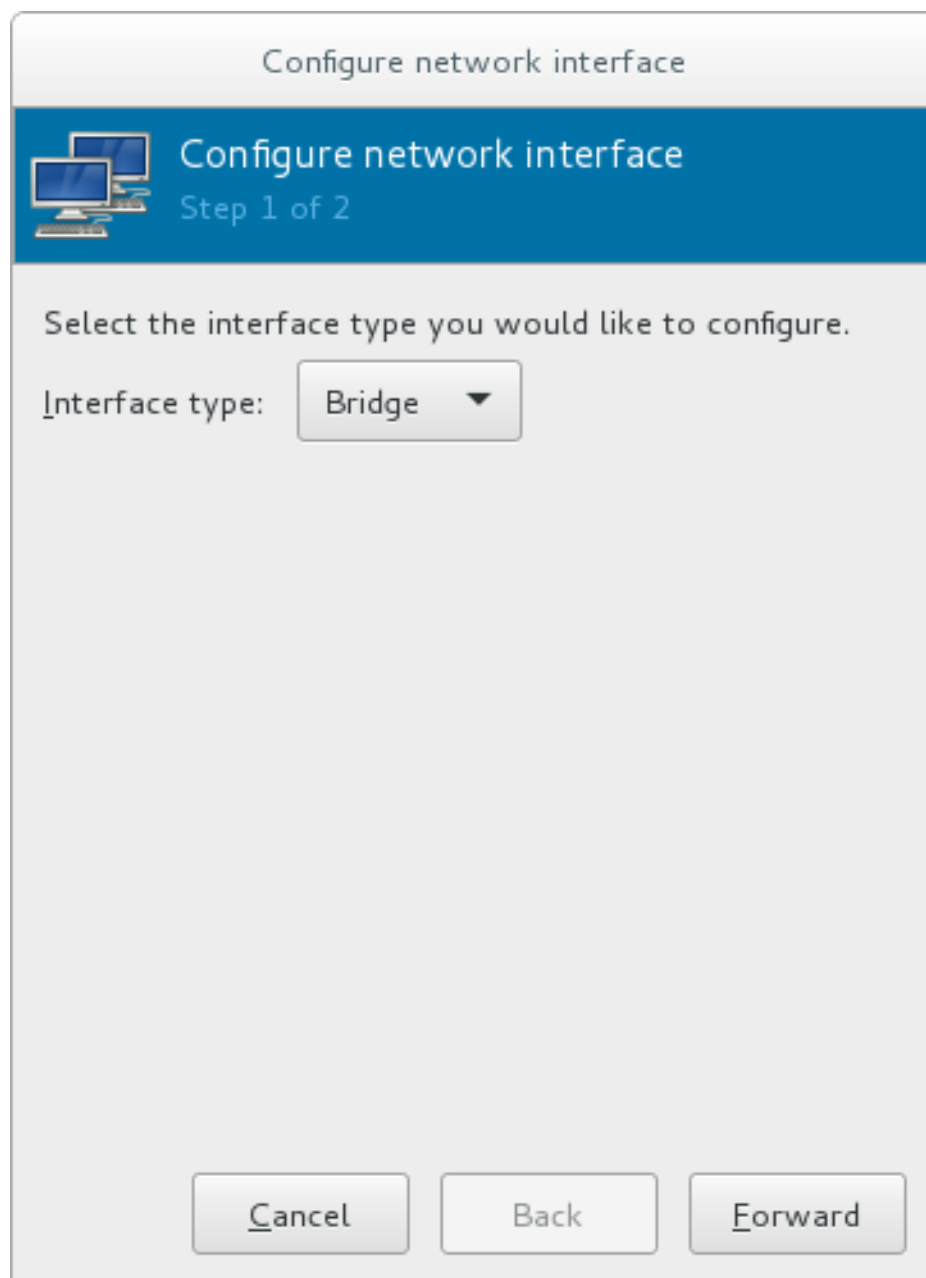


図6.1 ブリッジの追加

5.
  - a. **名前** フィールドに、*br0* などのブリッジの名前を入力します。
  - b. ドロップダウンメニューから **開始モード** を選択します。以下のいずれかから選択します。
    - **none** - ブリッジを非アクティブ化します。
    - **onboot** - 次回のゲスト仮想マシンの再起動時にブリッジをアクティブ化します。
    - **hotplug** - ゲスト仮想マシンが実行中の場合でもブリッジをアクティブ化します。
  - c. **今すぐ有効**にチェックボックスにチェックを入れ、ブリッジをただちにアクティブにします。
  - d. **IPの設定** または **ブリッジの設定** のいずれかを設定するには、該当する**設定** ボタンをクリックします。別のウィンドウが開き、ここで設定を指定することができます。必要な変更を加えてから **OK** をクリックします。



- e. 仮想マシンに接続する物理インターフェースを選択します。インターフェースが別のゲスト仮想マシンで使用されている場合には、警告メッセージが送信されます。
6. **完了** をクリックするとウィザードが閉じます。これにより、**Connections (接続)** メニューに戻ります。

**Configure network interface**

**Configure network interface**  
Step 2 of 2

Name:

Start mode:

Activate now: ☐

IP settings: IPv4: DHCP

Bridge settings: STP on, delay 0.00 sec

Choose interface(s) to bridge:

▼	Name	Type	In use by
<input type="checkbox"/>	lo	ethernet	
<input type="checkbox"/>	wlp4s0	ethernet	
<input type="checkbox"/>	enp0s25	ethernet	
<input type="checkbox"/>	virbr0-nic	ethernet	
<input type="checkbox"/>	virbr1-nic	ethernet	

図6.2 ブリッジの追加

使用するブリッジを選択して **適用** をクリックし、ウィザードを終了します。

インターフェースを停止するには、**インターフェースの停止** キーをクリックします。ブリッジが停止した後にインターフェースを削除するには、**インターフェースの削除** キーをクリックします。

### 6.4.3. libvirt を使用したブリッジネットワーク

ご使用の環境によっては、Red Hat Enterprise Linux 7 で `libvirt` ツールを使用してブリッジをセットアップするためにネットワークマネージャーを無効にする必要がある場合がありますが、これは Red Hat では推奨していません。また、ブリッジが作動するには `libvirtd` が実行中である必要があります。

『Red Hat Enterprise Linux 7 ネットワークガイド』で説明されているようにブリッジネットワークを Red Hat Enterprise Linux の物理ホストに設定することが推奨されます。

## 重要

`libvirt` は新規のカーネル調整可能パラメーターを使用してホストブリッジのフォワーディングデータベース (FDB) エントリーを管理できるため、複数仮想マシンでのブリッジの実行時にシステムのネットワークのパフォーマンスが強化されます。ホストの XML 設定ファイルでネットワークの **<bridge>** 要素の **macTableManager** 属性を **'libvirt'** に設定します。

```
<bridge name='br0' macTableManager='libvirt' />
```

これにより、すべてのブリッジポートで学習 (フラッド) モードがオフになり、`libvirt` は必要に応じて FDB に対するエントリーの追加または削除を行います。MAC アドレスの適切な転送ポートについての学習に関するオーバーヘッドを削減すると共に、カーネルがネットワークへのブリッジ接続をしている物理デバイスでプロミスカスモードを無効にできるため、さらにオーバーヘッドが削減されます。

## 第7章 KVM でのオーバーコミット

### 7.1. はじめに

KVM ハイパーバイザーにより、CPU およびメモリーが自動的にオーバーコミットされます。つまり、システム上の物理リソース以上に、仮想化 CPU およびメモリーを仮想マシンに割り当てることが可能です。このことが可能になる理由は、ほとんどのプロセスで割り当てられたリソースを常に 100% 必要としている訳ではないからです。

その結果、使用率の低い仮想化サーバーやデスクトップをより少数のホストで実行することができるため、リソースとしてのシステム数を節約でき、節電効果や冷却効果、およびサーバーのハードウェアに対する投資効果などの実質的な効果を得ることができます。

### 7.2. メモリーのオーバーコミット

KVM ハイパーバイザーで実行しているゲスト仮想マシン群には、そのマシン群専用に割り当てられた物理的な RAM ブロックはありません。代わりに、各ゲスト仮想マシンはホスト物理マシンの Linux プロセスとして動作します。つまり、メモリーが要求された場合にのみホスト物理マシンの Linux カーネルによってメモリーが割り当てられます。また、ホストのメモリー管理機能により、物理的なメモリーと swap 領域間でゲスト仮想マシンのメモリーを移動させることができます。

オーバーコミットを採用する際は、ホスト物理マシンのプロセス処理用に十分なメモリーを確保するだけでなく、すべてのゲスト仮想マシンに対応できるようにホスト物理マシン上に十分な swap 領域を配分する必要があります。原則として、ホスト物理マシンのオペレーティングシステムには最大 4GB のメモリーと最小 4GB の swap 領域が必要となります。適切な swap パーティションサイズ把握に関する詳細な説明は、[Red Hat ナレッジベース](#) を参照してください。



#### 重要

オーバーコミットが、全般的なメモリー関連の問題に対する理想的なソリューションとなるわけではありません。メモリー不足に対処するため推奨される方法は、ゲストにより少ないメモリーを割り当てるか、ホストにより多くの物理メモリーを追加するか、または swap 領域を使用することです。

スワップが頻繁に行われると、仮想マシンの動作が遅くなる場合があります。また、オーバーコミットによりシステムがメモリーを使い果たし (OOM)、Linux カーネルが重要なプロセスをシャットダウンする可能性があります。メモリーのオーバーコミットを実行する場合は、テストを十分に実施するようにしてください。オーバーコミットに関して支援が必要な場合は Red Hat サポートにお問い合わせください。

オーバーコミットはすべての仮想マシンで機能する訳ではありませんが、集中的なメモリー使用が最小限となるデスクトップ仮想化のセットアップや、複数の同一設定のゲスト仮想マシンを KSM で実行する場合などに有効であることが確認されています。KSM およびオーバーコミットの詳細については、『[Red Hat Enterprise Linux 7 仮想化のチューニングと最適化ガイド](#)』を参照してください。



#### 重要

[デバイス割り当て](#)が使用されている場合、割り当てデバイスで直接メモリーアクセス (DMA) を有効にするには、すべての仮想マシンメモリーの静的な事前割り当てを行う必要があります。このため、メモリーのオーバーコミットはデバイス割り当てと一緒にサポートされません。

### 7.3. 仮想化 CPU のオーバーコミット

KVM ハイパーバイザーは、仮想化 CPU (vCPU) のオーバーコミットに対応しています。仮想化 CPU は、ゲスト仮想マシンで許可される負荷の限界までオーバーコミットすることができます。vCPU をオーバーコミットする際は、負荷が 100% に近づくと要求がドロップされたり、長すぎる応答時間が発生する恐れがあるため十分に注意してください。

Red Hat Enterprise Linux 7 では、複数の仮想 CPU を持つゲスト (対称型マルチプロセッシング (SMP) のゲスト仮想マシンとも呼ばれる) をオーバーコミットできます。ただし、実際の物理 CPU にある数よりも多くのコアを仮想マシンで実行すると、パフォーマンスが大幅に低下する可能性があります。

たとえば、仮想 CPU 数が 4 つの仮想マシンは、デュアルコアではなくクアッドコアプロセッサのホストマシンで実行すべきです。実際の物理プロセッシングコア数を超えた SMP 仮想マシンをオーバーコミットすると、プログラムに割り当てられる CPU 時間が必要な時間に満たなくなるためにパフォーマンスが大幅に低下する原因となります。また、物理プロセッサコア 1 つあたり合計 10 個を超える仮想 CPU を割り当てることは推奨されません。

SMP ゲストでは、一部の固有の処理オーバーヘッドが引き継がれます。タイムスライスを使用してリソースをゲストに割り当てるとゲスト内の CPU 間の通信速度が低下する可能性があるため、CPU のオーバーコミットにより SMP オーバーヘッドが増える可能性があります。このオーバーヘッドは、多数の仮想 CPU を持つか、またはオーバーコミットの割合の高いゲストの場合に増加します。

仮想化 CPU のオーバーコミットを行う最適な状態は、単一のホストに複数のゲストがあり、各ゲストにホスト CPU 数と比較した場合の少数の仮想 CPU (vCPU) がある場合です。KVM が安全にサポートするのは単一ホストにおいて、負荷が 100% 未満のゲストで、(5 台の仮想マシン上の) 5 つの仮想 CPU に対して 1 つの物理 CPU という割合の場合です。KVM はすべての仮想マシン間で切り替えを実行し、負荷のバランスを取ります。

最適なパフォーマンスを得るために、各ゲスト内のプログラムを実行するのに必要な数の仮想 CPU のみをゲストに割り当てることを推奨します。



#### 重要

オーバーコミットしている環境では、メモリーやプロセッシングリソースを 100% 使用するアプリケーションは不安定になる可能性があります。実稼働環境でメモリーまたは CPU をオーバーコミットする際は、十分なテストを行ってから実行してください。CPU のオーバーコミットの割合と SMP の量はワークロードによって異なるためです。

## 第8章 KVM ゲストのタイミング管理

仮想化には、ゲスト仮想マシンの時刻管理におけるさまざまな課題が伴います。

- 仮想マシン内の割り込みは実際の割り込みではなく、ホストマシンがゲスト仮想マシンに挿入しているものです。このため、割り込みは常に同時かつ即時にすべてのゲスト仮想マシンに配信される訳ではありません。
- ホストは別のゲスト仮想マシンを実行していたり、別のプロセスを実行している場合もあり、したがって割り込みに通常必要となる正確なタイミングを得ることが常に可能であるとは限りません。

セッションの妥当性や移行その他のネットワークアクティビティの正確性を維持するにはタイムスタンプに依存する必要があるため、正確な時刻管理機能を実行していないゲスト仮想マシンには、ネットワークアプリケーションおよびプロセスに関連した問題が発生する可能性があります。

KVM では、ゲスト仮想マシンに準仮想化クロック (**kvm-clock**) を提供することでこの問題を回避します。しかし、タイミングをテストしてから、時刻管理が不正確な場合に影響を受ける可能性のあるアクティビティ (ゲストマイグレーション等) を実行することが依然として重要です。

### 重要

上記の問題を避けるためには、ホストおよびゲスト仮想マシンに **Network Time Protocol (NTP)** を設定する必要があります。Red Hat Enterprise Linux 6 およびそれ以前のバージョンを使っているゲストの場合、NTP は **ntpd** サービスで実装されます。詳細は、『[Red Hat Enterprise Linux 6 導入ガイド](#)』を参照してください。

Red Hat Enterprise Linux 7 を使っているシステムの場合、NTP 時刻同期サービスは **ntpd** または **chronyd** サービスにより提供されます。Chrony には、仮想マシンに対していくつかの優れた点があります。詳細は、『Red Hat Enterprise Linux 7 システム管理者のガイド』の「[chrony スイートを使用した NTP 設定](#)」および「[ntpd を使用した NTP 設定](#)」の章を参照してください。

### ゲスト仮想マシンの同期のしくみ

デフォルトで、ゲストは以下のようにハイパーバイザーと時間を同期します。

- ゲストシステムが起動すると、ゲストはエミュレートされた **RTC** (リアルタイムクロック) から時間を読み取ります。
- NTP プロトコルが開始されていると、ゲストのクロックは自動的に同期されます。それ以降、ゲストが通常の動作をしている間、NTP はゲストのクロック調整を実施します。
- ゲストが一時停止または復元プロセスの後に再起動する場合は、管理ソフトウェア (**virt-manager** 等) により、ゲストのクロックを特定の値に同期させるコマンドが発行される必要があります。この同期プロセスは、**QEMU** **ゲストエージェント** がゲストにインストールされ、機能をサポートしている場合に限り動作します。通常は、ホストのクロック値にゲストのクロックを同期させます。

### 一定のタイムスタンプカウンタ (TSC: Time Stamp Counter)

最新の Intel と AMD の CPU は、不変タイムスタンプカウンタ (TSC) を提供します。不変 TSC のカウント頻度は、たとえば CPU コア自体が節電ポリシーに従うために周波数を変更しても変わりません。不変 TSC の周波数を持つ CPU は、KVM ゲストのクロックソースとして TSC を使用するために必要です。

**constant\_tsc** フラグが付いている場合には、CPU は Constant TSC を搭載しています。CPU に

**constant\_tsc** フラグが付いているかどうかを確認するには、以下のコマンドを実行します。

```
$ cat /proc/cpuinfo | grep constant_tsc
```

いずれかの出力が表示される場合は、CPU には **constant\_tsc** ビットがあることになります。出力がない場合は、以下の説明に従ってください。

### Constant TSC を搭載していないホストの設定

Constant TSC の周波数のないシステムは、仮想マシンのクロックソースに TSC を使用できず、追加設定が必要になります。電源管理機能により正確な時刻管理が妨げられるので、KVM を使用してゲスト仮想マシンの正確な時間を管理するには、この機能を無効にする必要があります。



#### 重要

以下に説明する手順は、AMD リビジョン F の CPU のみが対象となります。

CPU に **constant\_tsc** ビットがない場合には、電源管理機能をすべて無効にします。各システムには、時刻管理に使用する複数のタイマーがあります。TSC はホスト上では安定していません。これは、**cpufreq** の変化やディープ C ステート、より高速の TSC を搭載したホストへの移行などが原因となる場合があります。ディープ C スリープ状態に入ると、TSC が停止する可能性があります。カーネルがディープ C ステートを使用しないようにするには、カーネルブートに **processor.max\_cstate=1** を追加します。この変更を永続化させるには、**/etc/default/grub** ファイルの **GRUB\_CMDLINE\_LINUX** キーの値を編集します。たとえば、毎回の起動時に緊急モード (emergency mode) を有効にするには、以下のようにエントリーを編集します。

```
GRUB_CMDLINE_LINUX="emergency"
```

GRUB 2 ブートメニューに複数のパラメーターを追加する際と同様に、**GRUB\_CMDLINE\_LINUX** キーには複数のパラメーターを指定できることに注意してください。

**cpufreq** を無効にするには (**constant\_tsc** なしのホスト上の場合のみ)、**kernel-tools** をインストールし、**cpupower.service** を有効にします (**systemctl enable cpupower.service**)。ゲスト仮想マシンが起動されるたびにこのサービスを無効にするには、**/etc/sysconfig/cpupower** の設定ファイルを変更し、**CPUPOWER\_START\_OPTS** および **CPUPOWER\_STOP\_OPTS** を変更します。有効な制限について

は、**/sys/devices/system/cpu/cpuid/cpufreq/scaling\_available\_governors** ファイルを参照してください。このパッケージまたは電源管理およびガバナーについての詳細は、『[Red Hat Enterprise Linux 7 電力管理ガイド](#)』を参照してください。

## 8.1. RED HAT ENTERPRISE LINUX ゲストに必要な時刻管理パラメーター

一部の Red Hat Enterprise Linux ゲスト仮想マシンでは、システムの時刻を正しく同期させるために追加のカーネルパラメーターが必要です。これらのパラメーターは、ゲスト仮想マシンの **/etc/grub2.cfg** ファイルの **/kernel** 行の末尾に追記することで設定できます。



#### 注記

Red Hat Enterprise Linux 5.5 以降、Red Hat Enterprise Linux 6.0 以降、および Red Hat Enterprise Linux 7 は、デフォルトのクロックソースに **kvm-clock** を使用します。**kvm-clock** を実行すると追加のカーネルパラメーターが必要なくなるので、Red Hat ではこれを推奨しています。



以下の表は、Red Hat Enterprise Linux のバージョンと各システムで必要となるパラメーターを表示しています。

表8.1 カーネルパラメーターの要件

Red Hat Enterprise Linux バージョン	ゲストの追加カーネルパラメーター
AMD64 および Intel 64 システム上の 7.0 およびそれ以降 (kvm-clock を実行)	追加のパラメーターは必要ありません
AMD64 および Intel 64 システム上の 6.1 およびそれ以降 (kvm-clock を実行)	追加のパラメーターは必要ありません
AMD64 および Intel 64 システム上の 6.0 (kvm-clock を実行)	追加のパラメーターは必要ありません
AMD64 および Intel 64 システム上の 6.0 (kvm-clock を実行せず)	notsc lpj= <i>n</i>
AMD64 および Intel 64 システム上の 5.5 (kvm-clock を実行)	追加のパラメーターは必要ありません
AMD64 および Intel 64 システム上の 5.5 (kvm-clock を実行せず)	notsc lpj= <i>n</i>
32 ビット AMD および Intel システム上の 5.5 (kvm-clock を実行)	追加のパラメーターは必要ありません
32 ビット AMD および Intel システム上の 5.5 (kvm-clock を実行せず)	clocksource=acpi_pm lpj= <i>n</i>
AMD64 および Intel 64 システム上の 5.4	notsc
32 ビット AMD および Intel システム上の 5.4	clocksource=acpi_pm
AMD64 および Intel 64 システム上の 5.3	notsc
32 ビット AMD および Intel システム上の 5.3	clocksource=acpi_pm



## 注記

**lpj** パラメーターは、ゲスト仮想マシンが稼働する特定の CPU の **loops per jiffy** 値と同じ値を必要とします。この値が不明な場合は、**lpj** パラメーターを設定しないでください。



## 警告

**divider** カーネルパラメーターはこれまで、高い即応性要件のない Red Hat Enterprise Linux 4 および 5 ゲスト仮想マシンやゲスト密度の高いシステム上にある Red Hat Enterprise Linux 4 および 5 ゲスト仮想マシンに推奨されてきました。しかし現在は、Red Hat Enterprise Linux 4 およびバージョン 5.8 より前の Red Hat Enterprise Linux 5 バージョンを実行するゲストでこのカーネルパラメーターを使用することは推奨されていません。

Red Hat Enterprise Linux 5 のバージョン 5.8 およびそれ以降において、**divider** は、タイマー割り込みの頻度を下げることでスループットを改善できます。たとえば、**HZ=1000** の場合に **divider** が **10** に設定される (つまり、**divider=10**) と、期間ごとのタイマー割り込み数がデフォルト値 (1000) から 100 (デフォルト値の  $1000 \div \text{divider 値 } 10$ ) に変わります。

[BZ#698842](#) では、**divider** パラメーターによる割り込みや tick recording との対話について記述されています。この不具合は Red Hat Enterprise Linux 5.8 で修正されています。ただし、Red Hat Enterprise Linux 4 またはバージョン 5.8 より前の Red Hat Enterprise Linux 5 バージョンを使用するゲストでは、依然として **divider** パラメーターによるカーネルパニックが発生する可能性があります。

Red Hat Enterprise Linux 6 以降には、割り込み数が固定されたクロック割り込みはありません。このバージョンはティックレスモード (tickless mode) で動作し、随時タイマーを動的に使用します。**divider** パラメーターは Red Hat Enterprise Linux 6 および Red Hat Enterprise Linux 7 では有用ではないため、これらのシステム上にあるゲストはこの不具合による影響を受けません。

## 8.2. スチールタイムアカウンティング

スチールタイムは、ゲスト仮想マシンが必要とする CPU 時間の内のホストが提供していない時間で、スチールタイムは、ホストがこれらのリソースを別のゲストなどに割り当てる場合に発生します。

スチールタイムは、**/proc/stat** の CPU 時間フィールドに報告されます。これは、**top** や **vmstat** などのユーティリティーによって自動的に報告されます。これは「%st」として表示されるか、または「st」列に表示されます。スチールタイムをオフにすることはできないことに注意してください。

スチールタイムが多いと CPU の競合を生じさせ、ゲストのパフォーマンス低下につながる可能性があります。CPU の競合を軽減するには、ゲストの CPU 優先度または CPU 割り当てのレベルを上げるか、またはホスト上で実行するゲスト数を減らします。



## 第9章 LIBVIRT を使用したネットワークブート

ゲスト仮想マシンは、PXE を有効にして起動できます。PXE により、ゲスト仮想マシンの起動が可能になり、ネットワーク自体から設定をロードできるようになります。このセクションでは、libvirt を使って PXE ゲストを設定する基本的なステップを説明します。

このセクションでは、ブートイメージの作成や PXE サーバーは説明されません。ここでは、プライベートまたはブリッジネットワークで libvirt を設定し、PXE ブートを有効にしてゲスト仮想マシンを起動する方法を説明します。



### 警告

ここでの手順は、例としてのみ示されています。次に進む前に、十分なバックアップがなされていることを確認してください。

### 9.1. ブートサーバーの準備

本章のステップを実行するには、以下が必要となります。

- PXE サーバー (DHCP および TFTP) - これは libvirt 内部サーバー、手動設定の DHCP および TFTP、dnsmasq、Cobbler 設定のサーバー、他のサーバーのいずれでも可能です。
- ブートイメージ - 手動設定または Cobbler 設定の PXELINUX

#### 9.1.1. プライベート libvirt ネットワーク上での PXE ブートサーバーの設定

以下の例ではデフォルト ネットワークを使用します。以下のステップを実行してください。

##### 手順9.1 PXE ブートサーバーの設定

1. PXE ブートイメージおよび設定内容を `/var/lib/tftpboot` に配置します。
2. 以下のコマンドを入力します。

```
# virsh net-destroy default
# virsh net-edit default
```

3. デフォルト ネットワークの設定ファイルで `<ip>` 要素を編集し、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組み込みます。ここで、`BOOT_FILENAME` はゲスト仮想マシンの起動に使用するファイル名を表します。

```
<ip address='192.168.122.1' netmask='255.255.255.0'>
  <tftp root='/var/lib/tftpboot' />
  <dhcp>
    <range start='192.168.122.2' end='192.168.122.254' />
    <bootp file='BOOT_FILENAME' />
  </dhcp>
</ip>
```

4. 以下を実行します。

```
# virsh net-start default
```

5. PXE を使用してゲストを起動します (「[PXE を使用したゲストの起動](#)」を参照)。

## 9.2. PXE を使用したゲストの起動

このセクションでは、PXE を使用してゲスト仮想マシンを起動する方法を説明します。

### 9.2.1. ブリッジネットワークの使用

#### 手順9.2 PXE およびブリッジネットワークを使用したゲストの起動

1. ブリッジが有効にされており、PXE ブートサーバーがネットワーク上で利用可能なことを確認します。
2. PXE ブートが有効な状態でゲスト仮想マシンを起動します。以下のコマンド例のように、**virt-install** コマンドを使用して PXE ブートが有効にされている新規の仮想マシンを作成することができます。

```
virt-install --pxe --network bridge=breth0 --prompt
```

または、ゲストネットワークがブリッジネットワークを使用するように設定されており、以下の例のように XML ゲスト設定ファイルの **<os>** 要素内に **<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>
<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>
```

### 9.2.2. プライベート libvirt ネットワークの使用

#### 手順9.3 プライベート libvirt ネットワークの使用

1. 「[プライベート libvirt ネットワーク上での PXE ブートサーバーの設定](#)」で説明されているように libvirt 上で PXE ブートを設定します。
2. PXE ブートが有効な状態で libvirt を使用してゲスト仮想マシンを起動します。以下のように **virt-install** コマンドを実行し、PXE を使用して新規の仮想マシンを作成し、インストールすることができます。

```
virt-install --pxe --network network=default --prompt
```

または、ゲストネットワークがプライベートネットワークを使用するように設定されており、以下の例のように XML ゲスト設定ファイルの `<os>` 要素内に `<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>
```

また、ゲスト仮想マシンがプライベートネットワークに接続されていることを確認します。

```
<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>
```

## 第10章 ハイパーバイザーおよび仮想マシンの登録

Red Hat Enterprise Linux 6 および 7 では、すべてのゲストに同じレベルのサブスクリプションサービスが割り当てられるようにすべてのゲスト仮想マシンが特定のハイパーバイザーにマップされる必要があります。これを実行するには、インストールおよび登録済みの各 KVM ハイパーバイザー上のすべてのゲスト仮想マシン (VM) を自動検出するサブスクリプションエージェントをインストールする必要があります。これにより、ホスト上に置かれるマッピングファイルが作成されます。このマッピングファイルにより、すべてのゲスト仮想マシンに以下の利点を提供されます。

- 仮想システムに固有のサブスクリプションはすぐに利用可能となり、関連付けられた仮想マシンすべてに適用できます。
- ハイパーバイザーから継承できるサブスクリプションの利点すべてはすぐに利用可能となり、関連付けられた仮想マシンすべてに適用できます。



### 注記

本章で提供される情報は Red Hat Enterprise Linux サブスクリプションのみに該当します。Red Hat Virtualization サブスクリプションまたは Red Hat Satellite サブスクリプションをお持ちの場合には、それらのサブスクリプションと共に提供されている **virt-who** 情報も参照してください。Red Hat Subscription Management についての詳細は、カスタマーポータルから入手できる [Red Hat Subscription Management](#) に関するガイドも参照してください。

### 10.1. ホスト物理マシンへの VIRT-WHO のインストール

#### 1. KVM ハイパーバイザーを登録します。

KVM ハイパーバイザーは、ホスト物理マシンのターミナルで root ユーザーとして **subscription-manager register [options]** コマンドを実行して登録します。# **subscription-manager register --help** メニューを使用するとさらに多くのオプションを利用できます。ユーザー名とパスワードを使用する場合には、サブスクリプションマネージャーが認識する認証情報を使用します。サブスクリプションの初回使用時にユーザーアカウントがない場合には、カスタマーサポートにお問い合わせください。'admin' としてパスワード 'secret' を使用して仮想マシンを登録する場合は以下のコマンドを送信します。

```
[root@rhel-server ~]# subscription-manager register --username=admin
--password=secret --auto-attach --type=hypervisor
```

#### 2. virt-who パッケージをインストールします。

ホスト物理マシンで以下のコマンドを実行することで **virt-who** パッケージをインストールします。

```
# yum install virt-who
```

#### 3. virt-who 設定ファイルを作成します。

設定ファイルを **/etc/virt-who.d/** ディレクトリーに追加します。ファイルには任意の名前を付けることができますが、意味のある名前を付ける必要があります。ファイルは **/etc/virt-who.d/** ディレクトリーに置く必要があります。ファイル内に以下のスニペットを追加し、ファイルを保存してから閉じてください。

```
[libvirt]
type=libvirt
```

#### 4. virt-who サービスを起動します。

ホスト物理マシンで以下のコマンドを実行することで virt-who サービスを起動します。

```
# systemctl start virt-who.service
# systemctl enable virt-who.service
```

#### 5. virt-who サービスがゲスト情報を受信していることを確認します。

この時点で、virt-who サービスはホストからドメインの一覧の収集を開始します。ホスト物理マシンで `/var/log/rhsm/rhsm.log` ファイルをチェックし、ファイルにゲストの一覧が含まれることを確認します。以下は例になります。

```
2015-05-28 12:33:31,424 DEBUG: Libvirt domains found: [{'guestId':
'58d59128-cfbb-4f2c-93de-230307db2ce0', 'attributes': {'active': 0,
'virtWhoType': 'libvirt', 'hypervisorType': 'QEMU'}, 'state': 5}]
```

### 手順10.1 カスタマーポータルでのサブスクリプション管理

#### 1. ハイパーバイザーのサブスクリライブ

仮想マシンはハイパーバイザーと同じサブスクリプションの利点を受けることができるため、ハイパーバイザーに有効なサブスクリプションがあることを確認し、そのサブスクリプションを仮想マシンが使用できることを確認するのは重要です。

##### a. カスタマーポータルにログインします。

Red Hat カスタマーポータル <https://access.redhat.com/> にログインし、ページ先頭にある **サブスクリプション** ボタンをクリックします。

##### b. システムのリンクをクリックします。

**サブスクリライバーのインベントリー** セクション (ページの下の方に位置) の **システム** リンクをクリックします。

##### c. ハイパーバイザーを選択します。

「システム」ページには、すべてのサブスクリライブ済みシステムの表があります。ハイパーバイザーの名前 (例: `localhost.localdomain`) をクリックします。 **Attach a subscription (サブスクリプションのアタッチ)** をクリックし、一覧表示されているすべてのサブスクリプションを選択します。 **Attach Selected (選択項目のアタッチ)** をクリックします。これにより、ホストの物理サブスクリプションがハイパーバイザーに割り当てられ、ゲストがサブスクリプションの利点を受けられるようになります。

#### 2. ゲスト仮想マシンのサブスクリライブ: 初回時の使用

このステップは、新規サブスクリプションを持つが、一度もゲスト仮想マシンをサブスクリライブしたことのない場合に適用できます。仮想マシンを追加する場合にはこのステップを省略してください。virt-who サービスを実行するマシン上でハイパーバイザープロファイルに割り当てられたサブスクリプションを消費するには、ゲスト仮想マシンのターミナルで `root` で以下のコマンドを実行して自動サブスクリライブを実行します。

```
[root@virt-who ~]# subscription-manager attach --auto
```

#### 3. 追加ゲスト仮想マシンのサブスクリライブ

サブスクリライブを初めて実行した場合にはこのステップを省略してください。仮想マシンを追加している場合は、このコマンドを実行しても、必ずしも同じサブスクリプションがゲスト仮想マシンに再アタッチされる訳ではないことに注意してください。これは、特定のゲスト仮想マシンに必要な分を解決するためにすべてのサブスクリプションを削除してから自動アタッチを許可することで、以前と異なるサブスクリプションが消費される可能性があるためです。お使いのシステムには影響しない場合もありますが、この点には留意する必要があります。以下

に説明していませんが、アタッチの手動の手順に従って仮想マシンをアタッチする場合、自動アタッチが機能しないため、それらの仮想マシンを手動で再アタッチする必要があります。以下のコマンドを使用して、古いゲストのサブスクリプションを削除してから自動アタッチを使用してサブスクリプションをすべてのゲストにアタッチします。ゲスト仮想マシンで以下のコマンドを実行します。

```
[root@virt-who ~]# subscription-manager remove --all
[root@virt-who ~]# subscription-manager attach --auto
```

#### 4. サブスクリプションがアタッチされていることを確認します。

ゲスト仮想マシンで以下のコマンドを実行して、サブスクリプションがハイパーバイザーに割り当てられていることを確認します。

```
[root@virt-who ~]# subscription-manager list --consumed
```

以下のような出力が表示されます。サブスクリプションの詳細に注意してください。「Subscription is current」と記載されているはずです。

```
[root@virt-who ~]# subscription-manager list --consumed
+-----+
Consumed Subscriptions
+-----+
Subscription Name: Awesome OS with unlimited virtual guests
Provides:    Awesome OS Server Bits
SKU:        awesomeos-virt-unlimited
Contract:    0
Account:     ##### Your account number #####
Serial:      ##### Your serial number #####
Pool ID:     XYZ123
```

1

```
Provides Management: No
Active:    True
Quantity Used:    1
Service Level:
Service Type:
Status Details:  Subscription is current
```

2

```
Subscription Type:
Starts:    01/01/2015
Ends:      12/31/2015
System Type:    Virtual
```

- |   |  |
|---|--|
| ❶ | システムにアタッチするサブスクリプションのIDがここに表示されます。IDは手動でサブスクリプションをアタッチする場合に必要になります。  |
| ❷ | サブスクリプションが最新であるかどうかを示します。サブスクリプションが最新でない場合、エラーメッセージが表示されます。1つの例として、どのホストにも報告されていないゲストが一時的なマップされていないゲストのサブスクリプションを使用しているとします。この場合、ゲストをサブスクライブする必要があります。この場合は「 <a href="#">サブスクリプションステータスでエラーが出ました。どうすればよいですか?</a> 」に記載されている情報を参照してください。 |

## 5. 追加ゲストの登録

ハイパーバイザーに新規のゲスト仮想マシンをインストールする場合、ゲスト仮想マシンで以下のコマンドを実行して新規の仮想マシンを登録し、ハイパーバイザーにアタッチされているサブスクリプションを使用する必要があります。

```
# subscription-manager register
# subscription-manager attach --auto
# subscription-manager list --consumed
```

## 10.2. 新規ゲスト仮想マシンの登録

新規ゲスト仮想マシンがすでに登録済みで実行中のホストで作成される場合には、**virt-who** サービスも実行中である必要があります。これにより、**virt-who** サービスがゲストをハイパーバイザーにマップできるため、システムは仮想システムとして適切に登録されます。仮想マシンを登録するには、以下のコマンドを実行します。

```
[root@virt-server ~]# subscription-manager register --username=admin --
password=secret --auto-attach
```

## 10.3. ゲスト仮想マシンのエントリーの削除

ゲスト仮想マシンが実行中の場合、ゲストのターミナルで **root** として以下のコマンドを実行し、システムの登録を解除します。

```
[root@virt-guest ~]# subscription-manager unregister
```

システムが削除されている場合は、仮想サービスはサービスが削除されたのか、または一時停止になっているのかを判別できません。その場合、以下のステップに従って、システムをサーバー側から手動で削除する必要があります。

1. サブスクリプションマネージャーにログインします。  
サブスクリプションマネージャーは [Red Hat カスタマーポータル](#) 上にあります。自分の名前とパスワードを使用し、画面上部のログインアイコンをクリックしてカスタマーポータルにログインします。
2. 「サブスクリプション」タブをクリックします。  
サブスクリプション タブをクリックします。
3. システムのリンクをクリックします。  
ページをスクロールダウンして **システム** リンクをクリックします。
4. システムを削除します。  
プロフィールを削除するには、表内で指定されたシステムのプロファイルを見つけ、その名前の横にあるチェックボックスを選択して **Delete (削除)** をクリックします。

## 10.4. VIRT-WHO の手動インストール

このセクションでは、ハイパーバイザーが提供するサブスクリプションを手動でアタッチする方法について説明します。

### 手順10.2 サブスクリプションを手動でアタッチする方法



## 1. サブスクリプション情報を一覧表示し、プール ID を見つけます。

まず、仮想タイプの利用可能なサブスクリプションを一覧表示する必要があります。以下のコマンドを入力します。

```
[root@server1 ~]# subscription-manager list --avail --match-
installed | grep 'Virtual' -B12
Subscription Name: Red Hat Enterprise Linux ES (Basic for
Virtualization)
Provides:
    Red Hat Beta
    Oracle Java (for RHEL Server)
    Red Hat Enterprise Linux Server
SKU:
-----
Pool ID:
XYZ123
Available:
40
Suggested:
1
Service Level:
Basic
Service Type:
L1-L3
Multi-Entitlement: No
Ends:
01/02/2017
System Type:
Virtual
```

表示されるプール ID に注目してください。この ID は次のステップで必要になるのでコピーしておいてください。

## 2. プール ID を選択してサブスクリプションをアタッチします。

直前のステップでコピーしておいたプール ID を使用して、**attach** コマンドを実行します。プール ID XYZ123 を取得したプール ID に置き換えます。以下のコマンドを入力します。

```
[root@server1 ~]# subscription-manager attach --pool=XYZ123

Successfully attached a subscription for: Red Hat Enterprise Linux
ES (Basic for Virtualization)
```

## 10.5. VIRT-WHO のトラブルシューティング

### 10.5.1. ハイパーバイザーのステータスが「red」になっているのはなぜですか？

シナリオ: サーバー側では、サブスクリプションを持たないハイパーバイザーにゲストをデプロイします。24 時間後にはハイパーバイザーはそのステータスを「red」とします。この状況を解決するには、そのハイパーバイザー用にサブスクリプションを取得する必要があります。または、ゲストをサブスクリプションを持つハイパーバイザーに永久的に移行します。

### 10.5.2. サブスクリプションステータスでエラーが出ました。どうすればよいですか？

シナリオ: 以下のエラーメッセージのいずれかが表示されます。

- System not properly subscribed (システムが正しくサブスクライブされていません)
- Status unknown (ステータス不明)
- Late binding of a guest to a hypervisor through virt-who (host/guest mapping) (virt-who によるゲストのハイパーバイザーへの遅延バインディング (ホスト/ゲストのマッピング))



エラーの理由を見つけるには、**/var/log/rhsm/** ディレクトリーにある **rhsm.log** という名前の virt-who ログファイルを開きます。

## 第11章 QEMU ゲストエージェントおよび SPICE エージェントによる仮想化の強化

QEMU ゲストエージェントや SPICE エージェントなどの Red Hat Enterprise Linux のエージェントは、システム上で仮想化ツールをより最適に実行するのに役立ちます。本章では、これらのエージェントについて説明します。



### 注記

ホストとゲストのパフォーマンスを最適化とチューニングを強化するには、『[Red Hat Enterprise Linux 7 仮想化のチューニングと最適化ガイド](#)』を参照してください。

### 11.1. QEMU ゲストエージェント

QEMU ゲストエージェントは、ゲスト内で実行され、ホストマシンが `libvirt` を使用してゲストオペレーティングシステムに対してコマンドを実行できるようにします。これにより、ファイルシステムのフリーズおよびフリーズ解除などの機能を容易に実行できます。ゲストオペレーティングシステムはその後、これらのコマンドに非同期的に応答します。QEMU ゲストエージェントパッケージの `qemu-guest-agent` はデフォルトで Red Hat Enterprise Linux 7 にインストールされます。

このセクションでは、`libvirt` コマンドとゲストエージェントが利用できる各種オプションについて説明します。



### 重要

信頼されるゲストによって実行される場合にのみ、QEMU ゲストエージェントは安全であることに注意してください。信頼されていないゲストはゲストエージェントのプロトコルを悪意のある方法で無視したり、これを誤用したりする可能性があります。ホスト上のサービス拒否攻撃を回避するためのビルトイン保護プログラムは存在しますが、ホストは、操作が予想どおりに実行されるためにゲストの連携を必要とします。

QEMU ゲストエージェントは、ゲストの実行中に仮想 CPU (vCPU) を有効/無効にするために使用できるため、ホットプラグおよびホットアンプラグ機能を使用せずに vCPU の数を調整できます。詳細は、『[仮想 CPU 数の設定](#)』を参照してください。

#### 11.1.1. QEMU ゲストエージェントとホスト間の通信設定

ホストマシンは、ホストとゲストマシン間の VirtIO シリアル接続で QEMU ゲストエージェントと通信します。VirtIO シリアルチャンネルは、キャラクターデバイスドライバ (通常 Unix ソケット) 経由でホストに接続し、ゲストはこのシリアルチャンネルでリスンします。



### 注記

`qemu-guest-agent` は、ホストが VirtIO シリアルチャンネルをリスンしているかどうかを検知しません。ただし、現時点でこのチャンネルはホストとゲスト間のイベントについてリスンするために使用されているため、リスナーなしのチャンネルへの書き込みによってゲスト仮想マシンで問題が発生する可能性は極めて低くなります。さらに、`qemu-guest-agent` プロトコルには同期マーカが含まれ、これによりホスト物理マシンは、コマンドの実行時にゲスト仮想マシンを同期に戻すよう強制できます。`libvirt` はこれらのマーカをすでに使用しているため、ゲスト仮想マシンは保留状態の未達の応答を安全に破棄することができます。

### 11.1.1.1. Linux ゲスト上での QEMU ゲストエージェントの設定

QEMU ゲストエージェントは実行中またはシャットダウンした仮想マシンで設定できます。実行中のゲストに設定される場合、ゲストはゲストエージェントをすぐに使用し始めます。ゲストがシャットダウンしている場合は、QEMU ゲストエージェントが次の起動時に有効にされます。

ゲストと QEMU ゲストエージェント間で通信を設定するには、**virsh** または **virt-manager** を使用できます。以下に、QEMU ゲストエージェントを Linux ゲストで設定する方法を説明します。

#### 手順11.1 シャットダウンした Linux ゲストで virsh を使用したゲストエージェントとホスト間の通信設定

1. 仮想マシンをシャットダウンします。

仮想マシン (この例の名前は *rhel7*) が QEMU ゲストエージェントの設定前にシャットダウンしていることを確認します。

```
# virsh shutdown rhel7
```

2. QEMU ゲストエージェントチャンネルをゲスト XML 設定に追加します。

QEMU ゲストエージェントの詳細を追加できるようゲストの XML ファイルを編集します。

```
# virsh edit rhel7
```

以下をゲストの XML ファイルに追加し、変更を保存します。

```
<channel type='unix'>
  <target type='virtio' name='org.qemu.guest_agent.0' />
</channel>
```

3. 仮想マシンを起動します。

```
# virsh start rhel7
```

4. ゲストに QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントがゲスト仮想マシンにまだインストールされていない場合は、これをインストールします。

```
# yum install qemu-guest-agent
```

5. ゲスト内の QEMU ゲストエージェントを起動します。

ゲスト内の QEMU ゲストエージェントサービスを起動します。

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

別の方法として、以下のステップに従って QEMU ゲストエージェントを実行中のゲストで設定することもできます。

#### 手順11.2 実行中の Linux ゲストでのゲストエージェントとホスト間の通信設定

1. QEMU ゲストエージェントの XML ファイルを作成します。

```
# cat agent.xml
```

```
<channel type='unix'>
  <target type='virtio' name='org.qemu.guest_agent.0' />
</channel>
```

2. **QEMU ゲストエージェントを仮想マシンに割り当てます。**

以下のコマンドを実行して、QEMU ゲストエージェントを実行中の仮想マシン (この例の名前は *rhel7*) に割り当てます。

```
# virsh attach-device rhel7 agent.xml
```

3. **ゲストに QEMU ゲストエージェントをインストールします。**

QEMU ゲストエージェントがゲスト仮想マシンにまだインストールされていない場合は、これをインストールします。

```
# yum install qemu-guest-agent
```

4. **ゲスト内の QEMU ゲストエージェントを起動します。**

ゲスト内の QEMU ゲストエージェントサービスを起動します。

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

### 手順11.3 virt-manager を使用した QEMU ゲストエージェントとホスト間の通信設定

1. **仮想マシンをシャットダウンします。**

QEMU ゲストエージェントの設定前に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。

仮想マシンをシャットダウンするには、**仮想マシンマネージャー**の仮想マシンの一覧からこれを選択し、メニューバーにある光スイッチのアイコンをクリックします。

2. **QEMU ゲストエージェントチャンネルをゲストに追加します。**

ゲストのウィンドウの先頭にある電球アイコンをクリックして仮想マシンのハードウェアの詳細画面を開きます。

**ハードウェアを追加** ボタンをクリックして **新しい仮想ハードウェアを追加** ウィンドウを開き、**チャンネル** を選択します。

**名前** ドロップダウンリストから QEMU ゲストエージェントを選択し、**完了** をクリックします。

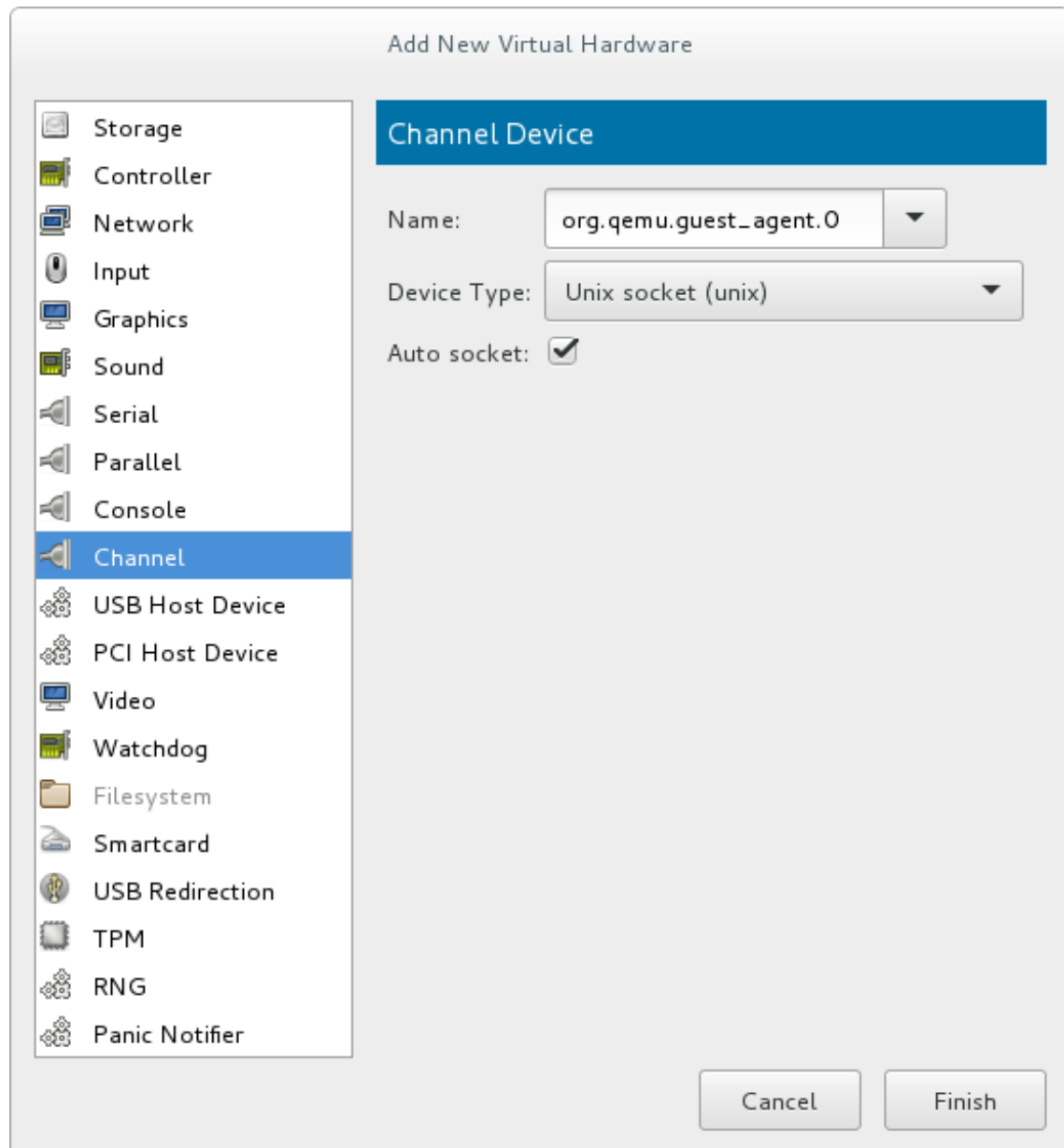



図11.1 QEMU ゲストエージェントチャンネルデバイスの選択

### 3. 仮想マシンを起動します。

仮想マシンを起動するには、**仮想マシンマネージャー**の仮想マシンの一覧から目的のマシンを

選択し、続いてメニューバーにある  をクリックします。

### 4. ゲストに QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントがまだゲスト仮想マシンにインストールされていない場合は、**virt-manager** でゲストを開き、QEMU ゲストエージェントをインストールします。

```
# yum install qemu-guest-agent
```

### 5. ゲスト内の QEMU ゲストエージェントを起動します。

ゲスト内の QEMU ゲストエージェントサービスを起動します。

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

QEMU ゲストエージェントが *rhel7* 仮想マシンに設定されました。

## 11.2. LIBVIRT による QEMU ゲストエージェントの使用

QEMU ゲストエージェントをインストールすると、さまざまな **libvirt** コマンドをより強力にすることができます。ゲストエージェントは以下の **virsh** コマンドを強化します。

- **virsh shutdown --mode=agent** - このシャットダウン方法は **virsh shutdown --mode=acpi** よりも信頼性があります。QEMU ゲストエージェントで使用される **virsh shutdown** は連携しているゲストをクリーンな状態でシャットダウンできるように保証されているためです。エージェントがない場合、**libvirt** は **ACPI** シャットダウンイベントの挿入に依存しなければなりませんが、一部のゲストはそのイベントを無視するため、シャットダウンされません。  
  
**virsh reboot** の場合と同じ構文で使用できます。
- **virsh snapshot-create --quiesce** - スナップショットが作成される前に、ゲストがその I/O を安定した状態にフラッシュできるようにします。これにより、**fsck** を実行したり、部分的なデータベーストランザクションを失わずにスナップショットを使用することが可能になります。ゲストエージェントは、ゲストの連携を可能にすることにより、ディスクコンテンツの高レベルの安定性を実現します。
- **virsh domfsfreeze** および **virsh domfsthaw** - 分離したゲストファイルシステムを休止します。
- **virsh domfstrim** - ゲストに対してそのファイルシステムをトリミングするように指示します。
- **virsh domtime** - ゲストのクロックを休止するか、または設定します。
- **virsh setvcpus --guest** - ゲストに対して、CPU をオフラインにするように指示します。
- **virsh domifaddr --source agent** - ゲストエージェント経由でゲストオペレーティングシステムの IP アドレスを照会します。
- **virsh domfsinfo** - 実行中のゲスト内にマウントされたファイルシステムの一覧を表示します。
- **virsh set-user-password** - ゲストにおけるユーザーアカウントのパスワードを設定します。

### 11.2.1. ゲストディスクバックアップの作成

**libvirt** は **qemu-guest-agent** と通信し、ゲスト仮想マシンファイルシステムのスナップショットが内部で一貫しており、随時使用可能であることを保証することができます。ゲストシステムの管理者はアプリケーション固有のフリーズ/フリーズ解除フックスクリプトを作成し、インストールすることができます。ファイルシステムをフリーズする前に、**qemu-guest-agent** は主なフックスクリプト (**qemu-guest-agent** パッケージ内に組み込まれている) を起動します。フリーズプロセスは、すべての仮想マシンアプリケーションを一時的に非アクティブにします。

スナップショットプロセスは以下のステップで構成されています。

- ファイルシステムのアプリケーション/データベースは作業バッファを仮想ディスクにフラッシュし、クライアント接続の受け入れを停止します。
- アプリケーションはそれらのデータファイルを一貫性のある状態にします。
- メインフックスクリプトが返されます。

- **qemu-guest-agent** はファイルシステムをフリーズし、管理スタックはスナップショットを取得します。
- スナップショットが確認されます。
- ファイルシステムの機能が再開します。

フリーズ解除が逆の順序で生じます。

ゲストのファイルシステムのスナップショットを作成するには、**virsh snapshot-create -- quiesce --disk-only** コマンドを実行(または、「[現在のゲスト仮想マシンのスナップショットの作成](#)」に詳しく説明されているように**virsh snapshot-create-as guest\_name --quiesce --disk-only**を実行)します。



## 注記

アプリケーション固有のフックスクリプトは、正常に実行されるために各種の SELinux パーミッションを必要とする場合があります。これは、データベースと対話するためにスクリプトをソケットに接続する必要がある場合と同様です。通常、このような状況に備えて、ローカルの SELinux ポリシーを作成し、インストールしておく必要があります。[表11.1「QEMU ゲストエージェントのパッケージコンテンツ」](#)内の **/etc/qemu-ga/fsfreeze-hook.d/** のラベルが付いた行にある **restorecon -Fvvr** コマンドを実行すると、ファイルシステムノードへのアクセスが設定なしで機能します。

**qemu-guest-agent** バイナリー RPM には以下のファイルが含まれます。

表11.1 QEMU ゲストエージェントのパッケージコンテンツ

ファイル名	説明
<b>/usr/lib/systemd/system/qemu-guest-agent.service</b>	QEMU ゲストエージェント用のサービス制御スクリプト (開始/停止)。
<b>/etc/sysconfig/qemu-ga</b>	<b>/usr/lib/systemd/system/qemu-guest-agent.service</b> 制御スクリプトで読み取られる QEMU ゲストエージェントの設定ファイル。設定内容はシェルスクリプトのコメントと共にファイルに記載されます。
<b>/usr/bin/qemu-ga</b>	QEMU ゲストエージェントのバイナリーファイル。
<b>/etc/qemu-ga</b>	フックスクリプトの root ディレクトリ。
<b>/etc/qemu-ga/fsfreeze-hook</b>	メインフックスクリプト。これに必要な変更はありません。

ファイル名	説明
<code>/etc/qemu-ga/fsfreeze-hook.d</code>	個別の、アプリケーション固有フックスクリプトのディレクトリー。ゲストのシステム管理者はこのディレクトリーにフックスクリプトを手動でコピーし、それらに適切なファイルモードビットが設定されていることを確認してから、このディレクトリー上で <b>restorecon -FvR</b> を実行する必要があります。
<code>/usr/share/qemu-kvm/qemu-ga/</code>	サンプルスクリプト (サンプルとしての使用のみを想定) を含むディレクトリー。ここに含まれるスクリプトは実行されません。

メインのフックスクリプトである `/etc/qemu-ga/fsfreeze-hook` は、独自のメッセージを、アプリケーション固有スクリプトの標準出力およびエラーメッセージと共に `/var/log/qemu-ga/fsfreeze-hook.log` のログファイルに記録します。詳細情報は、[libvirt アップストリームの Web サイト](#) を参照してください。

### 11.3. SPICE エージェント

SPICE エージェントは、ゲストオペレーティングシステムを SPICE クライアントに統合することで、**virt-manager** などのグラフィカルアプリケーションをよりスムーズに実行できるようにします。

たとえば、**virt-manager** でウィンドウのサイズを変更する場合、SPICE エージェントは、クライアントの解像度に対する X セッションの解像度の自動調整を可能にします。また SPICE エージェントはホストとゲスト間のコピーアンドペーストのサポートを提供し、マウス操作とカーソル操作のずれの発生を防ぎます。

SPICE エージェントの各種機能についてのシステム固有の情報については、**spice-vdagent** パッケージの README ファイルを参照してください。

#### 11.3.1. SPICE エージェントとホスト間の通信設定

SPICE エージェントは実行中またはシャットダウンした仮想マシンで設定できます。実行中のゲストに設定される場合、ゲストはゲストエージェントをすぐに使用し始めます。ゲストがシャットダウンしている場合は、SPICE エージェントが次の起動時に有効にされます。

ゲストと SPICE エージェント間で通信を設定するには、**virsh** または **virt-manager** を使用できます。以下に、SPICE エージェントを Linux ゲストで設定する方法を説明します。

#### 手順11.4 Linux ゲストでの **virsh** を使用したゲストエージェントとホスト間の通信設定

1. 仮想マシンをシャットダウンします。  
仮想マシン (この例の名前は *rhel7*) が SPICE エージェントの設定前にシャットダウンしていることを確認します。

```
# virsh shutdown rhel7
```

2. SPICE エージェントチャンネルをゲスト XML 設定に追加します。  
SPICE エージェントの詳細を追加できるようゲストの XML ファイルを編集します。



```
# virsh edit rhel7
```

以下をゲストの XML ファイルに追加し、変更を保存します。

```
<channel type='spicevmc'>
  <target type='virtio' name='com.redhat.spice.0' />
</channel>
```

### 3. 仮想マシンを起動します。

```
# virsh start rhel7
```

### 4. ゲストに SPICE エージェントをインストールします。

SPICE エージェントがゲスト仮想マシンにまだインストールされていない場合は、これをインストールします。

```
# yum install spice-vdagent
```

### 5. ゲスト内の SPICE エージェントを起動します。

ゲスト内の SPICE エージェントサービスを起動します。

```
# systemctl start spice-vdagent
```

別の方法として、以下のステップに従って SPICE エージェントを実行中のゲストで設定することもできます。

## 手順11.5 実行中の Linux ゲストでの SPICE エージェントとホスト間の通信設定

### 1. SPICE エージェントの XML ファイルを作成します。

```
# cat agent.xml
<channel type='spicevmc'>
  <target type='virtio' name='com.redhat.spice.0' />
</channel>
```

### 2. SPICE エージェントを仮想マシンに割り当てます。

以下のコマンドを実行して、SPICE エージェントを実行中の仮想マシン (この例の名前は *rhel7*) に割り当てます。

```
# virsh attach-device rhel7 agent.xml
```

### 3. ゲストに SPICE エージェントをインストールします。

SPICE エージェントがゲスト仮想マシンにまだインストールされていない場合は、これをインストールします。

```
# yum install spice-vdagent
```

### 4. ゲスト内の SPICE エージェントを起動します。

ゲスト内の SPICE エージェントサービスを起動します。

```
# systemctl start spice-vdagent
```

## 手順11.6 virt-manager を使用した SPICE エージェントとホスト間の通信設定

## 1. 仮想マシンをシャットダウンします。

SPICE エージェントの設定前に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。

仮想マシンをシャットダウンするには、**仮想マシンマネージャー**の仮想マシンの一覧からこれを選択し、メニューバーにある光スイッチのアイコンをクリックします。

## 2. SPICE エージェントチャンネルをゲストに追加します。

ゲストのウィンドウの先頭にある電球アイコンをクリックして仮想マシンのハードウェアの詳細画面を開きます。

**ハードウェアを追加** ボタンをクリックして **新しい仮想ハードウェアを追加** ウィンドウを開き、**チャンネル** を選択します。

**名前** ドロップダウンリストから **SPICE エージェント** を選択し、チャンネルアドレスを編集して **完了** をクリックします。

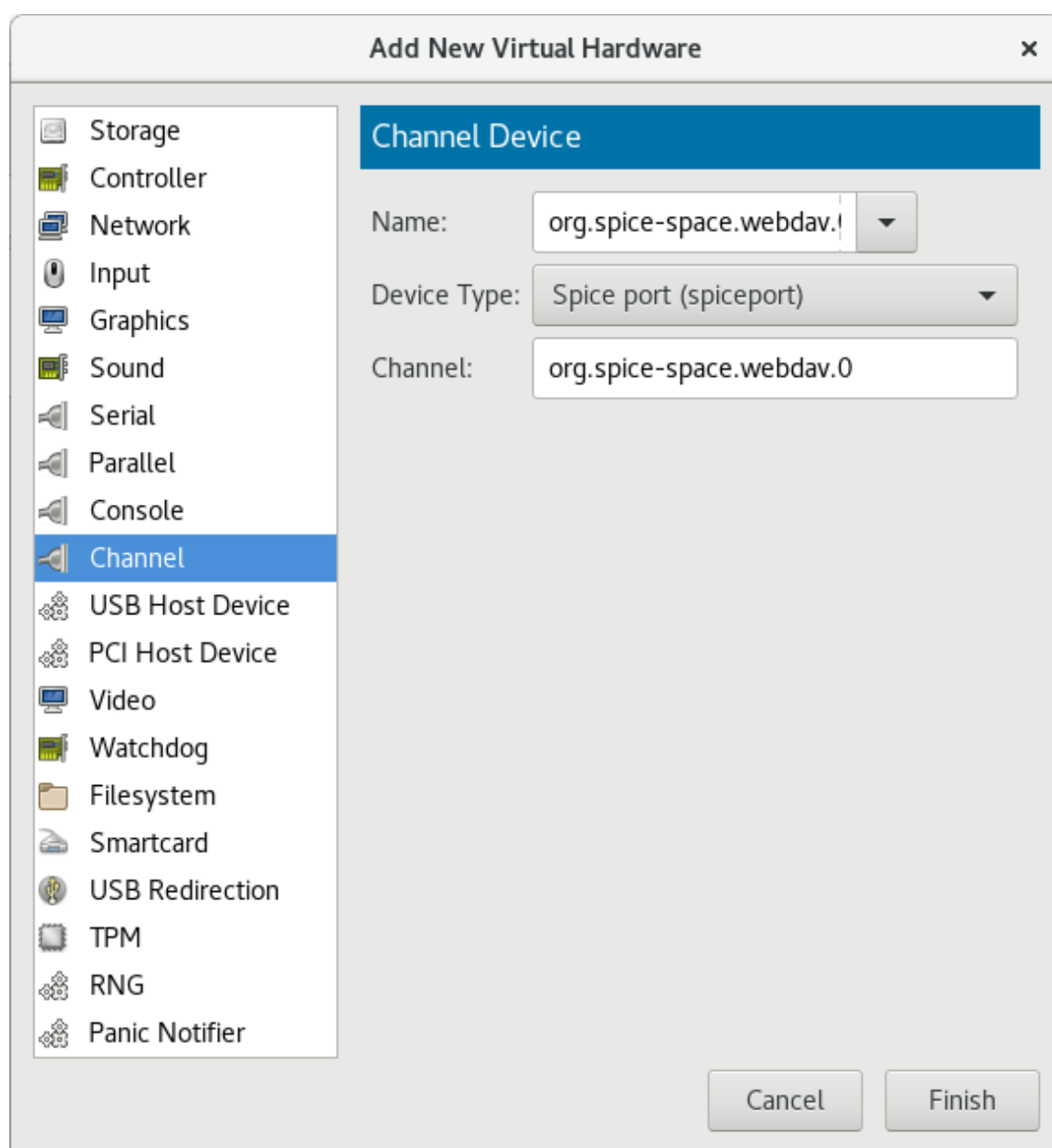



図11.2 SPICE エージェントチャンネルデバイスの選択

## 3. 仮想マシンを起動します。

仮想マシンを起動するには、**仮想マシンマネージャー**の仮想マシンの一覧から目的のマシンを

選択し、続いてメニューバーにある  をクリックします。

4. ゲストに **SPICE エージェント**をインストールします。

SPICE エージェントがまだゲスト仮想マシンにインストールされていない場合は、**virt-manager** でゲストを開き、SPICE エージェントをインストールします。

```
# yum install spice-vdagent
```

5. ゲスト内の **SPICE エージェント**を起動します。

ゲスト内の SPICE エージェントサービスを起動します。

```
# systemctl start spice-vdagent
```

SPICE エージェントが *rhel7* 仮想マシンに設定されました。

## 第12章 仮想化のネスト

### 12.1. 概要

Red Hat Enterprise Linux 7.3 では、*nested virtualization* は KVM ゲスト仮想マシンのテクノロジープレビューとして利用できます。この機能を使うと、物理ホスト (レベル 0 または **L0** とも言う) で実行されるゲスト仮想マシン (レベル 1 または **L1**) はハイパーバイザーとして機能し、独自の (**L2**) ゲスト仮想マシンを作成できます。

ネストされた仮想化は、制限のある環境でハイパーバイザーをデバッグしたり、数に制限のある物理リソース上の規模の大きい仮想デプロイメントをテストしたりするのに役立ちます。ただし、ネストされた仮想化はユーザーの本番稼動環境ではサポートされず、推奨もされておらず、これは主に開発およびテスト目的で使用されることが意図されていることに注意してください。

ネストされた仮想化は、Intel および AMD プロセッサでのみ利用できるテクノロジープレビューです。それ以外のプロセッサでは機能しない可能性があり、この使用は推奨されていません。

ネストされた仮想化はホストの仮想化拡張機能に依存した状態で機能します。これを **QEMU Tiny Code Generator (TCG)** エミュレーションを使用して仮想環境でゲストを実行することと混同しないようにしてください。後者は、Red Hat Enterprise Linux ではサポートされません。

### 12.2. 設定

以下のステップに従って、ネストされた仮想化を有効にし、これを設定して使用を開始してください。

1. **有効化:** この機能はデフォルトで無効にされています。有効にするには、**L0** ホスト物理マシンで以下の手順を実行してください。

#### Intel の場合:

1. ネストされた仮想化がホストシステムで利用できるかどうかを確認します。

```
$ cat /sys/module/kvm_intel/parameters/nested
```

このコマンドが **Y** または **1** を返す場合、この機能は有効にされています。

コマンドが **0** または **N** を返す場合、ステップ **b** および **c** を使用します。

2. **kvm\_intel** モジュールをアンロードします。

```
# modprobe -r kvm_intel
```

3. ネスト機能をアクティブにします。

```
# modprobe kvm_intel nested=1
```

4. ネスト機能は **L0** ホストの次回起動時まで有効になります。これを永続的に有効化するには、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
```

#### AMD の場合:

1. ネストされた仮想化がシステムで利用できるかどうかを確認します。

```
$ cat /sys/module/kvm_amd/parameters/nested
```

このコマンドが **Y** または **1** を返す場合、この機能は有効にされています。

コマンドが **0** または **N** を返す場合、ステップ **b** および **c** を使用します。

2. **kvm\_amd** モジュールをアンロードします。

```
# modprobe -r kvm_amd
```

3. ネスト機能をアクティブにします。

```
# modprobe kvm_amd nested=1
```

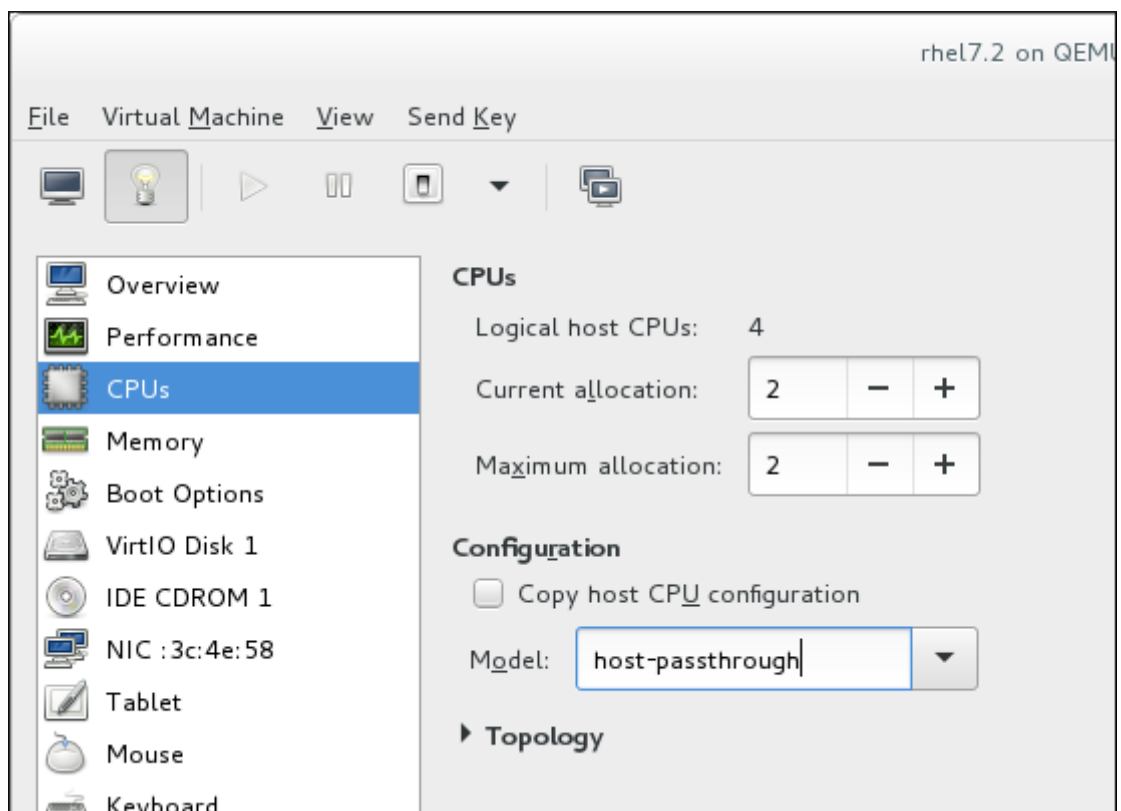
4. ネスト機能は **L0** ホストの次回起動時まで有効になります。これを永続的に有効化するには、以下の行を **/etc/modprobe.d/kvm.conf** ファイルに追加します。

```
options kvm_amd nested=1
```

2. 以下の方法のいずれかを使用して、仮想化のネスト化に向けて **L1** 仮想マシンを設定します。

#### virt-manager

1. 必要なゲストの GUI を開き、**仮想マシンの情報** を表示アイコンをクリックします。
2. **CPU 数** メニューを選択し、**設定** セクションの **モデル** フィールドに **host-passthrough** と入力してから (ドロップダウンの選択項目を使用しないでください)、**適用** をクリックします。



## ドメイン XML

以下の行をゲストのドメイン XML ファイルに追加します。

```
<cpu mode='host-passthrough' />
```

ゲストの XML 設定ファイルに **<cpu>** 要素がすでに含まれる場合は、これを書き直してください。

3. ネストされた仮想化の**使用を開始**するには、L1 ゲスト内に L2 ゲストをインストールします。これを実行するには、L1 ゲストのインストール時と同じ手順に従います。詳細は、「[3章 仮想マシンの作成](#)」を参照してください。

## 12.3. 制約および制限事項

- L0 ホストおよび L1 ゲストでは Red Hat Enterprise Linux 7.2 以降を実行することを強く推奨します。L2 ゲストには Red Hat でサポートされる任意のゲストシステムを含めることができます。
- L1 または L2 ゲストの移行はサポートされません。
- ハイパーバイザーとしての L2 ゲストの使用と L3 ゲストの作成はサポートされません。
- ホストで利用できるすべての機能が L1 ハイパーバイザーで利用できる訳ではありません。たとえば、IOMMU/VT-d または APICv は L1 ハイパーバイザーで使用することはできません。
- ネストされた仮想化を使用するには、ホスト CPU に必要な機能フラグがなければなりません。L0 および L1 ハイパーバイザーが正常に設定されている場合、L0 および L1 の両方で **cat /proc/cpuinfo** コマンドを使用し、以下のフラグが両方のハイパーバイザーのそれぞれの CPU について一覧表示されていることを確認します。
  - Intel の場合: **vmx** (Hardware Virtualization; ハードウェア仮想化) および **ept** (Extended Page Tables; 拡張ページテーブル)
  - AMD の場合: **svm** (vmx と同等) および **npt** (ept と同等)

## パート II. 管理

## 第13章 ストレージプール

本章では、さまざまなタイプのストレージプールを作成する方法について説明します。ストレージプールとは、管理者(ストレージ担当管理者の場合が多い)によって確保される一定量のストレージで、ゲスト仮想マシンなどに使用されます。ストレージプールは、ストレージ管理者またはシステム管理者のいずれかによってストレージボリュームに分割され、ボリュームはブロックデバイスとしてゲスト仮想マシンに割り当てられます。

たとえば、NFS サーバー担当のストレージ管理者が、ゲスト仮想マシンのデータすべてを格納するための共有ディスクを作成するとします。システム管理者は共有ディスクの詳細情報を使用して、仮想化ホストにストレージプールを定義します。この例では、管理者は `nfs.example.com:/path/to/share` を `/vm_data` にマウントします。ストレージプールが起動すると、`libvirt` は指定ディレクトリーに共有をマウントします。これは、システム管理者がログインし、`mount nfs.example.com:/path/to/share /vmdata` を実行したかのような動作になります。ストレージプールが自動起動するように設定されている場合、`libvirt` は `libvirt` の起動時に指定されたディレクトリーに NFS 共有ディスクをマウントします。

ストレージプールが起動すると、NFS 共有ディスク内のファイルはストレージボリュームとして報告され、ストレージボリュームのパスは `libvirt` API を使って問い合わせることができます。次に、ストレージボリュームのパスを、ゲスト仮想マシンの XML 定義内のゲスト仮想マシンのブロックデバイス用のソースストレージについて記述しているセクションにコピーできます。NFS の場合、`libvirt` API を使用するアプリケーションは、プールの制限サイズ(共有のストレージ最大容量)までストレージプール内にストレージボリューム(NFS 共有内のファイル)を作成したり、そのボリュームを削除したりすることができます。ボリュームの作成および削除は、ストレージプールのすべてのタイプで対応している訳ではありません。ストレージプールを停止(`pool-destroy`)すると起動の操作が無効になるような場合は、NFS 共有をアンマウントしてください。破棄の操作では共有にあるデータは変更されませんが、その名前は変更されます。詳細は、`man virsh` を参照してください。

2 番目の例として、iSCSI ストレージプールについて見てみましょう。ストレージ管理者は iSCSI ターゲットをプロビジョニングして一連の LUN を VM を実行するホストに提示します。その iSCSI ターゲットをストレージプールとして管理できるように `libvirt` を設定する場合、`libvirt` は、ホストが iSCSI ターゲットにログインし、`libvirt` が利用可能な LUN をストレージボリュームとして報告できるようにします。NFS の例と同様に、ストレージボリュームのパスは問い合わせが可能で、VM の XML 定義で使用されます。この場合、LUN は iSCSI サーバー上で定義され、`libvirt` はボリュームを作成したり、削除したりすることができません。

ゲスト仮想マシンの正常な操作のためにストレージプールとボリュームが必要な訳ではありません。ストレージプールとボリュームは、`libvirt` によりストレージの特定部分をゲスト仮想マシンで使用できるようにする方法を提供します。ストレージプールを使用しないシステムでは、システム管理者は、各自が選択するツールを使ってゲスト仮想マシンのストレージの可用性を確保する必要があります。たとえば、NFS 共有をホスト物理マシンの `fstab` に追加して、その共有が起動時にマウントされるようにします。

`libvirt` を使用してストレージプールおよびボリュームを管理する利点の1つに、`libvirt` のリモートプロトコルがあります。これにより、ゲスト仮想マシンによって必要とされるリソースの設定と共に、ゲスト仮想マシンのライフサイクルのあらゆる側面を管理することができます。これらの操作は完全に `libvirt` API 内で、リモートホストで実行できます。つまり、`libvirt` を使用した管理アプリケーションにより、ユーザーがホスト物理マシンでゲスト仮想マシンを設定するために必要なすべてのタスクを実行できるようになります。これらのタスクには、シェルアクセスやその他のコントロールチャネルなしで実行できるリソースの割り当て、ゲスト仮想マシンの実行、ゲスト仮想マシンのシャットダウン、およびリソースの割り当て解除が含まれます。

ストレージプールは仮想コンテナであるものの、`qemu-kvm` によって許可される最大サイズとホストマシン上のディスクサイズの2つの要素によって制限されます。ストレージプールはホストマシンのディスクサイズを超えることはできません。最大サイズは以下のようになります。



- **virtio-blk** =  $2^{63}$  バイトまたは 8 エクサバイト (raw ファイルまたはディスクを使用)
- **Ext4** = ~16 TB (4 KB ブロックサイズを使用)
- **XFS** = ~8 エクサバイト
- **qcow2** とホストファイルシステムはそれぞれ独自のメタデータを維持し、非常に大きなイメージサイズを使用する場合はスケーラビリティの評価または調整を行う必要があります。**raw** ディスクを使用すると、スケーラビリティや最大サイズに影響を与える可能性のある層の数が少なくなります。

**libvirt** では、ディレクトリーベースのストレージプール **/var/lib/libvirt/images/** ディレクトリーをデフォルトのストレージプールとして使用します。このデフォルトストレージプールは別のストレージプールに変更することができます。

- **ローカルのストレージプール**- ローカルのストレージプールは直接ホスト物理マシンサーバーに割り当てられます。ローカルのストレージプールには、ローカルのディレクトリー、直接割り当てられているディスク、物理パーティション、および LVM ボリュームグループなどが含まれます。これらのストレージボリュームはゲスト仮想マシンのイメージを格納するか、または追加ストレージとしてゲスト仮想マシンに割り当てられます。ローカルのストレージプールは直接ホスト物理マシンサーバーに割り当てられるため、ゲスト仮想マシンの移行や大量のゲスト仮想マシンを必要としない小規模な導入、テスト、および開発などに便利です。ローカルのストレージプールはライブマイグレーションには対応していないため、多くの実稼働環境には適していません。
- **ネットワーク接続の (共有) ストレージプール**- ネットワーク接続のストレージプールには、標準プロトコルを使っ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。ネットワーク接続のストレージは、ホスト物理マシン間での仮想マシンの移行に **virt-manager** を使用する場合は必須になりますが、**virsh** を使用する場合はオプションになります。ネットワーク接続のストレージプールは **libvirt** で管理します。ネットワーク接続のストレージプールに対応するプロトコルには、以下が含まれます。
  - ファイバーチャネルベースの LUN
  - iSCSI
  - NFS
  - GFS2
  - SCSI RDMA プロトコル (SCSI RCP) - InfiniBand アダプターと 10GbE iWARP アダプターで使用されるブロックエクスポートプロトコル



### 注記

マルチパスのストレージプールは、完全にサポートされていないため、作成したり、使用したりすることはできません。

### 例13.1 NFS ストレージプール

NFS サーバー担当のストレージ管理者が、ゲスト仮想マシンのデータを格納するための共有を作成するとしましょう。システム管理者は、ホスト物理マシン上のプールをこの共有の詳細を使って定義します (**nfs.example.com:/path/to/share** は **/vm\_data** にマウントされる)。プールが起動すると、**libvirt** は指定ディレクトリーに共有をマウントします。これは、システム管理者がログイン

し、`mount nfs.example.com:/path/to/share /vmdata` を実行したかのような動作になります。プールが自動起動するように設定されている場合、`libvirt` は、`libvirt` の起動時に指定されるディレクトリー上に NFS 共有をマウントします。

プールが起動すると、NFS で共有しているファイルがボリュームとして報告され、ストレージボリュームのパスの問い合わせが `libvirt API` を使って実行されます。このボリュームのパスは、ゲスト仮想マシンの XML 定義ファイル内のゲスト仮想マシンのブロックデバイス用のソースストレージについて記述しているセクションにコピーされます。NFS では、`libvirt API` を使用するアプリケーションで、プールの制限サイズ (共有のストレージ最大容量) までプール内にボリューム (NFS 共有内のファイル) を作成したり、そのボリュームを削除したりすることができます。ボリュームの作成および削除はすべてのプールタイプで対応している訳ではありません。プールを停止すると起動の操作が無効になるような場合は、NFS 共有をアンマウントしてください。破棄の操作により共有上にあるデータは変更されませんが、その名前は変更されます。詳細は `man virsh` をご覧ください。

## 注記

ゲスト仮想マシンが正常に機能するためにストレージプールとボリュームが必要な訳ではありません。プールとボリュームは、`libvirt` がストレージの一部をゲスト仮想マシンに使用できるようにする 1 つの手段です。ただし、独自にストレージを管理し、プールやボリュームを定義せずにゲスト仮想マシンを適切に動作させる方法を取る管理者もいます。プールを使用しないシステムでは、システム管理者は、各自が選択するツールを使ってゲスト仮想マシンのストレージの可用性を確保する必要があります。たとえば、NFS 共有をホスト物理マシンの `fstab` に追加して、その共有が起動時にマウントされるようにします。



## 警告

ゲストでストレージプールを作成する際には、『[Red Hat Enterprise Linux 7 仮想化セキュリティガイド](#)』に書かれた関連するセキュリティ上の考慮事項に従うようにしてください。

## 13.1. ディスクベースのストレージプール

このセクションでは、ゲスト仮想マシン用にディスクベースのストレージデバイスを作成する方法について説明します。



### 警告

ゲストには、ブロックデバイスまたはディスク全体 (例: `/dev/sdb`) への書き込みアクセスは付与しないようにし、パーティション (例: `/dev/sdb1`) や LVM ボリュームなどを使用するようにします。

ゲストにブロックデバイス全体を渡した場合、ゲストはそれにパーティションを設定するか、またはその上に独自の LVM グループを作成する可能性があります。これは、ホスト物理マシンがこれらのパーティションや LVM グループを検出し、エラーを出す原因になります。

## 13.1.1. `virsh` を使用したディスクベースのストレージプールの作成

以下の手順では、`virsh` コマンドを使用して、ディスクデバイスを使って新しいストレージプールを作成します。



### 警告

ディスクをストレージプール専用にすると、再フォーマットが行われ、ディスクデバイス上に現在格納されているすべてのデータが消去されます。以下の手順を開始する前に、ストレージデバイスのデータをバックアップすることを強く推奨します。

#### 1. ディスクに **GPT ディスクラベル**を作成します。

ディスクのラベルは、**GPT (GUID Partition Table)** ディスクラベルを使って付け直す必要があります。GPT ディスクラベルを使用すると、各デバイス上に最大 128 個もの数多くのパーティションを作成できます。MS-DOS パーティションテーブルに比べ、GPT パーティションテーブルの方がはるかに多くのパーティションのパーティションデータを格納することができます。

```
# parted /dev/sdb
GNU Parted 2.1
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel
New disk label type? gpt
(parted) quit
Information: You may need to update /etc/fstab.
#
```

#### 2. ストレージプールの設定ファイルを作成します。

新規デバイスに必要となるストレージプール情報が含まれた一時 XML テキストファイルを作成します。

このファイルは以下のような形式にし、以下のフィールドを含む必要があります。

**<name>guest\_images\_disk</name>**

**name** パラメーターはストレージプールの名前を指定します。この例では、**guest\_images\_disk** という名前を使用しています。

**<device path='/dev/sdb'/>**

**path** 属性を持つ **device** パラメーターはストレージデバイスのデバイスパスを指定します。この例では、デバイス **/dev/sdb** を使用しています。

**<target> <path>/dev</path></target>**

**path** サブパラメーターを持つファイルシステム **target** パラメーターは、ホスト物理マシンのファイルシステム上の位置を指定して、このストレージプールで作成されたボリュームを割り当てます。

例: **sdb1**、**sdb2**、**sdb3**。この例のように **/dev/** を使用すると、このストレージプールから作成したボリュームは **/dev/sdb1**、**/dev/sdb2**、**/dev/sdb3** となり、これらにアクセスできるようになります。

**<format type='gpt'/>**

**format** パラメーターは、パーティションテーブルのタイプを指定します。この例では、前述のステップで作成した GPT ディスクラベルのタイプと一致するよう **gpt** を使用しています。

テキストエディターを使って、ストレージプールデバイス用の XML ファイルを作成します。

### 例13.2 ディスクベースストレージデバイスのストレージプール

```
<pool type='disk'>
  <name>guest_images_disk</name>
  <source>
    <device path='/dev/sdb' />
    <format type='gpt' />
  </source>
  <target>
    <path>/dev</path>
  </target>
</pool>
```

### 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起動します。**virsh pool-list --all** コマンドを使用して、プールが起動していることを確認できます。

```
# virsh pool-start iscsirhel7guest
Pool iscsirhel7guest started
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
guest_images_disk                   active     no
```

### 4. デバイスを割り当てます。

前述のステップで作成した XML 設定ファイルと共に **virsh pool-define** コマンドを使用し、ストレージプールの定義を追加します。

```
# virsh pool-define ~/guest_images_disk.xml
Pool guest_images_disk defined from /root/guest_images_disk.xml
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
guest_images_disk                  inactive   no
```

#### 5. autostart をオンにします。

ストレージプールの **autostart** をオンにします。**Autostart** は、**libvirtd** サービスの起動時にストレージプールを起動するようにこのサービスを設定します。

```
# virsh pool-autostart guest_images_disk
Pool guest_images_disk marked as autostarted
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
guest_images_disk                  active     yes
```

#### 6. ストレージプールの設定を確認します。

ストレージプールが正しく作成されたこと、およびサイズが正確に報告されていること、さらに状態が **running** として報告されていることを確認します。

```
# virsh pool-info guest_images_disk
Name:          guest_images_disk
UUID:          551a67c8-5f2a-012c-3844-df29b167431c
State:         running
Capacity:      465.76 GB
Allocation:    0.00
Available:     465.76 GB
# ls -la /dev/sdb
brw-rw----. 1 root disk 8, 16 May 30 14:08 /dev/sdb
# virsh vol-list guest_images_disk
Name          Path
-----
```

#### 7. オプション: 一時設定ファイルを削除します。

不要な場合は、一時的なストレージプール XML 設定ファイルを削除します。

```
# rm ~/guest_images_disk.xml
```

これでディスクベースのストレージプールが利用できるようになります。

### 13.1.2. virsh を使用したストレージプールの削除

以下の手順は、**virsh** を使用してストレージプールを削除する方法を説明しています。

1. 同じプールを使用する他のゲスト仮想マシンに関連した問題を避けるには、ストレージプールを停止し、そのストレージプールで使用中のリソースすべてを解放するのが最良の方法です。

```
# virsh pool-destroy guest_images_disk
```

2. ストレージプールの定義を削除します。

```
# virsh pool-undefine guest_images_disk
```

## 13.2. パーティションベースのストレージプール

このセクションでは、フォーマット済みのブロックデバイス、つまりパーティションをストレージプールとして使用する方法を説明します。

以下の例では、ホスト物理マシンには **500GB** のままの単独パーティション (**/dev/sdc1**) である **500GB** のハードドライブ (**/dev/sdc**) があります。以下の手順を使用して、このハードドライブ用にストレージプールをセットアップします。

### 13.2.1. virt-manager を使用したパーティションベースのストレージプールの作成

以下の手順では、ストレージデバイスのパーティションを使用し、新規のストレージプールを作成します。

#### 手順13.1 virt-manager を使用したパーティションベースのストレージプールの作成

1. ファイルシステムを **ext4** に設定します。

コマンドウィンドウから以下のコマンドを入力し、ファイルシステムを **ext4** に設定します。

```
# mkfs.ext4 /dev/sdc1
```

2. ストレージプールの設定を開きます。

- a. **virt-manager** グラフィカルインターフェースで、メインウィンドウからホスト物理マシンを選択します。

**編集** メニューを開いて、**接続の詳細** を選択します。

- b. **接続の詳細** ウィンドウで **ストレージ** タブをクリックします。

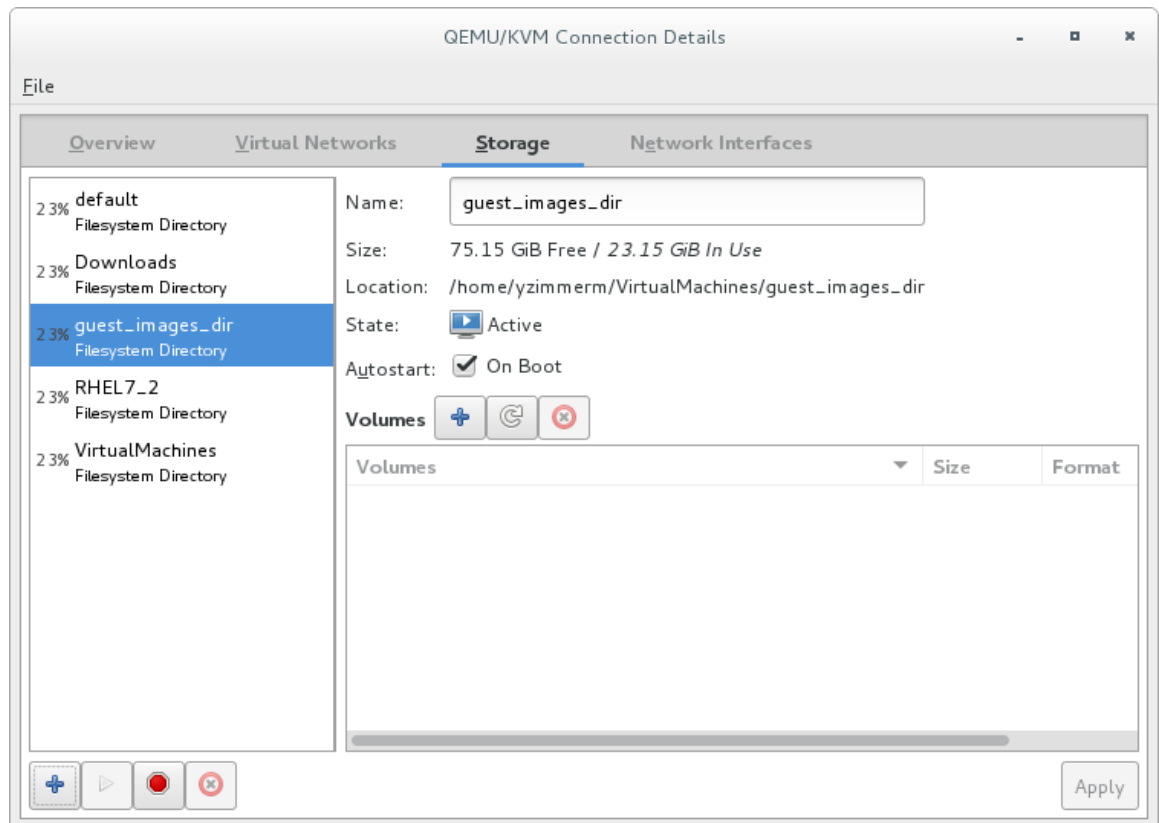


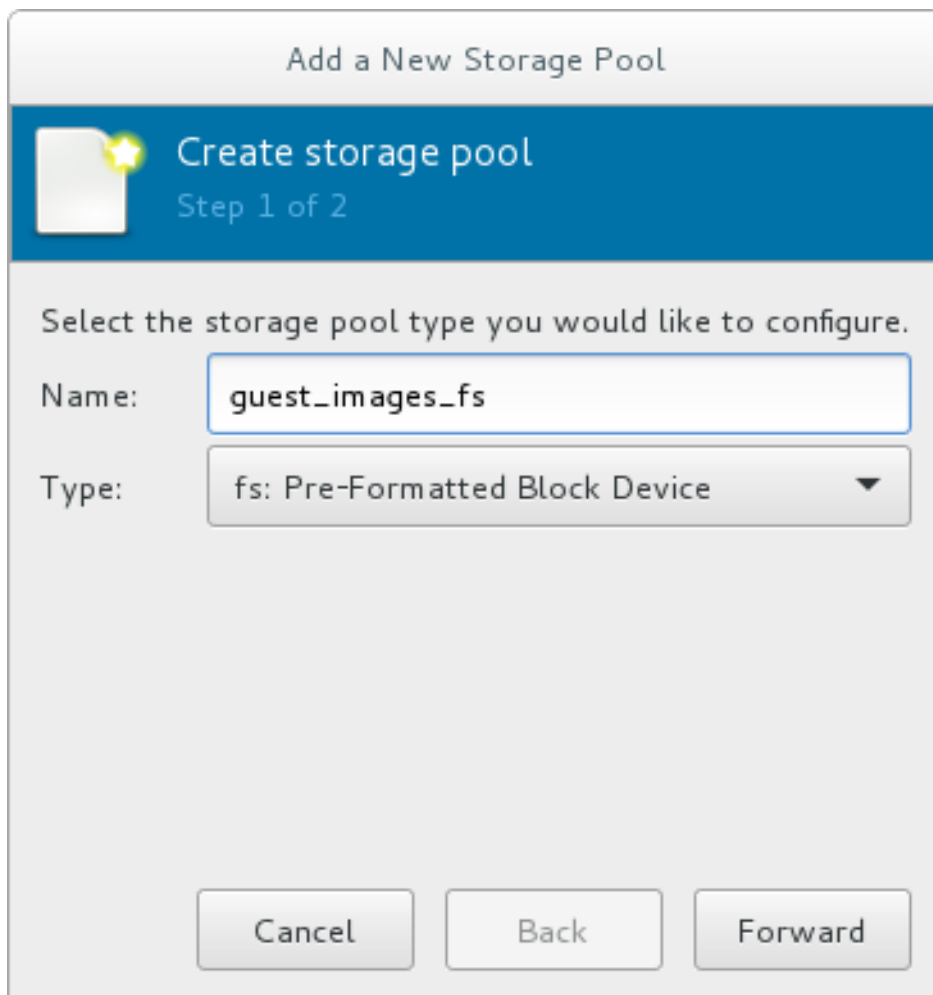
図13.1 「ストレージ」タブ

### 3. 新規ストレージプールを作成します。

#### a. 新規プールの追加 (パート 1)

+ ボタン (ウィンドウの下部にある) を押します。新規ストレージプールを追加 ウィザードが表示されます。

ストレージプールの名前を選択します。この例では、*guest\_images\_fs* という名前を使用しています。種類を **fs: 事前フォーマット済みブロックデバイス** に変更します。



Add a New Storage Pool

Create storage pool  
Step 1 of 2

Select the storage pool type you would like to configure.

Name:

Type:

Cancel Back Forward

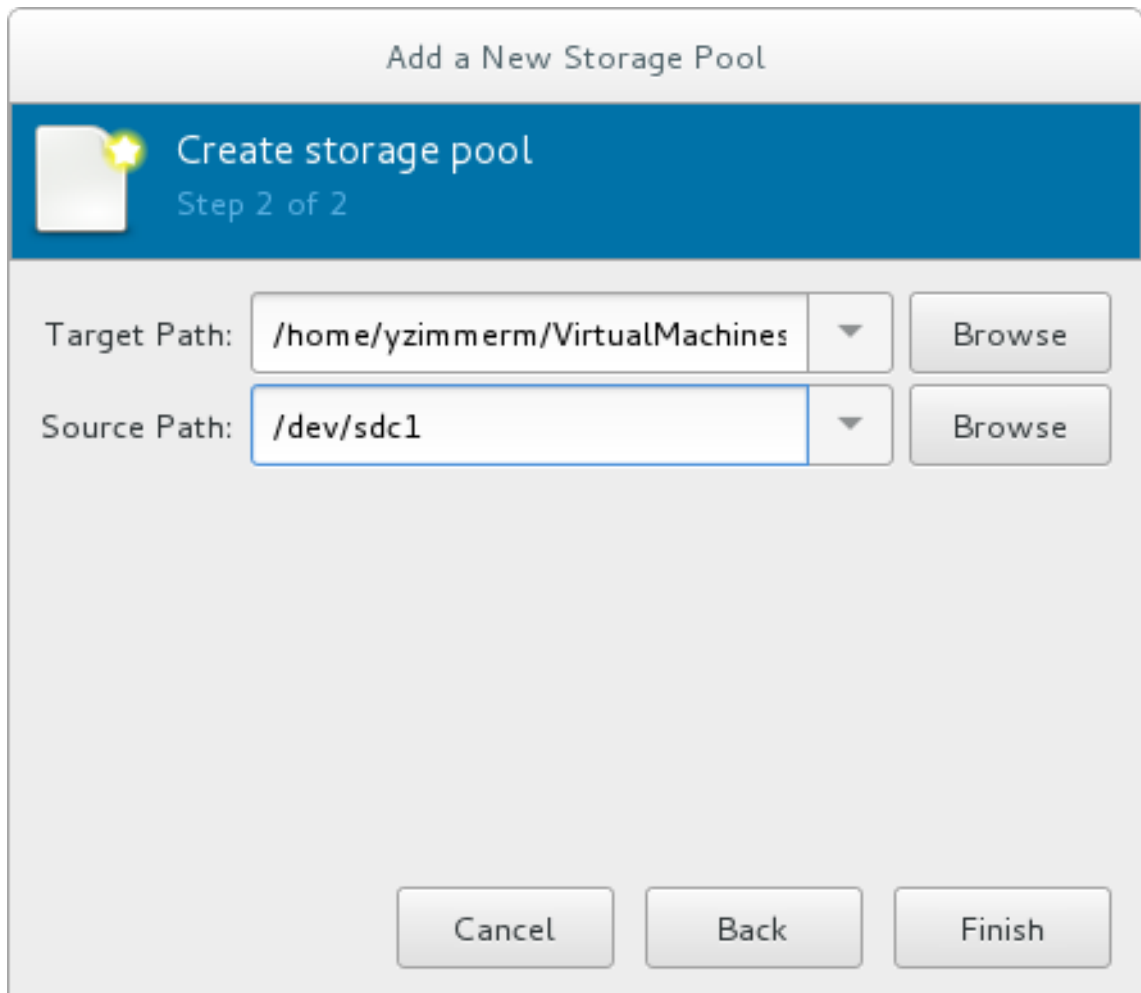
図13.2 ストレージプールの名前と種類

進む ボタンをクリックして次に進みます。


b. 新規プールの追加 (パート 2)

ターゲットパス および ソースパス フィールドを変更します。





Add a New Storage Pool



Create storage pool

Step 2 of 2

Target Path:

/home/yzimmerm/VirtualMachines

Browse

Source Path:

/dev/sdc1

Browse

Cancel

Back

Finish

図13.3 ストレージプールのパス

#### ターゲットパス

ストレージプールのソースデバイスのマウント先となる場所を **ターゲットパス** フィールドに入力します。その場所がまだ存在していない場合は、**virt-manager** がそのディレクトリーを作成します。

#### ソースパス

ソースパス フィールドにデバイスを入力します。

この例では、`/dev/sdc1` デバイスを使用しています。

詳細を確認して **完了** ボタンを押すと、ストレージプールが作成されます。

#### 4. 新規ストレージプールを確認します。

数秒後に、左側のストレージ一覧に新規ストレージプールが表示されます。サイズが予想通りの値で報告されていることを確認します (この例では、**2.88 GiB** 空き)。状態 フィールドで新規ストレージが **動作中** と報告されていることを確認します。

ストレージプールを選択します。自動起動 フィールドで、**起動時** のチェックボックスをクリックします。これで **libvirtd** サービスの起動時にはストレージデバイスも常に起動されるようになります。

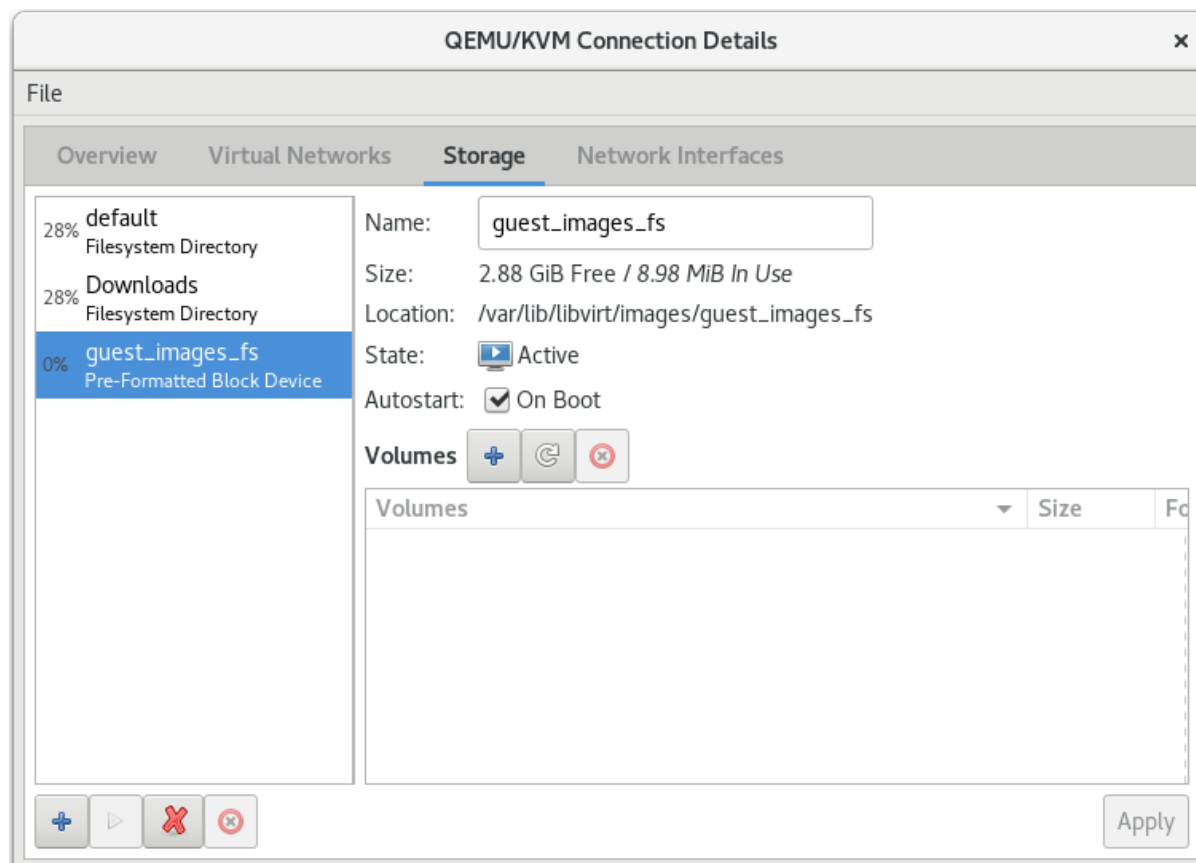



図13.4 ストレージ一覧の確認

これでストレージプールが作成されました。**接続の詳細** ウィンドウを閉じます。

### 13.2.2. virt-manager を使用したストレージプールの削除

以下の手順では、ストレージプールを削除する方法を説明します。

1. 同じプールを使用する他のゲスト仮想マシンに関連した問題を避けるには、ストレージプールを停止し、そのストレージプールで使用中のリソースをすべて解放するのが最良の方法です。これを実行するには、停止するストレージプールを選択して、ストレージウィンドウの下部に

ある  をクリックします。

2. ストレージプールを削除するには、 をクリックします。このアイコンが使用できるのは、ストレージプールが停止している場合のみです。

### 13.2.3. virsh を使用したパーティションベースのストレージプールの作成

このセクションでは、**virsh** コマンドを使用したパーティションベースストレージプールの作成方法を説明します。



### 警告

ディスク全体をストレージプールとして割り当てる場合は、この手順を使用しないでください (例: `/dev/sdb`)。ブロックデバイスやディスク全体への書き込みアクセスをゲストに付与しないようにしてください。この方法はパーティション (例: `/dev/sdb1`) をストレージプールに割り当てる場合にのみ使用するようにしてください。

## 手順13.2 `virsh` を使用したフォーマット済みのブロックデバイスのストレージプールの作成

### 1. ストレージプールの定義を作成します。

`virsh pool-define-as` コマンドを使用して 新規ストレージプールの定義を作成します。フォーマット済みディスクをストレージプールとして定義するために指定する必要のあるオプションには、以下の 3 つがあります。

#### パーティション名

**name** パラメーターはストレージプールの名前を指定します。この例では、`guest_images_fs` という名前を使用しています。

#### デバイス

**path** 属性を持つ **device** パラメーターはストレージデバイスのデバイスパスを指定します。この例では、パーティション `/dev/sdc1` を使用しています。

#### マウントポイント

フォーマット済みのデバイスをマウントするローカルファイルシステム上の **mountpoint** です。マウントポイントのディレクトリが存在しない場合は、**virsh** コマンドがそのディレクトリを作成します。

この例では、`/guest_images` ディレクトリを使用しています。

```
# virsh pool-define-as guest_images_fs fs - - /dev/sdc1 -
"/guest_images"
Pool guest_images_fs defined
```

これで、新規のプールが作成されました。

### 2. 新規プールを確認します。

現在のストレージプールを一覧表示します。

```
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
guest_images_fs                    inactive   no
```

### 3. マウントポイントを作成します。

**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マウントポイントを作成します。

```
# 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 ..
# 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、ファイルシステムをマウントポイントにマウントし、プールを使用できるようにします。

```
# virsh pool-start guest_images_fs
Pool guest_images_fs started
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
guest_images_fs                    active     no
```

#### 5. autostart をオンにします。

デフォルトでは、**virsh** で定義したストレージプールは、**libvirtd** の起動時に自動的に起動するようには設定されません。**virsh pool-autostart** コマンドを使用して自動起動をオンにします。これで、**libvirtd** が起動するたびにストレージプールも自動的に起動するようになります。

```
# virsh pool-autostart guest_images_fss
Pool guest_images_fs marked as autostarted

# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
guest_images_fs                    active     yes
```

#### 6. ストレージプールを確認します。

ストレージプールが正しく作成されたこと、およびサイズが予測される値で報告されていること、さらに状態が **running** として報告されていることを確認します。ファイルシステム上のマウントポイントに **"lost+found"** ディレクトリーがあることを確認します。このディレクトリーがあればデバイスはマウントされていることになります。

```
# 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
# 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
```

### 13.2.4. virsh を使用したストレージプールの削除

1. 同じプールを使用する他のゲスト仮想マシンに関連した問題を避けるには、ストレージプールを停止し、そのストレージプールで使用中のリソースすべてを解放するのが最良の方法です。

```
# virsh pool-destroy guest_images_disk
```

2. また、ストレージプールがあるディレクトリーを削除したい場合は、次のコマンドを使用します。

```
# virsh pool-delete guest_images_disk
```

3. ストレージプールの定義を削除します。

```
# virsh pool-undefine guest_images_disk
```

## 13.3. ディレクトリーベースのストレージプール

このセクションでは、ホスト物理マシンのディレクトリーにゲスト仮想マシンを格納する方法について説明します。

ディレクトリーベースのストレージプールは、**virt-manager** または **virsh** のコマンドラインツールを使用して作成することができます。

### 13.3.1. virt-manager 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

1. ローカルのディレクトリーを作成します。
  - a. オプション: ストレージプール用に新規ディレクトリーを作成します。  
ホストマシン上にストレージプール用のディレクトリーを作成します。この例では `/guest_images` というディレクトリー名を使用しています。

```
# mkdir /guest_images
```

- b. ディレクトリーの所有権を設定します。  
ディレクトリーのユーザーとグループの所有権を変更します。ディレクトリーは **root** ユーザーによって所有される必要があります。

```
# chown root:root /guest_images
```

- c. ディレクトリーの権限を設定します。

ディレクトリーのファイル権限を変更します。

```
# chmod 700 /guest_images
```

d. 変更を確認します。

権限が変更されていることを確認します。出力には、正しく設定された空のディレクトリーが表示されます。

```
# ls -la /guest_images
total 8
drwx-----. 2 root root 4096 May 28 13:57 .
dr-xr-xr-x. 26 root root 4096 May 28 13:57 ..
```

## 2. SELinuxのファイルコンテキストを設定します。

新しいディレクトリーに適切な SELinux コンテキストを設定します。プール名とディレクトリー名を一致させる必要はありません。ただし、ゲスト仮想マシンをシャットダウンする際に libvirt はコンテキストをデフォルト値に戻さなければなりません。このデフォルト値はディレクトリーのコンテキストによって決定されます。したがって、ディレクトリーに「virt\_image\_t」のラベルを明示的に付けておくと、ゲスト仮想マシンがシャットダウンした際にそのイメージに「virt\_image\_t」のラベルが付けられるため、ホスト物理マシンで実行されている他のプロセスと区別できるようになります。

```
# semanage fcontext -a -t virt_image_t '/guest_images(/.*)?'
# restorecon -R /guest_images
```

## 3. ストレージプールの設定を開きます。

- a. **virt-manager** グラフィカルインターフェースで、メインウィンドウからホスト物理マシンを選択します。

**編集** メニューを開き **接続の詳細** を選択します。

- b. **接続の詳細** ウィンドウで **ストレージ** タブをクリックします。

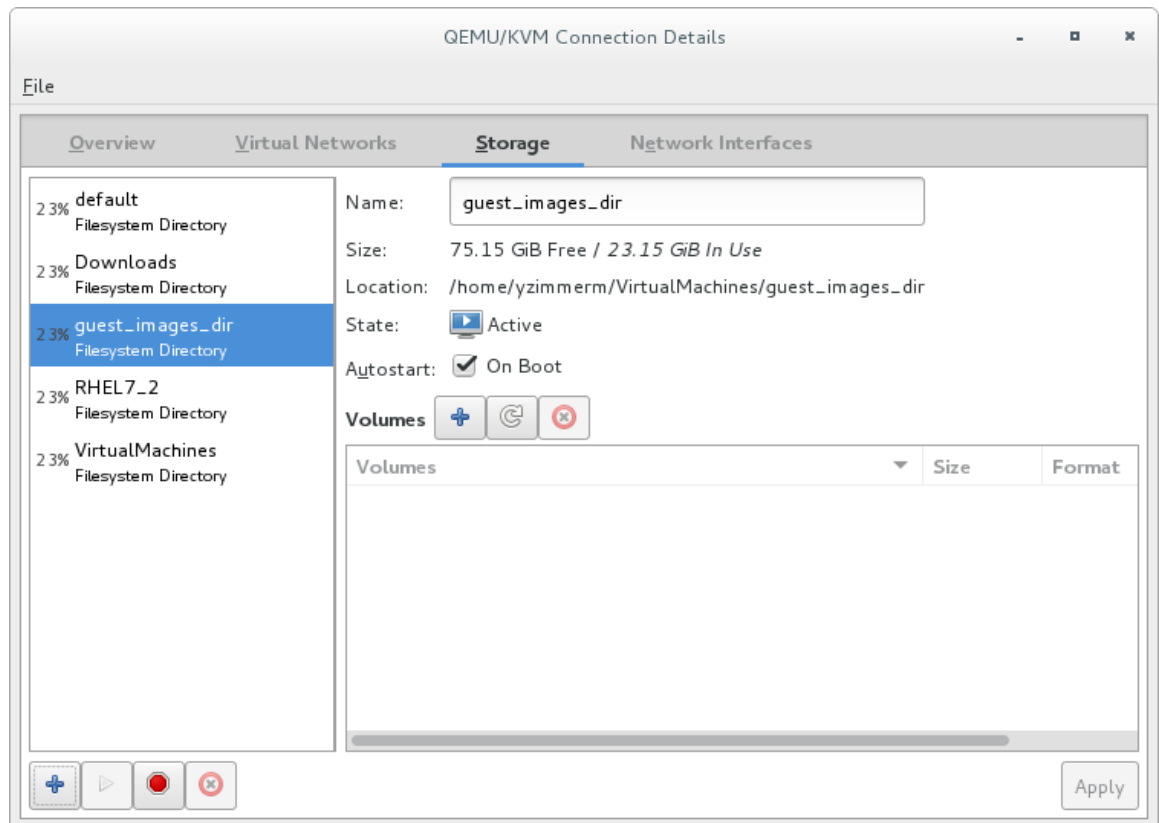


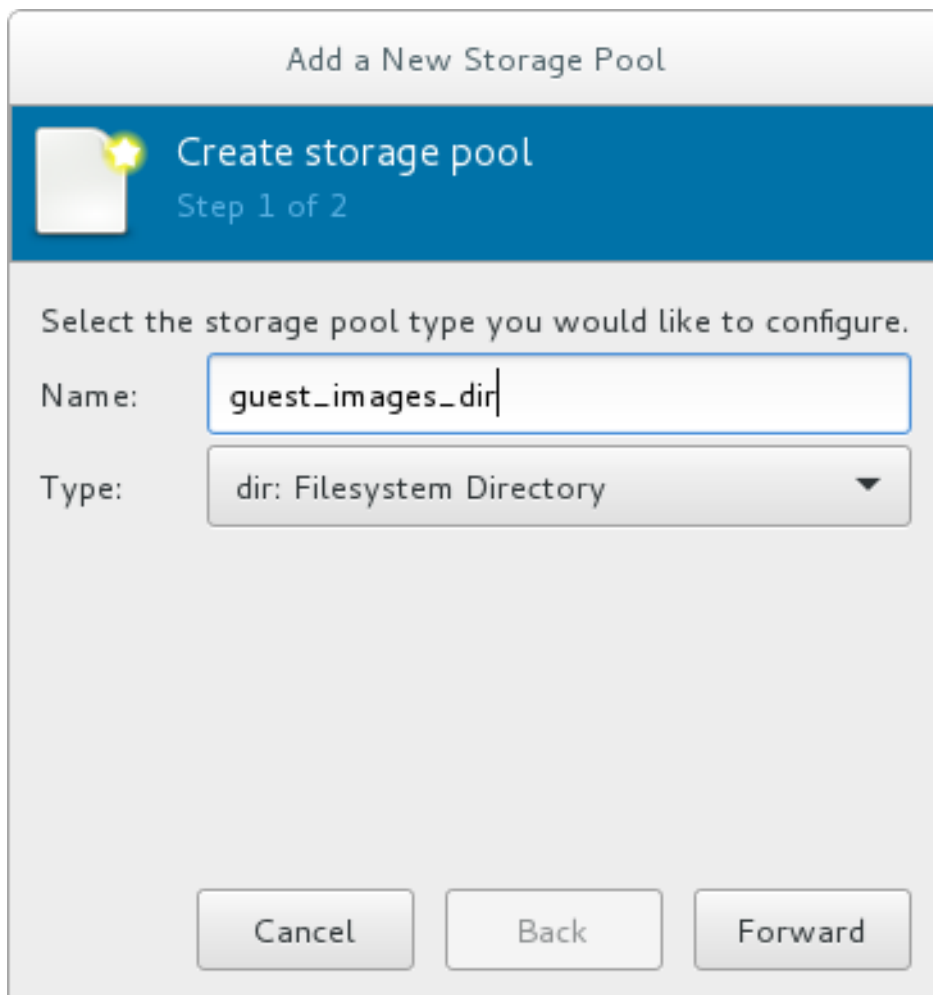
図13.5 「ストレージ」タブ

#### 4. 新しいストレージプールを作成します。

##### a. 新規プールの追加 (パート 1)

+ ボタンを押します (プールの追加ボタン)。新規ストレージプールを追加 ウィザードが表示されます。

ストレージプールの名前を選択します。この例では *guest\_images* という名前を使用しています。種類を **dir:** ファイルシステムディレクトリー にします。



Add a New Storage Pool

Create storage pool  
Step 1 of 2

Select the storage pool type you would like to configure.

Name:

Type:

Cancel Back Forward

図13.6 ストレージプールに名前を付ける

進む ボタンを押して先に進みます。

b. 新規プールの追加 (パート 2)

ターゲットパス フィールドを `/guest_images` などに変更します。



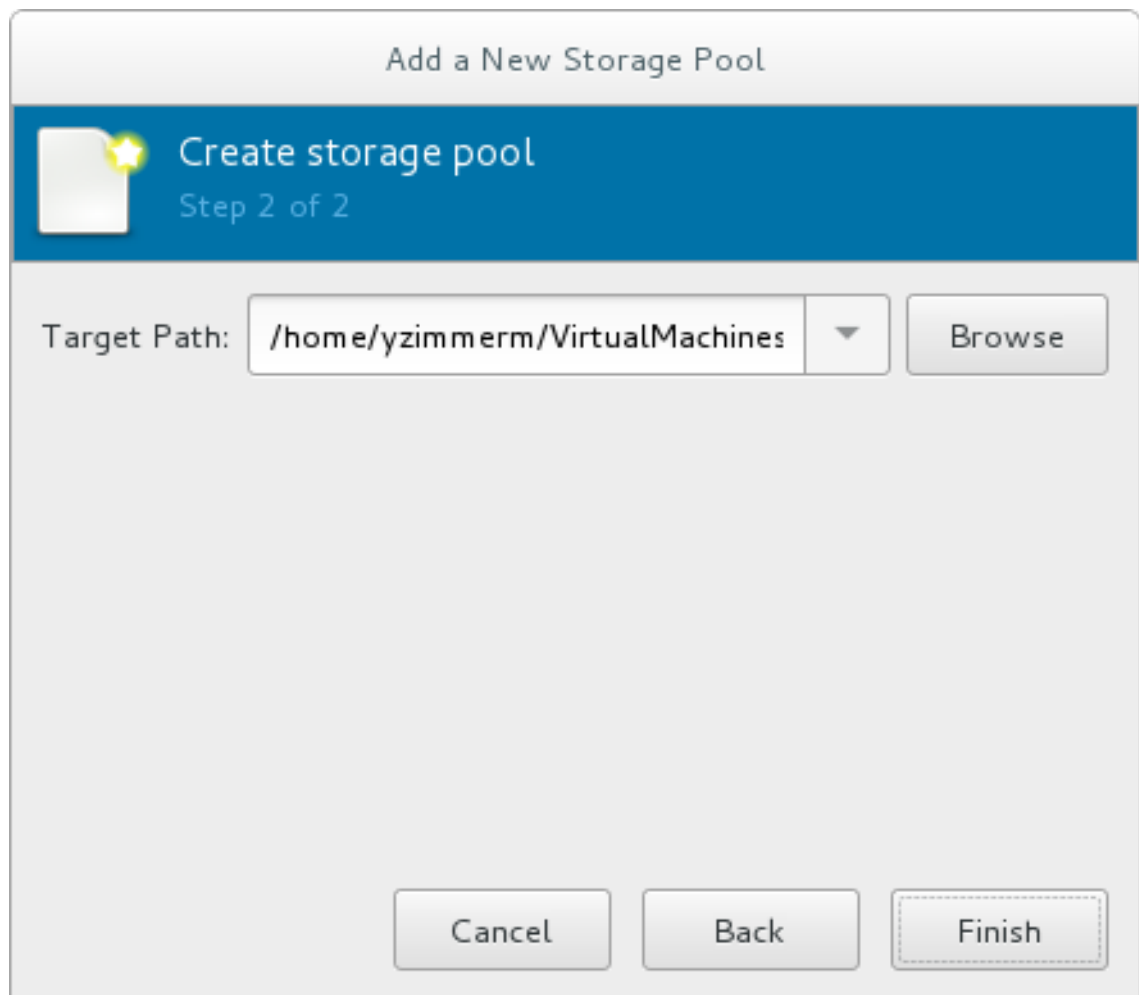


図13.7 ストレージプールのパスの選択

詳細を確認したら、**完了** ボタンを押してストレージプールを作成します。

5. 新しいストレージプールを確認します。

数秒後、左側のストレージ一覧に新しいストレージプールが表示されます。サイズが予想通りの値で報告されていることを確認します (この例では、**75.15 GiB** 空き)。状態 フィールドに新しいストレージが **動作中** と表示されていることを確認します。

ストレージプールを選択します。**自動起動** フィールドで **起動時** のチェックボックスにチェックが付いていることを確認します。チェックが付いていると、**libvirtd** サービスの起動時は常にストレージプールも起動します。

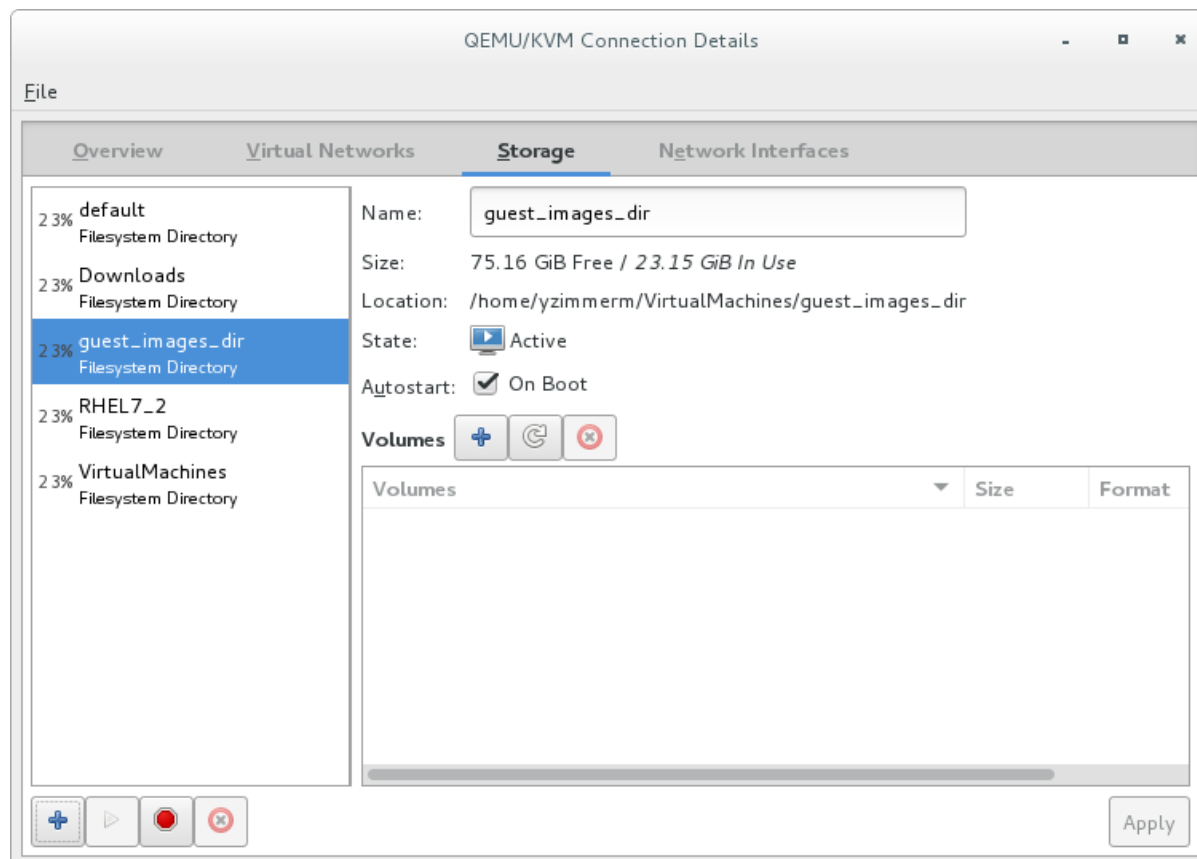



図13.8 ストレージプール情報の確認

これでストレージプールが作成されました。接続の詳細 ウィンドウを閉じます。

### 13.3.2. virt-manager を使用したストレージプールの削除

この手順では、ストレージプールを削除する方法を説明します。

1. 同じプールを使用する他のゲスト仮想マシンに関連した問題を避けるには、ストレージプールを停止し、そのストレージプールで使用中のリソースをすべて解放するのが最良の方法です。これを実行するには、停止するストレージプールを選択して、ストレージウィンドウの下部に

ある  をクリックします。

2. ストレージプールを削除するには、 をクリックします。このアイコンが使用できるのは、ストレージプールが停止している場合のみです。

### 13.3.3. virsh 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

1. ストレージプールの定義を作成します。  
新規のストレージプールを定義するために **virsh pool-define-as** コマンドを使用します。ディレクトリベースのストレージプールの作成には 2 つのオプションが必要になります。

- ストレージプールの名前。

この例では **guest\_images** という名前を使用しています。この例の **virsh** コマンドではすべてこの名前を使用しています。

- ゲストイメージファイル格納用のファイルシステムディレクトリーへのパス (このディレクトリーが存在しない場合は、**virsh** がディレクトリーを作成します)。

この例では `/guest_images` ディレクトリーを使用しています。

```
# virsh pool-define-as guest_images dir - - - - "/guest_images"
Pool guest_images defined
```

## 2. ストレージプールが一覧表示されていることを確認します。

ストレージプールオブジェクトが正しく作成されていること、および状態が **inactive** として表示されていることを確認します。

```
# virsh pool-list --all
Name                      State      Autostart
-----
default                   active     yes
guest_images              inactive   no
```

## 3. ローカルのディレクトリーを作成します。

以下のように **virsh pool-build** コマンドを使用し、*guest\_images* (例) ディレクトリーにディレクトリーベースのストレージプールをビルドします。

```
# virsh pool-build guest_images
Pool guest_images built
# ls -la /guest_images
total 8
drwx-----.  2 root root 4096 May 30 02:44 .
dr-xr-xr-x. 26 root root 4096 May 30 02:44 ..
# virsh pool-list --all
Name                      State      Autostart
-----
default                   active     yes
guest_images              inactive   no
```

## 4. ストレージプールを起動します。

**virsh** コマンドの **pool-start** を使ってディレクトリーストレージプールを有効にします。これにより、プールのボリュームがゲストのディスクイメージとして使用されます。

```
# virsh pool-start guest_images
Pool guest_images started
# virsh pool-list --all
Name                      State      Autostart
-----
default                   active     yes
guest_images              active     no
```

## 5. autostart をオンにします。

ストレージプールの **autostart** をオンにします。**Autostart** は、**libvirtd** サービスの起動時にはストレージプールも起動されるように設定します。

```
# virsh pool-autostart guest_images
Pool guest_images marked as autostarted
# virsh pool-list --all
```

Name	State	Autostart
-----	-----	-----
default	active	yes
guest_images	active	yes

#### 6. ストレージプールを設定を確認します。

ストレージプールが正しく作成されたこと、およびサイズが正確に報告されていること、さらに状態が **running** として報告されていることを確認します。ゲスト仮想マシンが実行されていない場合でもプールへのアクセスを可能にしておきたい場合には、**Persistent** に **yes** が表示されていることを確認します。サービスの起動時にプールを自動的に起動させる場合は、**Autostart** に **yes** が表示されていることを確認します。

```
# virsh pool-info guest_images
Name:          guest_images
UUID:          779081bf-7a82-107b-2874-a19a9c51d24c
State:         running
Persistent:    yes
Autostart:     yes
Capacity:      49.22 GB
Allocation:    12.80 GB
Available:     36.41 GB

# ls -la /guest_images
total 8
drwx-----. 2 root root 4096 May 30 02:44 .
dr-xr-xr-x. 26 root root 4096 May 30 02:44 ..
#
```

これでディレクトリーベースのストレージプールが利用できるようになります。

### 13.3.4. virsh を使用したストレージプールの削除

以下の手順では、**virsh** を使用してストレージプールを削除する方法を説明します。

1. 同じプールを使用する他のゲスト仮想マシンに関連した問題を避けるには、ストレージプールを停止し、そのストレージプールで使用中のリソースすべてを解放するのが最良の方法です。

```
# virsh pool-destroy guest_images_disk
```

2. また、ストレージプールがあるディレクトリーを削除したい場合は、次のコマンドを使用します。

```
# virsh pool-delete guest_images_disk
```

3. ストレージプールの定義を削除します。

```
# virsh pool-undefine guest_images_disk
```

## 13.4. LVM ベースのストレージプール

このセクションでは、LVM ボリュームグループをストレージプールとして使う方法について説明します。LVM ベースのストレージグループでは、LVM の柔軟性を十分に利用することができます。LVM の詳細については、『[Red Hat Enterprise Linux 7 論理ボリュームマネージャーの管理](#)』を参照してくだ

さい。



## 注記

以下の点に注意してください。

- 現在、シンプロビジョニングは LVM ベースのストレージプールでは利用できません。
- ホストがゲストによって使われる LVM の内容を不必要にスキャンするのを防ぐには、`/etc/lvm/lvm.conf` で **global\_filter** オプションを設定する必要があります。詳細については、『[Red Hat Enterprise Linux 7 論理ボリュームマネージャーの管理](#)』を参照してください。



## 警告

LVM ベースのストレージプールは全面的なディスクパーティションを必要とします。この手順を使用して新しいパーティションおよびデバイスをアクティブ化する場合、パーティションはフォーマットされ、すべてのデータは消去されます。ホストの既存ボリュームグループ (VG) を使用する場合は、いずれのデータも消去されません。以下の手順を開始する前に、ストレージデバイスのバックアップを取っておくことが推奨されます。

### 13.4.1. virt-manager 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には、既存の LVM ボリュームグループを使用するか、または新規の LVM ボリュームグループを空のパーティションに作成することができます。

LVM ボリュームグループ作成の詳細については、『[Red Hat Enterprise Linux 7 論理ボリュームマネージャーの管理](#)』を参照してください。

#### 1. ストレージプールの設定を開きます。

- a. **virt-manager** グラフィカルインターフェースで、メインウィンドウからホストを選択します。

**編集** メニューを開き **接続の詳細** を選択します。

- b. **ストレージ** タブをクリックします。

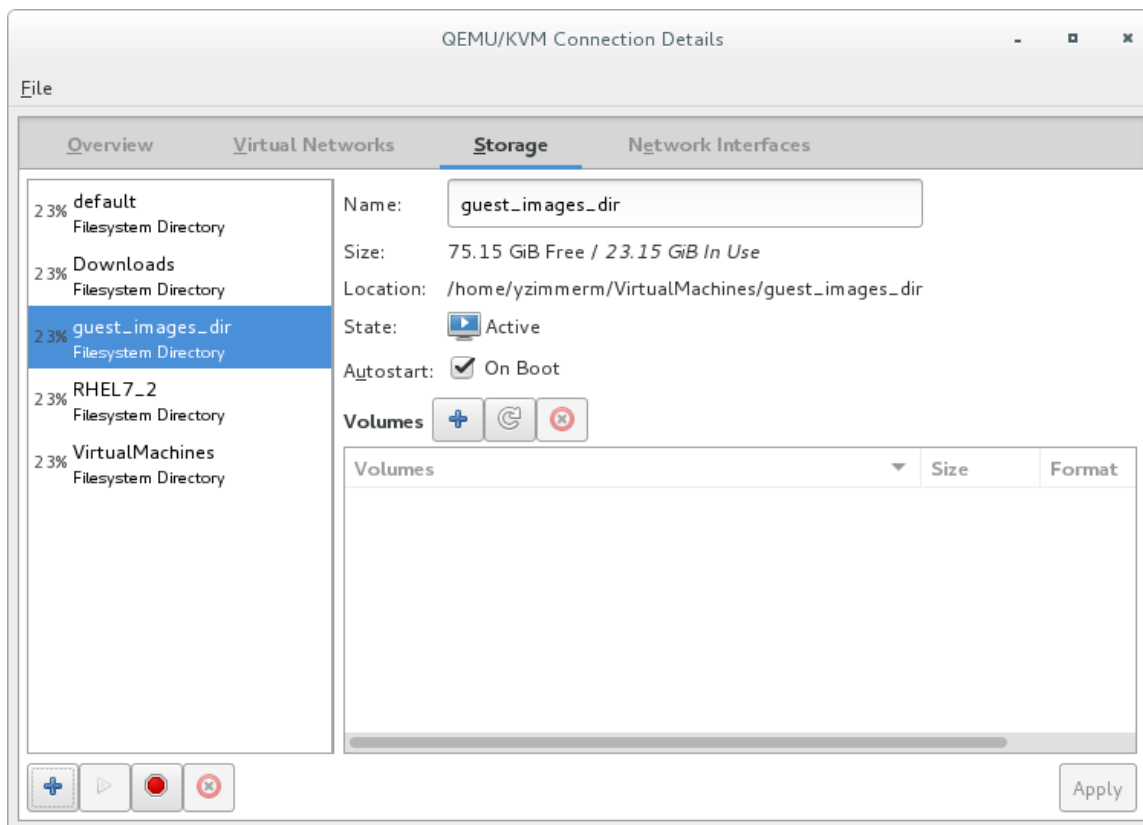


図13.9 「ストレージ」タブ

## 2. 新規ストレージプールを作成します。

### a. ウィザードを開始します。

+ ボタン (プールの追加ボタン) を押します。新規ストレージプールを追加 ウィザードが表示されます。

ストレージプールの名前を選択します。この例では、*guest\_images\_lvm* を使用しています。種類を **logical: LVM** ボリュームグループに変更します。

図13.10 LVM ストレージプールの追加

**進む** を押して次に進みます。

#### b. 新規プールの追加 (パート 2)

ターゲットパス フィールドと ソースパス フィールドに入力してから、**プールを構築** チェックボックスにチェックマークを付けます。

- **ターゲットパス** フィールドを使用して、既存の LVM ボリュームグループか、または新規ボリュームグループの名前の**いずれか**を選択します。デフォルト形式は、`storage_pool_name/lvm_Volume_Group_name`になります。

この例では、`/dev/guest_images_lvm` という名前の新しいボリュームグループを使用しています。

- 既存の LVM ボリュームグループが**ターゲットパス** で使用されている場合は、**ソースパス** フィールドはオプションになります。

新規の LVM ボリュームグループの場合には、ストレージデバイスの場所を **ソースパス** フィールドに入力します。この例では、空のパーティション `/dev/sdc` を使用しています。

- **プールを構築** チェックボックスにチェックマークを付けて、**virt-manager** に新規の LVM ボリュームグループを作成するよう指示します。既存のボリュームグループを使用する場合は、**プールを構築** チェックボックスは選択しないでください。

この例では、空のパーティションを使用して新規のボリュームグループを作成しているため、**プールを構築** チェックボックスを選択しておく必要があります。

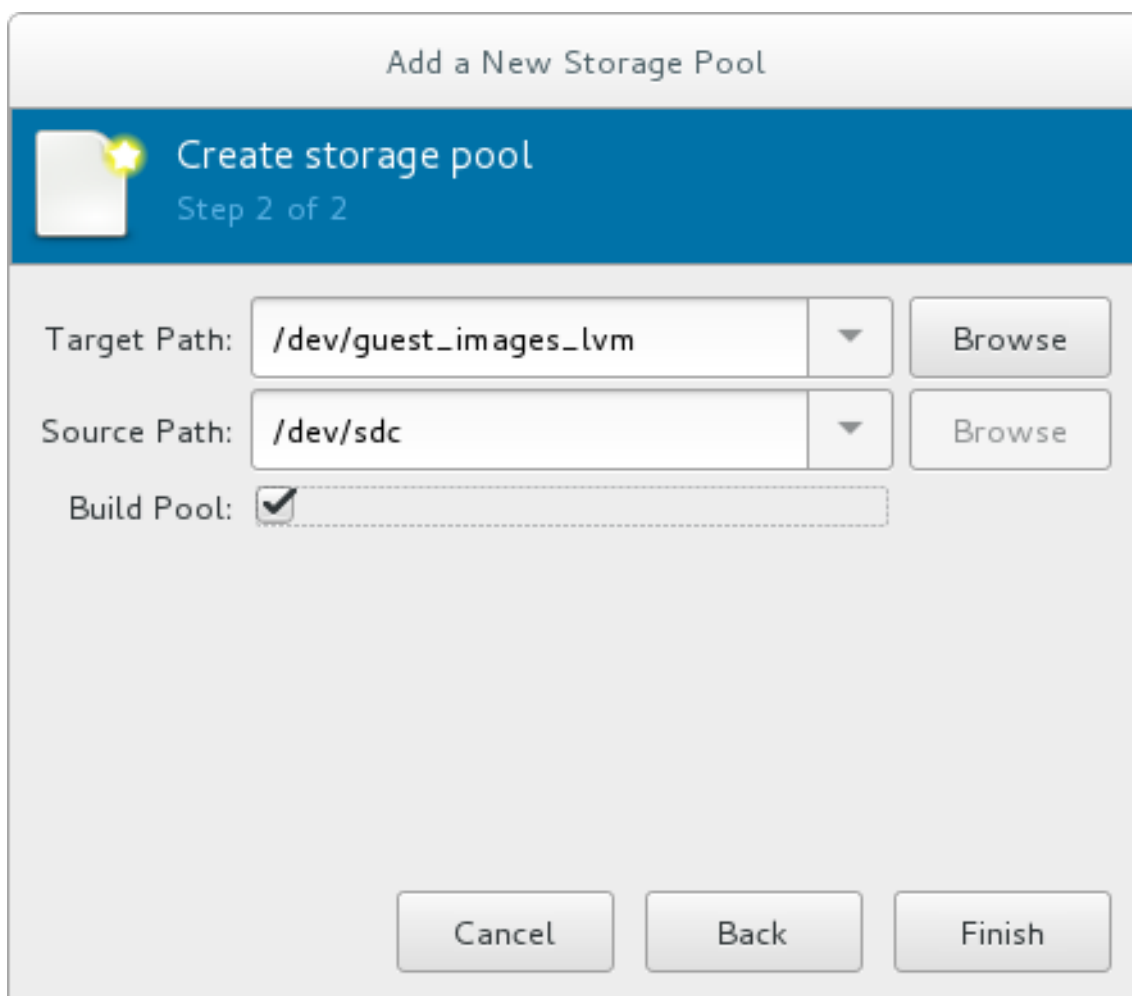


図13.11 ターゲットとソースの追加

詳細を確認して **完了** ボタンを押すと、LVM ボリュームグループがフォーマットされ、ストレージプールが作成されます。

- c. フォーマットするデバイスを確認します。  
警告メッセージが表示されます。

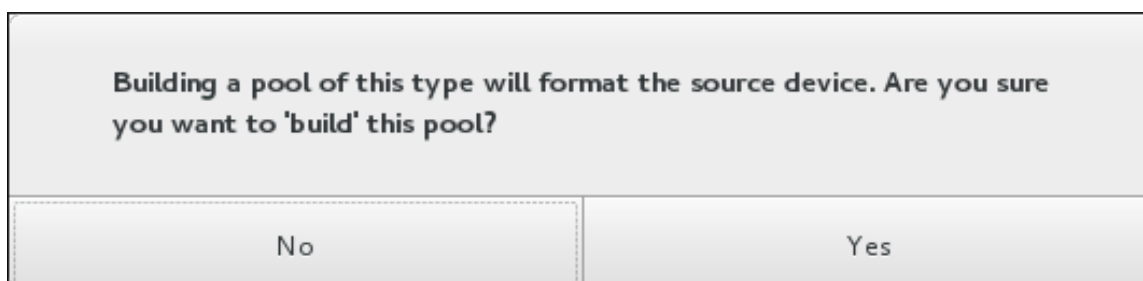


図13.12 警告メッセージ

**はい** ボタンを押すと、ストレージデバイス上のすべてのデータが消去され、ストレージプールが作成されます。

3. **新規ストレージプールを確認します。**  
数秒後に新しいストレージプールが左側の一覧に表示されます。詳細が予想通りであることを確認します。この例では、**75.15 GiB** 空きです。また、**状態** フィールドで新規ストレージプール



が **動作中** と報告されていることを確認します。

通常は **自動起動** チェックボックスにチェックマークを付けて有効にし、`libvirt` でストレージプールが自動的に起動されるようにしておく便利です。

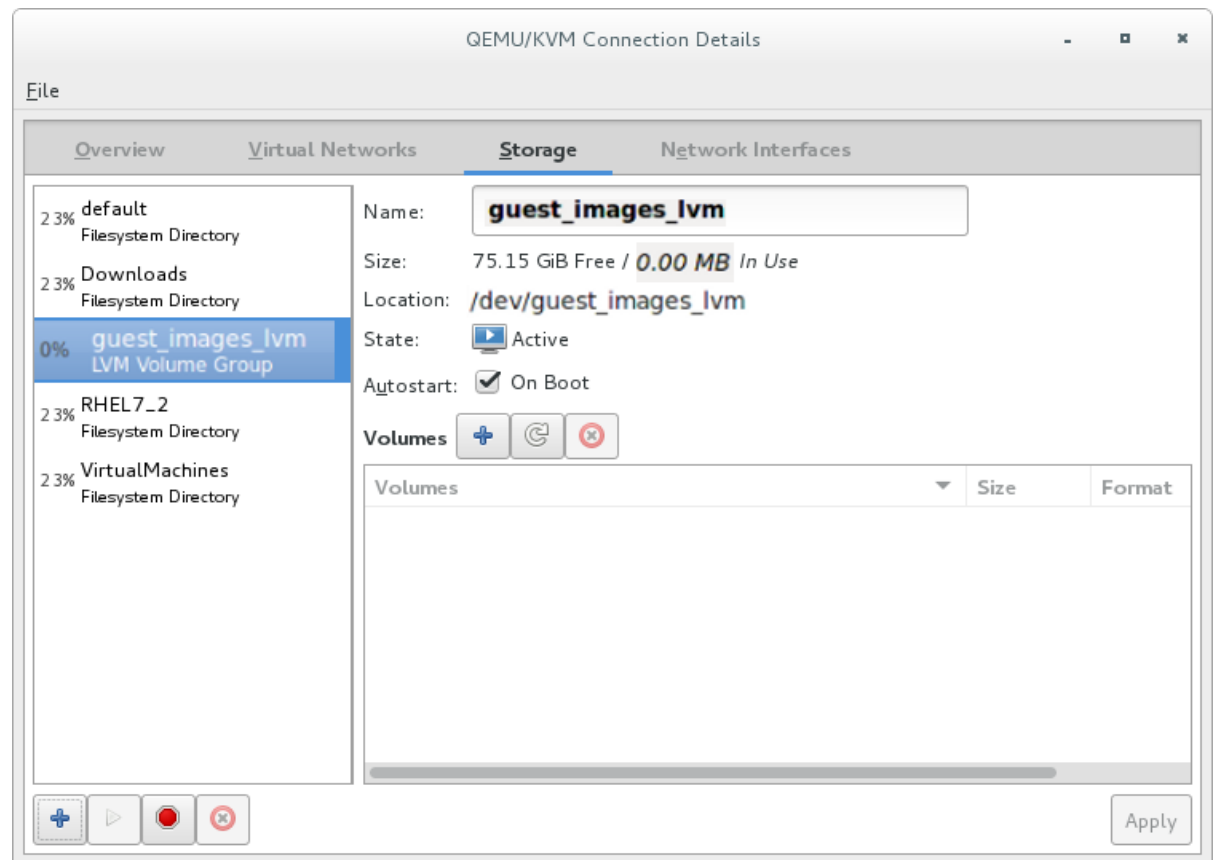



図13.13 LVM ストレージプールの詳細確認

これで作業は完了です。接続の詳細 ダイアログを閉じます。

### 13.4.2. virt-manager を使用したストレージプールの削除

以下の手順は、ストレージプールを削除する方法を説明します。

1. 同じプールを使用する他のゲスト仮想マシンに関連した問題を避けるには、ストレージプールを停止し、そのストレージプールで使用中のリソースをすべて解放するのが最良の方法です。

これを実行するには、停止するストレージプールを選択して、 をクリックします。

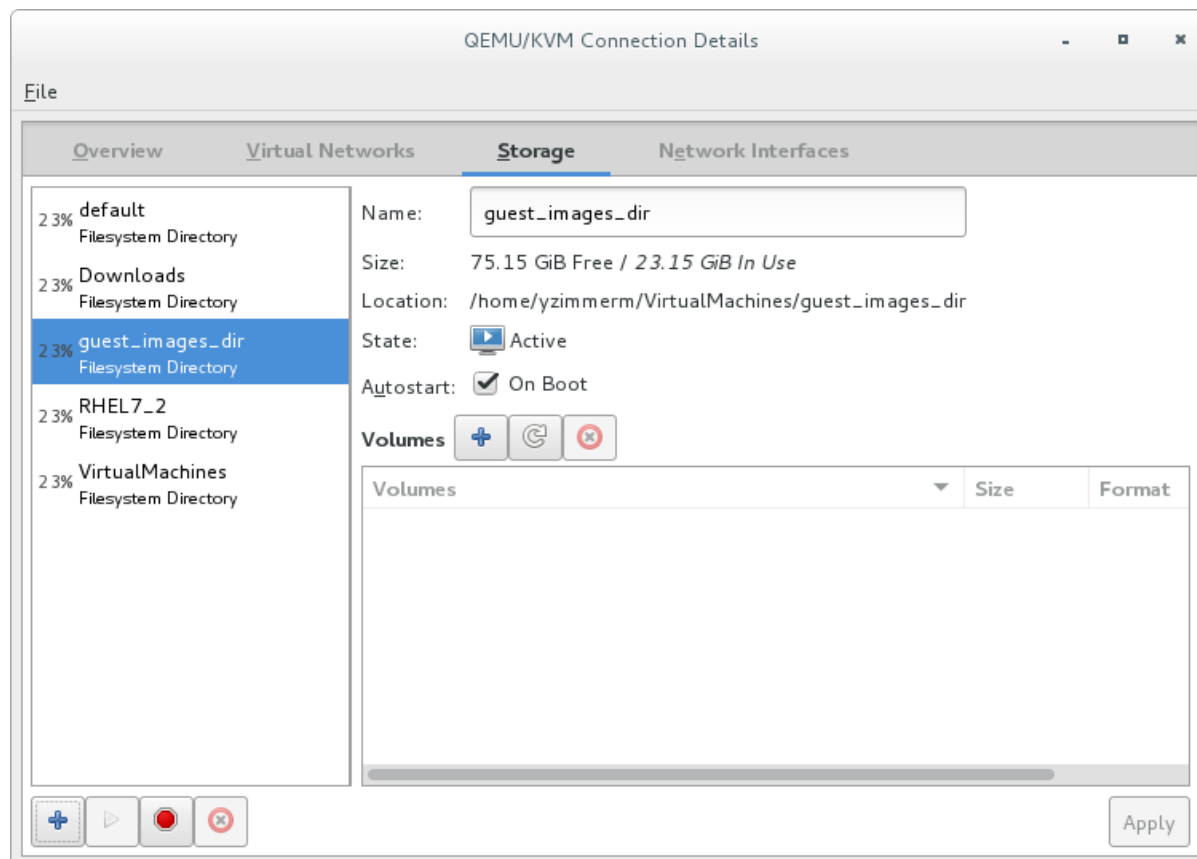



図13.14 停止アイコン

2. ストレージプールを削除するには、 をクリックします。このアイコンが使用できるのは、ストレージプールが停止している場合のみです。

### 13.4.3. virsh を使用した LVM ベースのストレージプールの作成

このセクションでは、**virsh** コマンドを使用して LVM ベースのストレージプールを作成する場合に必要な手順について簡単に説明します。ここでは、単ードライブ (**/dev/sdc**) からの **guest\_images\_lvm** という名前のプールを使用しています。これは、説明を行うための単なる例ですので、実際の設定を行う場合には適した値に置き換えてください。

#### 手順13.3 virsh を使用した LVM ベースのストレージプールの作成

1. プール名 **guest\_images\_lvm** を定義します。

```
# virsh pool-define-as guest_images_lvm logical - - /dev/sdc
libvirt_lvm \ /dev/libvirt_lvm
Pool guest_images_lvm defined
```

2. 指定した名前に基づいてプールを構築します。既存のボリュームグループをすでに使用している場合は、このステップを省略します。

```
# virsh pool-build guest_images_lvm
Pool guest_images_lvm built
```

3. 新規のプールを初期化します。

```
# virsh pool-start guest_images_lvm

Pool guest_images_lvm started
```

4. **vgs** コマンドを使用して、ボリュームグループ情報を表示します。

```
# vgs
VG          #PV #LV #SN Attr   VSize   VFree
libvirt_lvm    1   0   0 wz--n- 465.76g 465.76g
```

5. 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
```

6. **virsh** コマンドを使用して、利用可能なプールを一覧表示します。

```
# virsh pool-list --all
Name                               State      Autostart
-----
default                           active     yes
guest_images_lvm                  active     yes
```

7. 以下の一連のコマンドでは、このプール内に 3 つのボリュームを作成します (**volume1**、**volume2**、および **volume3**)。

```
# virsh vol-create-as guest_images_lvm volume1 8G
Vol volume1 created

# virsh vol-create-as guest_images_lvm volume2 8G
Vol volume2 created

# virsh vol-create-as guest_images_lvm volume3 8G
Vol volume3 created
```

8. **virsh** コマンドを使用して、このプール内の利用可能なボリュームを一覧表示します。

```
# virsh vol-list guest_images_lvm
Name                               Path
-----
volume1                          /dev/libvirt_lvm/volume1
volume2                          /dev/libvirt_lvm/volume2
volume3                          /dev/libvirt_lvm/volume3
```

9. 以下の 2 つのコマンド (**lvscan** と **lvs**) を使用して、新たに作成されたボリュームに関する詳細情報を表示します。

```
# lvscan
ACTIVE                               '/dev/libvirt_lvm/volume1' [8.00 GiB] inherit
ACTIVE                               '/dev/libvirt_lvm/volume2' [8.00 GiB] inherit
ACTIVE                               '/dev/libvirt_lvm/volume3' [8.00 GiB] inherit

# lvs
```

LV	VG	Attr	LSize	Pool	Origin	Data%	Move	Log
Copy%	Convert							
volume1	libvirt_lvm	-wi-a-	8.00g					
volume2	libvirt_lvm	-wi-a-	8.00g					
volume3	libvirt_lvm	-wi-a-	8.00g					

#### 13.4.4. virsh を使用したストレージプールの削除

以下は、`virsh` を使ってストレージプールを削除する方法を説明しています。

1. 同じプールを使用する他のゲストとの問題を避けるには、ストレージプールを停止してから使用中のリソースをすべて解放するのが最良の方法です。

```
# virsh pool-destroy guest_images_disk
```

2. また、ストレージプールがあるディレクトリーを削除したい場合は、次のコマンドを使用します。

```
# virsh pool-delete guest_images_disk
```

3. ストレージプールの定義を削除します。

```
# virsh pool-undefine guest_images_disk
```

### 13.5. iSCSI ベースのストレージプール

このセクションでは、ゲスト仮想マシンを格納するために iSCSI ベースのデバイスを作成する方法について説明します。これにより、ブロックストレージデバイスとして iSCSI を使用するなどのさらに柔軟なストレージオプションが許可されます。iSCSI デバイスは、Linux のマルチプロトコル SCSI ターゲットである LIO ターゲットを使用します。iSCSI のほかにも、LIO はファイバーチャネルおよびファイバーチャネルオーバーイーサネット (FCoE: Fibre Channel over Ethernet) もサポートします。

iSCSI (Internet Small Computer System Interface) とは、ストレージデバイスを共有するためのネットワークプロトコルです。iSCSI では、IP 層全体での SCSI の指示を使用してイニシエーター (ストレージクライアント) をターゲット (ストレージサーバー) に接続します。

#### 13.5.1. ソフトウェア iSCSI ターゲットの設定

Red Hat Enterprise Linux 7 で導入された iSCSI ターゲットは、`targetcli` パッケージで作成されます。これにより、ソフトウェアベースの iSCSI ターゲットを作成するためのコマンドセットが提供されます。

##### 手順13.4 iSCSI ターゲットの作成

1. 必須パッケージをインストールします。  
`targetcli` パッケージとすべての依存関係をインストールします。

```
# yum install targetcli
```

2. `targetcli` を起動します。  
`targetcli` コマンドセットを起動します。

■

```
# targetcli
```

### 3. ストレージオブジェクトを作成します。

「[LVM ベースのストレージプール](#)」で作成されたデバイスを使用して、以下のように 3 つのストレージオブジェクトを作成します。

- a. **/backstores/block** ディレクトリーに変更してから以下のコマンドを実行して、ブロックストレージオブジェクトを作成します。

```
# create [block-name][filepath]
```

例:

```
# create block1 dev=/dev/vdb1
```

- b. **fileio** ディレクトリーに変更してから以下のコマンドを実行して、**fileio** オブジェクトを作成します。

```
# create [fileioname] [imagename] [image-size]
```

例:

```
# create fileio1 /foo.img 50M
```

- c. **ramdisk** ディレクトリーに変更してから以下のコマンドを実行して、**ramdisk** オブジェクトを作成します。

```
# create [ramdiskname] [size]
```

例:

```
# create ramdisk1 1M
```

- d. このステップで作成したディスクの名前は後で必要になるため、忘れないようにしてください。

### 4. /iscsi ディレクトリーに移動します。

**iscsi** ディレクトリーに変更します。

```
#cd /iscsi
```

### 5. iSCSI ターゲットを作成します。

iSCSI ターゲットを以下の 2 つの方法で作成します。

- a. 追加パラメーターが指定されていない **create** は、IQN を自動生成します。
- b. **create iqn.2010-05.com.example.server1:iscsirhel7guest** は、指定されるサーバーに特定の IQN を作成します。

### 6. ターゲットポータルグループ (TPG) を定義します。

各 iSCSI ターゲットでは、ターゲットポータルグループ (TPG) を定義する必要があります。この例では、デフォルトの **tpg1** が使用されますが、さらに多くの **tpg** を追加することもできま

す。これは最も一般的な設定であるため、この例では **tpg1** を設定します。これを実行するには、**/iscsi** ディレクトリーにいることを確認してから、**/tpg1** ディレクトリーに変更します。

```
# /iscsi>iqn.iqn.2010-05.com.example.server1:iscsirhel7guest/tpg1
```

## 7. ポータル IP アドレスを定義します。

iSCSI 上でブロックストレージをエクスポートするには、ポータル、LUN および ACL をすべて最初に設定する必要があります。

ポータルには、ターゲットがリッスンし、イニシエーターが接続する IP アドレスおよび TCP ポートが含まれます。iSCSI は、デフォルトで設定されるポートのポート **3260** を使用します。このポートに接続するには、**/tpg** ディレクトリーから以下のコマンドを実行します。

```
# portals/ create
```

このコマンドには、このポートをリッスンするすべての利用可能な IP アドレスが含まれます。単一の IP アドレスのみがポートでリッスンするように指定するには、**portals/ create [ipaddress]** を実行します。指定される IP アドレスはポート **3260** をリッスンするように設定されます。

## 8. LUN を設定し、ストレージオブジェクトをファブリックに割り当てます。

このステップでは、[手順13.4「iSCSI ターゲットの作成」](#)で作成されるストレージデバイスを使用します。ステップ 6 で作成した TPG の **luns** ディレクトリーに、またはたとえば **iscsi>iqn.iqn.2010-05.com.example.server1:iscsirhel7guest** となるように必ず変更を加えてください。

- a. 最初の LUN を以下のように **ramdisk** に割り当てます。

```
# create /backstores/ramdisk/ramdisk1
```

- b. 2 番目の LUN を以下のようにブロックディスクに割り当てます。

```
# create /backstores/block/block1
```

- c. 3 番目の LUN を以下のように **fileio** ディスクに割り当てます。

```
# create /backstores/fileio/file1
```

- d. 結果として生じる LUN の一覧表示はこの画面の出力のようになります。

```
/iscsi/iqn.20...csirhel7guest/tpg1 ls

o- tpg1
.....[enabled, auth]
  o-
  acs.....[0 ACL]
  o-
  luns.....[3 LUNs]
    | o-
```

```

lun0.....
.....[ramdisk/ramdisk1]
    | o-
lun1.....
.[block/block1 (dev/vdb1)]
    | o-
lun2.....
..[fileio/file1 (foo.img)]
    o-
portals.....
.....[1 Portal]
    o- IP-
ADDRESS:3260.....
.....[OK]

```

## 9. 各イニシエーターに ACL を作成します。

このステップでは、イニシエーターが接続する際に認証の作成を許可します。さらに、指定された LUN の指定されたイニシエーターへの制限を許可します。ターゲットとイニシエーターの両方には、固有の名前があります。iSCSI イニシエーターは IQN を使用します。

- a. iSCSI イニシエーターの IQN を検索するには、イニシエーターの名前を置き換え、以下のコマンドを実行します。

```
# cat /etc/iscsi/initiatorname.iscsi
```

この IQN を使用して ACL を作成します。

- b. **acls** ディレクトリーに切り替えます。

- c. コマンド **create [iqn]** を実行するか、または特定の ACL を作成します。以下の例を参照してください。

```
# create iqn.2010-05.com.example.foo:888
```

または、すべてのイニシエーターの単一ユーザー ID およびパスワードを使用するようにカーネルターゲットを設定し、そのユーザー ID およびパスワードですべてのイニシエーターがログインできるようにするには、以下のコマンドを使用します (**userid** および **password** を置き換えます)。

```

# set auth userid=redhat
# set auth password=password123
# set attribute authentication=1
# set attribute generate_node_acls=1

```

10. **saveconfig** コマンドを使って設定を永続化します。これにより、直前の起動設定を上書きします。または、**targetcli** から **exit** を実行すると、デフォルトでターゲット設定が保存されます。

11. **systemctl enable target.service** でサービスを有効にし、次の起動時に保存された設定を適用します。

## 手順13.5 オプションのステップ

1. LVM ボリュームを作成します。

LVM ボリュームは、iSCSI のバックアップイメージに役に立ちます。LVM のスナップショットやサイズ変更は、ゲスト仮想マシンに使える便利な機能です。この例では、iSCSI でゲスト仮想マシンをホストするために RAID5 アレイ上の *virtstore* という名前の新規ボリュームグループ上に *virtimage1* という名前の LVM イメージを作成しています。

a. RAID アレイを作成します。

ソフトウェア RAID5 アレイの作成については、『[Red Hat Enterprise Linux 7 ストレージ管理ガイド](#)』で説明されています。

b. LVM ボリュームグループを作成します。

**vgcreate** コマンドを使用して *virtstore* という名前の論理ボリュームグループを作成します。

```
# vgcreate virtstore /dev/md1
```

c. LVM 論理ボリュームを作成します。

**lvcreate** コマンドを使用して、*virtimage1* という名前の論理ボリュームグループ (サイズは 20GB) を *virtstore* ボリュームグループ上に作成します。

```
# lvcreate --size 20G -n virtimage1 virtstore
```

これで新規論理ボリュームの *virtimage1* を iSCSI に使用する準備が整いました。



### 重要

カーネルのターゲットバックストアの LVM ボリュームを使用すると、イニシエーターが LVM を使ってエクスポートされたボリュームのパーティション設定を行う場合に問題が発生する可能性があります。これは、**global\_filter = ["r|^/dev/vg0|"]** を */etc/lvm/lvm.conf* に追加することによって解決できます。

## 2. オプション: 検出テスト

新規の iSCSI デバイスが検出可能かどうかを検証します。

```
# iscsiadm --mode discovery --type sendtargets --portal
server1.example.com
127.0.0.1:3260,1 iqn.2010-05.com.example.server1:iscsirhel7guest
```

## 3. オプション: デバイス接続テスト

新規デバイス (*iqn.2010-05.com.example.server1:iscsirhel7guest*) を割り当て、デバイスが割り当て可能であるかどうかを判別します。

```
# iscsiadm -d2 -m node --login
scsiadm: Max file limits 1024 1024

Logging in to [iface: default, target: iqn.2010-
05.com.example.server1:iscsirhel7guest, portal: 10.0.0.1,3260]
Login to [iface: default, target: iqn.2010-
05.com.example.server1:iscsirhel7guest, portal: 10.0.0.1,3260]
successful.
```

## 4. デバイスの割り当てを解除します。

■



```
# iscsiadm -d2 -m node --logout
scsiadm: Max file limits 1024 1024

Logging out of session [sid: 2, target: iqn.2010-
05.com.example.server1:iscsirhel7guest, portal: 10.0.0.1,3260]
Logout of [sid: 2, target: iqn.2010-
05.com.example.server1:iscsirhel7guest, portal: 10.0.0.1,3260]
successful.
```

これでiSCSIデバイスを仮想化に使用する準備が整いました。

### 13.5.2. virt-manager での iSCSI ストレージプールの作成

以下の手順では、**virt-manager** で iSCSI ターゲットを持つストレージプールを作成します。

#### 手順13.6 iSCSI デバイスの virt-manager への追加

1. ホストマシンのストレージ詳細を開きます。
  - a. **virt-manager** で、**編集** をクリックしてから、ドロップダウンメニューより **接続の詳細** を選択します。
  - b. **ストレージ** タブをクリックします。

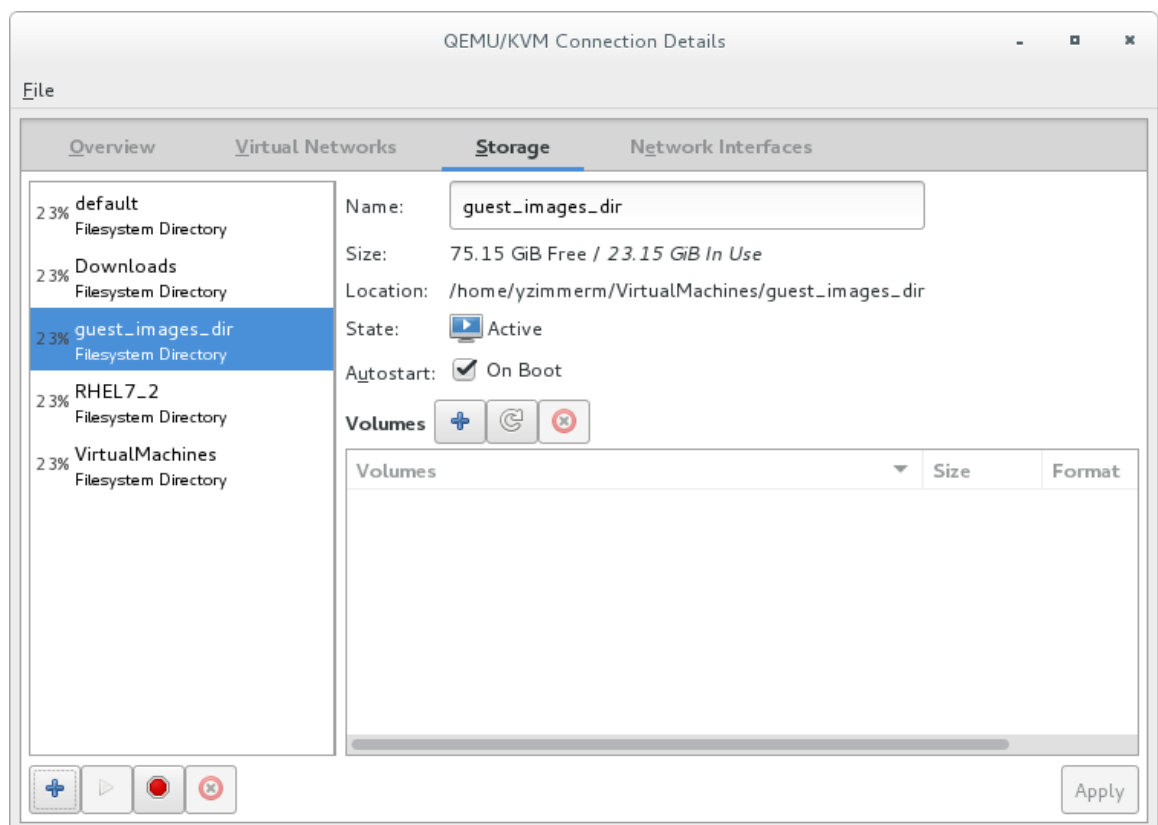
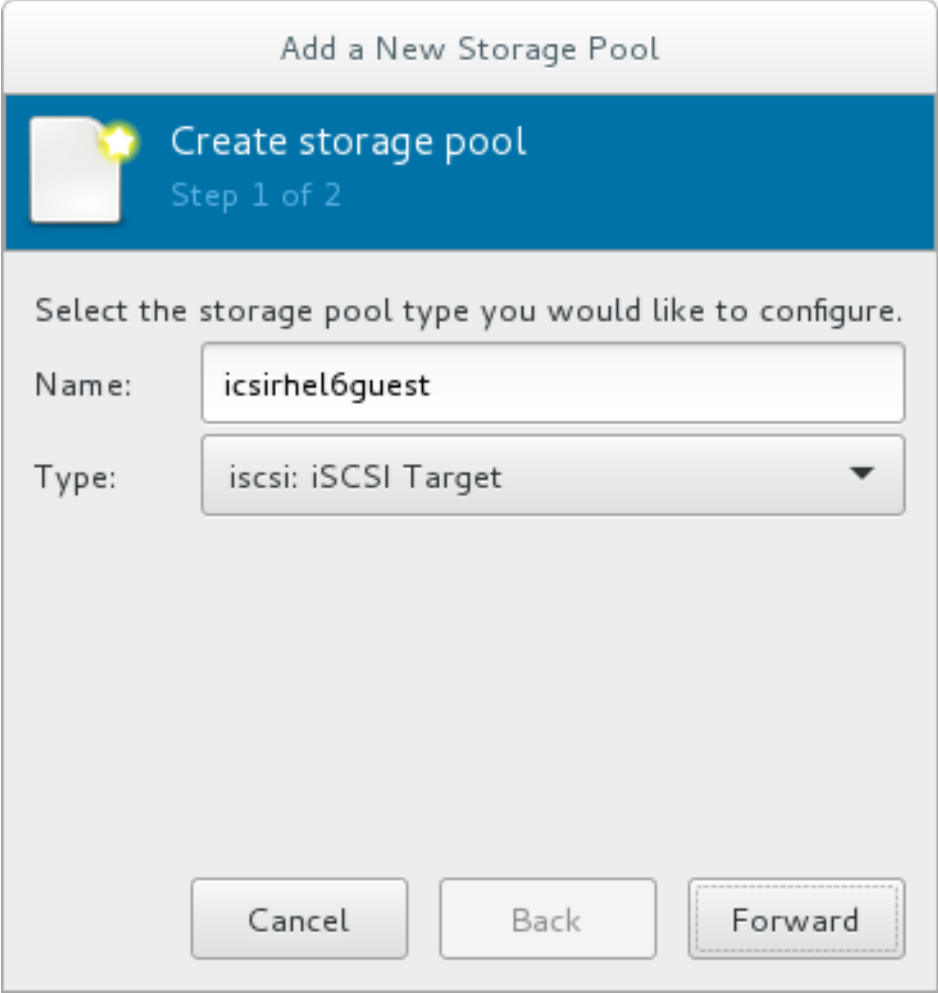


図13.15 ストレージメニュー

2. 新規プールを追加します (ステップ 1/2)。
  - + ボタン (プールの追加ボタン) を押します。 **新規ストレージプールを追加** ウィザードが表示されます。



Add a New Storage Pool

Create storage pool  
Step 1 of 2

Select the storage pool type you would like to configure.

Name: icsirhel6guest

Type: iscsi: iSCSI Target

Cancel Back Forward

図13.16 iSCSI ストレージプールの名前とタイプを追加します。

ストレージプールの名前を選択して、タイプを iSCSI に変更してから **進む** を押して次に進みます。

### 3. 新規プールを追加します (ステップ 2/2)。

このメニューのフィールドへの入力を完了するには、「[iSCSI ベースのストレージプール](#)」で  
使用した情報が必要になります。

- a. iSCSI ソースおよびターゲットを入力します。フォーマットはゲスト仮想マシンが処理するため、**フォーマット** オプションは選択できません。さらに、**ターゲットパス** を編集することは推奨されていません。デフォルトのターゲットパスの値 **/dev/disk/by-path/** は、ドライブパスをそのディレクトリーに追加します。ターゲットパスは、移行のためにすべてのホスト物理マシン上で同一である必要があります。
- b. iSCSI ターゲットのホスト名または IP アドレスを入力します。この例では、**host1.example.com** を使用します。
- c. ソース **IQN** フィールドには、iSCSI ターゲット IQN を入力します。「[iSCSI ベースのストレージプール](#)」を参照すると、これは **/etc/tgt/targets.conf** file に追加した情報であることが分かります。この例では、**iqn.2010-05.com.test\_example.server1:icsirhel7guest** を使用しています。
- d. (オプション) イニシエーターの **IQN** チェックボックスにチェックマークを付けて、イニシエーターの IQN を入力します。この例では、**iqn.2010-05.com.example.host1:icsirhel7** を使用しています。


e. **完了** をクリックすると、新規のストレージプールが作成されます。

図13.17 iSCSI ストレージプールの作成

### 13.5.3. virt-manager を使用したストレージプールの削除

以下の手順では、ストレージプールを削除する方法を説明します。

1. 同じプールを使用する他のゲスト仮想マシンに関連した問題を避けるには、ストレージプールを停止し、そのストレージプールで使用中のリソースをすべて解放するのが最良の方法です。

これを実行するには、停止するストレージプールを選択して、 をクリックします。

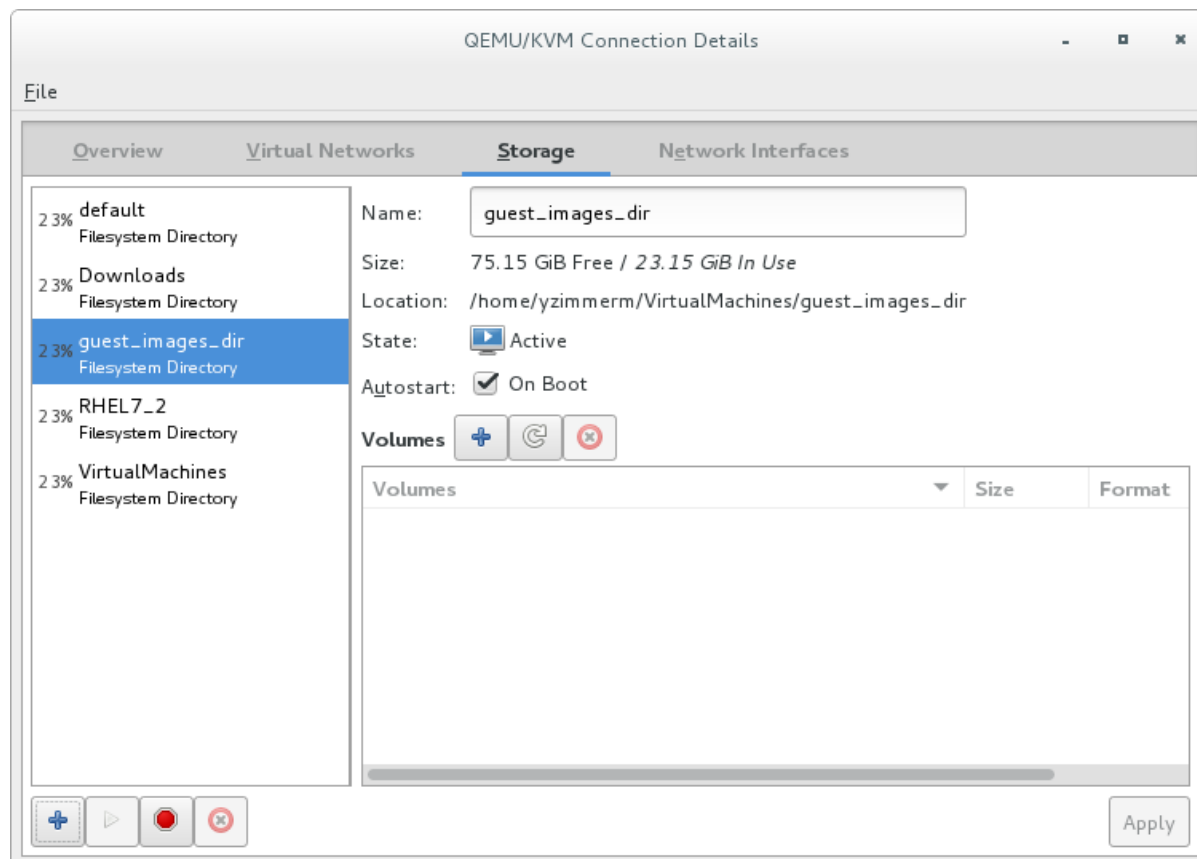



図13.18 ストレージプールの削除

2. ストレージプールを削除するには、 をクリックします。このアイコンが使用できるのは、ストレージプールが停止している場合のみです。

#### 13.5.4. virsh を使用した iSCSI ベースのストレージプールの作成

1. オプション: ストレージプールのセキュリティを保護します。  
必要な場合は、「[iSCSI ストレージプールのセキュリティ保護](#)」にあるステップで認証をセットアップします。
2. ストレージプールを定義します。  
ストレージプールの定義は、**virsh** コマンドラインツールで作成できます。**virsh** でのストレージプール作成は、複数のストレージプールをスクリプトで作成しているシステム管理者にとって便利な方法です。

**virsh pool-define-as** コマンドにはパラメーターがいくつかあり、以下の形式で使します。

```
virsh pool-define-as name type source-host source-path source-dev
source-name target
```

以下でパラメーターについて説明します。

##### type

たとえば、このプールを特定タイプ iSCSI として定義します。

##### name

ストレージプールの名前を設定します。この名前は固有である必要があります。

**source-host と source-path**

ホスト名と iSCSI IQN です。

**source-dev と source-name**

これらのパラメーターは iSCSI ベースのプールでは不要です。- 文字を使用してフィールドをブランクのままにします。

**target**

ホストマシン上で iSCSI デバイスをマウントする場所を定義します。

以下の例では、上記の **virsh pool-define-as** の例と同じ iSCSI ベースのストレージプールを作成します。

```
# virsh pool-define-as --name iscsirhel7pool --type iscsi \
    --source-host server1.example.com \
    --source-dev iqn.2010-05.com.example.server1:iscsirhel7guest \
    --target /dev/disk/by-path
Pool iscsirhel7pool defined
```

**3. ストレージプールが一覧表示されていることを確認します**

ストレージプールのオブジェクトが正しく作成されており、状態が **inactive** であることを確認します。

```
# virsh pool-list --all
Name                      State      Autostart
-----
default                   active    yes
iscsirhel7pool            inactive  no
```

**4. オプション: iSCSI ストレージプールへの直接接続を確立します。**

以下のステップはオプションですが、iSCSI ストレージプールへの直接の接続を確立することができます。デフォルトでこれは有効にされていますが、ホストマシンへの接続 (ネットワークへの直接の接続ではない) が設定されている場合、この例を反映するように仮想マシンのドメイン XML を編集することによって、設定を元に戻すことができます。

```
...
<disk type='volume' device='disk'>
  <driver name='qemu' />
  <source pool='iscsi' volume='unit:0:0:1' mode='direct' />
  <target dev='vda' bus='virtio' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x06'
function='0x0' />
</disk>
...
```

図13.19 ディスクタイプ要素 XML の例



## 注記

**disk device** を **disk** または **lun** として設定することで、同一の iSCSI ストレージプールを LUN またはディスクに使用できます。SCSI LUN ベースのストレージをゲストに追加するためのサンプル XML 設定については、「[SCSI LUN ベースのストレージのゲストへの追加](#)」を参照してください。

さらに、ホストマシンへの接続のために **source mode** を **mode='host'** と指定できます。

手順13.4「[iSCSI ターゲットの作成](#)」に詳しく説明されているように iSCSI サーバーに認証を設定している場合、**<disk>** サブ要素として使用される以下の XML はディスクの認証資格情報を提供します。「[iSCSI ストレージプールのセキュリティー保護](#)」は、libvirt シークレットの設定方法を説明しています。

```
<auth type='chap' username='redhat'>
  <secret usage='iscsirhel7secret' />
</auth>
```

## 5. ストレージプールを起動します。

**virsh pool-start** を使用してディレクトリーストレージプールを有効にします。これにより、ストレージプールをボリュームおよびゲスト仮想マシンに使用することができます。

```
# virsh pool-start iscsirhel7pool
Pool iscsirhel7pool started
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
iscsirhel7pool                     active     no
```

## 6. autostart をオンにします。

ストレージプールの **autostart** をオンにします。Autostart は、**libvirtd** サービスの起動時にストレージプールを起動するように設定します。

```
# virsh pool-autostart iscsirhel7pool
Pool iscsirhel7pool marked as autostarted
```

**iscsirhel7pool** プールが **autostart** を有効にしていることを確認します。

```
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
iscsirhel7pool                     active     yes
```

## 7. ストレージプールの設定を確認します。

ストレージプールが正しく作成されたこと、サイズが正しく報告されたこと、および状態が **running** として報告されていることを確認します。

```
# virsh pool-info iscsirhel7pool
Name:          iscsirhel7pool
UUID:          afcc5367-6770-e151-bcb3-847bc36c5e28
```

```

State:          running
Persistent:     unknown
Autostart:      yes
Capacity:       100.31 GB
Allocation:     0.00
Available:      100.31 GB

```

これで *iscsirhel7pool* という iSCSI ベースのプールが利用可能になります。

### 13.5.5. iSCSI ストレージプールのセキュリティー保護

**virsh** を使ってユーザー名およびパスワードパラメーターを設定することにより、iSCSI ストレージプールのセキュリティーを保護することができます。これは、プールを定義する前後に設定できますが、認証設定を有効にするには、プールが起動している必要があります。

#### 手順13.7 virsh を使用したストレージプールの認証設定

1. **libvirt シークレットファイルを作成します。**

以下の例を使って、**secret.xml** という libvirt シークレット XML ファイルを作成します。

```

# cat secret.xml
<secret ephemeral='no' private='yes'>
  <description>Passphrase for the iSCSI example.com
server</description>
  <auth type='chap' username='redhat' />
  <usage type='iscsi'>
    <target>iscsirhel7secret</target>
  </usage>
</secret>

```

2. シークレットファイルを定義します。

**virsh** を使って **secret.xml** ファイルを定義します。

```

# virsh secret-define secret.xml

```

3. シークレットファイルの **UUID** を確認します。

**secret.xml** の **UUID** を確認します。

```

# virsh secret-list

      UUID                                     Usage
-----
2d7891af-20be-4e5e-af83-190e8a922360  iscsi iscsirhel7secret

```

4. シークレットを **UUID** に割り当てます。

例のように以下のコマンド構文を使用して、シークレットを該当の **UUID** に割り当てます。

```

# MYSECRET=`printf %s "password123" | base64`
# virsh secret-set-value 2d7891af-20be-4e5e-af83-190e8a922360
$MYSECRET

```

これにより、CHAP ユーザー名およびパスワードが `libvirt` で制御されたシークレット一覧に設定されることを確認できます。

##### 5. 認証エントリーをストレージプールに追加します。

`virsh edit` を使用してストレージプールの XML ファイルの `<source>` エントリーを変更し、**`authentication type`**、**`username`**、および **`secret usage`** を指定して `<auth>` 要素を追加します。

以下は、認証が設定されたストレージプール XML 定義の例を示しています。

```
# cat iscsirhel7pool.xml
<pool type='iscsi'>
  <name>iscsirhel7pool</name>
  <source>
    <host name='192.168.122.1' />
    <device path='iqn.2010-
05.com.example.server1:iscsirhel7guest' />
    <auth type='chap' username='redhat'>
      <secret usage='iscsirhel7secret' />
    </auth>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```



#### 注記

**`<auth>`** サブ要素は、ゲスト XML の **`<pool>`** および **`<disk>`** 要素内の複数の異なる場所に存在します。**`<pool>`** の場合、**`<auth>`** は、認証が一部のプールソース (iSCSI および RBD) のプロパティーであり、プールソースの検索方法を記述しているため、**`<source>`** 要素内に指定されます。ドメインのサブ要素である **`<disk>`** の場合、iSCSI または RBD ディスクへの認証はディスクのプロパティーになります。ゲスト XML に設定される **`<disk>`** の例については、「[virsh 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。

##### 6. ストレージプール内の変更をアクティブにします。

以下の変更をアクティブにするには、ストレージプールを起動している必要があります。

ストレージプールをまだ起動していない場合は、「[virsh を使用した iSCSI ベースのストレージプールの作成](#)」にあるステップに従ってストレージプールを定義し、起動します。

プールがすでに起動している場合は、以下のコマンドを実行してストレージプールを停止し、再起動します。

```
# virsh pool-destroy iscsirhel7pool
# virsh pool-start iscsirhel7pool
```

### 13.5.6. virsh を使用したストレージプールの削除

以下では、`virsh` を使ってストレージプールを削除する方法を説明します。



1. 同じプールを使用する他のゲスト仮想マシンとの問題を避けるには、ストレージプールを停止してから使用中のリソースをすべて解放するのが最良の方法です。

```
# virsh pool-destroy iscsirhel7pool
```

2. ストレージプールの定義を削除します。

```
# virsh pool-undefine iscsirhel7pool
```

## 13.6. NFS ベースのストレージプール

このセクションでは、NFS マウントポイントを使ってストレージプールを作成および削除する方法について説明します。ここでは、NFS が既にホストマシンにマウントされていることを前提としています。NFS マウントポイント作成および追加の詳細については、『[Red Hat Enterprise Linux 7 ストレージ管理ガイド](#)』を参照してください。

### 13.6.1. virt-manager を使用した NFS ベースのストレージプールの作成

1. ホスト物理マシンの「ストレージ」タブを開きます  
**接続の詳細** ウィンドウ内の **ストレージ** タブを開きます。
  - a. **virt-manager** を開きます。
  - b. **virt-manager** のメインウィンドウからホスト物理マシンを選択します。**編集** メニューをクリックして、**接続の詳細** を選択します。
  - c. ストレージタブをクリックします。

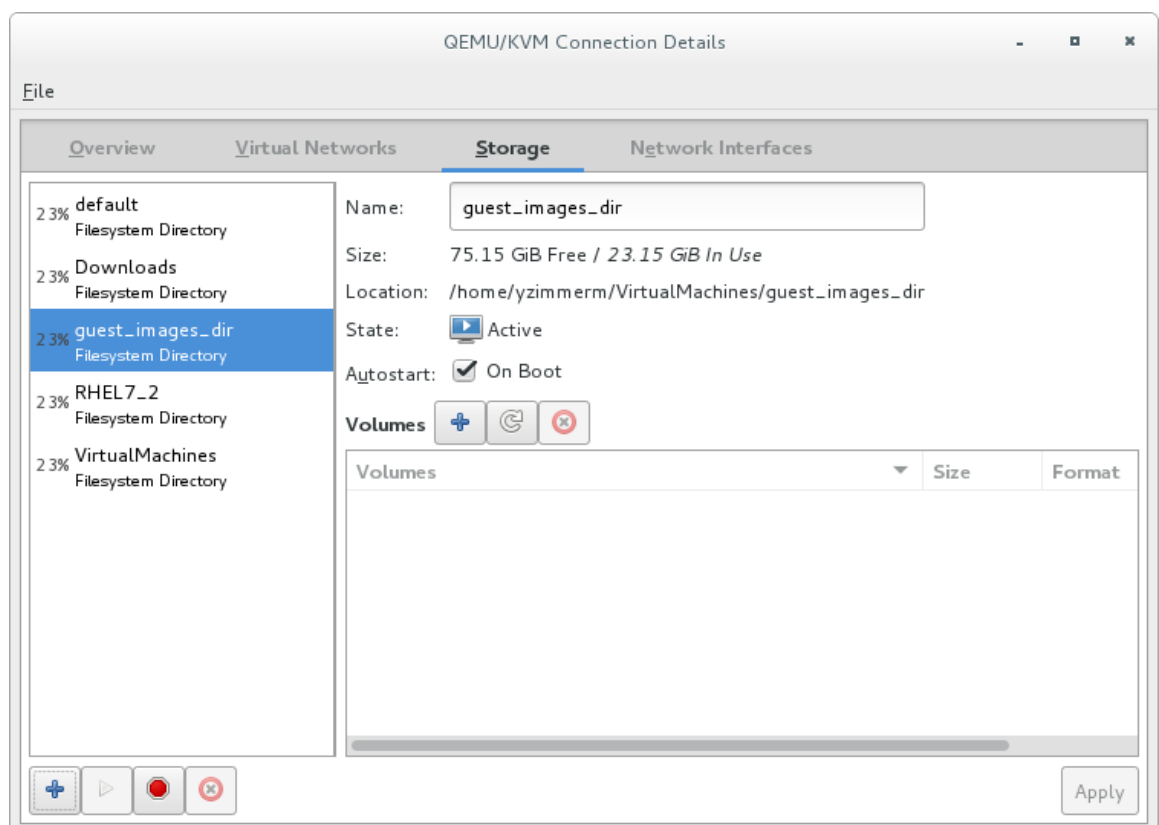
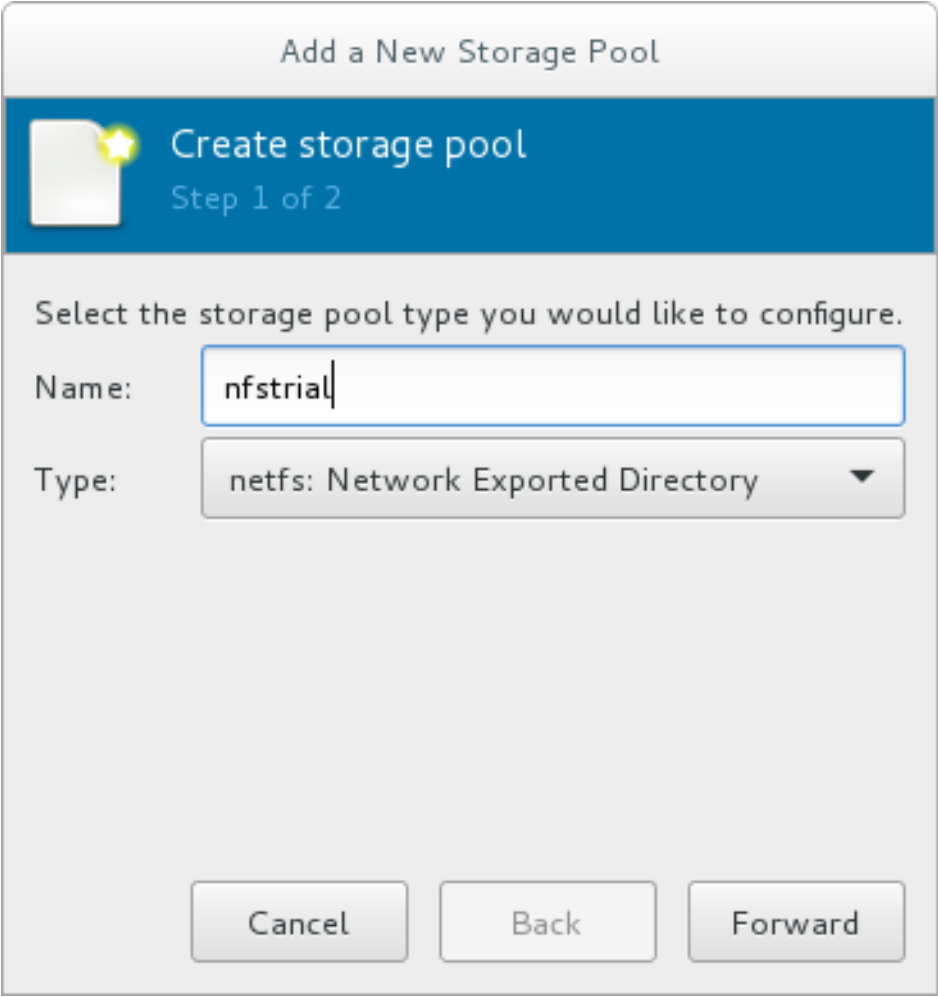


図13.20 「ストレージ」タブ

## 2. 新規プールの作成 (パート 1)

+ ボタン (プールの追加ボタン) を押します。新規ストレージプールを追加 ウィザードが表示されます。



Add a New Storage Pool

Create storage pool  
Step 1 of 2

Select the storage pool type you would like to configure.

Name: nfstrial

Type: netfs: Network Exported Directory

Cancel Back Forward

図13.21 NFS の名前と種類を追加

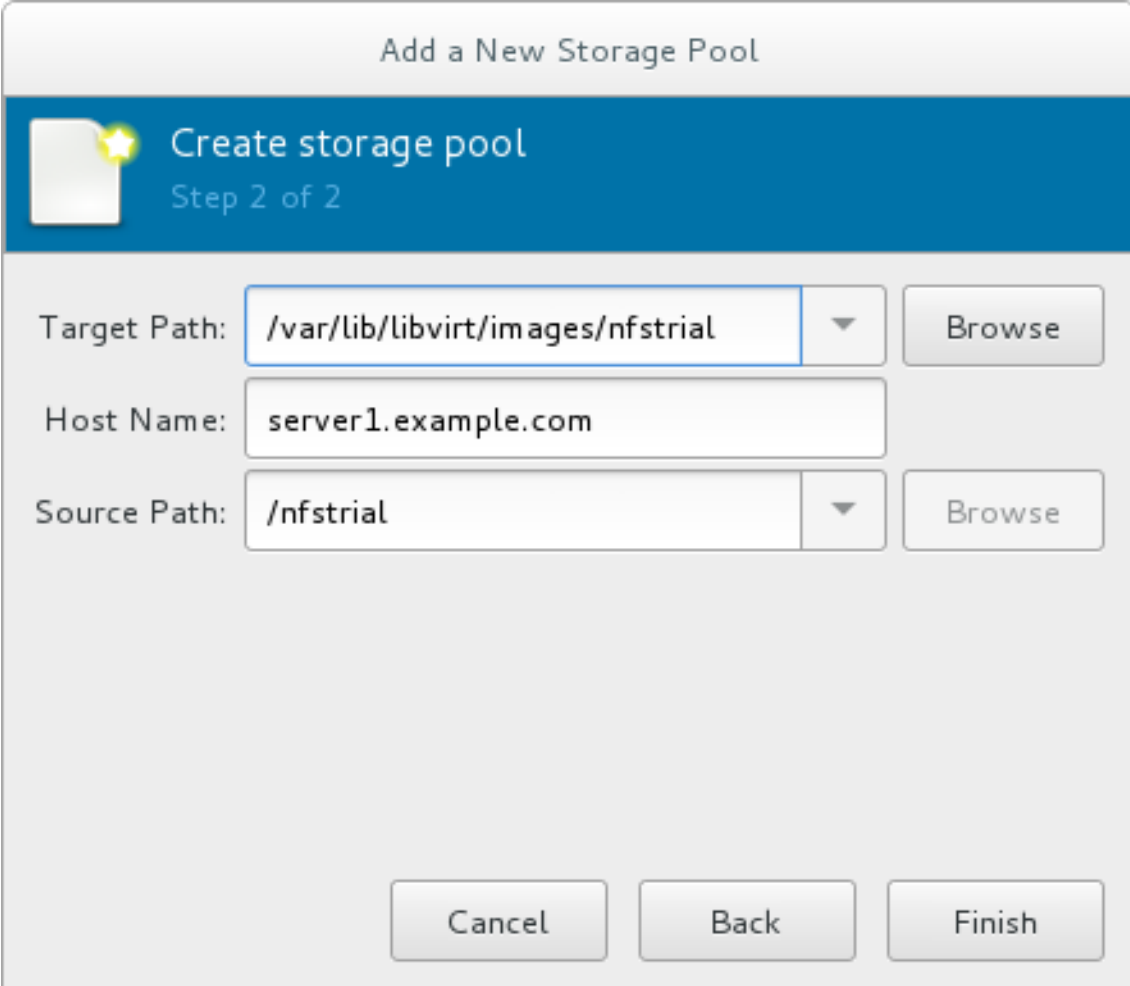
ストレージプールの名前を選択し、進むを押して次に進みます。

## 3. 新規プールの作成 (パート 2)

デバイスのターゲットパス、ホスト名、および NFS 共有パスを入力します。フォーマット オプションを **NFS** または **auto** (タイプを検出する) に設定します。ターゲットパスは移行のためにすべてのホスト物理マシン上で同一でなければなりません。

NFS サーバーのホスト名または IP アドレスを入力します。この例では、**server1.example.com** を使用しています。

NFS パスを入力します。この例では、**/nfstrial** を使用しています。



Target Path: /var/lib/libvirt/images/nfstrial Browse

Host Name: server1.example.com

Source Path: /nfstrial Browse

Cancel Back Finish

図13.22 NFS ストレージプールの作成

完了 を押すと、新規のストレージプールが作成されます。

### 13.6.2. virt-manager を使用したストレージプールの削除

以下の手順では、ストレージプールを削除する方法を説明します。

1. 同じプールを使用する他のゲストに関連した問題を避けるには、ストレージプールを停止し、そのストレージプールで使用中のリソースをすべて解放するのが最良の方法です。これを実行

するには、停止するストレージプールを選択して、 をクリックします。

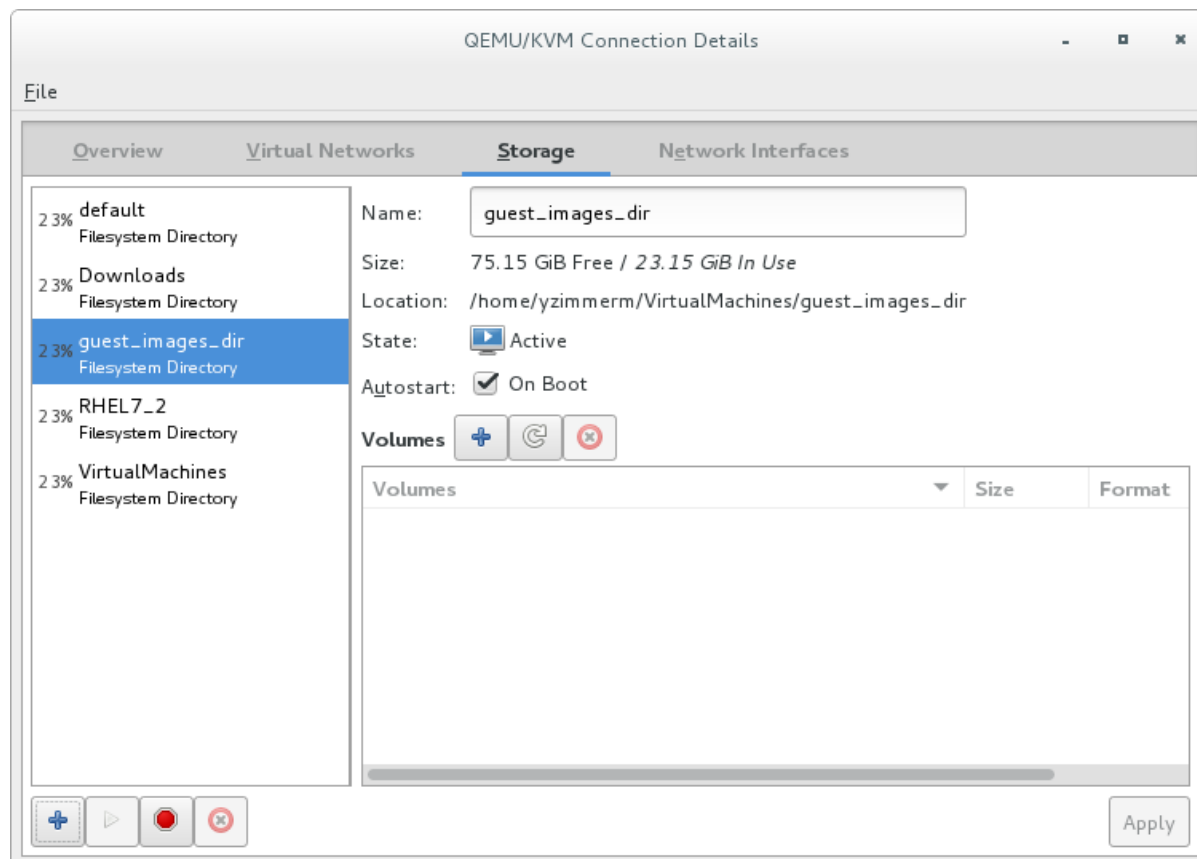


図13.23 停止アイコン

2. ゴミ箱アイコンをクリックしてストレージプールを削除します。ストレージプールを停止しておかないとこのアイコンは使用できません。

### 13.6.3. virsh を使用した NFS ベースのストレージプールの作成

#### 手順13.8 NFS ベースのストレージプールの作成

1. ストレージプールの定義を作成します。

新規の永続ストレージプールを定義するには、**virsh pool-define-as** コマンドを使用します。新規の非永続ストレージプールを定義するには、**virsh pool-create-as** コマンドを使用します。

永続ストレージプールは、ゲスト仮想マシンが実行されていない場合でもアクセスが可能で、ホストの再起動後にも存在し続けます。非永続ストレージプールは、ゲスト仮想マシンが実行されている場合に限り利用可能で、ホストの再起動後には存在しません。

この例では、永続ストレージプールが使われています。

```
# virsh pool-define-as nfspool netfs --sourcehost localhost --
source-path /home/path/to/mountpoint/directory --target
/tmp/nfspool-client
Pool nfspool defined
```

NFS ベースのストレージプールを作成するには、以下のオプションが必要です。

- ストレージプールの名前。

この例では、名前に **nfspool** が使われています。この例で使用されているこれ以降の全ての **virsh** コマンドはこの名前を使用します。

- ストレージプールの **タイプ**。NFS ベースのストレージプールの場合、タイプは **netfs** です。
- ステップ1で作成したマウントポイントが存在する **NFS サーバーのホスト名**。ホスト名または IP アドレスを指定することができます。この例では、ホスト名に **localhost** が使われています。
- **ソースパス** は、配信されるファイルの NFS サーバー上の場所です。

この例では、**/home/path/to/mountpoint/directory** ディレクトリーが使われています。

- NFS クライアントがファイルの参照コピーを保管する **ターゲット**。

この例では、ターゲットに **/tmp/nfspool-client** が使われています。

プールを定義せずに XML ファイルの設定結果を表示するには、コマンドに **--print-xml** オプションを追加します。上記のコマンドに **--print-xml** オプションを追加した例を以下に示します。

```
# virsh pool-define-as nfspool netfs --sourcehost localhost --
source-path /home/path/to/mountpoint/directory --target
/tmp/nfspool-client --print-xml
<pool type='netfs'>
  <name>nfspool</name>
  <source>
    <host name='localhost' />
    <dir path='/home/path/to/mountpoint/directory' />
  </source>
  <target>
    <path>/tmp/nfspool-client</path>
  </target>
</pool>
```

## 2. ストレージプールが一覧に表示されることを確認します。

ストレージプールオブジェクトが正しく作成されていること、および状態が **inactive** として表示されていることを確認します。

```
# virsh pool-list --all
Name                               State      Autostart
-----
default                           active     yes
nfspool                            inactive   no
```

**virsh pool-dumpxml** コマンドを実行して、出力を表示することもできます。

```
# virsh pool-dumpxml nfspool
<pool type='netfs'>
  <name>nfspool</name>
  <uuid>ad9bca0f-977f-4fe1-90c6-cb44f676f1ce</uuid>
  <capacity unit='bytes'>0</capacity>
  <allocation unit='bytes'>0</allocation>
  <available unit='bytes'>0</available>
```

```
<source>
  <host name='localhost' />
  <dir path='/home/vm-storage/nfspool' />
  <format type='auto' />
</source>
<target>
  <path>/tmp/nfspool-client</path>
</target>
</pool>
```

### 3. ローカルのディレクトリーを作成します。

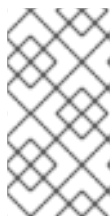
**virsh pool-build** コマンドを使用し、指定したディレクトリー (この例では *nfspool*) にディレクトリーベースのストレージプールをビルドします。

```
# virsh pool-build guest_images
Pool guest_images built
# ls -la /nfspool
total 8
drwx-----.  2 root root 4096 May 30 02:44 .
dr-xr-xr-x. 26 root root 4096 May 30 02:44 ..
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
nfspool                            inactive   no
```

### 4. ストレージプールを起動します。

**virsh pool-start** コマンドを使ってディレクトリーストレージプールを有効にします。これにより、プールのボリュームがゲストのディスクイメージとして使用されます。

```
# virsh pool-start nfspool
Pool nfspool started
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
nfspool                            active     no
```



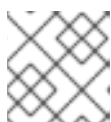
#### 注記

1つの手順で、ストレージプールをビルドして起動することができます。

```
# virsh-pool start nfspool --build
```

### 5. autostart をオンにします。

ストレージプールの **autostart** をオンにします。



#### 注記

このステップは必須ではありません。

**autostart** が設定されていると、**libvirtd** サービスの起動時に、**libvirtd** サービスによりストレージプールが起動します。

```
# virsh pool-autostart nfspool
Pool nfspool marked as autostarted
# virsh pool-list --all
Name                               State      Autostart
-----
default                            active     yes
nfspool                            active     yes
```

#### 6. ストレージプールの設定を確認します。

ストレージプールが正しく作成されたこと、サイズが正しく表示されていること、状態が **running** と表示されていることなどを確認します。プールを永続化させたい場合には、**Persistent** に **yes** が表示されているか確認します。サービスの起動時にプールを自動的に起動させたい場合は、**Autostart** に **yes** が表示されているか確認します。

```
# virsh pool-info nfspool
Name:          nfspool
UUID:          ad9bca0f-977f-4fe1-90c6-cb44f676f1ce
State:         running
Persistent:    yes
Autostart:     yes
Capacity:      123.63 GiB
Allocation:    10.87 GiB
Available:     112.76 GiB

# ls -la /tmp/nfspool-client
total 8
total 4
drwxr-xr-x.  2 root root 4096 Aug 28 15:59 .
drwxrwxrwt. 26 root root  640 Aug 28 16:07 ..
#
```

これで NFS ベースのストレージプールが使用できるようになりました。

#### 13.6.4. virsh を使用したストレージプールの削除

以下では、**virsh** を使ってストレージプールを削除する方法を説明します。

1. 同じプールを使用する他のゲスト仮想マシンとの問題を避けるには、ストレージプールを停止してから使用中のリソースをすべて解放するのが最良の方法です。

```
# virsh pool-destroy nfspool
```

2. 必要に応じて、ストレージプールが存在するディレクトリーを削除します (この操作は必須ではありません)。

```
# virsh pool-delete nfspool
```

3. ストレージプールの定義を削除します。

```
# virsh pool-undefine nfspool
Pool nfspool has been undefined
```

## 13.7. SCSI デバイスでの NPIV 仮想アダプター (vHBA) の使用

NPIV (N\_Port ID Virtualization) とは、単一の物理的なファイバーチャネルホストバスアダプター (HBA) の共有を可能にするソフトウェア技術です。

これにより、複数のゲストから複数物理ホストの同一ストレージを確認することが可能になり、ストレージの移行パスが容易になります。このため、適切なストレージパスが指定されていれば、移行でストレージを作成したりコピーしたりする必要がありません。

仮想化では、仮想ホストバスアダプター (vHBA) が仮想マシンの LUN を制御します。ホストが複数 KVM ゲスト間で 1 つのファイバーチャネルを共有するには、各仮想マシンに vHBA が作成される必要があります。単一の vHBA を複数の KVM ゲストで使用することはできません。

NPIV の各 vHBA は、その親 HBA および独自の World Wide Node Name (WWNN) と World Wide Port Name (WWPN) で識別されます。ストレージへのパスは、WWNN と WWPN の値で決定されます。親 HBA は、**scsi\_host#** または WWNN/WWPN ペアで定義することができます。



### 注記

親 HBA を **scsi\_host#** で定義しハードウェアがホストマシンに追加されると、**scsi\_host#** の割り付けが変更される場合があります。したがって、親 HBA は WWNN/WWPN ペアを使って定義することが推奨されます。

このセクションでは、vHBA を仮想マシンに永続的に設定する方法について説明します。



### 注記

vHBA を作成する前に、ホスト LUN のストレージレイ (SAN) 側のゾーン機能を設定してゲスト間を分離し、データが損傷する可能性を防ぐことをお勧めします。

### 13.7.1. vHBA の作成

#### 手順13.9 vHBA の作成

1. ホストシステム上で HBA を見つけます。  
ホストシステム上で HBA を見つけるには、**virsh nodedev-list --cap vports** コマンドを使用します。

たとえば、以下の出力は、vHBA をサポートする 2 つの HBA を持つホストを示しています。

```
# virsh nodedev-list --cap vports
scsi_host3
scsi_host4
```

2. HBA の詳細を確認します。  
**virsh nodedev-dumpxml HBA\_device** コマンドを使って HBA の詳細を確認します。



**virsh nodedev-dumpxml** コマンドからの XML 出力は、**<name>**、**<wwnn>**、および **<wwpn>** フィールドを一覧表示します。これらのフィールドは vHBA を作成するために使用されます。**<max\_vports>** 値は、サポートされる vHBA の最大数を示します。

```
# virsh nodedev-dumpxml scsi_host3
<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 設定で利用できる 127 の仮想ポートがあることを示しています。**<vports>** 値は、現在使用中の仮想ポートの数を示します。これらの値は、vHBA 作成後に更新されます。

### 3. vHBA ホストデバイスを作成します。

vHBA ホスト用に以下のどちらかのような XML ファイルを作成します (この例では、ファイル名は **vhba\_host3.xml** です)。

この例では、親 vHBA を定義するのに **scsi\_host#** が使われています。

```
# cat vhba_host3.xml
<device>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <capability type='fc_host'>
    </capability>
  </capability>
</device>
```

この例では、親 vHBA を定義するのに WWNN/WWPN ペアが使われています。

```
# cat vhba_host3.xml
<device>
  <name>vhba</name>
  <parent wwnn='20000000c9848140' wwpn='10000000c9848140' />
  <capability type='scsi_host'>
    <capability type='fc_host'>
    </capability>
  </capability>
</device>
```



### 注記

WWNN と WWPN の値は、[ステップ 2](#) で確認した HBA 詳細の値と一致している必要があります。

**<parent>** フィールドでは、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>** タグ内の詳細は、ホスト用に新規の vHBA デバイスを作成するために次のステップで使用します。**nodedev XML** 形式についての詳細は、[libvirt アップストリームページ](#) を参照してください。

#### 4. vHBA ホストデバイス上で新規 vHBA を作成します。

`vhba_host3` 上で vHBA を作成するには、**virsh nodeudev-create** コマンドを使用します。

```
# virsh nodeudev-create vhba_host3.xml
Node device scsi_host5 created from vhba_host3.xml
```

#### 5. vHBA を確認します。

新規 vHBA (`scsi_host5`) の詳細を **virsh nodeudev-dumpxml** コマンドで確認します。

```
# virsh nodeudev-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>
```

### 13.7.2. vHBA を使用したストレージプールの作成

vHBA 設定を保持するには、vHBA に基づく libvirt ストレージプールを定義することが推奨されます。

ストレージプールを使用することには 2 つの主な利点があります。

- libvirt コードにより、virsh コマンド出力から LUN のパスを簡単に特定できます。
- 仮想マシンの移行には、ターゲットマシン上に同じ vHBA 名を持つストレージプールを定義し、起動することのみが必要になります。これを実行するには、vHBA LUN、libvirt ストレージプールおよびボリューム名を仮想マシンの XML 設定に指定する必要があります。例については、「[仮想マシンが vHBA LUN を使用するよう設定する](#)」を参照してください。

#### 1. SCSI ストレージプールを作成します。

vHBA の永続的な設定を作成するには、まず以下の形式を使用して libvirt '**scsi**' ストレージ

プール XML ファイルを作成します。同一の物理 HBA 上のストレージプールを使用する単一の vHBA を作成する場合は、システム上の `/dev/disk/by-{path|id|uuid|label}` の場所のいずれかにするなど、`<path>` 値の安定した場所を使用することをお勧めします。

同一の物理 HBA 上のストレージプールを使用する複数の vHBA を使用する場合は、`<path>` フィールドの値は `/dev/` にしなければなりません。そうでないと、ストレージプールポリシーは vHBA のいずれかのみに表示され、ホストのデバイスを NPIV 設定で複数ゲストに表示することができなくなります。

`<path>` の詳細および `<target>` 内の要素については、<http://libvirt.org/formatstorage.html> を参照してください。

### 例13.3 サンプル SCSI ストレージプール SML 構文

以下の例では、`'scsi'` ストレージプールの名前は `vhbapool_host3.xml` となっています。

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' wwnn='5001a4a93526d0a1'
wwpn='5001a4ace3ee047d' />
  </source>
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <mode>0700</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</pool>
```

または、`vhbapool_host3.xml` が単一 HBA 上にある複数の vHBA の 1 つである場合に以下の構文が使用されます。

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' wwnn='5001a4a93526d0a1'
wwpn='5001a4ace3ee047d' />
  </source>
  <target>
    <path>/dev/</path>
    <permissions>
      <mode>0700</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</pool>
```



## 重要

どちらの場合も、プールには **type='scsi'** を使用し、ソースアダプタータイプは **'fc\_host'** にする必要があります。ホストの再起動時に使用する永続的な設定の場合、**wwnn** および **wwpn** 属性は、**libvirt** により **vHBA** に割り当てられる値である必要があります (この例では **scsi\_host5**)。

オプションで **'parent'** 属性は、親 **scsi\_host** デバイスを **vHBA** として特定するために **<adapter>** フィールドで使用できます。ここで、値は **virsh node-dev-create** で作成される **vHBA** の **scsi\_host** ではなく、その **vHBA** の親になることに注意してください。

**'parent'** 属性を指定することは、プール定義の重複の有無を確認する際にも役立ちます。これは、**'fc\_host'** および **'scsi\_host'** ソースアダプタープールの両方が使用されている場合により重要になります。これにより、新規の定義が別の既存ストレージプールの同一の **scsi\_host** を使用して重複していないことを確認できます。

以下の例は、ストレージプール設定の **<adapter>** フィールドで使用されるオプションの **'parent'** 属性を示しています。

```
<adapter type='fc_host' parent='scsi_host3' wwnn='5001a4a93526d0a1'
wwpn='5001a4ace3ee047d' />
```

### 2. プールを定義します。

ストレージプール (この例では **vhbapool\_host3** という名前) を永続的に定義するには、**virsh pool-define** コマンドを使用します。

```
# virsh pool-define vhbapool_host3.xml
Pool vhbapool_host3 defined from vhbapool_host3.xml
```

### 3. プールを起動します。

以下のコマンドでストレージプールを起動します。

```
# virsh pool-start vhbapool_host3
Pool vhbapool_host3 started
```



## 注記

プールを起動する際に、**libvirt** は同じ **wwpn:wwnn** 識別子を持つ **vHBA** がすでに存在するかどうかを検査します。存在しない場合は、指定した **wwpn:wwnn** の新規 **vHBA** が作成されます。同様に、プールを破棄する際には、**libvirt** は同じ **wwpn:wwnn** の値を使用する **vHBA** も破棄します。

### 4. autostart を有効にします。

最後に、この後のホストの再起動で仮想マシンで使用される **vHBA** が自動的に定義されるようにするために、ストレージプールの **autostart** 機能を設定します (この例では、プールの名前は **vhbapool\_host3**)。

```
# virsh pool-autostart vhbapool_host3
```

## 13.7.3. 仮想マシンが **vHBA LUN** を使用するよう設定する

vHBA のストレージプールを作成したら、仮想マシンの XML に仮想マシンのディスクボリュームを作成することにより、vHBA LUN を仮想マシンの設定に追加します。たとえば、以下の例のようにストレージの **pool** および **volume** を **<source>** パラメーターに指定します。

```
<disk type='volume' device='disk'>
  <driver name='qemu' type='raw' />
  <source pool='vhbapool_host3' volume='unit:0:4:0' />
  <target dev='hda' bus='ide' />
</disk>
```

**disk** ではなく **lun** デバイスを指定するには、以下の例を参照してください。

```
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw' />
  <source pool='vhbapool_host3' volume='unit:0:4:0' mode='host' />
  <target dev='sda' bus='scsi' />
  <shareable />
</disk>
```

SCSI LUN ベースのストレージをゲストに追加する際の XML 設定のサンプルについては、「[SCSI LUN ベースのストレージのゲストへの追加](#)」を参照してください。

ハードウェア故障の際に LUN への再接続を確実にするには、**fast\_io\_fail\_tmo** および **dev\_loss\_tmo** オプションを編集することが推奨されます。詳細については、「[ハードウェア故障後に公開されている LUN に再接続](#)」を参照してください。

#### 13.7.4. vHBA ストレージプールの破棄

ストレージプールで作成された vHBA は、**virsh pool-destroy** コマンドで破棄することができます。

```
# virsh pool-destroy vhbapool_host3
```

**virsh pool-destroy** コマンドを実行すると、「[vHBA の作成](#)」で作成された vHBA が削除されることに注意してください。

プールと vHBA が破棄されたことを確認するには、以下のコマンドを実行します。

```
# virsh nodedev-list --cap scsi_host
```

**scsi\_host5** はリストに表示されなくなります。

### 13.8. GLUSTERFS ストレージプール

このセクションでは、GlusterFS ベースのストレージプールを有効にする方法について記載しています。Red Hat Enterprise Linux 6.5 には、GlusterFS を使った仮想マシン作成のネイティブサポートがあります。GlusterFS は FUSE を使用するユーザー領域のファイルシステムです。これがゲスト仮想マシンで有効にされると、KVM ホスト物理マシンは 1 つ以上の GlusterFS ストレージボリュームからゲスト仮想マシンのイメージを起動し、GlusterFS ストレージボリュームのイメージをゲスト仮想マシンのデータディスクとして使用できます。

#### 13.8.1. virsh を使用した GlusterFS ストレージプールの作成

このセクションでは、Gluster サーバーとアクティブな Gluster ボリュームを準備する方法について説明します。

### 手順13.10 Gluster サーバーとアクティブな Gluster ボリュームの準備

1. 次のコマンドを使って、状態を一覧表示し、Gluster サーバーの IP アドレスを取得します。

```
# gluster volume status
Status of volume: gluster-vol1
Gluster process      Port Online Pid
-----
-----
Brick 222.111.222.111:/gluster-vol1    49155 Y 18634

Task Status of Volume gluster-vol1
-----
-----
There are no active volume tasks
```

2. まだこれを実行していない場合は、**glusterfs-fuse** をインストールしてから **virt\_use\_fusefs** を有効にします。次に、以下のコマンドを実行して、Gluster サーバーに接続する 1 つのホストを用意します。

```
# setsebool virt_use_fusefs on
# getsebool virt_use_fusefs
virt_use_fusefs --> on
```

3. **pool type** を **gluster** に指定し、Gluster ストレージプール (以下の例では **glusterfs-pool.xml**) を設定するために新しい XML ファイルを作成し、以下のデータを追加します。

```
<pool type='gluster'>
  <name>glusterfs-pool</name>
  <source>
    <host name='111.222.111.222' />
    <dir path='/' />
    <name>gluster-vol1</name>
  </source>
</pool>
```

#### 図13.24 GlusterFS XML ファイルの内容

4. 次のコマンドを使って、Gluster プールを定義し、これを開始します。

```
# virsh pool-define glusterfs-pool.xml
Pool gluster-pool defined from glusterfs-pool.xml

# virsh pool-list --all
Name                State      Autostart
-----
gluster-pool        inactive   no

# virsh pool-start gluster-pool
```

```
Pool gluster-pool started

# virsh pool-list --all
Name                               State      Autostart
-----
gluster-pool                       active     no

# virsh vol-list gluster-pool
Name                               Path
-----
qcow2.img                         gluster://111.222.111.222/gluster-
vol1/qcow2.img
raw.img                           gluster://111.222.111.222/gluster-vol1/raw.img
```

### 13.8.2. virsh を使用した GlusterFS ストレージプールの削除

このセクションでは、`virsh` を使ってストレージプールを削除する方法について詳しく説明します。

#### 手順13.11 GlusterFS ストレージプールの削除

1. 次のコマンドを使って、ストレージプールの状態を非アクティブに設定します。

```
# virsh pool-destroy gluster-pool
Pool gluster-pool destroyed
```

2. 次のコマンドを使って、プールが非アクティブの状態であることを確認します。

```
# virsh pool-list --all
Name                               State      Autostart
-----
gluster-pool                       inactive   no
```

3. 次のコマンドを使って GlusterFS ストレージプールの定義を解除します。

```
# virsh pool-undefine gluster-pool
Pool gluster-pool has been undefined
```

4. 次のコマンドを使ってプールの定義が解除されていることを確認します。

```
# virsh pool-list --all
Name                               State      Autostart
-----
```

## 第14章 ストレージボリューム

### 14.1. はじめに

ストレージプールは複数のストレージボリュームに分割されます。ストレージボリュームは、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、および libvirt で処理されるその他のストレージタイプを抽象化したものです。ストレージボリュームは、基礎となるハードウェアに関係なく、ローカルストレージデバイスとしてゲスト仮想マシンに提示されます。以下のセクションには、**virsh** が許可するすべてのコマンドと引数が含まれている訳ではないことに注意してください。詳細は、「[ストレージボリュームコマンド](#)」を参照してください。

#### 14.1.1. ボリュームの参照

追加のパラメーターおよび引数については、「[ボリューム情報の一覧表示](#)」を参照してください。

特定のボリュームを参照する方法として、以下の 3 つのアプローチを使用することができます。

##### ボリュームとストレージプールの名前

ボリュームは、ボリュームが属しているストレージプールの ID とボリューム名を使って参照します。**virsh** コマンドラインの場合、**--pool storage\_pool volume\_name** の形式が取られます。

たとえば、名前が *firstimage* というボリュームが *guest\_images* プールにあるとします。

```
# virsh vol-info --pool guest_images firstimage
Name:          firstimage
Type:          block
Capacity:      20.00 GB
Allocation:    20.00 GB

virsh #
```

##### ホスト物理システム上のストレージへの完全パス

ボリュームは、ファイルシステム上の完全パスを使って参照することもできます。このアプローチを選択する場合、プールの ID を含める必要はありません。

たとえば、ホスト物理マシンシステムに */images/secondimage.img* として表示される *secondimage.img* という名前のボリュームの場合、このイメージは */images/secondimage.img* としても参照されます。

```
# virsh vol-info /images/secondimage.img
Name:          secondimage.img
Type:          file
Capacity:      20.00 GB
Allocation:    136.00 kB
```

##### 固有のボリュームキー

仮想化システムでボリュームをはじめて作成すると、固有の ID が生成され、ボリュームに割り当てられます。この固有 ID は **ボリュームキー** と呼ばれます。このボリュームキーの形式は使用するストレージによって異なります。

LVM などのブロックベースのストレージで 사용되는場合、ボリュームキーは以下のような形式になります。



```
c3pKz4-qPvC-Xf7M-7WNM-WJc8-qSiz-mtvpGn
```

ファイルベースのストレージで使用する場合には、ボリュームキーはボリュームストレージへの完全パスのコピーになります。

```
/images/secondimage.img
```

たとえば *Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr* のボリュームキーを持つボリュームは、以下のようになります。

```
# virsh vol-info Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
Name:          firstimage
Type:          block
Capacity:      20.00 GB
Allocation:    20.00 GB
```

**virsh** は、ボリューム名、ボリュームパス、またはボリュームキーの間で変換するためのコマンドを提供します。

#### vol-name

ボリュームパスまたはボリュームキーを指定するとボリューム名を返します。

```
# virsh vol-name /dev/guest_images/firstimage
firstimage
# virsh vol-name Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
```

#### vol-path

ボリュームキー、またはストレージプール ID およびボリューム名を指定するとボリュームパスを返します。

```
# virsh vol-path Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
/dev/guest_images/firstimage
# virsh vol-path --pool guest_images firstimage
/dev/guest_images/firstimage
```

#### vol-key コマンド

ボリュームパスまたはストレージプール ID とボリューム名を指定するとボリュームキーを返します。

```
# virsh vol-key /dev/guest_images/firstimage
Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
# virsh vol-key --pool guest_images firstimage
Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
```

詳細は、「[ボリューム情報の一覧表示](#)」を参照してください。

## 14.2. ボリュームの作成

このセクションでは、ブロックベースのストレージプール内にディスクボリュームを作成する方法に

ついて紹介しています。以下の例では、**virsh vol-create-as** コマンドを使って *guest\_images\_disk* ストレージプール内に **GB** 単位のサイズを指定したストレージボリュームを作成します。このコマンドはボリュームの必要に応じて繰り返し実行する必要があるため、この例では **3** つのボリュームを作成します。追加のパラメーターおよび引数については、「[ストレージボリュームの作成](#)」を参照してください。

```
# virsh vol-create-as guest_images_disk volume1 8G
Vol volume1 created

# virsh vol-create-as guest_images_disk volume2 8G
Vol volume2 created

# virsh vol-create-as guest_images_disk volume3 8G
Vol volume3 created

# virsh vol-list guest_images_disk
Name                               Path
-----
volume1                           /dev/sdb1
volume2                           /dev/sdb2
volume3                           /dev/sdb3

# parted -s /dev/sdb print
Model: ATA ST3500418AS (scsi)
Disk /dev/sdb: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt

Number  Start   End     Size    File system  Name      Flags
  2      17.4kB  8590MB  8590MB                     primary
  3      8590MB  17.2GB  8590MB                     primary
  1      21.5GB  30.1GB  8590MB                     primary
```

### 14.3. ボリュームのクローン作成

新しいボリュームは、クローン元となるボリュームと同じストレージプール内のストレージから割り当てられます。**virsh vol-clone** は、**--pool** 引数を備えている必要があり、これがクローン作成するボリュームが含まれるストレージプールの名前を決定します。コマンドの残りの部分は、これからクローン作成されるボリューム (*volume3*) とすでにクローン作成された新規ボリューム (*clone1*) の名前を付けます。**virsh vol-list** コマンドはストレージプール (*guest\_images\_disk*) にあるボリュームを一覧表示します。追加のコマンドおよび引数については、「[ストレージボリュームのクローン作成](#)」を参照してください。

```
# virsh vol-clone --pool guest_images_disk volume3 clone1
Vol clone1 cloned from volume3

# virsh vol-list guest_images_disk
Name                               Path
-----
volume1                           /dev/sdb1
volume2                           /dev/sdb2
volume3                           /dev/sdb3
clone1                            /dev/sdb4
```

```
# parted -s /dev/sdb print
Model: ATA ST3500418AS (scsi)
Disk /dev/sdb: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
```

Number	Start	End	Size	File system	Name	Flags
1	4211MB	12.8GB	8595MB	primary		
2	12.8GB	21.4GB	8595MB	primary		
3	21.4GB	30.0GB	8595MB	primary		
4	30.0GB	38.6GB	8595MB	primary		

## 14.4. ボリュームの削除と消去

ボリュームの削除および消去に必要な `virsh` コマンドの詳細は、「[ストレージボリュームの削除](#)」を参照してください。

## 14.5. ストレージデバイスのゲストへの追加

このセクションでは、ゲストにストレージデバイスを追加する方法を説明しています。必要に応じてストレージのみを追加することができます。このセクションでは、以下のタイプのストレージについて説明します。

- ファイルベースのストレージ。「[ファイルベースのストレージのゲストへの追加](#)」を参照してください。
- ブロックデバイス (CD-ROM、DVD およびフロッピーディスクを含む)。「[ハードドライブと他のブロックデバイスのゲストへの追加](#)」を参照してください。
- SCSI コントローラーおよびデバイス。お使いのホスト物理マシンでこれに対応できる場合、最大100のSCSI コントローラーを任意のゲスト仮想マシンに追加することができます。「[ゲスト仮想マシンのストレージコントローラーの管理](#)」を参照してください。

### 14.5.1. ファイルベースのストレージのゲストへの追加

ファイルベースのストレージは、ホスト物理マシンのファイルシステム上に格納されるファイルの集合であり、ゲストに対して仮想化されたハードドライブとして動作します。ファイルベースのストレージを追加するには、次の手順を実行します。

#### 手順14.1 ファイルベースのストレージの追加

1. ストレージファイルを作成するか、または既存のファイルを使用します (IMG ファイルなど)。以下のコマンドでは、どちらもゲスト用の追加ストレージとして使用できる 4GB のファイルを作成します。
  - ファイルベースのストレージイメージには、事前割り当てファイルを使用することをお勧めします。以下の `dd` コマンドを使って事前割り当てファイルを作成します。

```
# dd if=/dev/zero of=/var/lib/libvirt/images/FileName.img bs=1G
count=4
```

- または、事前割り当てファイルの代わりにスパースファイルを作成することもできます。スパースファイルの方が短い時間で作成できるため、テストの目的で使用することに適しています。ただし、データ整合性またはパフォーマンス関連の問題があるため、実稼働環

境に使用することはお勧めしません。

```
# dd if=/dev/zero of=/var/lib/libvirt/images/FileName.img bs=1G
seek=4096 count=4
```

2. 新しいファイル内に **<disk>** 要素を記述して追加のストレージを作成します。この例では、このファイル名は **NewStorage.xml** になります。

**<disk>** 要素は、ディスクのソースと仮想ブロックデバイスのデバイス名を記述します。デバイス名はゲスト内のすべてのデバイスの中で固有の名前にするようにしてください。このデバイス名によって、ゲストが仮想ブロックデバイスを検索する際に使用するバスが指定されます。次の例では、**FileName.img** という名前のファイルベースのストレージコンテナをソースとする **virtio** ブロックデバイスを定義しています。

```
<disk type='file' device='disk'>
  <driver name='qemu' type='raw' cache='none' />
  <source file='/var/lib/libvirt/images/FileName.img' />
  <target dev='vdb' />
</disk>
```

デバイス名は、IDE や SCSI ディスクを指定する「**hd**」や「**sd**」で始まる名前にすることができます。設定ファイルには **<address>** サブ要素も組み込むことができ、バス上に新しいデバイスの位置を指定することができます。**virtio** ブロックデバイスの場合、これは **PCI** アドレスになるはずです。**<address>** サブ要素を省略すると、**livbirt** により次に使用できる **PCI** スロットの検索および割り当てが行われます。

3. 以下のようにして **CD-ROM** を接続します。

```
<disk type='file' device='cdrom'>
  <driver name='qemu' type='raw' cache='none' />
  <source file='/var/lib/libvirt/images/FileName.img' />
  <readonly />
  <target dev='hdc' />
</disk >
```

4. **NewStorage.xml** 内で定義されているデバイスをゲスト (**Guest1**) に追加します。

```
# virsh attach-device --config Guest1 ~/NewStorage.xml
```



### 注記

この変更は、ゲストが破棄され、再起動された後にのみ適用されます。また、永続デバイスが追加できるのは永続ドメインに対してのみになります。永続ドメインとは、ドメインの設定を **virsh define** コマンドを使って保存したドメインを指します。

ゲストが実行中の場合で、そのゲストが破棄されるまでの間に新しいデバイスを一時的に追加する場合は **--config** オプションを省略します。

```
# virsh attach-device Guest1 ~/NewStorage.xml
```



## 注記

**virsh** コマンドでは **attach-disk** コマンドを使用することができます。このコマンドでは、より簡単な構文で限られた数のパラメーターを設定でき、XML ファイルを作成する必要がありません。以下に示すように、**attach-disk** コマンドは前述の **attach-device** コマンドと同じように使用できます。

```
# virsh attach-disk Guest1
/var/lib/libvirt/images/FileName.img vdb --cache none
```

**virsh attach-disk** コマンドでも **--config** オプションを使用できる点に注意してください。

### 5. ゲストマシンを起動します (まだ稼働していない場合):

```
# virsh start Guest1
```



## 注記

以下は Linux ゲストに固有のステップになります。他のオペレーティングシステムは、複数の異なる方法で新規のストレージデバイスを処理します。他のシステムについては、該当するオペレーティングシステムのドキュメントを参照してください。

### 6. ディスクドライブのパーティション設定

これでゲストは **/dev/vdb** というハードディスクデバイスを持っていることになります。必要であれば、このディスクドライブにパーティションを設定し、フォーマットします。追加したデバイスが表示されない場合は、ゲストのオペレーティングシステムにディスクのホットプラグに関する問題が発生していることを示します。

- a. 新規デバイスに対して **fdisk** を開始します。

```
# fdisk /dev/vdb
Command (m for help):
```

- b. 新規パーティションを作成するために **n** を入力します。

- c. 次のように出力されます。

```
Command action
e   extended
p   primary partition (1-4)
```

プライマリーパーティションを作成するために **p** を入力します。

- d. 使用できるパーティション番号を選択します。この例では、**1** が入力され 1 番目のパーティションが選択されています。

```
Partition number (1-4): 1
```

- e. **Enter** を押して、デフォルトとなる 1 番目のシリンダーを入力します。

```
First cylinder (1-400, default 1):
```

- f. パーティションのサイズを選択します。この例では、**Enter** が押されてディスク全体が割り当てられています。

```
Last cylinder or +size or +sizeM or +sizeK (2-400, default 400):
```

- g. **t** を入力して、パーティションタイプを設定します。

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

- h. 直前のステップで作成したパーティションを選択します。この例ではパーティション番号が **1** のパーティションを選択しています。この例で作成されたパーティションは1つだけであるため、**fdisk** によってパーティション1が自動的に選択されています。

```
Partition number (1-4): 1
```

- i. Linux パーティションを作成するために **83** を入力します。

```
Hex code (type L to list codes): 83
```

- j. **w** を入力して、変更を書き込み、終了します。

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

- k. **ext3** ファイルシステムで新しいパーティションをフォーマットします。

```
# mke2fs -j /dev/vdb1
```

7. マウントディレクトリを作成して、ゲスト上にディスクをマウントします。この例では、ディレクトリは **myfiles** に置かれています。

```
# mkdir /myfiles
# mount /dev/vdb1 /myfiles
```

これで、ゲストは仮想化されたファイルベースの追加ストレージデバイスを持つことになります。ただし、このストレージはゲストの **/etc/fstab** ファイル内で定義しない限り、再起動後も永続的にマウントされることはない点に注意してください。

```
/dev/vdb1    /myfiles    ext3        defaults    0 0
```

### 14.5.2. ハードドライブと他のブロックデバイスのゲストへの追加

システム管理者は、ゲスト用にストレージ領域を拡張したり、システムデータをユーザーデータから分離したりするために追加のハードドライブを使用するオプションを選択できます。

#### 手順14.2 物理ブロックデバイスのゲストへの追加

1. この手順は、ホスト物理マシン上のハードドライブをゲストに追加する方法を説明しています。これは、**CD-ROM**、**DVD**、およびフロッピーデバイスを含むすべての物理ブロックデバイスに適用されます。

ホスト物理マシンにハードディスクデバイスを物理的に割り当てます。ドライブにデフォルトでアクセスできない場合に、ホスト物理マシンを設定します。

2. 次のいずれかを行います。

- a. 新しいファイル内に **disk** 要素を記述して追加のストレージを作成します。この例では、このファイル名は **NewStorage.xml** です。次の例は、ホスト物理マシンのパーティション **/dev/sr0**: 用のデバイススペースの追加ストレージコンテナが含まれる設定ファイルのセクションになります。

```
<disk type='block' device='disk'>
  <driver name='qemu' type='raw' cache='none' />
  <source dev='/dev/sr0' />
  <target dev='vdc' bus='virtio' />
</disk>
```

- b. 直前のセクションの指示に従って、デバイスをゲスト仮想マシンに割り当てます。または、以下のように **virsh attach-disk** コマンドを使用することもできます。

```
# virsh attach-disk Guest1 /dev/sr0 vdc
```

次のオプションを選択できることに注意してください。

- 以下のように、**virsh attach-disk** コマンドも **--config**、**--type**、および **--mode** オプションを受け入れます。

```
# virsh attach-disk Guest1 /dev/sr0 vdc --config --type cdrom
--mode readonly
```

- または、デバイスがハードドライブの場合には、**--type** で **--type disk** を使用することもできます。

3. これでゲスト仮想マシンは、Linux の場合は **/dev/vdc** という名前 (ゲスト仮想マシン OS の選択によって異なるがこれに類する名前) の新しいハードディスクデバイスを持つことになり、ゲスト仮想マシンのオペレーティングシステムに適した標準的な手順に従ってゲスト仮想マシンからディスクを初期化できるようになります。具体例については、[手順14.1「ファイルベースのストレージの追加」](#)を参照してください。



#### 警告

ブロックデバイスをゲストに追加する際には、『[Red Hat Enterprise Linux 7 仮想化セキュリティガイド](#)』に書かれた関連するセキュリティ上の考慮事項に従うようにしてください。

### 14.5.3. SCSI LUN ベースのストレージのゲストへの追加

ホストの SCSI LUN デバイスは、ホストの設定に応じて 3 つのメカニズムを使用してゲストに完全に公開できます。このように SCSI LUN デバイスを公開することで、SCSI コマンドをゲスト上で LUN に対して直接実行できるようになります。これは、ファイバーチャネルストレージをホスト間で共有できると共に、LUN をゲスト間で共有するための役に立つ方法になります。



## 重要

オプションの **sgio** 属性は、特権のない SCSI Generical I/O (SG\_IO) コマンドが **device='lun'** ディスクについてフィルターされるかどうかを制御します。**sgio** 属性は **'filtered'** または **'unfiltered'** として指定できますが、SG\_IOioctl コマンドを永続性保存の状態でゲスト上で渡せるようにするには **'unfiltered'** に設定する必要があります。

**sgio='unfiltered'** の設定のほかにも、ゲスト間で LUN を共有するために **<shareable>** 要素を設定する必要があります。**sgio** 属性は、指定がない場合はデフォルトで **'filtered'** に設定されます。

**<disk>** XML 属性 **device='lun'** は、以下のゲストディスク設定について有効です。

- **type='block'** を **<source dev='/dev/disk/by-{path|id|uuid|label}' />** に対して指定します。

```
<disk type='block' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw' />
  <source dev='/dev/disk/by-path/pci-0000\:04\:00.1-fc-
0x203400a0b85ad1d7-lun-0' />
  <target dev='sda' bus='scsi' />
  <shareable />
</disk>
```



## 注記

**<source>** デバイス名のコロンの前にはバックスラッシュが必要なことに注意してください。

- **type='network'** を **<source protocol='iscsi'... />** に対して指定します。

```
<disk type='network' device='disk' sgio='unfiltered'>
  <driver name='qemu' type='raw' />
  <source protocol='iscsi' name='iqn.2013-07.com.example:iscsi-
net-pool/1'>
    <host name='example.com' port='3260' />
  </source>
  <auth username='myuser'>
    <secret type='iscsi' usage='libvirtiscsi' />
  </auth>
  <target dev='sda' bus='scsi' />
  <shareable />
</disk>
```

- **type='volume'** を、iSCSI または NPIV/vHBA ソースプールを SCSI ソースプールとして使用する場合に指定します。



以下のサンプル XML は、ゲストで iSCSI ソースプール (*iscsi-net-pool* という名前) を SCSI ソースプールとして使用していることを示しています。

```
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw' />
  <source pool='iscsi-net-pool' volume='unit:0:0:1'
mode='host' />
  <target dev='sda' bus='scsi' />
  <shareable />
</disk>
```

### 注記

**<source>** タグ内で **mode=** オプションは任意ですが、使用する場合には **'direct'** ではなく **'host'** に設定する必要があります。**'host'** に設定される場合、**libvirt** はローカルホスト上のデバイスへのパスを検索します。**'direct'** に設定される場合、**libvirt** はソースプールのソースホストデータを使用してデバイスへのパスを生成します。

上記のサンプルの iSCSI プール (*iscsi-net-pool*) には、以下のような設定があります。

```
# virsh pool-dumpxml iscsi-net-pool
<pool type='iscsi'>
  <name>iscsi-net-pool</name>
  <capacity unit='bytes'>11274289152</capacity>
  <allocation unit='bytes'>11274289152</allocation>
  <available unit='bytes'>0</available>
  <source>
    <host name='192.168.122.1' port='3260' />
    <device path='iqn.2013-12.com.example:iscsi-chap-netpool' />
    <auth type='chap' username='redhat'>
      <secret usage='libvirtiscsi' />
    </auth>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <mode>0755</mode>
    </permissions>
  </target>
</pool>
```

iSCSI ソースプールで利用可能な LUN の詳細を確認するには、以下のコマンドを入力します。

```
# virsh vol-list iscsi-net-pool
Name                                Path
-----
unit:0:0:1                          /dev/disk/by-path/ip-192.168.122.1:3260-iscsi-
iqn.2013-12.com.example:iscsi-chap-netpool-lun-1
unit:0:0:2                          /dev/disk/by-path/ip-192.168.122.1:3260-iscsi-
iqn.2013-12.com.example:iscsi-chap-netpool-lun-2
```

- **type='volume'** を、NPIV/vHBA ソースプールを SCSI ソースプールとして使用する場合に指定します。

以下のサンプル XML は、ゲストで NPIV/vHBA ソースプール (*vhbapool\_host3* という名前) を SCSI ソースプールとして使用していることを示しています。

```
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw' />
  <source pool='vhbapool_host3' volume='unit:0:1:0' />
  <target dev='sda' bus='scsi' />
  <shareable />
</disk>
```

上記のサンプルの NPIV/vHBA プール (*vhbapool\_host3*) には、以下のような設定があります。

```
# virsh pool-dumpxml vhbapool_host3
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <capacity unit='bytes'>0</capacity>
  <allocation unit='bytes'>0</allocation>
  <available unit='bytes'>0</available>
  <source>
    <adapter type='fc_host' parent='scsi_host3' managed='yes'
wwnn='5001a4a93526d0a1' wwpn='5001a4ace3ee045d' />
  </source>
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <mode>0700</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</pool>
```

vHBA で利用可能な LUN の詳細を確認するには、以下のコマンドを入力します。

```
# virsh vol-list vhbapool_host3
Name                                Path
-----
unit:0:0:0                          /dev/disk/by-path/pci-0000:10:00.0-fc-
0x5006016044602198-lun-0
unit:0:1:0                          /dev/disk/by-path/pci-0000:10:00.0-fc-
0x5006016844602198-lun-0
```

SCSI デバイスで NPIV vHBA を使用方法についての詳細は、「[仮想マシンが vHBA LUN を使用するよう設定する](#)」を参照してください。

以下の手順は、SCSI LUN ベースのストレージデバイスをゲストに追加する例を示しています。上記の **<disk device='lun'>** ゲストのディスク設定のいずれもこの方法で割り当てることができます。ご使用の環境に応じて設定を置き換えてください。

#### 手順14.3 SCSI LUN ベースのストレージのゲストへの割り当て

1. 新しいファイルに `<disk>` 要素を書き込むことによってデバイスファイルを作成し、このファイルを XML 拡張子を付けて保存します (例: `sda.xml`)。

```
# cat sda.xml
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw' />
  <source pool='vhbapool_host3' volume='unit:0:1:0' />
  <target dev='sda' bus='scsi' />
  <shareable />
</disk>
```

2. `sda.xml` で作成したデバイスをゲスト仮想マシン (例: `Guest1`) に関連付けます。

```
# virsh attach-device --config Guest1 ~/sda.xml
```



### 注記

**virsh attach-device** コマンドを **--config** オプションを指定して実行するには、デバイスをゲストに永続的に追加するためにゲストを再起動する必要があります。または、**--persistent** オプションを **--config** の代わりに使用することができます。これを、デバイスをゲストにホットプラグするために使用することもできます。

別の方法として、SCSI LUN ベースのストレージを **virt-manager** を使用してゲスト上で割り当てるか、または設定することができます。**virt-manager** を使用してこれを設定するには、**ハードウェアを追加** ボタンをクリックして、必要なパラメーターと共に仮想ディスクを追加するか、またはこのウィンドウから既存の SCSI LUN デバイスの設定を変更します。Red Hat Enterprise Linux 7.2 以降では、SGIO の値も **virt-manager** で設定できます。

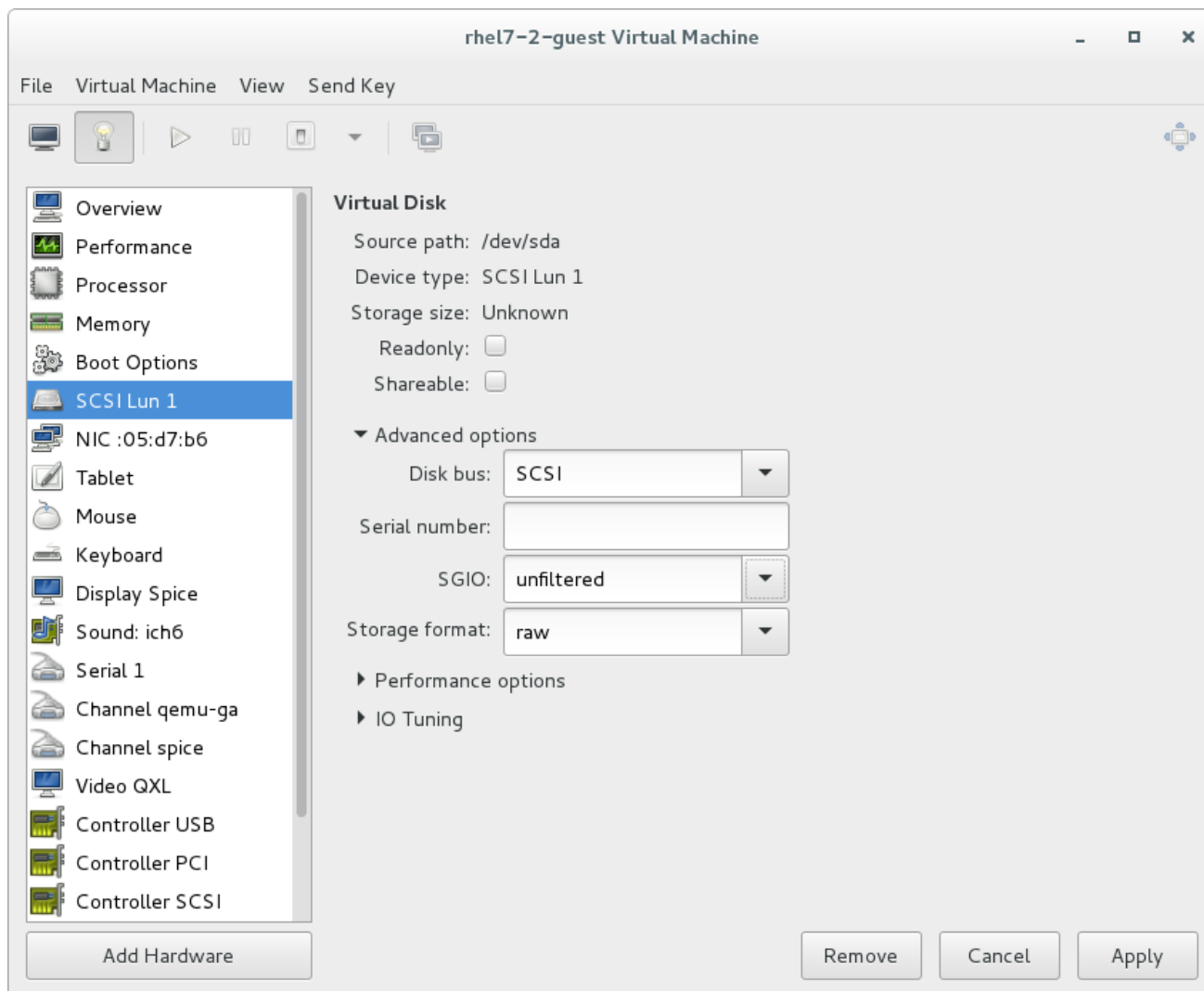


図14.1 virt-manager を使用した SCSI LUN ストレージの設定

### ハードウェア故障後に公開されている LUN に再接続

ハードウェア (ホストのバスアダプター等) の故障により公開されているファイバーチャネル (FC) LUN への接続が失われた場合、ハードウェアの故障が修復された後でも、ゲストの公開されている LUN は故障したままのように見えることがあります。これを防ぐためには、**dev\_loss\_tmo** および **fast\_io\_fail\_tmo** カーネルオプションを編集します。

- **dev\_loss\_tmo** は、SCSI デバイスが故障してから SCSI レイヤーが故障と識別するまでの時間をコントロールします。タイムアウトを防ぐには、オプションを最大値の **2147483647** に設定することをお勧めします。
- **fast\_io\_fail\_tmo** は、SCSI デバイスが故障してから SCSI レイヤーが I/O にフェイルバックするまでの時間をコントロールします。**dev\_loss\_tmo** がカーネルによって無視されないようにするには、このオプションの値を **dev\_loss\_tmo** の値より小さい数値に設定します。

**dev\_loss\_tmo** および **fast\_io\_fail** の値を変更するには、以下に示すどちらかの操作を行います。

- **/etc/multipath.conf** ファイルを編集し、**defaults** セクションの値を設定する。

```
defaults {
    ...
    fast_io_fail_tmo    20
    dev_loss_tmo       infinity
}
```

- FC ホストまたはリモートポートのレベルで **dev\_loss\_tmo** および **fast\_io\_fail** を設定する。例を以下に示します。

```
# echo 20 >
/sys/devices/pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-
0/fc_remote_ports/rport-1:0-0/fast_io_fail_tmo
# echo 2147483647 >
/sys/devices/pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-
0/fc_remote_ports/rport-1:0-0/dev_loss_tmo
```

新しい **dev\_loss\_tmo** および **fast\_io\_fail** の値が有効であることを確認するには、以下のコマンドを使います。

```
# find /sys -name dev_loss_tmo -print -exec cat {} \;
```

パラメーターが正しく設定されていれば、出力は次のようになります (ただし、**pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-0/fc\_remote\_ports/rport-1:0-0** には実際のデバイスが表示されます)。

```
# find /sys -name dev_loss_tmo -print -exec cat {} \;
...
/sys/devices/pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-
0/fc_remote_ports/rport-1:0-0/dev_loss_tmo
2147483647
...
```

#### 14.5.4. ゲスト仮想マシンのストレージコントローラーの管理

virtio ディスクとは異なり、SCSI デバイスでは、コントローラーがゲスト仮想マシンに存在する必要があります。このセクションでは、仮想 SCSI コントローラー (「ホストバスアダプター」または HBA とも知られる) を作成し、SCSI ストレージをゲスト仮想マシンに追加するための必要なステップを詳しく説明します。

##### 手順14.4 仮想 SCSI コントローラーの作成

1. ゲスト仮想マシン (**Guest1**) の設定を表示して、すでに存在している SCSI コントローラーを探します。

```
# virsh dumpxml Guest1 | grep controller.*scsi
```

デバイスコントローラーが存在する場合は、このコマンドは以下のように1つ以上の行を出力します。

```
<controller type='scsi' model='virtio-scsi' index='0' />
```

2. 直前のステップでデバイスコントローラーが表示されない場合、新しいファイルでその 1 つを記述し、以下のステップを実行してそれを仮想マシンに追加します。
  - a. 新しいファイルに **<controller>** 要素を書き込むことによってデバイスコントローラーを作成し、このファイルを XML 拡張子を付けて保存します。たとえば、**NewHBA.xml** のようになります。

```
<controller type='scsi' model='virtio-scsi' />
```

- b. **virtio-scsi-controller.xml** で作成したばかりのデバイスコントローラーを使用中のゲスト仮想マシン (例: **Guest1**) に関連付けます。

```
# virsh attach-device --config Guest1 ~/virtio-scsi-controller.xml
```

この例では、**--config** オプションはディスク接続の場合と同様の動作をします。詳細は、[手順14.2「物理ブロックデバイスのゲストへの追加」](#)を参照してください。

3. 新規 **SCSI** ディスクまたは **CD-ROM** を追加します。新規ディスクは、「[ファイルベースのストレージのゲストへの追加](#)」および「[ハードドライブと他のブロックデバイスのゲストへの追加](#)」セクションにある方法を使用して追加することができます。**SCSI** ディスクを作成するには、**sd** で始まるターゲットデバイス名を指定します。各コントローラーでは **1024** までの **virtio-scsi** ディスクがサポートされますが、より少ない数のディスクで (ファイル記述子などの) ホスト内で利用可能な他のすべてのリソースが消費される可能性もあります。

詳細については、次の Red Hat Enterprise Linux 6 ホワイトペーパーを参照してください: [The next-generation storage interface for the Red Hat Enterprise Linux Kernel Virtual Machine: virtio-scsi](#)

```
# virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.img sdb  
--cache none
```

ゲスト仮想マシン内のドライバーのバージョンによっては、実行中のゲスト仮想マシンを実行しても、新しいディスクをすぐには検出できない場合があります。『[Red Hat Enterprise Linux 7 ストレージ管理ガイド](#)』のステップに従ってください。

## 第15章 QEMU-IMG の使用

**qemu-img** は、KVM で使用される各種ファイルシステムのフォーマット、変更および確認を行うために使用するコマンドラインツールです。**qemu-img** のオプションとその使い方を以下に示します。



### 警告

実行中の仮想マシンまたはその他のプロセスで使用中のイメージを修正するために **qemu-img** を使用することはできません。これにより、イメージが破損する可能性があります。別のプロセスで修正中のイメージの照会により、不整合な状態が生じる可能性があります。

### 15.1. ディスクイメージの検査

ファイル名が *imgname* のディスクイメージで整合性チェックを行います。

```
# qemu-img check [-f format] imgname
```



### 注記

選択されたグループの形式のみが整合性チェックに対応しています。これらには、*qcow2*、*vdi*、*vhd*、*vmdk*、および *qed* が含まれます。

### 15.2. 変更のイメージへのコミット

**qemu-img commit** コマンドを使って、指定イメージファイル (*imgname*) に記録される変更をそのファイルのベースイメージにコミットします。オプションでファイルの形式を指定できます (*fmt*)。

```
# qemu-img commit [-f fmt] [-t cache] imgname
```

### 15.3. イメージの比較

**qemu-img compare** コマンドを使って、2つの指定されたイメージファイル (*imgname1* および *imgname2*) の内容を比較します。オプションで、ファイルの形式タイプ (*fmt*) を指定します。イメージには異なる形式と設定を指定できます。

デフォルトでは、サイズの異なるイメージは、大きい方のイメージにもう一方のイメージの後の領域に未割り当てまたはゼロ書き込みされたセクターのみが含まれる場合に同一と見なされます。さらに、1つのイメージにいずれのセクターも割り当てられておらず、もう一方にゼロバイトのみが含まれる場合、等しいものとして評価されます。**-s** オプションを指定する場合、イメージのサイズが異なり、1つのイメージに1つのセクターが割り当てられているものの2番目のイメージに割り当てられていない場合に同一のものと見なされません。

```
# qemu-img compare [-f fmt] [-F fmt] [-p] [-s] [-q] imgname1 imgname2
```

**qemu-img compare** コマンドは、以下の終了コードのいずれかを出して終了します。

- **0:** イメージは同一です。
- **1:** イメージは異なります。
- **2:** イメージのいずれかを開く際にエラーが発生しました。
- **3:** セクターの割り当ての検査時にエラーが発生しました。
- **4:** データを読み込み中にエラーが発生しました。

## 15.4. イメージのマッピング

**qemu-img map** コマンドを使用すると、指定されたイメージファイル (*imgname*) とそのバックアップファイルチェーンのメタデータをダンプできます。ダンプにより、(*imgname*) のすべてのセクターの割り当て状態が、バックアップファイルチェーンでそれを割り当てる最上位のファイルと共に表示されます。オプションでファイルの形式タイプ (*fmt*) を指定します。

```
# qemu-img map [-f fmt] [--output=fmt] imgname
```

出力形式には、**human** 形式と **json** 形式の 2 種類があります。

### 15.4.1. human 形式

デフォルト形式 (**human**) は、ファイルの非ゼロの、割り当て部分のみをダンプします。出力により、データの読み取り可能なファイルとファイル内のオフセットを特定します。各行には 4 つのフィールドが含まれます。以下は出力例になります。

Offset	Length	Mapped to	File
0	0x200000	0x500000	/tmp/overlay.qcow2
0x100000	0x100000	0x95380000	/tmp/backing.qcow2

最初の行は、イメージのオフセット 0 で開始する **0x200000 (131072)** バイトがオフセット **0x500000 (327680)** で開始する **tmp/overlay.qcow2** (raw 形式で開く) で利用可能であることを意味します。**human** 形式が指定されている場合は、圧縮され、暗号化されているか、raw 形式で利用できないデータによりエラーが発生します。



#### 注記

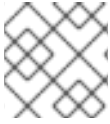
ファイル名には改行文字が含まれる場合があります。そのため、スクリプト内で **human** 形式の出力を解析することにはリスクが伴います。

### 15.4.2. json 形式

**json** オプションが指定されている場合、出力は **JSON** 形式の辞書の配列を返します。**human** オプションで提供される情報のほかにも、出力には以下の情報が含まれます。

- **data:** セクターにデータが含まれるかどうかを示すブール値のフィールド
- **zero:** データがゼロとして読み込まれるものとして既知であるかどうかを示すブール値のフィールド
- **depth:** **filename** のバックアップファイルの深さ



**注記**

**json** オプションが指定されている場合、**offset** フィールドはオプションになります。

**qemu-img map** コマンドおよび追加オプションについての詳細は、関連の **man** ページを参照してください。

## 15.5. イメージの変更

イメージファイルのイメージ形式に固有のオプションを変更します。オプションで、ファイルの形式タイプ (**fmt**) を指定します。

```
# qemu-img amend [-p] [-f fmt] [-t cache] -o options filename
```

**注記**

この操作は **qcow2** ファイル形式でのみサポートされます。

## 15.6. 既存イメージの別形式への変換

認識されているイメージ形式を別のイメージ形式に変換するには **convert** オプションを使用します。許可される形式の一覧については、「[qemu-img でサポートされる形式](#)」を参照してください。

```
# qemu-img convert [-c] [-p] [-f fmt] [-t cache] [-O output_fmt] [-o options] [-S sparse_size] filename output_filename
```

**-p** パラメーターは、コマンドの進捗を表示し (オプションのため、すべてのコマンドに使用できる訳ではありません)、**-S** フラグは、ディスクイメージ内に含まれるスパースファイルの作成を可能にします。スパースファイルは実際、ゼロ (0) のみを含む (つまり何も含まない) 物理ブロックの場合を除き、標準ファイルのように機能します。オペレーティングシステムがこのファイルを発見すると、いずれのディスク領域も使用していないものの、実際に存在し、実際のディスク領域を占めるものとしてこのファイルを処理します。これは、ディスクの使用領域が実際よりも多く表示されるため、ゲスト仮想マシンのディスクを作成する際にはとりわけ役に立ちます。たとえば、**10Gb** のディスクイメージで **-S** を **50Gb** に設定する場合、実際に使用されているのは **10Gb** のみであるにもかかわらず、**10Gb** のディスク領域のサイズが **60Gb** のように表示されます。

**output\_format** 形式を使って、**filename** ディスクイメージを **output\_filename** ディスクイメージに変換します。**-c** オプションを使うとディスクイメージをオプションで圧縮することができます。また、**-o** オプションを使って **-o encryption** のように設定すると暗号化することができます。**-o** パラメーターと共に使用できるオプションは複数あり、これらは選択する形式によって異なる点に注意してください。

暗号化や圧縮に対応しているのは **qcow2** および **qcow2** 形式のみになります。**qcow2** の暗号化では安全な **128** ビットキーによる **AES** 形式が使用されます。**qcow2** の圧縮は読み取り専用になります。このため、**qcow2** 形式から圧縮セクターを変換する場合には、このセクターは圧縮されていないデータとして新しい形式に記述されます。

**qcow** や **cow** などのサイズが拡大する可能性のある形式を使用する場合には、イメージを小さくすることができることからイメージ変換を使用することも便利です。空のセクターは検出されて、目的のイメージから削除されます。

## 15.7. 新規イメージまたはデバイスの作成およびフォーマット

サイズが **size**、形式が **format** となる **filename** という新しいディスクイメージを作成します。

```
# qemu-img create [-f format] [-o options] filename [size]
```

**-o backing\_file=filename** でベースイメージを指定すると、イメージは、それ自体とベースイメージとの相違部分のみを記録します。**commit** コマンドを使用しない限りバックアップファイルは変更されません。この場合、サイズを指定する必要はありません。

## 15.8. イメージ情報の表示

**info** パラメーターは、**filename** ディスクイメージの情報を表示します。**info** オプションの使い方を以下に示します。

```
# qemu-img info [-f format] filename
```

このコマンドはディスク上で予約されているサイズを検出する場合によく使用されます。予約サイズは表示サイズとは異なる場合があります。スナップショットがディスクイメージに格納されている場合は、それらも表示されます。このコマンドは、たとえば、ブロックデバイス上の **qcow2** イメージが使用している領域のサイズを表示します。これは、**qemu-img** を実行して確認できます。使用中のイメージが **qemu-img info** コマンドと **qemu-img check** コマンドの出力に一致するものであることを確認できます。

```
# qemu-img info /dev/vg-90.100-sluo/lv-90-100-sluo
image: /dev/vg-90.100-sluo/lv-90-100-sluo
file format: qcow2
virtual size: 20G (21474836480 bytes)
disk size: 0
cluster_size: 65536
```

## 15.9. イメージのバックアップファイルの再設定

**qemu-img rebase** はイメージのバックアップファイルを変更します。

```
# qemu-img rebase [-f fmt] [-t cache] [-p] [-u] -b backing_file [-F
backing_fmt] filename
```

バックアップファイルが **backing\_file** に変更され (**filename** の形式がこの機能に対応している場合)、バックアップファイルの形式は **backing\_format** に変更されます。



### 注記

バックアップファイルの変更 (再設定) に対応するのは **qcow2** 形式のみになります。

この再設定 **rebase** の動作には、**safe** モードと **unsafe** モードの 2 種類があります。

デフォルトでは **safe** モードが使用され、実際の再設定の操作が実行されます。新しいバックアップファイルは古いバックアップファイルとは異なる可能性があるため、**qemu-img rebase** コマンドではゲスト仮想マシンに表示される **filename** のコンテンツは変更されないようにします。**backing\_file** と **filename** の古いバックアップファイル間で異なるクラスターは、バックアップファイルに変更が加えられる前に **filename** にマージされます。

**safe** モードはイメージの変換と同様、時間を要する動作になります。正しく動作させるには古いバック

キングファイルが必要となります。

**qemu-img rebase** に **-u** オプションを渡すと **unsafe** モードが使用されます。このモードでは、**filename** のバックアップファイル名と形式のみが変更され、ファイルのコンテンツに対するチェックは行われません。新しいバックアップファイルを正しく指定していることを必ず確認してください。誤って指定していると、ゲストに表示されるイメージのコンテンツが破損することになります。

このモードはバックアップファイルの名前変更や移動を行う際に便利です。これを実行する際に、古いバックアップファイルへのアクセスは不要になります。たとえばこのモードは、バックアップファイルがすでに移動または名前変更されているイメージの修正を行う場合に使用できます。

## 15.10. ディスクイメージのサイズ変更

**size** というサイズで作成されたものとしてディスクイメージ **filename** を変更します。サイズを拡大および縮小できるのは **raw** 形式のイメージのみになります。**qcow2** バージョン 2 または **qcow2** バージョン 3 イメージは拡張することはできますが、縮小することはできません。

**filename** ディスクイメージのサイズを **size** バイトに設定するには以下を使用します。

```
# qemu-img resize filename size
```

ディスクイメージの現在のサイズに対してサイズを増減させることもできます。ディスクイメージのサイズを大きくする場合はバイト数の前に **+** を付け、小さくする場合はバイト数の前に **-** を付けます。単位の接尾辞を付けると、キロバイト (**K**)、メガバイト (**M**)、ギガバイト (**G**)、またはテラバイト (**T**) などの単位でイメージサイズを設定することができます。

```
# qemu-img resize filename [+|-]size[K|M|G|T]
```



### 警告

このコマンドを使ってディスクイメージを小さくする場合は、仮想マシン自体のファイルシステムとパーティション設定のツールを使用し、割り当てられているファイルシステムとパーティションのサイズをそれぞれ小さくしておく **必要があります**。先にこれを行っておかないとデータが失われます。

このコマンドを使ってディスクイメージを大きくしたら、仮想マシン自体のファイルシステムとパーティション設定のツールを使用し、そのデバイス上で新しい領域を実際に使用し始める必要があります。

## 15.11. スナップショットの一覧表示、作成、適用および削除

**qemu-img snapshot** コマンドの異なるパラメーターを使って、指定されたイメージ (**filename**) の既存スナップショット (**snapshot**) の一覧表示、適用、作成または削除を実行できます。

```
# qemu-img snapshot [ -l | -a snapshot | -c snapshot | -d snapshot ]  
filename
```

許可される引数は以下のようになります。

- **-l** は、指定されたディスクイメージに関連付けられたすべてのスナップショットを一覧表示します。
- 適用オプション **-a** は、ディスクイメージ (*filename*) を直前に保存された *snapshot* の状態に戻します。
- **-c** は、イメージ (*filename*) のスナップショット (*snapshot*) を作成します。
- **-d** は指定されたスナップショットを削除します。

## 15.12. QEMU-IMG でサポートされる形式

形式が **qemu-img** コマンドのいずれかで指定される場合、以下の形式タイプを使用することができます。

- **raw** - raw ディスクイメージ形式 (デフォルト) です。スピード面で最も速くなるファイルベース形式です。ファイルシステムがホール (未割り当てのブロック) に対応している場合 (たとえば **linux** の場合は **ext2** または **ext3**)、書き込みが行われたセクターのみに領域が予約されます。イメージが使用している実際のサイズを取得する場合は **qemu-img info** を使用し、Unix/Linux では **ls -ls** を使用します。生のイメージで最適なパフォーマンスを得ることができますが、利用できる機能は非常に基本的なものに限られます。たとえば、スナップショットは利用できません。
- **qcow2** - QEMU イメージ形式です。最も用途が多様となる形式で、機能も充実しています。オプションの AES 暗号化、**zlib** ベースの圧縮、仮想マシンの複数のスナップショットに対するサポート、イメージを小さくするなどのオプションを付ける場合に使用します。これらのオプションは、ホール (未割り当てのブロック) に対応しないファイルシステムで役に立ちます。機能が充実している分、パフォーマンスは低下します。

上記の形式のみがゲスト仮想マシンまたはホスト物理マシンで実行するために使用できますが、**qemu-img** は **raw** または **qcow2** 形式のいずれかに変換するために以下の形式を認識し、サポートします。イメージの形式は通常自動的に検出されます。これらの形式を **raw** または **qcow2** に変換することに加え、それらを **raw** または **qcow2** から元の形式に変換し直すこともできます。Red Hat Enterprise Linux 7 と共に提供される **qcow2** バージョンは 1.1 であることに注意してください。以前のバージョンの Red Hat Enterprise Linux と共に提供される形式は 0.10 です。イメージファイルを以前のバージョンの **qcow2** に戻すことができます。どのバージョンを使用しているかを確認するには、**qemu-img info qcow2 [imagefilename.img]** コマンドを実行します。**qcow** バージョンを変更するには、「[target 要素の設定](#)」を参照してください。

- **bochs**: Bochs のディスクイメージ形式です。
- **cloop**: Linux 圧縮ループ (Linux Compressed Loop) のイメージです。Knoppix CD-ROM などにある圧縮された CD-ROM イメージを直接再利用する場合にのみ役立ちます。
- **cow**: ユーザーモード Linux コピーオンライト (User Mode Linux Copy On Write) のイメージ形式です。**cow** は、旧バージョンとの互換性のために組み込まれています。
- **dmg**: Mac のディスクイメージ形式です。
- **nbd**: ネットワークブロックデバイスです。
- **parallels**: Parallels の仮想化ディスクイメージ形式です。

- **qcow**: 旧式の QEMU イメージ形式です。この形式は、旧式バージョンとの互換性のために組み込まれています。
- **qed**: 旧式の QEMU イメージ形式です。この形式は、旧式バージョンとの互換性のために組み込まれています。
- **vdi**: Oracle 仮想マシンの VirtualBox (Oracle VM VirtualBox) のハードディスクイメージ形式です。
- **vhdx** - Microsoft Hyper-V 仮想ハードディスク X ディスクイメージ形式です。
- **vmdk** - VMware 3 および 4 互換のイメージ形式です。
- **vvfat** - Virtual VFAT のディスクイメージ形式です。

## 第16章 KVM の移行

本章では、KVM ハイパーバイザーを実行するあるホスト物理マシンの別のマシンへのゲスト仮想マシンの移行について説明します。仮想マシンはハードウェア上で直接実行されるのではなく仮想化環境で実行されるため、ゲストを移行することができます。

### 16.1. 移行の定義と利点

ゲスト仮想マシンのメモリと仮想化しているデバイスの状態を移行先ホスト物理マシンに送信することで移行が行われます。移行するゲストのイメージは、ネットワーク接続されている共有ストレージに格納することをお勧めします。また、仮想マシンを移行する際の共有ストレージには [libvirt 管理のストレージプール](#) を使用することもお勧めします。

移行は **ライブ** (実行中) および **ライブ以外** (シャットダウン状態) のゲストの両方で実行できます。

**ライブマイグレーション**では、ゲスト仮想マシンを移行元のホストマシン上で稼働させたままそのメモリーページを移行先ホスト物理マシンに転送します。移行時に、KVM はすでに転送されたページに変更がないか移行元を監視し、最初のページがすべて転送されるとこれらの検出した変更の転送を開始します。また、KVM は移行時の転送速度を推定できるため、データ残量の転送時間が一定の設定時間になると (デフォルトでは 10 ミリ秒)、元のゲスト仮想マシンを停止してから残りのデータを転送し、移行先ホスト物理マシンでその同じゲスト仮想マシンを再開します。

一方、**ライブ以外の移行** (オフラインの移行) の場合には、ゲスト仮想マシンをいったん停止してからゲスト仮想マシンのメモリーを移行先ホスト物理マシンにコピーします。その後、ゲストは移行先ホスト物理マシンで再開し、移行元のホスト物理マシンでゲストが使用していたメモリーを解放します。この移行が完了するまでに要する時間はネットワークの帯域幅および待ち時間によって異なります。ネットワークが混雑している場合や帯域幅が狭い場合は、移行にかかる時間も長くなります。元のゲスト仮想マシンが KVM がそれらを移行先ホスト物理マシンに送信するよりも速いスピードでページを変更できる場合は、ライブマイグレーションだといつまでも完了しなくなる可能性があるため、オフラインの移行を使用する必要があります。

移行は以下の点で役立ちます。

#### 負荷分散

負荷分散: ホストマシンに負荷がかかりすぎた場合に使用率の低いホスト物理マシンにゲスト仮想マシンを移動したり、別のホストマシンの使用率が低くなっている場合にそちらにゲスト仮想マシンを移動することができます。

#### ハードウェアの非依存性

ホスト物理マシン上のハードウェアデバイスのアップグレード、追加または削除などを行う必要がある場合、ゲスト仮想マシン群を他のホスト物理マシンに安全に移動することができます。つまり、ゲスト仮想マシンはハードウェアを改善する際に生じ得るダウンタイムを経験することはありません。

#### 省エネ

仮想マシンを他のホスト物理マシンに再配分することで、アンロードされたホストシステムの電源を電力使用量の少ない時間帯にオフにして節電と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特別な状況のために、仮想マシンを別の場所に移動することができます。

### 16.2. 移行の要件と制限事項



KVM 移行を使用する前に、システムが移行の要件を満たしていることを確認し、その制限事項について理解しておく必要があります。

## 移行の要件

- 次のいずれかのプロトコルを使用してゲスト仮想マシンを共有ストレージにインストールする場合:
  - ファイバーチャネルベースの LUN
  - iSCSI
  - NFS
  - GFS2
  - SCSI RDMA プロトコル (SCSI RCP): Infiniband および 10GbE iWARP アダプターで使用されているブロックエクスポートプロトコル
- **libvirtd** サービスが有効で、実行されていることを確認します。

```
# systemctl enable libvirtd.service
# systemctl restart libvirtd.service
```

- 効果的に移行を実行できるかどうかは、**/etc/libvirt/libvirtd.conf** ファイルのパラメーターの設定に依存しています。このファイルを編集するには、以下の手順に従ってください。

### 手順16.1 libvirtd.conf の設定

1. **libvirtd.conf** を開くには、**root** として次のコマンドを実行する必要があります。

```
# vim /etc/libvirt/libvirtd.conf
```

2. 必要に応じてパラメーターを変更し、ファイルを保存します。

3. **libvirtd** サービスを再起動します。

```
# systemctl restart libvirtd
```

- [表16.1 「ライブマイグレーションの互換性」](#) で移行できるプラットフォームとバージョンを確認してください。
- 共有ストレージメディアをエクスポートする別のシステムを使用します。ストレージは、移行に使用する 2 つのホスト物理マシンのいずれにもないことを確認してください。
- 共有ストレージは、移行元システムと移行先システムの同じ場所にマウントする必要があります。マウントするディレクトリの名前も同一にする必要があります。異なるパスでイメージを維持することは可能ですが、これは推奨されていません。[virt-manager](#) を使って移行を行う予定の場合、パス名は同一である必要があります。ただし、[virsh](#) を使って移行を行う予定の場合は、移行を行う際に **--xml** オプションや **pre-hook** を使うと異なるネットワーク設定やマウントディレクトリを使用することができます。**pre-hook** についてさらに詳しくは、[libvirt アップストリームドキュメント](#) を参照してください。また XML オプションの詳細は、「[24 章 ドメイン XML の操作](#)」をご覧ください。

- **bridge+tap** のパブリックネットワーク内の既存のゲスト仮想マシンで移行を行う場合、移行元ホストマシンと移行先ホストマシンは同じネットワーク内になければなりません。ネットワークが異なると、ゲスト仮想マシンのネットワークが移行後に機能しなくなります。

## 移行の制限

- ゲスト仮想マシンの移行には、KVM に基づく仮想化技術を使って Red Hat Enterprise Linux 上で使用する場合に以下の制限が適用されます。
  - ポイントツーポイントの移行 - 移行元のハイパーバイザーから移行先ハイパーバイザーを指定するために手動で実行する必要があります。
  - 検証またはロールバックは利用できません。
  - ターゲットは手動でのみ定義できます。
  - Red Hat Enterprise Linux 7 ではストレージのライブマイグレーションは実行できませんが、ゲスト仮想マシンの電源がオフの時にストレージを移行することができます。ストレージのライブマイグレーション機能は Red Hat Virtualization で利用できます。詳細は、サービス担当者にお問い合わせください。



### 注記

virtio デバイスが搭載されているゲストマシンを移行する場合は、いずれかのプラットフォームの virtio デバイスのベクター番号を 32 以下に設定します。詳細は、「[Devices](#)」を参照してください。

## 16.3. ライブマイグレーションと RED HAT ENTERPRISE LINUX バージョンの互換性

以下に示す [表16.1「ライブマイグレーションの互換性」](#) は、サポートされているライブマイグレーションについて説明しています。

表16.1 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	例	ライブマイグレーション対応の有無	注記
前方	メジャーリリース	6.5+ → 7.x	完全サポート	問題がある場合はご報告ください。
後方	メジャーリリース	7.x → 6.y	サポートなし	
前方	マイナーリリース	7.x → 7.y (7.0 → 7.1)	完全サポート	問題がある場合はご報告ください。
後方	マイナーリリース	7.y → 7.x (7.1 → 7.0)	完全サポート	問題がある場合はご報告ください。

## 移行に関するトラブルシューティング



- **移行プロトコルに関する問題** 後方移行の実行時に「unknown section error」メッセージが表示される場合、これは一時的なエラーの可能性があるので移行プロセスを繰り返すことで問題を修復することができるはずです。解決されない場合には問題の報告をお願いします。
- **オーディオデバイスに関する問題** Red Hat Enterprise Linux 6.x から Red Hat Enterprise Linux 7.y への移行の際に、es1370 オーディオカードはサポートされなくなったことに注意してください。ac97 オーディオカードを代わりに使用してください。
- **ネットワークカードに関する問題** Red Hat Enterprise Linux 6.x から Red Hat Enterprise Linux 7.y への移行の際に、pcnet と ne2k\_pci のネットワークカードはサポートされなくなったことに注意してください。virtio-net ネットワークデバイスを代わりに使用してください。

## ネットワークストレージの設定

共有ストレージを設定してその共有ストレージにゲスト仮想マシンをインストールします。

または、「[共有ストレージの例: 単純な移行に NFS を使用する](#)」の NFS の例を使用します。

## 16.4. 共有ストレージの例: 単純な移行に NFS を使用する



### 重要

この例では、NFS を使ってゲスト仮想マシンのイメージを他の KVM ホスト物理マシンと共有します。これは大規模なインストールには実際的な方法ではありませんが、ここでは移行の手法を説明する目的で紹介します。2 台または 3 台を超えるゲスト仮想マシンを移行し、実行する場合には、この例を適用しないでください。また、**synch** パラメーターが有効にされている必要があります。これは、NFS ストレージを適切にエクスポートするのに必要です。

大規模な導入には iSCSI ストレージを選択するのが良いでしょう。設定の詳細については、「[iSCSI ベースのストレージプール](#)」を参照してください。

NFS を設定し、IP テーブルを開き、ファイアウォールを設定する方法についての詳細は、『[Red Hat Linux 7 ストレージ管理ガイド](#)』を参照してください。

NFS ファイルロック機能は KVM ではサポートされていないため使用しないでください。

### 1. libvirt イメージディレクトリーをエクスポートします。

移行を行う場合、移行対象となるシステムとは別のシステムにストレージを置かなければなりません。**/etc/exports** ファイルにデフォルトのイメージディレクトリーを追加することで、この別のシステム上にストレージをエクスポートします。

```
/var/lib/libvirt/images *.example.com(rw,no_root_squash,sync)
```

環境に応じて **hostname** パラメーターを変更します。

### 2. NFS を起動します。

a. NFS パッケージがまだインストールされていない場合はインストールを行います。

```
# yum install nfs-utils
```

b. NFS のポートが **iptables** で開かれていることを確認します (例: 2049)。さらに、NFS を **/etc/hosts.allow** ファイルに追加します。

c. NFS サービスを起動します。

```
# systemctl start nfs-server
```

### 3. ソースおよび移行先に共有ストレージをマウントします。

移行元および移行先システムに `/var/lib/libvirt/images` ディレクトリーをマウントします。

```
# mount storage_host:/var/lib/libvirt/images /var/lib/libvirt/images
```



#### 警告

移行元のホスト物理マシンに選択されるディレクトリーは、移行先ホスト物理マシン上のディレクトリーと全く同一である必要があります。これは、共有ストレージのすべてのタイプに適用されます。ディレクトリーが同一でない場合、**virt-manager** を使用する移行は失敗します。

## 16.5. VIRSH を使用した KVM のライブマイグレーション

ゲスト仮想マシンは **virsh** コマンドで別のホスト物理マシンに移行することができます。**migrate** コマンドは以下の形式のパラメーターを受け入れます。

```
# virsh migrate --live GuestName DestinationURL
```

ライブマイグレーションが不要な場合は、**--live** オプションは省略しても構いません。その他のオプションは「[virsh migrate コマンドの追加オプション](#)」に記載されています。

**GuestName** パラメーターは移行するゲスト仮想マシンの名前を表します。

**DestinationURL** パラメーターは移行先ホスト物理マシンの接続 URL です。移行先システムも Red Hat Enterprise Linux の同じバージョンを実行し、同じハイパーバイザーを使用して **libvirt** を実行している必要があります。



#### 注記

通常の移行とピアツーピアの移行では、**DestinationURL** パラメーターは異なる意味になります。

- 通常の移行の場合:**DestinationURL** は、移行元のゲスト仮想マシンから見た移行先ホスト物理マシンの URL になります。
- ピアツーピア移行の場合:**DestinationURL** は、移行元のホスト物理マシンから見た移行先ホスト物理マシンの URL になります。

コマンドを入力すると、移行先システムの **root** パスワードの入力が求められます。



## 重要

移行を正常に実行するには、名前解決が両側 (移行元と移行先) で機能する必要があります。一方から他方を検索できる必要があります。一方から他方に **ping** すると、名前解決が機能していることを確認できます。

### 例: **virsh** を使用したライブマイグレーション

この例では **host1.example.com** から **host2.example.com** に移行しています。環境に適したホスト物理マシンの名前を使用してください。この例では **guest1-rhel6-64** という名前の仮想マシンを移行しています。

この例では、共有ストレージが正しく設定され、前提条件をすべて満たしていることを仮定しています (「[移行の要件](#)」を参照)。

#### 1. ゲスト仮想マシンが実行中であることを確認します。

移行元システム **host1.example.com** で **guest1-rhel6-64** が実行中であることを確認します。

```
[root@host1 ~]# virsh list
Id Name                               State
-----
 10 guest1-rhel6-64                   running
```

#### 2. ゲスト仮想マシンを移行します。

次のコマンドを実行して、ゲスト仮想マシンの移行先 **host2.example.com** へのライブマイグレーションを実行します。移行先 URL の末尾に **/system** を追加し、**libvirt** に完全アクセスが必要であることを伝えます。

```
# virsh migrate --live guest1-rhel7-64
qemu+ssh://host2.example.com/system
```

コマンドを入力すると、移行先システムの **root** パスワードの入力が求められます。

#### 3. 待機します。

負荷やゲスト仮想マシンのサイズにより移行にかかる時間は異なります。**virsh** はエラーが発生した場合しか報告しません。ゲスト仮想マシンは、移行が完全に終了するまで移行元のホストマシンで継続的に実行されます。

#### 4. ゲスト仮想マシンが移行先ホストに到達していることを確認します。

移行先システムの **host2.example.com** で **guest1-rhel7-64** が実行されていることを確認します。

```
[root@host2 ~]# virsh list
Id Name                               State
-----
 10 guest1-rhel7-64                   running
```

これでライブマイグレーションが完了しました。



## 注記

libvirt では、TLS/SSL、UNIX ソケット、SSH、および暗号化されていない TCP などの各種のネットワーク方式に対応しています。他の方式を利用する方法についての詳細は、「[19章ゲストのリモート管理](#)」を参照してください。



## 注記

稼働していないゲスト仮想マシンの移行は、以下のコマンドで実行できます。

```
# virsh migrate --offline --persistent
```

### 16.5.1. virsh を使用した移行に関するヒント

複数の移行を別々のコマンドシェルで実行する同時ライブマイグレーションを実行することは可能です。ただし、それぞれの移行インスタンスは移行元と移行先で1つの MAX\_CLIENT 値を使用するため、慎重に計算した上で実行する必要があります。デフォルトの設定値は 20 なので、10 インスタンスまでは設定を変更せずに実行できます。設定値を変更する必要がある場合は、[手順16.1「libvirtd.conf の設定」](#)の手順を参照してください。

1. [手順16.1「libvirtd.conf の設定」](#)の手順に従って libvirtd.conf ファイルを開きます。
2. Processing controls のセクションを特定します。

```
#####
#
# Processing controls
#
# The maximum number of concurrent client connections to allow
# over all sockets combined.
#max_clients = 5000
#
# The maximum length of queue of connections waiting to be
# accepted by the daemon. Note, that some protocols supporting
# retransmission may obey this so that a later reattempt at
# connection succeeds.
#max_queued_clients = 1000
#
# The minimum limit sets the number of workers to start up
# initially. If the number of active clients exceeds this,
# then more threads are spawned, upto max_workers limit.
# Typically you'd want max_workers to equal maximum number
# of clients allowed
#min_workers = 5
#max_workers = 20
#
# The number of priority workers. If all workers from above
# pool will stuck, some calls marked as high priority
# (notably domainDestroy) can be executed in this pool.
#prio_workers = 5
#
# Total global limit on concurrent RPC calls. Should be
# at least as large as max_workers. Beyond this, RPC requests
```

```
# will be read into memory and queued. This directly impact
# memory usage, currently each request requires 256 KB of
# memory. So by default upto 5 MB of memory is used
#
# XXX this isn't actually enforced yet, only the per-client
# limit is used so far
#max_requests = 20

# Limit on concurrent requests from a single client
# connection. To avoid one client monopolizing the server
# this should be a small fraction of the global max_requests
# and max_workers parameter
#max_client_requests = 5

#####
```

3. **max\_clients** と **max\_workers** パラメーターの設定値を変更します。この2つのパラメーターの値は同一にすることをお勧めします。**max\_clients** は移行ごと2つのクライアントを使用し(移行元と移行先で1つずつ)、**max\_workers** は実行フェーズでは移行元の1ワーカー、移行先では0ワーカーを使用し、終了フェーズでは移行先で1ワーカーを使用します。

### 重要

**max\_clients** と **max\_workers** のパラメーター設定値は、libvirtd サービスに接続するすべてのゲスト仮想マシンによる影響を受けます。つまり、同じゲスト仮想マシンを使用し、かつ同時に移行を実行しているユーザーもすべて **max\_clients** と **max\_workers** パラメーターに設定された値による制限を受けます。このため、同時ライブマイグレーションを行う際は、まず最大値を注意深く検討する必要があります。

### 重要

**max\_clients** パラメーターは libvirt に接続できるクライアント数を制御します。一度に多数のコンテナが起動すると、この制限にすぐに達するか、この制限を超えてしまう可能性があります。**max\_clients** パラメーターの値は増やすことでこの状態を回避することはできますが、これにより、インスタンスに対するサービス拒否 (DoS) 攻撃に対してシステムをより脆弱な状態にする可能性があります。この問題を軽減するため、Red Hat Enterprise Linux 7.0 では **max\_anonymous\_clients** 設定が新たに導入されています。これは、許可できる接続の内、認証されていない接続の制限を指定します。実際のワークロードに合わせて **max\_clients** と **max\_anonymous\_clients** の組み合わせを実装することができます。

4. ファイルを保存してサービスを再起動します。



## 注記

数多くの SSH セッションが開始されたものの認証が行われていないために移行中に接続が落ちる場合があります。デフォルトでは、**sshd** で認証前の状態が許可されるのは 10 セッションのみです。この設定は **sshd** 設定ファイル内の **MaxStartups** パラメーターで制御されます (**/etc/ssh/sshd\_config** 内)。場合によっては、このパラメーターを調整する必要があります。DoS 攻撃 (また一般的にはリソースの過剰使用) を防ぐ目的でこの制限が設けられているため、このパラメーターを調整するには注意が必要です。値を高く設定しすぎると、当初の目的を達成することができなくなります。このパラメーターを変更するには、**/etc/ssh/sshd\_config** を編集して **MaxStartups** 行の先頭にある **#** を削除し、**10** (デフォルト値) をより大きな値に変更します。その後は必ずファイルを保存して **sshd** サービスを再起動してください。詳細は、**sshd\_config** の **man** ページを参照してください。

### 16.5.2. virsh migrate コマンドの追加オプション

--live 以外にも、virsh migrate コマンドは次のようなオプションを受け入れます。

- **--direct** - 直接の移行に使用します。
- **--p2p** - used for peer-to-peer migration
- **--tunneled** - トンネル移行に使用します。
- **--offline** - 移行先のドメインを開始したり、移行元ホスト上でこれを停止することなくドメイン定義を移行します。オフラインの移行は非アクティブなドメインで使用でき、**--persistent** オプションを指定して実行する必要があります。
- **--persistent** - 移行先ホスト物理マシンでドメインを永続的な状態に維持します。
- **--undefinesource** - 移行元ホスト物理マシンのドメインの定義を解除します。
- **--suspend** - 移行先ホスト物理マシンでドメインを一時停止の状態に維持します。
- **--change-protection** - 移行の実行中に、ドメインに対して互換性のない設定変更が行われないよう強制します。このフラグは、ハイパーバイザーでサポートされている場合に暗黙的に有効になります。ただし、ハイパーバイザーに変更保護のサポート機能がない場合には、これを明示的に使用して移行を拒否することができます。
- **--unsafe** - 移行を強制的に実行し、安全のための手順はすべて無視します。
- **--verbose** - 実行中の移行の進捗状況を表示します。
- **--compressed** - ライブマイグレーションの実行時に繰り返し転送されるメモリーページの圧縮をアクティブにします。
- **--abort-on-error** - ソフトエラー (例: I/O エラー) が移行時に発生する場合に、移行をキャンセルします。
- **--domain [name]** - ドメイン名、ID または UUID を設定します。
- **--desturi [URI]** - クライアント (通常の移行) または移行元 (p2p 移行) から表示される移行先ホストの接続 URI です。



- **--migrateuri [URI]** - 通常は省略できる移行 URI です。
- **--graphicsuri [URI]** - シームレスなグラフィックスの移行に使用されるグラフィックス URI です。
- **--listen-address [address]** - 移行先のハイパーバイザーが着信接続でバインドする必要のあるリッスンアドレスを設定します。
- **--timeout [seconds]** - ライブマイグレーションカウンターが N 秒を超えるとゲスト仮想マシンを強制的に一時停止します。これはライブマイグレーションの場合にのみ使用できます。タイムアウトが開始されると、一時停止しているゲスト仮想マシンで移行が継続されます。
- **--dname [newname]** - 移行時にドメイン名を変更する場合に使用します。このオプションも通常は省略できます。
- **--xml [filename]** - 基礎となるストレージにアクセスする際に移行元と移行先間で異なる名前を構成するなど、ドメイン XML のホスト固有の部分に多数の変更を加える場合、該当するファイル名を使用して、移行先で使用する代替 XML ファイルを指定できます。このオプションは通常は省略されます。
- **--migrate-disks [disk\_identifiers]** - このオプションは、移行時にコピーされるディスクを選択するために使用できます。これにより、一部のディスクが移行先にすでに存在する場合や有用でなくなる場合など、一部のディスクをコピーすることが望ましくない状況で、より効率的なライブマイグレーションを実行できます。**[disk\_identifiers]**は移行されるディスクのコンマ区切りの一覧に置き換える必要があります。これは、ドメイン XML ファイルの `<target dev= />` 行にあるそれぞれの引数で特定されます。

さらに、以下のコマンドも便利です。

- **virsh migrate-setmaxdowntime [domain] [downtime]** - 別のホストにライブマイグレーションされるドメインに許容可能な最大ダウンタイムを設定します。指定されるダウンタイムはミリ秒で表されます。指定されるドメインは、移行されるドメインと同じである必要があります。
- **virsh migrate-compcache [domain] --size** - ライブマイグレーション中に繰り返し転送されるメモリーページを圧縮するために使用されるキャッシュの容量 (バイト単位) を設定し、取得します。**--size** が使用されていない場合、コマンドは圧縮キャッシュの現在のサイズを表示します。**--size** が使用されていて、バイト単位で指定されている場合には、ハイパーバイザーは、現在のサイズが表示された後に指定サイズに一致する圧縮に変更するよう指示されます。**--size** 引数は、移行プロセスや **domjobinfo** から取得される圧縮キャッシュミスの増加数に対応して、ドメインのライブマイグレーション実行時に使用されることになっています。
- **virsh migrate-setspeed [domain] [bandwidth]** - 別のホストに移行される指定ドメインに移行帯域幅 (メビバイト/秒) を設定します。
- **virsh migrate-getspeed [domain]** - 指定ドメインに利用できる最大の移行帯域幅 (メビバイト/秒) を取得します。

詳細は、「[移行の制限](#)」または **virsh** の man ページを参照してください。

## 16.6. VIRT-MANAGER を使用した移行

このセクションでは、**virt-manager** を使ってあるホスト物理マシンから別のホスト物理マシンに KVM ゲスト仮想マシンを移行する方法について説明します。

1. 移行先のホスト物理マシンに接続します。  
**virt-manager インターフェース**において、**ファイル**メニューを選択し、次に **接続を追加** をクリックして移行先のホスト物理マシンに接続します。
2. 接続を追加します。  
**接続を追加** ウィンドウが表示されます。

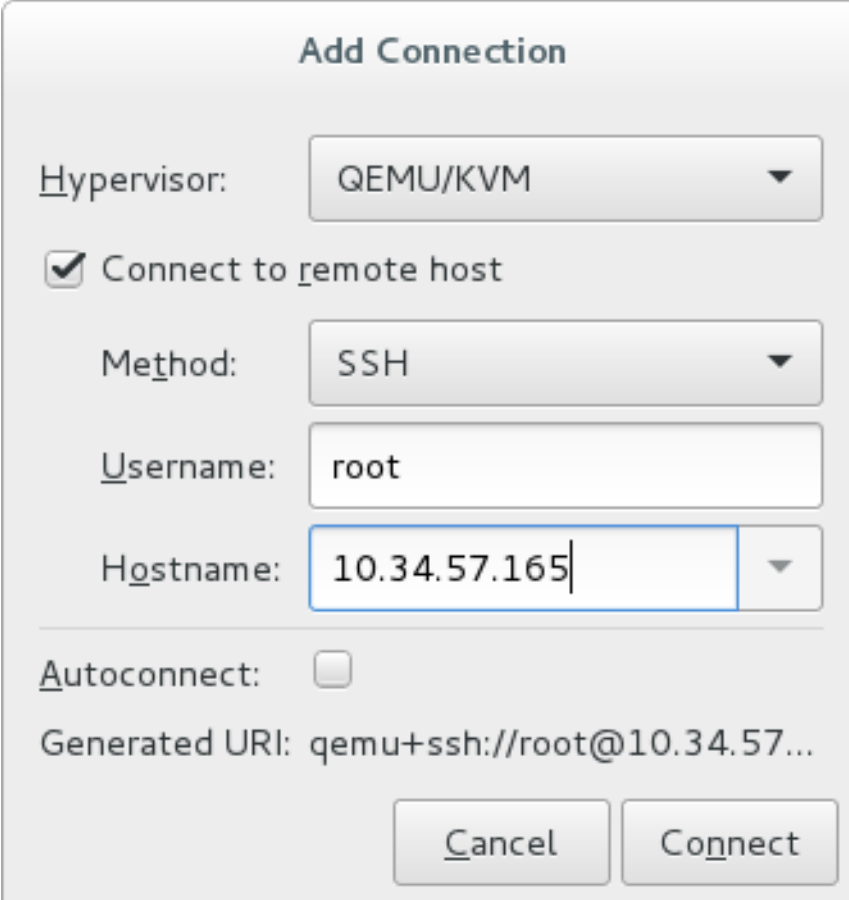


図16.1 移行先ホスト物理マシンへの接続の追加

以下のような詳細を入力します。

- **ハイパーバイザー:** **QEMU/KVM** を選択します。
- **メソッド:** 接続方法を選択します。
- **ユーザー名:** リモートのホスト物理マシンのユーザー名を入力します。
- **ホスト名:** リモートのホスト物理マシンのホスト名を入力します。



#### 注記

接続オプションの詳細については、「[リモート接続の追加](#)」を参照してください。

**接続** をクリックします。この例では **SSH** 接続を使用しているため、次のステップで指定ユーザーのパスワードを入力する必要があります。



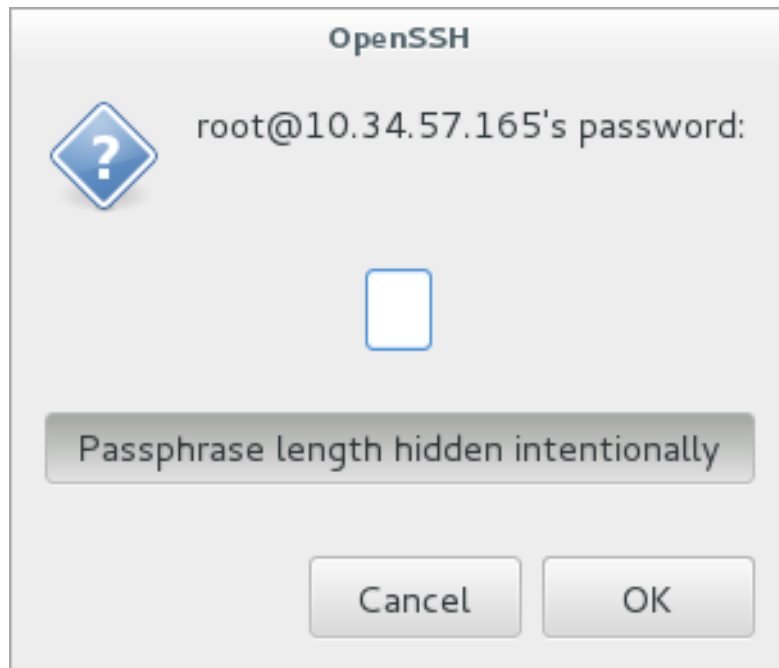


図16.2 パスワードの入力

3. 共有ストレージを設定します。

移行元および移行先の両ホストが、たとえば **NFS** を使ってストレージを共有していることを確認します。

4. ゲスト仮想マシンを移行します。

移行するゲストを右クリックして **移行** をクリックします。

**移行先のホスト** フィールドのドロップダウンリストを使い、ゲスト仮想マシンの移行先とするホスト物理マシンを選択し、**マイグレーション** をクリックします。

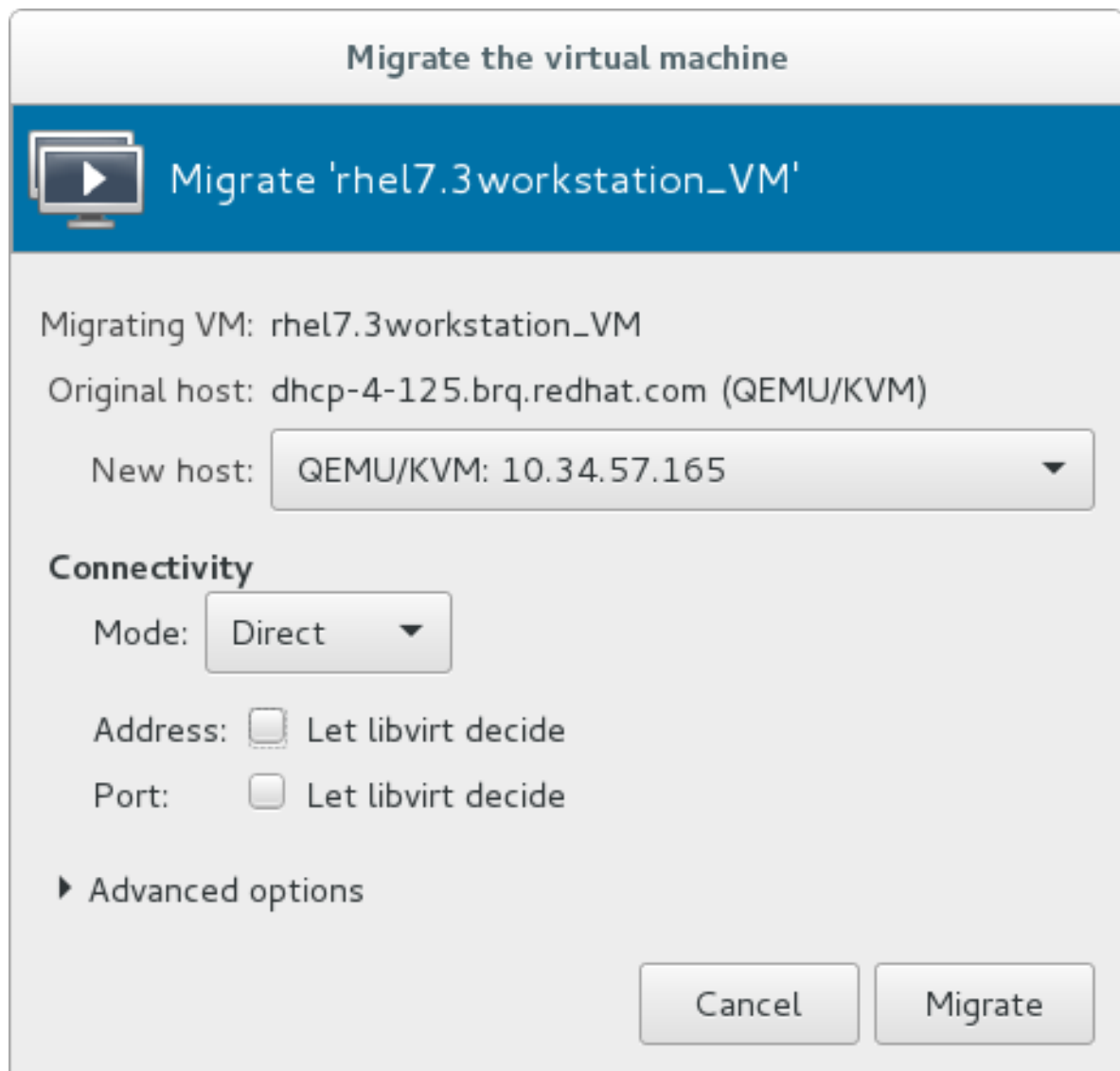


図16.3 移行先ホスト物理マシンの選択と移行プロセスの開始

進捗ウィンドウが表示されます。

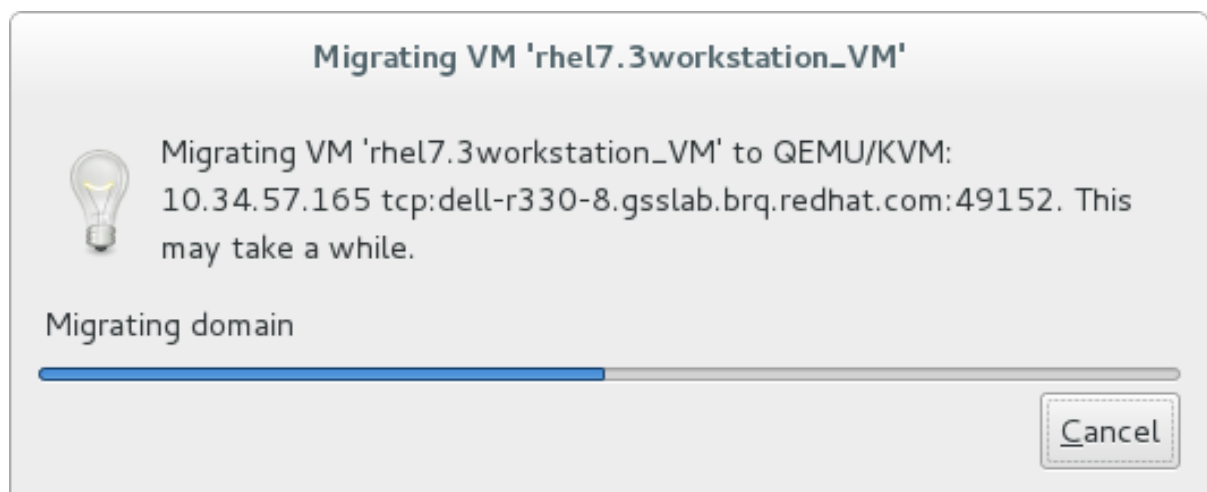


図16.4 進捗ウィンドウ

移行が正常に終了すると、**virt-manager** には、移行先ホストで実行されている新たに移行したゲスト仮想マシンが表示されます。

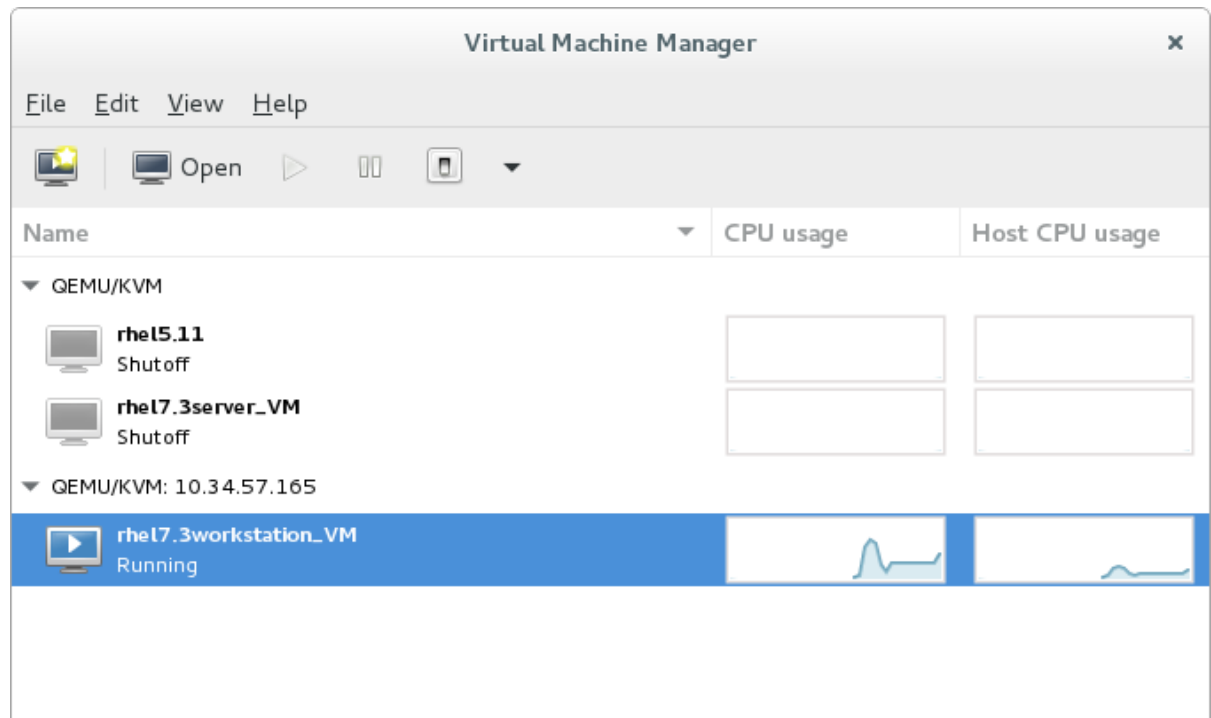


図16.5 移行先ホスト物理マシンで実行される移行済みゲスト仮想マシン

## 第17章 ゲスト仮想マシンデバイスの設定

Red Hat Enterprise Linux 7 は、ゲスト仮想マシンの以下の 3 つのクラスのデバイスに対応します。

- **エミュレートされたデバイス**は、実際のハードウェアを模倣する純粹の仮想デバイスです。変換されていないゲストオペレーティングシステムは標準のインボックスドライバを使ってこれらのデバイスと動作できるようになります。
- **Virtio デバイス (準仮想化とも呼ばれる)**は、仮想マシン内で最適に動作するように設計された仮想デバイスです。Virtio デバイスは、エミュレートされたデバイスと似ていますが、Linux 以外の仮想マシンにはこれらのデバイスが必要とするドライバがデフォルトで含まれていません。仮想マシンマネージャー (**virt-manager**) や Red Hat Virtualization Hypervisor といった仮想化管理ソフトウェアは、対応する Linux 以外のゲストオペレーティングシステム用にこれらのドライバを自動的にインストールします。Red Hat Enterprise Linux 7 は最高 216 の virtio デバイスに対応します。詳細については、「[5章 KVM 準仮想化 \(virtio\) ドライバ](#)」を参照してください。
- **割り当てデバイス**は、仮想マシンに公開されている物理デバイスです。この方法は、パススルーとも呼ばれます。デバイス割り当てにより、仮想マシンによる PCI デバイスへの排他的アクセスが可能となり、PCI デバイスを使用した各種のタスクを実行できるようになります。また、PCI デバイスがゲストオペレーティングシステムに物理的に接続されているかのように表示させ、動作させることができます。Red Hat Enterprise Linux 7 は、1 仮想マシンあたり最高 32 の割り当てデバイスに対応します。

デバイス割り当ては、[グラフィックデバイスの選択](#)を含めて PCIe デバイス上でサポートされています。パラレル PCI デバイスは割り当てデバイスとしてのサポートが可能ですが、セキュリティとシステム設定の競合により、極端な制限があります。

Red Hat Enterprise Linux 7 は、仮想マシンへの単一機能のスロットとして公開されるデバイスの PCI ホットプラグをサポートします。単一機能のホストデバイスとマルチ機能のホストデバイスの個別の機能は、このサポートを有効にするように設定できます。デバイスを仮想マシンへのマルチ機能の PCI スロットとして公開する設定は、ノンホットプラグアプリケーションの場合にお勧めします。

特定デバイスおよび関連する制限の詳細は、「[Devices](#)」を参照してください。

### 注記

割り込み再マッピングのプラットフォームサポートは、割り当てデバイスを持つゲストをホストから完全に分離するために必要です。このサポートがない場合、ホストは悪意のあるゲストからの割り込み挿入攻撃に対して脆弱になる可能性があります。ゲストが信頼される環境では、管理者は **vfio\_iommu\_type1** モジュールに対して **allow\_unsafe\_interrupts** オプションを使用する PCI デバイス割り当てを許可することを選択できます。これは、以下を含む **.conf** ファイル (例: **local.conf**) を **/etc/modprobe.d** に追加することで永続的に実行できます。

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

または **sysfs** エントリーを動的に使用して同じことを実行します。

```
# echo 1 >
/sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts
```

### 17.1. PCI デバイス

PCI デバイス割り当ては、Intel VT-d または AMD IOMMU 対応のハードウェアプラットフォーム上でのみ利用可能です。PCI デバイス割り当てを機能させるには、Intel VT-d または AMD IOMMU の仕様がホストの BIOS で有効にされている必要があります。

### 手順17.1 Intel システムでの PCI デバイス割り当ての準備

#### 1. Intel VT-d 仕様を有効にします。

Intel VT-d 仕様は、物理デバイスを仮想マシンに直接割り当てるためのハードウェアサポートを提供します。この仕様は、Red Hat Enterprise Linux で PCI デバイス割り当てを使用するために必要なものです。

Intel VT-d 仕様は、BIOS で有効にされている必要があります。システムメーカーの中には、この仕様をデフォルトで無効にしているところもあります。この仕様に言及するために使用される用語はメーカーによって異なります。それぞれの該当する用語については、システムメーカーの資料を参照してください。

#### 2. カーネルで Intel VT-d をアクティブにします。

カーネルで Intel VT-d をアクティブにするには、`intel_iommu=on` と `iommu=pt` パラメーターを `/etc/sysconfig/grub` ファイルの GRUB\_CMDLINX\_LINUX 行の終わりの引用符の内側に追加します。

以下は、Intel VT-d をアクティブにした `grub` ファイルの修正例です。

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ] &&
/usr/sbin/
rhcrashkernel-param || :) rhgb quiet intel_iommu=on iommu=pt"
```

#### 3. 設定ファイルを再生成します。

以下を実行して `/etc/grub2.cfg` を再生成します。

```
grub2-mkconfig -o /etc/grub2.cfg
```

UEFI ベースのホストの場合、ターゲットファイルは `/etc/grub2-efi.cfg` であることに注意してください。

#### 4. これで使用できるようになります。

システムを再起動して、変更を有効にします。これでシステムで PCI デバイス割り当てを使用できるようになりました。

### 手順17.2 AMD システムでの PCI デバイス割り当ての準備

#### 1. AMD IOMMU 仕様を有効にします。

AMD IOMMU 仕様は、Red Hat Enterprise Linux で PCI デバイス割り当てを使用するために必要なものです。この仕様は、BIOS で有効にされている必要があります。システムメーカーの中には、この仕様をデフォルトで無効にしているところもあります。

#### 2. IOMMU カーネルサポートを有効にします。

`amd_iommu=pt` を `/etc/sysconfig/grub` の GRUB\_CMDLINX\_LINUX 行の末尾の引用符の内側に追加し、AMD IOMMU 仕様が起動時に有効になるようにします。

#### 3. 設定ファイルを再生成します。

以下を実行して `/etc/grub2.cfg` を再生成します。

```
grub2-mkconfig -o /etc/grub2.cfg
```

UEFI ベースのホストの場合、ターゲットファイルは `/etc/grub2-efi.cfg` であることに注意してください。

#### 4. これで使用できるようになります。

システムを再起動して、変更を有効にします。これでシステムで PCI デバイス割り当てを使用できるようになりました。



#### 注記

IOMMU についての詳細は、「[付録D IOMMU グループの使用](#)」を参照してください。

### 17.1.1. virsh を使用した PCI デバイスの割り当て

以下のステップでは、KVM ハイパーバイザー上の仮想マシンに PCI デバイスを割り当てる方法を説明します。

以下の例では、PCI 識別子コードが `pci_0000_01_00_0` の PCIe ネットワークコントローラーと `guest1-rhel7-64` という名前の完全仮想化ゲストマシンを使用します。

#### 手順17.3 virsh を使用した PCI デバイスのゲスト仮想マシンへの割り当て

##### 1. デバイスを特定します。

最初に、仮想マシンへのデバイス割り当てに指定されている PCI デバイスを特定します。 `lspci` コマンドで利用可能な PCI デバイスを一覧表示します。 `lspci` の出力を `grep` を使って絞り込むことができます。

この例では、以下の出力で強調表示されているイーサネットコントローラーを使用します。

```
# lspci | grep Ethernet
00:19.0 Ethernet controller: Intel Corporation 82567LM-2 Gigabit
Network Connection
01:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
01:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
```

このイーサネットコントローラーは、`00:19.0` の短い識別子と共に表示されています。この PCI デバイスを仮想マシンに割り当てるには、`virsh` が使用する完全な識別子を見つける必要があります。

これを実行するには、`virsh nodeudev-list` コマンドを使用して、ホストマシンに接続されている特定の種類 (`pci`) のデバイスをすべて一覧表示します。次に出力を参照し、使用するデバイスの短い識別子にマップされている文字列を探します。

この例では、短い識別子 `00:19.0` を持つイーサネットコントローラーにマップされている文字列が強調表示されています。完全な識別子では、`:` と `.` 文字がアンダースコアに置き換えられていることに注意してください。

```
# virsh nodeudev-list --cap pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
```

```
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_10_1
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

使用するデバイスにマップされている PCI デバイス番号を書き留めてください。この番号は他のステップで必要になります。

## 2. デバイス情報を確認します。

ドメイン、バスおよび機能についての情報は、**virsh nodedev-dumpxml** コマンドの出力を参照してください。

```
# virsh nodedev-dumpxml pci_0000_00_19_0
<device>
  <name>pci_0000_00_19_0</name>
  <parent>computer</parent>
  <driver>
    <name>e1000e</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>0</bus>
    <slot>25</slot>
    <function>0</function>
    <product id='0x1502'>82579LM Gigabit Network
Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='7'>
      <address domain='0x0000' bus='0x00' slot='0x19'
function='0x0' />
    </iommuGroup>
  </capability>
</device>
```

図17.1 内容のダンプ



## 注記

IOMMU グループは、IOMMU から見たデバイスの可視性と分離度に基づいて決定されます。それぞれの IOMMU グループには、1 つ以上のデバイスが含まれる可能性があります。複数のデバイスが表示される場合、すべてのエンドポイントが、ゲストに割り当てられる IOMMU グループ内のすべてのデバイスに対して要求される必要があります。これは、追加のエンドポイントをゲストに割り当てるか、または **virsh nodedev-detach** を使用してエンドポイントをホストドライバーから分離するか、いずれかの方法で実行できます。単一グループに含まれるデバイスは、複数のゲスト間で分割したり、ホストとゲストの間で分割したりすることができないことがあります。PCIe ルートポート、スイッチポート、およびブリッジなどエンドポイント以外のデバイスはホストドライバーから分離することはできません。これらはエンドポイントの割り当てを妨げることはありません。

IOMMU グループ内のデバイスは、**virsh nodedev-dumpxml** 出力の **iommuGroup** セクションを使用して判別できます。グループの各メンバーは別個の「アドレス」フィールドで指定されます。この情報は、以下を使用して **sysfs** 内で見つけることもできます。

```
$ ls
/sys/bus/pci/devices/0000:01:00.0/iommu_group/devices/
```

この出力の一例を示します。

```
0000:01:00.0 0000:01:00.1
```

0000.01.00.0 のみをゲストに割り当てるには、ゲストを起動する前に、使用されていないエンドポイントをホストから切り離す必要があります。

```
$ virsh nodedev-detach pci_0000_01_00_1
```

### 3. 必要な設定の詳細を決定します。

設定ファイルに必要な値については、**virsh nodedev-dumpxml pci\_0000\_00\_19\_0** コマンドの出力を参照してください。

サンプルのデバイスには、**bus = 0**、**slot = 25**、**function = 0** の値が設定されています。10 進法の設定では、これらの値を使用します。

```
bus='0'
slot='25'
function='0'
```

### 4. 設定の詳細を追加します。

仮想マシン名を指定して **virsh edit** を実行し、**<source>** セクションにデバイスエントリを追加して PCI デバイスをゲスト仮想マシンに割り当てます。

```
# virsh edit guest1-rhel7-64
```

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0' bus='0' slot='25' function='0' />
  </source>
</hostdev>
```

### 図17.2 PCI デバイスの追加

または、仮想マシン名とゲストの XML ファイルを指定して **virsh attach-device** を実行します。

```
virsh attach-device guest1-rhel7-64 file.xml
```

### 5. 仮想マシンを起動します。

```
# virsh start guest1-rhel7-64
```

これで PCI デバイスは正しく仮想マシンに割り当てられ、ゲストオペレーティングシステムからアクセスできるようになります。

### 17.1.2. virt-manager を使用した PCI デバイスの割り当て

PCI デバイスは、グラフィカルな **virt-manager** ツールを使ってゲスト仮想マシンに追加することができます。以下の手順では、Gigabit イーサネットコントローラーをゲスト仮想マシンに追加します。

#### 手順17.4 virt-manager を使用した PCI デバイスのゲスト仮想マシンへの割り当て

1. ハードウェア設定を開きます。  
ゲスト仮想マシンを開き、**ハードウェアを追加** ボタンをクリックして新規デバイスを仮想マシンに追加します。

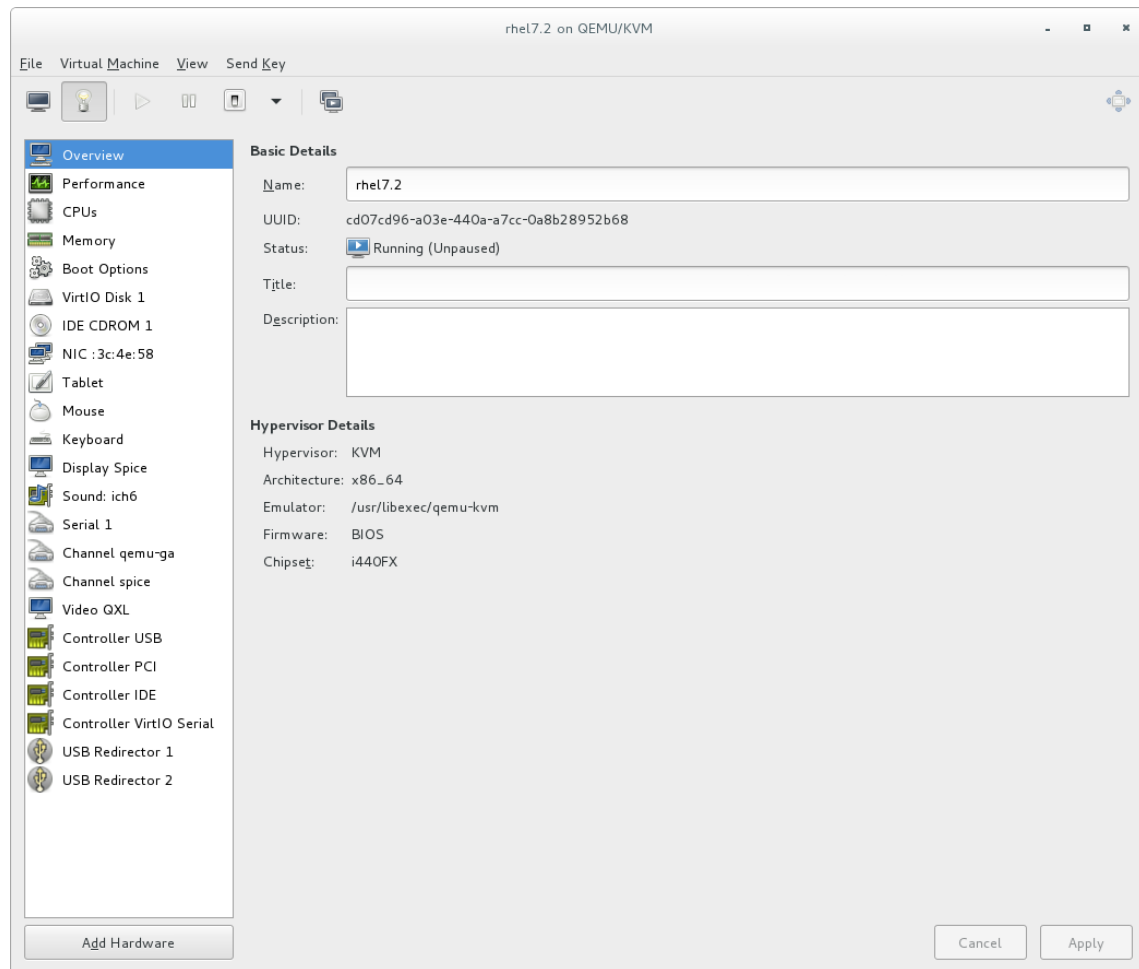


図17.3 仮想マシンのハードウェア情報ウィンドウ

## 2. PCI デバイスを選択します。

左側のハードウェア リストから **PCI** ホストデバイスを選択します。

未使用の **PCI** デバイスを選択します。別のゲストが使用中の **PCI** デバイスを選択するとエラーが発生するので注意してください。以下の例では、予備のオーディオコントローラーが使用されます。**完了** をクリックしてセットアップを終了します。

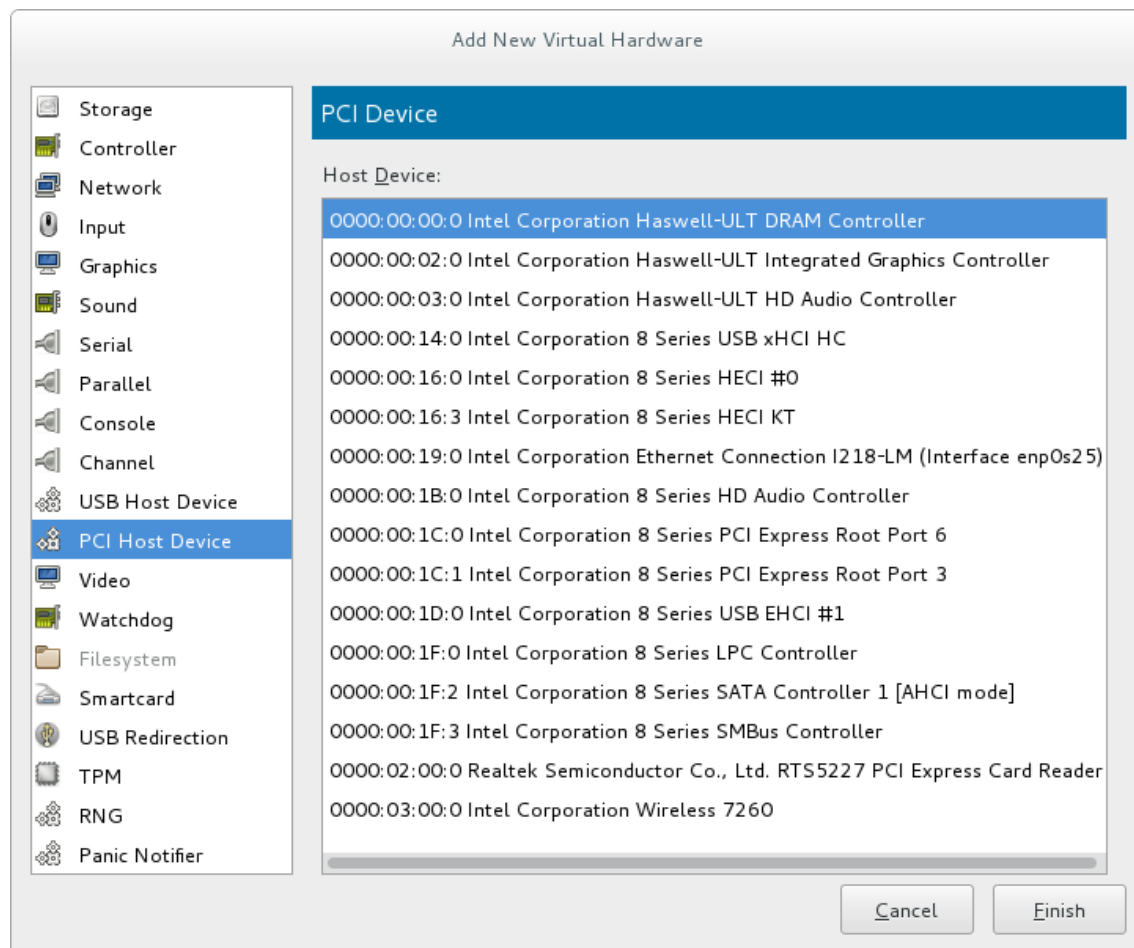


図17.4 新しい仮想ハードウェアを追加ウィザード

3. 新規デバイスを追加します。

セットアップが完了し、これでゲスト仮想マシンは PCI デバイスに直接アクセスできます。

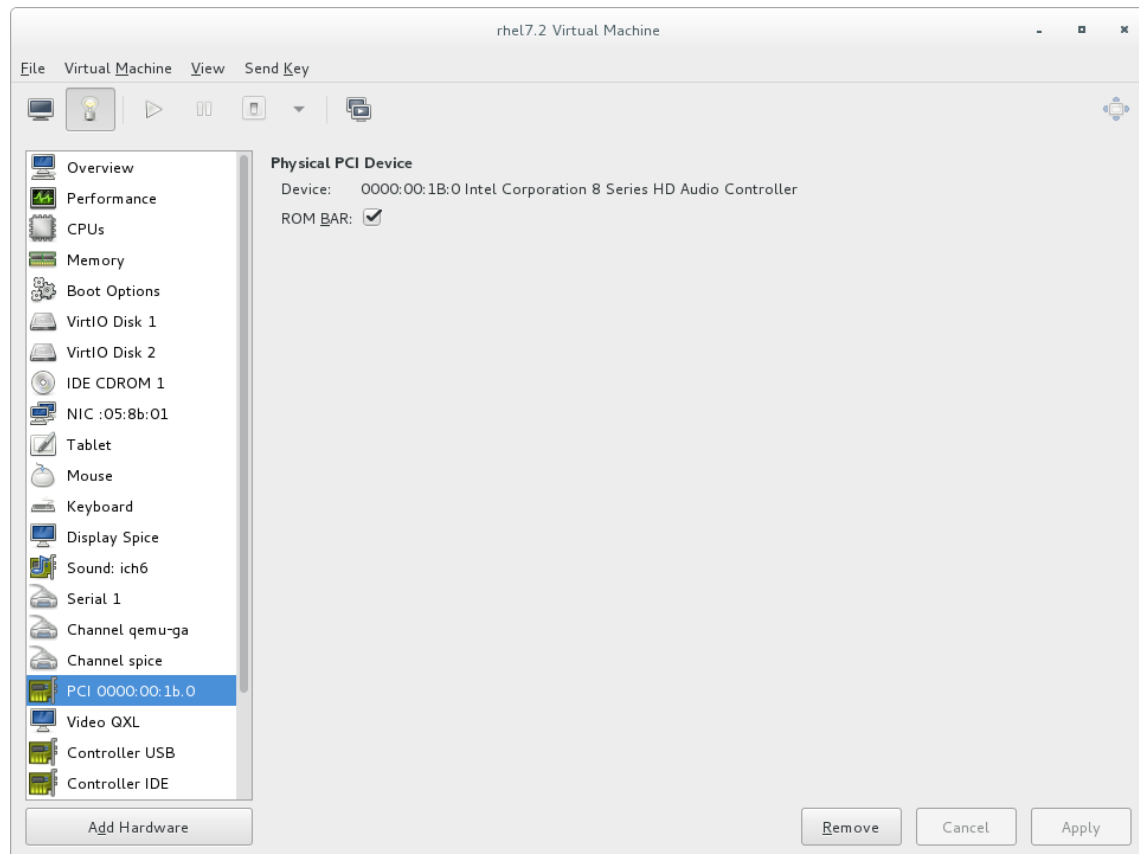


図17.5 仮想マシンのハードウェア情報ウィンドウ

## 注記

デバイスの割り当てに失敗する場合、ホストに依然として接続されている他のエンドポイントが同じ IOMMU グループ内にある可能性があります。**virt-manager** を使用してグループ情報を取得する方法はありませんが、**virsh** コマンドを使用して IOMMU グループの範囲を分析することができ、必要な場合はデバイスを分離することができます。

IOMMU グループについての詳細および **virsh** を使用してエンドポイントデバイスを切り離す方法については、「[virsh を使用した PCI デバイスの割り当て](#)」の注記を参照してください。

### 17.1.3. virt-install を使用した PCI デバイスの割り当て

**virt-install** コマンドを使ってゲストをインストールする際に、PCI デバイスを割り当てることができます。この場合、**--host-device** パラメーターを使用します。

#### 手順17.5 virt-install を使用したゲスト仮想マシンへのPCI デバイス割り当て

##### 1. デバイスを特定します。

ゲスト仮想マシンへのデバイス割り当てに指定されている PCI デバイスを特定します。

```
# lspci | grep Ethernet
00:19.0 Ethernet controller: Intel Corporation 82567LM-2 Gigabit
Network Connection
01:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
01:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
```

-

**virsh nodedev-list** コマンドは、システムに接続されている全デバイスを一覧表示し、各 PCI デバイスを文字列で特定します。出力を PCI デバイスに限定するには、以下のコマンドを入力します。

```
# virsh nodedev-list --cap pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_10_1
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

PCI デバイス番号は他のステップで必要になるので、書き留めます。

ドメイン、バスおよび各種機能についての情報は、**virsh nodedev-dumpxml** コマンドの出力を参照することができます。

```
# virsh nodedev-dumpxml pci_0000_01_00_0
```

```

<device>
  <name>pci_0000_01_00_0</name>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>1</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='7'>
      <address domain='0x0000' bus='0x00' slot='0x19'
function='0x0' />
    </iommuGroup>
  </capability>
</device>

```

図17.6 PCI デバイスファイルの内容



### 注記

IOMMU グループに複数のエンドポイントがあり、それらのすべてがゲストに割り当てられている訳ではない場合、ゲストを起動する前に以下のコマンドを実行して、他のエンドポイントをホストから手動で切り離す必要があります。

```
$ virsh nodedev-detach pci_0000_00_19_1
```

IOMMU グループの詳細は、「[virsh を使用した PCI デバイスの割り当て](#)」の注記を参照してください。

## 2. デバイスを追加します。

**virsh nodedev** コマンドの PCI 識別子の出力を **--host-device** パラメーターの値として使います。

```

virt-install \
  --name=guest1-rhel7-64 \
  --disk path=/var/lib/libvirt/images/guest1-rhel7-64.img,size=8 \
  --vcpus=2 --ram=2048 \
  --location=http://example1.com/installation_tree/RHEL7.0-Server-x86_64/os \
  --nonetworks \
  --os-type=linux \
  --os-variant=rhel7
  --host-device=pci_0000_01_00_0

```

## 3. インストールを完了します。

これでゲストのインストールが完了しました。PCI デバイスはゲストに接続されています。

### 17.1.4. 割り当てた PCI デバイスの接続解除

ホストの PCI デバイスがゲストマシンに割り当てられると、ホストはこのデバイスを使用できなくなります。PCI デバイスが **managed** モードにある場合は (**ドメイン XML ファイル** の **managed='yes'** パラメーターを使って設定)、必要に応じてゲストマシンに接続し、ゲストマシンから切り離し、再びホストマシンに接続することができます。PCI デバイスが **managed** モードにない場合は、**virsh** または **virt-manager** を使って PCI デバイスをゲストマシンから切り離し、再び接続することができます。

#### 手順17.6 virsh を使用した PCI デバイスのゲストからの接続解除

1. デバイスの接続を解除します。

以下のコマンドを使ってゲストの XML ファイル内から PCI デバイスを削除することで、ゲストから PCI デバイスを切り離します。

```
# virsh detach-device name_of_guest file.xml
```

2. デバイスをホストに再接続します (オプション)。

デバイスが **managed** モードの場合は、このステップを省略します。デバイスはホストに自動的に戻ります。

デバイスが **managed** モードを使用していない場合は、以下のコマンドを使用して PCI デバイスをホストマシンに再接続します。

```
# virsh nodedev-reattach device
```

たとえば、**pci\_0000\_01\_00\_0** デバイスをホストに再接続するには、以下のようにします。

```
# virsh nodedev-reattach pci_0000_01_00_0
```

これでこのデバイスはホストで使用できます。

#### 手順17.7 virt-manager を使用した PCI デバイスのゲストからの接続解除

1. 仮想ハードウェアの詳細画面を開きます。

**virt-manager** で、デバイスを含む仮想マシンをダブルクリックします。**仮想マシンの情報を表示** ボタンを選択し、仮想ハードウェアの一覧を表示します。

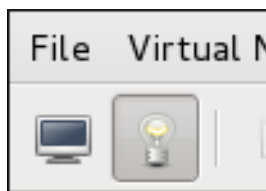


図17.7 仮想マシンの情報を表示ボタン

2. デバイスを選択し、削除します。

左側パネルの仮想デバイスの一覧より、接続を解除する PCI デバイスを選択します。



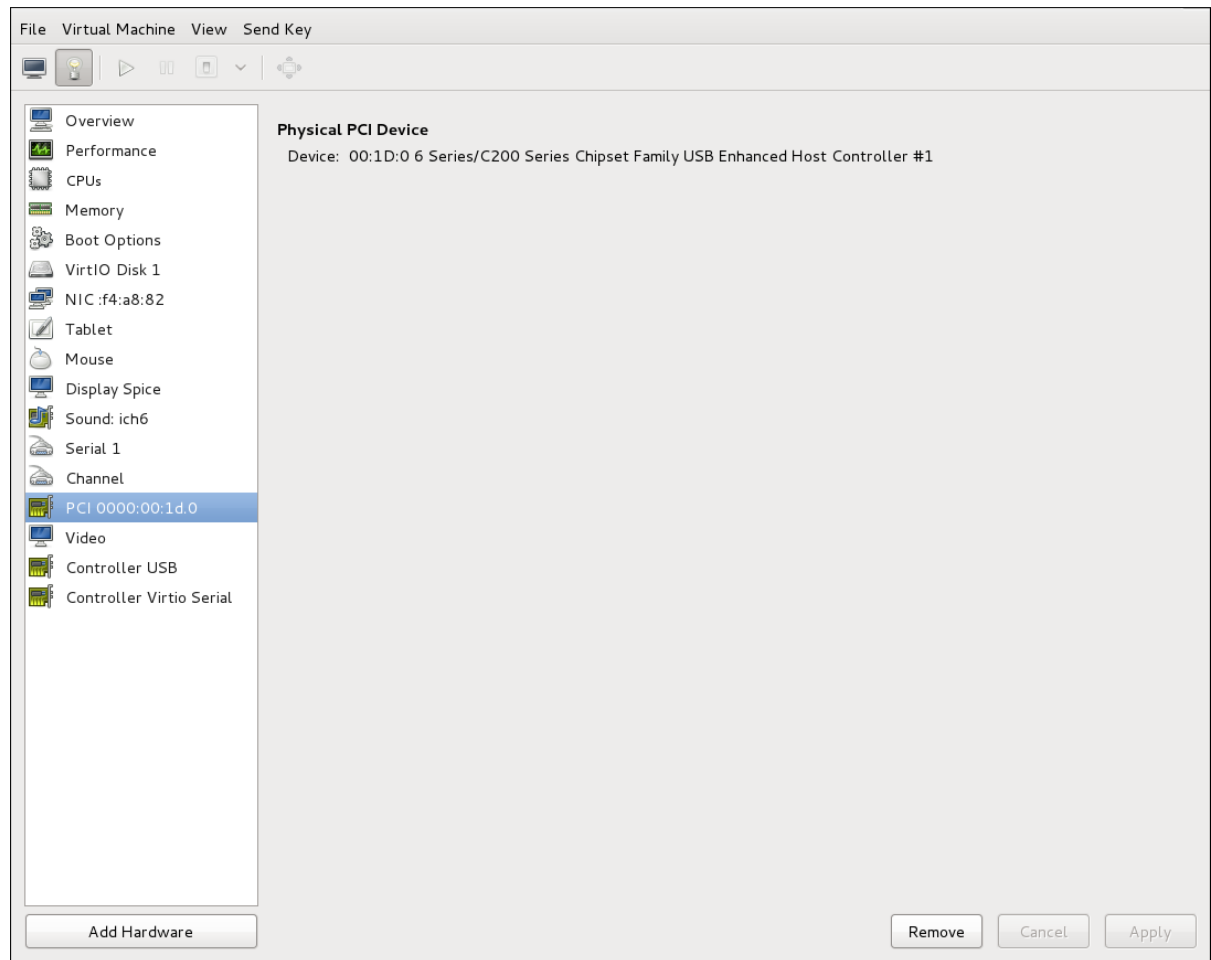


図17.8 接続解除する PCI デバイスの選択

**削除** ボタンをクリックします。これでデバイスがホストで使用可能になります。

### 17.1.5. PCI ブリッジの作成

PCI (Peripheral Component Interconnect) ブリッジは、ネットワークカード、モデムおよび音声カードなどのデバイスに接続するために使用されます。物理デバイスと同様に、仮想デバイスも PCI ブリッジに接続することができます。かつてゲスト仮想マシンに追加できる PCI デバイスの数は 31 のみでした。現在では、31 番目の PCI デバイスが追加されると、PCI ブリッジが自動的に 31 番目のスロットに配置され、追加の PCI デバイスはその PCI ブリッジに移行します。それぞれの PCI ブリッジには、追加の 31 デバイスに対応する 31 のスロットがあり、それらすべてをブリッジにすることができます。この方法で、900 を超えるデバイスをゲスト仮想マシンで利用可能にすることができます。このアクションは、ゲスト仮想マシンが実行中の場合には実行できないことに注意してください。シャットダウンしているゲスト仮想マシンに PCI デバイスを追加する必要があります。

#### 17.1.5.1. PCI ブリッジのホットプラグ/アンホットプラグサポート

PCI ブリッジのホットプラグ/アンホットプラグは、以下のデバイスタイプでサポートされています。

- virtio-net-pci
- virtio-scsi-pci
- e1000
- rtl8139

- virtio-serial-pci
- virtio-balloon-pci

### 17.1.6. PCI デバイス割り当ての制限

PCI デバイス割り当て (PCI デバイスの仮想マシンへのアタッチ) で PCIe デバイスのデバイス割り当てを有効にするには、ホストシステムが AMD IOMMU または Intel VT-d サポートを備えている必要があります。

Red Hat Enterprise Linux 7 では、ゲストデバイスドライバーによる PCI 設定領域へのアクセスは制限されています。この制限により、拡張された PCI 設定領域にあるデバイス機能に依存するドライバーが設定に失敗する場合があります。

Red Hat Enterprise Linux 7 仮想マシン 1 台あたりに割り当てられるデバイスの合計数は 32 という制限があります。これは、仮想マシンにある PCI ブリッジの数やそれらの機能がどのようにマルチファンクションスロットを構成するように組み合わせられるかにかかわらず、32 の PCI 機能に変換されることを意味します。

割り込み再マッピングのプラットフォームサポートは、割り当てデバイスを持つゲストをホストから完全に分離するために必要です。このサポートがない場合、ホストは悪意のあるゲストからの割り込み挿入攻撃に対して脆弱になる可能性があります。ゲストが信頼される環境では、管理者は **vfio\_iommu\_type1** モジュールに対して **allow\_unsafe\_interrupts** オプションを使用する PCI デバイス割り当て許可を選択できます。これは、以下を含む **.conf** ファイル (例: **local.conf**) を **/etc/modprobe.d** に追加することで永続的に実行できます。

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

または **sysfs** エントリーを動的に使用して同じことを実行します。

```
# echo 1 > /sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts
```

## 17.2. PCI デバイス割り当てと SR-IOV デバイス

PCI ネットワーク (<source> 要素でドメイン XML に指定される) は、直接のデバイス割り当て (パススルーと呼ばれることがある) を使用してゲストに直接接続されます。標準の単一ポート PCI イーサネットカードドライバーの設計上の制限により、この方法で割り当てられるのは、シングルルート I/O 仮想化 (SR-IOV) の 仮想機能 (VF) デバイスのみになります。標準の単一ポート PCI または PCIe イーサネットカードをゲストに割り当てるには、従来の <hostdev> デバイスの定義を使用します。

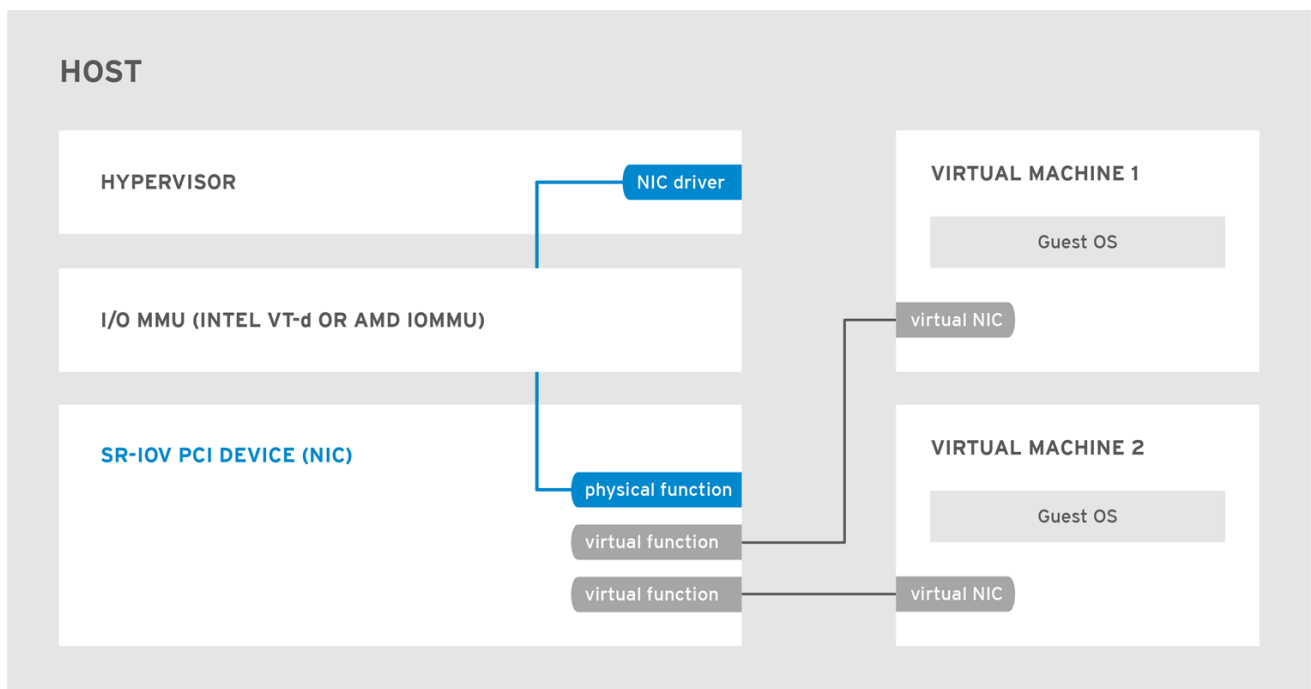
```

<devices>
<interface type='hostdev'>
  <driver name='vfio' />
  <source>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07'
function='0x0' />
  </source>
  <mac address='52:54:00:6d:90:02'>
  <virtualport type='802.1Qbh'>
    <parameters profileid='finance' />
  </virtualport>
</interface>
</devices>

```

図17.9 PCI デバイス割り当ての XML サンプル

PCI-SIG が開発したシングルルート I/O 仮想化 (SR-IOV) 仕様は、単一デバイスを複数の仮想マシンで共有できる PCI デバイス割り当てにおける標準です。SR-IOV は、仮想マシンのデバイスパフォーマンスを強化します。



RHEL\_437030\_0217

図17.10 SR-IOV のしくみ

SR-IOV は、シングルルート機能 (例: シングルイーサネットポート) を複数の別個の物理デバイスのように見せることができます。SR-IOV 対応の物理デバイスは、PCI 設定領域で複数機能のように見えるように設定することができます。各デバイスには、**Base Address Registers (BAR)** を備えた独自の設定領域があります。

SR-IOV は、以下の 2 つの PCI 機能を使用します。

- 物理機能 (PF) は、SR-IOV 機能を含む完全な PCIe デバイスです。物理機能は、通常の PCI デバイスとして検出され、管理され、設定されます。物理機能は、仮想機能を割り当てることで SR-IOV 機能を設定し、管理します。

- 仮想機能 (VF) は、I/O のみを処理する単純な PCIe 機能です。それぞれの仮想機能は物理機能から派生します。デバイス 1 台に備わる仮想機能の数は、デバイスのハードウェアによって制限されます。物理デバイスであるシングルイーサネットポートは、仮想マシンと共有可能な多くの仮想機能にマッピングできます。

ハイパーバイザーは 1 つまたは複数の仮想機能を仮想マシンに割り当てることができます。仮想機能の設定領域は、その後ゲストに提示される設定領域に割り当てられます。

仮想機能は実際のハードウェアリソースを必要とするので、それぞれの仮想機能は一度に 1 つのゲストにしか割り当てることができません。仮想マシンには複数の仮想機能を含めることができます。仮想機能は、通常のネットワークカードがオペレーティングシステムに対して表示されるようにネットワークカードとして表示されます。

SR-IOV ドライバーはカーネル内に実装されています。コア実装は PCI サブシステム内に内包されていますが、物理機能 (PF) と仮想機能 (VF) デバイスの両方にもドライバーサポートが必要です。SR-IOV 対応デバイスは、物理機能から仮想機能を割り当てることができます。仮想機能は、キューやレジスターセットなどのリソースで物理 PCI デバイスにバックアップされている PCI デバイスのように表示されます。

### 17.2.1. SR-IOV の利点

SR-IOV デバイスは、1 つの物理ポートを複数の仮想マシンと共有できます。

SR-IOV VF が仮想マシンに割り当てられている場合、VF からのすべてのネットワークトラフィックを特定の VLAN に配置するよう設定することができます (仮想マシンに対して透過的に)。仮想マシンは、自分のトラフィックが VLAN とタグ付けされていることを検知できず、このタグ付けを変更したり削除したりすることはできません。

仮想機能は、ネイティブに近いパフォーマンスを発揮し、準仮想化ドライバーやエミュレートされたアクセスよりもすぐれたパフォーマンスを提供します。仮想機能ではデータがハードウェアによって管理され、制御されるため、同一の物理サーバー上における仮想マシン間でのデータ保護が行われます。

これらの機能により、データセンター内でのホスト上の仮想マシン密度は高くなります。

SR-IOV は、複数ゲストとデバイスの帯域幅を有効活用できます。

### 17.2.2. SR-IOV の使用

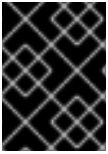
このセクションでは、SR-IOV 対応のマルチポイントネットワークカードの仮想機能をネットワークデバイスとして仮想マシンに割り当てる、PCI パススルーの使い方を説明します。

**virsh edit** または **virsh attach-device** コマンドで **<hostdev>** 内にデバイスエントリを追加することにより、SR-IOV 仮想機能を仮想マシンに割り当てられます。しかしこの操作は、通常のネットワークデバイスと違って SR-IOV VF は永久的な固有の MAC アドレスを持たず、ホストが再起動するたびに新たな MAC アドレスが割り当てられるので問題になる可能性があります。このため、再起動後にゲストに同じ仮想機能が割り当てられても、ホストが再起動すると、ゲストは新規の MAC アドレスを持たせるために新たなネットワークアダプターを決定します。その結果、ゲストは毎回新たなハードウェアが接続されたと認識し、通常はゲストのネットワークの再設定を要求することになります。

libvirt には、**<interface type='hostdev'>** インタフェースデバイスが含まれます。このインタフェースデバイスを使って、libvirt はまず指定されたネットワーク特定のハードウェア/スイッチの初期化を行います (MAC アドレスや VLAN タグ、802.1Qbh virtualport パラメーターの設定など)。その後、ゲストへの PCI デバイス割り当てを実行します。

**<interface type='hostdev'>** インタフェースデバイスの使用には、以下が必要です。

- SR-IOV 対応ネットワークカード
- Intel VT-d または AMD IOMMU 拡張をサポートするホストハードウェア
- 割り当てられる仮想機能の PCI アドレス



### 重要

SR-IOV デバイスを仮想マシンに割り当てるには、ホストハードウェアが Intel VT-d または AMD IOMMU 仕様に対応している必要があります。

SR-IOV ネットワークデバイスを Intel または AMD システムに割り当てるには、以下の手順を実行します。

## 手順17.8 SR-IOV ネットワークデバイスを Intel または AMD システムに割り当てる

1. Intel VT-d または AMD IOMMU 仕様を BIOS およびカーネル内で有効にします。  
Intel システムでは、Intel VT-d が有効にされていない場合、BIOS で有効にします。BIOS およびカーネルで Intel VT-d を有効にする方法については、[手順17.1「Intel システムでの PCI デバイス割り当ての準備」](#)を参照してください。

Intel VT-d が有効にされ、機能している場合は、このステップを省略してください。

AMD システムでは、AMD IOMMU 仕様が有効にされていない場合、BIOS で有効にします。BIOS で IOMMU を有効にする方法については、[手順17.2「AMD システムでの PCI デバイス割り当ての準備」](#)を参照してください。

2. サポートを確認します。

SR-IOV 機能に対応する PCI デバイスが検出されるかどうかを確認します。この例では、SR-IOV 対応の Intel 82576 ネットワークインタフェースカードが一覧表示されています。`lspci` コマンドを使ってデバイスが検出されたかどうかを確認します。

```
# lspci
03:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
03:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
```

出力が変更されて、他のデバイスがすべて削除されていることに注意してください。

3. 仮想機能をアクティブにします。

次のコマンドを実行します。

```
# echo ${num_vfs} > /sys/class/net/enp14s0f0/device/sriov_numvfs
```

4. 仮想機能を永続化します。

再起動後も仮想機能を存続させるためには、任意のエディターを使って以下のような `udev` ルールを作成し、必要な VF の数をネットワークインタフェースカードでサポートされる上限値までの間で指定します (この例では 2 つ)。以下の例では、`enp14s0f0` を PF ネットワークデバイス名に置き換え、`ENV{ID_NET_DRIVER}` の値を使用中のドライバーと一致させています。

```
# vim /etc/udev/rules.d/enp14s0f0.rules
```

```
ACTION=="add", SUBSYSTEM=="net", ENV{ID_NET_DRIVER}=="ixgbe",
ATTR{device/sriov_numvfs}="2"
```

これにより、この機能がブート時間に有効になります。

## 5. 新たな仮想機能を検査します。

**lspci** コマンドを使用して、Intel 82576 ネットワークデバイスに割り当てられ、新たに追加された仮想機能を一覧表示します。(または、**grep** を使って **仮想機能**、または仮想機能をサポートするデバイスを検索します。)

```
# lspci | grep 82576
0b:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
0b:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
0b:10.0 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.1 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.2 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.3 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.4 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.5 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.6 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.7 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.0 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.1 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.2 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.3 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.4 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.5 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
```

PCI デバイスの ID は、**lspci** コマンドの **-n** パラメーターを使って確認します。物理機能は、**0b:00.0** および **0b:00.1** に対応します。仮想機能にはすべて **Virtual Function** と記述されます。

## 6. **virsh** を使用してデバイスの有無を確認します。

デバイスを仮想マシンに追加する前に、**libvirt** サービスはそのデバイスを認識する必要があります。**libvirt** は、**lspci** 出力に類似する表示を使用します。**lspci** 出力では、すべての句読点とコロン (:) およびピリオド (.) は、アンダースコア (\_) に変換されます。

**virsh nodedev-list** コマンドと **grep** コマンドを使って、利用可能なホストデバイスの一覧から Intel 82576 ネットワークデバイスを選び出します。この例の **0b** は、Intel 82576 ネットワーク

トワークデバイスのフィルターです。これはお使いのシステムによっても異なり、結果として別のデバイスが加わる場合もあります。

```
# virsh nodedev-list | grep 0b
pci_0000_0b_00_0
pci_0000_0b_00_1
pci_0000_0b_10_0
pci_0000_0b_10_1
pci_0000_0b_10_2
pci_0000_0b_10_3
pci_0000_0b_10_4
pci_0000_0b_10_5
pci_0000_0b_10_6
pci_0000_0b_11_7
pci_0000_0b_11_1
pci_0000_0b_11_2
pci_0000_0b_11_3
pci_0000_0b_11_4
pci_0000_0b_11_5
```

この一覧には、仮想機能と物理機能の PCI アドレスが表示されます。

## 7. virsh を使用してデバイスの詳細情報を取得します。

**pci\_0000\_0b\_00\_0** は物理機能の 1 つであり、**pci\_0000\_0b\_10\_0** はこの物理機能に対応する最初の仮想機能です。**virsh nodedev-dumpxml** コマンドを使って、両方のデバイスのデバイス情報を表示します。

```
# virsh nodedev-dumpxml pci_0000_03_00_0
<device>
  <name>pci_0000_03_00_0</name>
  <path>/sys/devices/pci0000:00/0000:00:01.0/0000:03:00.0</path>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>3</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='virt_functions'>
      <address domain='0x0000' bus='0x03' slot='0x10'
function='0x0' />
      <address domain='0x0000' bus='0x03' slot='0x10'
function='0x2' />
      <address domain='0x0000' bus='0x03' slot='0x10'
function='0x4' />
      <address domain='0x0000' bus='0x03' slot='0x10'
function='0x6' />
      <address domain='0x0000' bus='0x03' slot='0x11'
function='0x0' />
      <address domain='0x0000' bus='0x03' slot='0x11'
function='0x2' />
```

```

        <address domain='0x0000' bus='0x03' slot='0x11'
function='0x4'/>
    </capability>
    <iommuGroup number='14'>
        <address domain='0x0000' bus='0x03' slot='0x00'
function='0x0'/>
        <address domain='0x0000' bus='0x03' slot='0x00'
function='0x1'/>
    </iommuGroup>
</capability>
</device>

```

```

# virsh nodedev-dumpxml pci_0000_03_11_5
<device>
  <name>pci_0000_03_11_5</name>
  <path>/sys/devices/pci0000:00/0000:00:01.0/0000:03:11.5</path>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igbvf</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>3</bus>
    <slot>17</slot>
    <function>5</function>
    <product id='0x10ca'>82576 Virtual Function</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='phys_function'>
      <address domain='0x0000' bus='0x03' slot='0x00'
function='0x1'/>
    </capability>
    <iommuGroup number='35'>
      <address domain='0x0000' bus='0x03' slot='0x11'
function='0x5'/>
    </iommuGroup>
  </capability>
</device>

```

この例では、仮想機能 **pci\_0000\_03\_10\_2** をステップ 8 の仮想マシンに追加します。仮想機能の **bus**、**slot**、および **function** パラメーターは、デバイスを追加するのに必要になります。

**/tmp/new-interface.xml** などの一時 XML ファイルにこれらのパラメーターをコピーします。

```

<interface type='hostdev' managed='yes'>
  <source>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x10'
function='0x2'/>
  </source>
</interface>

```



## 注記

仮想マシンが起動すると、物理アダプターが提供し、MAC アドレスが設定されたネットワークデバイスのタイプが認識されます。このMAC アドレスは、ホストおよびゲストが再起動しても変更されません。

以下の **<interface>** の例では、オプションの **<mac address>**、**<virtualport>**、**<vlan>** 要素の構文を示しています。実際には、**<vlan>** または **<virtualport>** を、例のように両方同時に使用するのではなく、どちらか一方を使用します。

```
...
<devices>
  ...
  <interface type='hostdev' managed='yes'>
    <source>
      <address type='pci' domain='0' bus='11' slot='16'
function='0' />
    </source>
    <mac address='52:54:00:6d:90:02'>
    <vlan>
      <tag id='42' />
    </vlan>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance' />
    </virtualport>
  </interface>
  ...
</devices>
```

MAC アドレスは、指定されない場合は自動生成されます。**<virtualport>** 要素は、802.1Qbh ハードウェアスイッチに接続する場合のみ使用されます。**<vlan>** 要素は、ゲストのデバイスを **42** とタグ付けされた VLAN に透過的に配置します。

## 8. 仮想機能を仮想マシンに追加します。

直前のステップで作成された一時ファイルで以下のコマンドを使用して、仮想機能を仮想マシンに追加します。これで新規デバイスは即時に割り当てられ、これ以降のゲストの再起動に備えて保存されます。

```
virsh attach-device MyGuest /tmp/new-interface.xml --live --config
```

**virsh attach-device** で **--live** を指定することで、新規デバイスを実行中のゲストに割り当てます。**--config** オプションを使うと、これ以降のゲストの再起動後も新規デバイスが確実に利用可能となります。

## 注記

**--live** オプションはゲストが実行中の場合にのみ受け入れられます。**--live** オプションが実行中ではないゲストで使用されると、**virsh** はエラーを返します。

仮想マシンが新規ネットワークインタフェースカードを検出します。この新規カードが、SR-IOV デバイスの仮想機能です。

### 17.2.3. SR-IOV デバイスの場合の PCI 割り当ての設定

SR-IOV ネットワークカードは、複数の仮想機能 (VF) を提供します。これらの仮想機能は、それぞれ PCI デバイス割り当てを使用してゲスト仮想マシンに割り当てることができます。いったん割り当てられると、それぞれの仮想機能は完全な物理ネットワークデバイスのように機能します。これにより、多くのゲスト仮想マシンが、ホスト物理マシン上で単一スロットのみを使用しているにもかかわらず、直接の PCI デバイス割り当てによるパフォーマンス上の利点を得られます。

これらの仮想機能 (VF) は、**<hostdev>** 要素を使用する従来の方法によりゲスト仮想マシンに割り当てられますが、SR-IOV VF ネットワークデバイスには永続的な固有の MAC アドレスがないため、ホスト物理マシンが再起動されるたびにゲスト仮想マシンのネットワーク設定を再設定する必要があるという問題が生じます。この問題に対処するには、VF をホスト物理マシンに割り当てる前に MAC アドレスを設定する必要があり、この設定はゲスト仮想マシンの起動時に毎回行う必要があります。他のオプションと同様にこの MAC アドレスを割り当てるには、以下の手順を参照してください。

#### 手順17.9 SR-IOV で PCI デバイスを割り当てるための MAC アドレス、vLAN、および仮想ポートの設定

**<hostdev>** 要素は、MAC アドレス割り当て、vLAN タグ ID 割り当て、または仮想ポートの割り当てなどの機能固有の項目に使用することはできません。**<mac>**、**<vlan>**、および **<virtualport>** 要素は **<hostdev>** の有効な子ではないためです。それらは **hostdev** インターフェースタイプ **<interface type='hostdev'>** で使用できます。このデバイスタイプは、**<interface>** と **<hostdev>** のハイブリッドとして機能します。そのため、PCI デバイスをゲスト仮想マシンに割り当てる前に、**libvirt** はゲスト仮想マシンの XML 設定ファイルに示されるネットワーク固有のハードウェアまたはスイッチ (MAC アドレスの設定、vLAN タグの設定、および/または 802.1Qbh スイッチとの関連付け) を初期化します。vLAN タグ設定についての情報は、「[vLAN タグの設定](#)」を参照してください。

##### 1. 情報を収集します。

**<interface type='hostdev'>** を使用するには、SR-IOV 対応ネットワークカード、また Intel VT-d または AMD IOMMU 拡張のいずれかをサポートするホスト物理マシンハードウェアが必要であり、割り当てる VF の PCI アドレスを把握しておく必要があります。

##### 2. ゲスト仮想マシンをシャットダウンします。

**virsh shutdown** コマンドを使用して、[ゲスト仮想マシンのシャットダウン](#) (ここで使用される名前は **guestVM**) を実行します。

```
# virsh shutdown guestVM
```

##### 3. XML ファイルを開いて編集します。

編集する XML ファイルを開くには、**--running** オプションを指定して **virsh save-image-edit** コマンドを実行します (詳細は、「[ゲスト仮想マシン設定の編集](#)」を参照してください)。この例の設定ファイルの名前は **guestVM.xml** です。

```
# virsh save-image-edit guestVM.xml --running
```

ユーザーのデフォルトエディターで **guestVM.xml** を開きます。

##### 4. XML ファイルを編集します。

以下のような **<devices>** エントリーを組み込むように設定ファイル (**guestVM.xml**) を更新します。

```

<devices>
  ...
  <interface type='hostdev' managed='yes'>
    <source>
      <address type='pci' domain='0x0' bus='0x00' slot='0x07'
function='0x0' /> <!--these values can be decimal as well-->
    </source>
    <mac address='52:54:00:6d:90:02' />
  <!--sets the mac address-->
    <virtualport type='802.1Qbh'>
  <!--sets the virtual port for the 802.1Qbh switch-->
      <parameters profileid='finance' />
    </virtualport>
    <vlan>
  <!--sets the vlan tag-->
      <tag id='42' />
    </vlan>
    </interface>
  ...
</devices>

```

図17.11 hostdev インターフェースタイプのドメイン XMLのサンプル

MACアドレスを指定しない場合、他のタイプのインターフェースデバイスの場合と同様に、アドレスは自動的に生成されます。さらに、**<virtualport>** 要素は、802.1Qgh ハードウェアスイッチ (802.1Qbg (別名「VEPA」)) に接続される場合にのみ使用されます。これらのスイッチは現在サポートされていません。

#### 5. ゲスト仮想マシンを再起動します。

ステップ 2 でシャットダウンしたゲスト仮想マシンを再起動するために **virsh start** コマンドを実行します。詳細は、「[仮想マシンの起動、再開および復元](#)」を参照してください。

```
# virsh start guestVM
```

ゲスト仮想マシンが起動すると、設定済みの MAC アドレスと共に、物理ホストマシンのアダプターによって指定されたネットワークデバイスが表示されます。この MAC アドレスは、ゲスト仮想マシンと物理ホストマシンの起動時に変更されることはありません。

### 17.2.4. SR-IOV 仮想機能のプールからの PCI デバイス割り当ての設定

特定の仮想機能 (VF) の PCI アドレスをゲストの設定にハードコーディングする上で、2つの重要な制限があります。

- 指定した VF は、ゲスト仮想マシンの起動時にはいつでも利用可能な状態でなければなりません。つまり、各 VF を管理者が単一のゲスト仮想マシンに対して永久的に割り当てなければならないことを意味しています (または各ゲスト仮想マシンの起動時に、すべてのゲスト仮想マシンで現在使用されていない VF の PCI アドレスを指定するよう設定ファイルを修正する必要があります)。
- ゲスト仮想マシンが別のホスト物理マシンに移行する場合、そのホスト物理マシンには、PCI バス上の同じ場所に全く同じハードウェアがなければなりません (または、ここでも起動前にゲスト仮想マシンの設定を変更する必要があります)。

これらの問題は、いずれも SR-IOV デバイスのすべての VF を含むデバイスプールで libvirt ネットワー

クを作成することによって回避できます。いったんこれが実行されると、このネットワークを参照するようにゲスト仮想マシンを設定できます。ゲストが起動するたびに、単一の VF がプールからゲスト仮想マシンに割り当てられます。ゲスト仮想マシンが停止すると、VF は別のゲスト仮想マシンが使用できるようにプールに戻ります。

### 手順17.10 デバイスポールの作成

1. ゲスト仮想マシンをシャットダウンします。

**virsh shutdown** コマンドを使用して、[ゲスト仮想マシンのシャットダウン](#) (ここで使用される名前は **guestVM**) を実行します。

```
# virsh shutdown guestVM
```

2. 設定ファイルを作成します。

任意のエディターを使用して、**/tmp** ディレクトリーに XML ファイル (例:**passthrough.xml**) を作成します。**pf dev='eth3'** は、お使いの SR-IOV デバイスの物理機能 (PF) の **netdev** 名に置き換えるようにしてください。

以下は、物理機能 (PF) をホスト物理マシンの「**eth3**」に設定し、SR-IOV アダプターのすべての VF のプールを利用可能にするネットワーク定義のサンプルです。

```
<network>
  <name>passthrough</name> <!-- This is the name of the file you
created -->
  <forward mode='hostdev' managed='yes'>
    <pf dev='myNetDevName' /> <!-- Use the netdev name of your SR-
IOV devices PF here -->
  </forward>
</network>
```

### 図17.12 ネットワーク定義のサンプルドメイン XML

3. 新しい XML ファイルをロードします。

**/tmp/passthrough.xml** を直前のステップで作成した XML ファイルの名前と場所に置き換え、以下のコマンドを入力します。

```
# virsh net-define /tmp/passthrough.xml
```

4. ゲストを再起動します。

**passthrough.xml** を直前のステップで作成した XML ファイルの名前に置き換え、以下を実行します。

```
# virsh net-autostart passthrough # virsh net-start passthrough
```

5. ゲスト仮想マシンを再起動します。

最初のステップでシャットダウンしたゲスト仮想マシンを再起動するために **virsh start** コマンドを実行します (例では、ゲスト仮想マシンのドメイン名として **guestVM** を使用しています)。詳細は、「[仮想マシンの起動、再開および復元](#)」を参照してください。

```
# virsh start guestVM
```

## 6. デバイスのパススルーを開始します。

単一デバイスのみが表示されていますが、**libvirt** はゲスト仮想マシンの初回起動時に、PF に関連付けられたすべての VF の一覧を自動的に派生させます。この起動には、以下のようなドメイン XML 内のインターフェース定義が使用されます。

```
<interface type='network'>
  <source network='passthrough'>
</interface>
```

図17.13 インターフェースネットワーク定義のサンプルドメイン XML

## 7. 検証します。

ネットワークを使用する最初のゲストの起動後に **virsh net-dumpxml passthrough** コマンドを実行して検証することができます。以下のような出力が得られます。

```
<network connections='1'>
  <name>passthrough</name>
  <uuid>a6b49429-d353-d7ad-3185-4451cc786437</uuid>
  <forward mode='hostdev' managed='yes'>
    <pf dev='eth3' />
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10'
function='0x1' />
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10'
function='0x3' />
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10'
function='0x5' />
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10'
function='0x7' />
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11'
function='0x1' />
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11'
function='0x3' />
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11'
function='0x5' />
  </forward>
</network>
```

図17.14 XML ダンプファイル *passthrough* の内容

### 17.2.5. SR-IOV の制限

SR-IOV は、以下のデバイスを使用する場合についてのみ十分なテストが行われています。

- Intel® 82576NS Gigabit Ethernet Controller (**igb** ドライバー)
- Intel® 82576EB Gigabit Ethernet Controller (**igb** ドライバー)
- Intel® 82599ES 10 Gigabit Ethernet Controller (**ixgbe** ドライバー)
- Intel® 82599EB 10 Gigabit Ethernet Controller (**ixgbe** ドライバー)



他の SR-IOV デバイスも機能する可能性があります、リリース時にはテストを実施していません。

## 17.3. USB デバイス

このセクションでは、USB デバイスの処理に必要なコマンドを扱います。

### 17.3.1. ゲスト仮想マシンへの USB デバイスの割り当て

web カメラ、カードリーダー、ディスクドライブ、キーボード、マウスなどのほとんどのデバイスは、USB ポートとケーブルを使ってコンピューターに接続されます。これらのデバイスをゲスト仮想マシンに渡す方法として 2 つの方法があります。

- **USB パススルーの使用** - これには、ゲスト仮想マシンをホストしているホスト物理マシンに、デバイスが物理的に接続されている必要があります。この場合、**SPICE** は必要ではありません。ホスト上の USB デバイスは、コマンドラインまたは **virt-manager** を使用してゲストに渡すことができます。**virt manager** の説明については、「[USB デバイスのゲスト仮想マシンへの割り当て](#)」を参照してください。**virt-manager** の説明は、デバイスのホットプラグ/アンプラグには該当しないことに注意してください。**USB** デバイスのホットプラグ/アンプラグが必要な場合は、[手順21.4「ゲスト仮想マシンが使用する USB デバイスのホットプラグ」](#)を参照してください。
- **USB リダイレクトの使用** - **USB** リダイレクトは、ホスト物理マシンがデータセンターで実行されている場合に最適に使用されます。ユーザーは、ローカルマシンまたはシンククライアントからゲスト仮想マシンに接続します。このローカルマシンには、1 つの **SPICE** クライアントがあります。ユーザーは、任意の **USB** デバイスをシンククライアントに割り当てることができ、**SPICE** クライアントはデータセンターのホスト物理マシンにこのデバイスをリダイレクトするので、これをシンククライアント上で実行されているゲスト仮想マシンで使うことができます。**virt-manager** を使用した **USB** リダイレクトについての説明は、「[USB リダイレクト](#)」を参照してください。

### 17.3.2. USB デバイスのリダイレクトへの制限の設定

フィルターを使って特定のデバイスをリダイレクトから除去するには、フィルタープロパティーを **device usb-redir** に渡します。フィルタープロパティーはフィルタールールで構成される文字列を取ります。ルールの形式は次の通りです。

```
<class>:<vendor>:<product>:<version>:<allow>
```

特定フィールドのいずれの値も受け入れるようにする場合は、**-1** の値を使用します。「|」を区切り文字として使用すると同じコマンドラインで複数のルールを使用することができます。デバイスがルールに渡したいずれの条件にも一致しない場合、そのデバイスのリダイレクトは許可されません。

#### 例17.1 ゲスト仮想マシンでリダイレクトの制限を設ける場合の例

1. ゲスト仮想マシンを用意します。
2. 次のコードの抜粋をゲスト仮想マシンのドメイン XML ファイルに追加します。

```
<redirdev bus='usb' type='spicevmc'>
  <alias name='redir0' />
  <address type='usb' bus='0' port='3' />
</redirdev>
<redirfilter>
  <usbdev class='0x08' vendor='0x1234' product='0xBEEF'
```

```
version='2.0' allow='yes' />
  <usbdev class='-1' vendor='-1' product='-1' version='-1'
allow='no' />
</redirfilter>
```

3. ゲスト仮想マシンを起動し、次のコマンドを実行して設定の変更を確認します。

```
#ps -ef | grep $guest_name
```

```
-device usb-redir,chardev=charredir0,id=redir0, /
filter=0x08:0x1234:0xBEEF:0x0200:1|-1:-1:-1:-1:0,bus=usb.0,port=3
```

4. USB デバイスをホスト物理マシンに差し込み、**virt-manager** を使ってゲスト仮想マシンに接続します。
5. メニュー内の **USB デバイスの選択** をクリックします。「**Some USB devices are blocked by host policy** (ホストのポリシーによりブロックされている USB デバイスがあります)」というメッセージが生成されます。確認の **OK** をクリックし、続行します。

フィルターが適用されます。

6. フィルターによるキャプチャーが正しく動作するよう USB デバイスの製造元と製品を確認し、ホスト物理マシンのドメイン XML に次の変更を加えて USB リダイレクトを許可します。

```
<redirfilter>
  <usbdev class='0x08' vendor='0x0951' product='0x1625'
version='2.0' allow='yes' />
  <usbdev allow='no' />
</redirfilter>
```

7. ゲスト仮想マシンを再起動し、**virt-viewer** を使ってゲスト仮想マシンに接続します。USB デバイスがトラフィックをゲスト仮想マシンにリダイレクトするようになります。

## 17.4. デバイスコントローラーの設定

ゲスト仮想マシンのアーキテクチャーにより、一部のデバイスバスは、仮想コントローラーに関連付けられた仮想デバイスのグループと共に複数回表示されることがあります。通常、**libvirt** は明示的な XML マークアップを必要とせずに、このようなコントローラーを自動的に推定できますが、仮想コントローラーの要素を明示的に設定した方がよい場合があります。

```
...
<devices>
  <controller type='ide' index='0' />
  <controller type='virtio-serial' index='0' ports='16' vectors='4' />
  <controller type='virtio-serial' index='1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0a'
function='0x0' />
  </controller>
  ...
</devices>
...
```

図17.15 仮想コントローラーのドメイン XML サンプル

各コントローラーには必須属性 **<controller type>** があります。これは、以下のいずれかである必要があります。

- ide
- fdc
- scsi
- sata
- usb
- ccid
- virtio-serial
- pci

**<controller>** 要素には、(**<address>** 要素の **controller** 属性で使用される) バスコントローラーが出現する順序を記述する 10 進整数の必須属性 **<controller index>** があります。**<controller type = 'virtio-serial'>** の場合、コントローラー経由で接続できるデバイス数を制御する 2 つの追加のオプション属性 (**ports** と **vectors** という名前) があります。

**<controller type = 'scsi'>** の場合、以下の値を取るオプション属性 **model** モデルがあります。

- auto
- buslogic
- ibmvscsi
- lsilogic
- lsisas1068
- lsisas1078
- virtio-scsi
- vmpvscsi



`<controller type='usb'>` の場合、以下の値を取るオプション属性 **model** モデルがあります。

- piix3-uhci
- piix4-uhci
- ehci
- ich9-ehci1
- ich9-uhci1
- ich9-uhci2
- ich9-uhci3
- vt82c686b-uhci
- pci-ohci
- nec-xhci

USB バスをゲスト仮想マシンに対して明示的に無効にする必要がある場合は、`<model='none'>` を使用することができることに注意してください。

コントローラー自身が PCI または USB バス上のデバイスである場合、オプションのサブ要素 `<address>` は、「[デバイスのアドレス設定](#)」に示される形式を使用して、コントローラーとマスターバスとの正確な関係を指定することができます。

オプションのサブ属性 `<driver>` は、ドライバー固有のオプションを指定することができます。現在、これはコントローラーのキューの数を指定する属性 `queues` のみをサポートします。パフォーマンスを最大化するには、仮想 CPU (vCPU) の数に一致する値を指定することが推奨されます。

USB コンパニオンコントローラーには、コンパニオンとマスターコントローラーとの正確な関係を指定するためのオプションのサブ要素 `<master>` があります。コンパニオンコントローラーは、そのマスターと同じバスにあり、コンパニオンの `index` 値は等しい値である必要があります。

使用できる XML の例を示します。

```
...
<devices>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0' bus='0' slot='4' function='7' />
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <master startport='0' />
    <address type='pci' domain='0' bus='0' slot='4' function='0'
multifunction='on' />
  </controller>
  ...
</devices>
...
```

図17.16 USB コントローラーのドメイン XML サンプル

PCI コントローラーには、以下の値を持つことのできるオプションの **model** 属性があります。

- pci-root
- pcie-root
- pci-bridge
- dmi-to-pci-bridge

暗黙的な PCI バスを提供するマシンタイプの場合、**index='0'** が指定された **pci-root** コントローラーが自動的に追加され、PCI デバイスを使用するために必要になります。**pci-root** にはアドレスがありません。PCI ブリッジは、デバイスの数が多すぎて **model='pci-root'** で指定される 1 つのバスに入らない場合や、ゼロより大きい PCI バスの数が指定されている場合に自動的に追加されます。さらに、PCI ブリッジを手動で指定することができますが、それらのアドレスは、すでに指定された PCI コントローラーによって提供される PCI バスのみを参照するものである必要があります。PCI コントローラーの **index** にギャップがあると、設定が無効になる可能性があります。以下の XML サンプルを **<devices>** セクションに追加することができます。

```
...
<devices>
  <controller type='pci' index='0' model='pci-root' />
  <controller type='pci' index='1' model='pci-bridge'>
    <address type='pci' domain='0' bus='0' slot='5' function='0'
multifunction='off' />
  </controller>
</devices>
...
```

#### 図17.17 PCI ブリッジのドメイン XML サンプル

暗黙的な PCI Express (PCIe) バスを提供するマシンタイプ (たとえば、Q35 チップセットに基づくマシンタイプ) の場合、**index='0'** が指定された **pcie-root** コントローラーがドメインの設定に自動的に追加されます。さらに、**pcie-root** にはアドレスがありませんが、31 スロット (1-31 までの番号) を提供し、PCIe デバイスを割り当てるためにのみ使用できます。**pcie-root** コントローラーを持つシステムの標準 PCI デバイスを接続するために、**model='dmi-to-pci-bridge'** が設定された **pci** コントローラーが自動的に追加されます。**dmi-to-pci-bridge** コントローラーは PCIe スロット (**pcie-root** によって提供される) にプラグインされ、それ自体は 31 の標準 PCI スロット (ホットプラグ不可能) を提供します。ホットプラグ可能な PCI スロットをゲストシステム内で設定するために、**pci-bridge** コントローラーが自動的に作成され、自動作成される **dmi-to-pci-bridge** コントローラーのスロットの 1 つに接続され、**libvirt** で自動判別される PCI アドレスを持つすべてのゲストデバイスが、この **pci-bridge** デバイス上に置かれます。

```

...
<devices>
  <controller type='pci' index='0' model='pcie-root' />
  <controller type='pci' index='1' model='dmi-to-pci-bridge'>
    <address type='pci' domain='0' bus='0' slot='0xe' function='0' />
  </controller>
  <controller type='pci' index='2' model='pci-bridge'>
    <address type='pci' domain='0' bus='1' slot='1' function='0' />
  </controller>
</devices>
...

```

図17.18 PCIe (PCI express) のドメイン XML サンプル

以下の XML 設定は USB 3.0 / xHCI エミュレーションに使用されます。

```

...
<devices>
  <controller type='usb' index='3' model='nec-xhci'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0xf'
function='0x0' />
  </controller>
</devices>
...

```

図17.19 USB3/xHCI デバイスに使用されるドメイン XML の例

## 17.5. デバイスのアドレス設定

多くのデバイスには、ゲスト仮想マシンに提示されるデバイスが仮想バス上のどこに配置されるかを説明するオプションの **<address>** サブ要素があります。アドレス (またはアドレス内のオプション属性) が入力で省略されている場合、**libvirt** は適切なアドレスを生成しますが、レイアウトにより多くの制御が必要な場合は明示的なアドレスが必要になります。**<address>** 要素を含むドメイン XML デバイスのサンプルについては、[図17.9 「PCI デバイス割り当ての XML サンプル」](#) を参照してください。

すべてのアドレスには、デバイスが置かれるバスを記述する必須属性 **type** があります。指定されるデバイスに使用するアドレスを選択することは、デバイスおよびゲスト仮想マシンのアーキテクチャーによって部分的に制限されます。たとえば、**<disk>** デバイスは **type='drive'** を使用し、**<console>** デバイスは、**i686** または **x86\_64** ゲスト仮想マシンのアーキテクチャーで **type='pci'** を使用します。さらに、各アドレスの **type** には、表で説明されるようにデバイスを配置するバス上の場所を制御する追加のオプション属性があります。

表17.1 サポートされているデバイスアドレスのタイプ

アドレスタイプ	説明
---------	----

アドレスタイプ	説明
type='pci'	<p>PCI アドレスには以下の追加属性が含まれます。</p> <ul style="list-style-type: none"> <li>● <b>domain</b> (2 バイト 16 進数整数。現在 <b>qemu</b> で使用されていません)</li> <li>● <b>bus</b> (0 から 0xff までの 16 進数値)</li> <li>● <b>slot</b> (0x0 から 0x1f までの 16 進数値)</li> <li>● <b>function</b> (0 から 7 までの値)</li> <li>● <b>multifunction</b> は、PCI コントロールレジスターに特定のスロット/機能のマルチファンクションビットをオンにするように制御します。この属性は、デフォルトで '<b>off</b>' になりますが、マルチファンクションが使用されるスロットのファンクション 0 では '<b>on</b>' に設定する必要があります。</li> </ul>
type='drive'	<p>ドライブアドレスには、以下の追加属性が含まれます。</p> <ul style="list-style-type: none"> <li>● <b>controller</b> (2 桁のコントローラー番号)</li> <li>● <b>bus</b> (2 桁のバス番号)</li> <li>● <b>target</b> (2 桁のバス番号)</li> <li>● <b>unit</b> (バス上の 2 桁のユニット番号)</li> </ul>
type='virtio-serial'	<p>それぞれの virtio-serial アドレスには以下の追加属性が含まれます。</p> <ul style="list-style-type: none"> <li>● <b>controller</b> (2 桁のコントローラー番号)</li> <li>● <b>bus</b> (2 桁のバス番号)</li> <li>● <b>slot</b> (バス内の 2 桁のスロット)</li> </ul>
type='ccid'	<p>スマートカードに使用される CCID アドレスには、以下の追加属性が含まれます。</p> <ul style="list-style-type: none"> <li>● <b>bus</b> (2 桁のバス番号)</li> <li>● <b>slot</b> 属性 (バス内の 2 桁のスロット)</li> </ul>
type='usb'	<p>USB アドレスには以下の追加属性が含まれます。</p> <ul style="list-style-type: none"> <li>● <b>bus</b> (0 から 0xffff までの 16 進数)</li> <li>● <b>port</b> (1.2 または 2.1.3.1 などの最高 4 つのオクテットからなるドット区切りの表記)</li> </ul>

アドレスタイプ	説明
type='isa'	ISA アドレスには以下の追加属性が含まれます。 <ul style="list-style-type: none"> <li>• iobase</li> <li>• irq</li> </ul>

## 17.6. 乱数ジェネレーターデバイス

乱数ジェネレーターは、オペレーティングシステムのセキュリティーにおいて非常に重要です。仮想オペレーティングシステムのセキュリティーを保護するために、Red Hat Enterprise Linux 7 には要求時に新規エントロピーをゲストに提供できる **virtio-rng** という仮想ハードウェアの乱数ジェネレーターが含まれます。

ホスト物理マシン上で、ハードウェア RNG インターフェースは **/dev/hwrng** に **chardev** を作成します。次に、この **chardev** が開かれ、ホスト物理マシンからエントロピーを取得するために読み込まれます。**rngd** デーモンと対で使用されると、ホスト物理マシンのエントロピーは、乱数度の主なソースであるゲスト仮想マシンの **/dev/random** に送られます。

乱数ジェネレーターの使用は、キーボード、マウスおよびその他の入力ゲスト仮想マシンのエントロピーを十分に生成しない場合にとくに便利です。仮想乱数ジェネレーターデバイスは、ホスト物理マシンからゲスト仮想マシンのオペレーティングシステムへのエントロピーの通過を許可します。このデバイスは **Windows** および **KVM** ゲスト仮想マシンの両方で利用できます。この手順は、[コマンドライン](#) または [virt-manager インターフェース](#) を使用して実行できます。**virtio-rng** についての詳細は、[Red Hat Enterprise Linux Virtual Machines: Access to Random Numbers Made Easy](#) を参照してください。

### 手順17.11 仮想マシンマネージャーを使用した virtio-rng の実装

1. ゲスト仮想マシンをシャットダウンします。
2. ゲスト仮想マシンを選択してから、**編集** メニューで **仮想マシンの詳細** を選択し、指定されたゲスト仮想マシンの詳細ウィンドウを開きます。
3. **ハードウェアを追加** ボタンをクリックします。
4. **新しい仮想ハードウェアを追加** ウィンドウで、**RNG** を選択し、**Random Number Generator** ウィンドウを開きます。

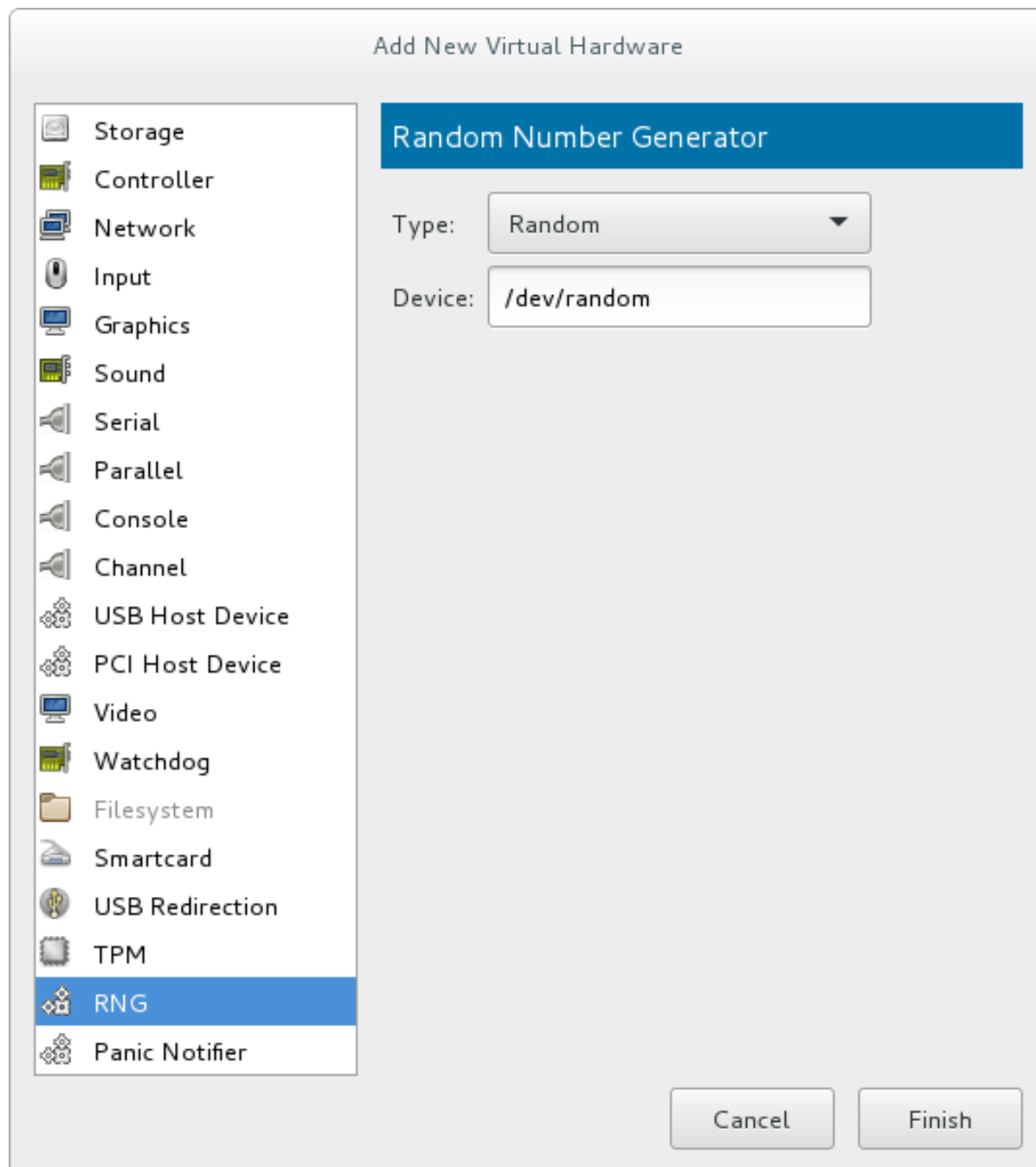


図17.20 乱数ジェネレーターウィンドウ

必要なパラメーターを入力したら **完了** をクリックします。パラメーターについては、「[virtio-rng 要素](#)」で説明されています。

#### 手順17.12 コマンドラインツールを使用した virtio-rng の実装

1. ゲスト仮想マシンをシャットダウンします。
2. **virsh edit domain-name** コマンドを使用し、必要なゲスト仮想マシンの XML ファイルを開きます。
3. 以下を含めるように **<devices>** 要素を編集します。

```

...
<devices>
  <rng model='virtio'>
    <rate period='2000' bytes='1234' />
    <backend model='random'>/dev/random</backend>
    <!-- OR -->
    <backend model='egd' type='udp'>
      <source mode='bind' service='1234' />
      <source mode='connect' host='1.2.3.4' service='1234' />
    </backend>
  </rng>
</devices>
...

```

図17.21 乱数ジェネレーターデバイス

乱数ジェネレーターデバイスは以下の XML 属性/要素を許可します。

#### virtio-rng 要素

- **<model>**: 必須の **model** 属性は、提供される RNG デバイスのタイプを指定します。
- **<backend model>**: **<backend>** 要素は、ゲストに使用されるエントロピーのソースを指定します。ソースモデルは **model** 属性を使用して設定されます。サポートされるソースモデルには、**'random'** および **'egd'** が含まれます。
  - **<backend model='random'>** - この **<backend>** タイプは、ブロック以外のキャラクターデバイスを入力として予想します。これらのデバイス例には、**/dev/random** および **/dev/urandom** があります。ファイル名は **<backend>** 要素のコンテンツとして指定されます。ファイル名が指定されていないと、ハイパーバイザーのデフォルトが使用されます。
  - **<backend model='egd'>** - このバックエンドは EGD プロトコルを使用してソースに接続されます。ソースはキャラクターデバイスとして指定されます。詳細は、キャラクターデバイスのホスト物理マシンインターフェースを参照してください。

## 17.7. GPU デバイスの割り当て

Red Hat Enterprise Linux 7 では、VGA 以外のグラフィックデバイスとして以下の GPU デバイスの PCI デバイス割り当てをサポートします。

- NVIDIA Quadro K シリーズ、M シリーズ、および P シリーズ (モデル 2000 シリーズ以降)
- NVIDIA GRID K シリーズ
- NVIDIA Tesla K シリーズおよび M シリーズ

現在、標準のエミュレートされた VGA インターフェースのほかに、2 つまでの GPU を仮想マシンに割り当てることができます。エミュレートされた VGA は起動前およびインストールに使用され、NVIDIA グラフィックスドライバがロードされると、NVIDIA GPU に引き継がれます。

この手順では、**lspci** からデバイスを特定した後にこの割り当てをホスト物理マシンから解除し、ゲスト仮想マシンに割り当てます。

### 1. ホスト物理マシンカーネルの IOMMU サポートを有効にします。

Intel VT-d システムの場合、これは **intel\_iommu=on** および **iommu=pt** パラメーターをカーネルコマンドラインに追加することによって実行されます。AMD-Vi システムの場合、オプションは **amd\_iommu=pt** になります。このオプションを有効にするには、以下のように **GRUB\_CMDLINE\_LINUX** 行を編集するか、またはこれを **/etc/sysconfig/grub** 設定ファイルに追加します。

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ] &&
/usr/sbin/rhcrashkernel-param || :) rhgb quiet intel_iommu=on
iommu=pt"
```



#### 注記

IOMMU についての詳細は、「[付録D IOMMU グループの使用](#)」を参照してください。

### 2. ブートローダー設定の再生成

このオプションを組み込むには、以下のコマンドを実行し、**grub2-mkconfig** を使用してブートローダー設定を再生成します。

```
# grub2-mkconfig -o /etc/grub2.cfg
```

UEFI ベースのホストの場合、ターゲットファイルは **/etc/grub2-efi.cfg** であることに注意してください。

### 3. ホスト物理マシンを再起動します。

このオプションを有効にするには、以下のコマンドを使ってホスト物理マシンを再起動します。

```
# reboot
```

## 手順17.13 GPU デバイスを、ホスト物理マシンドライバへのバインドから除外する

GPU 割り当ての場合、ホストドライバはデバイスの動的なバインド解除をサポートしないことが多いため、デバイスをホストドライバへのバインドから除外することをお勧めします。

### 1. PCI バスアドレスを特定します。

PCI バスアドレスとデバイスの ID を特定するには、以下の **lspci** コマンドを実行します。この例では、Quadro または GRID カードなどの VGA コントローラーが以下のように使用されます。

```
# lspci -Dnn | grep VGA
0000:02:00.0 VGA compatible controller [0300]: NVIDIA Corporation
GK106GL [Quadro K4000] [10de:11fa] (rev a1)
```

結果の検索は、このデバイスの PCI バスアドレスが 0000:02:00.0 であることを示し、デバイスの PCI ID が 10de:11fa であることを示しています。

### 2. ネイティブのホスト物理マシンドライバによる GPU デバイスの使用を防ぎます。

ネイティブのホスト物理マシンドライバが GPU デバイスを使用することを防ぐには、**pci-**



`stub` ドライバーで PCI ID を使用することができます。これを実行するには、以下のように追加オプションを `/etc/sysconfig/grub` にある `GRUB_CMDLINX_LINUX` 設定ファイルに追加します。

```
pci-stub.ids=10de:11fa
```

`pci-stub` 用に追加の PCI ID を追加するには、単にそれらをコンマで区切ります。

### 3. ブートローダー設定の再生成

このオプションを組み込むには、以下のコマンドを実行し、`grub2-mkconfig` を使用してブートローダー設定を再生成します。

```
# grub2-mkconfig -o /etc/grub2.cfg
```

UEFI ベースのホストの場合、ターゲットファイルは `/etc/grub2-efi.cfg` であることに注意してください。

### 4. ホスト物理マシンを再起動します。

このオプションを有効にするには、以下のコマンドを使ってホスト物理マシンを再起動します。

```
# reboot
```

`virsh` コマンドは、デバイスをさらに評価するために使用できますが、デバイスに関連して `virsh` を使用するには、`pci_` を追加し、区切り記号をアンダースコアに変換することで、PCI バスアドレスを `libvirt` と互換性のある形式に変換する必要があります。この例では、PCI デバイス `0000:02:00.0` の `libvirt` アドレスは `pci_0000_02_00_0` になります。`nodedev-dumpxml` オプションは、以下のようにデバイスに関する追加情報を提供します。

```
# 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>
    <!-- pay attention to this part -->
    <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>

```

図17.22 GPU 向けの XML ファイルの調整例

この出力のとりわけ重要な部分は **<iommuGroup>** 要素です。**iommuGroup** は、IOMMU 機能および PCI バストポロジーのために他のデバイスから切り離されていると見なされるデバイスセットを示します。**iommuGroup** 内のすべてのエンドポイントデバイス (PCIe root ポート、ブリッジまたはスイッチ ポートではないデバイスなど) は、ゲストに割り当てるためにネイティブのホストドライバーからのバインドを解除する必要があります。上記の例では、グループは GPU デバイス (0000:02:00.0) およびコンパニオンオーディオデバイス (0000:02:00.1) で構成されています。詳細は、「[付録D IOMMU グループの使用](#)」を参照してください。

## 注記

Nvidia オーディオ機能の割り当ては、レガシーの割り込みサポートに関連したハードウェアの問題によりサポートされません。GPU をゲストに割り当てるには、まずオーディオ機能をネイティブのホストドライバーから分離する必要があります。これは、**lspci** を使用してデバイスの PCI ID を検索し、これを **pci-stub.ids** オプションに追加するか、または **virsh** の **nodedev-detach** オプションを動的に使用して実行できます。以下は例になります。

```

# virsh nodedev-detach pci_0000_02_00_1
Device pci_0000_02_00_1 detached

```

GPU オーディオ機能は通常 GPU 自体を使用しない場合は役に立たないため、**pci-stub.ids** オプションを代わりに使用することが一般的に推奨されています。

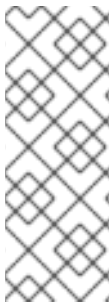
GPU の仮想マシンへの割り当ては、仮想マシン XML (**virsh edit [domain]**) を直接編集するか、

または **virsh attach-device** で GPU をドメインに割り当てるかのいずれかにより、**virt-manager** または **virsh** を使用して実行できます。**virsh attach-device** コマンドを使用している場合は、以下のようにデバイス用の XML フラグメントをまず作成する必要があります。

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <driver name='vfio' />
  <source>
    <address domain='0x0000' bus='0x02' slot='0x00' function='0x0' />
  </source>
</hostdev>
```

### 図17.23 GPU の割り当て用の XML ファイル例

これをファイルに保存し、**virsh attach-device [domain] [file] --persistent** を実行して XML を仮想マシンの設定に組み込みます。ゲスト仮想マシン内の既存のエミュレートされたグラフィックスデバイスのほかに、割り当てられた GPU が追加されることに注意してください。割り当てられた GPU は、仮想マシン内の二次的なグラフィックスデバイスとして処理されます。プライマリーグラフィックスデバイスとしての割り当てはサポートされず、仮想マシンの XML 内のエミュレートされたグラフィックスデバイスは削除することができません。



#### 注記

ゲスト内で割り当てられた Nvidia GPU を使用する場合、Nvidia ドライバーのみがサポートされます。他のドライバーは機能せず、エラーを出す可能性があります。Red Hat Enterprise Linux7 ゲストの場合、nouveau ドライバーが、インストール時にカーネルコマンドラインでオプションの **modprobe.blacklist=nouveau** を使用することでブラックリスト化される可能性があることに注意してください。他のゲスト仮想マシンについての情報は、オペレーティングシステム固有のドキュメントを参照してください。

KVM ゲスト内で割り当てられた GPU と共に使用する Xorg を設定する場合、BusID オプションを **xorg.conf** に追加して GPU のゲストアドレスを指定する必要があります。これは、ゲスト内で GPU の PCI バスアドレス (これはホストアドレスとは異なります) を判別する場合などに必要になります。

```
# lspci | grep VGA
00:02.0 VGA compatible controller: Device 1234:1111
00:09.0 VGA compatible controller: NVIDIA Corporation GK106GL [Quadro K4000] (rev a1)
```

この場合、アドレスは **00:09.0** です。ファイル **//etc/X11/xorg.conf** は以下の強調表示されたエントリを加えるように変更されます。

```
Section "Device"
    Identifier      "Device0"
    Driver          "nvidia"
    VendorName      "NVIDIA Corporation"
    BusID           "PCI:0:9:0"
EndSection
```

ゲストオペレーティングシステムに応じて、Nvidia ドライバーがロードされた状態で、ゲストはエミュレートされたグラフィックスと割り当てられたグラフィックスの両方を同時にサポートするか、またはエミュレートされたグラフィックスを無効にすることができます。割り当てられたグラフィックスフレームバッファは、**vir-manager** などのツールでは提供されないことに注意してください。割り当てられた GPU が物理ディスプレイに接続されていない場合、ゲストベースのリモートソリューションが

GPU デスクトップにアクセスするために必要になる場合があります。すべての PCI デバイス割り当ての場合と同様、割り当てられた GPU を持つゲストの移行はサポートされておらず、それぞれの GPU は単一ゲストによって排他的に所有されます。ゲストオペレーティングシステムにより、GPU のホットプラグサポートを利用できる場合があります。

## 第18章 仮想ネットワークの構築

本章では、**libvirt** を使った仮想ネットワークの作成、起動、停止、削除、変更などを行う際に理解しておく必要がある概念について説明します。

詳細は **libvirt** についての参照情報の章を参照してください。

### 18.1. 仮想ネットワークのスイッチ

Libvirt 仮想ネットワークでは **仮想ネットワークスイッチ** という概念を利用します。仮想ネットワークのスイッチは、ソフトウェアで構成され、ホスト物理マシンサーバー上で動作します。このスイッチに仮想マシン群 (ゲスト) が接続し、各ゲストのネットワークトラフィックはこのスイッチを経由することになります。

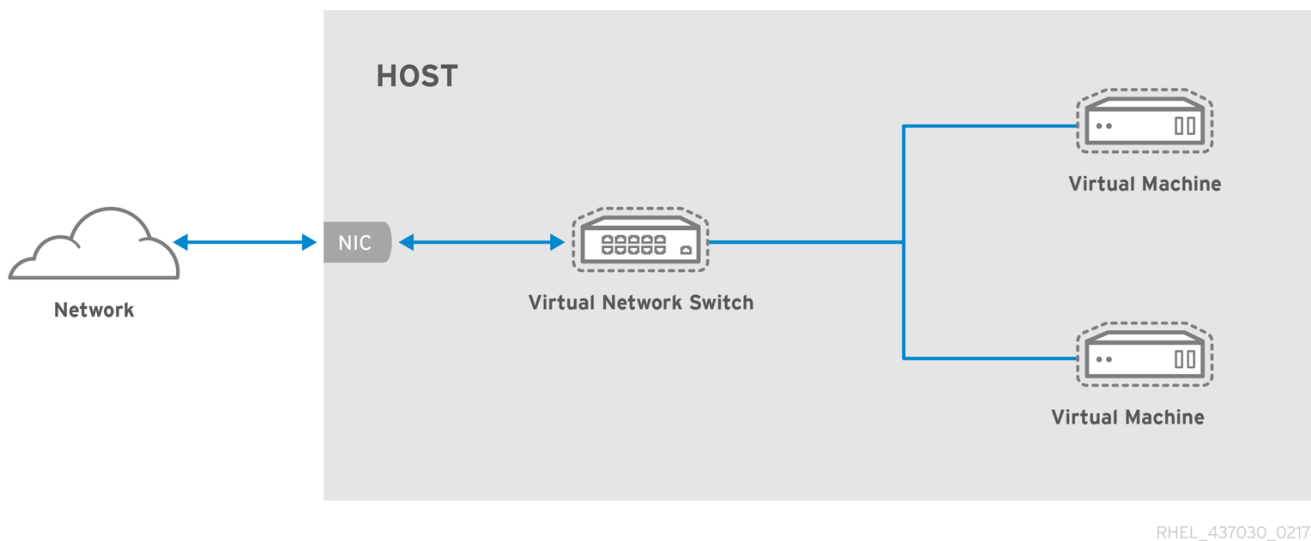


図18.1 ゲストを 2 つ持つ仮想ネットワークの例

Linux ホスト物理マシンのサーバーは、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ースとして表します。**libvirt** デーモン (**libvirtd**) を最初にインストールし、起動する際に、仮想ネットワークスイッチを表すデフォルトのネットワークインターフェースは **virbr0** になります。

上記の **virbr0** インターフェースは、他のインターフェースと同様 **ip** コマンドで表示させることができます。

```
$ 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 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
```

### 18.2. ブリッジモード

ブリッジモードの使用時には、すべてのゲスト仮想マシンがホスト物理マシンとして同一サブネット内に表示されます。同一の物理ネットワーク上にある他の物理マシンはこれらの仮想マシンを認識し、仮想マシンにアクセス可能です。ブリッジングは、OSI ネットワークモデルの第 2 層で動作します。

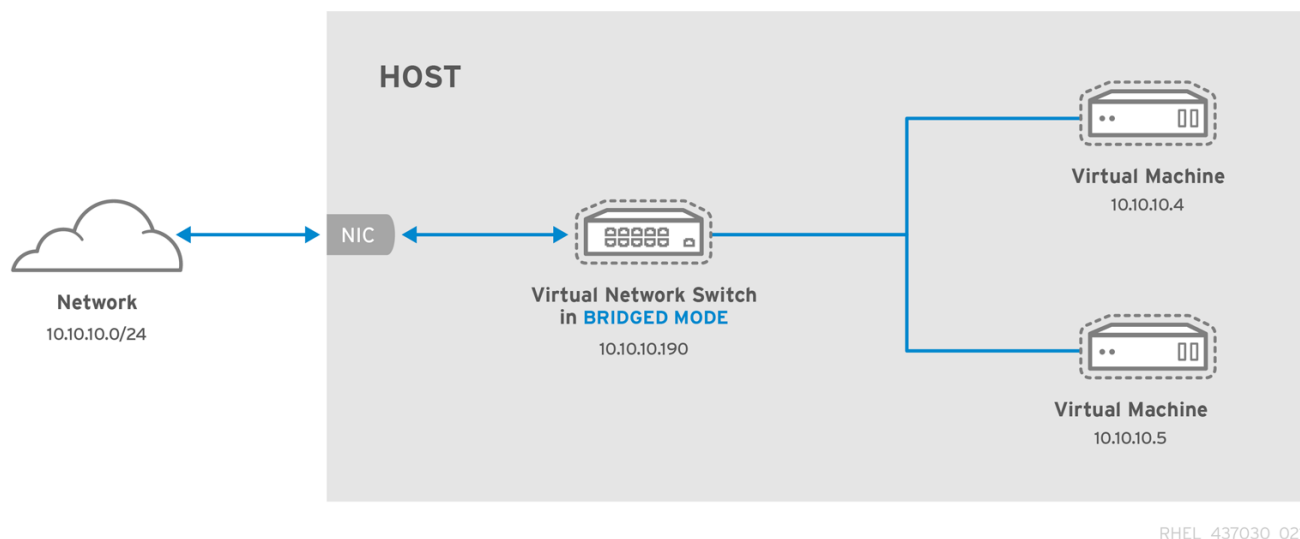


図18.2 ブリッジモードの仮想ネットワークスイッチ

複数の物理インターフェースは **ボンド** で結合し、ハイパーバイザー上で使用することが可能です。このボンドはブリッジに追加され、ゲスト仮想マシンもブリッジに追加されます。ただし、ボンディングドライバーにはいくつかの操作モードがあり、このなかで仮想ゲストマシンが使用されているブリッジと機能するものは、**2、3** しかありません。



#### 警告

ブリッジモードを使う場合、ゲスト仮想マシンと使用すべきボンディングモードは、**Mode 1、Mode 2、および Mode 4** のみです。**Modes 0、3、5、または 6** を使用すると、接続が切断される可能性があります。また、アドレス解決プロトコル (ARP) による監視は機能しないので、ボンディングモードの監視にはメディア非依存インタフェース (MII) による監視を使用してください。

ボンディングモードに関する詳細情報は、[関連するナレッジベースの記事](#) か『[Red Hat Enterprise Linux 7 ネットワークガイド](#)』を参照してください。

ブリッジネットワークモードを設定するのに使われる **bridge\_opts** パラメーターの詳細の説明については、『[Red Hat Virtualization 管理ガイド](#)』を参照してください。

## 18.3. NETWORK ADDRESS TRANSLATION

デフォルトでは、仮想ネットワークスイッチは **NAT** モードで動作します。**Source-NAT (SNAT)** や **Destination-NAT (DNAT)** ではなく **IP マスカレード** を使用します。IP マスカレードを使用すると、接続しているゲストが外部ネットワークとの通信にホスト物理マシンの IP アドレスを使用できるようになります。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、デフォルトではホスト物理マシンの外部にあるコンピューターはホスト内部にあるゲストと通信できません。これを以下の図で示します。

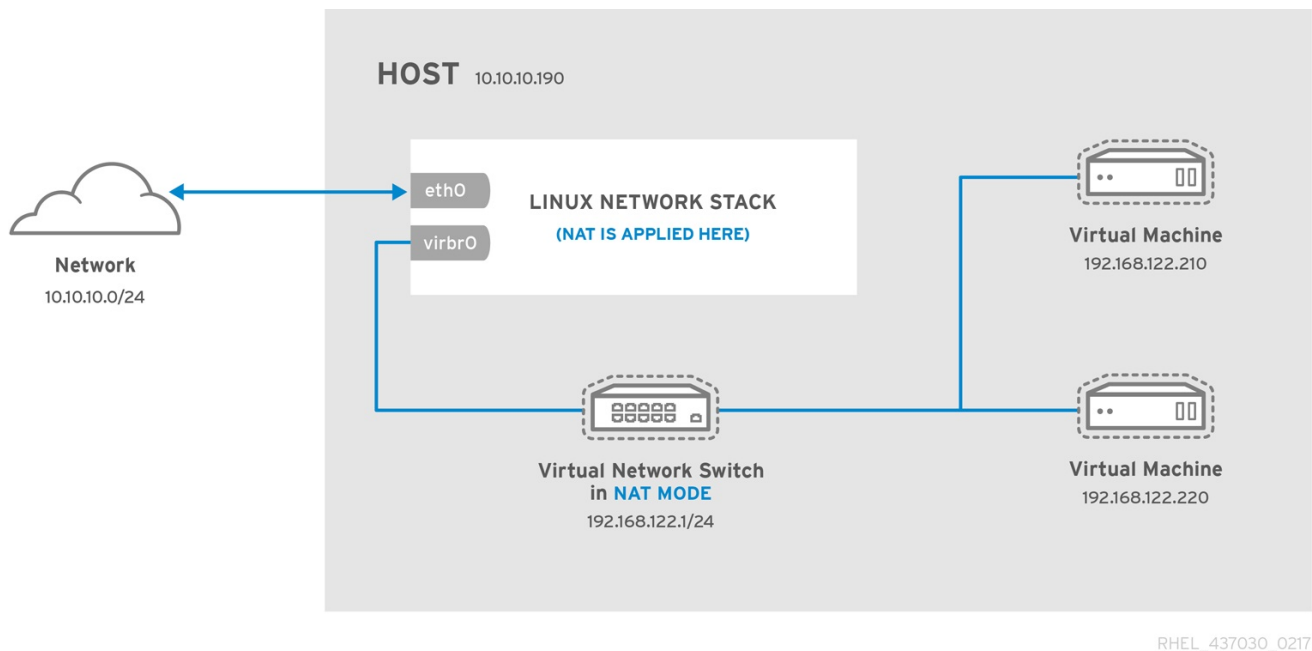


図18.3 ゲストを 2 つ持つ仮想ネットワークスイッチで NAT を使用している例



#### 警告

仮想ネットワークスイッチは **iptables** のルールで設定された **NAT** を使用します。スイッチが稼働したままの状態ではこれらのルールを編集することは推奨できません。正しくないルールが原因でスイッチが通信を行えなくなる恐れがあります。

スイッチが実行されていない場合、以下を実行して、ポートのマスカレード範囲を作成するために、**forward mode NAT** のパブリック IP 範囲を設定することができます。

```
# iptables -j SNAT --to-source [start]-[end]
```

## 18.4. DNS と DHCP

IP 情報は DHCP 経由でゲストに割り当てることができます。この割り当てを行うためアドレスのプールを仮想ネットワークスイッチに割り当てることができます。**libvirt** は **dnsmasq** プログラムを使用してこれを行います。**dnsmasq** のインスタンスは、このインスタンスを必要とする仮想ネットワークスイッチに自動的に設定され、起動されます。

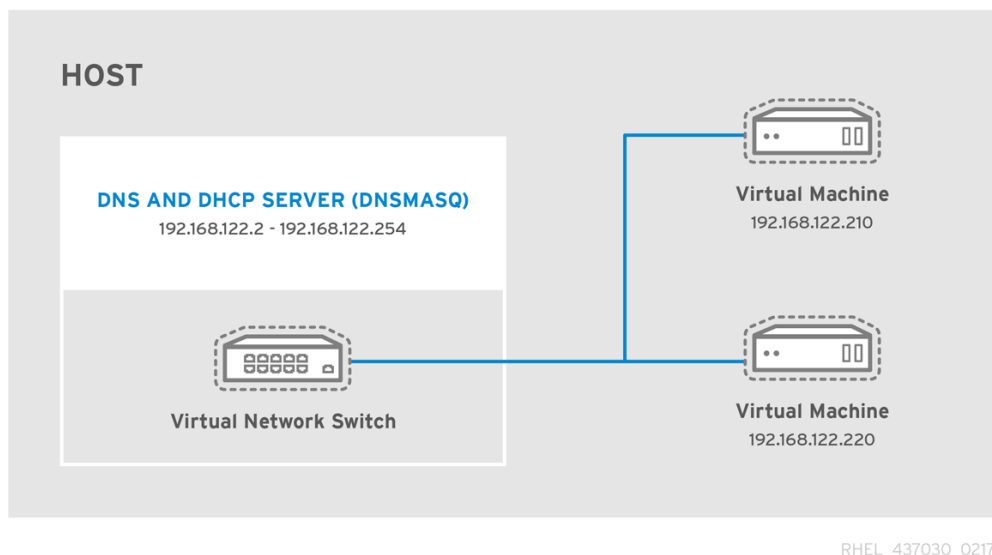


図18.4 dnsmasq を実行している仮想ネットワークスイッチ

## 18.5. ルーティングモード

ルーティングモードを使用する場合、仮想スイッチはホスト物理マシンにつながっている物理的な LAN に接続され、NAT を使わずにトラフィックの受け渡しを行います。仮想スイッチによりすべてのトラフィックが検査され、またネットワークパケット内に含まれる情報がルーティングを決定するために使用されます。このモードを使用すると、仮想マシンはすべてそれ自体のサブネット内に存在することになり、仮想スイッチ経由でルーティングが行われます。ただし、手作業で物理的なルーター設定を行わないと、物理的なネットワーク上に存在する他のホスト物理マシンからはこれらの仮想マシンは認識されないため、アクセスすることもできません。このため、このモードが常に理想的であるとは言えません。ルーティングモードは OSI ネットワークモデルの第 3 層で動作します。

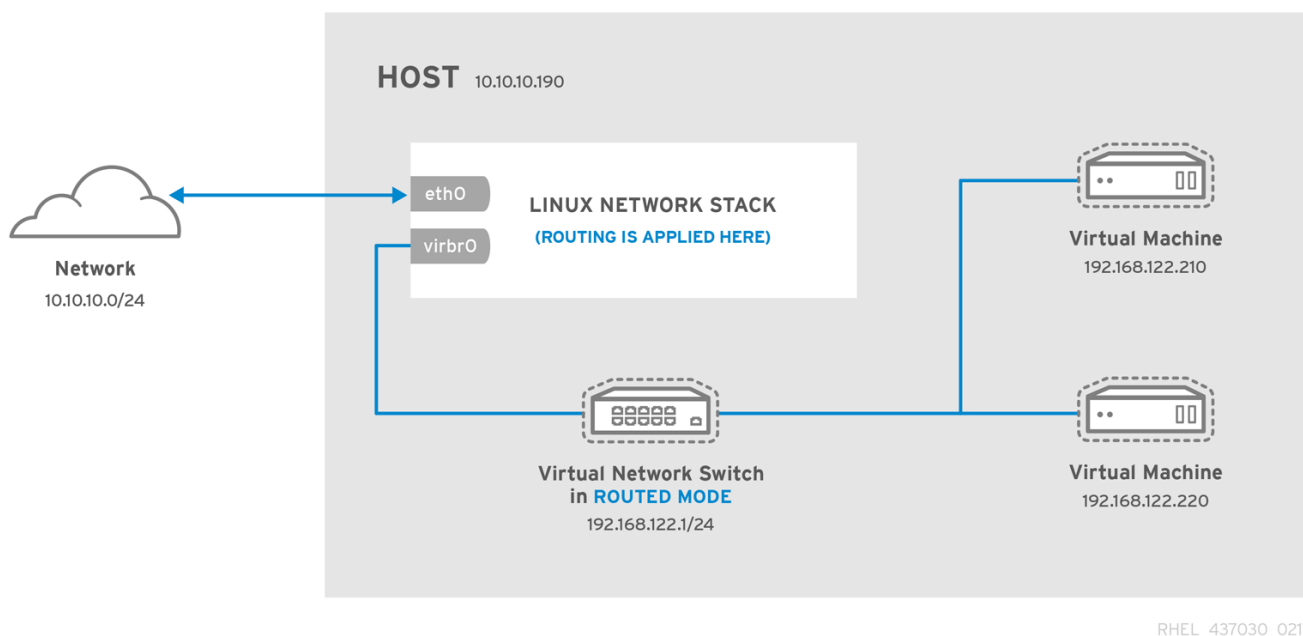


図18.5 ルーティングモードの仮想ネットワークスイッチ

## 18.6. 隔離モード

隔離モードを使用する場合、仮想スイッチに接続されているゲスト同士の通信およびホスト物理マシンとの通信は可能ですが、トラフィックはホスト物理マシンの外側へは通過していきません。また、ホスト物理マシンの外側からのトラフィックを受け取ることもできません。このモードで dnsmasq を使



用するには、DHCP などの基本的な機能が必要になります。ただし、このネットワークを物理的なネットワークと切り離しても、DNS 名の解決は行われます。したがって、DNS 名は解決できるのに ICMP エコー要求 (ping) のコマンドは失敗するといった状況になる可能性があります。

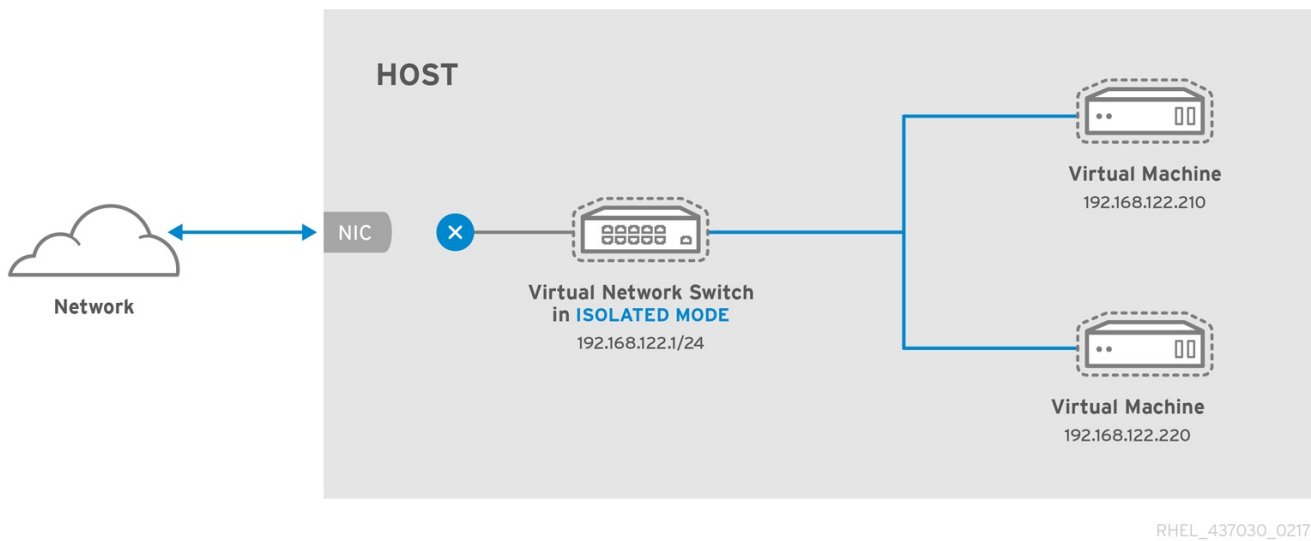


図18.6 隔離モードの仮想ネットワークスイッチ

## 18.7. デフォルト設定

libvirt デーモン (**libvirt**) の初回インストール時には、NAT モードの仮想ネットワークスイッチ初期設定が含まれます。この設定を使用すると、インストールしているゲストがホスト物理マシンを経由して外部ネットワークと通信できるようになります。以下の図は、この **libvirt** のデフォルト設定を表しています。

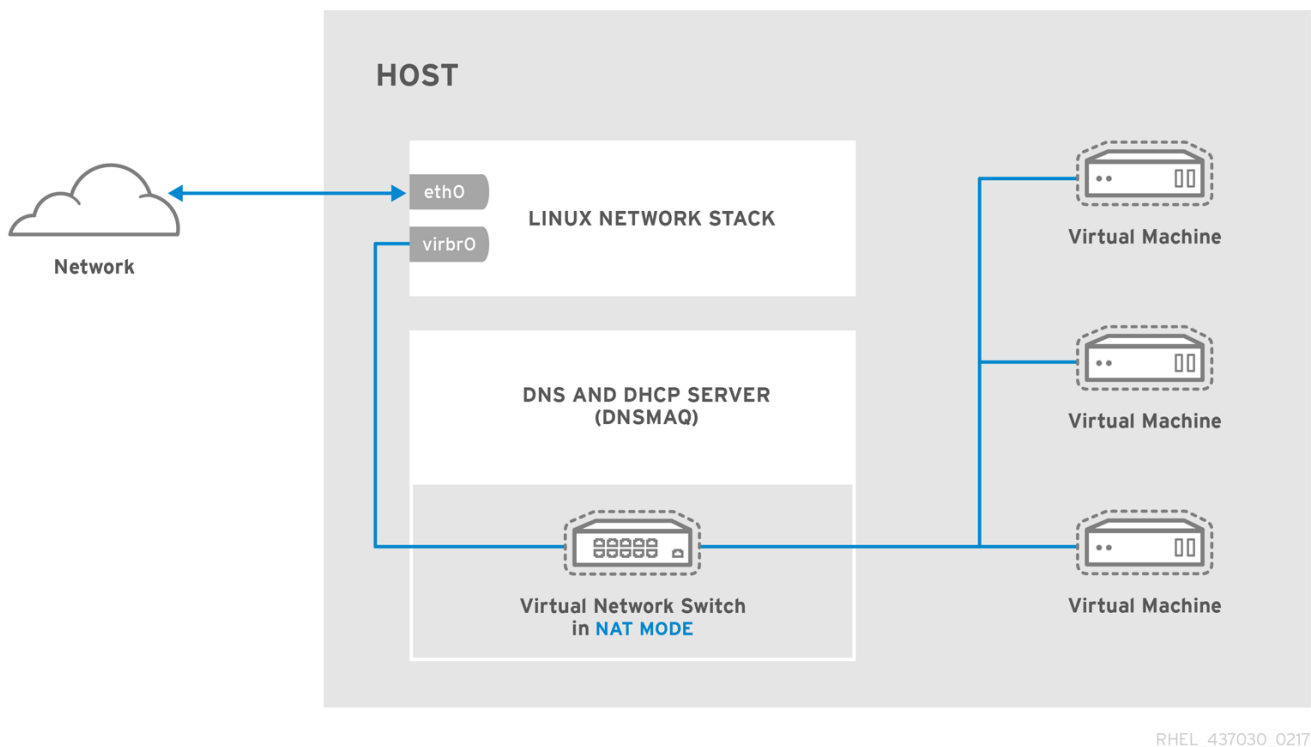


図18.7 libvirt のデフォルトネットワーク設定



## 注記

仮想ネットワークを特定の物理インターフェースに制限することができます。これは、複数のインターフェース (**eth0**、**eth1**、**eth2** など) を持つ物理システムの場合に役立ちます。また、ルーティングモードおよび NAT モードの場合にのみ役立ち、**dev=<interface>** オプションで指定できます。また、新しい仮想ネットワークを作成する際に **virt-manager** で指定することもできます。

## 18.8. 一般的な事例

本セクションでは、複数の異なるネットワークモードを紹介し、その使用例を説明します。

### 18.8.1. ブリッジモード

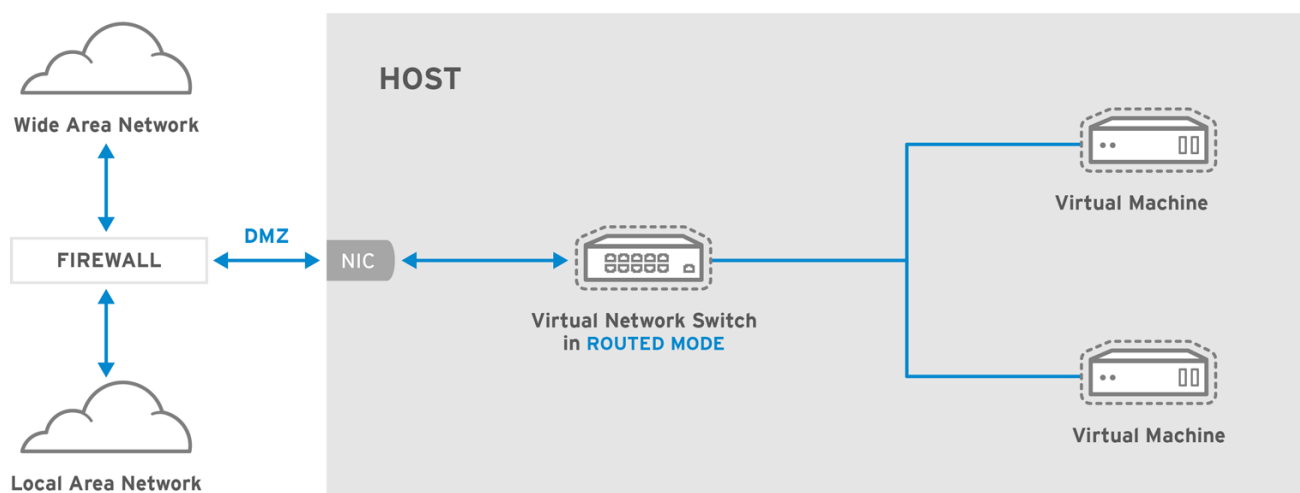
ブリッジモードは、OSI モデルの第 2 層で作動します。このモードの使用時には、ゲスト仮想マシンすべてが同一サブネット上でホスト物理マシンとして表示されます。ブリッジモードの一般的な使用例には、以下のものがあります。

- ホスト物理マシンとともに既存ネットワーク内でゲスト仮想マシンをデプロイし、エンドユーザーに対して仮想マシンと物理マシンの違いを明らかにする。
- 既存の物理ネットワーク設定に変更を加えずにゲスト仮想マシンをデプロイする。
- 既存の物理ネットワークに容易にアクセス可能である必要があるゲスト仮想マシンをデプロイする。ゲスト仮想マシンを、DHCP などの既存のブロードキャストドメイン内でサービスにアクセスする必要のある物理ネットワーク上に配置する。
- ゲスト仮想マシンを VLAN が使用されている既存のネットワークに接続する。

### 18.8.2. ルーティングモード

#### DMZ

安全を確保する目的で制御されたサブネットワーク内に 1 ノードまたは複数ノードを配置しているネットワークの例を見めます。こうした特殊なサブネットワークの導入は一般的に行われており、このサブネットワークは **DMZ** と呼ばれています。レイアウトの詳細は、以下の図を参照してください。



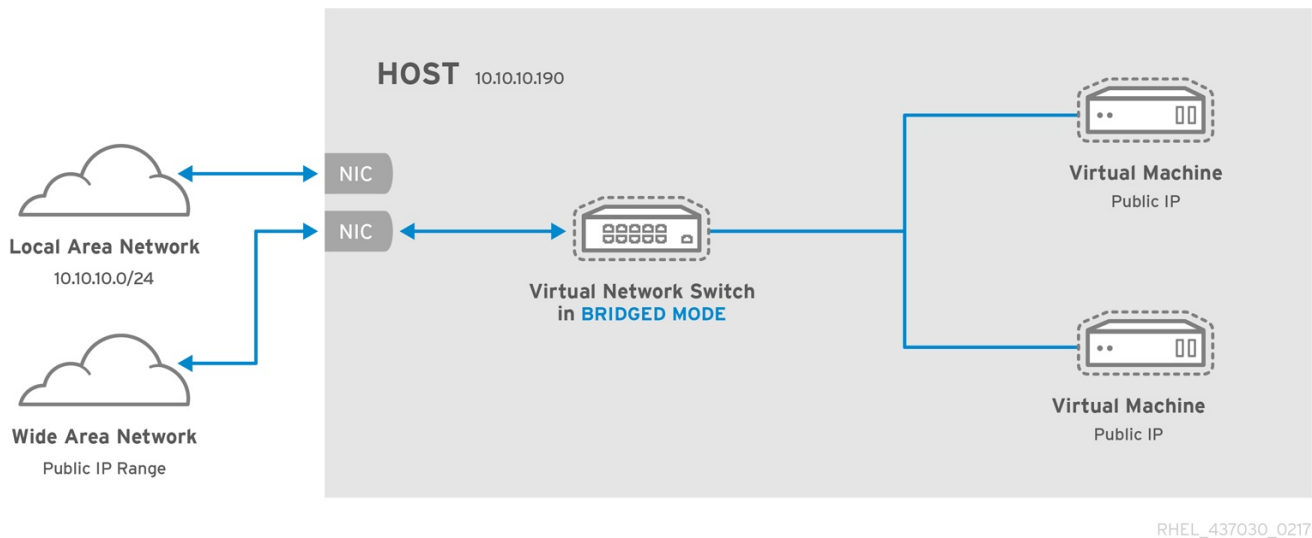
RHEL\_437030\_0217

図18.8 DMZ の設定例

DMZ 内のホスト物理マシンは、通常 WAN (外部) ホスト物理マシンおよび LAN (内部) ホスト物理マシンにサービスを提供します。管理や運営の方法は場所ごとにまたセキュリティーや信頼できるレベルによって異なる点を考慮すると、さまざまな場所から DMZ 内のホストにアクセスする必要のあるこのような環境ではルーティングモードが最適な設定となります。

### 仮想サーバーのホスティング

仮想サーバーホスティング会社を例に見てみましょう。この仮想サーバーホスティング会社は複数のホスト物理マシンを有し、それぞれのホスト物理マシンに物理的なネットワーク接続を 2 つずつ持たせています。一方のインターフェースは管理およびアカウントिंगに使用し、他方は仮想マシンの接続に使用しています。各ゲストには独自のパブリック IP アドレスを持たせていますが、ホスト物理マシンはプライベートの IP アドレスを使用します。これはゲストの管理作業を内部の管理者に制限するためです。この状況をわかりやすく説明するため以下に図を示します。



RHEL\_437030\_0217

図18.9 仮想サーバーホスティングの設定例

### 18.8.3. NAT モード

NAT (Network Address Translation) モードはデフォルトのモードです。直接ネットワークの可視性を必要としないテストなどに使用することができます。

### 18.8.4. 隔離モード

隔離モードを使用すると、通信を行うことができるのは仮想マシン同士のみに限ることができます。仮想マシンは物理的なネットワークでの通信は行えません。

## 18.9. 仮想ネットワークの管理

システムで仮想ネットワークを設定する

1. **編集** メニューから **接続の詳細** を選択します。
2. **接続の詳細** メニューが開きます。**仮想ネットワーク** タブをクリックします。

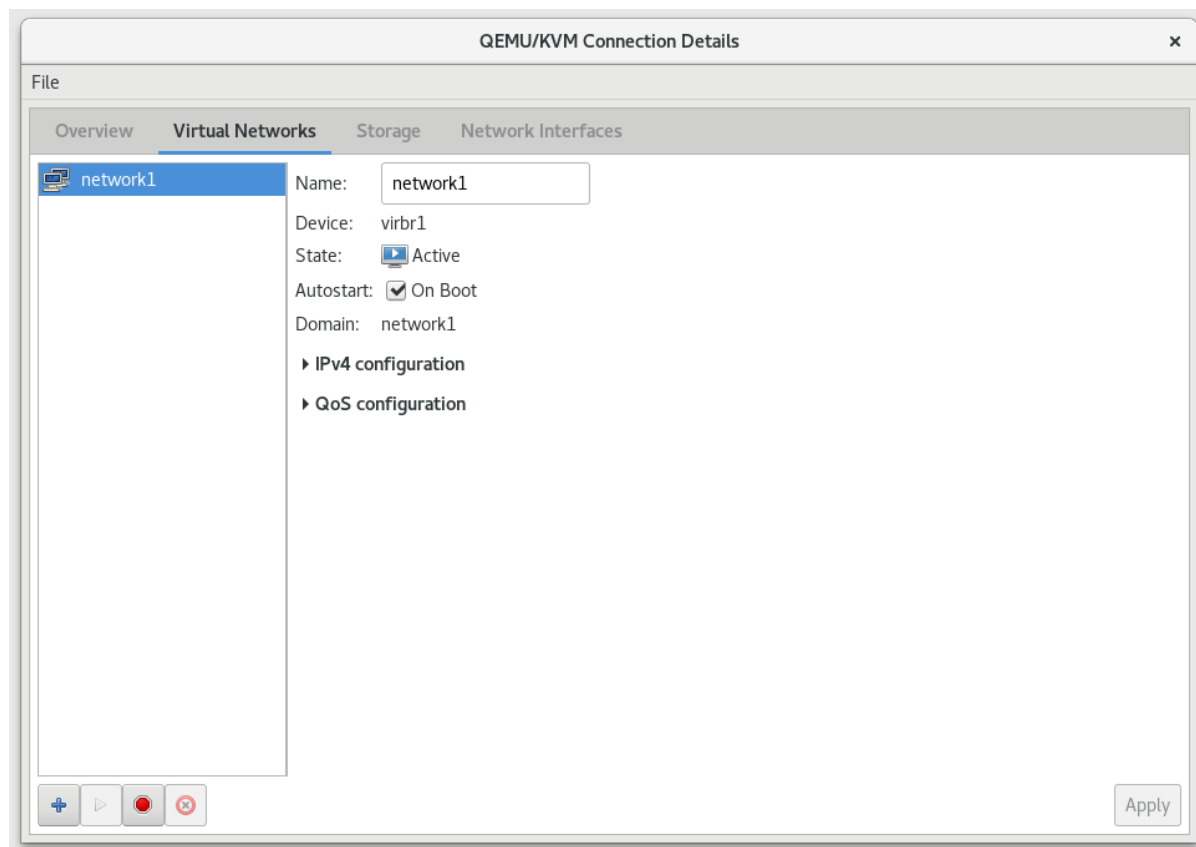


図18.10 仮想ネットワークの設定

3. メニュー左側のボックスに使用可能なネットワークがすべて表示されます。ボックス内のネットワークを選択して適宜編集を行い、仮想ネットワークの設定を修正することができます。

## 18.10. 仮想ネットワークの作成

仮想マシンマネージャー (virt-manager) を使用してシステム上に仮想ネットワークを作成する方法:

1. **接続の詳細** メニューから **仮想ネットワーク** タブを開きます。(+) マークのアイコンで表されている **ネットワークの追加** ボタンをクリックします。詳細は「[仮想ネットワークの管理](#)」を参照してください。

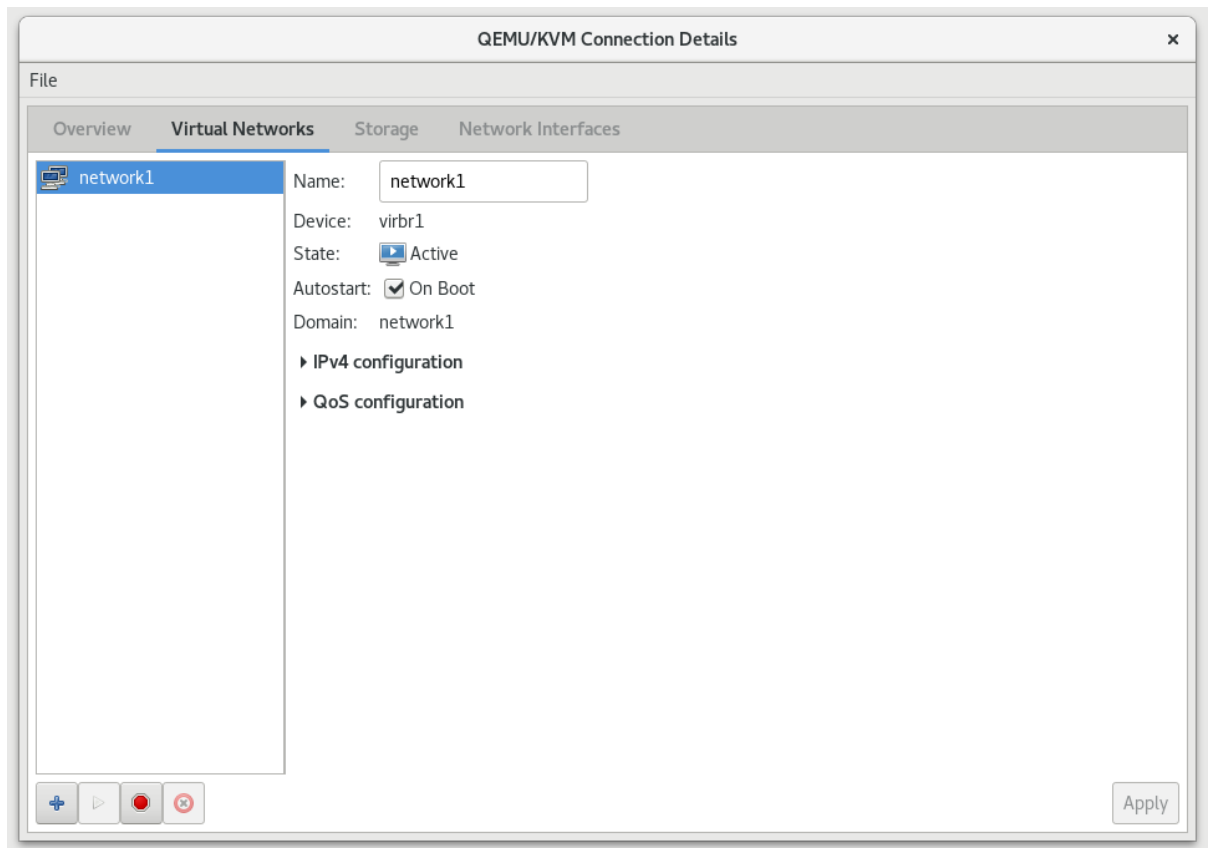
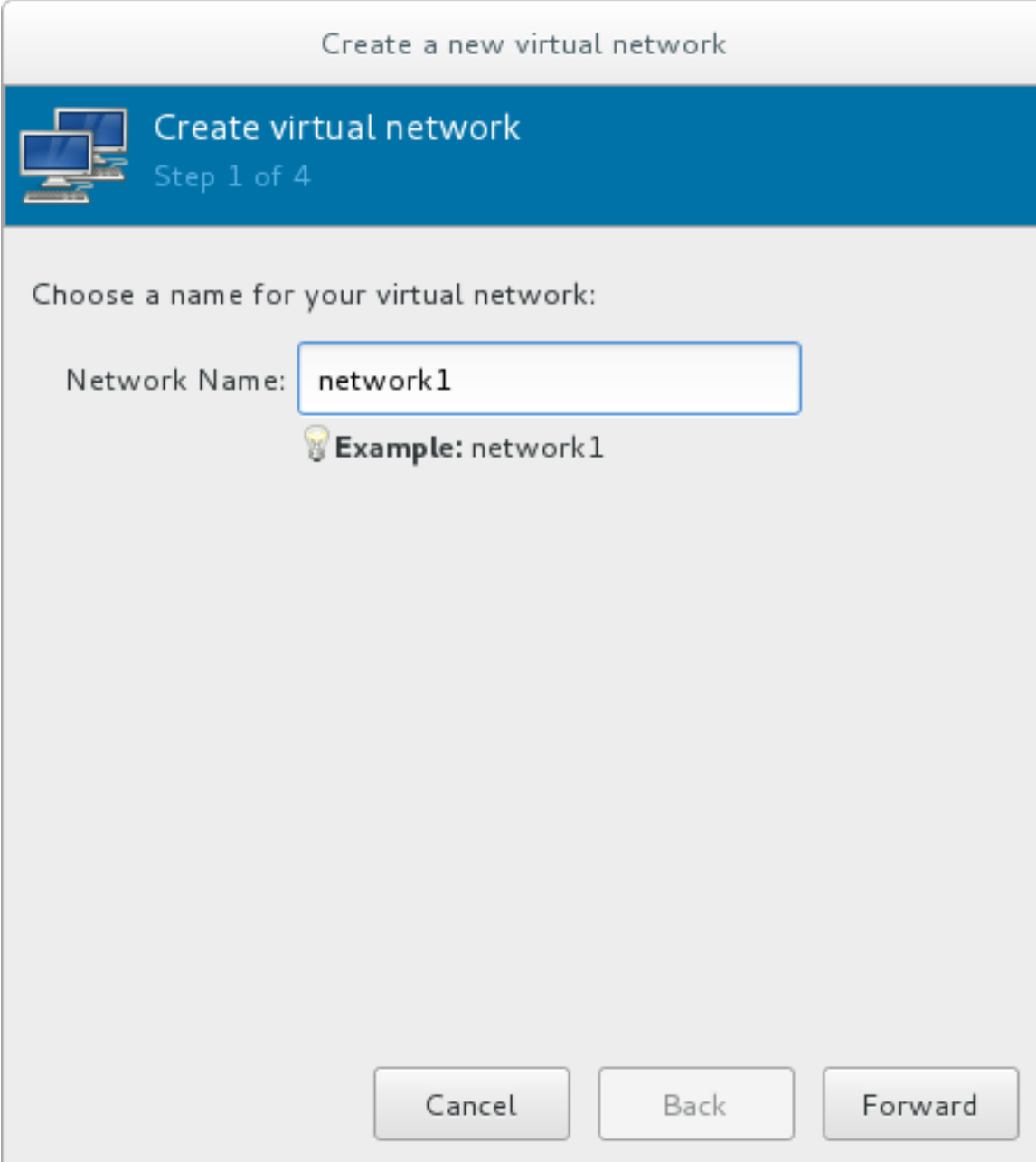


図18.11 仮想ネットワークの設定

新しい仮想ネットワークの作成ウィンドウが開きます。**進む** をクリックして先に進みます。




Create a new virtual network

Create virtual network  
Step 1 of 4

Choose a name for your virtual network:

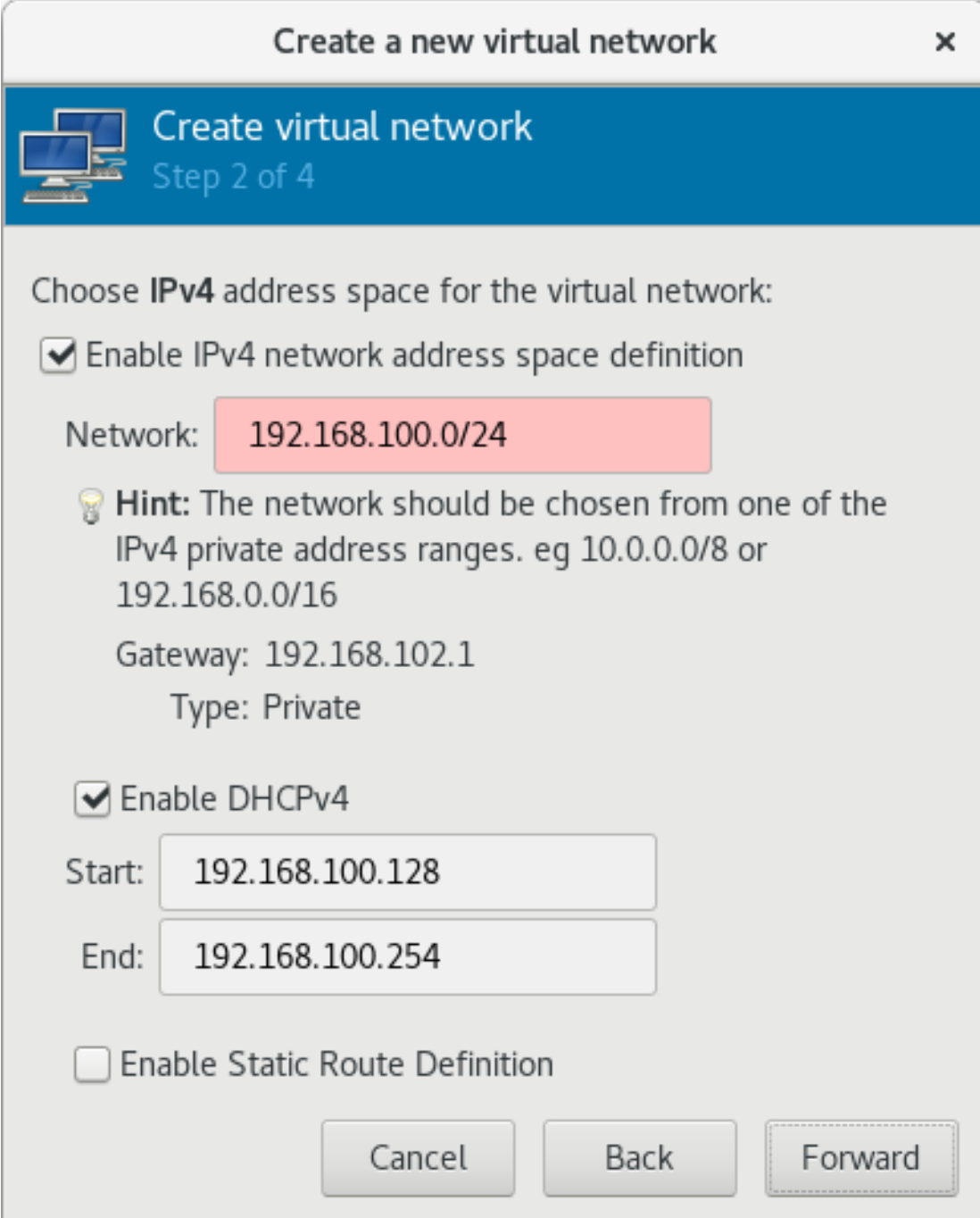
Network Name:

 **Example:** network1

Cancel Back Forward

図18.12 新規の仮想ネットワークの名前の入力

2. 仮想マシンの名前を入力して **進む** をクリックします。




Create a new virtual network

Create virtual network  
Step 2 of 4

Choose **IPv4** address space for the virtual network:

☒ Enable IPv4 network address space definition

Network: 192.168.100.0/24

 **Hint:** The network should be chosen from one of the IPv4 private address ranges. eg 10.0.0.0/8 or 192.168.0.0/16

Gateway: 192.168.102.1  
Type: Private

☒ Enable DHCPv4

Start: 192.168.100.128  
End: 192.168.100.254

☐ Enable Static Route Definition

Cancel Back Forward

図18.13 IPv4 アドレス空間の選択


3. **IPv4** ネットワークアドレス空間の定義を可能にする チェックボックスにチェックを入れます。

ネットワーク フィールドに仮想ネットワークの **IPv4** アドレス空間を入力します。

**DHCPv4** を有効化 チェックボックスにチェックを付けます。

IP アドレス範囲の **開始** と **終了** を指定して仮想ネットワークの **DHCP** 範囲を定義します。


Create a new virtual network

Create virtual network  
Step 2 of 4

Choose **IPv4** address space for the virtual network:

☒ Enable IPv4 network address space definition

Network:

 **Hint:** The network should be chosen from one of the IPv4 private address ranges. eg 10.0.0.0/8 or 192.168.0.0/16

Gateway: 192.168.100.1

Type: ?

☒ Enable DHCPv4

Start:

End:

☐ Enable Static Route Definition

図18.14 IPv4 アドレス空間の選択

**進む** をクリックして先に進みます。

- IPv6 を有効にする場合、**IPv6 ネットワークアドレス空間の定義を可能にする** にチェックを付けます。



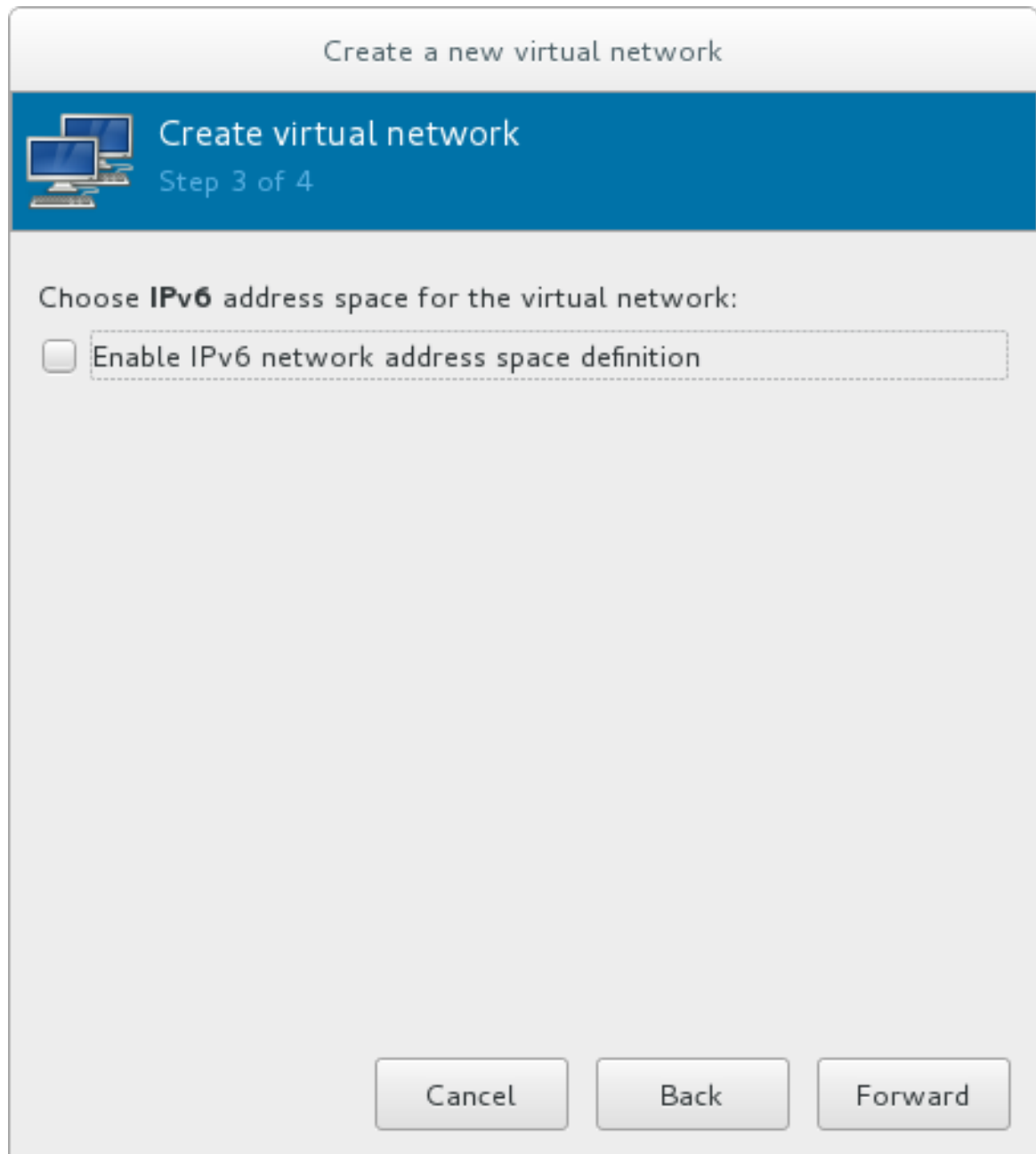



図18.15 IPv6の有効化

追加のフィールドが「新しい仮想ネットワークを作成」ウィンドウに表示されます。

Create a new virtual network




Create virtual network  
Step 3 of 4

Choose **IPv6** address space for the virtual network:

☒ Enable IPv6 network address space definition

Network:

 **Note:** The network could be chosen from one of the IPv6 private address ranges. eg FC00::/7. The prefix must be **64**.  
A typical IPv6 network address will look something like:  
fd00:e81d:a6d7:55::/64  
Gateway: fd00:100::1  
Type: ?

☐ Enable DHCPv6

☐ Enable Static Route Definition

Cancel

Back

Forward


図18.16 IPv6 の設定

ネットワーク フィールドに IPv6 アドレスを入力します。

5. DHCPv6 を有効にする場合、**DHCPv6 を有効化** チェックボックスにチェックを付けます。

追加のフィールドが「新しい仮想ネットワークを作成」ウィンドウに表示されます。


Create a new virtual network

 **Create virtual network**  
Step 3 of 4

Choose **IPv6** address space for the virtual network:

☒ Enable IPv6 network address space definition

Network:

 **Note:** The network could be chosen from one of the IPv6 private address ranges. eg FC00::/7. The prefix must be **64**. A typical IPv6 network address will look something like:  
fd00:dead:beef:55::/64

Gateway:  
Type: Private

☒ Enable DHCPv6

Start:

End:

☐ Enable Static Route Definition


図18.17 DHCPv6 の設定

(オプション) DHCPv6 範囲の開始と終了を編集します。

6. 静的ルートの定義を有効にする場合、**スタティックルートの定義を有効化** チェックボックスにチェックを付けます。

追加のフィールドが「新しい仮想ネットワークを作成」ウィンドウに表示されます。


Create a new virtual network

Create virtual network  
Step 3 of 4

Choose **IPv6** address space for the virtual network:

☒ Enable IPv6 network address space definition

Network:

 **Note:** The network could be chosen from one of the IPv6 private address ranges. eg FC00::/7. The prefix must be **64**. A typical IPv6 network address will look something like:  
fd00:dead:beef:55::/64

Gateway:  
Type: Private

☒ Enable DHCPv6

Start:

End:

☒ Enable Static Route Definition

to Network:

via Gateway:

図18.18 静的ルートの定義

該当するフィールドに、ネットワークへのルートに使用するネットワークアドレスおよびゲートウェイを入力します。

**進む (Forward)** をクリックします。

7. 仮想ネットワークを物理ネットワークに接続する方法を選択します。

Create a new virtual network

Create virtual network  
Step 4 of 4

Connected to a **physical network**:

☒ Isolated virtual network  
☐ Forwarding to physical network

Destination: Any physical device ▼

Mode: NAT ▼

☐ Enable IPv6 internal routing/networking

If an IPv6 network address is **not** specified, this will enable IPv6 internal routing between virtual machines. By default, IPv4 internal routing is enabled.

DNS Domain Name: network1

Cancel Back Finish

図18.19 物理ネットワークへの接続

仮想ネットワークを分離する場合、**隔離された仮想ネットワーク** ラジオボタンが選択されていることを確認します。

仮想ネットワークが物理ネットワークに接続できるようにするには、**物理ネットワーク**に**フォワード**を選択してから、**宛先**を**いずれかの物理デバイス**または特定の物理デバイスにするかどうかを選択します。さらに、**モード**を**NAT**または**ルーティング**にするかどうかを選択します。

仮想ネットワーク内で IPv6 ルーティングを有効にするには、**IPv6 の内部ルーティング/内部ネットワークを有効にする** チェックボックスにチェックを付けます。

仮想ネットワークの DNS ドメイン名を入力します。

**完了** をクリックし、仮想ネットワークを作成します。

- これで新しい仮想ネットワークが**接続の詳細** ウィンドウの **仮想ネットワーク** タブに表示されるようになります。

## 18.11. 仮想ネットワークのゲストへの接続

仮想ネットワークをゲストに接続する

1. **仮想マシンマネージャー** ウィンドウ内で、ネットワークを割り当てるゲストを強調表示します。

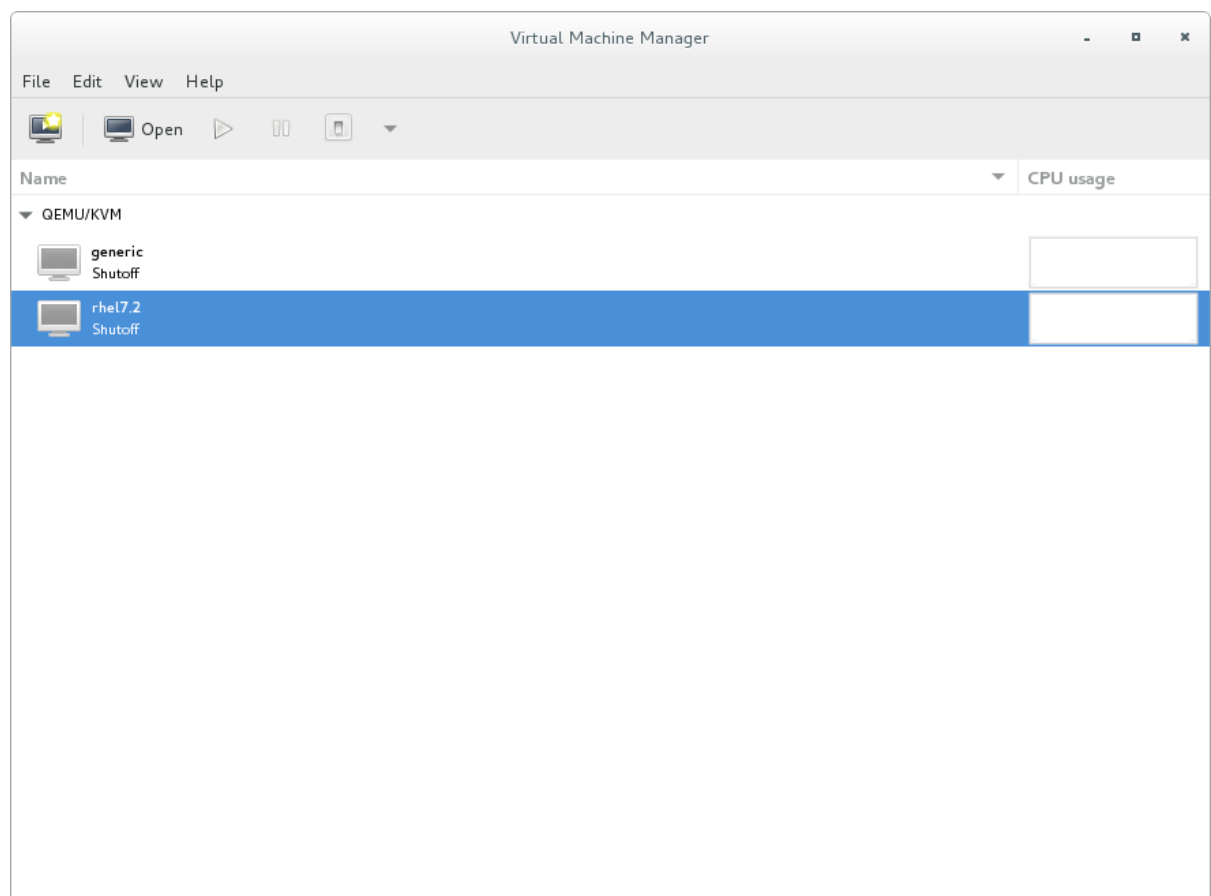


図18.20 表示する仮想マシンの選択

2. 仮想マシンマネージャーの **編集** メニューから **仮想マシンの詳細** を選択します。
3. 仮想マシンの詳細ウィンドウにある **ハードウェアを追加** ボタンをクリックします。
4. **新しい仮想ハードウェアを追加** ウィンドウ内の左側のペインから **ネットワーク** を選択し、**ネットワークソース** メニューでネットワーク名 (この例では **network1**) を選択します。必要に応じて **MAC アドレス** を修正し、**デバイスのモデル** を選択します。**完了** をクリックします。

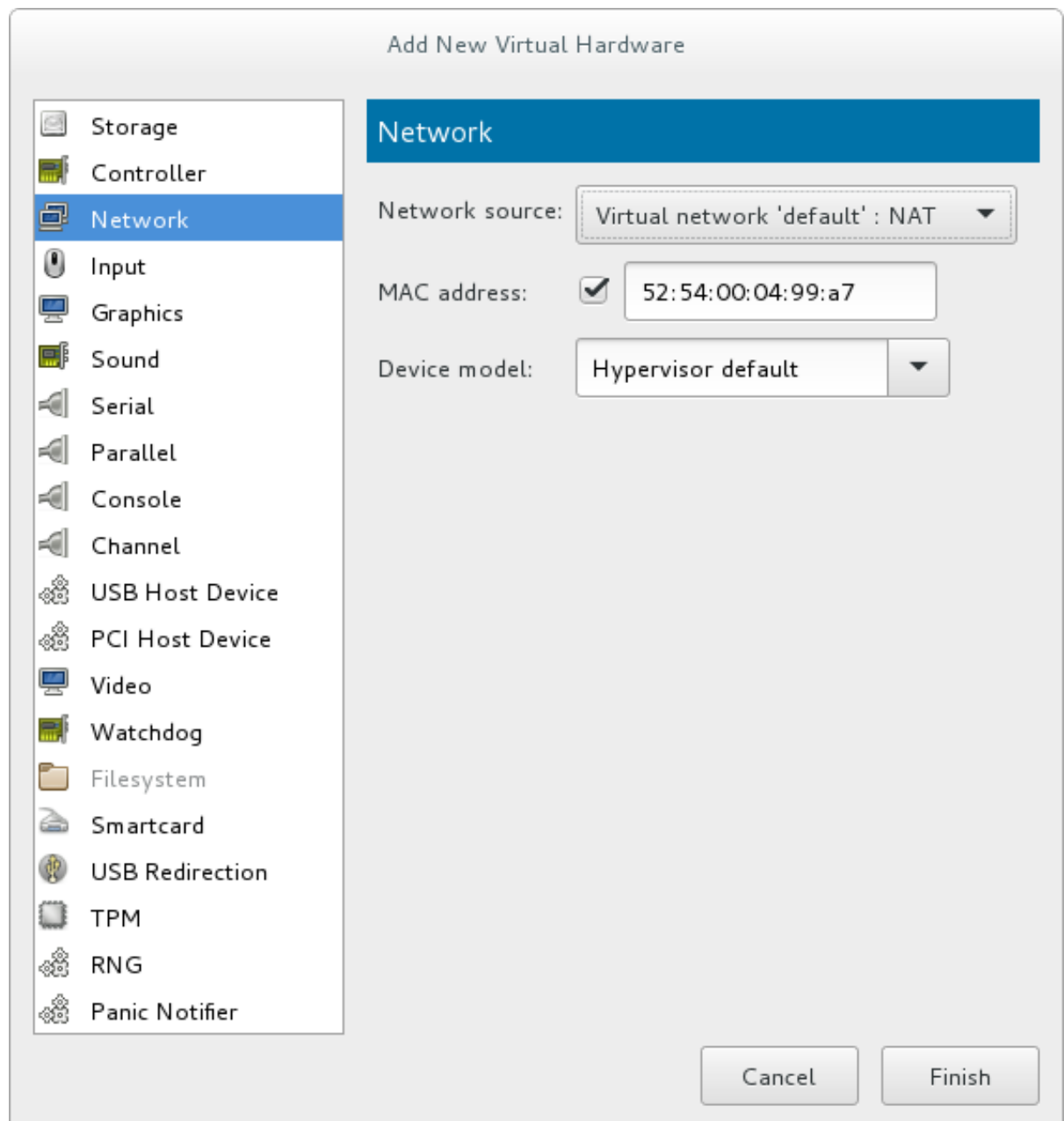


図18.21 新規の仮想ハードウェアの追加ウィンドウからネットワークを選択する

- これで新しいネットワークが仮想ネットワークインターフェースとして表示されるようになりました。ゲストの起動時に提供されるようになります。

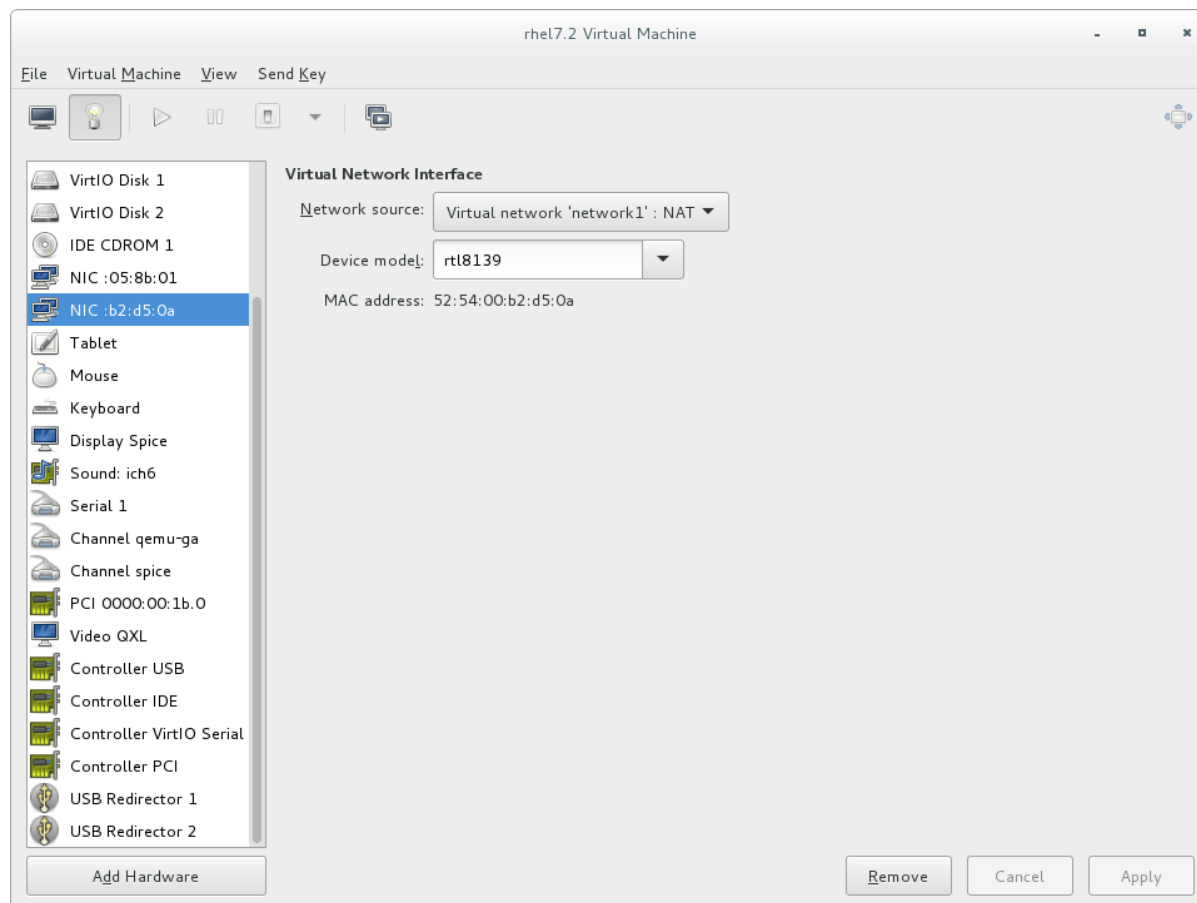


図18.22 ゲストのハードウェア一覧に表示される新しいネットワーク

## 18.12. 仮想 NIC を直接物理インターフェースへ接続

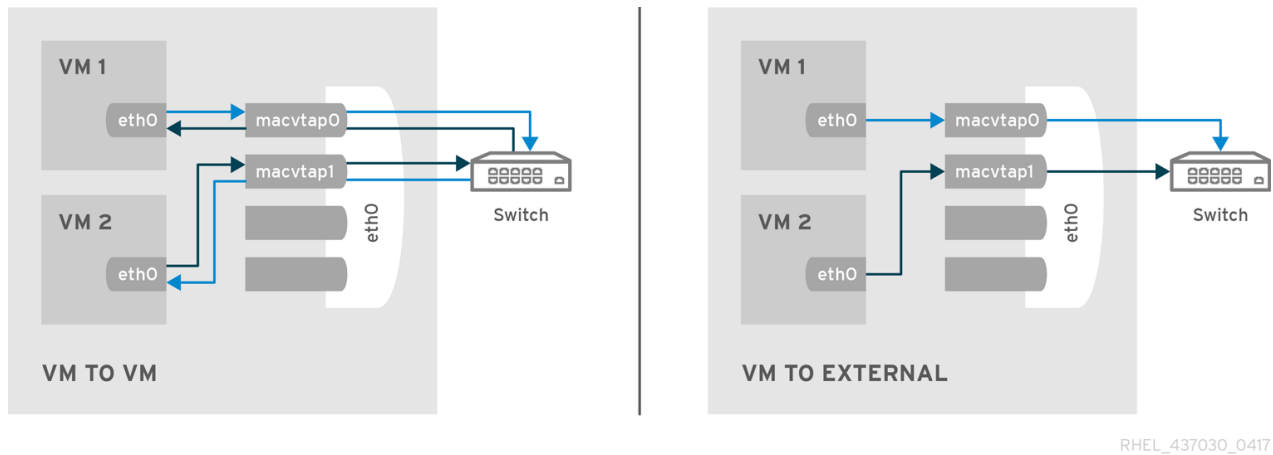
デフォルトの NAT 接続の代わりに、*macvtap* ドライバーを使ってゲストの NIC を直接ホストマシンの特定の物理インターフェースに接続することができます。これを、**デバイス割り当て** (パススルーとも呼ばれる) と混同しないようにしてください。Macvtap 接続には以下のモードがあり、それぞれメリットとユースケースが異なります。

### 物理インターフェースの配信モード

#### VEPA

仮想イーサネットポートアグリゲーター (VEPA) モードでは、ゲストからのパケットはすべて外部スイッチに送信されます。これにより、ユーザーはゲストのトラフィックを強制的にスイッチ経由にすることができます。VEPA モードが正常に機能するためには、外部スイッチは **ヘアピンモード** にも対応している必要があります。これにより、送信元ゲストと同じホストマシン上にあるゲスト宛のパケットが、外部スイッチによりホストに戻されます。



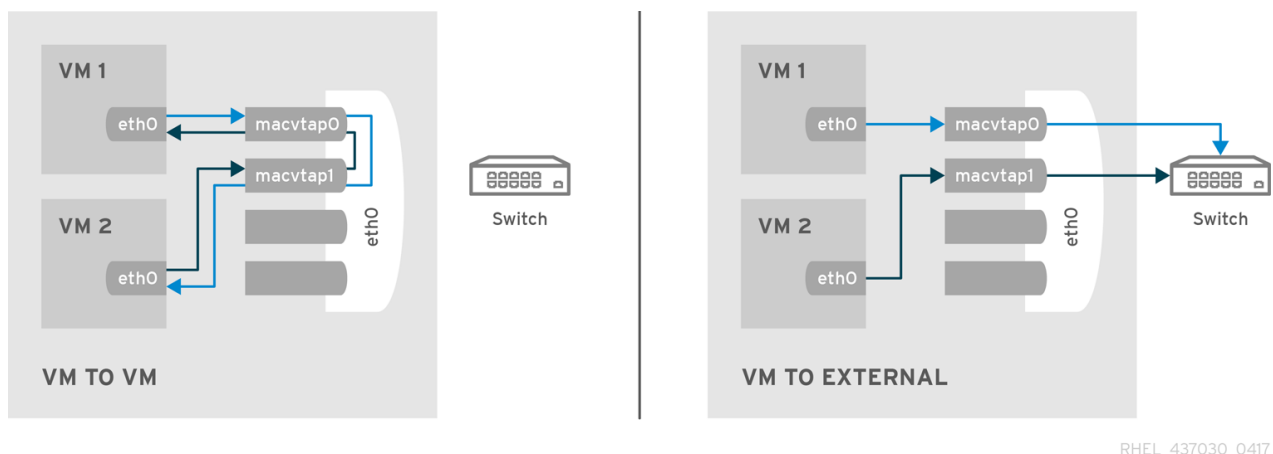


RHEL\_437030\_0417

図18.23 VEPA モード

**bridge**

送信元ゲストと同じホストマシン上にあるゲスト宛のパケットは、直接宛先の **macvtap** デバイスに配信されます。直接配信が正常に機能するためには、送信元デバイスと宛先のデバイスの両方が **bridge** モードになければなりません。どちらかのデバイスが **VEPA** モードにある場合は、ヘアピン機能に対応した外部スイッチが必要です。

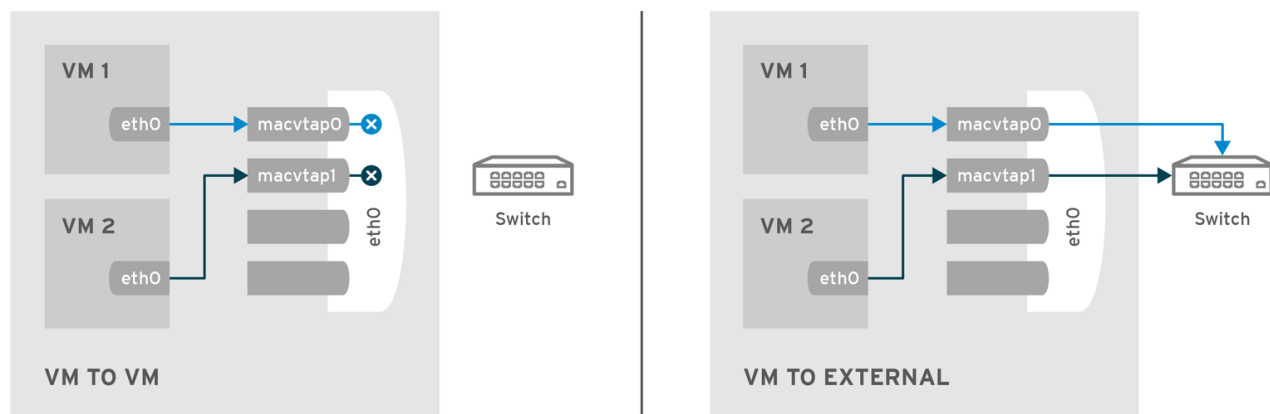


RHEL\_437030\_0417

図18.24 Bridge モード

**private**

すべてのパケットは外部スイッチに送信されます。それらのパケットが同じホストマシン上にある宛先のゲストに配信されるのは、それらが外部ルーターまたはゲートウェイ経由で送信され、それらがパケットをホストに戻す場合のみです。**private** モードは、同一ホスト上にある個々のゲストが互いに通信するのを防ぐために使うことができます。送信元または宛先デバイスのいずれかが **private** モードにある場合に、この手順が実行されます。

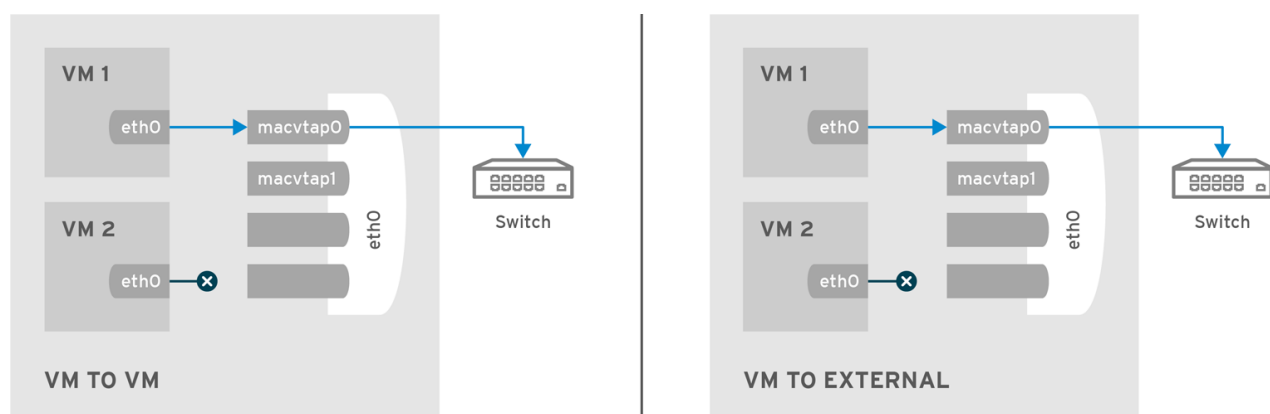


RHEL\_437030\_0417

図18.25 Private モード

### passthrough

この機能では、物理インターフェースデバイスまたは **SR-IOV** 仮想機能 (VF) を、移行機能を損なうことなく直接ゲストに接続します。すべてのパケットは、直接目的のネットワークデバイスに送信されます。**passthrough** モードでは、ゲスト間でネットワークデバイスを共有することはできません。従って、1つのゲストにパススルーできるネットワークデバイスは1つだけです。



RHEL\_437030\_0417

図18.26 Passthrough モード

Macvtap の設定は、ドメインの XML ファイルを変更して、または **virt-manager** インターフェースを使用して行います。

#### 18.12.1. ドメイン XML を使用した macvtap の設定

ゲストのドメイン XML ファイルを開き、以下のように **<devices>** 要素を変更します。

```
<devices>
...
<interface type='direct'>
  <source dev='eth0' mode='vepa' />
</interface>
</devices>
```

直接接続したゲスト仮想マシンのネットワークアクセスは、ホスト物理マシンの物理インターフェースが接続されているハードウェアスイッチで管理することができます。

スイッチが IEEE 802.1Qbg 標準に設定されている場合は、インターフェースに以下に示すように追加パラメーターを持たせることができます。virtualport 要素のパラメーターについては IEEE 802.1Qbg 標準の記載をご覧ください。その値についてはネットワーク固有となるため、ネットワーク管理者にお問い合わせください。802.1Qbg では、VSI (Virtual Station Interface) は仮想マシンの仮想インターフェースのことを指します。なお、IEEE 802.1Qbg では、VLAN ID にゼロ以外の値を設定する必要があります。

## Virtual Station Interface のタイプ

### managerid

VSI Manager ID で VSI タイプとインスタンス定義が含まれるデータベースを識別します。これは整数の値で、0 の値は予約されています。

### typeid

VSI Type ID でネットワークアクセスの特性を示す VSI タイプを識別します。VSI タイプは通所ネットワーク管理者によって管理されます。これは整数の値です。

### typeidversion

VSI Type Version では VSI Type の複数のバージョンを許可します。これは整数の値です。

### instanceid

VSI Instance ID は、VSI インスタンス (つまり、仮想マシンの仮想インターフェース) の作成時に生成されます。これは、グローバルに固有な識別子です。

### profileid

プロファイル ID には、このインターフェースに適用されるポートプロファイル名が含まれます。この名前は、ポートプロファイルのデータベースによってポートプロファイルからネットワークパラメーターに解決され、それらのネットワークパラメーターがこのインターフェースに適用されます。

4 種類のそれぞれのタイプの設定は、ドメインの XML ファイルを変更して行います。このファイルを開いたら、以下のようにモードの設定を変更します。

```
<devices>
...
<interface type='direct'>
  <source dev='eth0.2' mode='vepa' />
  <virtualport type="802.1Qbg">
    <parameters managerid="11" typeid="1193047" typeidversion="2"
instanceid="09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f" />
  </virtualport>
</interface>
</devices>
```

プロファイル ID を以下に示します。

```
<devices>
...
<interface type='direct'>
  <source dev='eth0' mode='private' />
  <virtualport type='802.1Qbh'>
    <parameters profileid='finance' />
```

```

</virtualport>
</interface>
</devices>
...

```

### 18.12.2. virt-manager を使用した macvtap の設定

仮想ハードウェアの詳細ウィンドウを開き ⇒ メニューで **NIC** を選択し ⇒ ネットワークソースにホストデバイス **name: macvtap** を選択し ⇒ 目的のソースモードを選択します。

続いて、**仮想ポート** サブメニューで、**Virtual Station Interface** のタイプを設定することができます。

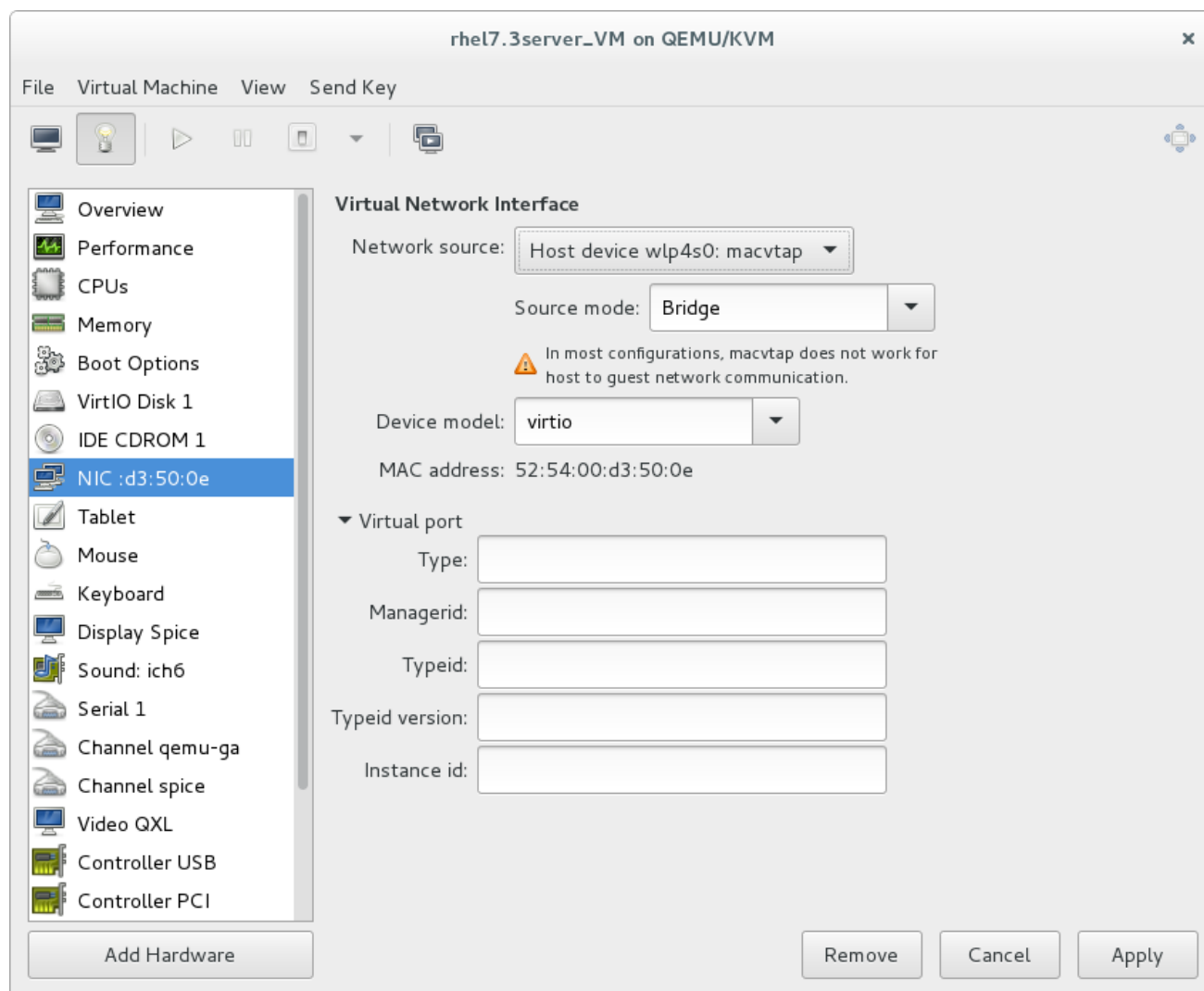


図18.27 virt-manager で macvtap を設定

## 18.13. 仮想 NIC に接続しているネットワークブリッジまたはホスト物理マシンの動的な変更

このセクションでは、ゲスト仮想マシンを稼働したままで安全性を確保しながら、そのゲスト仮想マシンの vNIC をあるブリッジから別のブリッジに移動する方法について説明します。

1. 次のような設定でゲスト仮想マシンを用意します。

```

<interface type='bridge'>
  <mac address='52:54:00:4a:c9:5e' />

```

```
<source bridge='virbr0'/>
<model type='virtio'/>
</interface>
```

2. インターフェースの更新用に XML ファイルを準備します。

```
# cat br1.xml
```

```
<interface type='bridge'>
  <mac address='52:54:00:4a:c9:5e'/>
  <source bridge='virbr1'/>
  <model type='virtio'/>
</interface>
```

3. ゲスト仮想マシンを起動し、ゲスト仮想マシンのネットワーク機能を確認してから、そのゲスト仮想マシンの `vnetX` が指定するブリッジに接続されていることを確認します。

```
# brctl show
bridge name      bridge id                STP enabled    interfaces
virbr0           8000.5254007da9f2        yes
virbr0-nic

vnet0
virbr1           8000.525400682996        yes
virbr1-nic
```

4. 次のコマンドを使って新しいインターフェースのパラメーターでゲスト仮想マシンのネットワークを更新します。

```
# virsh update-device test1 br1.xml
```

```
Device updated successfully
```

5. ゲスト仮想マシン上で **service network restart** を実行します。ゲスト仮想マシンが `virbr1` の新しい IP アドレスを取得します。ゲスト仮想マシンの `virbr0` が新しいブリッジ (`virbr1`) に接続されていることを確認します。

```
# brctl show
bridge name      bridge id                STP enabled    interfaces
virbr0           8000.5254007da9f2        yes            virbr0-nic
virbr1           8000.525400682996        yes            virbr1-nic
vnet0
```

## 18.14. ネットワークのフィルター機能の適用

このセクションでは `libvirt` のネットワークフィルター、フィルターの目的、その概要と XML 形式などについて紹介します。

### 18.14.1. はじめに

ネットワークフィルター機能の目的は、仮想化システムの管理者がネットワークトラフィックのフィルタールールを仮想マシンに設定し、実施することができるようにし、また仮想マシンによる送受信を

許可するネットワークトラフィックのパラメーターを管理できるようにすることです。ネットワークトラフィックのフィルタールールは、仮想マシンの起動時にホスト物理マシンで適用されます。フィルタールールは仮想マシン内から回避することができないため、仮想マシンのユーザーの観点からは強制的なルールとなります。

ゲスト仮想マシンから見ると、ネットワークフィルターのシステムにより、インターフェースごとに各仮想マシンのネットワークトラフィックフィルタールールを個別に設定できます。これらのルールは仮想マシンの起動時にホスト物理マシンで適用され、仮想マシンの実行中に変更することができます。ルールの変更を行う場合は、ネットワークフィルターの XML 記述を編集します。

複数の仮想マシンが同じ汎用ネットワークフィルターを利用できます。このようなフィルターを変更すると、このフィルターを参照している実行中の全仮想マシンのネットワークトラフィックフィルタールールが更新されます。実行中ではないマシンは起動時に更新されます。

前述の通り、ネットワークトラフィックフィルターのルールは、特定タイプのネットワーク設定用に設定された個別のネットワークインターフェースに適用できます。対応しているネットワークのタイプは次の通りです。

- ネットワーク
- イーサネット -- ブリッジモードで使用してください
- bridge

### 例18.1 ネットワークフィルターの例

トップレベルのフィルターの参照にはインターフェース XML が使用されます。次の例では、インターフェースの記述で **clean-traffic** フィルターを参照しています。

```
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e' />
    <filterref filter='clean-traffic' />
  </interface>
</devices>
```

ネットワークフィルターは XML で記述します。他のフィルターへの参照またはトラフィックフィルターのルールのいずれかを含ませるか、またはそれら両方の組み合わせを含ませることもできます。上記の参照フィルター **clean-traffic** は、他のフィルターへの参照のみを含み、実際のフィルタールールは含まれていません。他のフィルターへの参照を使用することができるため、フィルターのツリーを作成することが可能です。**clean-traffic** フィルターは **# virsh nwfilter-dumpxml clean-traffic** コマンドを使用して表示できます。

前述のように、1つのネットワークフィルターを複数の仮想マシンに参照させることができます。一般的にはインターフェースにはトラフィックフィルタールールに関連付けられた個別のパラメーターがあるため、フィルターの XML に記述されているルールは変数を使って一般化することができます。この場合、変数の名前をフィルターの XML で使用し、フィルターが参照される場所にその名前と値を入力します。

### 例18.2 記述の拡張

次の例では、インターフェースの記述にパラメーターの名前「IP」と「ドット表記の IP アドレス」の値を加えて拡張しています。

```
<devices>
```

```

<interface type='bridge'>
  <mac address='00:16:3e:5d:c7:9e' />
  <filterref filter='clean-traffic'>
    <parameter name='IP' value='10.0.0.1' />
  </filterref>
</interface>
</devices>

```

この例では、**clean-traffic** ネットワークトラフィックフィルタは IP アドレスパラメーター **10.0.0.1** で表され、ルールの指示によりこのインターフェースからのトラフィックはすべて必ず **10.0.0.1** をソースの IP アドレスとして使用するようになります。これがこのフィルタの目的です。

### 18.14.2. フィルターチェーン

フィルタールールはフィルターチェーンで編成されます。これらのチェーンは、各チェーン(ブランチ)内に複数のパケットフィルタールールのエントリーを持つツリー構造を形成していると考えられます。

パケットはルートチェーンでフィルター評価を開始し、次に他のチェーンでの評価を継続していき、ルートチェーンに戻ってくるか、または途中のチェーンでドロップされるか、または承認されます。

libvirt のネットワークフィルターシステムでは、ユーザーがトラフィックフィルター機能の有効化を選択した仮想マシンのネットワークインターフェースに個別のルートチェーンを自動生成します。ユーザーは、ルートチェーン内で直接インスタンス化されるフィルタールールを記述したり、プロトコル固有のフィルターチェーンを作成してプロトコル固有のルールを効率的に評価したりすることができます。

次のようなチェーンがあります。

- root
- mac
- stp (スパニングツリープロトコル)
- vlan
- arp と rarp
- ipv4
- ipv6

チェーン名にプロトコル名をプレフィックスとして付けるだけで、**mac**、**stp**、**vlan**、**arp**、**rarp**、**ipv4**、**ipv6** などそれぞれのプロトコルを評価するチェーンを複数作成することができます。

#### 例18.3 ARP トラフィックフィルタ

以下の例では、チェーンに「**arp-xyz**」や「**arp-test**」などのチェーン名を付け、そのチェーン内で ARP プロトコルのパケットが評価されるようにしています。

次のフィルター XML では **arp** チェーン内で ARP トラフィックをフィルタリングする例を示しています。

```
<filter name='no-arp-spoofing' chain='arp' priority='-500'>
  <uuid>f88f1932-debf-4aa1-9fbe-f10d3aa4bc95</uuid>
  <rule action='drop' direction='out' priority='300'>
    <mac match='no' srcmacaddr='$MAC' />
  </rule>
  <rule action='drop' direction='out' priority='350'>
    <arp match='no' arpsrcmacaddr='$MAC' />
  </rule>
  <rule action='drop' direction='out' priority='400'>
    <arp match='no' arpsrcipaddr='$IP' />
  </rule>
  <rule action='drop' direction='in' priority='450'>
    <arp opcode='Reply' />
    <arp match='no' arpdstmacaddr='$MAC' />
  </rule>
  <rule action='drop' direction='in' priority='500'>
    <arp match='no' arpdstipaddr='$IP' />
  </rule>
  <rule action='accept' direction='inout' priority='600'>
    <arp opcode='Request' />
  </rule>
  <rule action='accept' direction='inout' priority='650'>
    <arp opcode='Reply' />
  </rule>
  <rule action='drop' direction='inout' priority='1000' />
</filter>
```

ARP 固有となるルールをルートのチェーンではなく「arp」チェーン内に置くことで、ARP プロトコル以外のパケットを ARP プロトコル固有のルールで評価する必要がなくなります。このためトラフィックフィルタリングの効率性が向上することになります。ただし、プロトコル用のフィルタールールは必ずそのプロトコル用のチェーンに置くよう注意してください。これにより、他のルールが評価されなくなります。たとえば、IPv4 プロトコルのパケットは ARP チェーンを通らないため、ARP チェーン内の IPv4 ルールは評価されません。

### 18.14.3. フィルターチェーンの優先度

前述のように、フィルタールールを作成すると、すべてのチェーンはルートチェーンにつながれます。それらのチェーンがアクセスされる順序はそのチェーンの優先度によって変わります。次の表は、優先度を割り当てることができるチェーンとそのデフォルトの優先度を示しています。

表18.1 フィルターチェーンのデフォルト優先度値

チェーン (プレフィックス)	デフォルトの優先度
stp	-810
mac	-800
vlan	-750
ipv4	-700



チェーン (プレフィックス)	デフォルトの優先度
ipv6	-600
arp	-500
rarp	-400



## 注記

優先度の値が小さいチェーンは、値が大きいチェーンよりも前にアクセスされます。

表18.1「フィルターチェーンのデフォルト優先度値」に一覧表示されているチェーンには、フィルターノード内の優先度 (XML) 属性に [-1000 から 1000] の範囲の値を記述してカスタムの優先度を割り当てることもできます。たとえば、「[フィルターチェーン](#)」のフィルターは **arp** チェーンについて **-500** のデフォルト優先度を示しています。

### 18.14.4. フィルター内での変数の使用

ネットワークトラフィックフィルターのサブシステムで使用するよう **MAC** と **IP** の 2 種類の変数が予約されています。

ネットワークインターフェースの **MAC** アドレスには **MAC** が指定されます。この変数を参照するフィルタールールは自動的にインターフェースの **MAC** アドレスに置換されます。これは、ユーザー側で明示的に **MAC** パラメーターを指定する必要がないために便利です。前述の **IP** パラメーターと同様、**MAC** パラメーターを指定することはできますが、インターフェースが使用する **MAC** アドレスは **libvirt** で認識できるためお勧めしません。

パラメーター **IP** は、仮想マシン内のオペレーティングシステムが特定のインターフェースで使用する **IP** アドレスを表します。パラメーターが明示的な指定ではなく参照する形になっている場合、**libvirt** デーモンがインターフェースで使用されている **IP** アドレス (および **IP** パラメーターの値) を確定するように試行するため、この場合 **IP** パラメーターは特殊となります。現在の **IP** アドレス検出には限界があるため、この機能の使い方および使用した場合に予想される制限については「[制限](#)」をよくお読みください。「[フィルターチェーン](#)」で示した **XML** ファイルには **no-arp-spoofing** のフィルターが含まれています。これは、**MAC** と **IP** 変数を参照する場合にネットワークフィルター **XML** を利用する例です。

参照される変数には常に **\$** 文字のプレフィックスが付きます。変数の値の形式は **XML** で指定されるフィルター属性によって予想されるタイプにする必要があります。上記の例では、**IP** パラメーターに正式な **IP** アドレスを標準形式で持たせる必要があります。不適切な形式を指定すると、フィルターの変数が値に置換されないため、仮想マシンが起動しなくなったり、ホットプラグインを使用している場合はインターフェースが接続されなくなります。**XML** 属性に予想されるタイプのいくつかを [例 18.4「変数タイプの例」](#) に示します。

#### 例18.4 変数タイプの例

変数には複数の要素を含めることができるため (たとえば、変数 **IP** には特定のインターフェースで有効となる複数の **IP** アドレスを含めることができる)、**IP** 変数に複数の要素を指定する場合は以下のように記述します。

```
<devices>
  <interface type='bridge'>
```

```

<mac address='00:16:3e:5d:c7:9e' />
<filterref filter='clean-traffic'>
  <parameter name='IP' value='10.0.0.1' />
  <parameter name='IP' value='10.0.0.2' />
  <parameter name='IP' value='10.0.0.3' />
</filterref>
</interface>
</devices>

```

このXMLファイルは、1つのインターフェースに複数のIPアドレスを有効にするフィルターを作成します。各IPアドレスが別個のフィルタールールになります。したがって、上記のXMLと以下のルールを使用すると、3種類の異なるフィルタールールが作成されます (IPアドレスごと1ルール)。

```

<rule action='accept' direction='in' priority='500'>
  <tcp srpipaddr='$IP' />
</rule>

```

複数の要素を含む変数の各要素にアクセスすることができるため、以下のようなフィルタールールは *DSTPORTS* 変数の 2 番目の要素にアクセスします。

```

<rule action='accept' direction='in' priority='500'>
  <udp dstportstart='$DSTPORTS[1]' />
</rule>

```

### 例18.5 各種変数の使用

**\$VARIABLE[@<iterator id="x">]** の表記を使用すると、異なる一覧からの許容範囲内にあるルールのあらゆる組み合わせを表すことができるフィルタールールの作成が可能です。次のルールは、仮想マシンが *DSTPORTS* で指定されている複数のポート上で *SRCIPADDRESSES* に指定されている複数のソース IP アドレスからのトラフィックを受信できるよう許可します。このルールは、要素へのアクセスに 2 種類の独立した反復子を使って、変数 *SRCIPADDRESSES* と *DSTPORTS* のあらゆる組み合わせを生成します。

```

<rule action='accept' direction='in' priority='500'>
  <ip srcipaddr='$SRCIPADDRESSES[@1]' dstportstart='$DSTPORTS[@2]' />
</rule>

```

以下のように *SRCIPADDRESSES* と *DSTPORTS* に具体的な値を割り当てます。

```

SRCIPADDRESSES = [ 10.0.0.1, 11.1.2.3 ]
DSTPORTS = [ 80, 8080 ]

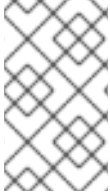
```

**\$SRCIPADDRESSES[@1]** と **\$DSTPORTS[@2]** を使って変数に値を割り当てると、以下のようにアドレスとポートのあらゆる組み合わせが作成されることになります。

- 10.0.0.1, 80
- 10.0.0.1, 8080
- 11.1.2.3, 80
- 11.1.2.3, 8080

**\$SRCIPADDRESSES[@1]** と **\$DSTPORTS[@1]** の表記など、1つの反復子を使って前述の変数にアクセスすると、両方の一覧に並列にアクセスされるため、次のような組み合わせになります。

- 10.0.0.1, 80
- 11.1.2.3, 8080



### 注記

**\$VARIABLE** は **\$VARIABLE[@0]** の簡略形になります。このセクションの冒頭で示したように、先の表記は常に反復子の役割となる **iterator id="0"** が付くと仮定しています。

## 18.14.5. 自動 IP アドレス検出と DHCP スヌーピング

このセクションでは、自動 IP アドレス検出と DHCP スヌーピングについての情報を提供します。

### 18.14.5.1. はじめに

仮想マシンのインターフェースで使用する IP アドレスの検出は、変数の IP が参照されるものの、値が割り当てられていない場合に自動的に作動します。使用する IP アドレスラーニングメソッドを指定する場合に変数 **CTRL\_IP\_LEARNING** を使用します。 *any*、*dhcp*、*none* が有効な値になります。

*any* の値では、*libvirt* に対し、パケットを使って仮想マシンで使用しているアドレスを確定するよう指示します。これは、変数 **CTRL\_IP\_LEARNING** が設定されていない場合のデフォルトになります。このメソッドの場合、検出する IP アドレスはインターフェースごとに1つのみとなります。ゲスト仮想マシンの IP アドレスが検出されると、その IP ネットワークトラフィックがそのアドレスにロックされ、IP アドレスのなりすましなどがそのフィルターの1つで防止されます。この場合、IP アドレスのなりすましと見られるような、仮想マシンのユーザーによるゲスト仮想マシン内からのインターフェースの IP アドレスの変更は行えなくなります。ゲスト仮想マシンを別のホスト物理マシンに移行したり、一時停止状態から再開させた場合、ゲスト仮想マシンが送信する最初のパケットによって特定のインターフェースで利用できる IP アドレスが再度確定されます。

*dhcp* の値では、*libvirt* に DHCP サーバーで割り当てられる有効なリースを持つアドレスしか受け取らないよう指示します。このメソッドは1つのインターフェースでの複数 IP アドレスの検出および使用に対応しています。ゲスト仮想マシンが一時停止状態から再開すると、有効な IP アドレスのリースがそのフィルターに適用されます。その他の場合、ゲスト仮想マシンは新しい IP アドレスを取得するために DHCP を使用します。ゲスト仮想マシンを別の物理的なホスト物理マシンに移行する場合には、ゲスト仮想マシンで DHCP プロトコルを再実行する必要があります。

**CTRL\_IP\_LEARNING** を *none* に設定すると、*libvirt* では IP アドレスラーニングは行われず、明示的な値を割り当てない IP の参照はエラーになります。

### 18.14.5.2. DHCP スヌーピング

**CTRL\_IP\_LEARNING=dhcp** (DHCP スヌーピング) により、とくに IP アドレスの割り当てを信頼できる DHCP サーバーに限るフィルターと併用したい場合に、なりすまし防止に対する安全性が強化されます。これを有効にするには、**DHCPSEVER** 変数を有効な DHCP サーバーの IP アドレスに設定し、この変数を使って着信 DHCP の応答を処理するフィルターを指定します。

DHCP スヌーピングが有効にされており、DHCP リースの有効期限が切れた場合、ゲスト仮想マシンは DHCP サーバーから新しい有効なリースを取得するまで、その IP アドレスを使用できなくなります。ゲスト仮想マシンを移行する場合、IP アドレスを使用するために新しい有効な DHCP リースを取得す

る必要があります (たとえば、仮想マシンをいったんダウンさせてから再起動する必要があります)。



注記

自動 DHCP 検出では、ゲスト仮想マシンがインフラストラクチャーの DHCP サーバーと通信する DHCP トラフィックをリッスンします。libvirt でのサービス拒否攻撃を回避するには、パケット評価の速度制限を行います。つまり、ゲスト仮想マシンがインターフェース上で 1 秒間に大量の DHCP パケットを送信している場合には、そのパケットすべてには評価を行わないようにして、フィルターが適用されないようにします。通常の DHCP クライアントの動作の場合、1 秒に送信する DHCP パケット数は多くないことが想定されます。また、DHCP パケットが送信されないようにするためにインフラストラクチャー内の全ゲスト仮想マシンに適切なフィルターを設定することが重要になります。したがって、ポート 67 からポート 68 へのゲスト仮想マシンによる UDP および TCP トラフィック送信を防止するか、または DHCPSEVER 変数を全ゲスト仮想マシンで使用して DHCP サーバーのメッセージが信頼できる DHCP サーバーからしか送信できないよう制限する必要があります。同時に、なりすまし防止の対策をサブネット内の全ゲスト仮想マシンで有効にしておく必要があります。

例18.6 DHCP スヌーピング用に IP をアクティブ化

以下の XML は、DHCP スヌーピングメソッドを使って IP アドレスラーニングをアクティブにした例を示しています。

```
<interface type='bridge'>
  <source bridge='virbr0' />
  <filterref filter='clean-traffic'>
    <parameter name='CTRL_IP_LEARNING' value='dhcp' />
  </filterref>
</interface>
```

18.14.6. 予約済み変数

表18.2「予約済み変数」は、予約済みとみなされ、libvirt によって使用される変数を示しています。

表18.2 予約済み変数

変数名	定義
MAC	インターフェースの MAC アドレス
IP	インターフェースで使用中の IP アドレス一覧
IPV6	現在のところ実装されていません。インターフェースで使用中の IPV6 アドレスの一覧
DHCPSEVER	信頼できる DHCP サーバーの IP アドレス一覧
DHCPSEVERV6	現在のところ実装されていません。信頼できる DHCP サーバーの IPV6 アドレスの一覧

変数名	定義
CTRL_IP_LEARNING	IP アドレス検出モードの選択

18.14.7. 要素と属性の概要

ネットワークフィルターすべてに必要なルート要素は 2 つの属性を持つ **<filter>** になります。name 属性は特定フィルターの固有名を指定します。chain 属性はオプションですが、基礎となるホスト物理マシンのファイアウォールサブシステムによってより効率的な処理を実現するために特定のフィルターを編成することができます。現在、システムで対応しているチェーンは、root、ipv4、ipv6、arp、および rarp のみです。

18.14.8. 他のフィルターへの参照

いずれのフィルターにも他のフィルターへの参照を持たせることができます。フィルターツリー内の各フィルターは複数回参照できますが、フィルター間の参照がループとならないようにする必要があります。

例18.7 clean traffic フィルターの例

以下は、他のいくつかのフィルターを参照している clean-traffic ネットワークフィルターの XML です。

```
<filter name='clean-traffic'>
  <uuid>6ef53069-ba34-94a0-d33d-17751b9b8cb1</uuid>
  <filterref filter='no-mac-spoofing' />
  <filterref filter='no-ip-spoofing' />
  <filterref filter='allow-incoming-ipv4' />
  <filterref filter='no-arp-spoofing' />
  <filterref filter='no-other-l2-traffic' />
  <filterref filter='qemu-announce-self' />
</filter>
```

別のフィルターを参照させる場合、XML ノードの **<filterref>** をフィルターノード内に指定する必要があります。このノードには参照先のフィルター名を値として持つ属性フィルターを持たせる必要があります。

新しいネットワークフィルターはいつの時点で定義しても構いません。また、libvirt がまだ認識できないネットワークフィルターへの参照を含めることもできます。ただし、仮想マシンの起動後、またはフィルターを参照するネットワークインターフェースのホットプラグ後には、フィルターツリー内の全ネットワークフィルターが利用可能な状態になければなりません。利用できないフィルターがある場合には仮想マシンが起動しなくなるか、またはネットワークインターフェースの接続が不可能になります。

18.14.9. フィルタールール

以下の XML は、発信 IP パケット内の IP アドレス (変数 IP の値から取得される) が期待したアドレスではない場合に、トラフィックをドロップするルールを実施するネットワークトラフィックフィルターの簡単な例を示しています。これにより仮想マシンからの IP アドレスのなりすましを防ぎます。

例18.8 ネットワークトラフィックフィルターの例

```
<filter name='no-ip-spoofing' chain='ipv4'>
  <uuid>fce8ae33-e69e-83bf-262e-30786c1f8072</uuid>
  <rule action='drop' direction='out' priority='500'>
    <ip match='no' srcipaddr='$IP' />
  </rule>
</filter>
```

トラフィックフィルターのルールはルールノードで開始します。このノードには以下の属性を最大 3 つまで含めることができます。

- **action** を **mandatory** にすると、次の値を取ることができます。
  - **drop** (ルールに一致すると、さらに分析することなくパケットを破棄し、メッセージは出力されません)
  - **reject** (ルールに一致すると、さらに分析することなく ICMP 拒否メッセージを生成します)
  - **accept** (ルールに一致すると、さらに分析することなくパケットを受け取ります)
  - **return** (ルールに一致すると、このフィルターを通過しますがさらに分析するため呼び出しフィルターに制御を戻します)
  - **continue** (ルールに一致すると、さらに分析するため次のルールに移動します)
- **direction** を **mandatory** にすると、次の値を取ることができます。
  - **in** - 着信トラフィック
  - **out** - 発信トラフィック
  - **inout** - 着信と発信のトラフィック
- **priority** はオプションです。ルールの優先度は、ルールが他のルールに対して相対的にインスタンス化される順序を制御します。小さい値のルールは大きい値のルールより先にインスタンス化されます。有効な値は **-1000** から **1000** の範囲です。この属性が指定されないと、デフォルトでは優先度 **500** が指定されます。ルートチェーン内のフィルタールールは、その優先度に基づいて、ルートチェーンに接続されるフィルターで分類されます。これにより、フィルターチェーンへのアクセスを持たせながらフィルタールール同士を交互配置することができるようになります。詳細は「[フィルターチェーンの優先度](#)」を参照してください。
- **statematch** はオプションです。「**0**」または「**false**」に設定すると、基礎となる接続状態のマッチングをオフにします。デフォルト設定は「**1**」または「**true**」です。

詳細は、「[高度なフィルター設定について](#)」を参照してください。

前述の [例18.7「clean traffic フィルターの例](#)」では、**type ip** のトラフィックはチェーン **ipv4** に関連付けられ、ルールは **priority=500** になることを示しています。**type ip** のトラフィックがチェーン **ipv4** に関連付けられる別のフィルターが参照される場合は、そのフィルターのルールは先のルールの **priority=500** に基づいて順序付けられます。

ルールにはトラフィックのフィルターを行う単一ルールを含めることができます。上記の例では、タイプ **ip** のトラフィックがフィルターされます。

#### 18.14.10. サポート対象プロトコル

以下のセクションでは、ネットワークフィルターのサブシステムで対応しているプロトコルの詳細について示します。このタイプのトラフィックルールはネスト化されたノードとしてルールノードで指定されます。ルールがフィルターするトラフィックのタイプにより、属性は異なります。上記の例では、単一の **srcipaddr** 属性を示しています。この属性は ip トラフィックフィルターノード内で有効になります。次のセクションでは有効な属性と期待されるデータタイプについて示します。次のようなデータタイプが使用可能です。

- **UINT8**: 8 ビットの整数; 0-255 の範囲
- **UINT16**: 16 ビットの整数; 0-65535 の範囲
- **MAC\_ADDR**: ドット付き 10 進数形式の MAC アドレス (00:11:22:33:44:55 など)
- **MAC\_MASK**: MAC アドレス形式による MAC アドレスマスク (FF:FF:FF:FC:00:00 など)
- **IP\_ADDR**: ドット付き 10 進数形式の IP アドレス (10.1.2.3 など)
- **IP\_MASK**: ドット付き 10 進数形式 (255.255.248.0) または CIDR マスク (0-32) による IP アドレスマスク
- **IPV6\_ADDR**: 数値形式の IPv6 アドレス (FFFF::1)
- **IPV6\_MASK**: 数値形式 (FFFF:FFFF:FC00::) または CIDR マスク (0-128) による IPv6 マスク
- **STRING**: 文字列
- **BOOLEAN**: 'true'、'yes'、'1'、または 'false'、'no'、'0'
- **IPSETFLAGS**: 最大 6 つの 'src' または 'dst' 要素で記述される ipset のソースフラグと宛先フラグで、パケットヘッダーのソース部分または宛先部分いずれかの機能を選択します (src,src,dst など)。ここに入力する 'selectors' の数は参照される ipset のタイプによって異なります。

**IP\_MASK** または **IPV6\_MASK** のタイプを除き、すべての属性は **no** の値の **match** 属性を使って無効にすることができます。無効にした複数の属性を 1 つのまとめることもできます。次の XML の抜粋部分で抽象属性を使った一例を示します。

```
[...]
<rule action='drop' direction='in'>
  <protocol match='no' attribute1='value1' attribute2='value2' />
  <protocol attribute3='value3' />
</rule>
[...]
```

ルールの動作によりそのルールが評価されるとともに、特定のプロトコル属性の境界内でそのルールが論理的に調べられます。単一属性の値がルールで指定された値に一致しない場合、評価のプロセスではそのルール全体が省略されることになります。したがって、上記の例では、プロトコルプロパティの **attribute1** が **value1** に一致せず、プロトコルプロパティの **attribute2** が **value2** に一致せず、かつプロトコルプロパティ **attribute3** が **value3** に一致する場合、着信トラフィックのみがドロップされます。

#### 18.14.10.1. MAC (イーサネット)

プロトコル ID: mac

このタイプのルールはルートチェーンに入ります。

表18.3 MAC プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcmacmask	MAC_MASK	送信側の MAC アドレスに適用されるマスク
dstmacaddr	MAC_ADDR	宛先の MAC アドレス
dstmacmask	MAC_MASK	宛先の MAC アドレスに適用されるマスク
protocolid	UINT16 (0x600-0xffff), STRING	レイヤー 3 プロトコル ID、有効な文字列 [arp, rarp, ipv4, ipv6]
comment	STRING	最長 256 文字のテキスト文字列

フィルターは以下のように記述できます。

```
[...]
<mac match='no' srcmacaddr='$MAC' />
[...]
```

#### 18.14.10.2. VLAN (802.1Q)

プロトコル ID: vlan

このタイプのルールはルートチェーンまたは vlan チェーンのいずれかに入ります。

表18.4 VLAN プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcmacmask	MAC_MASK	送信側の MAC アドレスに適用されるマスク
dstmacaddr	MAC_ADDR	宛先の MAC アドレス
dstmacmask	MAC_MASK	宛先の MAC アドレスに適用されるマスク
vlan-id	UINT16 (0x0-0xffff, 0 - 4095)	VLAN ID



属性名	データタイプ	定義
encap-protocol	UINT16 (0x03c-0xffff), String	カプセル化されたレイヤー 3 プロトコル ID、有効な文字列 arp、ipv4、ipv6
comment	STRING	最長 256 文字のテキスト文字列

### 18.14.10.3. STP (Spanning Tree Protocol)

プロトコル ID: stp

このタイプのルールはルートチェーンまたは stp チェーンのいずれかに入ります。

表18.5 STP プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcmacmask	MAC_MASK	送信側の MAC アドレスに適用されるマスク
type	UINT8	BPDU (Bridge Protocol Data Unit) タイプ
flags	UINT8	BPDU flagdstmacmask
root-priority	UINT16	ルート優先度範囲の始点
root-priority-hi	UINT16 (0x0-0xffff, 0 - 4095)	ルート優先度範囲の終点
root-address	MAC_ADDRESS	ルートの MAC アドレス
root-address-mask	MAC_MASK	ルートの MAC アドレスマスク
roor-cost	UINT32	ルートのパスコスト (範囲の始点)
root-cost-hi	UINT32	ルートのパスコスト (範囲の終点)
sender-priority-hi	UINT16	送信側優先度の範囲の終点
sender-address	MAC_ADDRESS	BPDU 送信側 MAC アドレス
sender-address-mask	MAC_MASK	BPDU 送信側 MAC アドレスマスク
port	UINT16	ポート識別子 (範囲の始点)

属性名	データタイプ	定義
port_hi	UINT16	ポート識別子 (範囲の終点)
msg-age	UINT16	メッセージエイジタイマー (範囲の始点)
msg-age-hi	UINT16	メッセージエイジタイマー (範囲の終点)
max-age-hi	UINT16	最大エイジ時間の範囲の終点
hello-time	UINT16	Hello タイムタイマー (範囲の始点)
hello-time-hi	UINT16	Hello タイムタイマー (範囲の終点)
forward-delay	UINT16	フォワード遅延 (範囲の始点)
forward-delay-hi	UINT16	フォワード遅延 (範囲の終点)
comment	STRING	最長 256 文字のテキスト文字列

#### 18.14.10.4. ARP/RARP

プロトコル ID: arp または rarp

このタイプのルールはルートチェーンまたは arp/rarp チェーンのいずれかに入ります。

表18.6 ARP と RARP のプロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcmacmask	MAC_MASK	送信側の MAC アドレスに適用されるマスク
dstmacaddr	MAC_ADDR	宛先の MAC アドレス
dstmacmask	MAC_MASK	宛先の MAC アドレスに適用されるマスク
hwtype	UINT16	ハードウェアのタイプ
protocoltype	UINT16	プロトコルのタイプ

属性名	データタイプ	定義
opcode	UINT16, STRING	Opcode の有効な文字列: Request、Reply、 Request_Reverse、 Reply_Reverse、 DRARP_Request、 DRARP_Reply、DRARP_Error、 InARP_Request、ARP_NAK
arpsrcmacaddr	MAC_ADDR	ARP/RARP パケット内のソース MAC アドレス
arpdstmacaddr	MAC_ADDR	ARP/RARP パケット内の宛先 MAC アドレス
arpsrcipaddr	IP_ADDR	ARP/RARP パケット内のソース IP アドレス
arpdstipaddr	IP_ADDR	ARP/RARP パケット内の宛先 IP アドレス
gratuitous	BOOLEAN	余計な ARP パケットをチェック するかどうかを指定するブール値
comment	STRING	最長 256 文字のテキスト文字列

#### 18.14.10.5. IPv4

プロトコル ID: ip

このタイプのルールはルートチェーンまたは ipv4 チェーンのいずれかに入ります。

表18.7 IPv4 プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcmacmask	MAC_MASK	送信側の MAC アドレスに適用され るマスク
dstmacaddr	MAC_ADDR	宛先の MAC アドレス
dstmacmask	MAC_MASK	宛先の MAC アドレスに適用され るマスク
srcipaddr	IP_ADDR	ソース IP アドレス

属性名	データタイプ	定義
srcipmask	IP_MASK	ソース IP アドレスに適用されるマスク
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
protocol	UINT8, STRING	レイヤー 4 プロトコルの識別子。 <b>protocol</b> に有効な文字列: tcp、udp、udplite、esp、ah、icmp、igmp、sctp
srcportstart	UINT16	有効なソースポート範囲の開始点。プロトコルが必要。
srcportend	UINT16	有効なソースポート範囲の終了点。プロトコルが必要。
dstportstart	UNIT16	有効な宛先ポート範囲の開始点。プロトコルが必要。
dstportend	UNIT16	有効な宛先ポート範囲の終了点。プロトコルが必要。
comment	STRING	最長 256 文字のテキスト文字列

#### 18.14.10.6. IPv6

プロトコル ID: ipv6

このタイプのルールはルートチェーンまたは ipv6 チェーンのいずれかに入ります。

表18.8 IPv6 プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcmacmask	MAC_MASK	送信側の MAC アドレスに適用されるマスク
dstmacaddr	MAC_ADDR	宛先の MAC アドレス
dstmacmask	MAC_MASK	宛先の MAC アドレスに適用されるマスク

属性名	データタイプ	定義
srcipaddr	IP_ADDR	ソース IP アドレス
srcipmask	IP_MASK	ソース IP アドレスに適用されるマスク
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
protocol	UINT8, STRING	レイヤー 4 プロトコル識別子。 <b>protocol</b> に有効な文字列: <b>tcp</b> 、 <b>udp</b> 、 <b>udplite</b> 、 <b>esp</b> 、 <b>ah</b> 、 <b>icmpv6</b> 、 <b>sctp</b>
srcportstart	UNIT16	有効なソースポート範囲の開始点。プロトコルが必要。
srcportend	UNIT16	有効なソースポート範囲の終了点。プロトコルが必要。
dstportstart	UNIT16	有効な宛先ポート範囲の開始点。プロトコルが必要。
dstportend	UNIT16	有効な宛先ポート範囲の終了点。プロトコルが必要。
comment	STRING	最長 256 文字のテキスト文字列

#### 18.14.10.7. TCP/UDP/SCTP

プロトコル ID: **tcp**、**udp**、**sctp**

このタイプのトラフィックについてはチェーンパラメーターは無視されるため、省略するかまたはルートに設定します。

表18.9 TCP/UDP/SCTP プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcipaddr	IP_ADDR	ソース IP アドレス
srcipmask	IP_MASK	ソース IP アドレスに適用されるマスク

属性名	データタイプ	定義
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
scripto	IP_ADDR	ソース IP アドレス範囲の開始点
srcipfrom	IP_ADDR	ソース IP アドレス範囲の終了点
dstipfrom	IP_ADDR	宛先 IP アドレス範囲の開始点
dstipto	IP_ADDR	宛先 IP アドレス範囲の終了点
srcportstart	UNIT16	有効なソースポート範囲の開始点。プロトコルが必要。
srcportend	UNIT16	有効なソースポート範囲の終了点。プロトコルが必要。
dstportstart	UNIT16	有効な宛先ポート範囲の開始点。プロトコルが必要。
dstportend	UNIT16	有効な宛先ポート範囲の終了点。プロトコルが必要。
comment	STRING	最長 256 文字のテキスト文字列
state	STRING	NEW、ESTABLISHED、RELATED、INVALID または NONE のコンマで区切った一覧
flags	STRING	TCP のみ: マスク/フラグの形式。マスクおよびフラグを SYN、ACK、URG、PSH、FIN、RST または NONE か ALL のコンマで区切った一覧
ipset	STRING	libvirt の外側で管理されている IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要。

#### 18.14.10.8. ICMP

プロトコル ID: icmp

注意: このタイプのトラフィックについてはチェーンパラメーターは無視されるため、省略するかまたはルートに設定します。

表18.10 ICMP プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcmacmask	MAC_MASK	送信側の MAC アドレスに適用されるマスク
dstmacaddr	MAC_ADDR	宛先 MAC アドレス
dstmacmask	MAC_MASK	宛先 MAC アドレスに適用されるマスク
srcipaddr	IP_ADDR	ソース IP アドレス
srcipmask	IP_MASK	ソース IP アドレスに適用されるマスク
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
srcipfrom	IP_ADDR	ソース IP アドレス範囲の開始点
scripto	IP_ADDR	ソース IP アドレス範囲の終了点
dstipfrom	IP_ADDR	宛先 IP アドレス範囲の開始点
dstipto	IP_ADDR	宛先 IP アドレス範囲の終了点
type	UNIT16	ICMP タイプ
code	UNIT16	ICMP コード
comment	STRING	最長 256 文字のテキスト文字列
state	STRING	NEW、ESTABLISHED、RELATED、INVALID または NONE のコンマで区切った一覧
ipset	STRING	libvirt の外側で管理されている IPSet の名前

属性名	データタイプ	定義
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要。

#### 18.14.10.9. IGMP、ESP、AH、UDPLITE、'ALL'

プロトコル ID: igmp、esp、ah、udplite、all

このタイプのトラフィックについてはチェーンパラメーターは無視されるため、省略するかまたはルートに設定します。

表18.11 IGMP、ESP、AH、UDPLITE、'ALL'

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcmacmask	MAC_MASK	送信側の MAC アドレスに適用されるマスク
dstmacaddr	MAC_ADDR	宛先 MAC アドレス
dstmacmask	MAC_MASK	宛先 MAC アドレスに適用されるマスク
srcipaddr	IP_ADDR	ソース IP アドレス
srcipmask	IP_MASK	ソース IP アドレスに適用されるマスク
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
srcipfrom	IP_ADDR	ソース IP アドレス範囲の開始点
scripto	IP_ADDR	ソース IP アドレス範囲の終了点
dstipfrom	IP_ADDR	宛先 IP アドレス範囲の開始点
dstipto	IP_ADDR	宛先 IP アドレス範囲の終了点
comment	STRING	最長 256 文字のテキスト文字列



属性名	データタイプ	定義
state	STRING	NEW、ESTABLISHED、RELATED、INVALID または NONE のコンマで区切った一覧
ipset	STRING	libvirt の外側で管理されている IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要。

#### 18.14.10.10. IPV6 経由の TCP/UDP/SCTP

プロトコル ID: tcp-ipv6、udp-ipv6、sctp-ipv6

このタイプのトラフィックについてはチェーンパラメーターは無視されるため、省略するかまたはルールに設定します。

表18.12 IPv6 経由の TCP、UDP、SCTP プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcipaddr	IP_ADDR	ソース IP アドレス
srcipmask	IP_MASK	ソース IP アドレスに適用されるマスク
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
srcipfrom	IP_ADDR	ソース IP アドレス範囲の開始点
scripto	IP_ADDR	ソース IP アドレス範囲の終了点
dstipfrom	IP_ADDR	宛先 IP アドレス範囲の開始点
dstipto	IP_ADDR	宛先 IP アドレス範囲の終了点
srcportstart	UINT16	有効なソースポート範囲の開始点
srcportend	UINT16	有効なソースポート範囲の終了点
dstportstart	UINT16	有効な宛先ポート範囲の開始点

属性名	データタイプ	定義
dstportend	UINT16	有効な宛先ポート範囲の終了点
comment	STRING	最長 256 文字のテキスト文字列
state	STRING	NEW、ESTABLISHED、RELATED、INVALID または NONE のコンマで区切った一覧
ipset	STRING	libvirt の外側で管理されている IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要。

### 18.14.10.11. ICMPv6

プロトコル ID: icmpv6

このタイプのトラフィックについてはチェーンパラメーターは無視されるため、省略するかまたはルートに設定します。

表18.13 ICMPv6 プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcipaddr	IP_ADDR	ソース IP アドレス
srcipmask	IP_MASK	ソース IP アドレスに適用されるマスク
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
srcipfrom	IP_ADDR	ソース IP アドレス範囲の開始点
scripto	IP_ADDR	ソース IP アドレス範囲の終了点
dstipfrom	IP_ADDR	宛先 IP アドレス範囲の開始点
dstipto	IP_ADDR	宛先 IP アドレス範囲の終了点
type	UINT16	ICMPv6 タイプ

属性名	データタイプ	定義
code	UINT16	ICMPv6 コード
comment	STRING	最長 256 文字のテキスト文字列
state	STRING	NEW、ESTABLISHED、RELATED、INVALID または NONE のコンマで区切った一覧
ipset	STRING	libvirt の外側で管理されている IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要。

#### 18.14.10.12. IPv6 経由の IGMP、ESP、AH、UDPLITE、'ALL'

プロトコル ID: igmp-ipv6、esp-ipv6、ah-ipv6、udplite-ipv6、all-ipv6

このタイプのトラフィックについてはチェーンパラメーターは無視されるため、省略するかまたはルートに設定します。

表18.14 IPv6 経由の IGMP、ESP、AH、UDPLITE、'ALL' プロトコルタイプ

属性名	データタイプ	定義
srcmacaddr	MAC_ADDR	送信側の MAC アドレス
srcipaddr	IP_ADDR	ソース IP アドレス
srcipmask	IP_MASK	ソース IP アドレスに適用されるマスク
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
srcipfrom	IP_ADDR	ソース IP アドレス範囲の開始点
scripto	IP_ADDR	ソース IP アドレス範囲の終了点
dstipfrom	IP_ADDR	宛先 IP アドレス範囲の開始点
dstipto	IP_ADDR	宛先 IP アドレス範囲の終了点
comment	STRING	最長 256 文字のテキスト文字列

属性名	データタイプ	定義
state	STRING	NEW、ESTABLISHED、RELATED、INVALID または NONE のコンマで区切った一覧
ipset	STRING	libvirt の外側で管理されている IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要。

### 18.14.11. 高度なフィルター設定について

次のセクションでは高度なフィルター設定について解説します。

#### 18.14.11.1. 接続の追跡

ネットワークフィルターのサブシステム (Linux 上) では、IP テーブルの接続追跡のサポートを使用します。ネットワークトラフィックの方向性を強制する (状態一致) ほか、ゲスト仮想マシンに対する同時接続数をカウントし、制限するのに役立ちます。たとえば、ゲスト仮想マシン側で TCP ポート 8080 をサーバーとして開くと、クライアントはポート 8080 でゲスト仮想マシンに接続することができます。接続の追跡と方向性の強制により、逆方向となるポート 8080 から (TCP クライアントから) ホスト物理マシンへの接続の開始は妨げられます。さらに重要な点は、追跡を行うことにより、リモート操作によりゲスト仮想マシンへの接続を確立するような攻撃を防ぐことができます。たとえば、ゲスト仮想マシン内のユーザーが攻撃者のサイトでポート 80 への接続を確立した場合でも、攻撃者は逆方向となる TCP ポート 80 からこのゲスト仮想マシンへの接続は開始できません。デフォルトでは、接続の追跡およびトラフィックの方向性の強制を有効にする接続状態一致はオンに設定されます。

#### 例18.9 TCP ポートへの接続をオフにする XML 例

以下の XML の抜粋例では、TCP ポート 12345 への着信接続に対してこの機能をオフにしています。

```
[...]
<rule direction='in' action='accept' statematch='false'>
  <cp dstportstart='12345' />
</rule>
[...]
```

これにより、TCP ポート 12345 への着信トラフィックが許可されますが、仮想マシン内の TCP ポート 12345 (クライアント) からの接続開始も有効になります。これは状況に適する場合もありますが、適さない場合もあります。

#### 18.14.11.2. 接続数の制限

ゲスト仮想マシンで確立できる接続数を制限する場合、特定タイプのトラフィックに対して接続数の制限を設けるルールを指定する必要があります。たとえば、仮想マシンに同時に ping を許可するのは 1 つの IP アドレスのみに制限したり、同時に許可されるアクティブな着信 SSH 接続を 1 つに制限する場合などにこの設定が必要になります。

### 例18.10 接続数に制限を設定した XML サンプル

次の XML の抜粋例を使って接続数を制限することができます。

```
[...]
<rule action='drop' direction='in' priority='400'>
  <tcp connlimit-above='1' />
</rule>
<rule action='accept' direction='in' priority='500'>
  <tcp dstportstart='22' />
</rule>
<rule action='drop' direction='out' priority='400'>
  <icmp connlimit-above='1' />
</rule>
<rule action='accept' direction='out' priority='500'>
  <icmp />
</rule>
<rule action='accept' direction='out' priority='500'>
  <udp dstportstart='53' />
</rule>
<rule action='drop' direction='inout' priority='1000'>
  <all />
</rule>
[...]
```

### 注記

制限ルールは、トラフィックを許可するルールの前に XML に記載する必要があります。例18.10「接続数に制限を設定した XML サンプル」の XML ファイルでは、ポート 22 へ送信される DNS トラフィックがゲスト仮想マシンに到着するのを許可するルールが追加され、SSH デーモンによる DNS ルックアップの失敗に関連する理由で SSH セッションが確立されなくなることを防ぐようにされています。このルールを省くと、SSH クライアントが接続を試行する際に予期しないハングを招く可能性があります。トラフィックの追跡に関連するタイムアウトを扱う場合は十分な注意が必要です。ユーザーがゲスト仮想マシン内で終了させた可能性のある ICMP ping でホスト物理マシンの接続追跡システムに長期のタイムアウトが設定されていると、別の ICMP ping を通過させないことがあります。

最適なソリューションとしては、ホスト物理マシンの `sysfs` 内のタイムアウトをコマンド「`# echo 3 > /proc/sys/net/netfilter/nf_conntrack_icmp_timeout`」で調整する方法です。このコマンドにより、ICMP 接続の追跡タイムアウトが 3 秒に設定されます。1 つの ping が終了すると、別の ping が 3 秒後に開始されます。

何らかの理由でゲスト仮想マシンにより TCP 接続が正しく閉じられなかった場合、とくにホスト物理マシンでの TCP のタイムアウト値が長時間設定されている場合などは、接続が長い期間、開いたままになります。また、アイドル接続により接続追跡システム内でタイムアウトが生じ、パケットが交換されると接続が再度アクティブになる場合があります。

ただし、制限が低すぎると新たに開始された接続がアイドル接続を TCP バックオフに強制することがあります。したがって、接続の制限は高目に設定し、新しい TCP 接続での変動がアイドル接続に関連した異常なトラフィック動作の原因にならないようにします。

### 18.14.11.3. コマンドラインツール

`virsh` はネットワークフィルターのライフサイクルサポートで拡張されています。ネットワークフィルターサブシステムに関連するコマンドはすべて **`nwfilter`** のプレフィックスで開始されます。使用できるコマンドは次の通りです。

- **`nwfilter-list`**: 全ネットワークフィルターの UUID と名前を一覧表示します。
- **`nwfilter-define`**: 新しいネットワークフィルターを定義するか、または既存のフィルターを更新します (フィルター名の入力要)。
- **`nwfilter-undefine`**: 指定したネットワークフィルターを削除します (フィルター名の入力要)。現在使用中のネットワークフィルターは削除しません。
- **`nwfilter-dumpxml`**: 指定したネットワークフィルターを表示します (フィルター名の入力要)。
- **`nwfilter-edit`**: 指定したネットワークフィルターを編集します (フィルター名の入力要)。

### 18.14.11.4. 既存のネットワークフィルター

以下は、`libvirt` で自動的にインストールされるサンプルのネットワークフィルターの一覧です。

表18.15 ICMPv6 プロトコルタイプ

プロトコル名	説明
<b><code>allow-arp</code></b>	ゲスト仮想マシンに対するすべての着信および発信アドレス解決プロトコル (ARP) トラフィックを許可します。
<b><code>no-arp-spoofing</code>, <code>no-arp-mac-spoofing</code>, and <code>no-arp-ip-spoofing</code></b>	これらのフィルターは、ゲスト仮想マシンによる ARP トラフィックのなりすましを防ぎます。さらに、ARP 要求と返信メッセージだけを許可し、パケットに以下の項目が含まれるよう強制します。 <ul style="list-style-type: none"> <li>● <b><code>no-arp-spoofing</code></b> - ゲストの MAC および IP アドレス</li> <li>● <b><code>no-arp-mac-spoofing</code></b> - ゲストの MAC アドレス</li> <li>● <b><code>no-arp-ip-spoofing</code></b> - ゲストの IP アドレス</li> </ul>
<b><code>low-dhcp</code></b>	ゲスト仮想マシンによる DHCP 経由の IP アドレスの要求を許可します (すべての DHCP サーバーから)。
<b><code>low-dhcp-server</code></b>	ゲスト仮想マシンによる指定 DHCP サーバーからの IP アドレスの要求を許可します。DHCP サーバーのドット付き 10 進数 IP アドレスをこのフィルターへの参照内に指定する必要があります。変数名は <b><code>DHCPSEVER</code></b> にしてください。

プロトコル名	説明
<b>low-ipv4</b>	仮想マシンに対するすべての着信および発信 IPv4 トラフィックを許可します。
<b>low-incoming-ipv4</b>	仮想マシンへの着信 IPv4 トラフィックだけを許可します。このフィルターは、 <b>clean-traffic</b> フィルターの一部です。
<b>no-ip-spoofing</b>	ゲスト仮想マシンが、パケット内部のアドレスとは異なるソース IP アドレスを持つ IP パケットを送信することを防ぎます。このフィルターは、 <b>clean-traffic</b> フィルターの一部です。
<b>no-ip-multicast</b>	ゲスト仮想マシンが IP マルチキャストパケットを送信することを防ぎます。
<b>no-mac-broadcast</b>	特定の MAC アドレスへの発信 IPv4 トラフィックを拒否します。このフィルターは、 <b>clean-traffic</b> フィルターの一部です。
<b>no-other-l2-traffic</b>	ネットワークで使用される他のフィルターで指定したトラフィックを除き、すべての第 2 層ネットワークトラフィックを拒否します。このフィルターは、 <b>clean-traffic</b> フィルターの一部です。
<b>no-other-rarp-traffic</b> 、 <b>qemu-announce-self</b> 、 <b>qemu-announce-self-rarp</b>	これらのフィルターは、QEMU の自己アナウンス型逆アドレス解決プロトコル (RARP) パケットは許可しますが、他の RARP トラフィックはすべて拒否します。これらのフィルターも、すべて <b>clean-traffic</b> フィルターに含まれています。
<b>clean-traffic</b>	MAC、IP および ARP のなりすましを防ぎます。このフィルターはビルディングブロックとして他の複数のフィルターを参照します。

これらのフィルターはビルディングブロックに過ぎないため、ネットワークトラフィックフィルターの機能を果たすためには他のフィルターと組み合わせて使用する必要があります。上記の一覧で最もよく使用されるフィルターは **clean-traffic** フィルターです。たとえば、このフィルター自体を **no-ip-multicast** フィルターと組み合わせ、パケットのなりすまし防止に加え、仮想マシンによる IP マルチキャストトラフィックの送信も防ぎます。

#### 18.14.11.5. 独自のフィルターの記述

libvirt で提供されるのは数種類のサンプルネットワークフィルターのみとなるため、独自のフィルターの記述を検討するのもよいでしょう。独自のフィルターを記述する場合、ネットワークフィルターサブシステムについて、またこのシステムが内部でどのように機能するのを知っておく必要があるかもしれません。また、意図するトラフィック以外のトラフィックは通過できず、意図するトラフィックのみが確実に通過できるようフィルターを行うプロトコルについて十分な知識と理解が必要になります。

ネットワークフィルターサブシステムは、現在 Linux のホスト物理マシン上でのみ利用できるため、QEMU および KVM タイプの仮想マシンでのみ機能します。Linux では、これは **ebtables**、**iptables**、

ip6tables のサポートに基づいて構築され、これらの機能が使用されます。「[サポート対象プロトコル](#)」の一覧を参照してください。以下のプロトコルは **ebtables** を使って実装できます。

- mac
- stp (スパニングツリープロトコル)
- vlan (802.1Q)
- arp、rarp
- ipv4
- ipv6

IPv4 経由で実行されるプロトコルはすべて **iptables** を使ってサポートされます。IPv6 経由で実行されるプロトコルは **ip6tables** を使って実装されます。

Linux ホスト物理マシンを使用すると、**libvirt** のネットワークフィルターサブシステムで作成されたトラフィックフィルタールールはすべて **ebtables** で実装したフィルターサポートを最初に通過してから、**iptables** または **ip6tables** フィルターに移行します。フィルターツリーに **mac**、**stp**、**vlan**、**arp**、**rarp**、**ipv4**、**ipv6** などのいずれかのプロトコルのルールがある場合、まず **etables** のルールと一覧表示されている値が自動的に使用されます。

同じプロトコルの複数のチェーンを作成することができます。チェーン名には、以前にエミュレートされたプロトコルのいずれかのプレフィックスを含める必要があります。**ARP** トラフィックの処理用に追加のチェーンを作成するには、**arp-test** などの名前を持つチェーンを指定することができます。

たとえば、**ip** プロトコルフィルターを使用し、許可される **UDP** パケットのポート、ソースと宛先の IP アドレス、プロトコルの属性を指定して、ソースと宛先のポートごとに **UDP** トラフィック上でフィルターをかけることが可能です。これにより、**ebtables** を使った早期の **UDP** トラフィックのフィルターが可能になります。ただし、**UDP** パケットなどの **IP** または **IPv6** パケットが **ebtables** 層を通過した後に、**iptables** または **ip6tables** ルールをインスタンス化する 1 つ以上のルールがフィルターツリーにある場合は、**UDP** パケットを通過させるルールをこれらのフィルター層にも指定する必要があります。これは、適切な **udp** または **udp-ipv6** のトラフィックフィルターノードを含むルールで実行できます。

### 例18.11 カスタムフィルターの作成

以下の要件を満たすフィルターが必要であると仮定します。

- 仮想マシンのインターフェースでの **MAC**、**IP**、**ARP** のなりすましを防ぐ
- 仮想マシンのインターフェースの **TCP** ポート **22** と **80** のみを開く
- 仮想マシンによるインターフェースからの **ping** トラフィック送信を許可する。ただしインターフェース上での仮想マシンに対する **ping** には応答させない
- 仮想マシンによる **DNS** ルックアップを許可する (ポート **53** への **UDP**)

なりすまし防止の要件は既存の **clean-traffic** フィルターで満たすことができるため、カスタムフィルターからこのフィルターを参照させることで防止することができます。

**TCP** ポート **22** と **80** のトラフィックを許可するため、2 種類のルールを追加します。ゲスト仮想マシンによる **ping** トラフィックの送信を許可するために **ICMP** トラフィックにルールを 1 つ追加します。単純にするため、ゲスト仮想マシンからの一般的な **ICMP** トラフィックの開始を許可し、**ICMP echo** の要求および応答メッセージに対する **ICMP** トラフィックの指定を行いません。他のすべてのトラフィックについては、ゲスト仮想マシンからの発信および着信すべてが遮断されます。これを



実行するには、他のすべてのトラフィックをドロップするルールを追加します。ゲスト仮想マシン名は **test**、フィルターを関連付けるインターフェースは **eth0** とし、作成するフィルターには **test-eth0** という名前を付けるとします。

上記をネットワークフィルターの XML に反映させると以下のようになります。

```
<filter name='test-eth0'>
  <!-- - This rule references the clean traffic filter to prevent MAC, IP
  and ARP spoofing. By not providing an IP address parameter, libvirt will
  detect the IP address the guest virtual machine is using. - ->
  <filterref filter='clean-traffic' />

  <!-- - This rule enables TCP ports 22 (ssh) and 80 (http) to be
  reachable - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='22' />
  </rule>

  <rule action='accept' direction='in'>
    <tcp dstportstart='80' />
  </rule>

  <!-- - This rule enables general ICMP traffic to be initiated by the
  guest virtual machine including ping traffic - ->
  <rule action='accept' direction='out'>
    <icmp />
  </rule>>

  <!-- - This rule enables outgoing DNS lookups using UDP - ->
  <rule action='accept' direction='out'>
    <udp dstportstart='53' />
  </rule>

  <!-- - This rule drops all other traffic - ->
  <rule action='drop' direction='inout'>
    <all />
  </rule>

</filter>
```

#### 18.14.11.6. カスタムフィルターのサンプル

上記の XML 内のルールの1つにはゲスト仮想マシンのソースまたは宛先アドレスとなるいずれかの IP アドレスが含まれていますが、トラフィックのフィルターは正しく動作します。これは、ルールの評価はインターフェースごとに内部的に発生しますが、ルールがソースや宛先の IP アドレスではなく、パケットを送信したか、または受信するインターフェース (**tap**) に基づいて追加で評価されるためです。

#### 例18.12 ネットワークインターフェースの詳細部分のサンプル XML

以下の XML の抜粋は、**test** ゲスト仮想マシンのドメイン XML 内にあるネットワークインターフェースの記述例を示しています。

```
[...]
<interface type='bridge'>
```

```

        <source bridge='mybridge' />
        <filterref filter='test-eth0' />
    </interface>
    [...]

```

ICMP トラフィックをさらに厳密に制御し、ICMP echo の要求はゲスト仮想マシンからしか送信できないようにすると共に、ICMP echo の応答もゲスト仮想マシンでしか受信できないようにするには、上記の ICMP ルールを次の 2 つのルールに置き換えます。

```

<!-- - enable outgoing ICMP echo requests- ->
<rule action='accept' direction='out'>
    <icmp type='8' />
</rule>

```

```

<!-- - enable incoming ICMP echo replies- ->
<rule action='accept' direction='in'>
    <icmp type='0' />
</rule>

```

### 例18.13 カスタムフィルターのサンプル 2

この例では、前述の例と同じようなフィルターを構築していますが、ゲスト仮想マシン内にある ftp サーバーに関連して要件の一覧を拡張しています。このフィルターの要件は次の通りです。

- ゲスト仮想マシンのインターフェースでの MAC、IP、ARP のなりすましを防ぐ
- ゲスト仮想マシンのインターフェースの TCP ポート 22 と 80 のみを開く
- ゲスト仮想マシンによるインターフェースからの ping トラフィック送信を許可するが、インターフェース上でのゲスト仮想マシンに対する ping には応答させない
- ゲスト仮想マシンによる DNS ルックアップを許可する (ポート 53 への UDP)
- ftp サーバーを有効にして (アクティブモード)、ゲスト仮想マシンの内側で実行できるようにする

ゲスト仮想マシン内で FTP サーバーを実行できるようにする追加要件は、FTP 制御トラフィックに対してポート 21 をアクセス可能にし、ゲスト仮想マシンによるゲスト仮想マシンの TCP ポート 20 から FTP クライアント (FTP アクティブモード) へ向けて発信する TCP 接続の確立を許可することになります。このフィルターは複数の方法で記述することができますが、以下の例では 2 種類のソリューションを示します。

1 つ目のソリューションでは、Linux ホスト物理マシンの接続追跡フレームワークとのつながりを提供する TCP プロトコルの状態属性を利用します。ゲスト仮想マシンが開始した FTP データ接続の場合 (FTP アクティブモード)、RELATED 状態を使ってゲスト仮想マシンによって開始された FTP データ接続が既存の FTP 制御接続の結果である (または関連性がある) ことを検出することができます。これにより、パケットがファイアウォールを通過できるようになります。ただし、RELATED 状態は、FTP データパスの TCP 発信接続の 1 番目のパケットにのみ有効になります。以後、状態は ESTABLISHED になり、この状態が着信と発信の両方向に一律に適用されます。これはすべてゲスト仮想マシンの TCP ポート 20 から発信された FTP データトラフィックに関連します。これが次のソリューションにつながります。

```

<filter name='test-eth0'>
    <!-- - This filter (eth0) references the clean traffic filter to

```

```

prevent MAC, IP, and ARP spoofing. By not providing an IP address
parameter, libvirt will detect the IP address the guest virtual machine
is using. - ->
<filterref filter='clean-traffic' />

<!-- - This rule enables TCP port 21 (FTP-control) to be reachable - ->
<rule action='accept' direction='in'>
  <tcp dstportstart='21' />
</rule>

<!-- - This rule enables TCP port 20 for guest virtual machine-
initiated FTP data connection related to an existing FTP control
connection - ->
<rule action='accept' direction='out'>
  <tcp srcportstart='20' state='RELATED,ESTABLISHED' />
</rule>

<!-- - This rule accepts all packets from a client on the FTP data
connection - ->
<rule action='accept' direction='in'>
  <tcp dstportstart='20' state='ESTABLISHED' />
</rule>

<!-- - This rule enables TCP port 22 (SSH) to be reachable - ->
<rule action='accept' direction='in'>
  <tcp dstportstart='22' />
</rule>

<!-- -This rule enables TCP port 80 (HTTP) to be reachable - ->
<rule action='accept' direction='in'>
  <tcp dstportstart='80' />
</rule>

<!-- - This rule enables general ICMP traffic to be initiated by the
guest virtual machine, including ping traffic - ->
<rule action='accept' direction='out'>
  <icmp />
</rule>

<!-- - This rule enables outgoing DNS lookups using UDP - ->
<rule action='accept' direction='out'>
  <udp dstportstart='53' />
</rule>

<!-- - This rule drops all other traffic - ->
<rule action='drop' direction='inout'>
  <all />
</rule>

</filter>

```

RELATED 状態を使ってフィルターを試す前に、適切な接続追跡モジュールがホスト物理マシンのカーネルにロードされていることを確認する必要があります。カーネルのバージョンによっては、ゲスト仮想マシンで FTP 接続が確立される前に次の 2 つのコマンドのいずれかを実行する必要があります。

- `# modprobe nf_conntrack_ftp` - 利用可能な場合、あるいは
- `# modprobe ip_conntrack_ftp` - 上記のコマンドが利用できない場合

FTP 以外のプロトコルを `RELATED` 状態と併用する場合は、該当するモジュールをロードする必要があります。各プロトコルに使用できるモジュールは、`ftp`、`tftp`、`irc`、`sip`、`sctp` および `amanda` です。

2 つ目のソリューションでは、前述のソリューションより多くの接続状態フラグを利用します。このソリューションでは、トラフィックの一番最初のパケットが検出されると接続の `NEW` 状態が有効になる点を利用しています。続いて、最初のパケットのフローが許可されると、そのフローは接続とみなされ、`ESTABLISHED` 状態に移行します。したがって、`ESTABLISHED` 接続のパケットのゲスト仮想マシンへの着信、またゲスト仮想マシンからの発信を許可する汎用ルールを記述することができます。`NEW` 状態で識別できる一番最初のパケットについての特定ルールを記述し、データを受け取ることができるポートを指定して実行できます。明示的に許可されていないポートに向けたパケットはすべてドロップされるため、`ESTABLISHED` 状態にはなりません。そのポートから送信される後続パケットもすべてドロップされます。

```
<filter name='test-eth0'>
  <!-- - This filter references the clean traffic filter to prevent MAC,
  IP and ARP spoofing. By not providing an IP address parameter, libvirt
  will detect the IP address the VM is using. - -->
  <filterref filter='clean-traffic' />

  <!-- - This rule allows the packets of all previously accepted
  connections to reach the guest virtual machine - -->
  <rule action='accept' direction='in'>
    <all state='ESTABLISHED' />
  </rule>

  <!-- - This rule allows the packets of all previously accepted and
  related connections be sent from the guest virtual machine - -->
  <rule action='accept' direction='out'>
    <all state='ESTABLISHED,RELATED' />
  </rule>

  <!-- - This rule enables traffic towards port 21 (FTP) and port 22
  (SSH) - -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='21' dstportend='22' state='NEW' />
  </rule>

  <!-- - This rule enables traffic towards port 80 (HTTP) - -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='80' state='NEW' />
  </rule>

  <!-- - This rule enables general ICMP traffic to be initiated by the
  guest virtual machine, including ping traffic - -->
  <rule action='accept' direction='out'>
    <icmp state='NEW' />
  </rule>

  <!-- - This rule enables outgoing DNS lookups using UDP - -->
  <rule action='accept' direction='out'>
    <udp dstportstart='53' state='NEW' />
  </rule>
</filter>
```

```

</rule>

<!-- - This rule drops all other traffic - -->
<rule action='drop' direction='inout'>
  <all/>
</rule>

</filter>

```

### 18.14.12. 制限

以下は、ネットワークフィルターサブシステムについての既知の制限の一覧です。

- 仮想マシンの移行サポートは、ゲスト仮想マシンのトップレベルのフィルターで参照されるフィルターツリー全体がターゲットのホスト物理マシンでも使用できる場合に限られます。たとえば、ネットワークフィルター **clean-traffic** はすべての **libvirt** インストールで使用できる状態でなければなりません。これにより、このフィルターを参照するゲスト仮想マシンの移行が可能になります。バージョンの互換性が問題とならないよう、定期的にパッケージの更新を行い、常に最新の **libvirt** バージョンを使用するようにしてください。
- インターフェースに関連しているネットワークトラフィックフィルターを失わないよう、移行はバージョン **0.8.1** またはそれ以降の **libvirt** インストール間で行うようにしてください。
- **VLAN (802.1Q)** パケットがゲスト仮想マシンによって送信された場合、プロトコル ID が **arp**、**rarp**、**ipv4**、**ipv6** のルールではフィルターを行うことができません。このようなパケットをフィルターできるのは、プロトコル ID が **MAC** か **VLAN** の場合のみです。したがって、**clean-traffic** フィルターの例である [例18.1「ネットワークフィルターの例」](#) は期待通りには機能しません。

## 18.15. トンネルの作成

このセクションでは、複数の異なるトンネルを使用したシナリオを実施する方法について説明します。

### 18.15.1. マルチキャストトンネルの作成

マルチキャストグループは、仮想ネットワークを表すためにセットアップされます。ネットワークデバイスが同じマルチキャストグループにあるゲスト仮想マシンは、異なるホスト物理マシン間であっても相互に通信することができます。このモードは、権限のないユーザーも使用できます。デフォルトの **DNS** または **DHCP** サポートはなく、発信ネットワークアクセス也没有ありません。発信ネットワークアクセスを提供するには、ゲスト仮想マシンの1つに、適切なルートを提供する、最初の4つのネットワークタイプのいずれかに接続されている2つ目の **NIC** がなければなりません。マルチキャストプロトコルは、ゲスト仮想マシンのユーザーモードと互換性があります。指定するソースアドレスは、マルチキャストアドレスブロックに使用されるアドレスから取られる必要があります。

マルチキャストトンネルを作成するには、以下の **XML** 詳細を **<devices>** 要素に組み込みます。

```

...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
      <source address='230.0.0.1' port='5558' />
    </interface>
  </devices>
...

```

図18.28 マルチキャストトンネルのドメイン XML サンプル

### 18.15.2. TCP トンネルの作成

TCP クライアント/サーバーアーキテクチャーは仮想ネットワークを提供します。この設定では、他のすべてのゲスト仮想マシンがクライアントとして設定される中、1つのゲスト仮想マシンがネットワークのサーバーエンドを提供します。すべてのネットワークトラフィックは、ゲスト仮想マシンサーバーを経由してゲスト仮想マシンのクライアント間で経路指定されます。このモードは、権限のないユーザーも利用できます。このモードは、デフォルトの DNS または DHCP サポートを提供せず、発信ネットワークアクセスも提供しないことに注意してください。発信ネットワークアクセスを提供するには、ゲスト仮想マシンの1つに、適切なルートを提供する最初の4つのネットワークタイプのいずれかに接続されている2つ目の NIC がなければなりません。

TCP トンネルを作成するには、以下の XML 詳細を **<devices>** 要素に組み込みます。

```

...
<devices>
  <interface type='server'>
    <mac address='52:54:00:22:c9:42'>
      <source address='192.168.0.1' port='5558' />
    </interface>
    ...
    <interface type='client'>
      <mac address='52:54:00:8b:c9:51'>
        <source address='192.168.0.1' port='5558' />
      </interface>
    </devices>
...

```

図18.29 TCP トンネルのドメイン XML サンプル

## 18.16. VLAN タグの設定

*virtual local area network* (vLAN) タグは、**virsh net-edit** コマンドを使用して追加されます。このタグは、SR-IOV デバイスの PCI デバイスの割り当てで使用することもできます。詳細は、「[SR-IOV デバイスの場合の PCI 割り当ての設定](#)」を参照してください。

```

<network>
  <name>ovs-net</name>
  <forward mode='bridge' />
  <bridge name='ovsbr0' />
  <virtualport type='openvswitch'>
    <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f' />
  </virtualport>
  <vlan trunk='yes'>
    <tag id='42' nativeMode='untagged' />
    <tag id='47' />
  </vlan>
  <portgroup name='dontpanic'>
    <vlan>
      <tag id='42' />
    </vlan>
  </portgroup>
</network>

```

図18.30 vSetting VLAN タグ (対応するネットワークタイプの場合のみ)

ネットワークタイプがゲストに透過的な vlan タグに対応している場合 (のみ)、オプションの **<vlan>** 要素は、このネットワークを使用するすべてのゲストのトラフィックに適用する1つ以上の vlan タグを指定することができます。(openvswitch および type='hostdev' SR-IOV ネットワークはゲストトラフィックの透過的な VLAN タグに対応しません。標準的な Linux ブリッジや libvirt の独自の仮想ネットワークなど、これ以外のものもこのタグに対応しません。802.1Qbh (vn-link) および 802.1Qbg (VEPA) スイッチは、ゲストトラフィックを特定の vlan にタグ付けするための独自の方法 (libvirt 外) を提供します。tag 属性は、使用する vlan タグを指定します。ネットワークに複数の **<vlan>** 要素が定義されている場合、ユーザーはすべての指定タグを使用して VLAN トランッキングを実行することを必要していると想定されます。単一タグの VLAN トランッキングが必要な場合、オプション属性の trunk='yes' を VLAN 要素に追加できます。

openvswitch を使用するネットワーク設定の場合、「native-tagged」(ネイティブ-タグ付け) および「native-untagged」(ネイティブ-タグ解除) の vlan モードを設定することができます。これは、**<tag>** 要素でオプションの nativeMode 属性を使用します。nativeMode は「tagged」または「untagged」に設定することができます。この要素の id 属性は、ネイティブの vlan を設定します。

**<vlan>** 要素は、ドメインの **<interface>** 要素に直接指定されるのと同様に、**<portgroup>** 要素にも指定することができます。vlan タグが複数の場所に指定される場合、**<interface>** の設定が優先され、次に interface config で選択される **<portgroup>** の設定が適用されます。**<network>** の **<vlan>** は、**<portgroup>** または **<interface>** にいずれの値も指定されない場合にのみ選択されます。

## 18.17. QoS の仮想ネットワークへの適用

Quality of Service (QoS) は、ネットワーク上のすべてのユーザーにとっての最適な体験を保証するリソースコントロールシステムを指し、遅延、ジッター (ゆらぎ)、またはパケットの喪失を防ぎます。QoS はアプリケーション固有、またはユーザー/グループ固有の設定にすることができます。詳細は、「[Quality of service \(QoS\)](#)」を参照してください。

## 第19章 ゲストのリモート管理

このセクションでは、ゲストをリモートで管理する方法について説明しています。

### 19.1. トランスポートモード

リモート管理用に、**libvirt** では次のようなトランスポートモードに対応しています。

#### Transport Layer Security (TLS)

Transport Layer Security TLS 1.0 (SSL 3.1) で認証され、暗号化される TCP/IP ソケットは、通常パブリックポート番号でリスンします。これを使用するには、クライアントとサーバーの証明書を生成する必要があります。標準のポートは 16514 です。詳細は、「[TLS と SSL 経由のリモート管理](#)」を参照してください。

#### SSH

Secure Shell protocol (SSH) 接続上のトランスポートです。**libvirt** デーモン (**libvirtd**) がリモートマシン上で実行されている必要があります。ポート 22 が SSH アクセス用に開いていなければなりません。いずれかの SSH キー管理 (**ssh-agent** ユーティリティなど) を使用しないとパスワードの入力が求められます。詳細については、「[SSH によるリモート管理](#)」を参照してください。

#### UNIX ソケット

UNIX ドメインソケットはローカルマシン上でのみアクセス可能です。ソケットは暗号化されておらず、認証のために SELinux または UNIX 権限を使用します。標準のソケット名は `/var/run/libvirt/libvirt-sock` と `/var/run/libvirt/libvirt-sock-ro` (読み取り専用の接続) です。

#### ext

**ext** パラメーターは、**libvirt** の範囲外の手段によりリモートマシンに接続できる外部プログラムに使用されます。このパラメーターはサポートされていません。

#### TCP

暗号化されていない TCP/IP ソケットです。実稼働での使用には推奨されません。通常は無効になっていますが、管理者はテストを行う場合や信頼できるネットワークでこれを有効にできます。デフォルトのポートは 16509 です。

デフォルトのトランスポートモードは、他に指定がない場合は TLS です。

#### リモート URI

URI (Uniform Resource Identifier) は、リモートホストに接続するために **virsh** と **libvirt** によって使用されます。また URI は **virsh** コマンドの **--connect** パラメーターと一緒に使用すると、リモートホストで単一コマンドや移行を実行することができます。リモート URI は一般的なローカル URI を取り、ホスト名またはトランスポート名、またはそれら両方を追加して形成されます。特殊なケースとして、「リモート」の URI スキームを使用すると、リモート **libvirtd** サーバーは最適なハイパーバイザードライバを探索するように指示されます。これはローカル接続用に NULL URI を渡すのと同様です。

**libvirt** URI は一般的な形式を取ります (角括弧 [] 内の中身はオプションの関数を表します)。

```
driver[+transport]://[username@][hostname][:port]/path[?extraparameters]
```

ハイパーバイザー (ドライバ) が **QEMU** の場合、パスは必須になることに注意してください。

以下は有効なリモート URI のいくつかの例です。



- `qemu://hostname/`

外部の場所を対象とする場合は、トランスポートメソッドまたはホスト名を指定する必要があります。詳細は、[libvirt アップストリームドキュメント](#)を参照してください。

### リモート管理の例

- **host2** という名前のリモート KVM ホストに接続します。SSH トランスポートを使用し、SSH ユーザー名は **virtuser** です。それぞれの `connect` コマンドは `connect [URI] [--readonly]` です。`virsh connect` コマンドの詳細は、「[virsh Connect を使用したハイパーバイザーへの接続](#)」を参照してください。

```
qemu+ssh://virtuser@host2/
```

- TLS を使用して、**host2** という名前のホスト上のリモート KVM ハイパーバイザーに接続します。

```
qemu://host2/
```

### テスト用サンプル

- ローカルの KVM ハイパーバイザーに非標準の UNIX ソケットで接続します。この例では、UNIX ソケットへの完全パスが明示的に指定されています。

```
qemu+unix:///system?socket=/opt/libvirt/run/libvirt/libvirt-sock
```

- 暗号化していない TCP/IP 接続で libvirt デーモンに接続します。IP アドレスが **10.1.1.10** でポートが **5000** のサーバーへの接続です。この例ではデフォルト設定の **test** ドライバーが使用されています。

```
test+tcp://10.1.1.10:5000/default
```

### 追加の URI パラメーター

追加パラメーターをリモート URI に追加することができます。以下の表に認識されているパラメーターを示します。これ以外のパラメーターはすべて無視されます。パラメーターの値は URI エスケープされる必要があることに注意してください（つまり、疑問符 (?) をパラメーターの前に付けると、特殊文字が URI 形式に変換されます）。

表19.1 追加の URI パラメーター

名前	トランスポートモード	説明	使用法の例
----	------------	----	-------

名前	トランスポートモード	説明	使用法の例
name	すべてのモード	name はリモート <b>virConnectOpen</b> 関数に渡されます。通常 name はリモート URI から <b>transport</b> 、 <b>hostname</b> 、 <b>port number</b> 、 <b>username</b> 、および追加パラメータを取り除いたものになります。ただし、一部の複雑なケースでは、name を明示的に指定するのが適切な場合があります。	name=qemu:///system
command	ssh と ext	外部コマンドです。外部トランスポートの場合に必須になります。ssh の場合、デフォルトは ssh です。command の PATH が検索されます。	command=/opt/openssh/bin/ssh
socket	unix と ssh	UNIX ドメインソケットへのパスで、デフォルトを上書きします。ssh トランスポートの場合、これがリモート <b>netcat</b> コマンドに渡されます ( <b>netcat</b> を参照)。	socket=/opt/libvirt/run/libvirt/libvirt-sock
no_verify	tls	ゼロ以外の値が設定されていると、クライアント側のサーバー証明書チェックが無効になります。サーバー側のクライアント証明書チェックまたは IP アドレスチェックを無効にする場合は、 <b>libvirtd</b> の設定を変更する必要があります。	no_verify=1
no_tty	ssh	ゼロ以外の値に設定すると、リモートマシンに自動的にログインできない場合に <b>SSH</b> がパスワードの入力を求めないようにします。ターミナルにアクセスできない場合にこれを使用します。	no_tty=1

## 19.2. SSH によるリモート管理

**ssh** パッケージは、リモートの仮想化サーバーに安全に管理機能を送信できる暗号化されたネットワークプロトコルを提供します。ここで説明される方法では、**SSH** 接続を介して安全にトンネル化した **libvirt** 管理用接続を使ってリモートのマシン群を管理します。認証はすべてローカルの **SSH** エージェントで収集したパスフレーズまたはパスワード、および **SSH** パブリックキーの暗号を使って行われます。さらに、各ゲストの **VNC** コンソールも **SSH** 経由でトンネル化されます。

**SSH** を使って仮想マシンをリモートで管理する場合は、以下の点に注意してください。

- 仮想マシンの管理を行う場合、リモートのマシンには **root** でログインしてアクセスする必要があります。
- 初期接続のセットアップには時間がかかる場合があります。
- すべてのホストまたはゲスト上でユーザーのキーを無効にする場合の標準的な方法や普通の方法というものはありません。
- リモートマシンの台数が増えると、**SSH** のスケーラビリティは低下します。



### 注記

**Red Hat Virtualization** を利用すると多数の仮想マシン群のリモート管理が可能になります。詳細は、[Red Hat Virtualization のドキュメント](#) を参照してください。

**SSH** アクセスには以下のパッケージが必要になります。

- **openssh**
- **openssh-askpass**
- **openssh-clients**
- **openssh-server**

**virt-manager** の **SSH** アクセスを設定する - パスワードなしの場合とパスワードを必要とする場合

次の手順では、**SSH** キーのセットアップを行っていないゼロの状態から開始することを想定しています。**SSH** キーのセットアップや他のシステムへのキーのコピーがすでに完了している場合は、この手順は省略して構いません。



### 重要

**SSH** キーはユーザー固有となるため所有者以外は使用できません。キーの所有者はそのキーを生成したユーザーになります。異なるユーザー間でのキーの共有はできません。

リモートホストへの接続を行う場合、そのキーを所有しているユーザーが **virt-manager** を実行しなければなりません。つまり、リモートのシステムが **root** 以外のユーザーによって管理されている場合、**virt-manager** は特権のないモードで実行されなければなりません。リモートのシステムがローカルの **root** ユーザーによって管理されている場合は、**root** ユーザーは **SSH** キーを作成し、所有する必要があります。

ローカルホストは、特権を持たないユーザーが **virt-manager** を使って管理することはできません。

### 1. オプション: ユーザーの切り替え

必要に応じてユーザーの切り替えを行います。ここでは、他のホストおよびローカルホストをリモートで管理するためにローカルの **root** ユーザーを使用します。

```
$ su -
```

### 2. SSH キーペアの生成

**virt-manager** を使用するマシン上でパブリックキーを生成します。ここではキーの格納先にデフォルトの **~/.ssh/** ディレクトリーを使用します。

```
# ssh-keygen -t rsa
```

### 3. キーをリモートのホスト群にコピー

パスワードがないリモートログインまたはパスフレーズによるリモートログインを行うには、管理対象のシステムに SSH キーを配信しておく必要があります。**ssh-copy-id** コマンドを使って、指定されたシステムアドレス (この例では **root@host2.example.com**) の **root** ユーザーにキーをコピーします。

```
# ssh-copy-id -i ~/.ssh/id_rsa.pub root@host2.example.com  
root@host2.example.com's password:
```

この後に、マシンにログインしてから **.ssh/authorized\_keys** ファイルを確認して予期しないキーが追加されていないことを確認します。

```
ssh root@host2.example.com
```

必要に応じて、他のシステムにも同じ手順を繰り返します。

### 4. オプション: パスフレーズを **ssh-agent** に追加

必要に応じて、SSH キーのパスフレーズを **ssh-agent** に追加します。ローカルホストで次のコマンドを使い、パスフレーズを追加し (ある場合)、パスワード入力をしないログインを有効にします。

```
# ssh-add ~/.ssh/id_rsa
```

このコマンドは、**ssh-agent** が実行されていない場合は実行できません。エラーや競合を避けるため、SSH パラメーターが正しく設定されていることを確認してください。詳細は、『[Red Hat Enterprise Linux 7 システム管理者のガイド](#)』を参照してください。

## libvirt デーモン (libvirtd)

**libvirt** デーモンは仮想マシンの管理用インターフェースを提供します。**libvirtd** デーモンは、管理する必要のあるすべてのリモートホストにインストールし、実行しておく必要があります。

```
$ ssh root@somehost  
# systemctl enable libvirtd.service  
# systemctl start libvirtd.service
```

**libvirtd** と **SSH** の設定が完了したら、仮想マシンへのリモートアクセスおよびリモート管理が可能になるはずです。また、この時点で **VNC** を使ったゲストへのアクセスも可能になるはずです。

## virt-manager でリモートホスト群にアクセスする

リモートホスト群は **virt-manager** GUI ツールで管理することができます。パスワード入力を行わないログインを行うには、**virt-manager** を実行するユーザーが SSH キーを所有していなければなりません。

1. **virt-manager** を開始します。
2. ファイル ⇒ 接続を追加 の順に開きます。

図19.1 接続を追加のメニュー

3. ドロップダウンメニューを使ってハイパーバイザーのタイプを選択し、リモートホストに接続のチェックボックスをクリックして接続のメソッド (この例では SSH 経由のリモートトンネル) を開き、ユーザー名と ホスト名 を入力します。次に **接続** をクリックします。

### 19.3. TLS と SSL 経由のリモート管理

TLS と SSL プロトコルを使って仮想マシンを管理することができます。TLS と SSL によりスケーラビリティが向上しますが、SSH を使用する場合より複雑になります (「[SSH によるリモート管理](#)」を参照)。TLS と SSL は安全な接続を確保するために Web ブラウザーで使われる同一の技術です。**libvirt** 管理接続は、着信接続用の TCP ポートを開きます。この接続には安全な暗号化が行われ、x509 証明書に基づいて認証されます。TLS と SSL での管理に必要な認証用証明書を作成し、実装する方法を以下に示します。

#### 手順19.1 TLS 管理の認証局 (CA) キーを作成する

1. まず、**gnutls-utils** がインストールされていることを確認します。インストールされていない場合は、そのインストールを行います。

```
# yum install gnutls-utils
```

2. 次のコマンドを使ってプライベートキーを生成します。

```
# certtool --generate-privkey > cakey.pem
```

3. キーを生成したら、次にキーに自己署名できるよう署名ファイルを作成します。署名の詳細を含むファイルを作成して、**ca.info** という名前を付けます。このファイルには次の行を含めてください。

```
cn = Name of your organization
ca
cert_signing_key
```

4. 自己署名キーを次のコマンドで生成します。

```
# certtool --generate-self-signed --load-privkey cakey.pem --
template ca.info --outfile cacert.pem
```

ファイルを生成し終わったら、**rm** コマンドで **ca.info** ファイルを削除できます。生成プロセスで作成されたファイルには **cacert.pem** という名前が付けられます。このファイルがパブリックキー (証明書) になります。ロードしたファイル **cakey.pem** がプライベートキーです。セキュリティ保護のため、このファイルは共有スペースには保管しないようにし、このキーは機密扱いにしてください。

5. **cacert.pem** CA 証明書ファイルをすべてのクライアントおよびサーバーの **/etc/pki/CA/cacert.pem** ディレクトリーにインストールし、この **CA** で発行した証明書は信頼できる証明書であることを通知します。このファイルの内容を表示するには、次のコマンドを実行します。

```
# certtool -i --infile cacert.pem
```

認証局の設定は以上です。認証局のプライベートキーは安全な場所に保管してください。クライアントやサーバーの証明書を発行する際に必要となります。

## 手順19.2 サーバー証明書の発行

以下の手順は、**X.509 CommonName (CN)** フィールドをサーバーのホスト名に設定して証明書を発行する方法を示しています。**CN** は、クライアントがサーバーに接続する際に使用するホスト名と一致しなければなりません。この例では、クライアントは **qemu://mycommonname/system** という **URI** を使用してサーバーに接続するので、**CN** フィールドも同様に「**mycommonname**」にする必要があります。

1. サーバーのプライベートキーを作成します。

```
# certtool --generate-privkey > serverkey.pem
```

2. まず **server.info** という名前のテンプレートファイルを作成して認証局のプライベートキー用の署名を生成します。**CN** にはサーバーのホスト名と同じ名前を必ず設定してください。

```
organization = Name of your organization
cn = mycommonname
tls_www_server
encryption_key
signing_key
```

3. 証明書を作成します。

```
# certtool --generate-certificate --load-privkey serverkey.pem --
load-ca-certificate cacert.pem --load-ca-privkey cakey.pem \ --
template server.info --outfile servercert.pem
```

次の2種類のファイルが生成されます。

- **serverkey.pem** - サーバーのプライベートキー
  - **servercert.pem** - サーバーのパブリックキー
4. プライベートキーを保存する場所は機密扱いにしてください。ファイルの内容を表示するには、次のコマンドを使用します。

```
# certtool -i --infile servercert.pem
```

このファイルを開いた場合、**CN=** パラメーターが前の手順で設定した **CN** と同じであることを確認してください。この例の場合は **mycommonname** になります。

5. この2つのファイルを次の場所にインストールします。
- **serverkey.pem** - サーバーのプライベートキーです。このファイルは「**/etc/pki/libvirt/private/serverkey.pem**」に配置します。
  - **servercert.pem** - サーバーの証明書です。このファイルはサーバーの「**/etc/pki/libvirt/servercert.pem**」に配置します。

### 手順19.3 クライアント証明書の発行

1. すべてのクライアント (**virt-manager** など **libvirt** でリンクしているすべてのプログラム) について、適切な名前に設定された **X.509 Distinguished Name (DN)** フィールドで証明書を発行する必要があります。これを実行するかどうかについては企業レベルで検討する必要があります。

たとえば、次のような情報を使用するとします。

```
C=USA,ST=North Carolina,L=Raleigh,O=Red Hat,CN=name_of_client
```

2. プライベートキーを作成します。

```
# certtool --generate-privkey > clientkey.pem
```

3. まず **client.info** という名前のテンプレートファイルを作成して、認証局のプライベートキーの署名を生成します。ファイルには次の行が含まれます (地域や場所に応じてフィールドをカスタマイズしてください)。

```
country = USA
state = North Carolina
locality = Raleigh
organization = Red Hat
cn = client1
tls_www_client
encryption_key
signing_key
```

4. 次のコマンドで証明書に署名します。

```
# certtool --generate-certificate --load-privkey clientkey.pem --
load-ca-certificate cacert.pem \ --load-ca-privkey cakey.pem --
template client.info --outfile clientcert.pem
```

5. 証明書をクライアントマシンにインストールします。

```
# cp clientkey.pem /etc/pki/libvirt/private/clientkey.pem
# cp clientcert.pem /etc/pki/libvirt/clientcert.pem
```

## 19.4. VNC サーバーの設定

Virtual Network Computing (VNC) を使ってホストとゲストマシン間のグラフィカルデスクトップ共有を設定するには、接続先のゲスト上に VNC サーバーを設定する必要があります。そのためには、ゲストの XML ファイルの **devices** 要素でグラフィックタイプが VNC に指定されている必要があります。詳細は、「[グラフィカルフレームバッファ](#)」を参照してください。

VNC サーバーに接続するには、**virt-viewer** ユーティリティーまたは **virt-manager** インターフェースを使います。

## 19.5. NSS による仮想マシンのリモート管理の強化

Red Hat Enterprise Linux 7.3 以降では、libvirt の Network Security Services (NSS) モジュールを使用して、他のリモートログインサービスと共に SSH、TLS、SSL を使用してゲストに接続するのがより簡単になりました。さらにこのモジュールによる **ping** などのホスト名変換を使用する各種ユーティティーへの各種のメリットがあります。

この機能を使用できるようにするには、**libvirt-nss** パッケージをインストールします。

```
# yum install libvirt-nss
```



### 注記

libvirt-nss がインストールできない場合は、Red Hat Enterprise Linux の **Optional** リポジトリが有効になっていることを確認してください。詳細については、『[Red Hat Enterprise Linux 7 システム管理者のガイド](#)』を参照してください。

最後に、以下の例に示されるように **libvirt** を **/etc/nsswitch.conf** ファイルの **hosts** 行に追加してモジュールを有効にします。

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

**hosts** 行に表示された順に、各モジュールが指定されたリモートゲストを検出するために参照されます。結果として、libvirt の NSS モジュールは、ホストドメイン名を IP アドレスに変換するモジュールに追加されます。これにより、たとえば静的 IP アドレスを設定せずにゲストの **ホスト名** の値を使うだけで、NAT モードのリモートゲストに接続できるようになります。



```
# ssh root@guest-hostname
root@guest-hostname's password:
Last login: Thu Aug 10 09:12:31 2017 from 192.168.122.1
[root@guest1-rhel7 ~]#
```



## 注記

ゲストのホスト名は、例えば **virsh list** で表示されるゲスト名とは異なる場合があります。ゲストのホスト名を表示または設定するには、**hostnamectl** コマンドを使用します。

## 第20章 仮想マシンマネージャー (VIRT-MANAGER) を使用したゲストの管理

この章では、仮想マシンマネージャー (**virt-manager**) のウィンドウ、ダイアログボックス、GUI コントロールなどについて説明します。

**virt-manager** は、ホストシステムおよびリモートのホストシステム上のハイパーバイザー群やゲスト群をグラフィカルに表示することができます。**virt-manager** は次のような仮想化管理に関する作業を行うことができます。

- ゲストの定義と作成
- メモリーの割り当て
- 仮想 CPU の割り当て
- 動作性能の監視
- ゲストの保存、復元、一時停止と開始、シャットダウンと起動
- テキストコンソールとグラフィカルコンソールへのリンク
- ライブマイグレーションとオフラインマイグレーション



### 重要

使用しているユーザーについて留意してください。1 ユーザーを使用してゲスト仮想マシンを作成する場合、別のユーザーを使用してそれについての情報を取得することはできません。これは、**virt-manager** で仮想マシンを作成する場合はとくに重要になります。デフォルトユーザーは、とくに指定がない場合は **root** になります。**virsh list --all** コマンドを使用して仮想マシンを一覧表示できない場合には、仮想マシンの作成に使用したユーザーとは異なるユーザーを使用してコマンドを実行したことが原因である可能性があります。

### 20.1. VIRT-MANAGER の起動

**virt-manager** セッションを開始するには、アプリケーションメニューを開き システムツールメニュー、仮想マシンマネージャー (**virt-manager**) の順で選択します。

**virt-manager** のメインウィンドウが開きます。

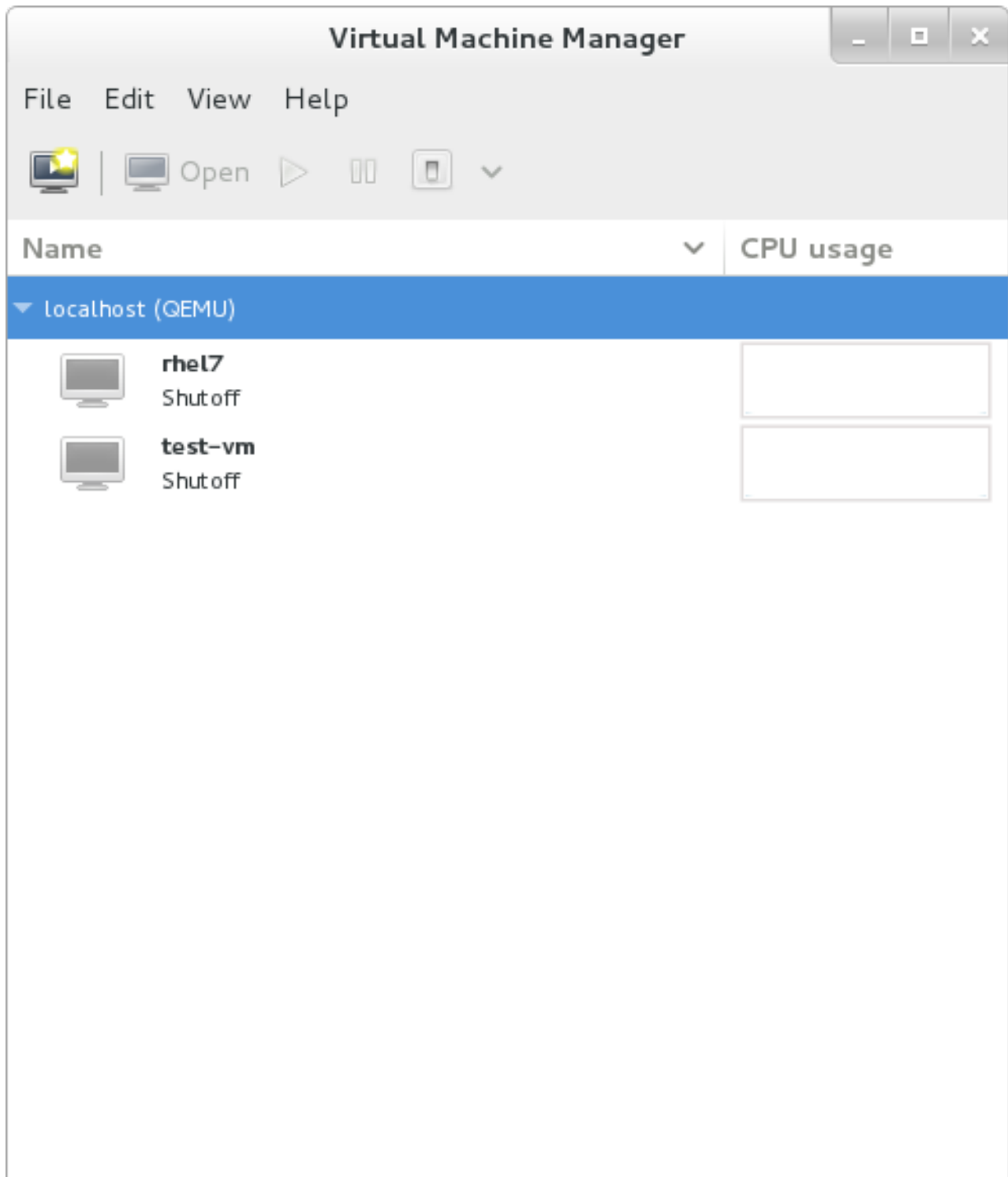


図20.1 virt-manager の起動

または、以下のコマンドで示すように **ssh** を実行して **virt-manager** をリモートで起動することもできます。

```
# ssh -X host's address  
[remotehost]# virt-manager
```

**ssh** を使った仮想マシンとホストの管理については、「[SSH によるリモート管理](#)」で詳しく説明しています。

## 20.2. 仮想マシンマネージャーのメインウィンドウ

このメインウィンドウには、実行中のゲストとゲストによって使用されるリソースがすべて表示されます。ゲスト名をダブルクリックしてゲストを選択します。

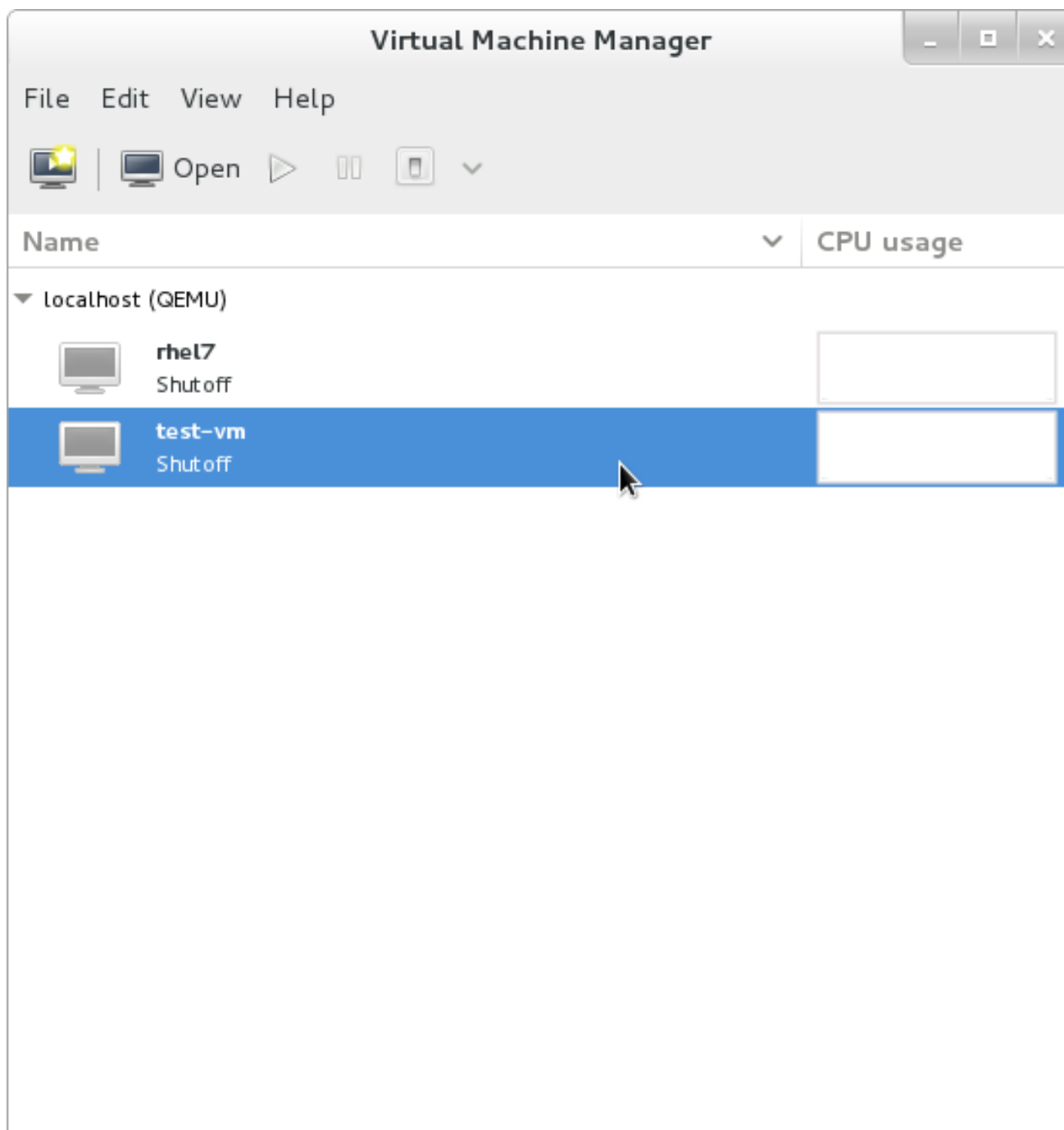


図20.2 仮想マシンマネージャーのメインウィンドウ

### 20.3. 仮想ハードウェアの詳細ウィンドウ

仮想ハードウェアの詳細ウィンドウには、ゲストに設定されている仮想ハードウェアに関する情報が表示されます。仮想ハードウェアのリソースの追加、削除、編集はこのウィンドウで行うことができます。ツールバー内にあるアイコンをクリックすると、仮想ハードウェアの詳細ウィンドウが表示されます。

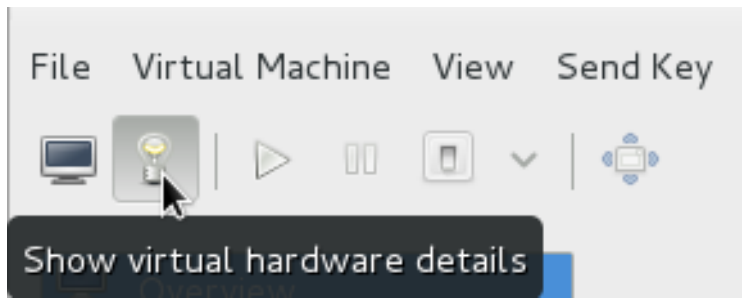


図20.3 仮想ハードウェアの詳細アイコン

上記に示すアイコンをクリックすると仮想ハードウェアの詳細ウィンドウが表示されます。

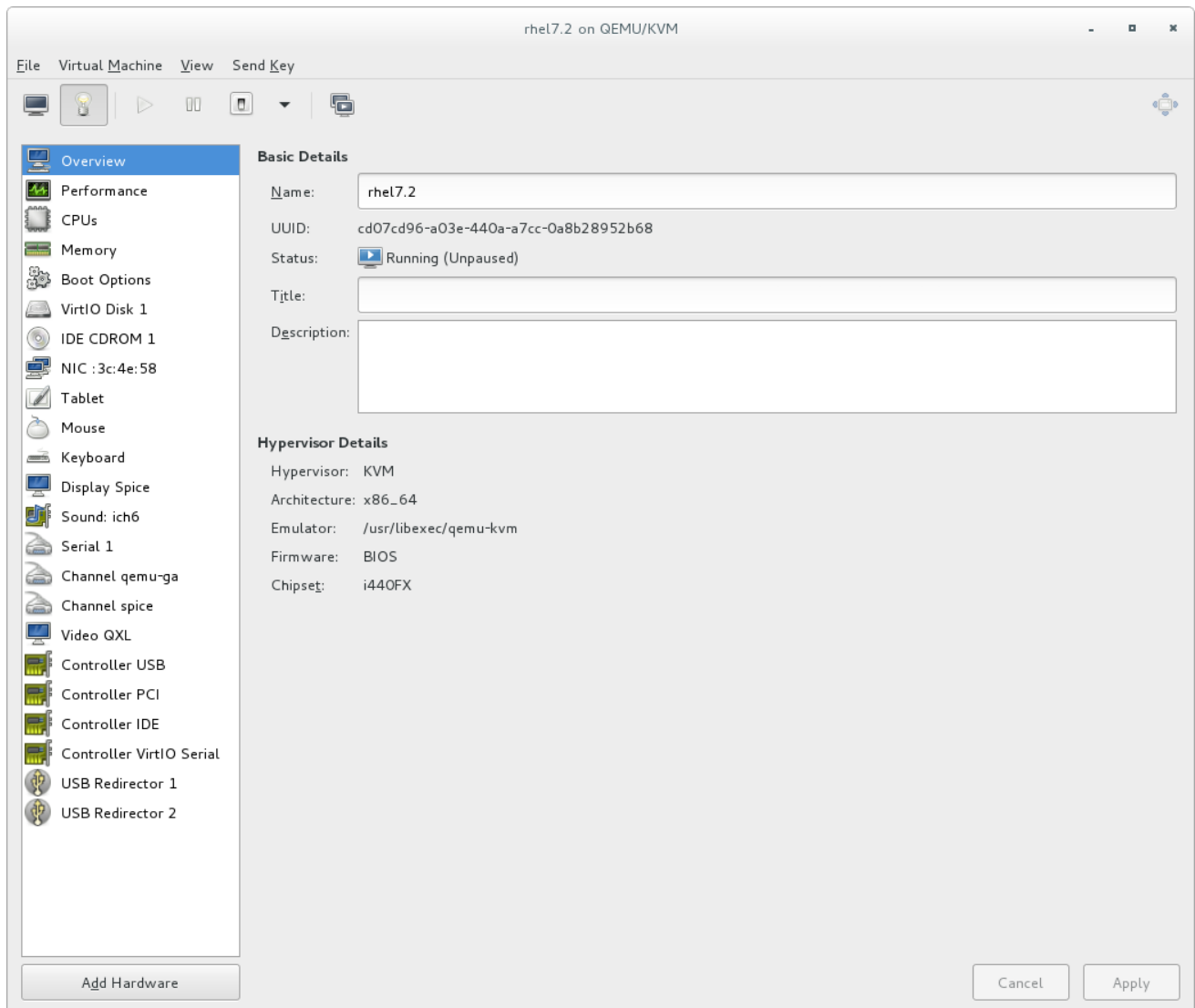


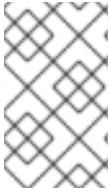
図20.4 仮想ハードウェアの詳細ウィンドウ

### 20.3.1. 起動オプションのゲスト仮想マシンへの適用

`virt-manager` を使用すると、ゲスト仮想マシンの起動時の動作を選択できます。起動オプションはゲスト仮想マシンが再起動されるまで有効になりません。仮想マシンを変更の電源を加える前にオフにするか、またはマシンを後で再起動することができます。これらのオプションのいずれも実行しない場合、次のゲストの再起動時に変更が発生します。

#### 手順20.1 起動オプションの設定





## 注記

USB デバイスをゲスト仮想マシンに割り当てるためには、まずそれをホスト物理マシンに割り当て、デバイスが機能することを確認する必要があります。ゲストが実行中の場合は、シャットダウンしてからこれを実行する必要があります。

### 手順20.2 Virt-Manager を使用した USB デバイスの割り当て

1. ゲスト仮想マシンの「仮想マシンの詳細」画面を開きます。
2. **ハードウェアを追加** をクリックします。
3. **新規の仮想ハードウェアの追加 (Add new virtual hardware)** ポップアップから **USB ホストデバイス (USB Host Device)** を選択し、一覧から割り当てるデバイスを選択して**完了 (Finish)** をクリックします。

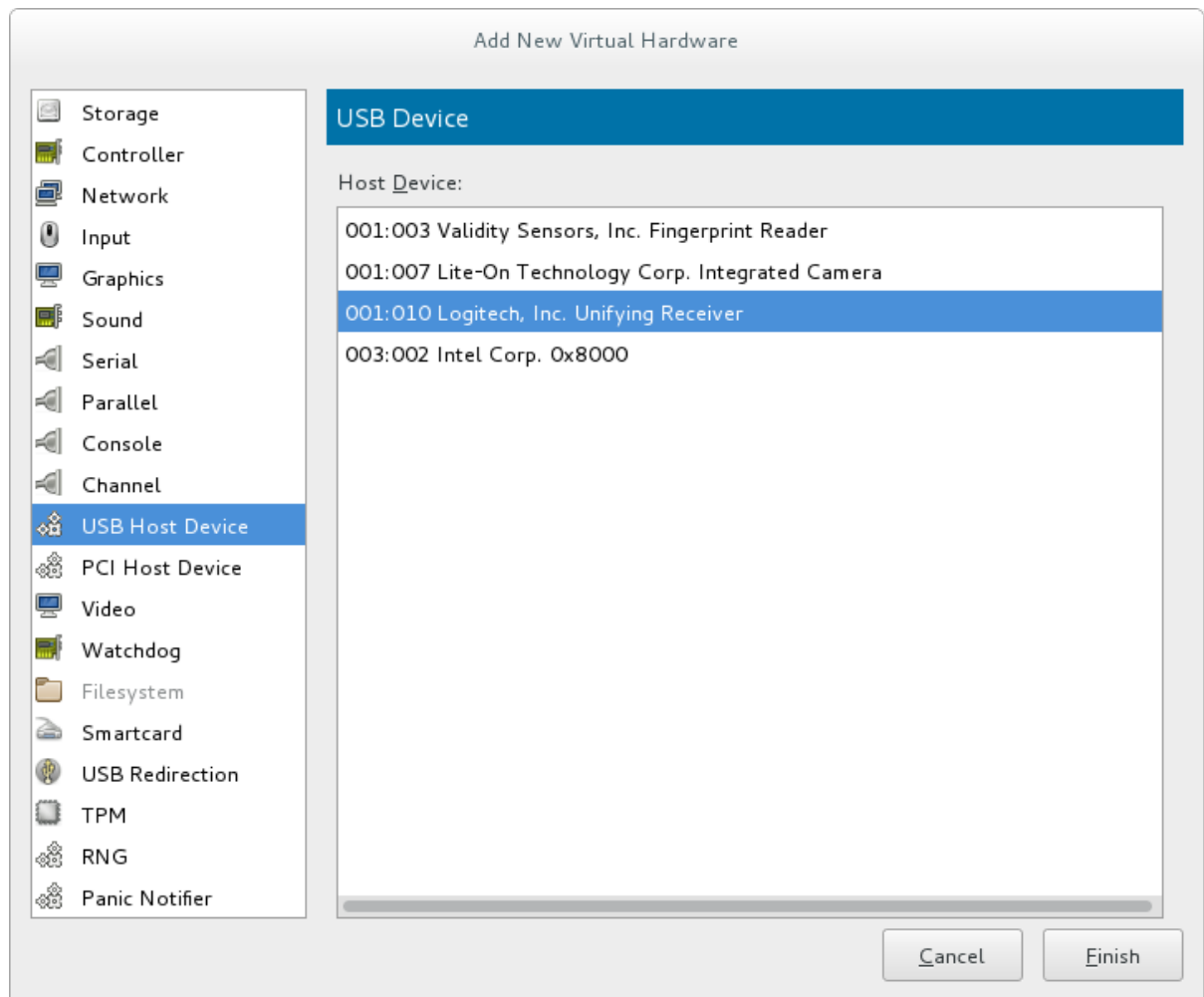


図20.6 USB デバイスの追加

4. ゲスト仮想マシンで USB デバイスを使用するには、まずゲスト仮想マシンを起動します。

### 20.3.3. USB リダイレクト

USB リダイレクトは、ホスト物理マシンがデータセンターで実行されている場合に最適に使用されます。ユーザーは、ローカルマシンまたはシンクライアントからゲスト仮想マシンに接続します。このローカルマシンには、1つの SPICE クライアントがあります。ユーザーは、任意の USB デバイスをシ

ンククライアントに割り当てることができ、SPICE クライアントはデータセンターのホスト物理マシンにこのデバイスをリダイレクトするので、これをシンククライアント上で実行されているゲスト仮想マシンでを使用することができます。

### 手順20.3 USB デバイスのリダイレクト

1. ゲスト仮想マシンの「仮想マシンの詳細」画面を開きます。
2. **ハードウェアを追加** をクリックします。
3. **新規の仮想ハードウェアの追加 (Add new virtual hardware)** ポップアップで、**USB Redirection** を選択します。**Type** ドロップダウンメニューで **Spice channel** を選択し、**完了 (Finish)** をクリックします。

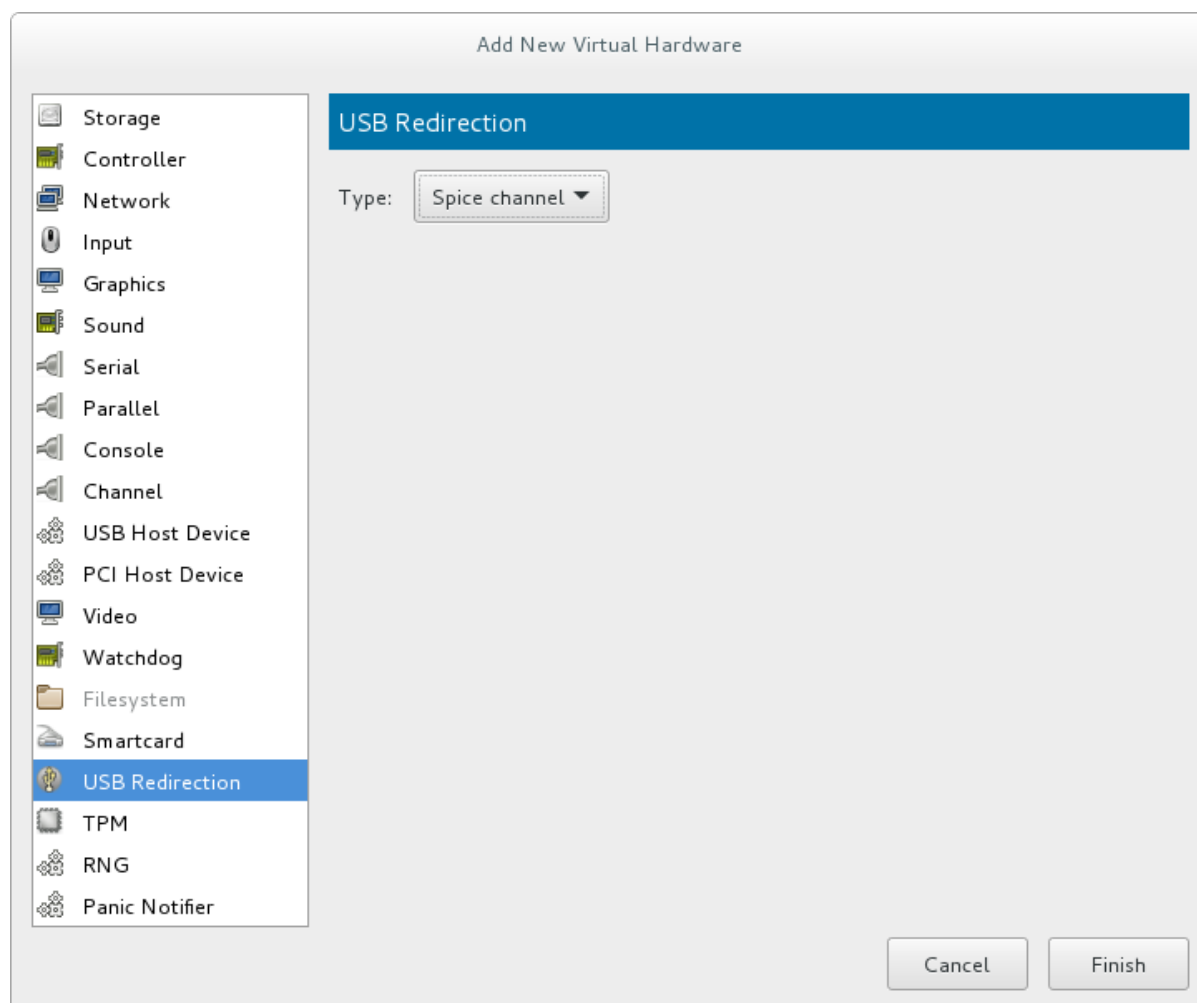


図20.7 新たな仮想ハードウェア追加ウィザード

4. 仮想マシンメニューを開いて、**USB デバイスのリダイレクト** を選択します。ポップアップウィンドウが USB デバイスの一覧と共に開かれます。





図20.8 USB デバイスの選択

5. チェックボックスにチェックを付けてリダイレクトする USB デバイスを選択し、**OK** をクリックします。

## 20.4. 仮想マシンのグラフィカルコンソール

このウィンドウにはゲストのグラフィカルコンソールが表示されます。グラフィカルなフレームバッファのエクスポートには複数の異なるプロトコルを使用することができます。**virt-manager** に対応しているのは **VNC** と **SPICE** になります。認証を求めるよう仮想マシンを設定している場合は、ディスプレイが表示される前に仮想マシンのグラフィカルコンソールによってパスワードの入力が求められます。

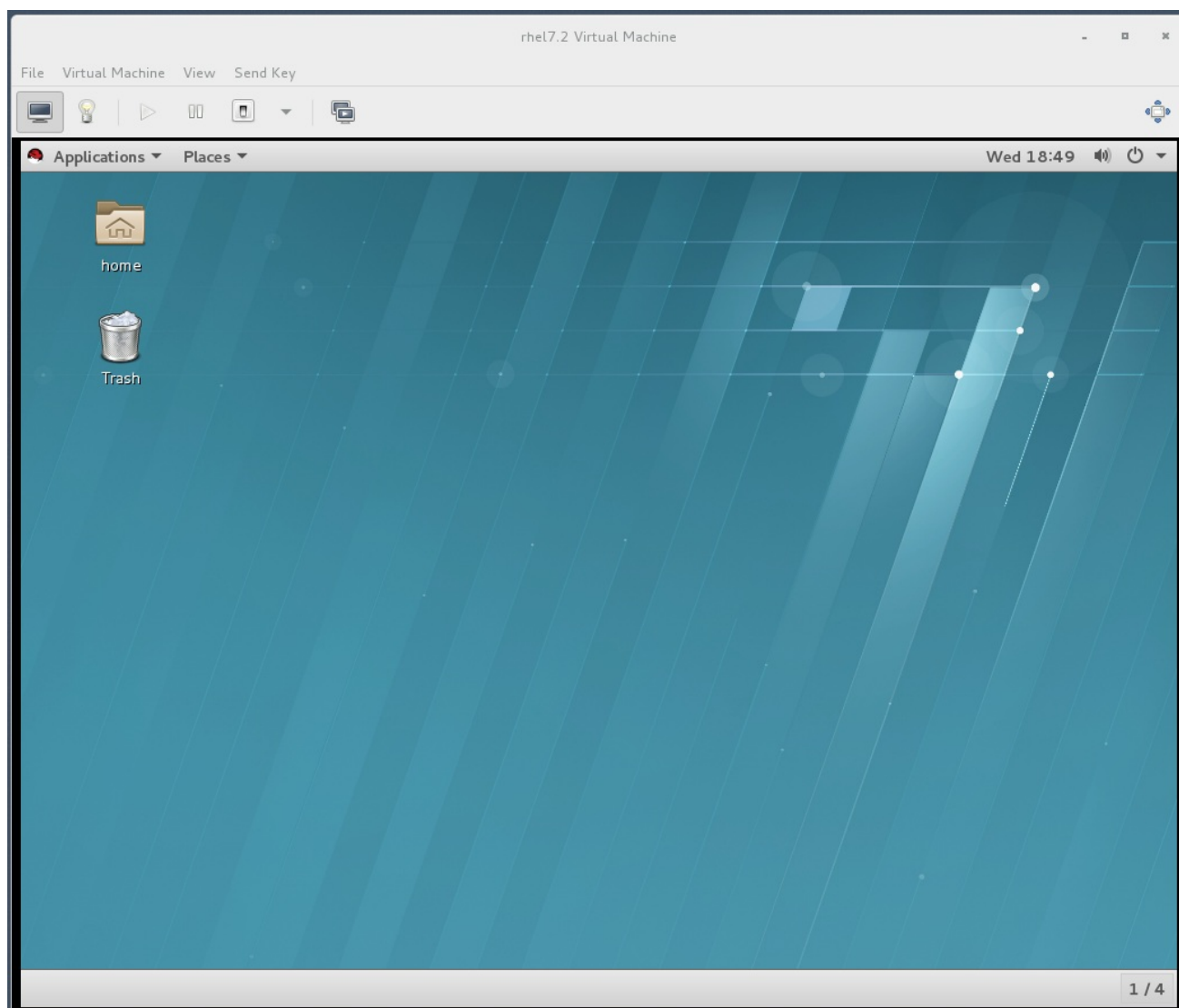


図20.9 グラフィカルのコソールウィンドウ

## 注記

多くのセキュリティー専門家の間では VNC は安全ではないと考えられていますが、Red Hat Enterprise Linux での仮想化に VNC を使用する際の安全性を確保する変更がいくつか加えられています。ゲストマシンがリッスンするのはローカルホストのループバックアドレスのみになります (**127.0.0.1**)。これにより、VNC 経由による仮想マシンおよび **virt-manager** にアクセスできるのはホスト上でシェルの特権を有するゲストのみになります。**virt-manager** が他のパブリックネットワークインターフェースをリッスンするよう設定し、別のメソッドを設定することもできますが、これは推奨されていません。

トラフィックを暗号化する SSH 経由でトンネル化をすることでリモートからの管理が可能になります。SSH でのトンネル化を行わずに VNC へのリモートアクセスを設定することはできますが、安全上の理由から推奨されていません。リモートからゲストの管理を行う場合は、[19章ゲストのリモート管理](#)の説明に従ってください。TLS を使用すると、ゲストとホストシステムの管理におけるエンタープライズレベルの安全性を得ることができます。

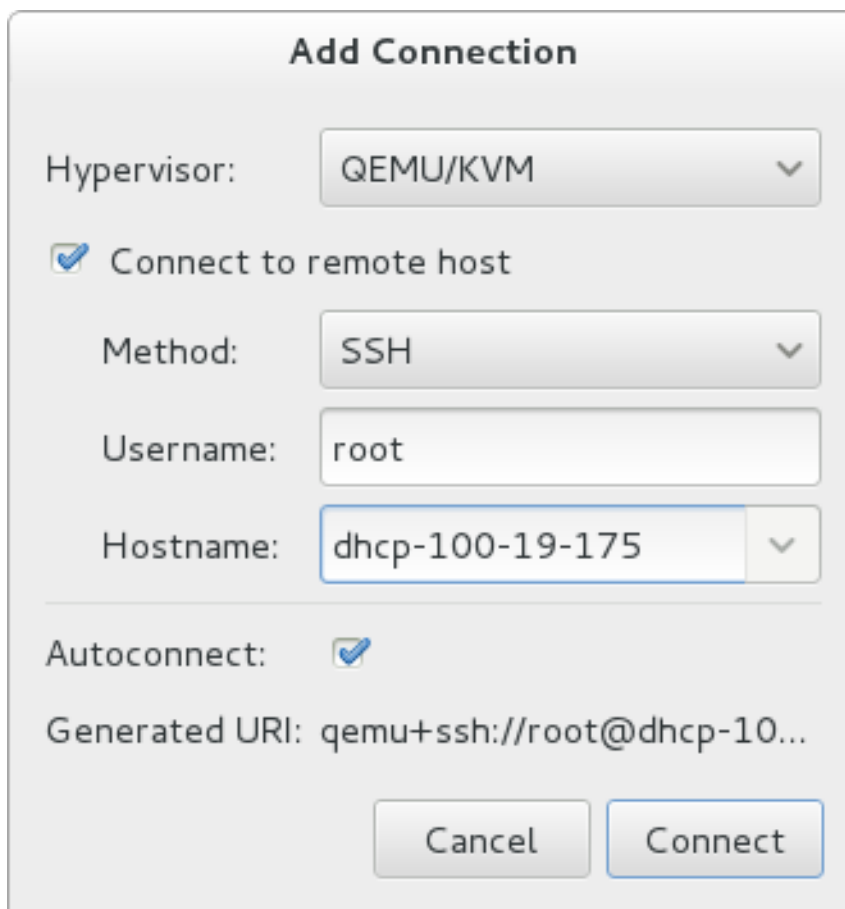
ローカルデスクトップでは、キーの組み合わせがゲストマシンに送信されないようにすることができます (**Ctrl+Alt+F1** など)。キーを送信のメニューオプションを使用するとキーの組み合わせを送信することができます。ゲストマシンのウィンドウから、キーを送信メニューをクリックして、送信するキーの組み合わせを選択します。また、このメニューでは画面の出力をキャプチャーすることもできます。

Red Hat Enterprise Linux では VNC の代替として SPICE を使用できます。

## 20.5. リモート接続の追加

**virt-manager** を使ってリモートシステムへの接続を設定する方法を説明します。

1. 新しい接続を作成するには、**ファイル** メニューを開き **接続を追加** メニューアイテムを選択します。
2. **接続を追加** ウィザードが表示されます。ハイパーバイザーを選択します。Red Hat Enterprise Linux 7 システムの場合は **QEMU/KVM** を選択します。ローカルシステムの場合は **ローカル** を選択するか、いずれかのリモート接続オプションを選択して **接続** をクリックします。この例では、SSH 経由のリモートトンネルを使用しています。これは、デフォルトのインストールで正常に動作します。リモート接続の設定方法については、[19章ゲストのリモート管理](#)を参照してください。



The screenshot shows the 'Add Connection' dialog box. It has a title bar 'Add Connection'. Inside, there are several fields and checkboxes. 'Hypervisor:' is a dropdown menu with 'QEMU/KVM' selected. Below it is a checked checkbox 'Connect to remote host'. Then 'Method:' is a dropdown menu with 'SSH' selected. 'Username:' is a text input field with 'root' entered. 'Hostname:' is a dropdown menu with 'dhcp-100-19-175' selected. Below these is an 'Autoconnect:' checkbox which is also checked. At the bottom, it shows 'Generated URI: qemu+ssh://root@dhcp-10...'. At the very bottom are two buttons: 'Cancel' and 'Connect'.

図20.10 接続の追加

3. 選択したホストの root パスワード入力が求められるのでこれを入力します。

これでリモートホストが接続され、**virt-manager** のメインウィンドウに表示されるようになります。

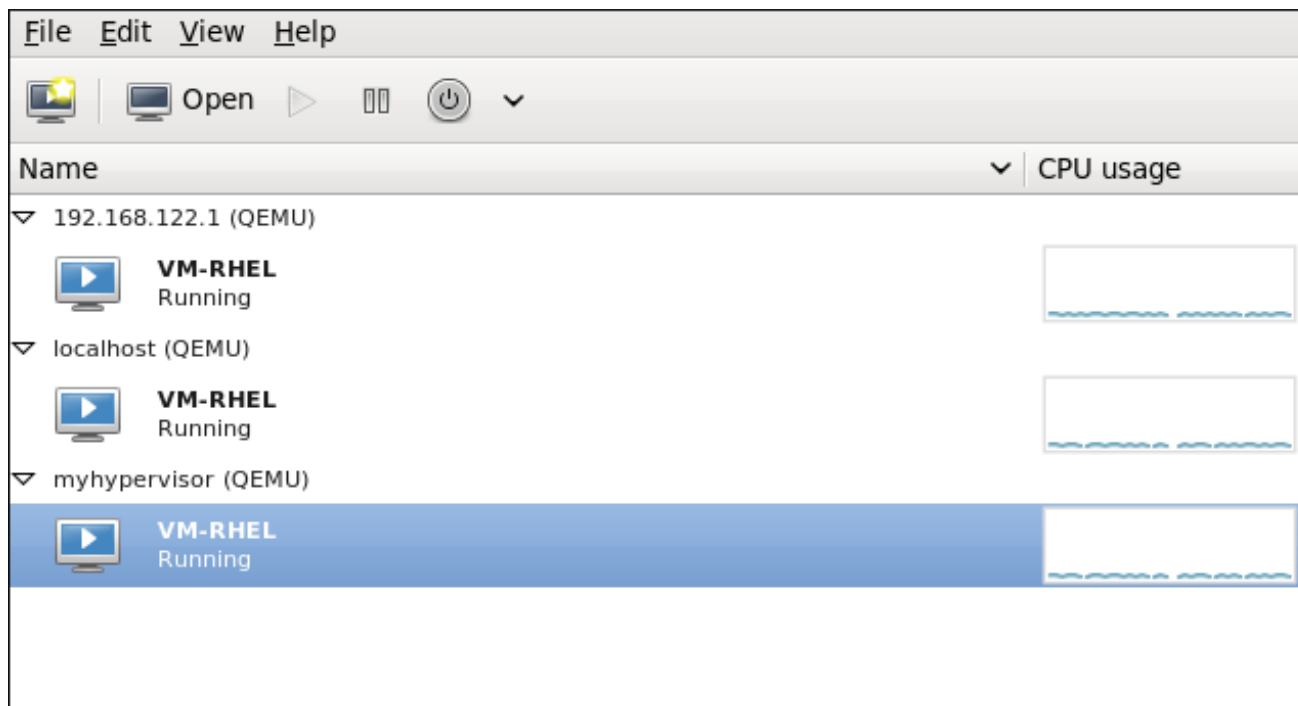


図20.11 virt-manager のメインウィンドウに表示されたリモートホスト

## 20.6. ゲストの詳細の表示

仮想マシンモニターを使用すると、システム上の仮想マシンのアクティビティー情報を表示できます。

仮想システムの詳細を表示する

1. 仮想マシンマネージャーのメインウィンドウで表示する仮想マシンを強調表示します。

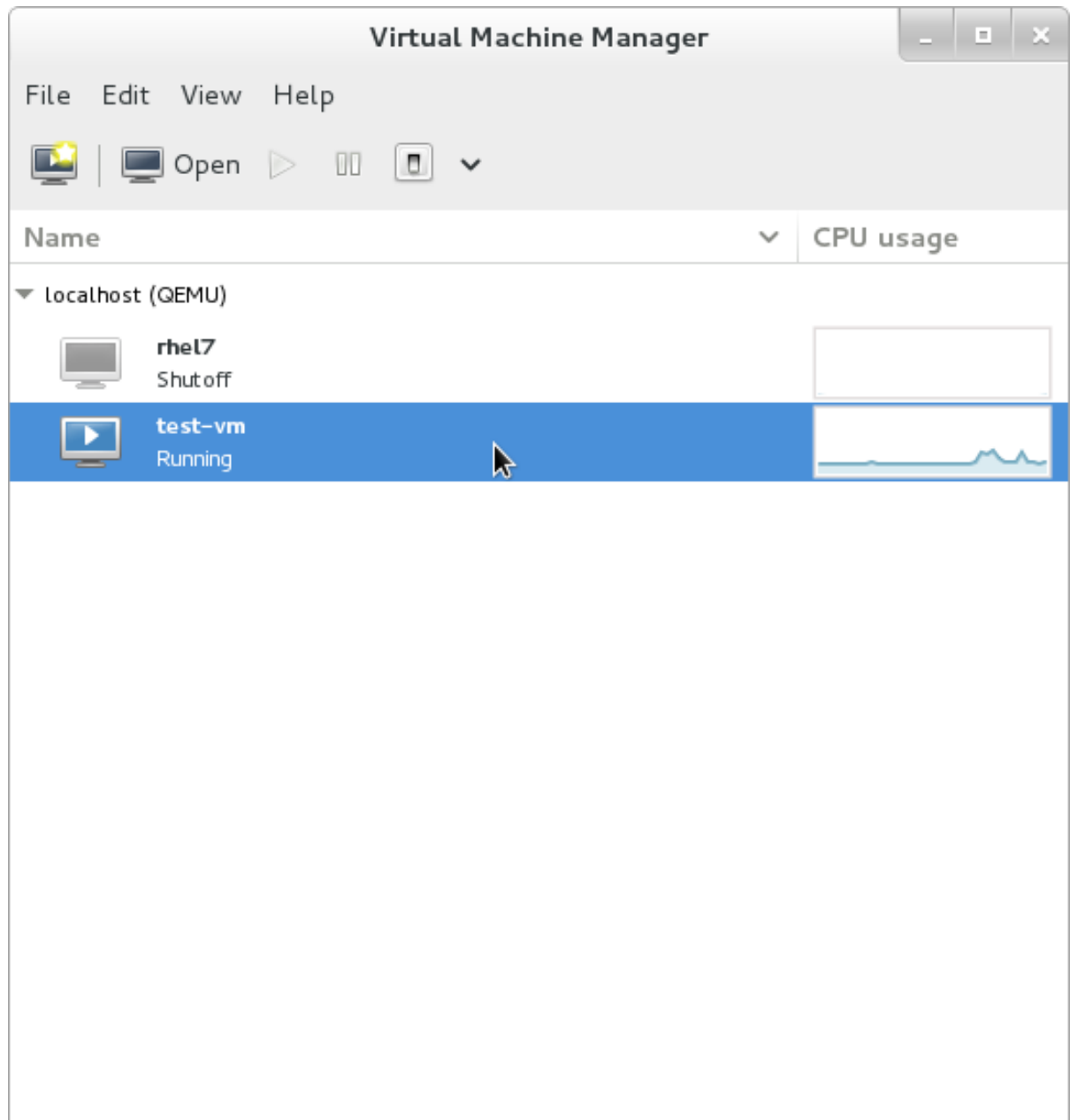


図20.12 表示する仮想マシンの選択

2. 仮想マシンマネージャーの **編集** メニューから **仮想マシンの詳細** を選択します。

仮想マシンの詳細ウィンドウを開くとコンソールが表示される場合があります。コンソールが表示される場合には、**表示** をクリックしてから **詳細** を選択します。デフォルトでは概要を示すウィンドウが開きます。このウィンドウに戻るには、左側のナビゲーションペインで **概要 (Overview)** を選択します。

**概要 (Overview)** のビューには、そのゲストの設定情報の要約が表示されます。

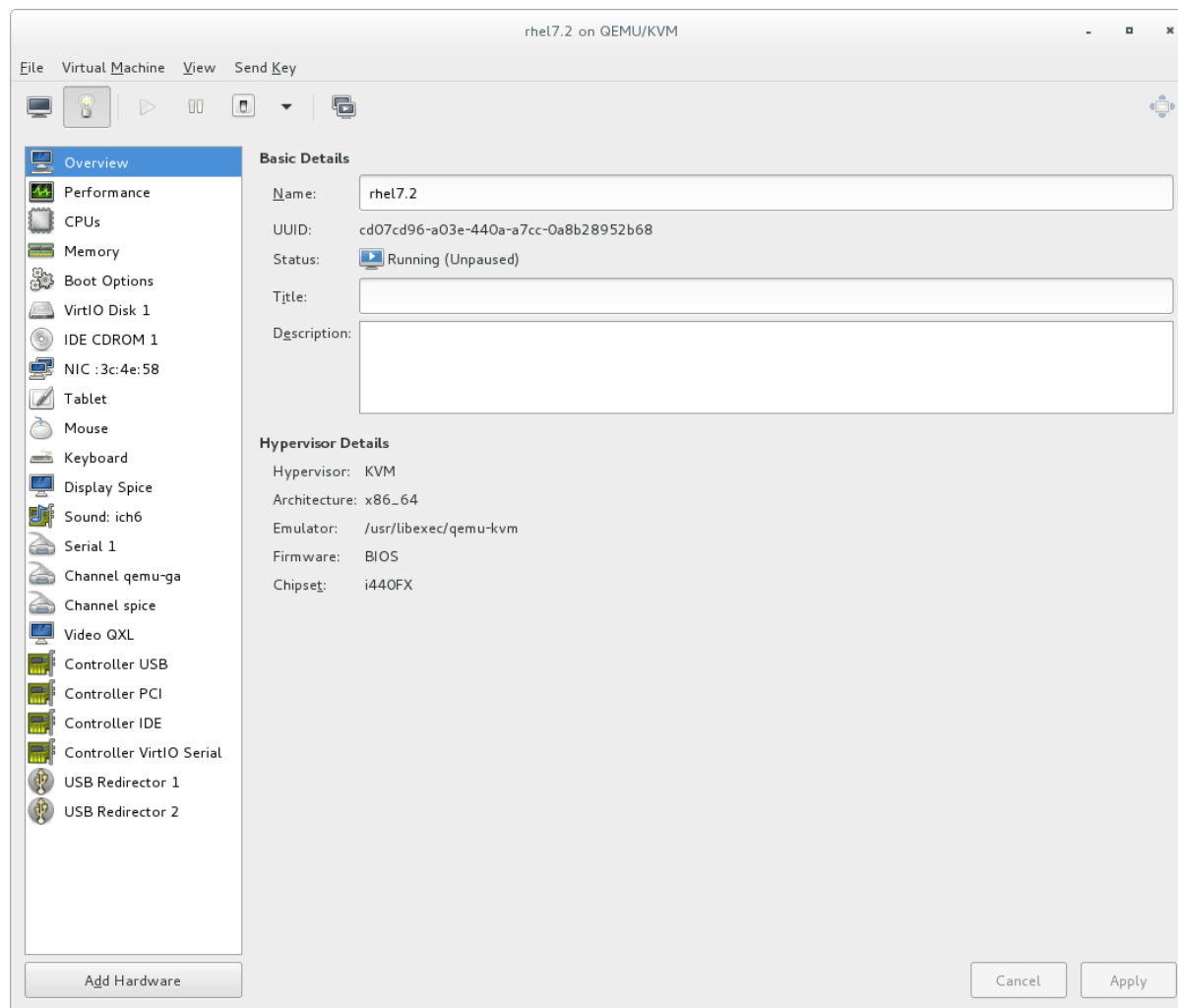


図20.13 ゲスト詳細の概要の表示

3. 左側のナビゲーションペインから **CPU (CPUs)** を選択します。**CPU (CPUs)** のビューでは、現在のプロセッサの割り当てを表示したり、変更を行ったりすることができます。

また、仮想マシンの実行中に仮想 CPU (vCPU) の数を増やすこともできます。これは **ホットプラグ** と呼ばれています。



### 重要

仮想 CPU の **ホットアンプラグ** は現在 Red Hat Enterprise Linux 7 ではサポートされていません。

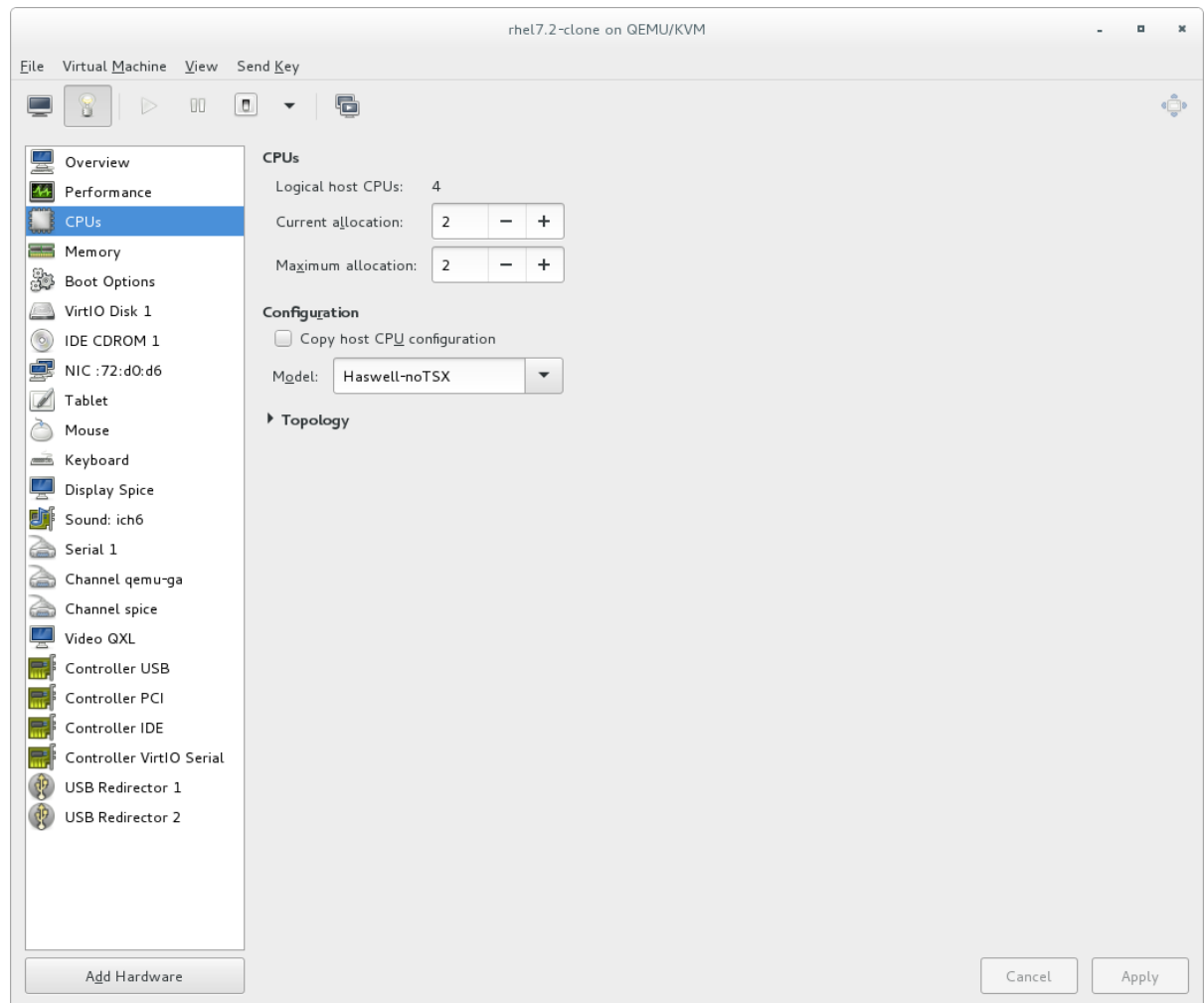


図20.14 プロセッサの割り当てパネル

4. 左側のナビゲーションペインから **メモリー (Memory)** を選択します。**メモリー (Memory)** のビューでは、現在のメモリーの割り当てを表示したり、変更を行ったりすることができます。

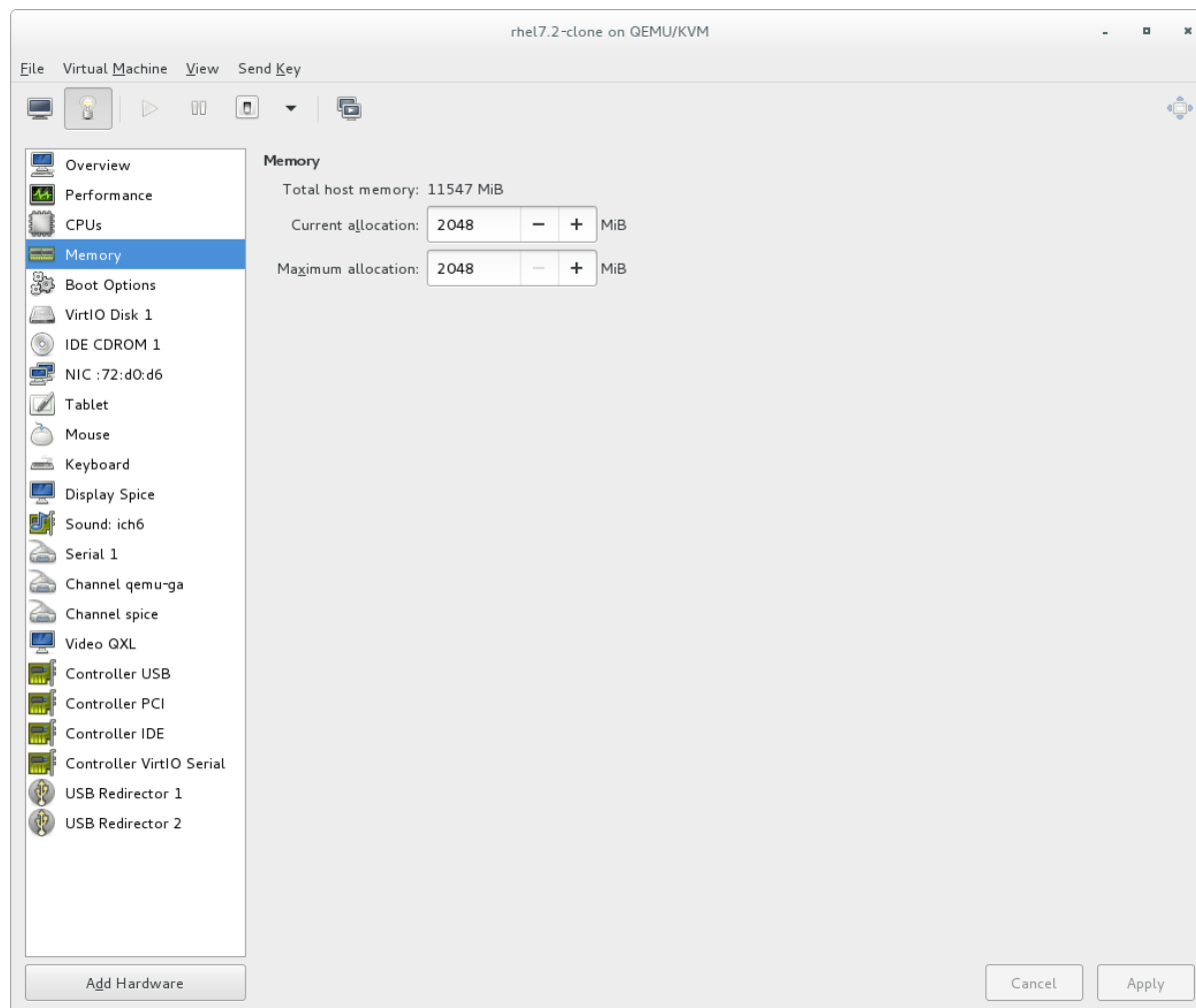


図20.15 メモリ割り当ての表示

5. 左側のナビゲーションペインから **Boot Options** を選択します。**Boot Options** のビューでは、ホストの起動時に仮想マシンが起動するかどうかや仮想マシンのブートデバイスの順序を含むブートオプションを表示したり、変更を行ったりすることができます。



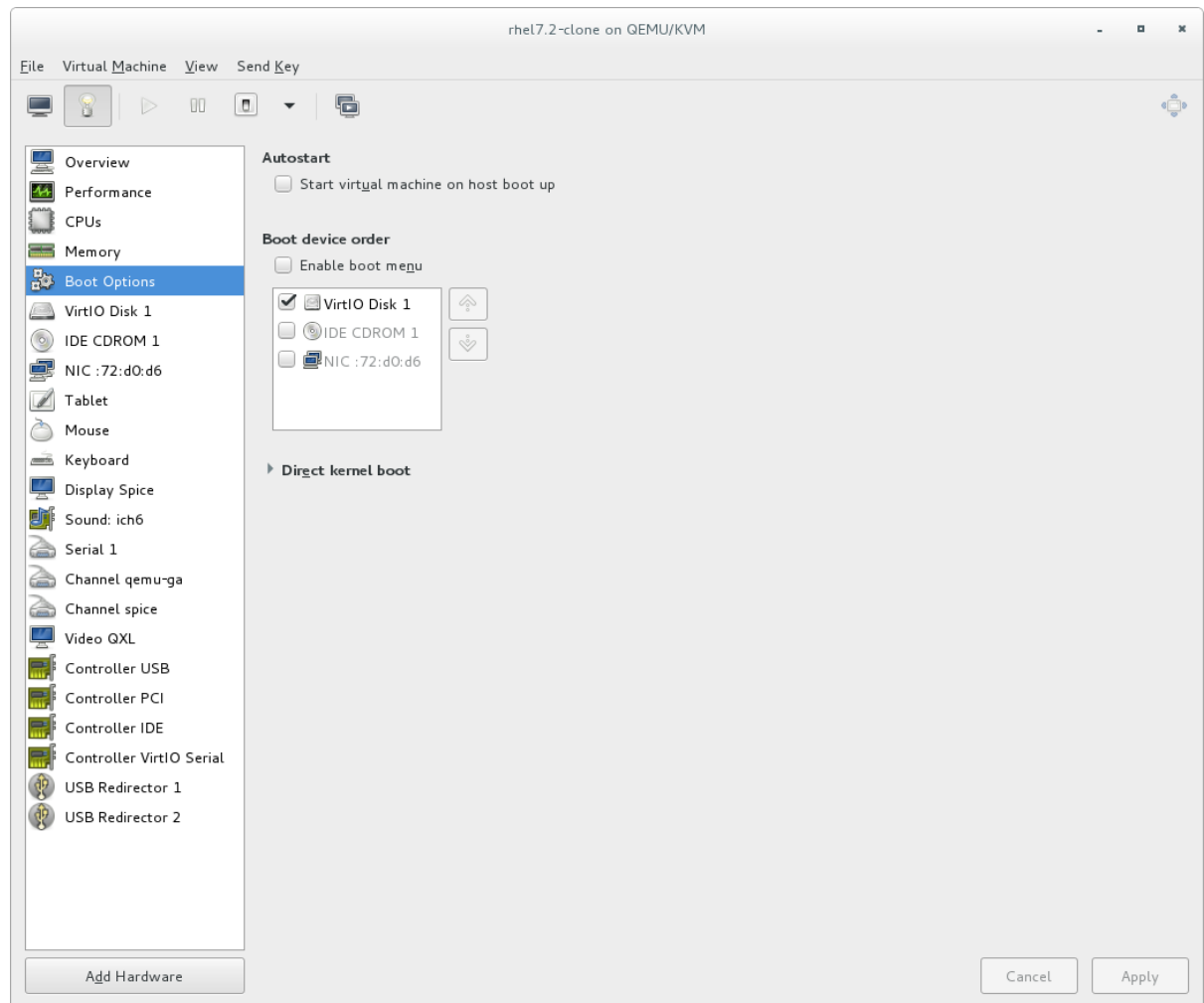


図20.16 起動オプションの表示

6. ナビゲーションペインには、その仮想マシンに接続されている各仮想ディスクが表示されます。仮想ディスクの修正や削除を行う場合はその仮想ディスクをクリックします。

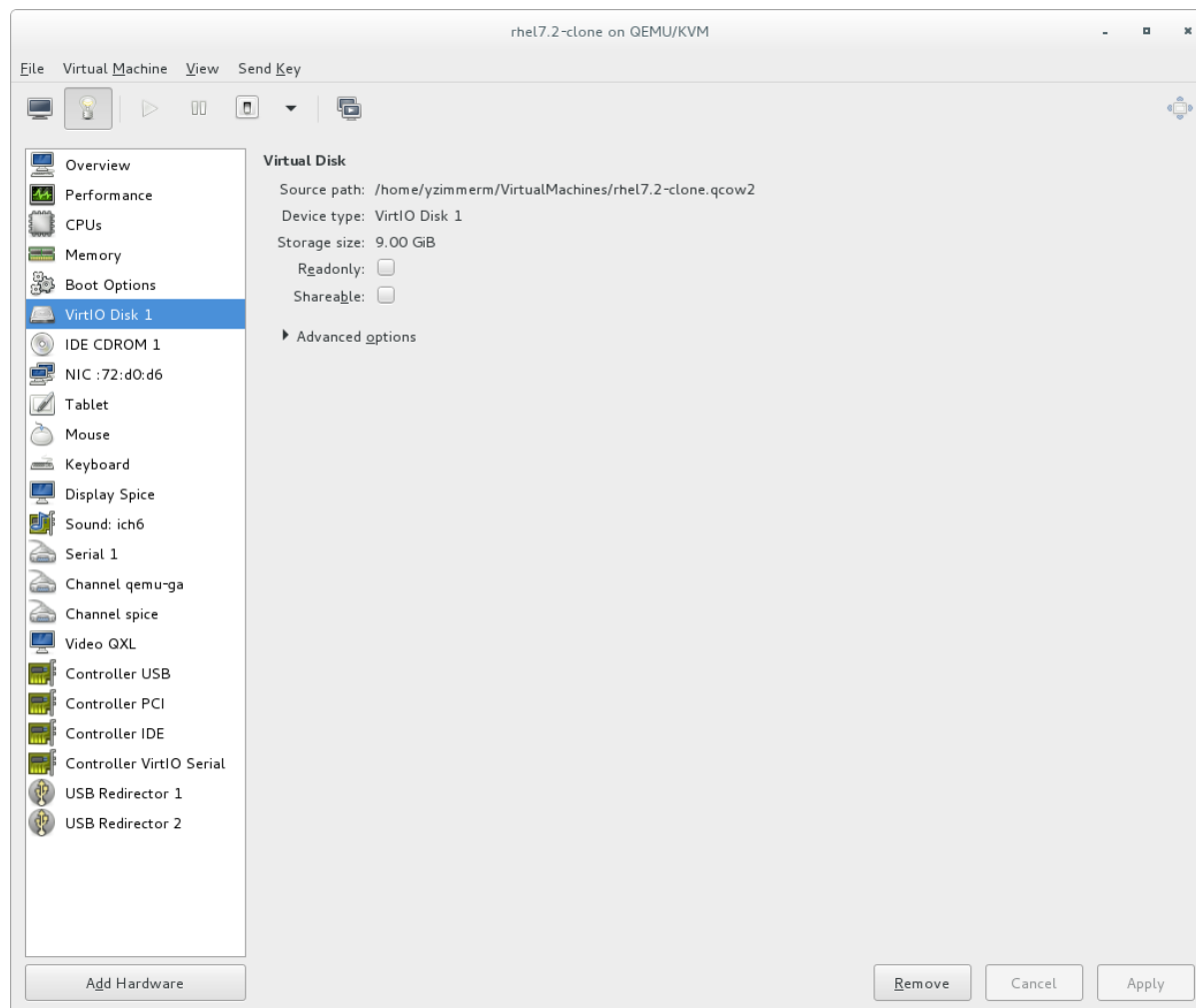


図20.17 ディスク設定の表示

7. ナビゲーションペインには、その仮想マシンに接続されている各仮想ネットワークインターフェースが表示されます。仮想ネットワークインターフェースの修正や削除を行う場合はその仮想ネットワークインターフェースをクリックします。

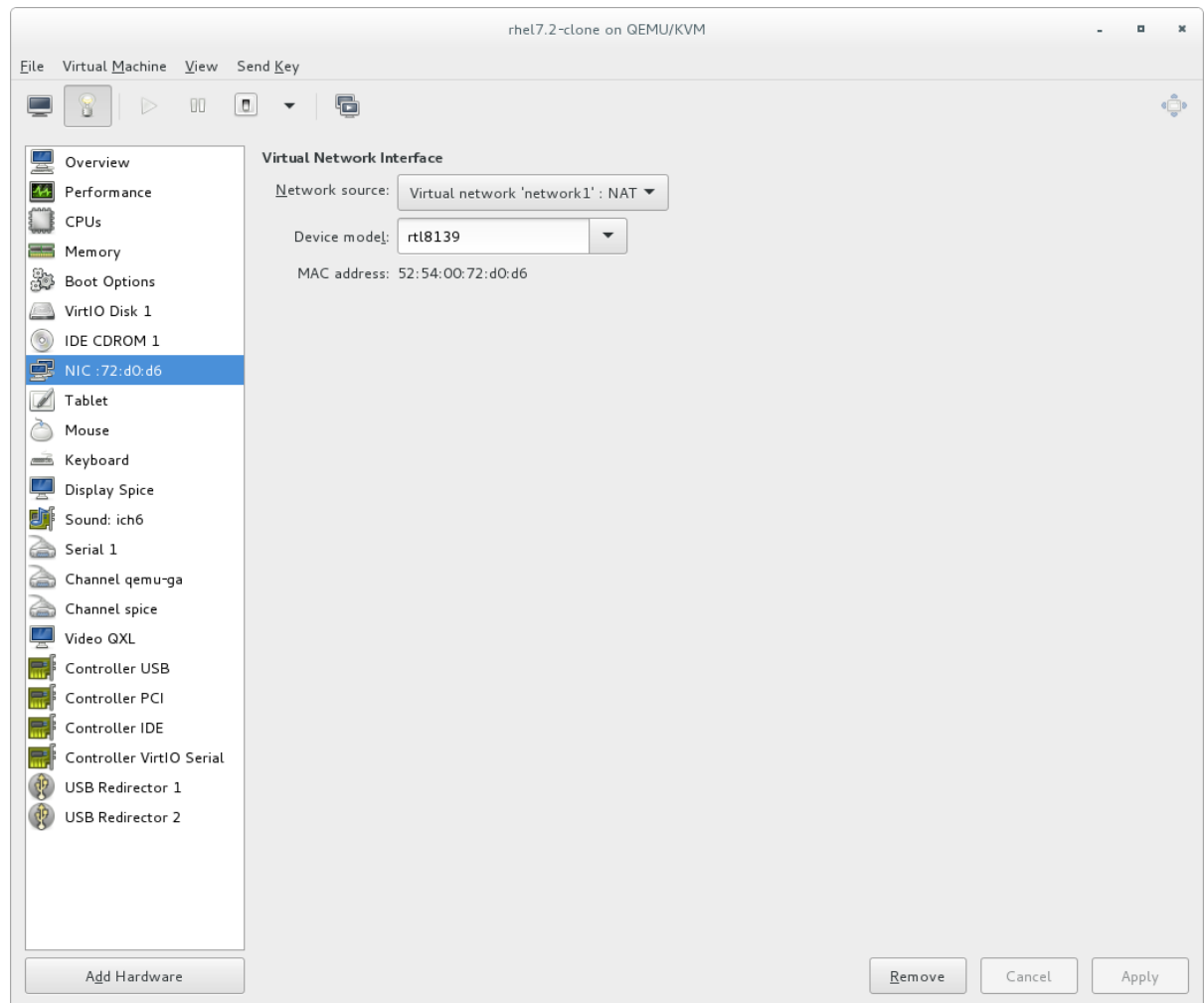


図20.18 ネットワーク設定の表示

## 20.7. スナップショットの管理


**virt-manager** を使うと、ゲストのスナップショットを作成したり、実行したり、削除したりできます。スナップショットとは、ある時点におけるゲストのハードディスク、メモリー、およびデバイス状態の保存されたイメージです。スナップショットが作成された後に、ゲストはいずれかの時点のスナップショットの設定に戻ることができます。

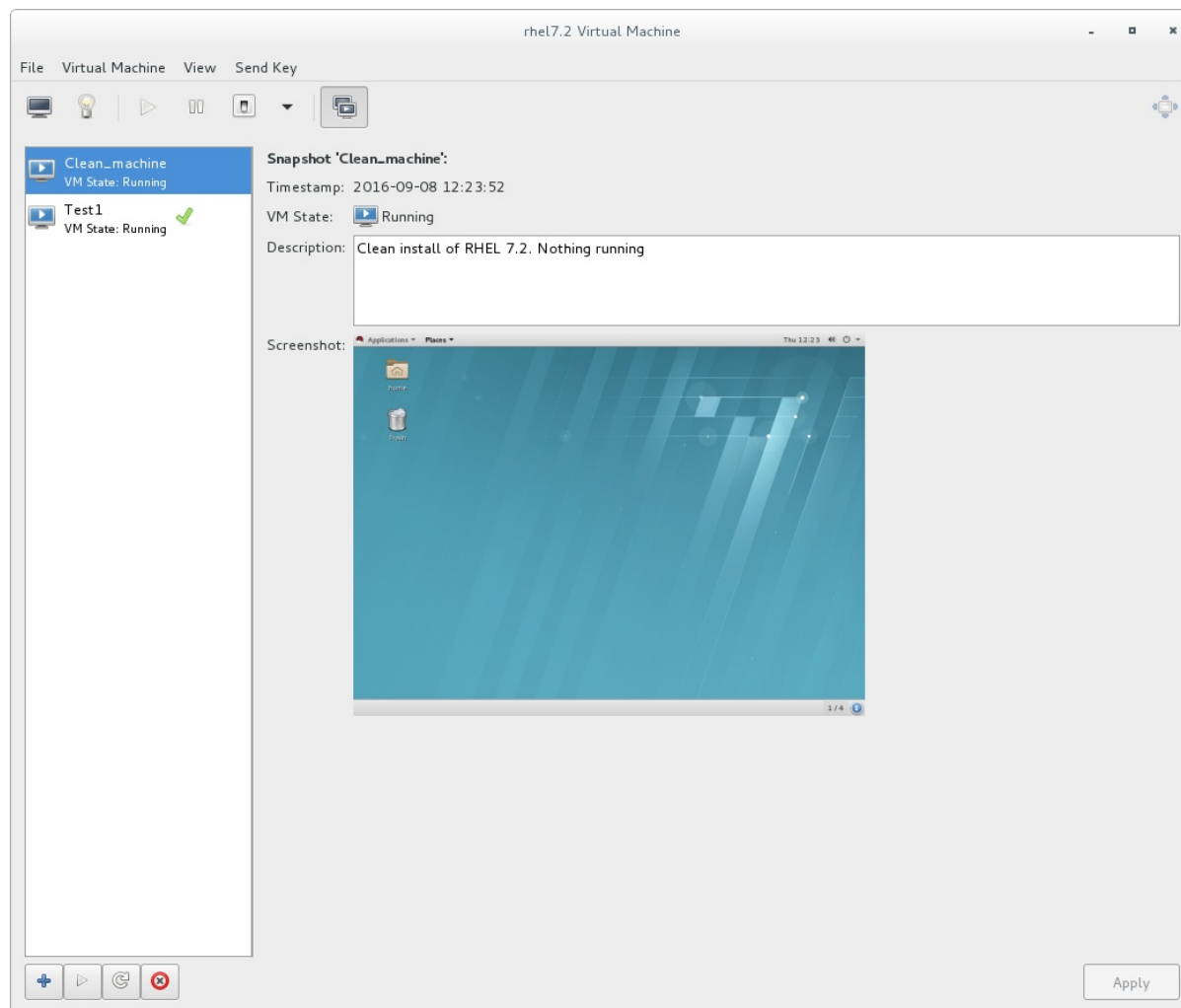



### 重要

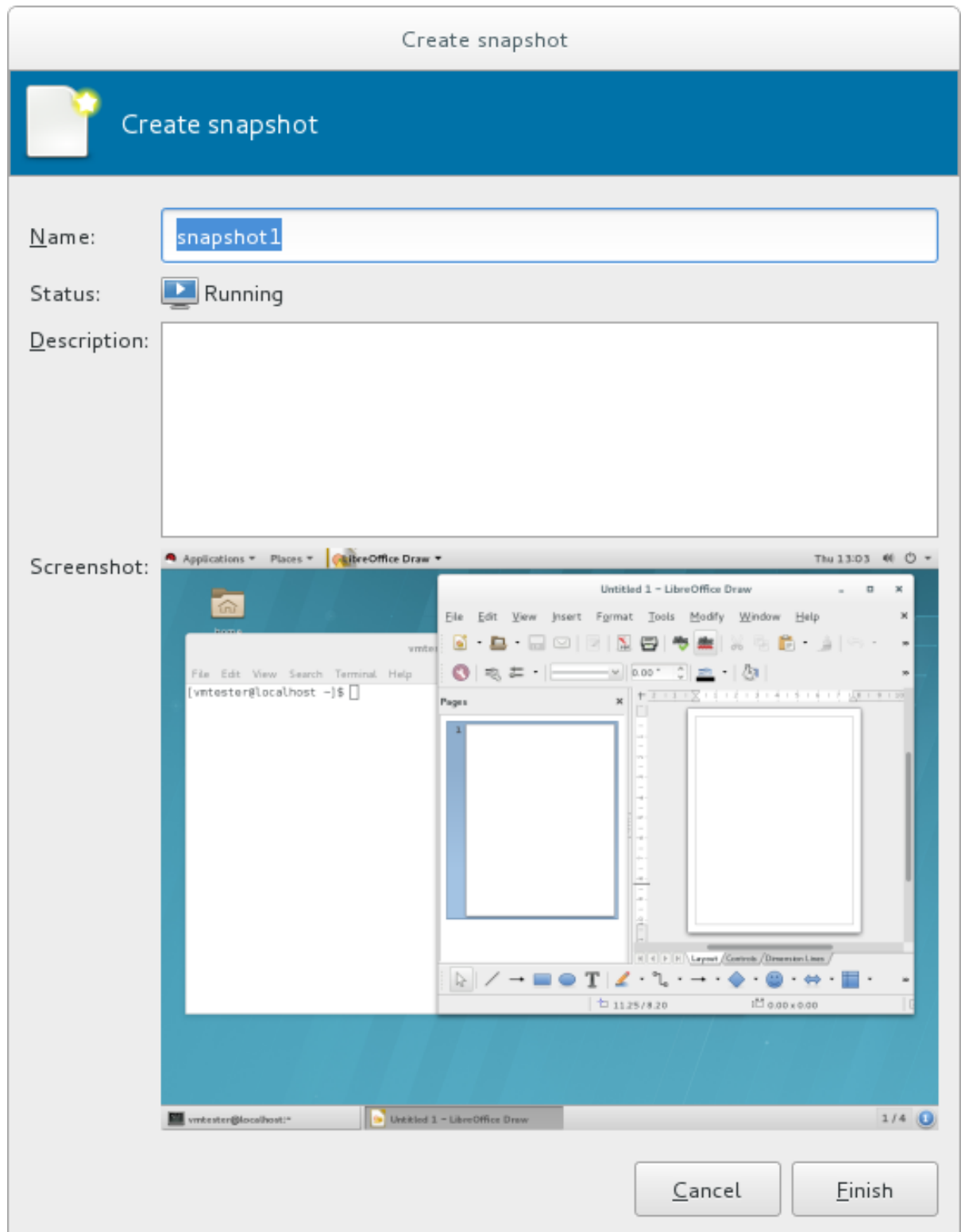
Red Hat では、他の仮想化ツールで処理されると柔軟性や信頼性が高くなるため、外部スナップショットの使用を推奨しています。ただし、現時点では **virt-manager** で外部スナップショットを作成することはできません。



外部スナップショットを作成するには、**virsh snapshot-create-as** コマンドを **--diskspec vda,snapshot=external** オプションと共に使用します。詳細は、「[libvirt による外部スナップショットの作成方法](#)」を参照してください。

- **virt-manager** でスナップショットを管理するには、ゲストコンソールで  をクリックしてスナップショット管理インターフェースを開きます。



- 新規スナップショットを作成するには、スナップショット一覧の下にある  をクリックします。スナップショット作成インターフェースでは、スナップショットの名前を入力し、オプションで説明を入力して完了 (**Finish**) をクリックします。



- ゲストをスナップショットの設定に戻すには、スナップショットを選択してから  をクリックします。
- 選択したスナップショットを削除するには、 をクリックします。



### 警告

仮想マシンの実行中にスナップショットを作成し、ロードすること(ライブスナップショットともいう)は、**qcow2** ディスクイメージでのみサポートされています。

さらに詳細のスナップショット管理について参照するには、**virsh snapshot-create** コマンドを実行します。**virsh**でのスナップショットの管理についての詳細は、「[スナップショットの管理](#)」を参照してください。

## 第21章 VIRSH を使用したゲスト仮想マシンの管理

**virsh** は、ゲスト仮想マシンを管理するためのコマンドラインツールであり、Red Hat Enterprise Linux 7 の仮想化を制御するための主要な手段として機能します。**virsh** コマンドラインツールは **libvirt** 管理 API をベースに構築されており、ゲスト仮想マシンの作成、導入、および管理に使用できます。**virsh** ユーティリティーは、仮想化管理スクリプトの作成に最適のもので、**root** 特権のないユーザーも読み取り専用モードで使用できます。**virsh** パッケージは、**libvirt-client** パッケージの一部として **yum** でインストールされます。

インストールの手順に関しては、「[仮想化パッケージの手動インストール](#)」を参照してください。実際のデモなど、**virsh** 導入全般についての情報は、[仮想化スタートガイド](#)を参照してください。本章では、使用状況に基づいて論理的な順序で設定される **virsh** コマンドについて扱います。

### 注記

**help** を使用する場合や **man** ページを参照する場合には、「**domain**」(ドメイン)という用語がゲスト仮想マシンの代わりに使用されることに注意してください。これは **libvirt** で使用される用語です。画面出力が表示される際に「**domain**」(ドメイン)が使用されている場合、これはゲストまたはゲスト仮想マシンに切り替わりません。すべての例ではゲスト仮想マシンの「**guest1**」が使用されます。すべての例でこれをゲスト仮想マシンの名前に置き換えてください。ゲスト仮想マシンの名前を作成する際には、短く覚えやすい整数 (0,1,2...)、テキスト文字列名を使用するか、または仮想マシンの完全な **UUID** も使用できます。

### 重要

どのユーザーとして使用しているかについて留意してください。あるユーザーを使用してゲスト仮想マシンを作成する場合、別のユーザーを使用してそのマシンについての情報を取得することはできません。これは、**virt-manager** で仮想マシンを作成する場合は特に重要になります。デフォルトユーザーは、特に指定がない場合は **root** になります。**virsh list --all** コマンドを使用して仮想マシンを一覧表示できない場合には、仮想マシンの作成に使用したユーザーとは異なるユーザーを使用してコマンドを実行したことが原因である可能性があります。詳細は、[重要](#) を参照してください。

## 21.1. ゲスト仮想マシンの状態および種類

**virsh** コマンドのいくつかは、ゲスト仮想マシンの状態によって影響を受けます。

- **一時的**- 一時的なゲストは、再起動後は存続しません。
- **永続的**- 永続的なゲスト仮想マシンは再起動後も存続し、削除されるまで存続します。

仮想マシンのライフサイクル期間中、**libvirt** は以下の状態のいずれかにゲストを分類します。

- **Undefined (未定義)**: これは、ゲスト仮想マシンが定義されていないか、または作成されていない状態です。そのため、**libvirt** はこの状態にあるゲストを認識せず、この状態のゲスト仮想マシンについて報告しません。
- **Shut off (シャットオフ)**: これは、ゲスト仮想マシンが定義されているものの、実行されていない状態です。永続的なゲストのみがシャットオフと見なされます。そのため、一時的なゲスト仮想マシンがこの状態に置かれると存在なくなります。
- **Running (実行中)**: この状態のゲスト仮想マシンは定義されており、現在機能している状態です。この状態は一時的および永続的なゲスト仮想マシンの両方に使用できます。

- **Paused (一時停止):** ゲスト仮想マシンのハイパーバイザーでの実行が一時停止になっているか、またはその状態が再開されるまで一時的に保存されている状態です。この状態のゲスト仮想マシンは、それらが一時停止になっていることを認識せず、再開される際に経過した時間を通知しません。
- **Saved (保存):** この状態は一時停止の状態と似ていますが、ゲスト仮想マシンの設定は永続的なストレージに保存されます。この状態のゲスト仮想マシンはそれが一時停止になっていることを認識せず、復元時に経過した時間を通知しません。

## 21.2. VIRSH バージョンの表示

**virsh version** コマンドは現在の **libvirt** のバージョンを表示し、ローカル **virsh** クライアントについての情報を表示します。以下のようになります。

```
$ virsh version
Compiled against library: libvirt 1.2.8
Using library: libvirt 1.2.8
Using API: QEMU 1.2.8
Running hypervisor: QEMU 1.5.3
```

**virsh version --daemon** は、ホストで実行されている **libvirt** デーモンについての情報を含む **libvirtd** バージョンおよびパッケージについての情報を取得するのに役立ちます。

```
$ virsh version --daemon
Compiled against library: libvirt 1.2.8
Using library: libvirt 1.2.8
Using API: QEMU 1.2.8
Running hypervisor: QEMU 1.5.3
Running against daemon: 1.2.8
```

## 21.3. ECHO を使用したコマンドの送信

**virsh echo [--shell][--xml] arguments** コマンドは、指定された形式で指定された引数を表示します。使用できる形式は、**--shell** および **--xml** です。照会したそれぞれの引数はスペースで区切られて表示されます。**--shell** オプションは、必要な場合に一重引用符を使用してフォーマットされた出力を生成するため、コマンドとして **bash** モードにコピーおよび貼り付けを実行するのに適しています。**--xml** 引数が使用される場合、出力は XML で使用できるようにフォーマットされるため、ゲストの設定用に保存し、使用できます。

## 21.4. VIRSH CONNECT を使用したハイパーバイザーへの接続

**virsh connect [hostname-or-URI] [--readonly]** コマンドは、**virsh** を使用してローカルのハイパーバイザーセッションを開始します。このコマンドの初回実行後は、**virsh** シェルが実行されるたびに自動的に実行されます。ハイパーバイザー接続 **URI** はハイパーバイザーへの接続方法を指定します。最もよく使用される **URI** は以下になります。

- **qemu:///system:** root ユーザーとして KVM ハイパーバイザーでゲスト仮想マシンを監視するデーモンにローカルで接続します。
- **xen:///session:** ユーザーとして KVM ハイパーバイザーを使用してユーザーのゲストローカルマシンのセットにローカルに接続します。
- **lxc:///** - ローカルの Linux コンテナに接続します。



- **xen:///** - ローカルの Xen ハイパーバイザーに接続します。

コマンドは以下のように実行します。ターゲットのゲストは、マシン名 (ホスト名) またはハイパーバイザーの URL (**virsh uri** コマンドの出力) で指定します。

```
$ virsh uri
qemu:///session
```

たとえば、ローカルユーザーとしてゲスト仮想マシンの独自のセットに接続するためのセッションを確立するには、以下を実行します。

```
$ virsh connect qemu:///session
```

読み取り専用の接続を開始するには、上記のコマンドに **--readonly** を付加します。URI についての詳細は、[リモート URI](#) を参照してください。URI について不明な場合は、**virsh uri** コマンドを実行すると表示されます。

## 21.5. ゲスト仮想マシンとハイパーバイザーについての情報表示

**virsh list** コマンドは、要求される検索パラメーターに一致するハイパーバイザーに接続されたゲスト仮想マシンを一覧表示します。このコマンドの出力は、3 コラムの表になります。各ゲスト仮想マシンの ID、名前、および **状態** が表示されます。

**virsh list** では、多くの検索パラメーターが使用できます。これらのオプションは、**man virsh** または **virsh list --help** コマンドを実行すると確認できます。



### 注記

このコマンドでは、**root** ユーザーが作成したゲスト仮想マシンのみが表示されます。作成したはずの仮想マシンが表示されない場合は、その仮想マシンを **root** ユーザーで作成しなかった可能性があります。

**virt-manager** インターフェイスを使用して作成するゲストは、デフォルトで **root** による作成となります。

### 例21.1 ローカル接続された仮想マシンを一覧表示する方法

以下の例では、お使いのハイパーバイザーに接続されている仮想マシンすべてが一覧表示されます。このコマンドは、永続的および **一時的** 仮想マシンの両方を表示することに注意してください。

```
# virsh list --all

Id          Name                               State
-----
 8  guest1    running
22  guest2    paused
35  guest3    shut off
38                guest4    shut off
```

### 例21.2 非アクティブなゲストマシンを一覧表示する方法

以下の例では、非アクティブまたは稼働していないゲストを一覧表示します。ここでは永続的仮想マシンのみが表示されることに注意してください。

```
# virsh list --inactive
```

	Id	Name	State
	35	guest3	shut off
	38	guest4	shut off

さらに、以下のコマンドを使用してハイパーバイザーについての基本的な情報を表示できます。

- **# virsh hostname** - 以下のように、ハイパーバイザーのホスト名を表示します。

```
# virsh hostname
dhcp-2-157.eus.myhost.com
```

- **# virsh sysinfo** - ハイパーバイザーのシステム情報の XML 表現を表示します (ある場合)。

```
# virsh sysinfo
<sysinfo type='smbios'>
  <bios>
    <entry name='vendor'>LENOVO</entry>
    <entry name='version'>GJET71WW (2.21 )</entry>
  [...]
```

## 21.6. 仮想マシンの起動、再開および復元

### 21.6.1. ゲスト仮想マシンの起動

**virsh startdomain; [--console] [--paused] [--autodestroy] [--bypass-cache] [--force-boot]** コマンドは、すでに定義されているものの、最後の保存された状態または新規の起動時以降は非アクティブになっている仮想マシンを起動します。デフォルトでは、**virsh managedsave** コマンドで保存されたドメインは、以前の状態に復元されます。それ以外の場合は、新規に起動します。このコマンドは以下の引数を取り、仮想マシンの名前が必要です。

- **--console** - **virsh** を実行中の端末をドメインのコンソールデバイスにアタッチします。これはランレベル 3 になります。
- **--paused** - ドライバーがサポートしている場合、ゲスト仮想マシンを一時停止の状態に起動します。
- **--autodestroy** - ゲスト仮想マシンは **virsh** の接続が解除されると自動的に破棄されます。
- **--bypass-cache** - ゲスト仮想マシンが **managedsave** にある場合に使用されます。
- **--force-boot** - **managedsave** オプションを破棄し、新規の起動を開始します。

#### 例21.3 仮想マシンの起動方法

以下の例では、**guest1** という仮想マシンを起動します。これは既に作成したもので、現在は非アクティブな状態にあるものです。また、このコマンドではゲストのコンソールを **virsh** を実行している端末にアタッチします。

```
# virsh start guest1 --console
Domain guest1 started
Connected to domain guest1
Escape character is ^]
```

### 21.6.2. 起動時に自動的に起動するように仮想マシンを設定

**virsh autostart [--disable] domain** コマンドは、ホストマシンの起動時にゲスト仮想マシンを自動的に起動します。**--disable** 引数をこのコマンドに追加すると、**autostart** が無効になります。この場合のゲストは、ホスト物理マシンの起動時に自動的に起動しません。

#### 例21.4 ホスト物理マシンの起動時に仮想マシンを自動的に起動させる方法

以下の例では、既に作成したもの **guest1** という仮想マシンが、ホストの起動時に自動的に起動するようにします。

```
# virsh autostart guest1
```

### 21.6.3. ゲスト仮想マシンの再起動

**virsh reboot domain [--mode modename]** コマンドを使用してゲスト仮想マシンを再起動します。このアクションは再起動後のみに返されるので、その時点からゲスト仮想マシンが実際に再起動するまで、時間のずれが発生する可能性があります。ゲスト仮想マシンの XML 設定ファイルで **on\_reboot** 要素を変更すると、ゲスト仮想マシンの再起動の動作を制御することができます。デフォルトでは、ハイパーバイザーは適切なシャットダウン方法を自動選択します。代替方法を指定する場合、**--mode** 引数で、**initctl**、**acpi**、**agent**、**signal** を含むコンマ区切りの一覧を指定できます。ドライバーがそれぞれのモードを試行する順序は定義されず、**virsh** で指定された順序には関連がありません。順序付けを厳密に制御するためには、1 度に 1 つのモードを使用してコマンドを繰り返します。

#### 例21.5 ゲスト仮想マシンの再起動方法

以下の例では、**guest1** という名前のゲスト仮想マシンを再起動します。この例では、再起動に **initctl** メソッドを使用しますが、必要に合わせてモードを選択することができます。

```
# virsh reboot guest1 --mode initctl
```

### 21.6.4. ゲスト仮想マシンの復元

**virsh restore <file> [--bypass-cache] [--xml /path/to/file] [--running] [--paused]** コマンドは、**virsh save** コマンドで以前に保存されたゲスト仮想マシンを復元します。**virsh save** コマンドについての情報は、「[ゲスト仮想マシンの設定の保存](#)」を参照してください。復元操作により、保存されたゲスト仮想マシンが再起動しますが、これには時間がかかる場合があります。ゲスト仮想マシンの名前および UUID は保存されますが、ID は仮想マシンの保存時の ID と必ずしも一致する訳ではありません。

**virsh restore** コマンドは、以下の引数を取ります。

- **--bypass-cache**: 復元により、ファイルシステムのキャッシュが回避されますが、このフラグを使用すると、復元操作が遅くなることに注意してください。
- **--xml**: この引数は、XML ファイル名と共に使用する必要があります。この引数は通常省略されますが、復元されるゲスト仮想マシンに代替 XML ファイルを指定するために使用できます。この際、変更はドメイン XML のホスト固有の部分にのみ加えられます。たとえば、これはゲストの保存後に取られるディスクのスナップショットによって生じる、基礎となるストレージのファイル名の違いを説明するために使用できます。
- **--running**: ゲスト仮想マシンを実行状態で起動するために保存イメージに記録された状態を上書きします。
- **--paused** - ゲスト仮想マシンを一時停止の状態で起動するために、保存イメージに記録された状態を上書きします。

### 例21.6 ゲスト仮想マシンの復元方法

以下の例では、ゲスト仮想マシン起動して、その設定ファイル *guest1-config.xml* で稼働させます。

```
# virsh restore guest1-config.xml --running
```

## 21.6.5. ゲスト仮想マシンの再開

**virsh resume domain** コマンドは、一時停止されたドメインの CPU を再開します。この操作は即座に実行されます。ゲスト仮想マシンは、一時停止された時点からの操作を再開します。ただし、この操作では、定義されていないゲスト仮想マシンは再開されないことに注意してください。この操作では **一時的** な仮想マシンは再開されず、永久的な仮想マシンのみが再開されます。

### 例21.7 一時停止しているゲスト仮想マシンの復元方法

以下では、*guest1* 仮想マシンが復元されます。

```
# virsh resume guest1
```

## 21.7. 仮想マシン設定の管理

このセクションでは、仮想マシン設定の管理についての情報を提供します。

### 21.7.1. ゲスト仮想マシンの設定の保存

**virsh save [--bypass-cache] domain file [--xml string] [--running] [--paused] [--verbose]** コマンドは、指定のドメインを停止し、ゲスト仮想マシンの現在の状態を指定されたファイルに保存します。ゲスト仮想マシンが使用しているメモリー量によっては、この操作はかなり時間がかかる場合があります。ゲスト仮想マシンの状態は、**virsh restore** (「[ゲスト仮想マシンの復元](#)」) コマンドで復元できます。

**virsh save** コマンドと **virsh suspend** コマンドの違いは、**virsh suspend** ではドメインの CPU は停止されますが、ドメインの **qemu** プロセスは継続され、そのメモリーイメージはホストシステム内に常駐します。このメモリーイメージは、ホストシステムが再起動すると失われます。

**virsh save** コマンドは、ホストシステムのハードディスク上にドメインの状態を保存し、**qemu** プロセスを終了します。こうすることで、保存された状態からドメインを再開することが可能になります。

**virsh save** のプロセスは、**virsh domjobinfo** コマンドで監視することが可能で、**virsh domjobabort** コマンドでキャンセルできます。

**virsh save** コマンドは、以下の引数を取ります。

- **--bypass-cache**: 復元により、ファイルシステムのキャッシュが回避されますが、このフラグを使用すると、復元操作が遅くなることに注意してください。
- **--xml**: この引数は、XML ファイル名と共に使用する必要があります。この引数は通常省略されますが、復元されるゲスト仮想マシンに代替 XML ファイルを指定するために使用できます。この際、変更はドメイン XML のホスト固有の部分にのみ加えられます。たとえば、これはゲストの保存後に取りられるディスクのスナップショットによって生じる、基礎となるストレージのファイル名の違いを説明するために使用できます。
- **--running**: ゲスト仮想マシンを実行状態で起動するために保存イメージに記録された状態を上書きします。
- **--paused** - ゲスト仮想マシンを一時停止の状態で起動するために、保存イメージに記録された状態を上書きします。
- **--verbose** - 保存の進捗を表示します。

#### 例21.8 ゲスト仮想マシンの実行中の設定を保存する方法

以下の例では、仮想マシン *guest1* の実行中の設定が **guest1-config.xml** ファイルに保存されます。

```
# virsh save guest1 guest1-config.xml --running
```

### 21.7.2. XML ファイルを使用したゲスト仮想マシンの定義

**virsh define filename** コマンドは、XML ファイルでゲスト仮想マシンを定義します。この場合、ゲスト仮想マシン定義は登録されますが、起動しません。ゲスト仮想マシンがすでに実行されている場合は、変更はドメインがシャットダウンされ、再起動の際に有効になります。

#### 例21.9 XML ファイルを使用したゲスト仮想マシンの作成方法

以下の例では、既存の *guest1-config.xml* XML ファイルから仮想マシンが作成されます。このファイルには、仮想マシンの設定が含まれています。

```
# virsh define guest1-config.xml
```

### 21.7.3. ゲスト仮想マシンの復元に使用される XML ファイルの更新

**注記**

このコマンドは、ゲスト仮想マシンが正常に稼働していない場合にのみ使用してください。これは一般的な使用を目的としたものではありません。

**virsh save-image-define filename [--xml /path/to/file] [--running] [--paused]** コマンドは、**virsh restore** コマンドで仮想マシンが復元される際に使用されるゲスト仮想マシンの XML ファイルを更新します。--xml 引数は、ゲスト仮想マシンの XML の代替 XML 要素を含む XML ファイル名である必要があります。たとえば、ゲストの保存後に基礎となるストレージのディスクのスナップショットを作成したファイル名が異なる場合、これを使用できます。保存されたイメージでは、ゲスト仮想マシンが実行中または一時停止の状態に復元するかについて記録されます。--running または --paused の引数を使用することで、復元する状態が決定されます。

**例21.10 ゲスト仮想マシンの実行中の設定を保存する方法**

以下の例では、*guest1-config.xml* 設定ファイルが実行中の状態で保存されます。

```
# virsh save-image-define guest1-config.xml --running
```

**21.7.4. ゲスト仮想マシン XML ファイルの抽出****注記**

このコマンドは、ゲスト仮想マシンが正常に稼働していない場合にのみ使用してください。これは一般的な使用を目的としたものではありません。

**virsh save-image-dumpxml file --security-info** コマンドは、保存状態のファイル (**virsh save** コマンドで使用される) が参照された際に有効であったゲスト仮想マシンの XML ファイルを抽出します。--security-info 引数を使用することにより、セキュリティ上の機密情報がファイルに組み込まれます。

**例21.11 最後の保存から XML 設定をプルする方法**

以下の例では、ゲスト仮想マシンが最後に [保存](#) された際に作成された設定ファイルをダンプします。この例では、作成されるダンプファイル名は *guest1-config.xml* となります。

```
# virsh save-image-dumpxml guest1-config.xml
```

**21.7.5. ゲスト仮想マシン設定の編集****注記**

このコマンドは、ゲスト仮想マシンが正常に稼働していない場合にのみ使用してください。これは一般的な使用を目的としたものではありません。

**virsh save-image-edit <file> [--running] [--paused]** コマンドは、**virsh save** コマンドで作成された XML 設定ファイルを編集します。**virsh save** コマンドについての情報は、「[ゲスト仮想マシンの設定の保存](#)」を参照してください。



ゲスト仮想マシンが保存されると、生成されるイメージファイルは、仮想マシンが **--running** または **--paused** 状態に復元される必要があるかどうかを示します。**save-image-edit** コマンドでこれらの引数を使用しない場合、状態はイメージファイル自体で決定されます。**--running** (実行中の状態を選択する場合) または **--paused** (一時停止の状態を選択する場合) を選択することにより、**virsh restore** が使用する状態を上書きできます。

#### 例21.12 ゲスト仮想マシンの設定を編集し、マシンを実行中の状態に復元する方法

以下の例では、**guest1-config.xml** という名前のゲスト仮想マシン設定ファイルが開かれ、デフォルトのエディターで編集可能となります。変更を保存したら、その新たな設定で仮想マシンが起動します。

```
# virsh save-image-edit guest1-config.xml --running
```

## 21.8. ゲスト仮想マシンのシャットオフ、シャットダウン、再起動および強制終了

### 21.8.1. ゲスト仮想マシンのシャットダウン

**virsh shutdown domain [--mode modename]** コマンドはゲスト仮想マシンをシャットダウンします。ゲスト仮想マシンの設定ファイルで **on\_shutdown** パラメーターを変更すると、ゲスト仮想マシンの再起動の動作を制御することができます。**on\_shutdown** パラメーターの変更は、ドメインがシャットダウンして再起動された後に有効になります。

**virsh shutdown** コマンドは、以下のオプションの引数を取ります。

- **--mode** はシャットダウンモードを選択します。これは、**acpi**、**agent**、**initctl**、**signal**、または **paravirt** のいずれかにする必要があります。

#### 例21.13 ゲスト仮想マシンのシャットダウン方法

以下の例では **acpi** モードで **guest1** 仮想マシンがシャットダウンされます。

```
# virsh shutdown guest1 --mode acpi
Domain guest1 is being shutdown
```

### 21.8.2. ゲスト仮想マシンの一時停止

**virsh suspend domain** コマンドはゲスト仮想マシンを一時停止します。

ゲスト仮想マシンが一時停止状態にある場合、システム RAM を消費しますが、プロセッサのリソースは消費しません。ディスクとネットワークの I/O は、ゲスト仮想マシンの一時停止中には発生しません。この操作は即時に行われ、ゲスト仮想マシンは **virsh resume** コマンドでのみ再起動できます。このコマンドを一時的な仮想マシンで実行すると、仮想マシンは削除されます。

#### 例21.14 ゲスト仮想マシンの一時停止方法

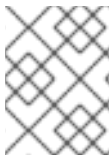
以下の例では **guest1** 仮想マシンが一時停止されます。

```
-
```

```
# virsh suspend guest1
```

### 21.8.3. 仮想マシンのリセット

**virsh reset domain** は、ゲストをシャットダウンすることなく、ゲスト仮想マシンを即時にリセットします。リセットは、マシンのリセットボタンをエミュレートします。ここですべてのゲストハードウェアは RST 行を認識し、内部の状態を再初期化します。ゲスト仮想マシンの OS をシャットダウンしないと、データが失われるリスクがあることに注意してください。



#### 注記

仮想マシンのリセットは、保留中のドメイン設定変更には適用されません。ドメイン設定の変更は、ドメインが完全にシャットダウンして再起動された後に有効になります。

#### 例21.15 ゲスト仮想マシンのリセット方法

以下の例では *guest1* 仮想マシンがリセットされます。

```
# virsh reset guest1
```

### 21.8.4. 実行中ゲスト仮想マシンの停止および再起動

**virsh managedsave domain --bypass-cache --running | --paused | --verbose** コマンドは、実行中の仮想マシンを保存および破棄 (停止) し、後でその状態から再起動できるようにします。**virsh start** コマンドを使用すると、この保存時点から自動的に起動します。**--bypass-cache** 引数を使用すると、ファイルシステムはキャッシュされません。このオプションを使用すると、保存プロセスが遅くなる可能性があり、**--verbose** オプションを使用するとダンププロセスの進捗が表示されることに注意してください。通常このコマンドでは、保存実行時のゲスト仮想マシンの状態により、実行中か一時停止の状態かが決定されます。ただし、これは実行中の状態に置かれていることを示す **--running** オプションを使用するか、または一時停止の状態を示す **--paused** オプションを使用すると、上書きできます。**managedsave** の状態を削除するには、**virsh managedsave-remove** コマンドを使用します。これにより、ゲスト仮想マシンが次の起動時に完全な起動を実行するように強制されます。**managedsave** の全体プロセスは **domjobinfo** コマンドを使用して監視でき、また **domjobabort** コマンドを使用してキャンセルできることに注意してください。

#### 例21.16 実行中のゲストを停止し、その設定を保存する方法

以下の例では、*guest1* 仮想マシンを停止し、実行中の設定が保存されることで、再起動が可能になります。

```
# virsh managedsave guest1 --running
```

## 21.9. 仮想マシンの削除

### 21.9.1. 仮想マシンの定義解除

```
virsh undefine domain [--managed-save] [storage] [--remove-all-storage] [--
```



**wipe-storage** **[--snapshots-metadata]** **[--nvram]** コマンドは、ドメインの定義を解除します。ドメインが非アクティブの場合は、設定は完全に削除されます。ドメインがアクティブ (実行中) の場合は、**一時的**なドメインに変換されます。ゲスト仮想マシンが非アクティブになると、設定は完全に削除されます。

このコマンドは、以下の引数を取ります。

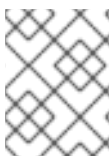
- **--managed-save:** この引数は、管理保護状態のゲスト仮想マシンの定義が解除されると、その関連付けられた管理保護状態のイメージも削除されることを保証します。この引数を使用せずに、管理保護状態のゲスト仮想マシンの定義解除を試みると失敗します。
- **--snapshots-metadata:** この引数は、停止中のゲスト仮想マシンの定義解除を実行する際に、スナップショット (**snapshot-list** で表示) も定義削除されることを保証します。スナップショットのメタデータが指定された停止中のゲスト仮想マシンの定義解除を試みると、いずれの場合も失敗することに注意してください。この引数を使用される際に、ゲスト仮想マシンがアクティブな場合には無視されます。
- **--storage:** この引数を使用すると、定義解除されるドメインと共に削除されるボリュームターゲット名またはストレージボリュームのソースパスのコンマ区切りの一覧が必要になります。このアクションにより、ストレージボリュームは削除される前に定義解除されます。これは停止中のゲスト仮想マシンでのみ実行できることに注意してください。また、これは **libvirt** が管理するストレージボリュームでのみ機能することにも注意してください。
- **--remove-all-storage:** ゲスト仮想マシンの定義解除のほかにも、関連付けられたすべてのストレージボリュームが削除されます。仮想マシンを削除する場合、同じ関連付けられたストレージを使用する仮想マシンが他にない場合にのみこのオプションを選択します。代替方法として、**virsh vol-delete** を使用できます。詳細は、「**ストレージボリュームの削除**」を参照してください。
- **--wipe-storage:** ストレージボリュームの削除に加え、コンテンツが完全消去されます。

### 例21.17 ゲスト仮想マシンを削除し、そのストレージボリュームを削除する方法

以下の例では、**guest1** 仮想マシンの定義が解除され、関連する全ストレージボリュームが削除されます。定義が解除されたゲストは **一時的**な状態になり、シャットダウン後には削除されます。

```
# virsh undefine guest1 --remove-all-storage
```

## 21.9.2. ゲスト仮想マシンの強制終了



### 注記

コマンドは、他の方法ではゲスト仮想マシンをシャットダウンできない場合にのみ使用してください。

**virsh destroy** コマンドでは、正常なシャットダウンの終了プロセスを経ずに、指定されたゲスト仮想マシンを直ちに停止させます。**virsh destroy** コマンドを使用すると、ゲスト仮想マシンのファイルシステムが破損する可能性があります。**virsh destroy** コマンドは、ゲスト仮想マシンが反応しない場合のみに使用してください。**virsh destroy** コマンドで **--graceful** オプションを使用すると、仮想マシンの電源を切る前に、ディスクイメージファイルのキャッシュのフラッシュを試行します。

### 例21.18 ハードシャットダウンによるゲスト仮想マシンの即時のシャットダウン方法

以下の例では、`guest1` 仮想マシンが反応しないなどの理由がある場合に、直ちにシャットダウンします。

```
# virsh destroy guest1
```

このコマンドの実行後には、`virsh undefine` コマンド操作を続けることができます。[例21.17「ゲスト仮想マシンを削除し、そのストレージボリュームを削除する方法」](#)を参照してください。

## 21.10. ゲスト仮想マシンのシリアルコンソールの接続

`virsh console domain [--devname devicename] [--force] [--safe]` コマンドは、ゲスト仮想マシンの仮想シリアルコンソールを接続します。これは、VNC や SPICE プロトコルを提供しないゲスト（つまり、[GUI ツール](#) 用のビデオ表示が提供されない）や、ゲストにネットワーク接続がない（つまり、SSH を使用した対話ができない）場合などに便利です。

オプションの `--devname` パラメーターは、ゲスト仮想マシンに設定された代替コンソールのデバイスエイリアス、シリアルまたはパラレルデバイスを参照します。このパラメーターが省略されると、プライマリコンソールが開かれます。`--safe` オプションを指定すると、ドライバーが安全なコンソール処理をサポートしている場合にのみ接続が試行されます。このオプションは、サーバーがコンソールデバイスへの排他的アクセスがあることを確認するように指定します。`force` オプションの指定も可能で、接続が遮断された場合などに既存セッションの接続解除を要求します。

### 例21.19 コンソールモードでゲスト仮想マシンを起動する方法

以下の例では、既に作成済みの `guest1` 仮想マシンを起動して、これが安全なコンソール処理を使用するシリアルコンソールに接続します。

```
# virsh console guest1 --safe
```

## 21.11. NMI (マスク不可能な割り込み) の挿入

`virsh inject-nmi domain` は NMI (マスク不可能な割り込み) メッセージをゲスト仮想マシンに挿入します。これは、修復不能なハードウェアエラーの発生時など、迅速な応答時間が重要となる場合に使用されます。また、`virsh inject-nmi` は、Windows ゲストでのクラッシュダンプのトリガーにも使用できます。

### 例21.20 NMI をゲスト仮想マシンに挿入する方法

以下の例では `guest1` 仮想マシンに NMI が送信されます。

```
# virsh inject-nmi guest1
```

## 21.12. ゲスト仮想マシン情報の取得

### 21.12.1. デバイスブロック統計の表示

デフォルトでは、**virsh domblkstat** コマンドは、ドメインで最初に定義されているブロックデバイスのブロック統計数字を表示します。他のブロックデバイスの統計数字を表示するには、**virsh domblklist domain** コマンドを使って全ブロックデバイスを一覧表示します。次に、**virsh domblklist** コマンドの出力から **Target** または **Source** の名前をドメイン名の後で指定して、特定のブロックデバイスを指定します。すべてハイパーバイザーで全フィールドが表示されるわけではないことに注意してください。出力を読みやすい形式にするには、**--human** 引数を使用します。

### 例21.21 ゲスト仮想マシンのブロック統計の表示方法

以下の例では、**guest1** 仮想マシンに定義されたデバイスが表示され、そのデバイスのブロック統計数字が一覧表示されます。

```
# virsh domblklist guest1

Target      Source
-----
vda         /VirtualMachines/guest1.img
hdc         -

# virsh domblkstat guest1 vda --human
Device: vda
number of read operations:      174670
number of bytes read:          3219440128
number of write operations:     23897
number of bytes written:       164849664
number of flush operations:     11577
total duration of reads (ns):   1005410244506
total duration of writes (ns):  1085306686457
total duration of flushes (ns): 340645193294
```

### 21.12.2. ネットワークインターフェース統計の取得

**virsh domifstat domain interface-device** コマンドは、所定のゲスト仮想マシンで実行中の指定されたデバイスについてのネットワークインターフェース統計を表示します。

ドメインにどのインターフェイスデバイスが定義されているかを判定するには、**virsh domiflist** コマンドを実行し、**Interface** コラムの出力を使用します。

### 例21.22 ゲスト仮想マシンのネットワーク統計の表示方法

以下の例では、**guest1** 仮想マシンに定義されているネットワークインターフェイスが表示されます。次に、そこで取得したインターフェイス (**macvtap0**) を使用してそのネットワーク統計が表示されます。

```
# virsh domiflist guest1
Interface  Type      Source      Model      MAC
-----
macvtap0   direct    em1         rtl8139     12:34:00:0f:8a:4a

# virsh domifstat guest1 macvtap0
macvtap0 rx_bytes 51120
macvtap0 rx_packets 440
macvtap0 rx_errs 0
macvtap0 rx_drop 0
```

```
macvtap0 tx_bytes 231666
macvtap0 tx_packets 520
macvtap0 tx_errs 0
macvtap0 tx_drop 0
```

### 21.12.3. ゲスト仮想マシンの仮想インターフェースのリンク状態の変更

**virsh domif-setlink domain interface-device state** コマンドは、指定したインターフェイスデバイスリンクの状態を **up** または **down** に設定します。ドメインで定義されているインターフェイスデバイスを判定するには **virsh domiflist** コマンドを実行し、インターフェイスデバイスオプションにその出力の **Interface** または **MAC** コラムを使用します。デフォルトでは、**virsh domif-setlink** は、実行中ドメインのリンク状態を変更します。ドメインの永続的な設定を変更するには、**-config** 引数を使用します。

#### 例21.23 ゲスト仮想マシンのインターフェースを有効にする方法

以下の例では、*rhel7* ドメインのインターフェイスデバイスを特定し、リンクを **down** にしてから最後に **up** に設定しています。

```
# virsh domiflist rhel7
Interface Type      Source      Model      MAC
-----
vnet0      network    default    virtio     52:54:00:01:1d:d0

# virsh domif-setlink rhel7 vnet0 down
Device updated successfully

# virsh domif-setlink rhel7 52:54:00:01:1d:d0 up
Device updated successfully
```

### 21.12.4. ゲスト仮想マシンの仮想インターフェースのリンク状態の一覧表示

**virsh domif-getlink domain interface-device** コマンドは、指定したインターフェイスデバイスリンクの状態を取得します。ドメインで定義されているインターフェイスデバイスを判定するには **virsh domiflist** コマンドを実行し、インターフェイスデバイスオプションにその出力の **Interface** または **MAC** コラムを使用します。デフォルトでは、**virsh domif-getlink** は、実行中ドメインのリンク状態を取得します。ドメインの永続的な設定を取得するには、**--config** オプションを使用します。

#### 例21.24 ゲスト仮想マシンのインターフェースのリンク状態を表示する

以下の例では、*rhel7* ドメインのインターフェイスデバイスを特定し、状態が **up** であることを確認しています。次にこれを **down** に変更して、その変更が成功したかどうかを確認しています。

```
# virsh domiflist rhel7
Interface Type      Source      Model      MAC
-----
vnet0      network    default    virtio     52:54:00:01:1d:d0

# virsh domif-getlink rhel7 52:54:00:01:1d:d0
52:54:00:01:1d:d0 up
```

```
# virsh domif-setlink rhel7 vnet0 down
Device updated successfully

# virsh domif-getlink rhel7 vnet0
vnet0 down
```

### 21.12.5. ネットワークインターフェース帯域幅パラメーターの設定

**virsh domiftune domain interface-device** コマンドは、指定したドメインのインターフェース帯域幅パラメーターを取得または設定します。ドメインで定義されているインターフェースデバイスを判定するには **virsh domiflist** コマンドを実行し、インターフェースデバイスオプションにその出力の **Interface** または **MAC** コラムを使用します。以下の構文を使用してください。

```
# virsh domiftune domain interface [--inbound] [--outbound] [--config] [--live] [--current]
```

**--config**、**--live**、および **--current** のオプションについては、「[スケジュールパラメーターの設定](#)」で説明しています。**--inbound** または **--outbound** のオプションが指定されないと、**virsh domiftune** は指定されたネットワークインターフェイスにクエリーを実行し、帯域幅設定を表示します。**--inbound** または **--outbound** のどちらか、もしくは両方を指定すると、平均、ピーク、バースト値を **virsh domiftune** が帯域幅に設定します。このうち少なくとも、平均値が必要になります。帯域幅設定をクリアにするには、0 (ゼロ) を提供します。平均、アベレージ、バースト値については、「[インターフェースデバイスの割り当て](#)」を参照してください。

#### 例21.25 ゲスト仮想マシンのネットワークインターネットパラメーターの設定方法

以下の例では、仮想マシン *guest1* に *eth0* パラメーターを設定しています。

```
# virsh domiftune guest1 eth0 outbound --live
```

### 21.12.6. メモリー統計の取得

**virsh dommemstat domain [<period in seconds>] [--config] [--live] [--current]** コマンドは、実行中のゲスト仮想マシンのメモリー統計を表示します。オプションの **period** スイッチを使用すると、秒単位の期間が必要になります。このオプションを 0 よりも大きな値に設定すると、バルーンドライバーが追加の統計数字を返せるようになり、続いて **dommemstat** コマンドを実行するとこれが表示されます。**period** オプションを 0 に設定すると、バルーンドライバーによる収集が停止しますが、すでにバルーンドライバーにある統計は消去されません。**period** オプションを設定しないと、**--live**、**--config**、または **--current** オプションは使用できません。**--live** オプションが指定される場合、ゲストの実行中の統計のみが収集されます。**--config** オプションが使用される場合、次の起動後にのみ永続的なゲストの統計が収集されます。**--current** オプションが使用される場合、現在の統計が収集されます。

**--live** および **--config** の両方のオプションを使用できますが、**--current** は排他的な使用になります。フラグが指定されていない場合、ゲストの状態によって統計収集の動作が決まります (実行中かどうかに応じて)。

#### 例21.26 実行中のゲスト仮想マシンのメモリー統計の収集方法

以下の例では、*rhel7* ドメインのメモリー統計数字が表示されます。

```
# virsh dommemstat rhel7
actual 1048576
swap_in 0
swap_out 0
major_fault 2974
minor_fault 1272454
unused 246020
available 1011248
rss 865172
```

### 21.12.7. ブロックデバイスのエラーの表示

**virsh domblkerror domain** コマンドは、**error** 状態になっているブロックデバイスすべてと、それらで検出されたエラーを一覧表示します。このコマンドは、**virsh domstate** コマンドを実行して、I/O エラーのためにゲスト仮想マシンが一時停止していることがレポートされた後に使用するとよいでしょう。

#### 例21.27 仮想マシンのブロックデバイスエラーの表示方法

以下の例では *guest1* 仮想マシンのブロックデバイスエラーが表示されます。

```
# virsh domblkerror guest1
```

### 21.12.8. ブロックデバイス容量の表示

**virsh domblkinfo domain** コマンドは、仮想マシン内の指定されたブロックデバイスの容量、割り当て、物理的ブロックサイズを一覧表示します。**virsh domblklist** コマンドで全ブロックデバイスを一覧表示し、**virsh domblklist** の出力のドメイン名の後にある **Target** もしくは **Source** 名を指定して表示するブロックデバイスを選択します。

#### 例21.28 ブロックデバイス容量の表示方法

以下の例では、*rhel7* 仮想マシン上にあるブロックデバイスを一覧表示し、各デバイスのブロックサイズを表示しています。

```
# virsh domblklist rhel7
Target      Source
-----
vda         /home/vm-images/rhel7-os
vdb         /home/vm-images/rhel7-data

# virsh domblkinfo rhel7 vda
Capacity:    10737418240
Allocation:  8211980288
Physical:    10737418240

# virsh domblkinfo rhel7 /home/vm-images/rhel7-data
```



```
Capacity:      104857600
Allocation:    104857600
Physical:      104857600
```

### 21.12.9. ゲスト仮想マシンに関連付けられたブロックデバイスの表示

**virsh domblklist domain [--inactive] [--details]** コマンドは、指定されたゲスト仮想マシンに関連付けられたすべてのブロックデバイスの表を表示します。

**--inactive** が指定されている場合、結果には次回起動時に使用されるデバイスが表示され、実行中のゲスト仮想マシンによって使用されている現在実行中のデバイスは表示されません。**--details** が指定されている場合、ディスクタイプおよびデバイス値は表に組み込まれます。この表に表示される情報は、**virsh domblkinfo** や **virsh snapshot-create** といった、提供されるブロックデバイスを必要とするコマンドで使うことができます。ディスクの **Target** や **Source** のコンテキストも、**virsh snapshot-create** コマンド用に **xmlfile** コンテキスト情報を生成する際に使用できます。

#### 例21.29 仮想マシンに関連付けられたブロックデバイスの表示方法

以下の例では、**rhel7** 仮想マシンに関連付けられているブロックデバイスの詳細が表示されます。

```
# virsh domblklist rhel7 --details
Type      Device      Target      Source
-----
file      disk        vda         /home/vm-images/rhel7-os
file      disk        vdb         /home/vm-images/rhel7-data
```

### 21.12.10. ゲスト仮想マシンに関連付けられた仮想インターフェースの表示

**virsh domiflist domain** コマンドを実行すると、指定されたドメインに関連付けられているすべての仮想インターフェースの表が表示されます。**virsh domiflist** コマンドには、仮想マシンの名前 (または **domain**) が必要で、**--inactive** 引数をオプションで取ることができます。このオプションを使用すると、実行中ではなく、非アクティブの設定がデフォルト設定で取得されます。**--inactive** を指定すると、次の起動時に使用されるデバイスが表示され、実行中のゲストが使用しているデバイスは表示されません。仮想インターフェースの MAC アドレスを必要とする Virsh コマンド (**detach-interface**、**domif-setlink**、**domif-getlink**、**domifstat**、および **domiftune** など) は、このコマンドで表示される出力を受け付けます。

#### 例21.30 ゲスト仮想マシンに関連付けられた仮想インターフェースの表示方法

以下の例では、**rhel7** 仮想マシンに関連付けられた仮想インターフェースを表示し、**vnet0** デバイスのネットワークインターフェースの統計数字を表示します。

```
# virsh domiflist rhel7
Interface  Type      Source      Model      MAC
-----
vnet0      network   default     virtio      52:54:00:01:1d:d0

# virsh domifstat rhel7 vnet0
vnet0 rx_bytes 55308
vnet0 rx_packets 969
```

```
vnet0 rx_errs 0
vnet0 rx_drop 0
vnet0 tx_bytes 14341
vnet0 tx_packets 148
vnet0 tx_errs 0
vnet0 tx_drop 0
```

## 21.13. スナップショットの使用

### 21.13.1. データのコピーによるバックアップチェーンの短縮化

このセクションでは、**virsh blockcommit domain <path> [<bandwidth>] [<base>] [--shallow] [<top>] [--active] [--delete] [--wait] [--verbose] [--timeout <number>] [--pivot] [--keep-overlay] [--async] [--keep-relative]** コマンドを使用してバックアップチェーンを短縮する方法について説明します。このコマンドには数多くのオプションがあり、これらは **help** メニューまたは **man** ページに一覧表示されています。

**virsh blockcommit** コマンドは、チェーンの一部にあるデータをバックアップファイルにコピーし、コミットされた部分をバイパスするためにチェーンの残りの部分をピボットします。たとえば、以下が現在の状態であるとしてします。

```
base ← snap1 ← snap2 ← active.
```

**virsh blockcommit** を使用して、**snap2** のコンテンツを **snap1** に移行します。これにより、チェーンから **snap2** を削除でき、より迅速にバックアップを作成することができます。

#### 手順21.1 バックアップチェーンの短縮化方法

- 以下のコマンドで **guest1** をゲスト仮想マシンの名前で、**disk1** をディスク名で置き換えて実行します。

```
# virsh blockcommit guest1 disk1 --base snap1 --top snap2 --wait --verbose
```

**snap2** のコンテンツが **snap1** に移行します。以下ようになります。

**base ← snap1 ← active** **snap2** は無効になり、削除することができません。



#### 警告

**virsh blockcommit** は、**--base** 引数に依存するすべてのファイルを破損させます (**--top** 引数に依存していたファイルを除く。これらは **base** を指すようになる)。これを防ぐには、複数のゲストが共有するファイルへの変更をコミットしないでください。**--verbose** オプションを使用すると、進捗が画面に出力されます。



### 21.13.2. イメージのフラット化によるバックアップチェーンの短縮化

**virsh blockpull** は、以下のように応用して使用することができます。

1. イメージにそのバックアップイメージチェーンのデータを設定することにより、イメージをフラット化します。これにより、イメージファイルはバックアップイメージやこれに類するものに依存しなくて済むような自己完結型のファイルになります。
  - 使用前: **base.img** ← **active**
  - 使用后: ゲストによる **base.img** の使用がなくなり、**Active** にすべてのデータが含まれます。
2. バックアップイメージチェーンの一部をフラット化します。これはスナップショットをトップレベルのイメージにフラット化するために使用でき、以下のようになります。
  - 使用前: **base** ← **sn1** ← **sn2** ← **active**
  - 使用后: **base.img** ← **active**。**active** には **sn1** および **sn2** からの全データが含まれ、ゲストは **sn1** も **sn2** も使用しません。
3. ディスクのイメージをホスト上の新規ファイルシステムに移動します。これにより、ゲストの実行中にイメージファイルを移動できます。以下のようになります。
  - 使用前 (元のイメージファイル): **/fs1/base.vm.img**
  - 使用后: **/fs2/active.vm.qcow2** が新規ファイルシステムで、**/fs1/base.vm.img** は使用されません。
4. ポストコピー型ストレージ移行のライブマイグレーションで役立ちます。ディスクイメージは、ライブマイグレーションの完了後に移行元ホストから移行先ホストにコピーされます。

つまり、以下のようになります。使用前の **/source-host/base.vm.img** から使用後の **/destination-host/active.vm.qcow2** になります。**/source-host/base.vm.img** は使用されなくなります。

#### 手順21.2 データのフラット化によるバックアップチェーンの短縮化

1. **virsh blockpull** の実行前にスナップショットを実行すると便利です。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。以下の例で **guest1** をゲスト仮想マシンの名前に、**snap1** をスナップショットの名前に置き換えます。

```
# virsh snapshot-create-as guest1 snap1 --disk-only
```

2. チェーンが **base** ← **snap1** ← **snap2** ← **active** のようになる場合、以下のコマンドを入力し、**guest1** をゲスト仮想マシンの名前に、**path1** をディスクのソースパス (**/home/username/VirtualMachines/\*** など) に置き換えます。

```
# virsh blockpull guest1 path1
```

このコマンドは、データを **snap2** から **active** にプルすることで **base** ← **snap1** ← **active** となり、**snap1** を **active** のバックアップファイルにします。

3. **virsh blockpull** が完了すると、チェーン内の追加イメージを作成したスナップショットの **libvirt** 追跡は意味をなさなくなります。以下のコマンドを使って、古くなったスナップショッ

トの追跡を削除します。この際、`guest1` をゲスト仮想マシンの名前に、`snap1` をスナップショットの名前に置き換えます。

```
# virsh snapshot-delete guest1 snap1 --metadata
```

`virsh blockpull` は他にも、以下の使い方ができます。

#### 例21.31 単一イメージをフラット化し、バックリングイメージチェーンのデータを設定する方法

以下の例では、`guest1` ゲスト上の `vda` 仮想ディスクをフラット化し、バックリングイメージチェーンからデータのあるイメージを設定し、設定アクションの完了を待機します。

```
# virsh blockpull guest1 vda --wait
```

#### 例21.32 バックリングイメージチェーンの一部をフラット化する方法

以下の例では、`/path/to/base.img` ディスクイメージを基に、`guest1` ゲスト上の `vda` 仮想ディスクをフラット化します。

```
# virsh blockpull guest1 vda /path/to/base.img --base --wait
```

#### 例21.33 ディスクイメージをホストの新規ファイルシステムに移動する方法

ディスクイメージをホストの新規ファイルシステムに移動するには、以下の 2 つのコマンドを実行します。各コマンドでは、`guest1` をゲスト仮想マシンの名前に、`disk1` を仮想ディスクの名前に置き換えます。また、XML ファイル名、ロケーションへのパスおよびスナップショットの名前も変更します。

```
# virsh snapshot-create guest1 --xmlfile /path/to/snap1.xml --disk-only
```

```
# virsh blockpull guest1 disk1 --wait
```

#### 例21.34 ポストコピー型ストレージ移行のライブマイグレーションを使用する方法:

ポストコピー型ストレージ移行のライブマイグレーションを使用するには、以下のコマンドを入力します。

移行先で以下のコマンドを入力し、バックリングファイルをホスト上のバックリングファイルの名前およびロケーションで置き換えます。

```
# qemu-img create -f qcow2 -o backing_file=/source-host/vm.img  
/destination-host/vm.qcow2
```

ソースで以下のコマンドを入力し、`guest1` をゲスト仮想マシンの名前で置き換えます。

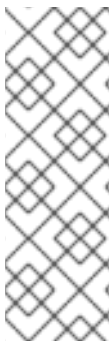
```
# virsh migrate guest1
```

移行先で以下のコマンドを入力し、*guest1* をゲスト仮想マシンの名前、*disk1* を仮想ディスク名で置き換えて実行します。

```
# virsh blockpull guest1 disk1 --wait
```

### 21.13.3. ゲスト仮想マシンのブロックデバイスのサイズ変更

**virsh blockresize** コマンドを使用して、ゲスト仮想マシンの実行中にゲスト仮想マシンのブロックデバイスのサイズを変更することができます。この際、固有のターゲット名 (**<target dev="name"/>**) またはソースファイル (**<source file="name"/>**) にも対応するブロックデバイスの絶対パスを使用します。これは、ゲスト仮想マシンに割り当てられているディスクデバイスのいずれかに適用できます (コマンド **virsh domblklist** を使用して、所定のゲスト仮想マシンに関連付けられたすべてのブロックデバイスの簡単な情報を表示する表を出力できます)。



#### 注記

ライブのイメージサイズの変更により、イメージのサイズは常に変更されますが、この変更はゲストによって即時に反映されない場合があります。最新のゲストカーネルでは、**virtio-blk** デバイスのサイズは自動的に更新されます (旧式のカーネルではゲストの再起動が必要です)。SCSI デバイスでは、コマンド **echo > /sys/class/scsi\_device/0:0:0:0/device/rescan** を使って、ゲスト内の再スキャンを手動でトリガーすることが求められます。さらに IDE の場合、ゲストが新たなサイズを認識する前にゲストを再起動しておく必要があります。

#### 例21.35 ゲスト仮想マシンのブロックデバイスのサイズを変更する方法

以下の例では、仮想マシン *guest1* のブロックデバイスが 90 バイトに変更されます。

```
# virsh blockresize guest1 90 B
```

### 21.14. グラフィカル表示に接続するための URI を表示

**virsh domdisplay** コマンドを実行すると、VNC、SPICE、または RDP 経由でゲスト仮想マシンのグラフィカル表示に接続するために使用される URI が出力されます。**--type** オプションを使用すると、ディスプレイタイプが指定できます。引数 **--include-password** を使用すると、SPICE チャネルのパスワードが URI に組み込まれます。

#### 例21.36 SPICE の URI を表示する方法

以下の例では SPICE 用の URI が表示されます。これは 仮想マシン *guest1* が使用しているグラフィカルディスプレイです。

```
# virsh domdisplay --type spice guest1
spice://192.0.2.1:5900
```

接続 URI に関する詳細は、[the libvirt upstream pages](#) を参照してください。

## 21.15. VNC ディスプレイの IP アドレスとポート番号の表示

**virsh vncdisplay** コマンドは、指定されたゲスト仮想マシンについての VNC ディスプレイの IP アドレスとポート番号を返します。ゲストの情報が取得できない場合、終了コード **1** が表示されます。

このコマンドが機能するには、ゲストの XML ファイルの **devices** 要素でグラフィックタイプが VNC に指定されている必要があります。詳細は、「[グラフィカルフレームバッファ](#)」を参照してください。

### 例21.37 VNC の IP アドレスとポート番号を表示する方法

以下の例では *guest1* 仮想マシンの VNC ディスプレイのポート番号が表示されます。

```
# virsh vncdisplay guest1
127.0.0.1:0
```

## 21.16. 使用されていないブロックの破棄

**virsh domfstrim domain [--minimum bytes] [--mountpoint mountPoint]** コマンドは、指定された実行中のゲスト仮想マシン内でマウントされている全ファイルシステムで **fstrim** ユーティリティを起動します。これにより、ファイルシステムで使用されていないブロックが破棄されます。引数 **--minimum** を使用する場合は、バイト単位の容量を指定する必要があります。この容量は連続する空き容量範囲の長さとしてゲストカーネルに送信されます。この容量よりも小さい値は無視される可能性があります。この値を増やすと、不適切にフラグメント化された空き容量を持つファイルシステムとの競合が生じます。この場合、すべてのブロックが破棄される訳ではないことに注意してください。デフォルトの最低値はゼロであり、これはすべての空きブロックが破棄されることを意味します。この値をゼロよりも大きな値に増やす場合、すべてのブロックが破棄される訳ではないものの、不適切にフラグメント化された空き容量を持つファイルシステムに対する **fstrim** 操作がより速く完了します。ユーザーが1つの特定のマウントポイントのみをトリム処理する場合は、**--mountpoint** 引数を使用し、マウントポイントを指定する必要があります。

### 例21.38 使用されていないブロックを破棄する方法

以下の例では、仮想マシン *guest1* で実行されているファイルシステムをトリムします。

```
# virsh domfstrim guest1 --minimum 0
```

## 21.17. ゲスト仮想マシンの検索コマンド

### 21.17.1. ホスト物理マシン名の表示

**virsh domhostname domain** コマンドは、ハイパーバイザーが公開できる場合、指定されたゲスト仮想マシンの物理ホスト名を表示します。

### 例21.39 ホスト物理マシン名の表示方法

以下の例では、ハイパーバイザーによる公開が可能な場合、*guest1* 仮想マシンの物理マシンのホスト名が表示されます。

```
# virsh domhostname guest1
```

### 21.17.2. 仮想マシンについての一般的な情報の表示

**virsh dominfo domain** コマンドは、指定されたゲスト仮想マシンについての基本的な情報を表示します。このコマンドは、オプションの **[--domain] guestname** と共に使用することもできます。

#### 例21.40 ゲスト仮想マシンについての一般的な情報の表示方法

以下の例では、仮想マシン *guest1* についての全般的情報が表示されます。

```
# virsh dominfo guest1
Id: 8
Name: guest1
UUID: 90e0d63e-d5c1-4735-91f6-20a32ca22c40
OS Type: hvm
State: running
CPU(s): 1
CPU time: 271.9s
Max memory: 1048576 KiB
Used memory: 1048576 KiB
Persistent: yes
Autostart: disable
Managed save: no
Security model: selinux
Security DOI: 0
Security label: system_u:system_r:svirt_t:s0:c422,c469 (enforcing)
```

### 21.17.3. 仮想マシンの ID 番号の表示

**virsh list** の出力には ID が含まれますが、**virsh domid domain>|<ID** は、実行中の場合にゲスト仮想マシンの ID を表示します。ID は仮想マシンを実行するたびに変更されます。ゲスト仮想マシンがシャットオフされている場合は、マシン名は一連のダッシュ ('-----') として表示されます。このコマンドは **[--domain guestname]** オプションと使用することもできます。

#### 例21.41 仮想マシンの ID 番号の表示

このコマンドを実行し、使用可能な出力を受信するには、仮想マシンが実行中である必要があります。以下の例では、仮想マシン *guest1* の ID 番号が出力されます。

```
# virsh domid guest1
8
```

### 21.17.4. ゲスト仮想マシンでの実行中のジョブの中止

**virsh domjobabort domain** コマンドは、指定されたゲスト仮想マシンで現在実行されているジョブを中止します。このコマンドは、**[--domain guestname]** オプションとすることもできます。

#### 例21.42 ゲスト仮想マシンで実行中のジョブを中止する方法

以下の例では、仮想マシン `guest1` で中止するジョブが実行中であるとします。実際にコマンドを実行する際には、`guest1` を仮想マシン名に置き換えます。

```
# virsh domjobabort guest1
```

### 21.17.5. ゲスト仮想マシンで実行中のジョブについての情報の表示

`virsh domjobinfo domain` コマンドは、移行の統計数値を含む、指定されたゲスト仮想マシンでの実行中のジョブについての情報を表示します。このコマンドは `[--domain guestname]` オプションまたは `--completed` オプションと使用すると、最近完了したジョブの統計についての情報が返されます。

#### 例21.43 統計的フィードバックの表示方法

以下の例では `guest1` 仮想マシンについての統計情報が一覧表示されます。

```
# virsh domjobinfo guest1
Job type:          Unbounded
Time elapsed:      1603          ms
Data processed:    47.004 MiB
Data remaining:    658.633 MiB
Data total:        1.125 GiB
Memory processed:  47.004 MiB
Memory remaining: 658.633 MiB
Memory total:      1.125 GiB
Constant pages:    114382
Normal pages:      12005
Normal data:        46.895 MiB
Expected downtime: 0            ms
Compression cache: 64.000 MiB
Compressed data:   0.000 B
Compressed pages:  0
Compression cache misses: 12005
Compression overflows: 0
```

### 21.17.6. ゲスト仮想マシン名の表示

`virsh domname domainID` コマンドは、ドメインの ID または UUID を一緒に渡すことで、ゲスト仮想マシンの名前が返されます。`virsh list --all` コマンドもゲスト仮想マシンの名前を表示しますが、このコマンドはゲスト名のみを一覧表示します。

#### 例21.44 ゲスト仮想マシンの名前の表示方法

以下の例では、ドメイン ID が 8 である仮想マシンの名前が表示されます。

```
# virsh domname 8
guest1
```

### 21.17.7. 仮想マシンの状態の表示

**virsh domstate domain** コマンドは、指定されたゲスト仮想マシンの状態を表示します。**--reason** 引数を使用すると、表示された状態の理由も表示されます。このコマンドは、**[--domain guestname]** オプション、さらには **--reason** オプション と使用することもでき、その場合は状態の理由が表示されます。コマンドでエラーが返される場合は、**virsh domblkerror** を実行してください。詳細は、「[ブロックデバイスのエラーの表示](#)」を参照してください。

#### 例21.45 ゲスト仮想マシンの現在の状態の表示方法

以下の例では *guest1* 仮想マシンの現在の状態が表示されます。

```
# virsh domstate guest1
running
```

### 21.17.8. 仮想マシンへの接続状態の表示

**virsh domcontrol domain** は、特定のゲスト仮想マシンを制御するハイパーバイザーへのインターフェースの状態を表示します。「OK」ではない、または「Error」の状態の場合、制御インターフェースが表示される状態に入ってから経過した秒数も出力されます。

#### 例21.46 ゲスト仮想マシンのインターフェースの状態の表示方法

以下の例では *guest1* 仮想マシンのインターフェースの現在の状態が表示されます。

```
# virsh domcontrol guest1
ok
```

### 21.18. QEMU 引数のドメイン XML への変換

**virsh domxml-from-native** コマンドは、QEMU 引数の既存のセットをドメイン XML 設定ファイルに変換する方法を提供します。この設定ファイルはその後 **libvirt** で使用できます。ただし、このコマンドは、既存の QEMU ゲストがコマンドラインから以前に起動されている場合に、これらのゲストを **libvirt** で管理できるように変換する目的でのみ使用される点になっている点に注意してください。このため、ここに説明されている方法は、新しいゲストをゼロから作成するためには使用しないでください。新しいゲストは **virsh**、[virt-install](#)、または [virt-manager](#) を使って作成する必要があります。追加情報については、[libvirt のアップストリームの web サイト](#) を参照してください。

#### 手順21.3 QEMU ゲストを libvirt に変換する方法

1. QEMU ゲストを引数ファイル (ファイルタイプは **\*.args**) で起動します。この例では、*demo.args* になります。

```
$ cat demo.args
LC_ALL=C
PATH=/bin
HOME=/home/test
USER=test
LOGNAME=test /usr/bin/qemu -S -M pc -m 214 -smp 1 -nographic -
monitor pty -no-acpi -boot c -hda /dev/HostVG/QEMUGuest1 -net none -
serial none -parallel none -usb
```

2. このファイルをドメイン XML ファイルに変換して、ゲストを `libvirt` で管理できるようにするには、以下のコマンドをターミナルに入力します。`qemu-guest1` をゲスト仮想マシンの名前、`demo.args` を QEMU args ファイルのファイル名で置き換えます。

```
# virsh domxml-from-native qemu-guest1 demo.args
```

このコマンドは、`demo.args` ファイルを以下のドメイン XML ファイルに変換します。

```
<domain type='qemu'>
  <uuid>00000000-0000-0000-0000-000000000000</uuid>
  <memory>219136</memory>
  <currentMemory>219136</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='hd' />
  </os>
  <clock offset='utc' />
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/bin/qemu</emulator>
    <disk type='block' device='disk'>
      <source dev='/dev/HostVG/QEMUGuest1' />
      <target dev='hda' bus='ide' />
    </disk>
  </devices>
</domain>
```

図21.1 ゲスト仮想マシンの新規設定ファイル

## 21.19. VIRSH DUMP を使ったゲスト仮想マシンのコアのダンプファイルの作成

ゲスト仮想マシンのトラブルシューティングの方法の1つ (`kdump` と `pvpanic` とは別に) は、`virsh dump domain corefilepath [--bypass-cache] [--live | --crash | --reset] [--verbose] [--memory-only] [--format=format]` コマンドを使用することです。このコマンドは、ゲスト仮想マシンのコアを含むダンプファイルを作成するので、これを分析することができます。

特に、`virsh dump` コマンドは、ゲスト仮想マシンのコアのダンプファイルを、コアファイルパスで指定したファイルに作成します。ハイパーバイザーによってはこのアクションを制限し、ユーザーがファイルと `corefilepath` パラメーターで指定したパスのパーミッションを手動で確保する必要がある場合があります。このコマンドは、[SR-IOV](#) デバイスおよびその他のパススルーデバイスでサポートされています。以下の引数もサポートされています。

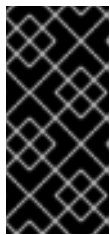
- **--bypass-cache** - 保存されるファイルは、ホストのファイルシステムのキャッシュを迂回します。ファイルのコンテンツには影響がありません。このオプションを選択すると、ダンプ操作が遅くなる可能性があることに注意してください。
- **--live** - ゲスト仮想マシンが実行を継続する際にファイルを保存し、ゲスト仮想マシンが一時停止/停止することはありません。



- **--crash** - ダンプファイルが保存される間に、ゲスト仮想マシンを一時停止の状態のままにするのではなく、クラッシュした状態にします。ゲスト仮想マシンは "Shut off" と表示され、理由が "Crashed" となります。
- **--reset** - ダンプファイルが正常に保存されると、ゲスト仮想マシンがリセットされます。
- **--verbose** - ダンププロセスの進捗が表示されます。
- **--memory-only** - このオプションを使用してダンプを実行すると、ダンプファイルのコンテンツにゲスト仮想マシンのメモリーと CPU 共通レジスターファイルのみが含まれるダンプファイルが作成されます。このオプションは、完全なダンプが失敗する場合に使用してください。ゲスト仮想マシンのライブマイグレーションを実行できない場合に、(パススルー PCI デバイスが原因で) ダンプが失敗する可能性があります。

**--format=format** オプションを使用すると、メモリーのためのダンプを保存できます。以下のフォーマットが使用できます。

- **elf** - デフォルトの未圧縮フォーマット
- **kdump-zlib** - zlib 圧縮による kdump-圧縮フォーマット
- **kdump-lzo** - LZO 圧縮による kdump-圧縮フォーマット
- **kdump-snappy** - Snappy 圧縮による kdump-圧縮フォーマット



### 重要

**crash** ユーティリティは、**virsh dump** コマンドのデフォルトのコアダンプファイル形式をサポートします。**crash** を使用して **virsh dump** で作成されるコアダンプファイルを分析する場合、**--memory-only** オプションを使用する必要があります。

プロセス全体は **virsh domjobinfo** コマンドで監視でき、**virsh domjobabort** コマンドでキャンセルできます。

#### 例21.47 virsh でダンプファイルを作成する方法

以下の例では、仮想マシン **guest1** のコアのダンプファイルが作成されて **core/file/path.file** ファイルに保存され、ゲストをリセットします。このコマンドは、ゲスト仮想マシンの動作がおかしくなった場合によく使われます。

```
# virsh dump guest1 core/file/path.file --reset
```

## 21.20. 仮想マシンの XML ダンプの作成 (設定ファイル)

**virsh dumpxml** コマンドは、ゲスト仮想マシンの XML 設定ファイルを返します。このファイルは必要に応じて、後で使用、保存または変更できます。

XML ファイル (**guest.xml**) はゲスト仮想マシンを再作成するために使用できます (「[ゲスト仮想マシンの XML 設定の編集](#)」を参照)。この XML 設定ファイルを編集することで、追加のデバイスを設定したり、追加のゲスト仮想マシンを実装したりすることができます。

**例21.48 ゲスト仮想マシンの XML ファイルを取得する方法**

以下の例では、仮想マシン `guest1` の XML 設定が取得され、`guest1.xml` ファイルにパイプされます。

```
# virsh dumpxml guest1 | guest1.xml
<domain type='kvm'>
  <name>guest1-rhel6-64</name>
  <uuid>b8d7388a-bbf2-db3a-e962-b97ca6e514bd</uuid>
  <memory>2097152</memory>
  <currentMemory>2097152</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
    <boot dev='hd' />
  </os>
  [...]

```

**21.21. 設定ファイルでのゲスト仮想マシンの作成**

ゲスト仮想マシンは XML 設定ファイルから作成することができます。以前に作成されたゲスト仮想マシンから既存の XML をコピーするか、または **virsh dumpxml** コマンドを使用することができます。

**例21.49 XML ファイルを使用したゲスト仮想マシンの作成方法**

以下の例では、既存の `guest1.xml` 設定ファイルから新規の仮想マシンを作成します。このファイルは前もって用意しておく必要があります。このファイルは **virsh dumpxml** コマンドを使用して取得できます。詳細は、[例21.48 「ゲスト仮想マシンの XML ファイルを取得する方法」](#)を参照してください。

```
# virsh create guest1.xml
```

**21.22. ゲスト仮想マシンの XML 設定の編集**

**virsh edit** コマンドを使用すると、指定したゲストのドメイン XML 設定ファイルを編集することができます。このコマンドを実行すると、**\$EDITOR** シェルパラメーターで指定されているテキストエディター (デフォルトでは **vi**) で XML ファイルが開きます。

**例21.50 ゲスト仮想マシンの XML 設定を編集する方法**

以下の例では、仮想マシン `guest1` に関連付けられている XML 設定ファイルがデフォルトのテキストエディターで開きます。

```
# virsh edit guest1
```

**21.23. 多機能 PCI デバイスの KVM ゲスト仮想マシンへの追加**

多機能 PCI デバイスを KVM ゲスト仮想マシンに追加するには、以下の手順に従います。

1. **virsh edit *guestname*** コマンドを実行して、ゲスト仮想マシンの XML 設定ファイルを編集します。
2. **<address>** 要素には、**multifunction='on'** 属性を追加します。これにより、特定の多機能 PCI デバイスの他の機能が有効になります。

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none' />
<source file='/var/lib/libvirt/images/rhel62-1.img' />
<target dev='vda' bus='virtio' />
<address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x0' multifunction='on' />
</disk>
```

2 種類の機能を備えた PCI デバイスの場合、XML 設定ファイルを修正して、2 番目のデバイスに 1 番目のデバイスと同じスロット番号と、**function='0x0'** などの異なる機能番号を持たせます。

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none' />
<source file='/var/lib/libvirt/images/rhel62-1.img' />
<target dev='vda' bus='virtio' />
<address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x0' multifunction='on' />
</disk>
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none' />
<source file='/var/lib/libvirt/images/rhel62-2.img' />
<target dev='vdb' bus='virtio' />
<address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x1' />
</disk>
```

3. **lspci** コマンドを実行します。KVM ゲスト仮想マシンの出力は **virtio** ブロックデバイスを表示します。

```
$ lspci

00:05.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:05.1 SCSI storage controller: Red Hat, Inc Virtio block device
```



## 注記

SeaBIOS アプリケーションは BIOS インターフェースとの互換性のために **real** モードで実行されます。これにより、利用可能なメモリー容量が制限されます。結果として、SeaBIOS は制限された数のディスクしか処理できなくなります。現在サポートされているディスク数は以下のとおりです。

- virtio-scsi – 64
- virtio-blk – 4
- ahci/sata – 24 (6 ポートが接続された 4 コントローラー)
- usb-storage – 4

回避策として、仮想マシンに多量のディスクを割り当てる場合、SeaBIOS が pci バスをスキャンする際にまず認識できるよう、システムディスクの pci スロット番号が小さく設定されていることを確認してください。さらに、ディスクごとのメモリーオーバーヘッドも小さくなるので、virtio-blk ではなく virtio-scsi デバイスを使用することもお勧めします。

## 21.24. 指定されたゲスト仮想マシンの CPU 統計の表示

**virsh cpu-stats domain --total start count** コマンドは、指定されたゲスト仮想マシンについての CPU 統計情報を提供します。デフォルトで、このコマンドはすべての CPU についての統計を合計値と共に表示します。**--total** オプションを使用すると、合計の統計情報のみを表示します。**--count** オプションを使用すると、**count** CPU の統計情報のみが表示されます。

### 例21.51 ゲスト仮想マシンの CPU 統計を生成する方法

以下の例では、仮想マシン *guest1* の CPU 統計情報が生成されます。

```
# virsh cpu-stats guest1

CPU0:
  cpu_time          242.054322158 seconds
  vcpu_time         110.969228362 seconds
CPU1:
  cpu_time          170.450478364 seconds
  vcpu_time         106.889510980 seconds
CPU2:
  cpu_time          332.899774780 seconds
  vcpu_time         192.059921774 seconds
CPU3:
  cpu_time          163.451025019 seconds
  vcpu_time          88.008556137 seconds
Total:
  cpu_time          908.855600321 seconds
  user_time         22.110000000 seconds
  system_time       35.830000000 seconds
```

## 21.25. ゲストコンソールのスクリーンショット

**virsh screenshot guestname [imagefilepath]** コマンドは、現在のゲスト仮想マシンコンソールのスクリーンショットを撮り、これをファイルに保存します。ファイルパスが提供されない場合は、現行ディレクトリーにスクリーンショットが保存されます。ハイパーバイザーが1つのゲスト仮想マシンの複数の表示に対応している場合、**--screen screenID** オプションを使用すると、キャプチャーする画面を指定できます。

### 例21.52 ゲストマシンのコンソールのスクリーンショット撮影

以下の例では、**guest1** マシンのコンソールのスクリーンショットを撮り、**/home/username/pics/guest1-screen.png** として保存します。

```
# virsh screenshot guest1 /home/username/pics/guest1-screen.ppm
Screenshot saved to /home/username/pics/guest1-screen.ppm, with type of
image/x-portable-pixmap
```

## 21.26. キー入力の組み合わせの指定されたゲスト仮想マシンへの送信

**virsh send-key domain --codeset --holdtime keycode** コマンドを使用すると、シーケンスを **keycode** として特定のゲスト仮想マシンに送信できます。各 **keycode** は、以下の対応する **codeset** からの数値またはシンボリック名とすることができます。

**--holdtime** を指定すると、それぞれのキー入力(指定された時間(ミリ秒単位) 保持されます。**--codeset** を使用すると、コードセットを指定できます。デフォルトは **Linux** ですが、以下のオプションも使用できます。

- **linux** - このオプションを選択すると、シンボリック名が対応する **Linux** キーの定数マクロ名に一致し、数値は **Linux** の汎用入力イベントサブシステムによって提供されるものになります。
- **xt** - これは、XT キーボードコントローラーによって定義される値を送信します。シンボリック名は一切指定されません。
- **atset1** - 数値は AT キーボードコントローラー、**set1** (XT と互換性のあるセット) で定義されるものです。**atset1** からの拡張キーコードは XT コードセットの拡張キーコードと異なる場合があります。シンボリック名は一切指定されません。
- **atset2** - 数値は AT キーボードコントローラー、**set 2** によって定義されるものです。シンボリック名は一切指定されません。
- **atset3** - 数値は AT キーボードコントローラー、**set 3** (PS/2 と互換性あり) で定義されるものです。シンボリック名は一切指定されません。
- **os\_x** - 数値は OS-X キーボード入力サブシステムで定義されるものです。シンボリック名は対応する OS-X キー定数マクロ名に一致します。
- **xt\_kbd** - 数値は **Linux** KBD デバイスで定義されるものです。それらは元の XT コードセットの種類ですが、拡張キーコードの異なるエンコードを持つ場合があります。シンボリック名は一切指定されません。
- **win32** - 数値は Win32 キーボード入力サブシステムで定義されるものです。シンボリック名は対応する Win32 キー定数マクロ名に一致します。

- **usb** - 数値は、キーボード入力の USB HID 仕様によって定義されるものです。シンボリック名は一切指定されません。
- **rfb** - 数値は、**raw** キーコードを送信するために RFB 拡張によって定義されるものです。これらは XT コードセットの種類ですが、拡張キーコードには、高ビットの第1バイトではなく、低ビットの第2バイトセットがあります。シンボリック名は一切指定されません。

### 例21.53 キー入力の組み合わせをゲスト仮想マシンに送信する方法

以下の例では、**Left Ctrl**、**Left Alt**、および **Delete** を Linux エンコードで *guest1* 仮想マシンに送信し、1 秒間保持します。これらのキーは同時に送信され、ゲストではランダムな順序で受信されます。

```
# virsh send-key guest1 --codeset Linux --holdtime 1000 KEY_LEFTCTRL
KEY_LEFTALT KEY_DELETE
```



### 注記

複数の **keycode** を指定した場合は、それらはゲストに同時に送信され、ランダムな順序で受信されます。**keycode** を特定の順序で受信させたい場合は、**virsh send-key** コマンドを受信させたい順序で複数回実行する必要があります。

## 21.27. ホストマシンの管理

本セクションでは、ホストシステム (コマンドでは *node* と呼ばれます) の管理に必要なコマンドについて説明します。

### 21.27.1. ホスト情報の表示

**virsh nodeinfo** コマンドは、モデル番号、CPU の数、CPU のタイプ、および物理メモリーのサイズを含むホストについての基本情報を表示します。この出力は、**virNodeInfo** の構造に対応しています。特に「**CPU socket(s)**」フィールドは NUMA セルごとの CPU ソケット数を示しています。

#### 例21.54 ホストマシン情報の表示

以下の例では、ホストについての情報が表示されます。

```
$ virsh nodeinfo
CPU model:           x86_64
CPU(s):              4
CPU frequency:       1199 MHz
CPU socket(s):       1
Core(s) per socket:  2
Thread(s) per core:  2
NUMA cell(s):        1
Memory size:         3715908 KiB
```

### 21.27.2. NUMA パラメーターの設定

**virsh numatune** コマンドは、指定されたゲスト仮想マシンの NUMA パラメーターの設定または取得

のいずれかを実行できます。ゲスト仮想マシンの設定 XML ファイル内で、これらのパラメーターは **<numatune>** 要素内にネスト化されます。フラグを使用しないと、現在の設定のみが表示されます。**numatune domain** コマンドには指定されたゲスト仮想マシン名が必要で、以下の引数を取ります。

- **--mode** - このモードは **strict**、**interleave**、または **preferred** のいずれかに設定できます。ゲスト仮想マシンが **strict** モードで起動されたのではない限り、ゲスト仮想マシンが稼働中の場合にそれらのモードを変更することはできません。
- **--nodeset** - ゲスト仮想マシンを実行するためにホスト物理マシンによって使用される NUMA ノードの一覧が含まれます。この一覧にはノードが含まれますが、それぞれはコンマで区切られ、ノード範囲にはダッシュ - が、ノードの除外にはキャレット ^ が使用されます。
- 各インスタンスごとに使用できるのは以下の 3 つのフラグの 1 つのみです。
  - **--config** は、永続的なゲスト仮想マシンの次の起動で実施されます。
  - **--live** は、実行中のゲスト仮想マシンのスケジューラー情報を設定します。
  - **--current** は、ゲスト仮想マシンの現在の状態に影響を与えます。

#### 例21.55 ゲスト仮想マシンの NUMA パラメーターを設定する方法

以下の例では、実行中の *guest1* 仮想マシンのノード 0、2、および 3 で NUMA モードを **strict** に設定します。

```
# virsh numatune guest1 --mode strict --nodeset 0,2-3 --live
```

このコマンドを実行すると *guest1* の実行中の設定が XML ファイルで以下の設定に変更されます。

```
<numatune>
    <memory mode='strict' nodeset='0,2-3' />
</numatune>
```

### 21.27.3. NUMA セルの空きメモリー容量の表示

**virsh freecell** コマンドは、指定された NUMA セル内のマシンで利用可能なメモリー容量を表示します。このコマンドは、指定されるオプションに応じて、マシンで利用可能なメモリーについての 3 つの異なる表示のいずれかを提供します。

#### 例21.56 仮想マシンおよび NUMA セルのメモリープロパティを表示する方法

以下のコマンドでは、全セル内で利用可能なメモリーの合計容量が表示されます。

```
# virsh freecell
Total: 684096 KiB
```

個別セル内で利用可能なメモリー容量を表示するには、**--all** オプションを使用します。

```
# virsh freecell --all
0:      804676 KiB
-----
```

```
Total:      804676 KiB
```

指定したセル内の個別メモリー容量を表示するには、**--cellno** オプションを使用します。

```
# virsh freecell --cellno 0
0: 772496 KiB
```

#### 21.27.4. CPU 一覧の表示

**virsh nodecpumap** コマンドは、ホストマシンで利用可能な CPU 数を表示します。また、現在オンラインの CPU 数も表示します。

##### 例21.57 ホストで利用可能な CPU 数の表示

以下の例では、ホストで利用可能な CPU 数が表示されます。

```
# virsh nodecpumap
CPUs present: 4
CPUs online: 1
CPU map: y
```

#### 21.27.5. CPU 統計の表示

**virsh nodecpustats [cpu\_number] [--percent]** コマンドは、ホストの CPU 負荷のステータスに関する統計情報を表示します。CPU を指定すると、その CPU についての情報のみが表示されます。**percent** オプションを指定すると、1 秒間隔で記録された各タイプの CPU 統計のパーセンテージが表示されます。

##### 例21.58 CPU の使用状況についての統計情報を表示する方法

以下の例では、ホスト CPU の負荷に関する全般的統計情報が返されます。

```
# virsh nodecpustats
user:      1056442260000000
system:    401675280000000
idle:      7549613380000000
iowait:    945935700000000
```

以下の例では、CPU 番号 2 の統計情報がパーセンテージで表示されます。

```
# virsh nodecpustats 2 --percent
usage:      2.0%
user:       1.0%
system:     1.0%
idle:      98.0%
iowait:     0.0%
```

#### 21.27.6. デバイスの管理



### 21.27.6.1. virsh を使用したデバイスの割り当てと更新

ストレージデバイスの割り当てについてさらに詳しくは、「[ファイルベースのストレージのゲストへの追加](#)」を参照してください。

#### 手順21.4 ゲスト仮想マシンが使用する USB デバイスのホットプラグ

USB デバイスの割り当ては、ゲスト仮想マシンの実行中にホットプラグインで実行することも、ゲストの停止中に実行することもできます。ゲストで使用するデバイスは、ホストマシンに割り当てられている必要があります。

1. 以下のコマンドを実行して、割り当てる USB デバイスを特定します。

```
# lsusb -v

idVendor          0x17ef Lenovo
idProduct         0x480f Integrated Webcam [R5U877]
```

2. XML ファイルを作成し、そのファイルに論理名 (例: **usb\_device.xml**) を指定します。ベンダーと製品 ID (16 進数の番号) は、検索時に表示されたものと全く同じものをコピーするようにしてください。この情報は、[図21.2 「USB デバイス XML スニペット」](#) に示されるように XML ファイルに追加します。このファイルの名前は次のステップで必要になるので留意してください。

```
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x17ef' />
    <product id='0x480f' />
  </source>
</hostdev>
```

**図21.2 USB デバイス XML スニペット**

3. 以下のコマンドを実行してデバイスを割り当てます。コマンドの実行時に、**guest1** を仮想マシンの名前に、**usb\_device.xml** を直前のステップで作成したデバイスのベンダーと製品 ID を含む XML ファイルの名前に置き換えます。次回の起動時に変更を有効にするには、**--config** を使用します。変更を現在のゲスト仮想マシンで有効にするには、**--current** 引数を使用します。追加の引数については **virsh man** ページを参照してください。

```
# virsh attach-device guest1 --file usb_device.xml --config
```

#### 例21.59 ゲスト仮想マシンからデバイスのホットアンプラグを実行する方法

以下の例では、**usb\_device1.xml** ファイルで設定した USB デバイスが **guest1** 仮想マシンから割り当て解除されます。

```
# virsh detach-device guest1 --file usb_device.xml
```

### 21.27.6.2. インターフェースデバイスの割り当て

**virsh attach-interface domain type source [<target>] [<mac>] [<script>] [<model>] [<inbound>] [<outbound>] [--config] [--live] [--current]** コマンドは以下の引数を取ります。

- **--type:** インターフェースのタイプを設定できます。
- **--source:** ネットワークインターフェースのソースを設定できます。
- **--live:** 実行中のゲスト仮想マシンの設定からその値を取得します。
- **--config:** 次回起動時に有効になります。
- **--current:** 現在の設定に基づいて値を取得します。
- **--target:** ゲスト仮想マシン内のターゲットデバイスを示します。
- **--mac:** ネットワークインターフェースの MAC アドレスを指定するには、このオプションを使用します。
- **--script:** デフォルトのスクリプトファイルの代わりにブリッジを処理するスクリプトファイルへのパスを指定するには、このオプションを使用します。
- **--model:** モデルのタイプを指定するには、このオプションを使用します。
- **--inbound:** インターフェースの受信帯域幅を制御します。使用できる値は、**average**、**peak**、および **burst** です。
- **--outbound:** インターフェースの送信帯域幅を制御します。使用できる値は、**average**、**peak**、および **burst** です。



### 注記

平均およびピークの値は 1 秒あたりのキロバイト単位で表示されますが、バースト値は [Network XML upstream documentation](#) で説明されているピーク速度時の単一バーストにおけるキロバイト単位で表示されます。

**type** は、物理ネットワークデバイスを指定する場合は **network** か、デバイスへのブリッジを指定する場合は **bridge** のいずれかにすることができます。**source** はデバイスのソースです。割り当てられたデバイスを除去するには、**virsh detach-device** コマンドを使用します。

#### 例21.60 デバイスをゲスト仮想マシンに割り当てる方法

以下の例では、**networkw** ネットワークデバイスが **guest1** 仮想マシンに割り当てられます。ゲストには、インターフェイスモデルは **virtio** として提示されます。

```
# virsh attach-interface guest1 networkw --model virtio
```

### 21.27.6.3. CDROM メディアの変更

**virsh change-media** コマンドは CDROM のメディアを別のソースまたはフォーマットに変更します。このコマンドは以下の引数を取ります。これらの引数の例および説明については、**man** ページを参照してください。

- **--path:** ディスクデバイスの完全修飾パスまたはターゲットを含む文字列です。
- **--source:** メディアのソースを含む文字列です。
- **--eject:** メディアを取り出します。
- **--insert:** メディアを挿入します。
- **--update:** メディアを更新します。
- **--current:** ハイパーバイザードライバーの実装によって、**--live** と **--config** のいずれかまたはその両方にすることができます。
- **--live:** 実行中のゲスト仮想マシンの動作中の設定を変更します。
- **--config:** 永続的な設定を変更します。次回起動時に反映されます。
- **--force:** メディアの変更を強制します。

### 21.27.7. ホストの一時停止

**virsh nodesuspend *targetduration*** コマンドは、ホストマシンを **s3 (suspend-to-RAM)**、**s4 (suspend-to-disk)**、または **Hybrid-Suspend** と同様のシステム全体でのスリープ状態に置き、リアルタイムクロックをセットアップして、設定された期間が経過した後にホストを起動します。**target** 変数は、**mem**、**disk**、または **hybrid** のいずれかに設定できます。これらのオプションは、一時停止するメモリー、ディスク、またはこれら 2 つの組み合わせを設定するために指定します。**--duration** を設定すると、設定された期間が経過した後にノードが起動するよう指示されます。期間は秒単位で設定されます。60 秒を超える期間に設定することが推奨されます。

#### 例21.61 ゲスト仮想マシンをディスク s4 に一時停止する方法

以下の例では、ホスト物理マシンをディスクに 90 秒間一時停止します。

```
# virsh nodesuspend disk 90
```

### 21.27.8. ノードメモリーパラメーターの設定および表示

**virsh node-memory-tune [shm-pages-to-scan] [shm-sleep-miliseconds] [shm-merge-across-nodes]** コマンドは、ノードメモリーパラメーターを表示し、これを設定することを可能にします。このコマンドでは、以下のパラメーターが設定できます。

- **--shm-pages-to-scan** - kernel samepage merging (KSM) サービスがスリープに入る前にスキャンするページ数を設定します。
- **--shm-sleep-miliseconds** - KSM サービスが次のスキャンの前にスリープするミリ秒数を設定します。
- **--shm-merge-across-nodes** - 異なる NUMA ノードからページをマージできるかどうかを指定します。

#### 例21.62 NUMA ノード間でメモリーページをマージする方法

以下の例では、NUMA ノードすべてからの全メモリーページをマージします。

```
# virsh node-memory-tune --shm-merge-across-nodes 1
```

### 21.27.9. ホスト上デバイスの一覧表示

**virsh nodedev-list --cap --tree** コマンドは、**libvirt** サービスが認識しているホスト上の利用可能なすべてのデバイスを一覧表示します。**--cap** を使用すると、一覧を機能タイプでフィルターしてそれぞれがコンマ区切りになりますが、**--tree** と共に使用することはできません。引数 **--tree** を使用すると、出力は以下に示すようなツリー構造になります。

#### 例21.63 ホスト上で利用可能なデバイスの表示方法

以下の例では、ホスト上で利用可能なデバイスをツリー構造で一覧表示します。結果は一部省略されていることに注意してください。

```
# virsh nodedev-list --tree
computer
|
+- net_lo_00_00_00_00_00_00
+- net_macvtap0_52_54_00_12_fe_50
+- net_tun0
+- net_virbr0_nic_52_54_00_03_7d_cb
+- pci_0000_00_00_0
+- pci_0000_00_02_0
+- pci_0000_00_16_0
+- pci_0000_00_19_0
|   |
|   +- net_eth0_f0_de_f1_3a_35_4f
[...]
```

以下の例では、ホスト上で利用可能な **SCSI** デバイスが表示されます。

```
# virsh nodedev-list --cap scsi
scsi_0_0_0_0
```

### 21.27.10. ホストマシン上でのデバイス作成

**virsh nodedev-create file** コマンドを使用すると、ホスト物理マシン上にデバイスを作成し、それをゲスト仮想マシンに割り当てることができます。**libvirt** は使用できるホストノードを自動的に検出しますが、このコマンドは **libvirt** が検出しなかったホストハードウェアの登録を可能にします。この指定されたファイルには、ホストデバイスの上位レベルの **<device>** XML 記述が含まれている必要があります。このファイルの例については、[例21.66 「デバイスの XML ファイルを取得する方法」](#) を参照してください。

#### 例21.64 XML ファイルによるデバイスの作成方法

以下の例では、PCI デバイス用の XML ファイルが作成済みで、**scsi\_host2.xml** として保存されていることを前提としています。以下のコマンドを実行すると、このデバイスをゲストに割り当てることができます。

```
# virsh nodedev-create scsi_host2.xml
```

### 21.27.11. デバイスの削除

**virsh nodedev-destroy** コマンドは、デバイスをホストから削除します。**virsh** ノードデバイスドライバーは永続的な設定をサポートしないため、ゲストを再起動するとデバイスも使用できなくなることにご注意ください。

異なる割り当てでは、デバイスが異なるバックエンドドライバー (**vfi**o、**kvm**) にバインドされる可能性があることにも注意してください。**--driver** 引数を使用することにより、必要なバックエンドドライバーを指定することができます。

#### 例21.65 デバイスをホスト物理マシンから削除する方法

以下の例では、ホストマシンから **scsi\_host2** という名前の SCSI デバイスが削除されます。

```
# virsh nodedev-destroy scsi_host2
```

### 21.27.12. デバイス設定の収集

**virsh nodedev-dumpxml device** コマンドは、指定したホストデバイスの XML 表現を出力します。この XML 表現には、デバイス名、デバイスが接続されている BUS、ベンダー、製品 ID、ケイパビリティ、および **libvirt** で使用できる情報が含まれます。引数 **device** はデバイス名か、または WWNN、WWPN フォーマット (HBA のみ) の WWN ペアのいずれかにすることができます。

#### 例21.66 デバイスの XML ファイルを取得する方法

以下の例では、**scsi\_host2** という SCSI デバイスの XML ファイルが取得されます。この名前は、**virsh nodedev-list** コマンドを実行することで確認できます。

```
# virsh nodedev-dumpxml scsi_host2
<device>
  <name>scsi_host2</name>
  <parent>scsi_host1</parent>
  <capability type='scsi_host'>
    <capability type='fc_host'>
      <wwnn>2001001b32a9da5b</wwnn>
      <wwpn>2101001b32a9da5b</wwpn>
    </capability>
  </capability>
</device>
```

### 21.27.13. デバイスのリセットのトリガー

**virsh nodedev-reset device** コマンドは、指定されたデバイスのリセットをトリガーします。このコマンドは、ゲスト仮想マシンのパススルーとホスト物理マシン間でノードデバイスを転送する前に実行すると有効です。**libvirt** は必要に応じてこのアクションを自動的に実行しますが、このコマンドは必要な場合に明示的なリセットを許可します。

#### 例21.67 ゲスト仮想マシンでデバイスをリセットする方法

以下の例では、ゲスト仮想マシン上の `scsi_host2` という名前のデバイスをリセットします。

```
# virsh nodedev-reset scsi_host2
```

## 21.28. ゲスト仮想マシン情報の取得

### 21.28.1. ゲスト仮想マシンのドメイン ID の取得

**virsh domid** コマンドはゲスト仮想マシンのドメイン ID を返します。これはゲストの起動または再起動が行われるたびに変更されることに注意してください。このコマンドには仮想マシンの名前か、または仮想マシンの UUID のいずれかが必要です。

#### 例21.68 ゲスト仮想マシンのドメイン ID を取得する方法

以下の例では、ゲスト仮想マシン `guest1` のドメイン ID が取得されます。

```
# virsh domid guest1
8
```

**domid** は停止中のゲスト仮想マシンだと、`-` を返すことに注意してください。仮想マシンが停止されていることを確認するには、**virsh list --all** コマンドを使用します。

### 21.28.2. ゲスト仮想マシンのドメイン名の取得

**virsh domname** コマンドは、ゲスト仮想マシンの ID または UUID を渡すと、ゲスト仮想マシンの名前を返します。ID はゲストが起動されるたびに変更されることに注意してください。

#### 例21.69 仮想マシンの名前を取得する方法

以下の例では、ゲスト仮想マシンの ID が `8` である仮想マシンの名前が表示されます。

```
# virsh domname 8
guest1
```

### 21.28.3. ゲスト仮想マシンの UUID の取得

**virsh domuuid** コマンドは、指定されたゲスト仮想マシンまたは ID の UUID または *Universally Unique Identifier* を返します。

#### 例21.70 ゲスト仮想マシンの UUID を表示する方法

以下の例では、ゲスト仮想マシン `guest1` の UUID が取得されます。

```
# virsh domuuid guest1
r5b2-mysql01 4a4c59a7-ee3f-c781-96e4-288f2862f011
```

### 21.28.4. ゲスト仮想マシン情報の表示

**virsh dominfo** コマンドは、仮想マシンの名前、ID または UUID と共にゲスト仮想マシンの情報を表示します。ID は仮想マシンが起動するたびに変更されることに注意してください。

#### 例21.71 ゲスト仮想マシンの全般的な詳細情報を表示する方法

以下の例では、仮想マシン *guest1* についての全般的詳細が表示されます。

```
# virsh dominfo guest1
Id:      8
Name:    guest1
UUID:    90e0d63e-d5c1-4735-91f6-20a32ca22c48
OS Type: hvm
State:   running
CPU(s):  1
CPU time: 32.6s
Max memory: 1048576 KiB
Used memory: 1048576 KiB
Persistent: yes
Autostart: disable
Managed save: no
Security model: selinux
Security DOI: 0
Security label: system_u:system_r:svirt_t:s0:c552,c818 (enforcing)
```

## 21.29. ストレージプールコマンド

**libvirt** を使用すると、ストレージボリュームを仮想マシン内のデバイスとして表示するために使用されるファイル、**raw** パーティション、およびドメイン固有の形式を含む、さまざまなストレージソリューションを管理できます。この機能についての詳細は、[libvirt upstream pages](#) を参照してください。ストレージプールを管理するためのコマンドの多くは、ゲスト仮想マシンに使用されるコマンドに類似したものです。

### 21.29.1. ストレージプール XML の検索

**virsh find-storage-pool-sources type** コマンドは、検索できる所定のソースのすべてのストレージプールを記述した XML を表示します。タイプには、**netfs**、**disk**、**dir**、**fs**、**iscsi**、**logical**、および **gluster** が含まれます。すべてのタイプがストレージのバックエンドドライバに対応し、さらに多くのタイプが利用可能であることに注意してください (詳細は、**man** ページを参照してください)。--**srcSpec** オプションを使用してテンプレートソースの XML ファイルを提供すると、プールのクエリーをさらに制限することができます。

#### 例21.72 利用可能なストレージプールの XML 設定を一覧表示する方法

以下の例では、システム上で利用可能なすべての論理ストレージプールの XML 設定が出力されます。

```
# virsh find-storage-pool-sources logical
<sources>
  <source>
    <device path='/dev/mapper/luks-7a6bfc59-e7ed-4666-a2ed-6dcbbff287149' />
```

```

        <name>RHEL_dhcp-2-157</name>
        <format type='lvm2' />
    </source>
</sources>

```

### 21.29.2. ストレージプールの検索

**virsh find-storage-pool-sources-as type** コマンドは、特定のタイプを渡すと、ストレージプールのソースを検出します。タイプには、**netfs**、**disk**、**dir**、**fs**、**iscsi**、**logical** および **gluster** が含まれます。すべてのタイプはストレージバックエンドドライバーに対応しており、さらに多くのタイプが利用できることに注意してください（詳細は、**man** ページを参照してください）。また、このコマンドはオプションの引数 **host**、**port** および **initiator** を取ります。これらのオプションのそれぞれは照会される内容を決定します。

#### 例21.73 ストレージプールソースを検索する方法

以下の例では、指定したホストマシン上でディスクベースのストレージプールを検索します。ホスト名が分からない場合は、まず **virsh hostname** コマンドを実行します。

```
# virsh find-storage-pool-sources-as disk --host myhost.example.com
```

### 21.29.3. ストレージプール情報の一覧表示

**virsh pool-info pool** コマンドは、指定されたストレージプールオブジェクトについての基本的な情報を一覧表示します。このコマンドには、ストレージプールの名前または **UUID** が必要になります。この情報を取得するには、**pool-list** を使用します。

#### 例21.74 ストレージプールで情報を取得する方法

以下の例では、**vdisk** という名前のストレージプール上の情報が取得されます。

```
# virsh pool-info vdisk

Name:          vdisk
UUID:
State:         running
Persistent:    yes
Autostart:     no
Capacity:      125 GB
Allocation:    0.00
Available:     125 GB
```

### 21.29.4. 利用可能なストレージプールの一覧表示

**virsh pool-list** コマンドは、**libvirt** に認識されるすべてのストレージプールオブジェクトを一覧表示します。デフォルトでは、アクティブなプールのみが一覧表示されますが、**--inactive** 引数を使用すると、アクティブではないプールのみが一覧表示され、**--all** 引数を使用すると、ストレージプールのすべてが一覧表示されます。このコマンドは、検索結果をフィルターする以下のオプションの引数を取ります。



- **--inactive**: アクティブではないストレージプールを一覧表示します。
- **--all**: アクティブおよびアクティブではないストレージプールを一覧表示します。
- **--persistent**: 永続的なストレージプールを一覧表示します。
- **--transient**: 一時的なストレージプールを一覧表示します。
- **--autostart**: 自動起動を有効にしたストレージプールを一覧表示します。
- **--no-autostart**: 自動起動を無効にしたストレージプールを一覧表示します。
- **--type type**: 指定したタイプ限定のプールを表示します。
- **--details**: ストレージプールの拡張された詳細を一覧表示します。

上記の引数のほかにも、一覧の内容をフィルターするために使用できるフィルターフラグの複数のセットがあります。**--persistent** は、一覧を永続的なプールに制限し、**--transient** は一覧を一時的なプールに制限します。また、**--autostart** は一覧を自動起動のプールに制限し、最後に **--no-autostart** は一覧を自動起動が無効にされたストレージプールに制限します。

**--type** を必要とするストレージプールコマンドではすべて、プールのタイプはコンマで区切る必要があります。有効なプールのタイプには、**dir**、**fs**、**netfs**、**logical**、**disk**、**iscsi**、**scsi**、**mpath**、**rbid**、および **sheepdog** と **gluster** があります。

**--details** オプションは **virsh** に対し、プールの永続性や容量に関連した情報を追加で表示します (ある場合)。



## 注記

このコマンドが古いサーバーに対して使用される場合、競合を継承する一連の API 呼び出しの使用が強制されます。これにより、一覧が収集されている間に複数の呼び出し間の状態が変更される場合、プールは一覧表示されなくなるか、または複数回表示される可能性があります。ただし、新しいサーバーの場合は、この問題は発生しません。

### 例21.75 すべてのストレージプールを一覧表示する方法

以下の例では、アクティブと非アクティブの両方のストレージプールが一覧表示されます。

```
# virsh pool-list --all
Name                State      Autostart
-----
default             active     yes
vdisk                active     no
```

## 21.29.5. ストレージプール一覧の更新

**virsh pool-refresh pool** コマンドは、ストレージプールに含まれるストレージボリュームの一覧を更新します。

### 例21.76 ストレージプール内のストレージボリュームの一覧を更新する方法

以下の例では、**vdisk** という名前のストレージプール内のボリューム一覧が更新されます。

```
# virsh pool-refresh vdisk

Pool vdisk refreshed
```

## 21.29.6. ストレージプールの作成、定義、および起動

### 21.29.6.1. ストレージプールの構築

**virsh pool-build pool** コマンドは、コマンドで指定された名前を使ってストレージプールを構築します。オプションの引数 **--overwrite** および **--no-overwrite** は FS ストレージプールのみを使用できるか、またはディスクまたは論理タイプベースのストレージプールで使用できます。[**--overwrite**] または [**--no-overwrite**] のいずれも指定されておらず、使用されるプールが FS の場合、タイプはディレクトリベースであることが想定されます。プール名に加えて、ストレージプール UUID も使用できます。

**--no-overwrite** が指定されない場合、コマンドはファイルシステムがすでにターゲットデバイスに存在するかどうかを判別するためにプローブし、存在する場合にはエラーを返します。存在しない場合にはターゲットデバイスをフォーマットするために **mkfs** を使用します。**--overwrite** が指定される場合、**mkfs** コマンドが実行され、ターゲットデバイス上のすべての既存データが上書きされます。

#### 例21.77 ストレージプールを構築する方法

以下の例では、**vdisk** という名前のディスクベースのストレージプールが作成されます。

```
# virsh pool-build vdisk

Pool vdisk built
```

### 21.29.6.2. XML ファイルからのストレージプールの定義

**virsh pool-define file** コマンドは、XML ファイルからストレージプールオブジェクトを作成しますが、これを起動しません。

#### 例21.78 XML ファイルからストレージプールを定義する方法

以下の例では、ストレージプールの設定と共に XML ファイルが作成済みであることが想定されています。

```
<pool type="dir">
  <name>vdisk</name>
  <target>
    <path>/var/lib/virt/images</path>
  </target>
</pool>
```

以下のコマンドでは、XML ファイル (この例では **vdisk.xml** という名前) からディレクトリタイプのストレージプールが構築されます。

```
# virsh pool-define vdisk.xml
```

```
Pool vdisk defined
```

ストレージプールが定義されたことを確認するには、[例21.75「すべてのストレージプールを一覧表示する方法」](#)に示されているように **virsh pool-list --all** コマンドを実行します。ただし、このコマンドの実行時には、プールが起動していないためにステータスは非アクティブになります。ストレージプールの説明については、[例21.82「ストレージプールの開始方法」](#)を参照してください。

### 21.29.6.3. ストレージプールの作成

**virsh pool-create file** コマンドはストレージプールをその関連付けられた XML ファイルから作成し、起動します。

#### 例21.79 XML ファイルによるストレージプールの作成方法

以下の例では、ストレージプールの設定と共に XML ファイルが作成済みであることが想定されています。

```
<pool type="dir">
  <name>vdisk</name>
  <target>
    <path>/var/lib/virt/images</path>
  </target>
</pool>
```

以下のコマンドでは、XML ファイル (この例では *vdisk.xml* という名前) からディレクトリータイプのストレージプールが構築されます。

```
# virsh pool-create vdisk.xml
```

```
Pool vdisk created
```

ストレージプールが作成されたことを確認するには、[例21.75「すべてのストレージプールを一覧表示する方法」](#)に示されているように **virsh pool-list --all** コマンドを実行します。ただし、このコマンドの実行時にはプールが起動していないためにステータスは非アクティブになります。ストレージプールの説明については、[例21.82「ストレージプールの開始方法」](#)を参照してください。

### 21.29.6.4. ストレージプールの作成

**virsh pool-create-as name** コマンドは、指定される生のパラメーターからプールオブジェクト名を作成し、これを開始します。このコマンドは以下のオプションを取ります。

- **--print-xml** - XML ファイルの内容を表示しますが、それからストレージプールを定義したり、作成したりすることはありません。
- **--type type** - ストレージプールのタイプを定義します。使用可能なタイプについては、「[利用可能なストレージプールの一覧表示](#)」を参照してください。

- **--source-host hostname** - 基礎となるストレージのソースのホスト物理マシンです。
- **--source-path path** - 基礎となるストレージの場所です。
- **--source-dev path** - 基礎となるストレージのデバイスです。
- **--source-name name** - ソースの基礎となるストレージの名前です。
- **--source-format format** - ソースの基礎となるストレージのフォーマットです。
- **--target path** - 基礎となるストレージのターゲットです。

### 例21.80 ストレージプールの作成および開始方法

以下の例では、`/mnt` ディレクトリーに `vdisk` という名前のストレージプールが作成されて、開始されます。

```
# virsh pool-create-as --name vdisk --type dir --target /mnt
Pool vdisk created
```

### 21.29.6.5. ストレージプールの定義

**virsh pool-define-as <name>** コマンドは指定された生のパラメーターからプールオブジェクト名を作成しますが、これを開始しません。このコマンドは以下のオプションを受け入れます。

- **--print-xml** - XML ファイルの内容を表示しますが、それからストレージプールを定義したり、作成したりすることはありません。
- **--type type** - ストレージプールのタイプを定義します。使用可能なタイプについては、「[利用可能なストレージプールの一覧表示](#)」を参照してください。
- **--source-host hostname** - 基礎となるストレージのソースのホスト物理マシンです。
- **--source-path path** - 基礎となるストレージの場所です。
- **--source-dev devicename** - 基礎となるストレージのデバイスです。
- **--source-name sourcename** - ソースの基礎となるストレージの名前です。
- **--source-format format** - ソースの基礎となるストレージのフォーマットです。
- **--target targetname** - 基礎となるストレージのターゲットです。

**--print-xml** が指定される場合、プールを作成したり定義したりせずに、プールオブジェクトの XML が出力されます。指定されない場合、プールでは指定されたタイプを構築する必要があります。**type** を必要とするすべてのストレージプールコマンドでは、プールのタイプはコマンドで区切る必要があります。有効なプールのタイプには、**dir**、**fs**、**netfs**、**logical**、**disk**、**iscsi**、**scsi**、**mpath**、**rbd**、**sheepdog** および **gluster** が含まれます。

### 例21.81 ストレージプールを定義する方法

以下の例では、**vdisk** という名前のストレージプールが定義されますが、開始はされません。このコマンドを実行した後に、**virsh pool-start** コマンドを使用してこのストレージプールをアクティベートします。

```
# virsh pool-define-as --name vdisk --type dir --target /mnt  
  
Pool vdisk defined
```

#### 21.29.6.6. ストレージプールの開始

**virsh pool-start pool** コマンドは指定されたストレージプールを開始します。これは事前に定義されており、アクティブな状態ではありません。このコマンドではプールの名前と共にストレージプールの UUID も使用する場合があります。

##### 例21.82 ストレージプールの開始方法

以下の例では、[例21.79「XML ファイルによるストレージプールの作成方法」](#)で構築した **vdisk** ストレージプールを開始します。

```
# virsh pool-start vdisk  
  
Pool vdisk started
```

プールが開始したことを確認するには、[例21.75「すべてのストレージプールを一覧表示する方法」](#)に示されるように **virsh pool-list --all** コマンドを開始し、ステータスがアクティブであることを確認します。

#### 21.29.6.7. ストレージプールの自動起動

**virsh pool-autostart pool** コマンドは、起動時にストレージプールが自動的に起動するようにします。このコマンドにはプール名または UUID が必要です。**pool-autostart** コマンドを無効にするには、コマンドで **--disable** 引数を使用します。

##### 例21.83 ストレージプールの自動起動を実行する方法

以下の例では、[例21.79「XML ファイルによるストレージプールの作成方法」](#)で構築した **vdisk** ストレージプールを自動起動します。

```
# virsh pool-autostart vdisk  
  
Pool vdisk autostarted
```

#### 21.29.7. ストレージプールの停止および削除

**virsh pool-destroy pool** コマンドはストレージプールを停止します。一旦停止すると、libvirt はプールを管理しなくなりますが、プールに含まれる生データは変更されず、後で **pool-create** コマンドを使って復元できます。

##### 例21.84 ストレージプールの停止方法

以下の例では、例21.79「XML ファイルによるストレージプールの作成方法」で構築した **vdisk** ストレージプールを停止します。

```
# virsh pool-destroy vdisk

Pool vdisk destroyed
```

**virsh pool-delete *pool*** コマンドは、指定されたストレージプールで使用されたリソースを破棄します。この操作は回復不可能で、元に戻すことができないことに注意してください。ただし、プール構成はこのコマンドの実行後もそのまま存在し、新規ストレージボリュームの作成が可能です。

#### 例21.85 ストレージプールを削除する方法

以下の例では、例21.79「XML ファイルによるストレージプールの作成方法」で構築した **vdisk** ストレージプールを削除します。

```
# virsh pool-delete vdisk

Pool vdisk deleted
```

**virsh pool-undefine *pool*** コマンドは、非アクティブなプールの設定の定義を解除します。

#### 例21.86 ストレージプールの定義を解除する方法

以下の例では、例21.79「XML ファイルによるストレージプールの作成方法」で構築した **vdisk** ストレージプールの定義を解除します。これにより、ストレージプールは一時的なものになります。

```
# virsh pool-undefine vdisk

Pool vdisk undefined
```

### 21.29.8. プール用の XML ダンプファイルの作成

**virsh pool-dumpxml *pool*** コマンドは、指定されたストレージプールオブジェクトについての XML 情報を返します。オプション **--inactive** を使用すると、現在のプール設定ではなくプールの次回開始時に使用される設定がダンプされます。

#### 例21.87 ストレージプール設定の取得方法

以下の例では、例21.79「XML ファイルによるストレージプールの作成方法」で構築した **vdisk** ストレージプール設定を取得します。設定ファイルは端末で開かれます。

```
# virsh pool-dumpxml vdisk
<pool type="dir">
  <name>vdisk</name>
  <target>
    <path>/var/lib/virt/images</path>
  </target>
</pool>
```

### 21.29.9. ストレージプール設定ファイルの編集

**pool-edit pool** コマンドは、指定されたストレージプールの XML 設定ファイルを編集用を開きます。

この方法は、適用前のエラーチェックが可能な、XML 設定ファイルの編集に使用できる唯一の方法になります。

#### 例21.88 ストレージプール設定の編集方法

以下の例では、例21.79「XML ファイルによるストレージプールの作成方法」で構築した **vdisk** ストレージプール設定を編集します。設定ファイルはデフォルトのエディターで開かれます。

```
# virsh pool-edit vdisk
<pool type="dir">
  <name>vdisk</name>
  <target>
    <path>/var/lib/virt/images</path>
  </target>
</pool>
```

### 21.30. ストレージボリュームコマンド

このセクションでは、ストレージボリュームを作成し、削除し、管理するためのコマンドについて説明します。ストレージボリュームの作成には、少なくとも1つのストレージプールが必要になります。たとえば、ストレージプールの作成方法を示す例については、例21.79「XML ファイルによるストレージプールの作成方法」を参照してください。ストレージプールについての情報は、13章ストレージプールを参照してください。ストレージボリュームについての情報は、14章ストレージボリュームを参照してください。

#### 21.30.1. ストレージボリュームの作成

**virsh vol-create-from pool file vol** コマンドは、別のボリュームを入力として使用してボリュームを作成します。このコマンドには、ストレージプール名またはストレージプール UUID のいずれかが必要になります。このコマンドは、以下のパラメーターおよびオプションを受け入れます。

- **--pool string** (必須): ストレージプールの名前、またはストレージボリュームに割り当てられるストレージプールの UUID が含まれます。このストレージプールは、この新規ストレージボリュームのベースとするために使用しているストレージボリュームに関連付けられているものと同じストレージプールである必要はありません。
- **--file string** (必須): ストレージボリュームのパラメーターが含まれる XML ファイルの名前が含まれます。
- **--vol string** (必須): この新規ストレージボリュームのベースとするために使用しているストレージボリュームの名前が含まれます。
- **--inputpool string** (オプション): 新規ストレージボリュームの入力として使用しているストレージボリュームに関連付けられるストレージプールに名前を付けることができます。



- **--prealloc-metadata** (オプション): 新規ストレージボリュームのメタデータを事前に割り当てます (完全割り当てではなく **qcow2** 用)。

例については、「[ボリュームの作成](#)」を参照してください。

### 21.30.2. パラメーターによるストレージボリュームの作成

**virsh vol-create-as pool name capacity** コマンドは、一連の引数からボリュームを作成します。**pool** 引数には、ボリュームを作成するストレージプールの名前または UUID が含まれます。このコマンドは以下の必要なパラメーターおよびオプションを取ります。

- **[--pool] string** (必須): 関連付けられたストレージプールの名前が含まれます。
- **[--name] string** (必須): 新規ストレージボリュームの名前が含まれます。
- **[--capacity] string** (必須): 整数で表されるストレージボリュームのサイズが含まれます。デフォルトは、特に指定がない場合はバイトになります。バイト、キロバイト、メガバイト、ギガバイト、およびテラバイトのそれぞれにサフィックス **b**、**k**、**M**、**G**、**T** を使用します。
- **--allocation string** (オプション): 整数で表される初期の割り当てサイズが含まれます。デフォルトは、特に指定がない場合はバイトになります。
- **--format string** (オプション): ファイル形式のタイプが含まれます。受け入れ可能なタイプには、**raw**、**bochs**、**qcow**、**qcow2**、**qed**、**host\_device**、および **vmdk** が含まれます。ただし、これらはファイルベースのストレージプールのみを対象にしています。デフォルトで使用する **qcow** バージョンはバージョン 3 です。バージョンを変更する場合は、「[target 要素の設定](#)」を参照してください。
- **--backing-vol string** (オプション): バックアップボリュームが含まれます。これは、スナップショットを取る場合に使用されます。
- **--backing-vol-format string** (オプション): バックアップボリュームの形式が含まれます。これは、スナップショットを取る場合に使用されます。
- **--prealloc-metadata** (オプション): メタデータを事前に割り当てることができます (完全割り当てではなく **qcow2** 用のみ)。

#### 例21.89 パラメーターのセットでストレージボリュームを作成する方法

以下の例では、100MB の **vol-new** という名前のストレージボリュームを作成します。これには、[例 21.79「XML ファイルによるストレージプールの作成方法」](#) で構築した **vdisk** ストレージプールが含まれます。

```
# virsh vol-create-as vdisk vol-new 100M

vol vol-new created
```

### 21.30.3. XML ファイルによるストレージボリュームの作成

**virsh vol-create pool file** コマンドは、ストレージボリュームパラメーターが含まれる XML ファイルから新規ストレージボリュームを作成します。



**例21.90 既存の XML ファイルからストレージボリュームを作成する方法**

以下の例では、下記の *vol-new.xml* をベースとしたストレージボリュームが作成されます。

```
<volume>
  <name>vol-new</name>
  <allocation>0</allocation>
  <capacity unit="M">100</capacity>
  <target>
    <path>/var/lib/virt/images/vol-new</path>
    <permissions>
      <owner>107</owner>
      <group>107</group>
      <mode>0744</mode>
      <label>virt_image_t</label>
    </permissions>
  </target>
</volume>
```

ストレージボリュームはストレージプール *vdisk* に関連付けられます。イメージへのパスは ***/var/lib/virt/images/vol-new*** です。

```
# virsh vol-create vdisk vol-new.xml

vol vol-new created
```

**21.30.4. ストレージボリュームのクローン作成**

**virsh vol-clone vol-name new-vol-name** コマンドは、既存のストレージボリュームのクローンを作成します。**virsh vol-create-from** を使用することもできますが、ストレージボリュームのクローン作成には推奨されません。**--pool string** オプションを使用すると、新規ストレージボリュームに関連付けられるストレージプールを指定できます。**vol** 引数は、ソースストレージボリュームの名前またはキー、あるいはパスです。**name** 引数は、新規ストレージボリューム名前を参照します。詳細は「[ボリュームのクローン作成](#)」を参照してください。

**例21.91 ストレージボリュームをクローン作成する方法**

以下の例では、*vol-new* というストレージボリュームを新規の *vol-clone* というボリュームにクローンします。

```
# virsh vol-clone vol-new vol-clone

vol vol-clone cloned from vol-new
```

**21.31. ストレージボリュームの削除**

**virsh vol-delete vol pool** コマンドは指定されたボリュームを削除します。このコマンドには、ボリュームが格納されているストレージプールの名前または **UUID** と、ストレージボリュームの名前が必要です。ボリューム名の代わりに、削除するボリュームのキーまたはパスも使用できます。

**例21.92 ストレージボリュームを削除する方法**

以下の例では、`vdisk` というストレージプールを格納している `new-vol` という名前のストレージボリュームが削除されます。

```
# virsh vol-delete new-vol vdisk

vol new-vol deleted
```

**21.32. ストレージボリュームの内容の削除**

`virsh vol-wipe vol pool` コマンドは、以前ボリュームに格納されていたデータが将来の読み取りでアクセスされないようにボリュームを消去します。このコマンドには、`--pool pool` が必要です。これはボリュームが置かれているストレージプールの名前または UUID であり、`pool` は消去するボリュームの名前、キーまたはパスです。引数 `--algorithm` および以下のサポートされるアルゴリズムのタイプを使用して、ボリュームをゼロで再作成する代わりに異なる消去アルゴリズムを選択できることに注意してください。

- **zero** - 1-pass all zeroes
- **nnsa** - リムーバブルおよび非リムーバブルハードディスクのサニタイズ用 4-pass NNSA Policy Letter NAP-14.1-C (XVI-8) : random x2, 0x00, verify.
- **dod** - リムーバブルおよび非リムーバブル固定ディスクのサニタイズ用 4-pass DoD 5220.22-M section 8-306 procedure: random, 0x00, 0xff, verify.
- **bsi** - German Center of Security in Information Technologies (<http://www.bsi.bund.de>) が推奨する 9-pass method : 0xff, 0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f.
- **gutmann** - Gutmann's paper で説明されている正規の 35-pass sequence.
- **schneier** - "Applied Cryptography" (1996) で Bruce Schneier が説明している 7-pass method : 0x00, 0xff, random x5.
- **pfitzner7** - Roy Pfitzner's 7-random-pass method: random x7
- **pfitzner33** - Roy Pfitzner's 33-random-pass method: random x33.
- **random** - 1-pass pattern: random.s

**注記**

アルゴリズムの可用性は、ホストにインストールされている「scrub」バイナリーのバージョンによって制限される可能性があります。

**例21.93 ストレージボリュームの内容を削除する方法 (ストレージボリュームを消去する方法)**

以下の例では、`vdisk` というストレージプールに関連付けられている `new-vol` という名前のストレージボリュームのコンテンツが削除されます。

```
# virsh vol-wipe new-vol vdisk  
  
vol new-vol wiped
```

### 21.33. ストレージボリューム情報の XML ファイルへのダンプ

**virsh vol-dumpxml vol** コマンドは、ボリューム情報を取り、この内容で XML ファイルを作成し、これを標準出力 (stdout) ストリームの設定に出力します。オプションで、**--pool** オプションを使用して関連するストレージプールの名前を指定できます。

#### 例21.94 ストレージボリュームの内容をダンプする方法

以下の例では、ストレージボリューム *vol-new* のコンテンツを XML ファイルにダンプします。

```
# virsh vol-dumpxml vol-new
```

### 21.34. ボリューム情報の一覧表示

**virsh vol-info vol** コマンドは、指定されたストレージボリュームについての基本的な情報を一覧表示します。ストレージボリューム名、キー、またはパスのいずれかを指定する必要があります。のコマンドは、オプションの **--pool** も受け付けます。ここではストレージボリュームに関連付けられるストレージプールを指定できます。プール名または UUID のいずれかを指定することができます。

#### 例21.95 ストレージボリュームについての情報を表示する方法

以下の例では、ストレージボリューム *vol-new* についての情報を取得します。このコマンドを実行する際には、ストレージボリューム名を実際のものに置き換えてください。

```
# virsh vol-info vol-new
```

**virsh vol-list pool** コマンドは、指定されたストレージプールに関連付けられているすべてのボリュームを一覧表示します。このコマンドには、ストレージプールの名前または UUID が必要です。**--details** オプションを使用すると、**virsh** でボリュームタイプおよび容量に関連する情報が追加表示されます (存在する場合)。

#### 例21.96 ストレージボリュームに関連付けられているストレージボリュームを表示する方法

以下の例では、ストレージプール *vdisk* に関連付けられている全ストレージボリュームを一覧表示します。

```
# virsh vol-list vdisk
```

### 21.35. ストレージボリューム情報の取得

**virsh vol-pool vol** コマンドは、指定されたストレージボリュームのプール名または UUID を返します。デフォルトではストレージプール名が返されます。**--uuid** オプションが使用される場合、プー

ル UUID が代わりに返されます。このコマンドには、要求された情報を返すために使用するストレージプールのキーまたはパスが必要です。

#### 例21.97 ストレージボリュームの名前または UUID を表示する方法

以下の例では、`/var/lib/virt/images/vol-new`にあるストレージボリュームの名前を取得します。

```
# virsh vol-pool /var/lib/virt/images/vol-new  
  
vol-new
```

**vol-path --pool *pool-or-uuid* vol-name-or-key** コマンドは、所定のボリュームのパスを返します。このコマンドには、ボリュームに使用されるストレージプールの名前または UUID である **--pool *pool-or-uuid*** が必要です。さらに、パスが要求される際に使用されるボリュームの名前またはキーである **vol-name-or-key** が必要です。

**vol-name vol-key-or-path** コマンドは所定のボリュームの名前を返します。ここで、**vol-key-or-path** は名前を返すために使用されるボリュームのキーまたはパスです。

**vol-key --pool *pool-or-uuid* vol-name-or-path** コマンドは、所定のボリュームのボリュームキーを返します。ここで、**--pool *pool-or-uuid*** はボリュームに使用されるストレージプールの名前または UUID であり、**vol-name-or-path** はボリュームキーを返すために使用されるボリュームの名前またはパスです。

## 21.36. ストレージボリュームのアップロードおよびダウンロード

**vol-upload --pool *pool-or-uuid* --offset bytes --length bytes vol-name-or-key-or-path local-file** コマンドは、指定された *local-file* の内容をストレージボリュームにアップロードします。このコマンドには、ボリュームに使用されるストレージプールの名前または UUID である **--pool *pool-or-uuid*** が必要です。さらに、アップロードするボリュームの名前、キーまたはパスである **vol-name-or-key-or-path** も必要です。**--offset** 引数は、データの書き込みを開始するストレージボリューム内の位置を指します。**--length *length*** は、アップロードされるデータ量の上限を定めます。*local-file* が指定された **--length** の値よりも大きい場合はエラーが発生します。

**vol-download --pool *pool-or-uuid* --offset bytes -length bytes vol-name-or-key-or-path local-file** コマンドは、ストレージボリュームからローカルファイルのコンテンツをダウンロードします。

このコマンドには **--pool *pool-or-uuid*** オプションが必要です。ここで *pool-or-uuid* はボリュームに使用されるストレージプールの名前または UUID です。さらに、ダウンロードするボリュームの名前、キーまたはパスである **vol-name-or-key-or-path** も必要です。引数 **--offset** を使用すると、データの読み込みを開始するストレージボリューム内の位置が決定されます。**--length *length*** は、ダウンロードされるデータの量の上限を定めます。

## 21.37. ストレージボリュームのサイズ変更

**vol-resize --pool *pool-or-uuid* vol-name-or-path pool-or-uuid capacity --allocate --delta --shrink** コマンドは、所定のボリュームの容量 (バイト単位) のサイズ変更を行います。このコマンドには、ボリュームに使用されるストレージプールの名前または UUID である **--pool *pool-or-uuid*** が必要です。さらに、このコマンドにはサイズ変更するボリュームの名前、キーまたはパスである **vol-name-or-key-or-path** が必要です。

新たな容量は、**--allocate** が指定されない限りスパースになります。通常、容量は新規のサイズになりますが、**--delta** がある場合、既存のサイズに追加されます。**--shrink** がない場合、ボリュームを縮小する試みは失敗します。

**--shrink** が指定されていない限り、容量は負の値にならず、負の符号は不要であることに注意してください。**capacity** は、サフィックスがない場合、デフォルトがバイトになる単位付き整数です。また、このコマンドはアクティブなゲストによって使用されていないストレージボリュームの場合にのみ安全であることにも注意してください。ライブのサイズ変更については、「[ゲスト仮想マシンのブロックデバイスのサイズ変更](#)」を参照してください。

## 21.38. ゲスト仮想マシン別の情報の表示

### 21.38.1. ゲスト仮想マシンの表示

**virsh** を使ってアクティブなゲスト仮想マシンの一覧と、それらの現在の状態を表示するには、以下を実行します。

```
# virsh list
```

以下のオプションも使用できます。

- **--all** - すべてのゲスト仮想マシンを一覧表示します。

```
# virsh list --all
 Id Name                               State
-----
 0 Domain-0                           running
 1 Domain202                          paused
 2 Domain010                          shut off
 3 Domain9600                          crashed
```



#### 注記

**virsh list --all** を実行しても何も表示されない場合は、**root** ユーザーで仮想マシンが作成されていない可能性があります。

**virsh list --all** コマンドでは、以下の状態を認識します。

- **running - running** は、ゲスト仮想マシンが CPU 上で現在稼働中であることを示します。
- **idle - idle** 状態は、ゲスト仮想マシンが休止中のため、稼働していないか、または稼働できないことを示します。ゲスト仮想マシンが IO を待機している場合 (従来の待機状態) か、または作業がないためにスリープ状態になっている場合に、この状態になります。
- **paused** - ゲスト仮想マシンが一時停止になると、メモリーと他のリソースは消費しますが、ハイパーバイザーからの CPU リソースはスケジュールされません。**paused** 状態は、**virt-manager** または **virsh suspend** コマンドで **paused** ボタンを使用すると発生します。
- **in shutdown - in shutdown** 状態は、シャットダウンプロセスにあるゲスト仮想マシンの状態です。ゲスト仮想マシンには、シャットダウン信号が送られて、その稼働を正常に停止するプロセスに入ります。これはすべてのゲスト仮想マシンのオペレーティングシステ

ムでは機能しないかもしれません。一部のオペレーティングシステムはこれらの信号に反応しません。

- **shut off - shut off** 状態は、ゲスト仮想マシンが稼働していないことを示します。これは、ゲスト仮想マシンが完全にシャットダウンするか、または起動していないことが原因です。
- **crashed - crashed** 状態は、ゲスト仮想マシンがクラッシュしていることを示し、ゲスト仮想マシンがクラッシュ時に再起動しないように設定をされている場合にのみ発生します。
- **pmsuspended** - ゲストは、ゲスト電源管理により一時停止されています。
- **--inactive** - 定義されているものの、現在アクティブではないゲスト仮想マシンを一覧表示します。これには、**shut off** と **crashed** の状態も含まれます。
- **--managed-save - managed save** 状態を有効にしているゲストは **saved** として表示されます。このオプションでゲストにフィルターをかける場合は、**--all** または **--inactive** のいずれかのオプションを使用する必要があることに注意してください。
- **--name** - このオプションを使用すると、デフォルトのテーブル形式ではなく、ゲスト名を一覧表示します。このオプションは **--uuid** オプションとは一緒に使用できません。后者ではゲストの UUID のみが一覧表示され、**--table** オプションと使用するとテーブル形式の出力になります。
- **--title** - ゲストの **title** フィールドも一覧表示します。これには通常、ゲストについての説明が含まれています。このオプションは、デフォルトのフォーマット (**--table**) で使用する必要があります。例を示します。

```
$ virsh list --title
```

Id	Name	State
Title		
-----		
0	Domain-0	running
	Mailserver1	
2	rhelvm	paused

- **--persistent** - 永続的なゲストのみが一覧表示されます。一時的なゲストを表示するには、**--transient** を使用します。
- **--with-managed-save - managed save** で設定されたゲストを一覧表示します。これを使用せずに設定されたゲストを一覧表示するには、コマンド **--without-managed-save** オプションを使用します。
- **--state-running** - 実行中のゲストを一覧表示します。一時停止のゲストについては **--state-paused**、オフにされているゲストについては **--state-shutoff** を使用します。**--state-other** は、フォールバックとしてすべての状態を一覧表示します。
- **--autostart** - 自動起動のゲストのみを一覧表示します。この機能が無効になっているゲストを一覧表示するには、**--no-autostart** を使用します。

- **--with-snapshot** - スナップショットイメージを一覧表示できるゲストを一覧表示します。スナップショットなしのゲストに絞り込むには、**--without-snapshot** オプションを使用します。

### 21.38.2. 仮想 CPU 情報の表示

**virsh** を使ってゲスト仮想マシンから仮想 CPU 情報を表示するには、以下を実行します。

```
# virsh vcpuinfo {domain-id, domain-name or domain-uuid}
```

**virsh vcpuinfo** の出力例:

```
# virsh vcpuinfo guest1
VCPU:          0
CPU:           2
State:         running
CPU time:      7152.4s
CPU Affinity:  yyyy

VCPU:          1
CPU:           2
State:         running
CPU time:      10889.1s
CPU Affinity:  yyyy
```

### 21.38.3. ホスト物理マシンの CPU への仮想 CPU ピニング

**virsh vcpupin** コマンドは、仮想 CPU を物理 CPU に割り当てます。

```
# virsh vcpupin guest1
VCPU: CPU Affinity
-----
0: 0-3
1: 0-3
```

**vcpupin** コマンドは、以下の引数を取ります。

- **--vcpu** には仮想 CPU (**vcpu**) 番号が必要です。
- **[--cpulist]** **string** は、設定するホスト物理マシンの CPU 番号を一覧表示するか、またはクエリーするオプションを省略します。
- **--config** は次回の起動に影響を与えます。
- **--live** - 実行中のゲスト仮想マシンに影響を与えます。
- **--current** - 現在のゲスト仮想マシンに影響を与えます。

### 21.38.4. 所定ドメインの仮想 CPU 数についての情報の表示

**virsh vcpucount** コマンドでは、**domain**名またはドメイン ID が必要です。

```
# virsh vcpucount guest1
```

maximum	config	2
maximum	live	2
current	config	2
current	live	2

**vcpucount** は、以下の引数を取ることができます。

- **--maximum** は、仮想 CPU (vcpu) の上限 (cap) を取得します。
- **--active** は、現在アクティブな仮想 CPU (vcpu) の数を取得します。
- **--live** - 実行中のゲスト仮想マシンの値を取得します。
- **--config** は、次回起動時に使用される値を取得します。
- **--current** は、現在のゲスト仮想マシンの状態に基づいて値を取得します。
- **--guest** カウントは、ゲスト側から返されるカウントです。

### 21.38.5. 仮想 CPU アフィニティーの設定

物理 CPU に対する仮想 CPU アフィニティーを設定するには、以下のコマンドを実行します。

```
# virsh vcpupin domain-id vcpu cpulist
```

**domain-id** パラメーターは、ゲスト仮想マシンの ID 番号または名前です。

**vcpu** パラメーターにはゲスト仮想マシンに割り当てられている仮想 CPU の番号を指定します。**vcpu** パラメーターを指定する必要があります。

**cpulist** パラメーターは物理 CPU の識別子番号をコンマで区切った一覧です。**cpulist** パラメーターによって仮想 CPU を実行できる物理 CPU が決定されます。

**--config** などの追加のパラメーターは次の起動に影響を与えますが、**--live** は稼働中のゲスト仮想マシンに、**--current** は現在のゲスト仮想マシンに影響を与えます。

### 21.38.6. 仮想 CPU 数の設定

このコマンドを使用して、ゲスト仮想マシン内でアクティブな仮想 CPU の数を変更します。デフォルトでは、このコマンドはアクティブなゲスト仮想マシンで機能します。ゲスト仮想マシンの次回起動時に使用する非アクティブな設定を変更するには、**--config** フラグを使用します。**virsh** を使用してゲスト仮想マシンに割り当てられた CPU の数を変更するには、以下のようになります。

```
# virsh setvcpus {domain-name, domain-id or domain-uuid} count [--config]
[--live] | [--current]] [--maximum] [--guest]
```

例を以下に示します。

```
# virsh setvcpus guestVM1 2 --live
```

上記コマンドでは、vCPU を **guestVM1** に割り当てる数を 2 つに設定します。このアクションは、**guestVM1** の実行中に行われます。





## 重要

vCPU のホットアンプラグは現在 Red Hat Enterprise Linux 7 ではサポートされていません。

値は、ホスト、ハイパーバイザー、またはゲスト仮想マシンの元の記述による制限によって制限される可能性があります。Xen の場合、ゲスト仮想マシンが準仮想化されている場合は稼働中のゲスト仮想マシンの仮想 CPU のみを調整できます。

**--config** フラグが指定されている場合、変更はゲスト仮想マシンの保存された XML 設定に対して行われ、ゲストの次回起動時にのみ有効になります。

**--live** が指定されている場合、ゲスト仮想マシンはアクティブであるはずであり、変更はただちに有効になります。このオプションにより、vCPU のホットプラグが可能になります。**--config** フラグと **--live** フラグの両方は、ハイパーバイザーにサポートされている場合、一緒に指定することができます。

**--current** が指定されている場合、フラグが現在のゲスト仮想マシンに影響を与えます。

フラグが指定されている場合、**--live** フラグが想定されます。ゲスト仮想マシンがアクティブではない場合にコマンドは失敗します。さらに、フラグが指定されていない場合、**--config** フラグも想定されるかどうかはハイパーバイザーによって異なります。これにより、XML 設定が変更を永続化するために調整されるかどうかが決まります。

**--maximum** フラグは、ゲスト仮想マシンの次回起動時にホットプラグできる仮想 CPU の最大数を制御します。そのため、このフラグは **--config** フラグとのみ使用する必要があり、**--live** フラグと一緒に使用することはできません。

**--count** 値は、ゲスト仮想マシンに割り当てられた CPU 数よりも多くすることはできないことに留意してください。

**--guest** が指定される場合、フラグは現在のゲスト仮想マシンの CPU 状態を変更します。

### 21.38.7. メモリ一割り当ての設定

virsh を使ってゲスト仮想マシンのメモリ一割り当てを変更するには、以下を実行します。

```
# virsh setmem {domain-id or domain-name} count
```

例を以下に示します。

```
# virsh setmem vr-rhel6u1-x86_64-kvm --kilobytes 1025000
```

**count** はキロバイトで指定する必要があります。新規のカウント値はゲスト仮想マシンに指定した数を超過することはできません。64 MB 未満の値は、ほとんどのゲスト仮想マシンのオペレーティングシステムでは機能しない可能性があります。それ以上の値が最大メモリー数値である場合は、アクティブなゲスト仮想マシンに影響しません。新規の値が利用可能なメモリーを下回る場合、縮小されてゲスト仮想マシンがクラッシュする原因になるかもしれません。

このコマンドには次のようなオプションがあります。

- **[--domain]** *string* {ドメイン名、id または uuid}
- **[--size]** {number} 新たに設定するメモリーサイズ、整数で指定 (デフォルトは KiB)

有効なメモリー単位には以下が含まれます。

- バイト、**b** または **bytes**
- キロバイト、**KB** ( $10^3$  または 1,000 バイトのブロック)
- キビバイト、**k** または **KiB** ( $2^{10}$  または 1024 バイトのブロック)
- メガバイト、**MB** ( $10^6$  または 1,000,000 バイトのブロック)
- メビバイト、**M** または **MiB** ( $2^{20}$  または 1,048,576 バイトのブロック)
- ギガバイト、**GB** ( $10^9$  または 1,000,000,000 バイトのブロック)
- ギビバイト、**G** または **GiB** ( $2^{30}$  または 1,073,741,824 バイトのブロック)
- テラバイト、**TB** ( $10^{12}$  または 1,000,000,000,000 バイトのブロック)
- テビバイト、**T** または **TiB** ( $2^{40}$  または 1,099,511,627,776 バイトのブロック)

libvirt により値はすべて直近のキビバイトに切り上げられ、またハイパーバイザーで対応できる単位までさらに切り上げられる場合がありますので注意してください。また、ハイパーバイザーの中には、4000KiB (または  $4000 \times 2^{10}$  または 4,096,000 バイト) などの最小値を強制するものがあります。この値の単位は **memory unit** というオプションの属性で確定されます。この属性では、測定単位がキビバイト (KiB) にデフォルト設定されます ( $2^{10}$  または 1024 バイトブロック単位)。

- **--config** は次回起動時に有効になります。
- **--live** は実行中のゲスト仮想マシンのメモリーを制御します。
- **--current** は、現在のゲスト仮想マシン上のメモリーを制御します。

### 21.38.8. ドメインのメモリー割り当ての変更

**virsh setmaxmem domain size --config --live --current** コマンドは、以下のようにゲスト仮想マシンの最大メモリー割り当ての設定を許可します。

```
# virsh setmaxmem guest1 1024 --current
```

最大メモリーに指定できるサイズは、サポートされるサフィックスが指定されない限り、キビバイト単位でデフォルト設定される単位付き整数になります。以下の引数をこのコマンドと共に使用できます。

- **--config** - 次回起動時に有効になります。
- **--live** - すべてのハイパーバイザーがメモリーの上限のライブ変更を許可する訳ではないため、ハイパーバイザーがこのアクションをサポートする場合に、実行中のゲスト仮想マシンのメモリーを制御します。
- **--current** - 現在のゲスト仮想マシン上のメモリーを制御します。

### 21.38.9. ゲスト仮想マシンのブロックデバイス情報の表示

**virsh domblkstat** コマンドを使用すると、実行中のゲスト仮想マシンについてのブロックデバイス統計情報を表示できます。より分かりやすい表示にするには、**--human** オプションを使用します。

```
# virsh domblkstat GuestName block-device
```

### 21.38.10. ゲスト仮想マシンのネットワークデバイス情報の表示

**virsh domifstat** コマンドを使用して、実行中のゲスト仮想マシンについてのネットワークインターフェース統計を表示できます。

```
# virsh domifstat GuestName interface-device
```

## 21.39. 仮想ネットワークの管理

このセクションでは、**virsh** コマンドで仮想ネットワークを管理する方法について説明します。仮想ネットワークの一覧を表示するには、以下を実行します。

```
# virsh net-list
```

このコマンドを実行すると以下のような出力が生成されます。

```
# virsh net-list
Name                      State      Autostart
-----
default                   active     yes
vnet1                     active     yes
vnet2                     active     yes
```

特定の仮想ネットワークのネットワーク情報を表示するには、以下を実行します。

```
# virsh net-dumpxml NetworkName
```

指定した仮想ネットワークに関する情報が XML 形式で表示されます。

```
# virsh net-dumpxml vnet1
<network>
  <name>vnet1</name>
  <uuid>98361b46-1581-acb7-1643-85a412626e70</uuid>
  <forward dev='eth0' />
  <bridge name='vnet0' stp='on' forwardDelay='0' />
  <ip address='192.168.100.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.100.128' end='192.168.100.254' />
    </dhcp>
  </ip>
</network>
```

仮想ネットワークの管理で使用される他の **virsh** コマンドを以下に示します。

- **virsh net-autostart network-name** : **libvirt** デーモンの起動時に、*network-name* が自動的に起動するようマークします。 **--disable** オプションで *network-name* のマークを解除します。

- **virsh net-create XMLfile** : 既存ファイルの XML 定義を使用して、新規の (一時的) ネットワークを開始します。
- **virsh net-define XMLfile** : 既存ファイルの XML 定義を使用して新規のネットワークを定義しますが、これを起動しません。
- **virsh net-destroy network-name** : *network-name* で指定されたネットワークを破棄します。
- **virsh net-name networkUUID** : 指定した *networkUUID* をネットワーク名に変換します。
- **virsh net-uuid network-name** : 指定した *network-name* をネットワーク UUID に変換します。
- **virsh net-start nameOfInactiveNetwork** : 非アクティブのネットワークを起動します。
- **virsh net-undefine nameOfInactiveNetwork** : ネットワークの非アクティブな XML 定義を削除します。これはネットワークの状態には影響を与えません。コマンド実行時にドメインが稼働していれば、ネットワークは稼働し続けます。ただし、ネットワークは永続的ではなく、一時的なものになります。

libvirt には、ドメインによって使用され、実際のネットワークデバイスにリンクされる仮想ネットワークを定義する機能があります。この機能についての詳細は、[libvirt のアップストリームの web サイト](#)にあるドキュメントを参照してください。仮想ネットワークのコマンドの多くはドメインに使用されるコマンドと似ていますが、仮想ネットワークに名前を付ける方法は、その名前自体または UUID のいずれかを使用する方法になります。

### 21.39.1. 仮想ネットワークの自動起動

**virsh net-autostart** コマンドは、ゲスト仮想マシンの起動時に自動的に起動される仮想ネットワークを設定します。

```
# virsh net-autostart network [--disable]
```

このコマンドは、**autostart** コマンドを無効にする **--disable** オプションを受け入れます。

### 21.39.2. XML ファイルによる仮想ネットワークの作成

**virsh net-create** コマンドは、XML ファイルから仮想ネットワークを作成します。libvirt で使用する XML ネットワーク形式の説明については、[libvirt のアップストリームの web サイト](#)を参照してください。このコマンドで、*file* は XML ファイルのパスになります。XML ファイルから仮想ネットワークを作成するには、以下を実行します。

```
# virsh net-create file
```

### 21.39.3. XML ファイルによる仮想ネットワークの定義

**virsh net-define** コマンドは、XML ファイルから仮想ネットワークを定義します。ネットワークは定義されますが、インスタンス化は行われません。

```
# virsh net-define file
```

#### 21.39.4. 仮想ネットワークの停止

**virsh net-destroy** コマンドは、名前または UUID で指定した仮想ネットワークを破棄 (停止) します。これはすぐに有効になります。指定されたネットワークを停止するには、**network**が必要になります。

```
# virsh net-destroy network
```

#### 21.39.5. ダンプファイルの作成

**virsh net-dumpxml** コマンドは、指定された仮想ネットワークの情報を標準出力に XML ダンプとして出力します。--**inactive** が指定されている場合、物理機能は関連付けられた仮想機能に拡張されません。

```
# virsh net-dumpxml network [--inactive]
```

#### 21.39.6. 仮想ネットワークの XML 設定ファイルの編集

以下のコマンドは、ネットワークの XML 設定ファイルを編集します。

```
# virsh net-edit network
```

XML ファイルの編集に使用するエディターは **\$VISUAL** または **\$EDITOR** 環境変数で指定でき、デフォルトは **vi** になります。

#### 21.39.7. 仮想ネットワークについての情報の取得

**virsh net-info** は、**network** オブジェクトの基本情報を返します。

```
# virsh net-info network
```

#### 21.39.8. 仮想ネットワークについての情報の一覧表示

**virsh net-list** コマンドは、アクティブなネットワークの一覧を返します。--**all** を指定する場合は、定義済みであるものの、アクティブではないネットワークも含まれます。--**inactive** が指定される場合、アクティブではないネットワークのみが一覧表示されます。さらに、返されたネットワークを絞り込む場合は、永続的なネットワークを一覧表示するには --**persistent** を、一時的なネットワークを一覧表示するには --**transient** を、自動起動が有効なネットワークを一覧表示するには --**autostart** を使用し、さらに自動起動が無効なネットワークを一覧表示するには --**no-autostart** を使用して実行できます。

注記: 古いサーバーと通信する際、このコマンドには、固有の競合が関係する一連の API 呼び出しの使用が強制されます。これにより、一覧が収集されている間に複数の呼び出し間の状態が変更される場合、プールは一覧表示されなくなるか、または複数回表示される可能性があります。ただし、新しいサーバーの場合は、この問題は発生しません。

仮想ネットワークを一覧表示するには、以下を実行します。

```
# virsh net-list [--inactive | --all] [--persistent] [<--transient>] [--autostart] [<--no-autostart>]
```

### 21.39.9. ネットワーク **UUID** からネットワーク名への変換

**virsh net-name** コマンドは、ネットワーク **UUID** をネットワーク名に変換します。

```
# virsh net-name network-UUID
```

### 21.39.10. ネットワーク名からネットワーク **UUID** への変換

**virsh net-uuid** コマンドは、ネットワーク名をネットワーク **UUID** に変換します。

```
# virsh net-uuid network-name
```

### 21.39.11. 非アクティブの（定義済み）ネットワークの起動

**virsh net-start** コマンドは、(以前に定義された) 非アクティブなネットワークを開始します。

```
# virsh net-start network
```

### 21.39.12. 停止状態のネットワークの設定の定義解除

**virsh net-undefine** コマンドは、非アクティブなネットワーク設定の定義を解除します。

```
# virsh net-undefine network
```

### 21.39.13. 既存のネットワーク定義ファイルの更新

```
# virsh net-update network directive section XML [--parent-index index]
[ [--live] [--config] | [--current] ]
```

**virsh net-update** コマンドは、以下のいずれかの **directives** を既存ネットワーク定義の指定されたセクションに発行することで、これを更新します。

- **add-first**
- **add-last** または **add** (これらは同義)
- **delete**
- **modify**

**section** は以下のいずれかになります。

- **bridge**
- **domain**
- **ip**
- **ip-dhcp-host**
- **ip-dhcp-range**

- **forward**
- **forward interface**
- **forward-pf**
- **portgroup**
- **dns-host**
- **dns-txt**
- **dns-srv**

各セクションは、変更される要素につながる XML 要素の階層を連結したもので命名されています。たとえば、**ip-dhcp-host** は、ネットワークの **<ip>** 要素内の **<dhcp>** 要素に格納されている **<host>** 要素を変更します。

XML は、変更される完全 XML 要素のタイプのテキスト (たとえば、**<host mac="00:11:22:33:44:55' ip='1.2.3.4'/>**) か、完全 XML 要素を格納しているファイル名になります。提供されたテキストの最初の文字が **<** であれば XML テキストであり、最初の文字が **<** でなければ、使用する XML テキストを格納しているファイル名になります。 **--parent-index** オプションを使うと、要求される要素が格納されている親要素を指定することができます (0 ベース)。

たとえば、**dhcp <host>** 要素は、ネットワーク内の複数の **<ip>** 要素に格納されることが可能です。親インデックスが提供されない場合、最適な **<ip>** 要素 (通常は、**<dhcp>** 要素を既に持っているもの) が選択されますが、 **--parent-index** が提供されている場合は、特定の **<ip>** インスタンスが修正されます。 **--live** を指定すると、実行中のネットワークに適用されます。 **--config** を指定すると、永続的ネットワークの次回起動時に適用されます。 **--current** を指定すると、現在のネットワーク状態に適用されます。 **--live** と **--config** フラグは同時に使用することができますが、 **--current** とはできません。フラグを何も指定しないと、 **--current** を指定した場合と同じになります。

#### 21.39.14. virsh を使用したゲスト仮想マシンの移行

virsh を使用した移行についての情報は、「[virsh を使用した KVM のライブマイグレーション](#)」というタイトルのセクションに記載されています。「[virsh を使用した KVM のライブマイグレーション](#)」を参照してください。

#### 21.39.15. ゲスト仮想マシンの静的 IP アドレスの設定

ゲスト仮想マシンが DHCP から IP アドレスを取得するように設定されているものの、それを予測可能な静的 IP アドレスを持つように設定する必要がある場合、以下の手順を使用して libvirt で使用される DHCP サーバー設定を変更することができます。この手順では、この変更を行うためにゲストインターフェースの MAC アドレスを把握しておく必要があります。そのため、この操作をゲストの作成後に実行するか、または作成前にゲストの MAC アドレスを決定して、ゲスト仮想マシンの作成時にこの同じアドレスを手動で設定する必要があります。

さらに、この手順は libvirt 仮想ネットワークに接続されているゲストインターフェースで、転送モードの **"nat"**、**"route"** が設定されているか、または転送モードが設定されていない場合にのみ機能することに注意する必要があります。この手順は、ネットワークが **forward mode="bridge"** または **"hostdev"** で設定されている場合には機能しません。その場合、DHCP サーバーはネットワークの別の場所に置かれているため、libvirt の制御下にはありません。この場合、静的 IP エントリをリモート DHCP サーバーに作成する必要があります。これを実行するには、サーバーと共に提供されるドキュメントを参照してください。

## 手順21.5 静的 IP アドレスの設定

この手順はホスト物理マシンで実行されます。

### 1. ゲスト XML 設定ファイルをチェックします。

**virsh domiflist guest1** コマンドを実行してゲストのネットワーク設定を表示します。**guest1** を仮想マシンの名前で置き換えます。表が表示されるので、「**Source**」列を確認します。これはネットワークの名前です。この例では、ネットワークは **default** になります。この名前は **MAC** アドレスと共に残りの手順で使用されます。

```
# virsh domiflist guest1
Interface  Type          Source      Model      MAC
-----
vnet4      network      default     virtio     52:54:00:48:27:1D
```

### 2. DHCP 範囲の確認

設定する IP アドレスは、ネットワークに指定される **dhcp** 範囲内にある必要があります。さらに、ネットワーク上のその他の既存の静的 IP アドレスとの競合を避ける必要があります。使用されるアドレスと利用可能なアドレスの範囲を確認するには、ホストマシンで以下のコマンドを実行します。

```
# virsh net-dumpxml default | egrep 'range|host\ mac'

<range start='198.51.100.2' end='198.51.100.254' />
<host mac='52:54:00:48:27:1C:1D' ip='198.51.100.2' />
```

表示される出力はサンプルとは異なり、より多くの行と複数のホスト **MAC** 行が表示されます。各ゲストの静的 IP アドレスには1つの行があります。

### 3. 静的 IP アドレスの設定

ホストマシンで以下のコマンドを使用し、**default** をネットワーク名で置き換えます。

```
# virsh net-update default add ip-dhcp-host '<host
mac='52:54:00:48:27:1D' ip='198.51.100.3' />' --live --config
```

**--live** オプションを使用すると、この変更がすぐに有効になり、**--config** オプションは変更を永続化します。また、このコマンドは、有効な IP および **MAC** アドレスを使用していれば、まだ作成されていないゲスト仮想マシンでも機能します。**MAC** アドレスは有効なユニキャスト **MAC** アドレス (最初の偶数の数字のペアと : で区切られる 16 進数の数値のペア) になります。**libvirt** が新規のランダム **MAC** アドレスを作成する場合、最初の 3 つの数字ペアに **52:54:00** を使用するので、この規則に準拠することが推奨されます。

### 4. インターフェースの再起動 (オプション)

ゲスト仮想マシンが現在実行中の場合に、ゲスト仮想マシンが **DHCP** アドレスを再要求することを強制する必要があります。ゲストが実行されていない場合、新規 IP アドレスが次の起動時に実施されます。インターフェースを再起動するには、以下のコマンドをホストマシンのターミナルに入力します。

```
# virsh domif-setlink guest1 52:54:00:48:27:1D down
# sleep 10
# virsh domif-setlink guest1 52:54:00:48:27:1D up
```

このコマンドにより、ゲスト仮想マシンのオペレーティングシステムは、イーサネットケーブルがアンプラグされ、10 秒後に再度プラグされたと認識します。**sleep** コマンドは、多くの



DHCP クライアントが IP アドレスを再要求せずにケーブルの短い切断を可能にするので重要です。**up** コマンドが実行されると、10 秒程度で DHCP クライアントは古い IP アドレスを破棄し、新規アドレスを要求します。ある理由でこのコマンドが失敗する場合、ゲストオペレーティングシステムの管理インターフェースからゲストのインターフェースをリセットする必要があります。

## 21.40. インターフェースコマンド

以下のコマンドはホストのインターフェースを操作するため、ゲスト仮想マシンから実行することはできません。これらのコマンドは、ホスト物理マシンのターミナルから実行する必要があります。



### 警告

マシンの **NetworkManager** サービスが無効にされており、**network** サービスが代わりに使用されている場合は、このセクションに記載のコマンドのみがサポートされます。

これらのホストインターフェースは、ゲスト仮想マシンの **<interface>** 要素内の名前 (**system-created bridge interface** など) で使用されることがよくありますが、ホストインターフェースを特定のゲスト設定 XML に関連付けるようにとの要件はまったくありません。ホストインターフェースのコマンドの多くは、ゲスト仮想マシンに使用されるものに類似しており、インターフェースに名前を付ける方法は、その名前を使用するか、またはその **MAC** アドレスを使用するかのいずれかになります。ただし、**iface** 引数の **MAC** アドレスの使用は、アドレスが固有な場合にのみ機能します (よくあることとして、インターフェースとブリッジが同じ **MAC** アドレスを共有する場合、**MAC** アドレスを使用すると、曖昧さが原因でエラーが生じるため、代わりに名前を使用する必要があります)。

### 21.40.1. XML ファイルによるホスト物理マシンインターフェースの定義と起動

**virsh iface-define file** コマンドは、XML ファイルからホストインターフェースを定義します。このコマンドはインターフェースのみを定義し、これを起動することはありません。

```
# virsh iface-define iface.xml
```

すでに定義されたインターフェースを起動するには、**iface-start interface** を実行します。ここで、**interface** はインターフェース名です。

### 21.40.2. ホストインターフェース用 XML 設定ファイルの編集

**virsh iface-edit interface** コマンドは、ホストインターフェースの XML 設定ファイルを編集します。これは、XML 設定ファイルを編集するための唯一 推奨される方法です (これらのファイルについては、[24章 ドメイン XML の操作](#)を参照してください)。

### 21.40.3. ホストインターフェースの一覧表示

**virsh iface-list** は、アクティブなホストインターフェースの一覧を表示します。**--all** を指定すると、この一覧には、定義されているものの、アクティブではないインターフェースも含まれます。**--inactive** を指定すると、非アクティブなインターフェースのみが一覧表示されます。

#### 21.40.4. MAC アドレスのインターフェース名への変換

**virsh iface-name interface** コマンドは、MAC アドレスがホストのインターフェース間で固有な場合に、ホストインターフェースの MAC アドレスをインターフェース名に変換します。このコマンドには、インターフェースの MAC アドレスである *interface* が必要になります。

**virsh iface-mac interface** コマンドは、ホストのインターフェース名を MAC アドレスに変換します。ここでの *interface* は、インターフェース名になります。

#### 21.40.5. 特定ホスト物理マシンのインターフェースの停止と定義解除

**virsh iface-destroy interface** コマンドは、指定したホストインターフェースを破棄 (停止) します。これは、ホストで **virsh if-down** を実行する場合と同じことになります。このコマンドは、インターフェースのアクティブな使用を無効にし、ただちに有効になります。

インターフェースの定義解除を行うには、インターフェース名を使用して **virsh iface-undefine interface** コマンドを使用します。

#### 21.40.6. ホスト設定ファイルの表示

**virsh iface-dumpxml interface --inactive** コマンドは、標準出力 (stdout) への XML ダンプとしてホストのインターフェースを表示します。--inactive 引数が指定される場合、出力には、次の起動時に使用されるインターフェースの永続状態が反映されます。

#### 21.40.7. ブリッジデバイスの作成

**virsh iface-bridge** コマンドは、*bridge* という名前のブリッジデバイスを作成し、既存のネットワークデバイス *interface* を新規ブリッジに割り当てます。新規のブリッジは、STP が有効な状態、かつ遅延 0 の状態でただちに機能し始めます。

```
# virsh iface-bridge interface bridge
```

これらの設定は、--no-stp オプション、--no-start オプション、および遅延の秒数を使って変更できることに注意してください。インターフェースの IP アドレス設定は、新規のブリッジデバイスに移行します。ブリッジを破棄する方法については、「[ブリッジデバイスの破棄](#)」を参照してください。

#### 21.40.8. ブリッジデバイスの破棄

**virsh iface-unbridge bridge --no-start** コマンドは、*bridge* という名前の指定されたブリッジデバイスを破棄し、その基礎となるインターフェースを通常の使用状態に戻し、ブリッジデバイスから基礎となるデバイスにすべての IP アドレス設定を移行します。この基礎となるインターフェースは、--no-start 引数が使用されない限り再起動しますが、再起動することが通常は推奨されることに注意してください。ブリッジを作成するために使用するコマンドについては、「[ブリッジデバイスの作成](#)」を参照してください。

#### 21.40.9. インターフェーススナップショットの操作

**virsh iface-begin** コマンドは、現在のホストインターフェースの設定のスナップショットを作成し、それは後にコミット (**virsh iface-commit** を使用) されるか、または復元 (**virsh iface-rollback** を使用) されます。新規ホストインターフェイスを定義、起動して、システム構成が正しくないために何らかの不具合が発生した場合などに、これが便利です。スナップショットがすでに存在す

る場合、このコマンドは直前のスナップショットがコミットされるか、または復元されない限り失敗します。外部の変更が `libvirt` API 外のホストインターフェースに対して、スナップショットの作成時からその後のコミットまたはロールバックの間までに行われる場合、未定義の動作が生じます。

`virsh iface-begin` を実行してから行われたすべての変更を機能しているものとして宣言するために、`virsh iface-commit` コマンドを使用し、次にロールバックポイントを削除します。`virsh iface-begin` でインターフェースのスナップショットが起動されていない場合、このコマンドは失敗します。

`virsh iface-rollback` を使用すると、すべてのホストインターフェースの設定が `virsh iface-begin` コマンドを最後に実行した時点に記録した状態に戻されます。`virsh iface-begin` コマンドが事前に実行されていない場合、`virsh iface-rollback` は失敗します。ホスト物理マシンが `virsh iface-commit` の実行前に再起動する場合、自動ロールバックが実行され、ホストの設定が `virsh iface-begin` が最後に実行された状態に回復します。これは、ネットワーク設定への不適切な変更により、変更を元に戻す目的でホストにアクセスできなくなったが、ホストの電源がオンになっているか、または再起動が強制される場合に役立ちます。

### 例21.98 スナップショットの使用例

新規ホストインターフェイスを定義して起動します。

```
# virsh iface-begin
# virsh iface-define eth4-if.xml
# virsh if-start eth4
```

不具合が発生しネットワークが停止したら、変更をロールバックします。

```
# virsh iface-rollback
```

適切に機能していれば、変更をコミットします。

```
# virsh iface-commit
```

## 21.41. スナップショットの管理

以下のセクションでは、ゲスト仮想マシンのスナップショットを操作するために実行できるアクションについて説明します。スナップショットは、指定された時点のゲスト仮想マシンのディスク、メモリー、およびデバイスの状態を取得し、今後の使用に備えて保存します。スナップショットには、OS イメージの「クリーンな」コピーを保存することから、破壊的な操作となりかねない操作を実行する前のゲスト仮想マシンの状態を保存することに至るまで数多くの使用方法があります。スナップショットは固有の名前で識別されます。スナップショットのプロパティを表すために使用される XML 形式のドキュメントについては、[libvirt のアップストリームの web サイト](#) を参照してください。



### 重要

Red Hat Enterprise Linux 7 は、ゲスト仮想マシンが一時停止または電源オフの状態でのスナップショットの作成のみをサポートしています。稼働中のゲストのスナップショット作成 (ライブスナップショットと呼ばれる) は、Red Hat Virtualization で実行できません。詳細については、サービス担当者にご連絡ください。

### 21.41.1. スナップショットの作成

**virsh snapshot-create** コマンドは、ゲスト仮想マシンの XML ファイルで指定されたプロパティでゲスト仮想マシンのスナップショットを作成します (**<name>** および **<description>** 要素、および **<disks>**)。スナップショットを作成するには、以下を実行します。

```
# virsh snapshot-create domain XML file [--redefine [--current] [--no-metadata] [--halt] [--disk-only] [--reuse-external] [--quiesce] [--atomic]
```

ゲスト仮想マシン名、ID、または UID は、ゲスト仮想マシン要件として使用することができます。XML 要件は、少なくとも **name**、**description**、および **disks** 要素を含む文字列です。

残りのオプションの引数は以下のようになります。

- **--disk-only** - ゲスト仮想マシンのメモリー状態はスナップショットに含まれません。
- XML ファイル文字列が完全に省略されると、**libvirt** はすべてのフィールドの値を選択します。新規のスナップショットは **snapshot-current** に記載されるように現行のスナップショットになります。さらに、スナップショットには、ゲスト仮想マシンの状態を含む通常のシステムチェックポイントではなく、ディスクの状態のみが含まれます。ディスクスナップショットは完全なシステムチェックポイントよりも高速ですが、ディスクスナップショットに戻すには、**fsck** またはジャーナルの再生が必要になる場合があります。ディスクスナップショットは電源コードが不意に引き抜かれる場合のようなディスク状態になるためです。**--halt** および **--disk-only** を混在させると、ディスクにフラッシュされていなかったすべてのデータが失われることに注意してください。
- **--halt** - スナップショットの作成後にゲスト仮想マシンを非アクティブな状態のままにします。**--halt** および **--disk-only** を混在させると、その時点でディスクにフラッシュされていないすべてのデータとメモリー状態が失われます。
- **--redefine** は、**virsh snapshot-dumpxml** で生成されるすべての XML 要素が有効な場合、スナップショットを以下の目的でできるように指定します。あるマシンから別のマシンにスナップショットの階層を移行する。一時的なゲスト仮想マシンが途中でなくなり、後で同じ名前および UUID で再作成する際に階層を再作成する。スナップショットのメタデータに若干の変更を加える (例: スナップショットに組み込まれるゲスト仮想マシン XML のホスト固有の部分など)。このフラグが指定されると、**xmlfile** 引数は必須となり、ゲスト仮想マシンの現在のスナップショットは、**--current** フラグも指定されない限り変更されることがあります。
- **--no-metadata** はスナップショットを作成しますが、すべてのメタデータはただちに破棄されます (つまり、**libvirt** はスナップショットを現在の状態で処理せず、**--redefine** が後に使用されて **libvirt** にメタデータについて再度指示しない限り、スナップショットには戻りません)。
- **--reuse-external** を使用して、スナップショット XML が既存ファイルの移行先を含む外部スナップショットを要求する場合、移行先が存在している必要があり、再利用されます。それ以外の場合には、スナップショットは、既存ファイルのコンテンツが失われるのを回避するために拒否されます。
- **--quiesce** が指定されると、**libvirt** はゲストエージェントを使用してゲスト仮想マシンのマウントされたファイルシステムをフリーズおよびフリーズ解除するよう試行します。ただし、ゲスト仮想マシンにゲストエージェントがない場合、スナップショットの作成は失敗します。スナップショットにはゲスト仮想マシンのメモリー状態が格納されます。スナップショットは外部である必要があります。
- **--atomic** を指定すると、スナップショットが成功するか、または変更なしに失敗することを **libvirt** が保証します。すべてのハイパーバイザーがこの機能をサポートしている訳ではないこ



とに注意してください。このフラグが指定されていない場合、一部のスーパーバイザーは部分的にこのアクションを実行した後、失敗する可能性があります。部分的な変更が生じたかどうかを確認するには、**virsh dumpxml** を使用する必要があります。

スナップショットのメタデータがあると、永続的なゲスト仮想マシンの定義解除の試行が回避されます。ただし、**一時的**なゲスト仮想マシンの場合、ゲスト仮想マシンが実行を停止すると、スナップショットのメタデータは通知なしに失われます (**destroy** などのコマンドによるか、または内部ゲストアクションによる)。

### 21.41.2. 現在のゲスト仮想マシンのスナップショットの作成

**virsh snapshot-create-as** コマンドは、ゲスト仮想マシンのスナップショットを、ゲスト仮想マシン XML ファイルで指定されたプロパティを使って作成します (**name** および **description** 要素など)。これらの値が XML 文字列に含まれない場合、**libvirt** が値を選択します。スナップショットを作成するには、以下を実行します。

```
# snapshot-create-as domain [--print-xml] | [--no-metadata] [--halt] [--reuse-external]] [name] [description] [--disk-only] [--quiesce]] [--atomic] [--memspec memspec]] [--diskspec] diskspec]
```

残りのオプションの引数は以下のようになります。

- **--print-xml** は、実際にスナップショットを作成するのではなく、出力として **snapshot-create** の適切な XML を作成します。
- **--halt** は、スナップショットの作成後にゲスト仮想マシンを非アクティブな状態に維持します。
- **--disk-only** は、ゲスト仮想マシンの状態が含まれないスナップショットを作成します。
- **--memspec** を使用してチェックポイントを内部または外部にするかどうかを制御できます。このフラグは必須であり、この後に、**[file=name[, snapshot=type]]** 形式の **memspec** が続きます。ここでタイプは、「none」、「internal」または「external」にすることができます。**file=name** にリテラルコンマを組み込むには、2 番目のコンマでこれをエスケープします。
- **--diskspec** オプションは、**--disk-only** および外部チェックポイントが外部ファイルを作成する方法を制御するために使用できます。このオプションは、ドメイン XML の **<disk>** 要素の番号に基づいて複数回使用される可能性があります。それぞれの **<diskspec>** は **disk[, snapshot=type][, driver=type][, file=name]** 形式で表わされます。特定ディスクで **--diskspec** を省略すると、仮想マシン設定のデフォルトの動作が使用されます。リテラルコンマを **disk** または **file=name** に組み込むには、2 番目のコンマでそれをエスケープします。リテラルの **--diskspec** は、**domain**、**name**、および **description** の 3 つすべてがない限り、それぞれのディスク仕様の前に置かれる必要があります。たとえば、**diskspec** が **vda, snapshot=external, file=/path/to,, new** の場合、以下の XML が生成されます。

```
<disk name='vda' snapshot='external'>
  <source file='/path/to,new' />
</disk>
```



## 重要

Red Hat では、他の仮想化ツールで処理されると柔軟性や信頼性が高くなるため、外部スナップショットの使用を推奨しています。外部スナップショットを作成するには、**virsh-create-as** コマンドで **--diskspec vda, snapshot=external** オプションを使用します。

このオプションを使用しないと **virsh** は内部スナップショットを作成しますが、これは安定性と最適化の面から推奨されません。詳細は、「[libvirt による外部スナップショットの作成方法](#)」を参照してください。

- **--reuse-external** が指定され、ドメイン XML または **diskspec** オプションが既存ファイルの移行先を含む外部スナップショットを要求する場合、移行先が存在している必要があり、再利用されます。そうでない場合には、スナップショットは既存ファイルのコンテンツが失われるのを回避するために拒否されます。
- **--quiesce** が指定されると、**libvirt** はゲストエージェントを使用してゲスト仮想マシンのマウントされたファイルシステムをフリーズおよびフリーズ解除するよう試行します。ただし、ドメインにゲストエージェントがない場合、スナップショットの作成は失敗します。現在、これには **--disk-only** を渡すことも必要です。
- **--no-metadata** はスナップショットデータを作成しますが、すべてのメタデータはただちに破棄されます (つまり、**libvirt** は、**snapshot-create** が後に使用され、**libvirt** に対してメタデータについて再び指示がなされない限り、スナップショットを現状の状態で処理せず、スナップショットには戻りません)。このフラグには **--print-xml** との互換性はありません。
- **--atomic** を指定すると、スナップショットが成功するか、または変更なしに失敗することを **libvirt** が保証します。すべてのハイパーバイザーがこの機能をサポートしている訳ではないことに注意してください。このフラグが指定されていない場合、一部のスーパーバイザーは部分的にこのアクションを実行した後失敗する可能性があります。部分的な変更が生じたかどうかを確認するには、**virsh dumpxml** を使用する必要があります。



## 警告

**64-bit ARM platform** ホストで稼働中の KVM ゲストのスナップショット作成は、現在機能していません。**64-bit ARM** 上の KVM はテクノロジープレビューとして提供されていることに注意してください。

### 21.41.3. 現在使用中のスナップショットの表示

**virsh snapshot-current** コマンドは、現在使用されているスナップショットをクエリーするために使用されます。

```
# virsh snapshot-current domain [--name] | [--security-info] |
[snapshotname]}
```

**snapshotname** が使用されていない場合、ゲスト仮想マシンの現行スナップショットのスナップショット XML (ある場合) は出力として表示されます。**--name** が指定されている場合、完全な XML ではなく、現在のスナップショット名のみが出力として送信されます。**--security-info** が指定され

る場合、セキュリティー上の機密情報が XML に組み込まれます。**snapshotname** を使用する場合、ゲスト仮想マシンに戻さずに、既存の指定されたスナップショットを現在のスナップショットにする要求が生成されます。

#### 21.41.4. snapshot-edit

このコマンドは、現在使用されているスナップショットを編集するために使用されます。

```
# virsh snapshot-edit domain [snapshotname] [--current] [--rename] [--clone]]
```

**snapshotname** と **--current** の両方が指定される場合、編集されたスナップショットを現在のスナップショットにするように強制します。**snapshotname** が省略されている場合、現在のスナップショットを編集するために **--current** を指定する必要があります。

これは、以下のコマンドシーケンスと同等ですが、エラーチェック機能が一部含まれます。

```
# virsh snapshot-dumpxml dom name > snapshot.xml
# vi snapshot.xml [note - this can be any editor]
# virsh snapshot-create dom snapshot.xml --redefine [--current]
```

**--rename** を指定すると、スナップショットの名前が変更されます。**--clone** を指定すると、スナップショット名の変更により、スナップショットのメタデータのクローンが作成されます。いずれも指定されていない場合、編集によりスナップショット名が変更されることはありません。単一の **qcow2** ファイル内の内部スナップショットなど、一部のスナップショットのコンテンツは元のスナップショット名からしかアクセスできないため、スナップショット名は注意して変更する必要があります。

#### 21.41.5. snapshot-info

**snapshot-info domain** コマンドは、スナップショットについての情報を表示します。

```
# snapshot-info domain {snapshot | --current}
```

指定された **snapshot** についての基本的な情報を出力するか、または **--current** で現在のスナップショットを出力します。

#### 21.41.6. snapshot-list

所定のゲスト仮想マシンの利用可能なスナップショットすべてを一覧表示します。デフォルトでは、スナップショット名、作成時およびゲスト仮想マシンの状態の列が表示されます。これを使用するには、以下を実行します。

```
# virsh snapshot-list domain [{--parent | --roots | --tree}] [{[--from]
snapshot | --current} [--descendants]] [--metadata] [--no-metadata] [--leaves]
[--no-leaves] [--inactive] [--active] [--disk-only] [--internal]
[--external]
```

オプションの引数は以下のものです。

- **--parent** は、各スナップショットの親の名前を指定する出力テーブルに列を追加します。このオプションは、**--roots** または **--tree** と共に使用することはできません。

- **--roots** は、親のないスナップショットのみを表示するために一覧をフィルターします。このオプションは、**--parent** または **--tree** と共に使用することはできません。
- **--tree** は、出力をツリー形式で表示し、スナップショット名のみを一覧表示します。このオプションは、**--roots** または **--parent** と共に使用することはできません。
- **--from** は、所定スナップショットの子であるスナップショットの一覧をフィルターします。または、**--current** が指定される場合、一覧から現在のスナップショットで開始させるようにします。一覧が分離してか、または **--parent** と共に使用される場合、**--descendants** も表示されない限り、直接の子に限定されます。**--tree** と共に使用される場合、**--descendants** の使用が暗黙的に示されます。このオプションには、**--roots** との互換性はありません。**--tree** オプションも存在しない限り、**--from** の開始点または **--current** はこの一覧に含まれません。
- **--leaves** が指定されると、一覧は子のないスナップショットのみにフィルターされます。同様に、**--no-leaves** が指定されると、一覧は子を持つスナップショットのみにフィルターされます。(いずれのオプションも省略すると、フィルターは実行されず、両方を指定すると、サーバーがフラグを認識するかどうかによって、同じ一覧が生成されるか、またはエラーが出されます)。フィルターオプションには **--tree** との互換性はありません。
- **--metadata** が指定されると、一覧が **libvirt** メタデータに関連するスナップショットのみにフィルターされるため、永続的なゲスト仮想マシンの定義解除が回避されるか、または一覧が**一時的**なゲスト仮想マシンの破棄によって失われます。同様に、**--no-metadata** が指定される場合、一覧は **libvirt** メタデータのなしに存在するスナップショットのみにフィルターされます。
- **--inactive** が指定されると、一覧はゲスト仮想マシンの停止時に取られたスナップショットにフィルターされます。**--active** が指定される場合、一覧はゲスト仮想マシンの実行時に取られたスナップショットにフィルターされ、このスナップショットには、実行中の状態に戻るためのメモリー状態が含まれます。**--disk-only** が指定される場合、一覧はゲスト仮想マシンの実行中に取られたスナップショットにフィルターされますが、このスナップショットにはディスク状態のみが含まれます。
- **--internal** が指定されると、一覧は既存のディスクイメージの内部ストレージを使用するスナップショットにフィルターされます。**--external** が指定される場合、一覧はディスクイメージの外部ファイルまたはメモリー状態を使用するスナップショットにフィルターされます。

### 21.41.7. snapshot-dumpxml

**virsh snapshot-dumpxml domain snapshot** コマンドは、**snapshot** という名前のゲスト仮想マシンのスナップショットのスナップショット XML を出力します。これを使用するには、以下を実行します。

```
# virsh snapshot-dumpxml domain snapshot [--security-info]
```

また、**--security-info** オプションには、セキュリティ上の機密情報も含まれます。現在のスナップショットの XML に簡単にアクセスするには、**virsh snapshot-current** を使用します。

### 21.41.8. snapshot-parent

所定スナップショット、または **--current** を使って現在のスナップショットの親スナップショット(ある場合)の名前を出力します。これを使用するには、以下を実行します。

```
# virsh snapshot-parent domain {snapshot | --current}
```



## 21.41.9. snapshot-revert

所定ドメインを **snapshot** で指定されるスナップショットか、または **--current** により現在のスナップショットに戻します。



### 警告

これは破壊的なアクションであることに注意してください。最後にスナップショットが取られてから加えられたドメインへの変更は失われます。さらに、**snapshot-revert** の完了後のドメインの状態が元のスナップショットが取られた時点のドメインの状態になることにも注意してください。

スナップショットを元に戻すには、以下を実行します。

```
# virsh snapshot-revert domain {snapshot | --current} [--running | --
  paused]] [--force]
```

通常、スナップショットに戻すと、ドメインはスナップショットが取られた時点の状態に置かれます。ただし、例外として、ゲスト仮想マシンの状態が設定されていないディスクのスナップショットは、ドメインをアクティブでない状態にします。 **--running** または **--paused** オプションのいずれかを渡すことにより、追加の状態変更が実行されます (アクティブでないドメインの起動、または実行中のドメインの一時停止など)。一時的なドメインはアクティブではないため、一時的なドメインのディスクスナップショットに戻る場合に、これらのフラグのいずれかを使用する必要があります。

**snapshot revert** で **--force** の使用が必要になる追加のリスクの伴う場合として 2 つのケースがあります。1 つは、設定を戻すために必要な詳細なドメイン情報がスナップショットにない場合です。この場合、**libvirt** は現在の設定がスナップショットの作成時に使用された設定に一致することを証明できないため、 **--force** を指定することにより、**libvirt** に対し、スナップショットには現在の設定との互換性がある (互換性がない場合はドメインの実行が失敗する可能性がある) ことを保証します。もう 1 つのケースは、実行中のドメインをアクティブな状態に戻す場合で、この場合、既存の **VNC** または **Spice** 接続を中断するなどの障害が暗示されるため、既存のハイパーバイザーを再利用するのではなく、新規のハイパーバイザーを作成する必要があります。この状態は、互換性がない可能性のある設定を使用するアクティブなスナップショットや、 **--start** または **--pause** フラグで組み合わされたアクティブでないスナップショットについて発生します。

## 21.41.10. snapshot-delete

**virsh snapshot-delete domain** コマンドは、指定されたドメインのスナップショットを削除します。これを実行するには、以下を実行します。

```
# virsh snapshot-delete domain {snapshot | --current} [--metadata] [--
  children | --children-only]
```

このコマンドは、**snapshot** という名前のドメインのスナップショットを削除するか、または **--current** で現在のスナップショットを削除します。このスナップショットに子のスナップショットがある場合、このスナップショットの変更は子にマージされます。オプションの **--children** が使用さ

れる場合、このスナップショットとこのスナップショットのすべての子が削除されます。--**children-only** が使用される場合、このスナップショットの子が削除され、このスナップショットは元の状態のままにされます。これらの 2 つのフラグは相互排他的です。

--**metadata** が使用されると、libvirt で維持されるスナップショットのメタデータが削除され、一方でスナップショットのコンテンツは外部ツールがアクセスできるように元の状態にされます。そうしないと、スナップショットの削除により、その時点からのデータのコンテンツも削除されてしまいます。

## 21.42. ゲスト仮想マシンの CPU モデルの設定

### 21.42.1. はじめに

それぞれのハイパーバイザーには、ゲスト仮想マシンにデフォルトで表示する CPU 情報に関して独自のポリシーがあります。ゲスト仮想マシンで利用できるホスト物理マシンの CPU 機能をハイパーバイザー側で決めるものもあれば、QEMU/KVM ではゲスト仮想マシンには **qemu32** または **qemu64** という汎用モデルを表示します。こうしたハイパーバイザーでは、より高度なフィルター機能を備え、すべての物理 CPU をいくつかのグループに分類して、ゲスト仮想マシンに対して表示するベースライン CPU モデルを各グループに 1 つずつ用意します。これにより、同じグループに分類される物理 CPU が使用される場合、ホスト物理マシン間でのゲスト仮想マシンの安全な移行が可能になります。libvirt では一般的にはポリシー自体の強制は行わず、より高い層で適したポリシーを指定するというメカニズムを提供します。CPU のモデル情報を取得し、ゲスト仮想マシンに適した CPU モデルを定義する方法を理解することは、ホスト物理マシン間でゲスト仮想マシンを正常に移行する上で非常に重要となります。ハイパーバイザーは認識できる機能しかエミュレートできないため、そのハイパーバイザーのリリース後に作成された機能についてはエミュレートできません。

### 21.42.2. ホスト物理マシン CPU モデルに関する認識

**virsh capabilities** コマンドは、ハイパーバイザーの接続とホスト物理マシンの機能を記述する XML ドキュメントを表示します。表示される XML スキーマは、ホスト物理マシンの CPU モデルの情報を提供するように拡張されています。CPU モデルの記述において大きな課題となるものの 1 つは、すべてのアーキテクチャがそれぞれの機能の公開に対して異なるアプローチを取っていることです。QEMU/KVM および libvirt は、CPU モデル名の文字列を命名されたフラグー式と合わせるスキームを使用します。

既知の CPU モデルすべてを一覧表示するデータベースを取得することは実際的ではありません。そのため、libvirt はベースライン CPU モデル名の小規模な一覧を維持します。この一覧は、実際のホスト物理マシン CPU と最大数の CPUID ビットを共有するものを選択して、残りの部分は名前付きの機能として一覧表示します。libvirt は、ベースライン CPU に含まれる機能は表示しないことに注意してください。これは、一見すると不具合のように思われる可能性があります。このセクションで後述されるように、この情報について知る必要はありません。

### 21.42.3. VFIO IOMMU デバイスのサポートの判別

**virsh domcapabilities** コマンドを使用して VFIO のサポートを判別します。以下の出力例を参照してください。

```
# virsh domcapabilities

[...output truncated...]

<enum name='pciBackend'>
  <value>default</value>
  <value>vfio</value>

[...output truncated...]
```

図21.3 VFUIのサポートの判別

#### 21.42.4. ホスト物理マシンのプールに適合する互換性のある CPU モデルの決定

1 つのホスト物理マシンが持つ CPU 機能を認識することができるようになったら、次のステップは、どの CPU 機能をそのゲスト仮想マシンに公開するのが最適であるかを決定することです。ゲスト仮想マシンを別のホスト物理マシンに移行する必要が全くないことが判明している場合、そのホスト物理マシンの CPU モデルは修正なしに単純に渡されます。一部の仮想化データセンターには、すべてのサーバーが100%同一のCPUを保持していることを保証する設定のセットが含まれる場合があります。そのような場合も、ホスト物理マシンのCPUモデルは修正なしに単純に渡されます。しかし、さらに一般的なケースとしては、ホスト物理マシン間にCPUのバリエーションが存在することがあります。このようなCPUの混在環境では、妥協できる共通のCPUを決定する必要があります。これは単純な手順ではないため、libvirtはこのタスクに適格なAPIを提供します。libvirtが、それぞれホスト物理マシン用のCPUモデルを記述しているXML文書の一覧を受け取ると、内部でこれらをCPUIDマスクに変換し、それらの接点を算出して、CPUIDマスクの結果をXML CPU記述に変換し直します。

以下は、**virsh capabilities** が実行される際に、libvirt が基本ワークステーションの機能として報告する内容についての例です。

```
<capabilities>
  <host>
    <cpu>
      <arch>i686</arch>
      <model>pentium3</model>
      <topology sockets='1' cores='2' threads='1' />
      <feature name='lahf_lm' />
      <feature name='lm' />
      <feature name='xtpr' />
      <feature name='cx16' />
      <feature name='ssse3' />
      <feature name='tm2' />
      <feature name='est' />
      <feature name='vmx' />
      <feature name='ds_cpl' />
      <feature name='monitor' />
      <feature name='pni' />
      <feature name='pbe' />
      <feature name='tm' />
      <feature name='ht' />
      <feature name='ss' />
      <feature name='sse2' />
      <feature name='acpi' />
      <feature name='ds' />
      <feature name='clflush' />
      <feature name='apic' />
    </cpu>
  </host>
</capabilities>
```

図21.4 ホスト物理マシンの CPU モデル情報のプル

ここで、同じ **virsh capabilities** コマンドを使ってこれを別のサーバーと比較します。

```

<capabilities>
  <host>
    <cpu>
      <arch>x86_64</arch>
      <model>phenom</model>
      <topology sockets='2' cores='4' threads='1' />
      <feature name='osvw' />
      <feature name='3dnowprefetch' />
      <feature name='misalignsse' />
      <feature name='sse4a' />
      <feature name='abm' />
      <feature name='cr8legacy' />
      <feature name='extapic' />
      <feature name='cmp_legacy' />
      <feature name='lahf_lm' />
      <feature name='rdtscp' />
      <feature name='pdpe1gb' />
      <feature name='popcnt' />
      <feature name='cx16' />
      <feature name='ht' />
      <feature name='vme' />
    </cpu>
    ...snip...
  </host>
</capabilities>

```

### 図21.5 ランダムサーバーからの CPU 記述の生成

この CPU 記述に直前のワークステーションの CPU 記述との互換性があるかを判別するには、**virsh cpu-compare** コマンドを使用します。

この縮小されたコンテンツは **virsh-caps-workstation-cpu-only.xml** という名前のファイル内に格納されて、このファイルに **virsh cpu-compare** コマンドを実行することができます。

```

# virsh cpu-compare virsh-caps-workstation-cpu-only.xml
Host physical machine CPU is a superset of CPU described in virsh-caps-
workstation-cpu-only.xml

```

以上の出力から分かるように、サーバー CPU 内のいくつかの機能がクライアント CPU 内に見つからないため、**libvirt** は CPU には厳密には互換性がないことを正しく報告しています。クライアントとサーバー間の移行を可能にするには、XML ファイルを開いていくつかの機能をコメントアウトする必要があります。どの機能を削除するかを決定するには、両方のマシンの CPU 情報が含まれる **both-cpus.xml** で **virsh cpu-baseline** コマンドを実行します。# **virsh cpu-baseline both-cpus.xml** を実行すると、次のような結果になります。

```
<cpu match='exact'>
  <model>pentium3</model>
  <feature policy='require' name='lahf_lm' />
  <feature policy='require' name='lm' />
  <feature policy='require' name='cx16' />
  <feature policy='require' name='monitor' />
  <feature policy='require' name='pni' />
  <feature policy='require' name='ht' />
  <feature policy='require' name='sse2' />
  <feature policy='require' name='clflush' />
  <feature policy='require' name='apic' />
</cpu>
```

図21.6 複合 CPU ベースライン

この複合ファイルは共通の要素を示します。共通でない要素はすべてコメントアウトされます。

### 21.43. ゲスト仮想マシンの CPU モデルの設定

単純なデフォルト設定では、ゲスト仮想マシンの CPU 設定は、XML が表示するホスト物理マシンの機能と同じ基本的な XML 表現を受け入れます。つまり **virsh cpu-baseline** コマンドの XML は、**domain** 要素下で、トップレベルにあるゲスト仮想マシン XML に直接コピーすることができます。以前の XML スニペットには、ゲスト仮想マシンの XML 内に CPU を記述する際に利用できるいくつかの追加の属性があります。これらはほとんど無視できますが、ここでそれらの属性の機能について簡単に説明します。トップレベルの **<cpu>** 要素には **match** という属性があり、以下の値を持つことができます。

- **match='minimum'** - ホスト物理マシン CPU には、少なくともゲスト仮想マシン XML 内に記述されている CPU 機能がなければなりません。ホスト物理マシンにゲスト仮想マシンの設定範囲を超える追加の機能がある場合は、それらはゲスト仮想マシンに対しても表示されます。
- **match='exact'** - ホスト物理マシン CPU には、少なくともゲスト仮想マシン XML 内に記述されている CPU 機能がなければなりません。ホスト物理マシンにゲスト仮想マシンの設定範囲を超える追加の機能がある場合は、それらはゲスト仮想マシンには非表示になります。
- **match='strict'** - ホスト物理マシン CPU には、ゲスト仮想マシン XML 内に記述してあるものと全く同じ CPU 機能がなければなりません。

次の機能拡張により、**<feature>** の各要素に「**policy**」属性を追加で持たせ、以下のような値を設定することができます。

- **policy='force'** - ホスト物理マシンが持たない機能の場合でもゲスト仮想マシンに対してその機能を表示します。これは通常、ソフトウェアエミュレーションの場合にのみ役立ちます。



#### 注記

**force** ポリシーを使用した場合でも、ハイパーバイザーが特定の機能をエミュレートしないこともあります。

- **policy='require'** - ホスト物理マシンが持たない機能の場合でも、ゲスト仮想マシンに対してその機能を表示し、その後失敗します。これは適切なデフォルトと言えます。
- **policy='optional'** - サポートされる機能の場合、その機能をゲスト仮想マシンに表示します。

- **policy='disable'** - ホスト物理マシンが持つ機能の場合、その機能はゲスト仮想マシンには非表示となります。
- **policy='forbid'** - ホスト物理マシンが持つ機能の場合、失敗してゲスト仮想マシンの起動が拒否されます。

「forbid」ポリシーは、CPUID マスク内にない機能の場合でも、不正な動作をしているアプリケーションがその機能の使用を試みており、この機能を持つホスト物理マシン上でゲスト仮想マシンを間違えて実行することを避けたい状況での特定のシナリオに対応します。「optional」ポリシーは、移行に関して特殊な動作をします。ゲスト仮想マシンが最初に起動する際にそのフラグは「optional」ですが、ゲスト仮想マシンのライブマイグレーションの実行時にこのポリシーは「require」に変わります。この理由は移行時に機能が消滅することを防ぐためです。

## 21.44. ゲスト仮想マシンのリソースの管理

**virsh** を使用すると、ゲスト仮想マシンごとにリソースのグループ化および割り当てができます。これは **libvirt** デーモンによって管理され、これは **cgroups** を作成して、ゲスト仮想マシンの代わりにそれらを管理します。システム管理者による作業は、指定された仮想マシンに対して **tunable** (調整可能パラメーター) を照会するか、または設定するかのみになります。**Libvirt** は、仮想マシンのチューニングおよび監視に以下の **cgroups** を使用します。

- **memory** - メモリーコントローラーは RAM と **swap** 使用量の制限を設定することを許可し、グループ内のすべてのプロセスの累積的な使用の照会を許可します。
- **cpuset** - CPU セットコントローラーは、グループ内のプロセスを CPU セットにバインドし、CPU 間の移行を制御します。
- **cpuacct** - CPU アカウンティングコントローラーは、プロセスのグループの CPU 使用量についての情報を提供します。
- **cpu** - CPU スケジューラーコントローラーは、グループ内のプロセスの優先付けを制御します。これは **nice** レベルの特権を付与することに似ています。
- **devices** - デバイスコントローラーは、キャラクターおよびブロックデバイスのアクセス制御リストを付与します。
- **freezer** - フリーザーコントローラーは、グループ内のプロセスの実行を一時停止し、再開します。これはグループ全体に対する **SIGSTOP** に似ています。
- **net\_cls** - ネットワーククラスコントローラーは、プロセスを **tc** ネットワーククラスに関連付けることにより、ネットワークの利用を管理します。

**cgroups** は、**libvirt** 内で **systemd** により設定されます。以下の **virsh** チューニングコマンドが **cgroups** の設定方法に影響します。

- **schedinfo** - 「[スケジュールパラメーターの設定](#)」で説明されています。
- **blkdeviotune** - 「[ディスク I/O スロットリング](#)」で説明されています。
- **blkiotune** - 「[ブロック I/O パラメーターの表示または設定](#)」で説明されています。
- **domiftune** - 「[ネットワークインターフェース帯域幅パラメーターの設定](#)」で説明されています。
- **memtune** - 「[メモリーチューニングの設定](#)」で説明されています。



cgroups についての詳細は、『[Red Hat Enterprise Linux 7 リソース管理ガイド](#)』を参照してください。

## 21.45. スケジュールパラメーターの設定

**virsh schedinfo** コマンドは、ホストマシン上で仮想マシンプロセスのホストスケジュールのパラメーターを変更します。以下の形式になります。

```
# virsh schedinfo domain --set --current --config --live
```

それぞれのパラメーターを以下に説明します。

- **domain** - ゲスト仮想マシンのドメインです。
- **--set** - ここに置かれる文字列は、呼び出されるコントローラーまたはアクションです。文字列は **parameter=value** という形式になります。必要な場合は、追加のパラメーターまたは値も追加してください。
- **--current** - **--set** と共に使用される場合、現在のスケジューラー情報として指定された **set** 文字列を使用します。**--set** なしに使用される場合、現在のスケジューラー情報が表示されます。
- **--config** - **--set** と共に使用される場合、次の起動時に指定された **set** 文字列が使用されます。**--set** なしに使用される場合、設定ファイルに保存されるスケジューラー情報が表示されます。
- **--live** - **--set** と共に使用される場合、現在実行中のゲスト仮想マシン上で指定された **set** 文字列が使用されます。**--set** なしに使用される場合、実行中の仮想マシンで現在使用されている設定が表示されます。

スケジューラーは、**cpu\_shares**、**vcpu\_period** および **vcpu\_quota** のパラメーターのいずれかで設定できます。これらのパラメーターは、vCPU スレッドに適用されます。

cgroup フィールド名へのパラメーターのマッピングは以下のようになります。

- **cpu\_shares**:cpu.shares
- **vcpu\_period**:cpu.cfs\_period\_us
- **vcpu\_quota**:cpu.cfs\_quota\_us

### 例21.99 schedinfo show

以下の例は、シェルのゲスト仮想マシンのスケジュール情報を表示します。

```
# virsh schedinfo shell
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
```

### 例21.100 schedinfo set



この例では、**cpu\_shares** は **2046** に変更されています。これは現在の状態に影響を与えますが、設定ファイルには影響がありません。

```
# virsh schedinfo --set cpu_shares=2046 shell
Scheduler      : posix
cpu_shares     : 2046
vcpu_period    : 1000000
vcpu_quota     : -1
```

**libvirt** は **emulator\_period** と **emulator\_quota** のパラメーターもサポートしています。これらはエミュレータープロセスの設定を変更します。

## 21.46. ディスク I/O スロットリング

**virsh blkdeviotune** コマンドは、指定されたゲスト仮想マシンのディスク I/O スロットリングを設定します。これにより、ゲスト仮想マシンが共有リソースを過剰に使用し、他のゲスト仮想マシンのパフォーマンスに影響が及ぶことが避けられます。以下の形式を使用する必要があります。

```
#virsh blkdeviotune domain <device> [--config] [--live] | [--current]]
[[total-bytes-sec] | [read-bytes-sec] [write-bytes-sec]] [[total-iops-sec]
[read-iops-sec] [write-iops-sec]]
```

必要なパラメーターはゲスト仮想マシンのドメイン名のみです。ドメイン名を一覧表示するには、**virsh domblklist** コマンドを実行します。**--config**、**--live**、および **--current** 引数は、「[スケジュールパラメーターの設定](#)」の場合と同様に機能します。制限が設定されない場合、現在の I/O 制限の設定を照会します。それ以外の場合は、以下のフラグで制限を変更します。

- **--total-bytes-sec** - 1 秒あたりのバイト単位の合計スループット制限を指定します。
- **--read-bytes-sec** - 1 秒あたりのバイト単位の読み込みスループット制限を指定します。
- **--write-bytes-sec** - 1 秒あたりのバイト単位の書き込みスループット制限を指定します。
- **--total-iops-sec** - 1 秒あたりの合計 I/O 操作回数を指定します。
- **--read-iops-sec** - 1 秒あたりの読み込み I/O 操作回数を指定します。
- **--write-iops-sec** - 1 秒あたりの書き込み I/O 操作回数を指定します。

詳細については、**virsh** の man ページの **blkdeviotune** セクションを参照してください。ドメイン XML 例については、[図24.28 「デバイス - ハードドライブ、フロッピーディスク、CD-ROM」](#) を参照してください。

## 21.47. ブロック I/O パラメーターの表示または設定

**blkiotune** コマンドは、指定されたゲスト仮想マシンの I/O パラメーターを設定するか、または表示します。以下の形式を使用する必要があります。

```
# virsh blkiotune domain [--weight weight] [--device-weights device-
weights] [--device-read-iops-sec -device-read-iops-sec] [--device-write-
iops-sec device-write-iops-sec] [--device-read-bytes-sec device-read-
```

```
bytes-sec] [--device-write-bytes-sec device-write-bytes-sec] [--config]
[--live] | [--current]]
```

このコマンドの詳細情報については、『[仮想化のチューニングと最適化ガイド](#)』を参照してください。

## 21.48. メモリーチューニングの設定

**virsh memtune virtual\_machine --parameter size** コマンドは、『[仮想化のチューニングと最適化ガイド](#)』で扱われています。

## 第22章 オフラインツールを使用したゲスト仮想マシンディスクへのアクセス

### 22.1. はじめに

Red Hat Enterprise Linux 7 は、ゲスト仮想マシンのディスクや他のディスクイメージへのアクセス、編集、作成などを実行できる数多くの **libguestfs** ユーティリティーを提供しています。これらのツールは次のような複数の目的で 사용할 ことができます。

- ゲスト仮想マシンのディスクにあるファイルの表示またはダウンロード
- ゲスト仮想マシンのディスクでのファイルの編集またはアップロード
- ゲスト仮想マシン設定の読み込みまたは書き込み
- ファイル、ディレクトリー、ファイルシステム、パーティション、論理ボリュームおよびその他オプションを含む新規ディスクイメージの準備
- 起動に失敗したゲスト仮想マシン、起動設定に変更を必要とするゲスト仮想マシンのレスキューおよび修復
- ゲスト仮想マシンのディスク使用量の監視
- 企業のセキュリティ標準に準拠しているかどうかなど、ゲスト仮想マシンのコンプライアンスの監査
- テンプレートからのクローン作成およびテンプレートの修正による複数ゲスト仮想マシンの導入
- CD、DVD ISO イメージおよびフロッピーディスクイメージの読み込み



#### 警告

本章に記載するユーティリティーを使用して、実行中の仮想マシンに接続しているゲスト仮想マシンまたはディスクイメージに書き込みを行うことが **決してない** ようにしてください。また、こうしたディスクイメージを書き込みモードで開くことがないようにしてください。

これはゲスト仮想マシンのディスクが破損する原因となります。ツールはこの実行を防ぎますが、すべてのケースに対応しない場合もあります。ゲスト仮想マシンが実行中である可能性が少しでもある場合には、**Red Hat** はこれらのユーティリティーを使用しないよう強くお勧めします。

安全度を高めるために、特定のユーティリティーは、変更を保存しない読み取り専用モードで 사용할 ことができます (**--ro** オプションを使用)。



## 注記

**libguestfs** と関連するユーティリティーに関する基礎知識については **Linux** の **man** ページをご覧ください。API については **guestfs(3)** に記載されています。**guestfish** については **guestfish(1)** に記載されています。仮想化ユーティリティーについては独自の **man** ページをご覧ください (**virt-df(1)** など)。トラブルシューティング情報については、「[libguestfs のトラブルシューティング](#)」を参照してください。

### 22.1.1. リモート接続の使用についての注意点

Red Hat Enterprise Linux 7 の一部の仮想化コマンドを使用すると、リモート **libvirt** 接続を指定できません。以下は例になります。

```
# virt-df -c qemu:///remote/system -d Guest
```

ただし、Red Hat Enterprise Linux 7 の **libguestfs** ユーティリティーは、リモート **libvirt** ゲストのディスクにアクセスできず、リモート URL を使用するコマンドは予想通りに機能しません。

ただし、Red Hat Enterprise Linux 7 から開始すると、**libguestfs** はネットワークブロックデバイス (NBD) 上でリモートディスクソースにアクセスできます。ディスクイメージは **qemu-nbd** コマンドを使用してリモートマシンからエクスポートでき、**nbd:// URL** を使用してこれにアクセスできます。以下のようにファイアウォール (ポート 10809) でポートを開く必要がある場合があります。

リモートシステム上: **qemu-nbd -t disk.img**

ローカルシステム上: **virt-df -a nbd:///remote**

以下の **libguestfs** コマンドが影響を受けます。

- **guestfish**
- **guestmount**
- **virt-alignment-scan**
- **virt-cat**
- **virt-copy-in**
- **virt-copy-out**
- **virt-df**
- **virt-edit**
- **virt-filesystems**
- **virt-inspector**
- **virt-ls**
- **virt-rescue**
- **virt-sysprep**
- **virt-tar-in**

- **virt-tar-out**
- **virt-win-reg**

## 22.2. 用語について

このセクションでは、本章で使用されている用語について説明します。

- **libguestfs (GUEST FileSystem LIBrary)** は基礎となる C ライブラリーです。ディスクイメージを開く場合やファイルの読み込みや書き込みを行う場合などに基本的な機能を提供します。この API に対して直接 C プログラムを記述することができます。
- **guestfish (GUEST Filesystem Interactive SHell)** はインタラクティブなシェルです。コマンドラインまたはシェルスクリプトで 사용할 수 있습니다。libguestfs API のすべての機能を公開します。
- libguestfs の上部に各種の virt ツールがビルドされており、これらはコマンドラインから特定の単一タスクを実行するための手段になります。ツールには **virt-df**、**virt-rescue**、**virt-resize** および **virt-edit** などがあります。
- **augeas** は、Linux 設定ファイルを編集するためのライブラリーです。これは libguestfs とは別のライブラリーですが、libguestfs の値の多くはこのツールの組み合わせから来ています。
- **guestmount** は、libguestfs と FUSE をつなぐインターフェースになります。これは、ホスト物理マシン上のディスクイメージからファイルシステムをマウントするために主に使用されます。この機能は必須ではありませんが、便利な機能になります。

## 22.3. インストール

libguestfs、guestfish、libguestfs のツール、guestmount をインストールするには次のコマンドを実行します。

```
# yum install libguestfs libguestfs-tools
```

言語バインディングなどの libguestfs 関連のパッケージをすべてインストールする場合は次のコマンドを入力します。

```
# yum install '*guestf*'
```

## 22.4. GUESTFISH シェル

**guestfish** はインタラクティブシェルで、コマンドラインまたはシェルスクリプトからゲスト仮想マシンのファイルシステムにアクセスするために使用することができます。libguestfs API の機能はすべてシェルで利用することができます。

仮想マシンのディスクイメージを表示または編集する場合は、まず次のコマンドを入力します。以下のパスは必要なディスクイメージに置き換えてください。

```
$ guestfish --ro -a /path/to/disk/image
```

**--ro** は、ディスクイメージを読み取り専用で開くという意味です。このモードは書き込みアクセスを許可しないため、常に安全なモードになります。ゲスト仮想マシンが実行中ではないことが**確実な**場合か、またはライブゲスト仮想マシンにディスクイメージが接続されていない場合以外は、このオプションは省略しないでください。ライブゲスト仮想マシンの編集には **libguestfs** は使用できません。無

理に編集を行おうとするとディスクは破損し、元に戻せなくなります。

**/path/to/disk/image** は、そのディスクへのパスになります。ファイル、ホスト物理マシンの論理ボリューム (**/dev/VG/LV** など)、ホスト物理マシンのデバイス (**/dev/cdrom**)、または **SAN LUN** (**/dev/sdf3**) のいずれかになります。



#### 注記

**libguestfs** と **guestfish** には **root** 権限は必要ありません。アクセスするディスクイメージがその読み取りや書き込みに **root** を必要とする場合にのみ **root** での実行が必要になります。

**guestfish** をインタラクティブに起動すると次のようなプロンプトが表示されます。

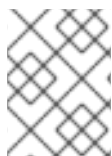
```
$ guestfish --ro -a /path/to/disk/image

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

><fs>
```

プロンプト時に **run** と入力してライブラリーを開始し、ディスクイメージを接続します。これを初めて行う場合には 30 秒ほど時間がかかる場合がありますが、それ以降からの起動は大幅に速くなります。



#### 注記

**libguestfs** は、**KVM** (使用可能な場合) などのハードウェア仮想化加速機能を使ってこのプロセスを加速させます。

**run** コマンドを入力すると、次のセクションで説明されているように他のコマンドを使用できるようになります。

### 22.4.1. **guestfish** を使用したファイルシステムの表示

このセクションでは、**guestfish** でファイルシステムを表示する方法についての情報を提供します。

#### 22.4.1.1. 手動による一覧表示

**list-filesystems** コマンドを使って **libguestfs** によって検出されるファイルシステムを一覧表示します。以下の出力では **Red Hat Enterprise Linux 4** のディスクイメージを表示しています。

```
><fs> run
><fs> list-filesystems
/dev/vda1: ext3
/dev/VolGroup00/LogVol00: ext3
/dev/VolGroup00/LogVol01: swap
```

他にも便利なコマンドには、**list-devices**、**list-partitions**、**lvs**、**pvs**、**vfs-type**、**file** などがあります。以下のように **help** コマンドを入力すると、各コマンドの詳細情報を取得できます。

```
><fs> help vfs-type
NAME
    vfs-type - get the Linux VFS type corresponding to a mounted device

SYNOPSIS
    vfs-type mountable

DESCRIPTION
    This command gets the filesystem type corresponding to the filesystem
on
    "device".

    For most filesystems, the result is the name of the Linux VFS module
    which would be used to mount this filesystem if you mounted it without
    specifying the filesystem type. For example a string such as "ext3" or
    "ntfs".
```

ファイルシステムの実際の内容を表示するには、まずファイルシステムをマウントする必要があります。

ファイルやディレクトリを表示してダウンロードする場合は、**ls**、**ll**、**cat**、**more**、**download**、**tar-out**などの**guestfish**コマンドを使用することができます。



### 注記

このシェルには現行の作業ディレクトリという概念がありません。たとえば、普通のシェルとは異なり、**cd**などのコマンドを使ってディレクトリを変更することはできません。パスはすべてスラッシュ (/) を先頭に付けた完全修飾パスにしなければなりません。**Tab** キーのパスの補完機能を使用することができます。

**guestfish** シェルを終了するには、**exit** と入力するか、または **Ctrl+d** を押します。

#### 22.4.1.2. guestfish による検出

手作業でファイルシステムを表示し、マウントする代わりに、**guestfish** 自体にイメージを検出させて、そのファイルシステムをゲスト仮想マシンでのマウントと同様にマウントさせることができます。これを実行するには、コマンドラインに **-i** オプションを追加します。

```
$ guestfish --ro -a /path/to/disk/image -i

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

Operating system: Red Hat Enterprise Linux AS release 4 (Nahant Update 8)
/dev/VolGroup00/LogVol00 mounted on /
/dev/vda1 mounted on /boot

><fs> ll /
total 210
```

```
drwxr-xr-x. 24 root root 4096 Oct 28 09:09 .
drwxr-xr-x 21 root root 4096 Nov 17 15:10 ..
drwxr-xr-x. 2 root root 4096 Oct 27 22:37 bin
drwxr-xr-x. 4 root root 1024 Oct 27 21:52 boot
drwxr-xr-x. 4 root root 4096 Oct 27 21:21 dev
drwxr-xr-x. 86 root root 12288 Oct 28 09:09 etc
...
```

`guestfish` が検出とマウントを実行するには `libguestfs` バックエンドを起動する必要があるため、`-i` オプションを使用する場合は `run` コマンドは必要ありません。`-i` オプションはよく使用される Linux のゲスト仮想マシンの多くで正常に動作します。

#### 22.4.1.3. ゲスト仮想マシンへの名前別のアクセス

`libvirt` が認識できる名前を指定すると、コマンドラインからゲスト仮想マシンにアクセスすることができます (つまり、`virsh list --all` で表示される名前)。ゲスト仮想マシンの名前でそのゲスト仮想マシンにアクセスするには `-d` オプションを使用します。`-i` オプションは付けても付けなくても構いません。

```
$ guestfish --ro -d GuestName -i
```

#### 22.4.2. `guestfish` を使用したファイルの追加

`guestfish` を使ってファイルを追加するには、完全な URI が必要です。ファイルはローカルファイルであるか、またはファイルをネットワークブロックデバイス (NBD) またはリモートブロックデバイス (RBD) にある可能性があります。

URI に使用される形式は、以下の例のいずれかになるはずです。ファイルがローカルの場合は `///` を使用します。

- `guestfish -a disk.img`
- `guestfish -a file:///directory/disk.img`
- `guestfish -a nbd://example.com[:port]`
- `guestfish -a nbd://example.com[:port]/exportname`
- `guestfish -a nbd:///socket=/socket`
- `guestfish -a nbd:///exportname?socket=/socket`
- `guestfish -a rbd:///pool/disk`
- `guestfish -a rbd://example.com[:port]/pool/disk`

#### 22.4.3. `guestfish` を使用したファイルの編集

ファイルの変更、ディレクトリーの作成、またはゲスト仮想マシンへのその他の変更を実行する場合には、このセクションの先頭にある「ゲスト仮想マシンをシャットダウンしておくこと」という警告にまず注意してください。`guestfish` で実行中のディスクに編集や変更を加えたりすると、**ディスクの破損を招きます**。このセクションでは、`/boot/grub/grub.conf` ファイルの編集例を示します。ゲスト仮想マシンを確実にシャットダウンしたら、以下のようなコマンドを使って `--ro` フラグを省略し、書き込みアクセス権を取得できます。



```
$ guestfish -d RHEL3 -i
```

```
Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.
```

```
Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell
```

```
Operating system: Red Hat Enterprise Linux AS release 3 (Taroon Update 9)
/dev/vda2 mounted on /
/dev/vda1 mounted on /boot
```

```
><fs> edit /boot/grub/grub.conf
```

ファイルを編集するコマンドには **edit**、**vi**、および **emacs** などがあります。ファイルやディレクトリを作成するコマンドには **write**、**mkdir**、**upload**、および **tar-in** など多数あります。

#### 22.4.4. guestfish を使用したその他の動作

ファイルシステムの形式、パーティションの作成、LVM 論理ボリュームの作成およびサイズ変更などその他多くの動作を **mkfs**、**part-add**、**lvresize**、**lvcreate**、**vgcreate**、**pvcreate** などのコマンドを使って実行することができます。

#### 22.4.5. guestfish を使用したシェルスクリプトの作成

インタラクティブな **guestfish** の使い方に慣れたら、必要に応じてシェルスクリプトを作成すると便利です。以下に新しい MOTD (本日のメッセージ) をゲストに追加する簡単なシェルスクリプトを示します。

```
#!/bin/bash -
set -e
guestname="$1"

guestfish -d "$guestname" -i <<'EOF'
    write /etc/motd "Welcome to Acme Incorporated."
    chmod 0644 /etc/motd
EOF
```

#### 22.4.6. Augeas と libguestfs を使用したスクリプトの作成

**libguestfs** と **Augeas** を組み合わせると、Linux ゲスト仮想マシンの設定を操作するスクリプトを作成する場合に便利です。たとえば、以下のスクリプトでは **Augeas** を使ってゲスト仮想マシンのキーボード設定を解析し、レイアウトを出力します。以下の例は **Red Hat Enterprise Linux** を実行するゲスト仮想マシンでのみ機能することに注意してください。

```
#!/bin/bash -
set -e
guestname="$1"

guestfish -d "$1" -i --ro <<'EOF'
    aug-init / 0
    aug-get /files/etc/sysconfig/keyboard/LAYOUT
EOF
```

**Augeas** は設定ファイルの修正に使用することもできます。上記のスクリプトを使ってキーボードのレイアウトを変更することができます。

```
#!/bin/bash -
set -e
guestname="$1"

guestfish -d "$1" -i <<'EOF'
  aug-init / 0
  aug-set /files/etc/sysconfig/keyboard/LAYOUT '"gb"'
  aug-save
EOF
```

これらの2つのスクリプトには異なる点が3カ所あります。

1. 2番目の例には **--ro** オプションがありません。ゲスト仮想マシンへの書き込み権限を与えるためです。
2. **aug-get** コマンドが **aug-set** コマンドに変わっています。値を新たに取得するのではなく、それを修正するためです。新しい値は **"gb"** になります (引用符を含む)。
3. **Augeas** によって変更をディスクに書き込むよう **aug-save** コマンドが使用されています。



### 注記

**Augeas** の詳細については、Web サイト <http://augeas.net> をご覧ください。

**guestfish** では本ガイドでは取り上げきれないほどの多くのことができます。たとえば、ディスクイメージをゼロから作成することができます。

```
guestfish -N fs
```

または、ディスクイメージからディレクトリ全体をコピーします。

```
><fs> copy-out /home /tmp/home
```

詳細については **guestfish(1)** の **man** ページをご覧ください。

## 22.5. その他のコマンド

このセクションでは、ゲスト仮想マシンのディスクイメージの表示や編集を行う際に使用する **guestfish** に類するシンプルなツールについて説明します。

- **virt-cat** は **guestfish** の **download** コマンドに似ています。単一ファイルをダウンロードしてゲスト仮想マシンに表示します。たとえば、以下のようになります。

```
# virt-cat RHEL3 /etc/ntp.conf | grep ^server
server      127.127.1.0      # local clock
```

- **virt-edit** は **guestfish** の **edit** コマンドに似ています。ゲスト仮想マシン内の単一ファイルをインタラクティブに編集するために使用できます。たとえば、起動しない **Linux** ベースのゲスト仮想マシンの **grub.conf** ファイルを編集しなければならないとします。

```
# virt-edit LinuxGuest /boot/grub/grub.conf
```

**virt-edit** には、単一ファイルに一方的に直接の変更を加えることができる別モードもあります。この非インタラクティブなモードで編集を行う場合は **-e** オプションを使用します。たとえば、このコマンドでは Linux ゲスト仮想マシンの **root** パスワードをパスワードなしに変更します。

```
# virt-edit LinuxGuest /etc/passwd -e 's/^root:.*?:/root::/'
```

- **virt-ls** は **guestfish** の **ls**、**ll**、**find** の各コマンドに似ています。単一ディレクトリーまたは複数ディレクトリーを (再帰的に) 一覧表示するために使用されます。たとえば、以下のコマンドでは Linux ゲスト仮想マシンの **/home** の下にあるファイルとディレクトリーを再帰的に一覧表示します。

```
# virt-ls -R LinuxGuest /home/ | less
```

## 22.6. VIRT-RESCUE: レスキューシェル

このセクションでは、レスキューシェルについての情報を提供します。

### 22.6.1. はじめに

このセクションでは、仮想マシンのレスキュー CD のようなものと捉えることができる **virt-rescue** について説明します。レスキューシェルでゲスト仮想マシンを起動するため、エラー修正などのメンテナンスを行ってゲスト仮想マシンを修復することができます。

**virt-rescue** と **guestfish** には機能的に重複している部分があります。両者の使い方の違いを区別しておくことは大切です。**virt-rescue** は、通常の Linux ファイルシステムツールを使って特別な変更をインタラクティブに行う場合に使用します。とくに問題が発生したゲスト仮想マシンを修復する場合などに適しています。**virt-rescue** をスクリプト化することはできません。

一方 **guestfish** は型通りのコマンド一式 (**libguestfs API**) を使って、スクリプト化によって構造化した変更を行う場合に便利です。また、**guestfish** はインタラクティブに使用することもできます。

### 22.6.2. virt-rescue の実行

ゲスト仮想マシンで **virt-rescue** を使用する前に、そのゲスト仮想マシンが実行されていないことを確認してください。ゲスト仮想マシンが実行しているとディスクを破損させることになります。ゲスト仮想マシンがライブでないことを確認してから、以下を入力します。

```
$ virt-rescue -d GuestName
```

(上記の **GuestName** には **libvirt** が認識できるゲスト名を入力します) または、

```
$ virt-rescue -a /path/to/disk/image
```

(上記のパスは任意のファイル、論理ボリューム、LUN などいずれでも構いません) ゲスト仮想マシンのディスクが含まれます。

**virt-rescue** によってレスキュー仮想マシンが起動するため、その出力スクロールが表示されます。最終的には、以下が表示されます。

```
Welcome to virt-rescue, the libguestfs rescue shell.
```

```
Note: The contents of / are the rescue appliance.
```

```
You have to mount the guest virtual machine's partitions under /sysroot
before you can examine them.
```

```
bash: cannot set terminal process group (-1): Inappropriate ioctl for
device
```

```
bash: no job control in this shell
```

```
><rescue>
```

シェルプロンプトはここでは通常の **bash** シェルになるため、使用できる通常の Red Hat Enterprise Linux コマンド数が少なくなります。たとえば、以下を入力できます。

```
><rescue> fdisk -l /dev/vda
```

上記のコマンドはディスクパーティションを一覧表示します。ファイルシステムをマウントするには、**/sysroot** の下にマウントすることをお勧めします。このディレクトリーは、レスキューマシン内にあるユーザーが何でもマウントできる空ディレクトリーになります。**/** の下にあるファイル群はレスキュー仮想マシン自体のファイルになることに注意してください。

```
><rescue> mount /dev/vda1 /sysroot/
```

```
EXT4-fs (vda1): mounted filesystem with ordered data mode. Opts: (null)
```

```
><rescue> ls -l /sysroot/grub/
```

```
total 324
```

```
-rw-r--r--. 1 root root      63 Sep 16 18:14 device.map
```

```
-rw-r--r--. 1 root root  13200 Sep 16 18:14 e2fs_stage1_5
```

```
-rw-r--r--. 1 root root  12512 Sep 16 18:14 fat_stage1_5
```

```
-rw-r--r--. 1 root root  11744 Sep 16 18:14 ffs_stage1_5
```

```
-rw-----. 1 root root   1503 Oct 15 11:19 grub.conf
```

```
[...]
```

ゲスト仮想マシンの修復が完了したら、**exit** を入力するか、または **Ctrl+d** でシェルを終了します。

**virt-rescue** にはコマンドラインのオプションが多数あります。最もよく使用されるオプションを示します。

- **--ro**: ゲスト仮想マシン上で読み取り専用モードで動作します。変更は保存されません。ゲスト仮想マシンを実験的に使用する場合にこのモードを使用できます。シェルを終了すると、変更はすべて破棄されます。
- **--network**: レスキューシェルからのネットワークアクセスを可能にします。RPM や他のファイルをゲスト仮想マシンにダウンロードする場合など必要に応じて使用します。

## 22.7. VIRT-DF: ディスク使用量の監視

このセクションでは、ディスク使用量の監視についての情報を提供します。

### 22.7.1. はじめに

このセクションでは、ディスクイメージやゲスト仮想マシンのファイルシステムの使用量を表示する **virt-df** について説明します。Linux **df** コマンドに似ていますが、これは仮想マシン用になります。

## 22.7.2. virt-df の実行

ディスクイメージ内にあるすべてのファイルシステムの使用量を表示する場合は、以下を入力します。

```
# virt-df -a /dev/vg_guests/RHEL7
Filesystem              1K-blocks      Used  Available  Use%
RHEL6:/dev/sda1          101086        10233     85634    11%
RHEL6:/dev/VolGroup00/LogVol00 7127864      2272744    4493036   32%
```

(`/dev/vg_guests/RHEL7` は Red Hat Enterprise Linux 7 ゲスト仮想マシンのディスクイメージになります。上記の例では、パスはこのディスクイメージがあるホスト物理マシンの論理ボリュームになります。)

**virt-df** を単体で使用してすべてのゲスト仮想マシンの情報を一覧表示することもできます (libvirt が認識できるすべてのゲスト仮想マシン)。**virt-df** コマンドは、**-h** (人間が読める形式) や **-i** (ブロックの代わりに **inode** を表示) などの標準 **df** と同じオプションのいくつかを認識します。

```
# virt-df -h -d domname
Filesystem              Size          Used  Available  Use%
F14x64:/dev/sda1        484.2M        66.3M     392.9M    14%
F14x64:/dev/vg_f14x64/lv_root 7.4G         3.0G         4.4G    41%
RHEL6brewx64:/dev/sda1  484.2M        52.6M     406.6M    11%
RHEL6brewx64:/dev/vg_rhel6brewx64/lv_root
                        13.3G         3.4G         9.2G    26%
```



### 注記

**virt-df** には読み取り専用のアクセス権のみが必要になるため、ライブのゲスト仮想マシン上でこのコマンドを安全に使用することができます。ただし、上記の数値は、ゲスト仮想マシン内で **df** コマンドを実行した場合の数値と全く同一になると期待することはできません。ディスク上にあるものは、ライブゲスト仮想マシンの状態との同期に若干のずれがあります。しかし、分析や監視を行う上で問題となるほどの違いではありません。

**virt-df** は、統計値をモニタリングツールやデータベースなどに統合する目的で設計されています。これにより、システム管理者はディスク使用量の傾向に関するレポートを生成し、ゲスト仮想マシンがディスク領域を使いきってしまいそうな場合には警報を発することができるようになります。これには、**--csv** オプションを使ってマシンが読み取り可能なコンマ区切りの値 (**CSV - Comma-Separated-Values**) の出力を生成する必要があります。**CSV** 出力はほとんどのデータベース、表計算ソフトウェア、その他各種ツール、およびプログラミング言語などで認識することができます。生の **CSV** を以下に示します。

```
# virt-df --csv -d RHEL6Guest
Virtual Machine,Filesystem,1K-blocks,Used,Available,Use%
RHEL6brewx64,/dev/sda1,102396,24712,77684,24.1%
RHEL6brewx64,/dev/sda2,20866940,7786652,13080288,37.3%
```

## 22.8. VIRT-RESIZE: オフラインのゲスト仮想マシンのサイズ変更

このセクションでは、オフラインのゲスト仮想マシンのサイズ変更についての情報を提供します。

### 22.8.1. はじめに

このセクションでは、ゲスト仮想マシンのサイズを拡張したり縮小したりするツール **virt-resize** について説明します。このツールはオフラインの (シャットダウンされている) ゲスト仮想マシンの場合にのみ動作します。ゲスト仮想マシンのイメージをコピーしてオリジナルのディスクイメージはそのまま残します。これは、オリジナルのイメージをバックアップとして使用できるために理想的なツールとなりますが、ディスク領域の 2 倍の容量が必要になります。

## 22.8.2. ディスクイメージの拡張

このセクションでは、ディスクイメージの簡単な拡張方法について説明します。

1. サイズ変更するディスクイメージの場所を確認します。libvirt ゲスト仮想マシンの場合は **virsh dumpxml GuestName** コマンドを使用できます。
2. ゲスト仮想マシンを拡張する方法を決定します。以下の出力にあるように、ゲスト仮想マシンのディスクで **virt-df -h** と **virt-filesystems** を実行します。

```
# virt-df -h -a /dev/vg_guests/RHEL6
Filesystem                Size      Used    Available   Use%
RHEL6:/dev/sda1            98.7M    10.0M      83.6M      11%
RHEL6:/dev/VolGroup00/LogVol100  6.8G     2.2G      4.3G      32%

# virt-filesystems -a disk.img --all --long -h
/dev/sda1 ext3 101.9M
/dev/sda2 pv 7.9G
```

以下で、使用方法を説明します。

- 1 番目の (boot) パーティションを約 100MB から 500MB に拡大する
- ディスクの合計サイズを 8GB から 16GB に拡大する
- 2 番目のパーティションを拡大して残りの領域すべてを埋める
- **/dev/VolGroup00/LogVol100** を拡大して 2 番目のパーティションの新規領域を埋める

1. ゲスト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。
2. バックアップとして元のディスクの名前を変更します。名前の変更方法については、元のディスクのホスト物理マシンのストレージ環境によって異なります。ファイルとして格納されている場合は **mv** コマンドを使用します。論理ボリュームの場合は (例で示すように) **lvrename** を使用します。

```
# lvrename /dev/vg_guests/RHEL6 /dev/vg_guests/RHEL6.backup
```

3. 新規ディスクを作成します。この例では、合計ディスクサイズを 16GB に拡大することが要件になっています。ここでは論理ボリュームを使用しているので、以下のコマンドが使用されます。

```
# lvcreate -L 16G -n RHEL6 /dev/vg_guests
Logical volume "RHEL6" created
```

4. 上記の要件をコマンドで表すと以下のようになります。

```
# virt-resize \
```

```
/dev/vg_guests/RHEL6.backup /dev/vg_guests/RHEL6 \
--resize /dev/sda1=500M \
--expand /dev/sda2 \
--LV-expand /dev/VolGroup00/LogVol00
```

1 番目と 2 番目の引数は入力ディスクと出力ディスクになります。 **--resize /dev/sda1=500M** は 1 番目のパーティションを 500MB にサイズ変更します。 **--expand /dev/sda2** は 2 番目のパーティションを拡大し、残りの領域すべてを埋めます。 **--LV-expand /dev/VolGroup00/LogVol00** はゲスト仮想マシンの論理ボリュームを拡張し、2 番目のパーティションの追加領域を埋めます。

**virt-resize** は、その出力に実行内容を表示します。

```
Summary of changes:
/dev/sda1: partition will be resized from 101.9M to 500.0M
/dev/sda1: content will be expanded using the 'resize2fs' method
/dev/sda2: partition will be resized from 7.9G to 15.5G
/dev/sda2: content will be expanded using the 'pvresize' method
/dev/VolGroup00/LogVol00: LV will be expanded to maximum size
/dev/VolGroup00/LogVol00: content will be expanded using the
'resize2fs' method
Copying /dev/sda1 ...
[#####]
Copying /dev/sda2 ...
[#####]
Expanding /dev/sda1 using the 'resize2fs' method
Expanding /dev/sda2 using the 'pvresize' method
Expanding /dev/VolGroup00/LogVol00 using the 'resize2fs' method
```

5. 仮想マシンを起動してみます。正しく起動したら(全体を十分に確認した後)、バックアップのディスクを削除します。起動に失敗してしまう場合は、仮想マシンをシャットダウンして新しいディスクを削除し、バックアップのディスク名を元の名前に戻します。

6. 変更後のサイズを表示するには、**virt-df** または **virt-filesystems** を使用します。

```
# virt-df -h -a /dev/vg_pin/RHEL6
Filesystem                Size      Used    Available
Use%
RHEL6:/dev/sda1            484.4M    10.8M     448.6M
3%
RHEL6:/dev/VolGroup00/LogVol00 14.3G      2.2G     11.4G    16%
```

ゲスト仮想マシンのサイズ変更は精密に実行されない場合があるため、**virt-resize** が失敗する場合は **virt-resize(1)** の man ページにある数多くのヒントを参考にしてください。旧バージョンの Red Hat Enterprise Linux ゲスト仮想マシンなどの場合には、GRUB に関するヒントにとくに注意する必要があるかもしれません。

## 22.9. VIRT-INSPECTOR: ゲスト仮想マシンの検査

このセクションでは、ゲスト仮想マシンの検査についての情報を提供します。

### 22.9.1. はじめに

**virt-inspector** は、ディスクイメージに含まれているオペレーティングシステムを判別するためにディスクイメージの検査を行うツールです。

## 22.9.2. インストール

**virt-inspector** とその関連ドキュメントをインストールするには次のコマンドを入力します。

```
# yum install libguestfs-tools
```

サンプルの XML 出力や出力の Relax-NG スキーマなどのドキュメントは `/usr/share/doc/libguestfs-devel-*/` にインストールされます。「\*」は **libguestfs** のバージョン番号です。

## 22.9.3. virt-inspector の実行

**virt-inspector** は、以下の例のようにどのディスクイメージや **libvirt** ゲスト仮想マシンに対しても実行することができます。

```
$ virt-inspector -a disk.img > report.xml
```

または、次のように実行することもできます。

```
$ virt-inspector -d GuestName > report.xml
```

結果として XML レポート (**report.xml**) が生成されます。以下のように、通常は **<operatingsystem>** 要素を 1 つだけ持つトップレベルの **<operatingsystems>** 要素が XML ファイルの主要なコンポーネントになります。

```
<operatingsystems>
  <operatingsystem>

    <!-- the type of operating system and Linux distribution -->
    <name>linux</name>
    <distro>rhel</distro>
    <!-- the name, version and architecture -->
    <product_name>Red Hat Enterprise Linux Server release 6.4
  </product_name>
    <major_version>6</major_version>
    <minor_version>4</minor_version>
    <package_format>rpm</package_format>
    <package_management>yum</package_management>
    <root>/dev/VolGroup/lv_root</root>
    <!-- how the filesystems would be mounted when live -->
    <mountpoints>
      <mountpoint dev="/dev/VolGroup/lv_root"></mountpoint>
      <mountpoint dev="/dev/sda1">boot</mountpoint>
      <mountpoint dev="/dev/VolGroup/lv_swap">swap</mountpoint>
    </mountpoints>

    <!-- filesystems -->
    <filesystem dev="/dev/VolGroup/lv_root">
      <label></label>
      <uuid>b24d9161-5613-4ab8-8649-f27a8a8068d3</uuid>
      <type>ext4</type>
```



```

        <content>linux-root</content>
        <spec>/dev/mapper/VolGroup-lv_root</spec>
    </filesystem>
    <filesystem dev="/dev/VolGroup/lv_swap">
        <type>swap</type>
        <spec>/dev/mapper/VolGroup-lv_swap</spec>
    </filesystem>
<!-- packages installed -->
<applications>
    <application>
        <name>firefox</name>
        <version>3.5.5</version>
        <release>1.fc12</release>
    </application>
</applications>

</operatingsystem>
</operatingsystems>

```

こうしたレポートの処理は W3C 標準 XPath 照会で行うのが最適です。Red Hat Enterprise Linux 7 には単純なインスタンスに使用できるコマンドラインプログラム (**xpath**) が同梱されています。ただし、長期にわたって高度な使い方をする場合、XPath ライブラリーを使い慣れているプログラミング言語で使用することを考慮されてください。

たとえば、次のような XPath クエリーを使ってすべてのファイルシステムデバイスを一覧表示できます。

```

$ virt-inspector GuestName | xpath //filesystem/@dev
Found 3 nodes:
-- NODE --
dev="/dev/sda1"
-- NODE --
dev="/dev/vg_f12x64/lv_root"
-- NODE --
dev="/dev/vg_f12x64/lv_swap"

```

または、インストールしているすべてのアプリケーション名を一覧表示するには次のように入力します。

```

$ virt-inspector GuestName | xpath //application/name
[...long list...]

```

## 22.10. プログラミング言語での API の使用

libguestfs の API は、Red Hat Enterprise Linux 7 の C、C++、Perl、Python、Java、Ruby、OCaml の言語で直接使用することができます。

- C と C++ のバインディングをインストールするには、以下のコマンドを入力します。

```
# yum install libguestfs-devel
```

- Perl バインディングをインストールする場合

```
# yum install 'perl(Sys::Guestfs)'
```

- Python バインディングをインストールする場合

```
# yum install python-libguestfs
```

- Java バインディングをインストールする場合

```
# yum install libguestfs-java libguestfs-java-devel libguestfs-javadoc
```

- Ruby バインディングをインストールする場合

```
# yum install ruby-libguestfs
```

- OCaml バインディングをインストールする場合

```
# yum install ocaml-libguestfs ocaml-libguestfs-devel
```

各言語のバインディングは基本的には同じですが、構造上の若干の変更点があります。以下に A C ステートメント示します。

```
guestfs_launch (g);
```

Perl では、以下のように表されます。

```
$g->launch ()
```

または、OCaml では以下のように表されます。

```
g#launch ()
```

このセクションでは C の API のみを詳しく説明します。

C と C++ バインディングでは、エラーのチェックは手作業で行う必要があります。他のバインディングでは、エラーは例外に変換されます。以下の例に示す追加のエラーチェックは他の言語では必要ありませんが、例外をキャッチするためにコードを追加したいと思われるかもしれません。libguestfs API のアーキテクチャーに関する注意点を以下に示します。

- libguestfs API は同期します。各呼び出しは完了するまでブロックします。非同期に呼び出しを行いたい場合はスレッドを作成する必要があります。
- libguestfs API はスレッドセーフではありません。1 スレッドにつき 1 ハンドルしか使用できません。複数のスレッドで 1 つのハンドルを共有したい場合は、2 つのスレッドが 1 つのハンドルで同時にコマンドを実行できないよう独自の相互排除を実装してください。
- 同じディスクイメージ上で複数のハンドルを開かないようにしてください。ハンドルがすべて読み取り専用の場合は許容されますが、推奨はされていません。
- 他の何かがディスクイメージを使用している可能性がある場合は書き込み用のディスクイメージは追加しないでください (ライブ仮想マシンなど)。これを行うとディスクの破損を招きます。
- 現在使用中のディスクイメージで読み取り専用のハンドルを開くことは可能ですが (ライブの

仮想マシンなど)、予測できない結果を招いたり、整合性を失う場合があります。とくにディスクイメージを読み込んでいる最中に大量の書き込みがあった場合にこのような結果になる可能性があります。

### 22.10.1. C プログラムでの API との対話

C プログラムは<guestfs.h> ヘッダーファイルを組み込み、ハンドルを作成するところから開始します。

```
#include <stdio.h>
#include <stdlib.h>
#include <guestfs.h>

int
main (int argc, char *argv[])
{
    guestfs_h *g;

    g = guestfs_create ();
    if (g == NULL) {
        perror ("failed to create libguestfs handle");
        exit (EXIT_FAILURE);
    }

    /* ... */

    guestfs_close (g);

    exit (EXIT_SUCCESS);
}
```

このプログラムをファイル (**test.c**) に保存します。このプログラムをコンパイルしてから次の 2 つのコマンドで実行します。

```
gcc -Wall test.c -o test -lguestfs
./test
```

この時点では何も出力されないはずです。このセクションの以降の部分では、このプログラムを拡張して新規のディスクイメージの作成やパーティション設定、**ext4** ファイルシステムでの形式、そのファイルシステム内でファイル作成を実行する方法の事例を示します。ディスクイメージ名は **disk.img** であり、これは現行のディレクトリーに作成されます。

プログラムの概要を以下に示します。

- ハンドルの作成
- ディスクのハンドルへの追加
- **libguestfs** バックエンドの起動
- パーティション、ファイルシステムおよびファイル群の作成
- ハンドルを閉じて終了

以下に、修正したプログラムを示します。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <guestfs.h>

int
main (int argc, char *argv[])
{
    guestfs_h *g;
    size_t i;

    g = guestfs_create ();
    if (g == NULL) {
        perror ("failed to create libguestfs handle");
        exit (EXIT_FAILURE);
    }

    /* Create a raw-format sparse disk image, 512 MB in size. */
    int fd = open ("disk.img", O_CREAT|O_WRONLY|O_TRUNC|O_NOCTTY, 0666);
    if (fd == -1) {
        perror ("disk.img");
        exit (EXIT_FAILURE);
    }
    if (ftruncate (fd, 512 * 1024 * 1024) == -1) {
        perror ("disk.img: truncate");
        exit (EXIT_FAILURE);
    }
    if (close (fd) == -1) {
        perror ("disk.img: close");
        exit (EXIT_FAILURE);
    }

    /* Set the trace flag so that we can see each libguestfs call. */
    guestfs_set_trace (g, 1);

    /* Set the autosync flag so that the disk will be synchronized
     * automatically when the libguestfs handle is closed.
     */
    guestfs_set_autosync (g, 1);

    /* Add the disk image to libguestfs. */
    if (guestfs_add_drive_opts (g, "disk.img",
        GUESTFS_ADD_DRIVE_OPTS_FORMAT, "raw", /* raw format */
        GUESTFS_ADD_DRIVE_OPTS_READONLY, 0, /* for write */
        -1 /* this marks end of optional arguments */ )
        == -1)
        exit (EXIT_FAILURE);

    /* Run the libguestfs back-end. */
    if (guestfs_launch (g) == -1)
        exit (EXIT_FAILURE);

    /* Get the list of devices. Because we only added one drive
     * above, we expect that this list should contain a single
```

```

    * element.
    */
char **devices = guestfs_list_devices (g);
if (devices == NULL)
    exit (EXIT_FAILURE);
if (devices[0] == NULL || devices[1] != NULL) {
    fprintf (stderr,
            "error: expected a single device from list-devices\n");
    exit (EXIT_FAILURE);
}

/* Partition the disk as one single MBR partition. */
if (guestfs_part_disk (g, devices[0], "mbr") == -1)
    exit (EXIT_FAILURE);

/* Get the list of partitions. We expect a single element, which
 * is the partition we have just created.
 */
char **partitions = guestfs_list_partitions (g);
if (partitions == NULL)
    exit (EXIT_FAILURE);
if (partitions[0] == NULL || partitions[1] != NULL) {
    fprintf (stderr,
            "error: expected a single partition from list-
partitions\n");
    exit (EXIT_FAILURE);
}

/* Create an ext4 filesystem on the partition. */
if (guestfs_mkfs (g, "ext4", partitions[0]) == -1)
    exit (EXIT_FAILURE);

/* Now mount the filesystem so that we can add files. */
if (guestfs_mount_options (g, "", partitions[0], "/") == -1)
    exit (EXIT_FAILURE);

/* Create some files and directories. */
if (guestfs_touch (g, "/empty") == -1)
    exit (EXIT_FAILURE);

const char *message = "Hello, world\n";
if (guestfs_write (g, "/hello", message, strlen (message)) == -1)
    exit (EXIT_FAILURE);

if (guestfs_mkdir (g, "/foo") == -1)
    exit (EXIT_FAILURE);

/* This uploads the local file /etc/resolv.conf into the disk image. */
if (guestfs_upload (g, "/etc/resolv.conf", "/foo/resolv.conf") == -1)
    exit (EXIT_FAILURE);

/* Because 'autosync' was set (above) we can just close the handle
 * and the disk contents will be synchronized. You can also do
 * this manually by calling guestfs_umount_all and guestfs_sync.
 */
guestfs_close (g);

```

```

/* Free up the lists. */
for (i = 0; devices[i] != NULL; ++i)
    free (devices[i]);
free (devices);
for (i = 0; partitions[i] != NULL; ++i)
    free (partitions[i]);
free (partitions);

exit (EXIT_SUCCESS);
}

```

次の2つのコマンドを使用してこのプログラムをコンパイルしてから実行します。

```

gcc -Wall test.c -o test -lguestfs
./test

```

プログラムの実行が正しく終了すると、**disk.img** という名前のディスクイメージが作成されるはずです。このイメージは **guestfish** で調べることができます。

```

guestfish --ro -a disk.img -m /dev/sda1
><fs> ll /
><fs> cat /foo/resolv.conf

```

デフォルトでは (C と C++ のバインディングの場合のみ)、**libguestfs** は **stderr** にエラーを出力します。エラーハンドラーを設定すると、この動作を変更することができます。詳細については **guestfs(3)** の **man** ページに記載されています。

## 22.11. VIRT-SYSPREP: 仮想マシン設定のリセット

**virt-sysprep** コマンドラインツールは、クローンを作成できるよう、ゲスト仮想マシンをリセットしたり、この設定を解除したりするために使用できます。このプロセスでは、SSH ホストキー、永続的なネットワーク MAC 設定、およびユーザーアカウントを削除します。また、**virt-sysprep** は、SSH キー、ユーザー、またはロゴの追加など、仮想マシンをカスタマイズすることもできます。各ステップは必要に応じて有効にしたり無効にしたりすることができます。

**virt-sysprep** を使用するには、ゲスト仮想マシンをオフラインにする必要があるため、コマンドを実行する前にこれをシャットダウンしてください。**virt-sysprep** は、コピーを作成せずに配置済みのゲストまたはディスクイメージを変更することに注意してください。ゲスト仮想マシンの既存のコンテナを保持するには、最初にディスクのスナップショット、コピー、またはクローンを作成する必要があります。ディスクのコピーまたはクローンに関する詳細は、[libguestfs.org](http://libguestfs.org) を参照してください。

ディスクイメージにアクセスするために **root** が不要な場合には、**root** で **virt-sysprep** を使用しないことを推奨します。この場合、ディスクイメージの権限を、**root** 以外のユーザーが **virt-sysprep** を実行して書き込みできるように変更することが適切です。

**virt-sysprep** をインストールするには、以下のコマンドを入力します。

```

$ sudo yum install /usr/bin/virt-sysprep

```

以下のコマンドオプションは **virt-sysprep** と使用することができます。

表22.1 **virt-sysprep** コマンド

コマンド	説明	例
<code>--help</code>	特定のコマンドまたは <b>virt-sysprep</b> コマンドについての簡潔なヘルプエントリを表示します。ヘルプの詳細は、 <b>virt-sysprep</b> の <code>man</code> ページを参照してください。	<code>\$ virt-sysprep --help</code>
<code>-a [file] or --add [file]</code>	ゲスト仮想マシンからのディスクイメージである指定のファイルを追加します。ディスクイメージの形式は自動検出されます。これを無効にし、特定の形式を強制実行するには、 <b>--format</b> オプションを使用します。	<code>\$ virt-sysprep --add /dev/vms/disk.img</code>
<code>-a [URI] or --add [URI]</code>	リモートディスクを追加します。URI 形式は <code>guestfish</code> と互換性があります。詳細は、「 <a href="#">guestfish を使用したファイルの追加</a> 」を参照してください。	<code>\$ virt-sysprep -a rbd://example.com[:port]/pool/disk</code>
<code>-c [URI] or --connect [URI]</code>	<code>libvirt</code> を使用している場合は、指定の URI に接続します。省略されている場合は、KVM ハイパーバイザーから接続されます。ゲストブロックデバイスを直接指定する場合は ( <b>virt-sysprep -a</b> )、 <code>libvirt</code> は一切使用されません。	<code>\$ virt-sysprep -c qemu:///system</code>
<code>-d [guest] or --domain [guest]</code>	指定されたゲスト仮想マシンからすべてのディスクを追加します。ドメイン UUID をドメイン名の代わりに使用できます。	<code>\$ virt-sysprep --domain 90df2f3f-8857-5ba9-2714-7d95907b1c9e</code>
<code>-n or --dry-run</code>	ゲスト仮想マシンに読み取り専用の「dry run」 <code>Sysprep</code> 操作を実行します。これは <code>sysprep</code> 操作を実行しますが、最終的にディスクへの変更を破棄します。	<code>\$ virt-sysprep -n</code>
<code>--enable [operations]</code>	指定された 操作 を有効にします。使用可能な操作を一覧表示するには、 <code>--list</code> コマンドを使用します。	<code>\$ virt-sysprep --enable ssh-hostkeys,udev-persistent-net</code>

コマンド	説明	例
<code>--operation or --operations</code>	実行する <b>sysprep</b> 操作を選択します。操作を無効にするには、操作名の前に <code>-</code> を使用します。	<b>\$ virt-sysprep --operations ssh-hotkeys,udev-persistent-net</b> は両方の操作を有効にしますが、 <b>\$ virt-sysprep --operations firewall-rules, -tmp-files</b> は <b>firewall-rules</b> 操作を有効にし、 <b>tmp-files</b> 操作を無効にします。有効な操作の一覧については、 <a href="https://libguestfs.org">libguestfs.org</a> を参照してください。
<code>--format [raw qcow2 auto]</code>	<code>-a</code> オプションは、デフォルトでディスクイメージの形式を自動検出します。これを使用すると、コマンドラインに続く <code>-a</code> オプションのディスク形式が強制実行されます。 <code>--format auto</code> を使用すると、後続の <code>-a</code> オプションの自動検出に切り替わります (上記の <code>-a</code> コマンドを参照)。	<b>\$ virt-sysprep --format raw -a disk.img</b> は <b>disk.img</b> の <b>raw</b> 形式 (自動検出なし) を強制しますが、 <b>virt-sysprep -format raw -a disk.img --format auto -a another.img</b> は <b>disk.img</b> の <b>raw</b> 形式 (自動検出なし) を強制実行してから <b>another.img</b> の自動検出に戻ります。信頼されていない <b>raw</b> 形式のゲストディスクイメージがある場合、このオプションを使用してディスク形式を指定する必要があります。これにより、悪意のあるゲストに関連するセキュリティ上の問題を防ぐことができます。
<code>--list-operations</code>	<b>virt-sysprep</b> プログラムでサポートされる操作を一覧表示します。これらは 1 行に 1 項目ずつ一覧表示され、フィールドには 1 つ以上のスペースが使用されます。出力の最初のフィールドには、 <b>--enable</b> フラグに指定できる操作名が入ります。2 つ目のフィールドには、操作がデフォルトで有効である場合は <code>*</code> 文字が入り、有効にされていない場合は空白になります。同じラインの追加フィールドには操作の説明が入ります。	<b>\$ virt-sysprep --list-operations</b>



コマンド	説明	例
<code>--mount-options</code>	ゲスト仮想マシンの各マウントポイントについてマウントオプションを設定します。マウントポイントとオプションのペア (mountpoint:options) のセミコロンので区切られた一覧を使用します。この一覧を引用符で囲み、シェルでこれを保護する必要がある場合があります。	<code>\$ virt-sysprep --mount-options "/:notime"</code> は、 <b>notime</b> 操作でルートディレクトリーをマウントします。
<code>-q</code> or <code>--quiet</code>	ログメッセージが出力されないようにします。	<code>\$ virt-sysprep -q</code>
<code>-v</code> or <code>--verbose</code>	デバッグ目的で詳細なメッセージを有効にします。	<code>\$ virt-sysprep -v</code>
<code>-V</code> or <code>--version</code>	<code>virt-sysprep</code> バージョン番号を表示し、終了します。	<code>\$ virt-sysprep -V</code>
<code>--root-password</code>	<code>root</code> パスワードを設定します。新規パスワードを明示的に指定したり、選択したファイルの最初の行の文字列を使用したりできます。後者がより安全な方法です。	<code>\$ virt-sysprep --root-password password:123456 -a guest.img</code>  または <code>\$ virt-sysprep --root-password file:SOURCE_FILE_PATH -a guest.img</code>

詳細は [libguestfs ドキュメント](#) を参照してください。

## 22.12. VIRT-CUSTOMIZE: 仮想マシン設定のカスタマイズ

**virt-customize** コマンドラインツールは仮想マシンをカスタマイズするために使用できます。たとえば、パッケージをインストールしたり、設定ファイルを編集したりして実行できます。

**virt-customize** を使用するには、ゲスト仮想マシンをオフラインにする必要があるため、コマンドを実行する前にこれをシャットダウンしてください。**virt-customize** は、コピーを作成せずに配置済みのゲストまたはディスクイメージを変更することに注意してください。ゲスト仮想マシンの既存のコンテンツを保持するには、最初にディスクのスナップショット、コピー、またはクローンを作成する必要があります。ディスクのコピーまたはクローンに関する詳細は、[libguestfs.org](http://libguestfs.org) を参照してください。



## 警告

実行中の仮想マシン上で **virt-customize** を使用したり、他のディスク編集ツールと同時に使用したりすると、ディスクが破損する恐れがあります。このコマンドを使用する前に、仮想マシンを **シャットダウン** してください。また、ディスクイメージを同時に編集することはできません。

**virt-customize** を **root** で実行することは推奨されていません。

**virt-customize** をインストールするには、以下のコマンドのいずれかを実行します。

```
$ sudo yum install /usr/bin/virt-customize
```

または

```
$ sudo yum install libguestfs-tools-c
```

以下のコマンドオプションは **virt-customize** と使用することができます。

表22.2 **virt-customize** オプション

コマンド	説明	例
<b>--help</b>	特定のコマンドまたは <b>virt-customize</b> パッケージについての簡潔な <b>help</b> エントリーを表示します。ヘルプの詳細は、 <b>virt-customize</b> の <b>man</b> ページを参照してください。	<b>\$virt-customize --help</b>
<b>-a [file]</b> or <b>--add [file]</b>	ゲスト仮想マシンからのディスクイメージである指定のファイルを追加します。ディスクイメージの形式は自動検出されます。これを無効にし、特定の形式を強制実行するには、 <b>--format</b> オプションを使用します。	<b>\$virt-customize --add /dev/vms/disk.img</b>
<b>-a [URI]</b> or <b>--add [URI]</b>	リモートディスクを追加します。URI 形式は <b>guestfish</b> と互換性があります。詳細は、「 <a href="#">guestfish を使用したファイルの追加</a> 」を参照してください。	<b>\$virt-customize -a rbd://example.com[:port]/pool/disk</b>

コマンド	説明	例
<code>-c [URI] or --connect [URI]</code>	<code>libvirt</code> を使用している場合は、指定の URI に接続します。省略されている場合は、KVM ハイパーバイザーから接続されます。ゲストブロックデバイスを直接指定する場合は ( <b><code>virt-customize -a</code></b> )、 <code>libvirt</code> は一切使用されません。	<b><code>\$ virt-customize -c qemu:///system</code></b>
<code>-d [guest] or --domain [guest]</code>	指定されたゲスト仮想マシンからすべてのディスクを追加します。ドメイン UUID をドメイン名の代わりに使用できます。	<b><code>\$ virt-customize --domain 90df2f3f-8857-5ba9-2714-7d95907b1c9e</code></b>
<code>-n or --dry-run</code>	ゲスト仮想マシンに読み取り専用の「 <b><code>dry run</code></b> 」カスタマイズ操作を実行します。これはカスタム操作を実行しますが、最終的にディスクへの変更を破棄します。	<b><code>\$ virt-customize -n</code></b>
<code>--format [raw qcow2 auto]</code>	<code>-a</code> オプションは、デフォルトでディスクイメージの形式を自動検出します。これを使用すると、コマンドラインに続く <code>-a</code> オプションのディスク形式が強制実行されます。 <b><code>--format auto</code></b> を使用すると、後続の <code>-a</code> オプションの自動検出に切り替わります (上記の <b><code>-a</code></b> コマンドを参照)。	<b><code>\$ virt-customize --format raw -a disk.img</code></b> は <code>disk.img</code> の <code>raw</code> 形式 (自動検出なし) を強制しますが、 <b><code>virt-customize --format raw -a disk.img --format auto -a another.img</code></b> は <code>disk.img</code> の <code>raw</code> 形式 (自動検出なし) を強制実行してから <b><code>another.img</code></b> の自動検出に戻ります。信頼されていない <code>raw</code> 形式のゲストディスクイメージがある場合、このオプションを使用してディスク形式を指定する必要があります。これにより、悪意のあるゲストに関連するセキュリティ上の問題を防ぐことができます。
<code>-m [MB] or --memsize [MB]</code>	<b><code>--run</code></b> スクリプトに割り当てられたメモリの容量を変更します。 <b><code>--run</code></b> スクリプトまたは <b><code>-install</code></b> オプションがメモリ不足の問題を生じさせる場合は、メモリの割り当てを増やしてください。	<b><code>\$ virt-customize --memsize 1024</code></b>

コマンド	説明	例
<code>--network</code> または <code>--no-network</code>	インストール時にゲストからのネットワークアクセスを有効または無効にします。デフォルトでは有効にされています。 <b>--no-network</b> を使用してアクセスを無効にします。このコマンドは、ゲストの起動後のネットワークへのアクセスに影響を与えません。詳細は、 <a href="#">libguestfs ドキュメント</a> を参照してください。	<code>\$ virt-customize -a http://[user@]example.com[:port]/disk.img</code>
<code>-q</code> or <code>--quiet</code>	ログメッセージが出力されないようにします。	<code>\$ virt-customize -q</code>
<code>-smp [N]</code>	<b>--install</b> スクリプトで利用できる <i>N</i> 仮想 CPU を有効にします。 <i>N</i> は 2 以上にする必要があります。	<code>\$ virt-customize -smp 4</code>
<code>-v</code> or <code>--verbose</code>	デバッグ目的で詳細なメッセージを有効にします。	<code>\$ virt-customize --verbose</code>
<code>-V</code> or <code>--version</code>	<code>virt-customize</code> バージョン番号を表示し、終了します。	<code>\$ virt-customize --V</code>
<code>-x</code>	<code>libguestfs</code> API 呼び出しのトレースを有効にします。	<code>\$ virt-customize -x</code>

**virt-customize** コマンドはゲストのカスタマイズ方法を設定するためのカスタマイズオプションを使用します。以下は、**--selinux-relabel** カスタマイズオプションについての情報を提供します。

**--selinux-relabel** カスタマイズオプションは、ゲストのファイルを再ラベルするため、それらは正しい SELinux ラベルを持ちます。このオプションはファイルの再レベルを即時に試行します。これに失敗する場合、**/.autorelabel** はイメージ上でアクティベートされます。これにより、次のイメージの起動時に再ラベル操作がスケジュールされます。



### 注記

このオプションは、SELinux をサポートするゲストにのみ使用する必要があります。

以下の例では、ゲストに **GIMP** および **Inkscape** パッケージをインストールし、ゲストの次回起動時に SELinux ラベルが正しくなるようにします。

#### 例22.1 virt-customize を使用したパッケージのゲストへのインストール

```
virt-customize -a disk.img --install gimp,inkscape --selinux-relabel
```

カスタマイズオプションを含む詳細情報は、[libguestfs.org](http://libguestfs.org) を参照してください。

### 22.13. VIRT-DIFF: 仮想マシンファイル間の相違点の一覧表示

**virt-diff** コマンドラインツールを使用して、2つの仮想マシンのディスクイメージのファイル間の相違点を一覧表示できます。出力は、実行後の仮想マシンのディスクイメージへの変更を表示します。このコマンドを使用して、オーバーレイ間の相違点を表示することもできます。



注記

**virt-diff** には読み取り専用のアクセス権のみが必要になるため、ライブのゲスト仮想マシン上でこのコマンドを安全に使用することができます。

このツールは、実行中の仮想マシンと選択したイメージ間のファイル名、ファイルサイズ、チェックサム、拡張属性、ファイル内容などの相違点を検出します。



注記

**virt-diff** コマンドは、ブートローダー、パーティションまたはファイルシステム内の未使用の領域、または「非表示」セクターをチェックしません。そのため、これをセキュリティまたはフォレンジクスツールとして使用することは推奨されません。

**virt-diff** をインストールするには、以下のコマンドのいずれかを実行します。

```
# yum install /usr/bin/virt-diff
```

または

```
# yum install libguestfs-tools-c
```

2つのゲストを指定するには、最初のゲストに **-a** または **-d** オプションを使用し、2つ目のゲストに **-A** または **-D** オプションを使用する必要があります。以下は例になります。

```
$ virt-diff -a old.img -A new.img
```

また、**libvirt** に認識されている名前も使用できます。以下は例になります。

```
$ virt-diff -d oldguest -D newguest
```

以下のコマンドオプションは **virt-diff** と併用できます。

表22.3 **virt-diff** オプション

コマンド	説明	例
--help	特定のコマンドまたは <b>virt-diff</b> パッケージについての簡潔な <b>help</b> エントリを表示します。 <b>help</b> の詳細は、 <b>virt-diff</b> の <b>man</b> ページを参照してください。	<b>\$ virt-diff --help</b>

コマンド	説明	例
<code>-a [file] or --add [file]</code>	<p>最初の仮想マシンのディスクイメージである指定のファイルを追加します。仮想マシンに複数のブロックデバイスがある場合、それらすべてに <b>-a</b> オプションを指定する必要があります。</p> <p>ディスクイメージの形式は自動検出されます。これを無効にし、特定の形式を強制実行するには、<b>-format</b> オプションを使用します。</p>	<code>\$ virt-customize --add /dev/vms/original.img -A /dev/vms/new.img</code>
<code>-a [URI] or --add [URI]</code>	リモートディスクを追加します。URI 形式は <code>guestfish</code> と互換性があります。詳細は、「 <a href="#">guestfish を使用したファイルの追加</a> 」を参照してください。	<code>\$ virt-diff -a rbd://example.com[:port]/pool/newdisk -A rbd://example.com[:port]/pool/olddisk</code>
<code>--all</code>	<b>--extra-stats</b> <b>--times</b> <b>--uids</b> <b>--xattrs</b> と同じです。	<code>\$ virt-diff --all</code>
<code>--atime</code>	デフォルトでは、ファイルアクセス時間の変更は関連性がないため、 <b>virt-diff</b> はこれらの変更を無視します。 <b>--atime</b> オプションを使用するとアクセス時間の相違点を表示できます。	<code>\$ virt-diff --atime</code>
<code>-A [file]</code>	2 つ目の仮想マシンのディスクイメージである指定の <code>file</code> または <code>URI</code> を追加します。	<code>\$ virt-diff --add /dev/vms/original.img -A /dev/vms/new.img</code>
<code>-c [URI] or --connect [URI]</code>	<code>libvirt</code> を使用している場合は、指定の <code>URI</code> に接続します。省略されている場合は、デフォルトの <code>libvirt</code> ハイパーバイザーに接続されます。ゲストブロックデバイスを直接指定する場合は ( <b>virt-diff -a</b> )、 <code>libvirt</code> は一切使用されません。	<code>\$ virt-diff -c qemu:///system</code>
<code>--csv</code>	コンマ区切り値 (CSV) の形式で結果を提供します。この形式はデータベースおよびスプレッドシートに簡単にインポートできます。詳細は、 <a href="#">注記</a> を参照してください。	<code>virt-diff --csv</code>

コマンド	説明	例
<code>-d [guest] or --domain [guest]</code>	指定されたゲスト仮想マシンのすべてのディスクを最初のゲスト仮想マシンとして追加します。ドメイン UUID をドメイン名の代わりに使用できます。	<code>\$ virt-diff --domain 90df2f3f-8857-5ba9-2714-7d95907b1c9e</code>
<code>-D [guest]</code>	指定されたゲスト仮想マシンのすべてのディスクを2つ目のゲスト仮想マシンとして追加します。ドメイン UUID をドメイン名の代わりに使用できます。	<code>\$ virt-diff --D 90df2f3f-8857-5ba9-2714-7d95907b1cd4</code>
<code>--extra-stats</code>	追加の統計情報を表示します。	<code>\$ virt-diff --extra-stats</code>
<code>--format or --format=[raw qcow2]</code>	デフォルトの <code>-a/-A</code> オプションは、ディスクイメージの形式を自動検出します。これを使用すると、コマンドラインに続く <code>-a/-A</code> オプションのディスク形式が強制実行されます。 <code>--format auto</code> を使用すると、後続の <code>-a</code> オプションの自動検出に切り替わります (上記の <code>-a</code> コマンド参照)。	<code>\$ virt-diff --format raw -a new.img: old.img</code> は <code>new.img</code> および <code>old.img</code> の raw 形式 (自動検出なし) を強制しますが、 <code>virt-sysprep --format raw -a new.img -format auto -a old.img</code> は <code>new.img</code> の raw 形式 (自動検出なし) を強制実行してから <code>old.img</code> の自動検出に戻ります。信頼されていない raw 形式のゲストディスクイメージがある場合、このオプションを使用してディスク形式を指定する必要があります。これにより、悪意のあるゲストに関連するセキュリティ上の問題を防ぐことができます。
<code>-h</code> または <code>--human-readable</code>	人間が判読できる形式でファイルサイズを表示します。	<code>\$ virt-diff -h</code>
<code>--time-days</code>	変更されたファイルのタイムフィールドを現在よりも前の日を指定して表示します (将来の場合は負の値になります)。  出力の <code>0</code> は現在の 86,399 秒 (23 時間 59 分 59 秒) 前から将来の 86,399 秒までの間になることに注意してください。	<code>\$ virt-diff --time-days</code>
<code>-v</code> or <code>--verbose</code>	デバッグ目的で詳細なメッセージを有効にします。	<code>\$ virt-diff --verbose</code>
<code>-V</code> or <code>--version</code>	<code>virt-diff</code> バージョン番号を表示し、終了します。	<code>\$ virt-diff -V</code>

コマンド	説明	例
<code>-x</code>	<code>libguestfs</code> API 呼び出しのトレースを有効にします。	<code>\$ virt-diff -x</code>



## 注記

コンマ区切りの値 (CSV) の形式は貼り付けが容易でない場合があります。そのため、シェルスクリプトの場合は `csvtool` を使用し、他の言語の場合は CSV 処理ライブラリー (Perl または Python の組み込み `csv` ライブラリーの場合の `Text::CSV` など) を使用する必要があります。さらに、大半のスプレッドシートおよびデータベースは CSV を直接インポートできます。

追加のオプションを含む詳細情報は、[libguestfs.org](http://libguestfs.org) を参照してください。

## 22.14. VIRT-SPARSIFY: 空のディスク領域の再要求

**virt-sparsify** コマンドラインツールは、仮想マシンディスク (またはすべてのディスクイメージ) をスパースにするために使用できます。これはシンプロビジョニングとしても知られています。ディスクイメージの空きディスク領域はホストの空き領域に変換されます。

**virt-sparsify** コマンドは、`ext2`、`ext3`、`ext4`、`btrfs`、`NTFS` などのほとんどのファイルシステムで使用できます。さらに、LVM 物理ボリュームでも使用できます。**virt-sparsify** は、仮想マシンのディスクイメージ上だけでなく、すべてのディスクイメージで機能します。



## 警告

実行中の仮想マシン上で **virt-sparsify** を使用したり、他のディスク編集ツールと同時に使用したりすると、ディスクが破損する恐れがあります。このコマンドを使用する前に、仮想マシンをシャットダウンしてください。また、ディスクイメージを同時に編集することはできません。

また、このコマンドは一部のディスク形式間の変換を実行できます。たとえば、**virt-sparsify** は Raw ディスクイメージをシンプロビジョニングされた `qcow2` イメージに変換できます。



## 注記

仮想マシンに複数のディスクがあり、ボリューム管理を使用する場合に **virt-sparsify** は機能しますが、効果性においては十分ではありません。

入力が `raw` である場合、デフォルトの出力は `raw sparse` になります。出力イメージのサイズは、スパース状態を認識するツールで確認する必要があります。

```
$ ls -lh test1.img
-rw-rw-r--. 1 rjones rjones 100M Aug  8 08:08 test1.img
$ du -sh test1.img
```



```
3.6M  test1.img
```

**ls** コマンドはイメージサイズが**100M**であることを示すことに注意してください。ただし、**du** コマンドはイメージサイズが**3.6M**であると正しく表示します。

### 重要な制限事項

以下は重要な制限事項の一覧になります。

- **virt-sparsify** を使用する前に、仮想マシンをシャットダウンしている必要があります。
- 最も悪いシナリオでは、**virt-sparsify** はソースディスクイメージの仮想サイズの2倍のサイズを要求する可能性があります。一時的なコピー分と移行先イメージ分が必要になる可能性があります。
- **--in-place** オプションを使用する場合、大規模な一時的な領域は不要になります。
- **virt-sparsify** を使用してディスクイメージのサイズを変更することはできません。ディスクイメージのサイズを変更するには、**virt-resize** を使用します。**virt-resize** についての詳細は、「[virt-resize: オフラインのゲスト仮想マシンのサイズ変更](#)」を参照してください。
- **virt-sparsify** は、暗号化されたディスクをスパース化できないため、暗号化されたディスクには使用できません。
- **virt-sparsify** はパーティション間の領域をスパースにすることができません。この領域はブートローダーなどの重要なアイテムに使用されることが多いため、これを未使用の領域と言うことはできません。
- **copy** モードでは、**qcow2** 内部スナップショットは移行先イメージにコピーされません。

### 例

**virt-sparsify** をインストールするには、以下のコマンドのいずれかを実行します。

```
# yum install /usr/bin/virt-sparsify
```

または

```
# yum install libguestfs-tools-c
```

ディスクをスパース化するには、以下を実行します。

```
# virt-sparsify /dev/sda1 /dev/device
```

**/dev/sda1** の内容を **/dev/device** にコピーし、出力をスパースにします。**/dev/device** がすでに存在する場合、これは上書きされます。**/dev/sda1** の形式は検出され、**/dev/device** の形式として使用されます。

形式間の変換を実行するには、以下を実行します。

```
# virt-sparsify disk.raw --convert qcow2 disk.qcow2
```

ソースディスクイメージ内で検索できるすべてのファイルシステムの空き領域のゼロ化およびスパース化を試行する。

特定のファイルシステムで空き領域がゼロで上書きされるのを防ぐには、以下を実行します。

```
# virt-sparsify --ignore /dev/device /dev/sda1 /dev/device
```

ファイルシステムの空き領域をゼロで上書きせずに、ディスクイメージのすべてのファイルシステムのスパース化されたディスクイメージを作成する。

一時的なコピーを作成せずにディスクイメージをスパースにするには、以下を実行します。

```
# virt-sparsify --in-place disk.img
```

指定されたディスクイメージをスパースにし、イメージファイルを上書きする。

### virt-sparsify オプション

以下のコマンドオプションは、**virt-sparsify** で使用できます。

表22.4 virt-sparsify オプション

コマンド	説明	例
--help	特定のコマンドまたはvirt-sparsify パッケージ全体についての簡潔なヘルプエントリを表示します。ヘルプの詳細は、virt-sparsify の man ページを参照してください。	\$ virt-sparsify --help
--check-tmpdir ignore continue warn fail	<p>tmpdir に操作を完了するのに十分な領域があるかどうかを推定します。指定したオプションは、操作の完了に必要な領域が十分でない場合の動作を決定します。</p> <ul style="list-style-type: none"> <li>● <b>ignore</b>: 問題は無視され、操作を継続します。</li> <li>● <b>continue</b>: エラーを報告し、操作を継続します。</li> <li>● <b>warn</b>: エラーを報告し、ユーザーが Enter を押すのを待機します。</li> <li>● <b>fail</b>: エラーを報告し、操作を中止します。</li> </ul> <p>このオプションは <b>--in-place</b> オプションと併用できません。</p>	<p>\$ virt-sparsify --check-tmpdir ignore /dev/sda1 /dev/device</p> <p>\$ virt-sparsify --check-tmpdir continue /dev/sda1 /dev/device</p> <p>\$ virt-sparsify --check-tmpdir warn /dev/sda1 /dev/device</p> <p>\$ virt-sparsify --check-tmpdir fail /dev/sda1 /dev/device</p>
--compress	出力ファイルを圧縮します。これは出力形式が qcow2 の場合にのみ機能します。このオプションは <b>--in-place</b> オプションと併用できません。	\$ virt-sparsify --compress /dev/sda1 /dev/device

コマンド	説明	例
--convert	<p>指定された形式を使用してスパースイメージを作成します。形式が指定されない場合、入力形式が使用されます。</p> <p>raw、qcow、vdi の出力形式がサポートされ、機能することが確認されています。</p> <p>qemu-img でサポートされるすべての形式を使用することができ、QEMU エミュレーターでサポートされるすべての形式を使用できます。</p> <p><b>--convert</b> オプションを使用することが推奨されています。これにより、<b>virt-sparsify</b> は入力形式を予測する必要がありません。</p> <p>このオプションは <b>--in-place</b> オプションと併用できません。</p>	<pre>\$ virt-sparsify -- convert raw /dev/sda1 /dev/device</pre> <pre>\$ virt-sparsify -- convert qcow2 /dev/sda1 /dev/device</pre> <pre>\$ virt-sparsify -- convert other_format indisk outdisk</pre>
--format	<p>入力ディスクイメージの形式を指定します。指定されていない場合、形式はイメージから検出されます。信頼されていない raw 形式のゲストディスクイメージが使用されている場合は、必ず形式を指定してください。</p>	<pre>\$ virt-sparsify -- format raw /dev/sda1 /dev/device</pre> <pre>\$ virt-sparsify -- format qcow2 /dev/sda1 /dev/device</pre>
--ignore	<p>指定されたファイルシステムまたはボリュームグループを無視します。</p> <p>ファイルシステムが指定されているが <b>--in-place</b> オプションが指定されていない場合、ファイルシステム上の空き領域はゼロ化されません。ただし、ゼロの既存ブロックはスパースになります。<b>--in-place</b> オプションが指定されている場合、ファイルシステムは完全に無視されます。</p> <p>ボリュームグループが指定されている場合、そのボリュームグループは無視されます。ボリュームグループ名は <b>/dev/</b> プレフィックスのない状態で使用する必要があります。たとえば、<b>--ignore vg_foo</b> のようになります。</p> <p><b>--ignore</b> オプションは、コマンドで複数回指定することができます。</p>	<pre>\$ virt-sparsify -- ignore filesystem1 /dev/sda1 /dev/device</pre> <pre>\$ virt-sparsify -- ignore volume_group/dev/sda1 /dev/device</pre>

コマンド	説明	例
<code>--in-place</code>	<p>一時的なコピーを作成するのではなく、イメージをインプレースでスパースにします。インプレースのスパース化はスパースコピー操作よりも効率的ですが、スパースコピーの場合ほどディスク領域を回復することができません。インプレースのスパース化は、破棄（またはトリムないしはマップ解除）サポートの使用により機能します。</p> <p>インプレースのスパース化を使用するには、インプレースでスパース化されるディスクイメージを指定します。</p> <p>インプレースのスパース化を指定する場合に、以下のオプションを使用することはできません。</p> <ul style="list-style-type: none"><li>● <b>--convert</b> および <b>--compress</b>: 大規模なディスク形式の変更が必要になるため。</li><li>● <b>--check-tmpdir</b>: 大規模な一時的な領域が不要であるため。</li></ul>	<pre>\$virt-sparsify --in-place disk.img</pre>
<code>-x</code>	<p><code>libguestfs</code> API 呼び出しのトレースを有効にします。</p>	<pre>\$virt-sparsify -x filesystem1 /dev/sda1 /dev/device</pre>

追加のオプションを含む詳細は、[libguestfs.org](https://libguestfs.org) を参照してください。

## 第23章 ゲスト仮想マシン管理の汎用ユーザーインターフェースツール

**virt-manager** に加えて、Red Hat Enterprise Linux 7 は、ゲスト仮想マシンのコンソールへのアクセスを可能にする以下のツールを提供しています。

### 23.1. VIRT-VIEWER

**virt-viewer** は、ゲスト仮想マシンのグラフィカルコンソールを表示するための最小限のコマンドラインユーティリティです。このコンソールは **VNC** または **SPICE** プロトコルを使用してアクセスされます。ゲストは、その名前、ID、または **UUID** に基づいて参照されます。ゲストが稼働していない場合、ゲストが起動するまで待機してからコンソールへの接続を試行するよう、ビューアーに指示できます。ビューアーはリモートホストに接続してコンソール情報を取得し、さらに同じネットワークトランスポートを使用してリモートコンソールに接続することができます。

**virt-manager** と比較すると、**virt-viewer** が提供する機能セットは小規模になりますが、その代わりにリソースの必要がより少なくなります。さらに **virt-manager** とは異なり、**virt-viewer** はほとんどの場合に **libvirt** への読み取り/書き込み権限はありません。したがって、権限を持たないユーザーもこれを使用して、ゲストにアクセスし、表示できますが、ゲストを設定することはできません。

**virt-viewer** をインストールするには、以下を実行します。

```
# sudo yum install virt-viewer
```

#### 構文

基本的な **virt-viewer** コマンドライン構文は以下のようになります。

```
# virt-viewer [OPTIONS] {guest-name|id|uuid}
```

**virt-viewer** で使用できるオプションの詳細の一覧については、**virt-viewer man** ページを参照してください。

#### ゲスト仮想マシンへの接続

オプションを指定せずに使用される場合、**virt-viewer** はローカルシステムのデフォルトのハイパーバイザーで接続できるゲストを一覧表示します。

デフォルトハイパーバイザーを使用する指定されたゲスト仮想マシンに接続するには、以下を実行します。

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

**KVM-QEMU** ハイパーバイザーを使用するゲスト仮想マシンに接続するには、以下を実行します。

```
# virt-viewer --connect qemu:///system guest-name
```

**TLS** を使用してリモートコンソールに接続するには、以下を実行します。

```
# virt-viewer --connect xen://example.org/ guest-name
```

**SSH** を使用してリモートホストのコンソールに接続するには、ゲスト設定を参照してから、コンソールへの直接のトンネル化されない接続を行います。

```
# virt-viewer --direct --connect xen+ssh://root@example.org/ guest-name
```

## ■ インタフェース

デフォルトで、**virt-viewer** インターフェースはゲストとの対話に使用する基本的なツールのみを提供します。

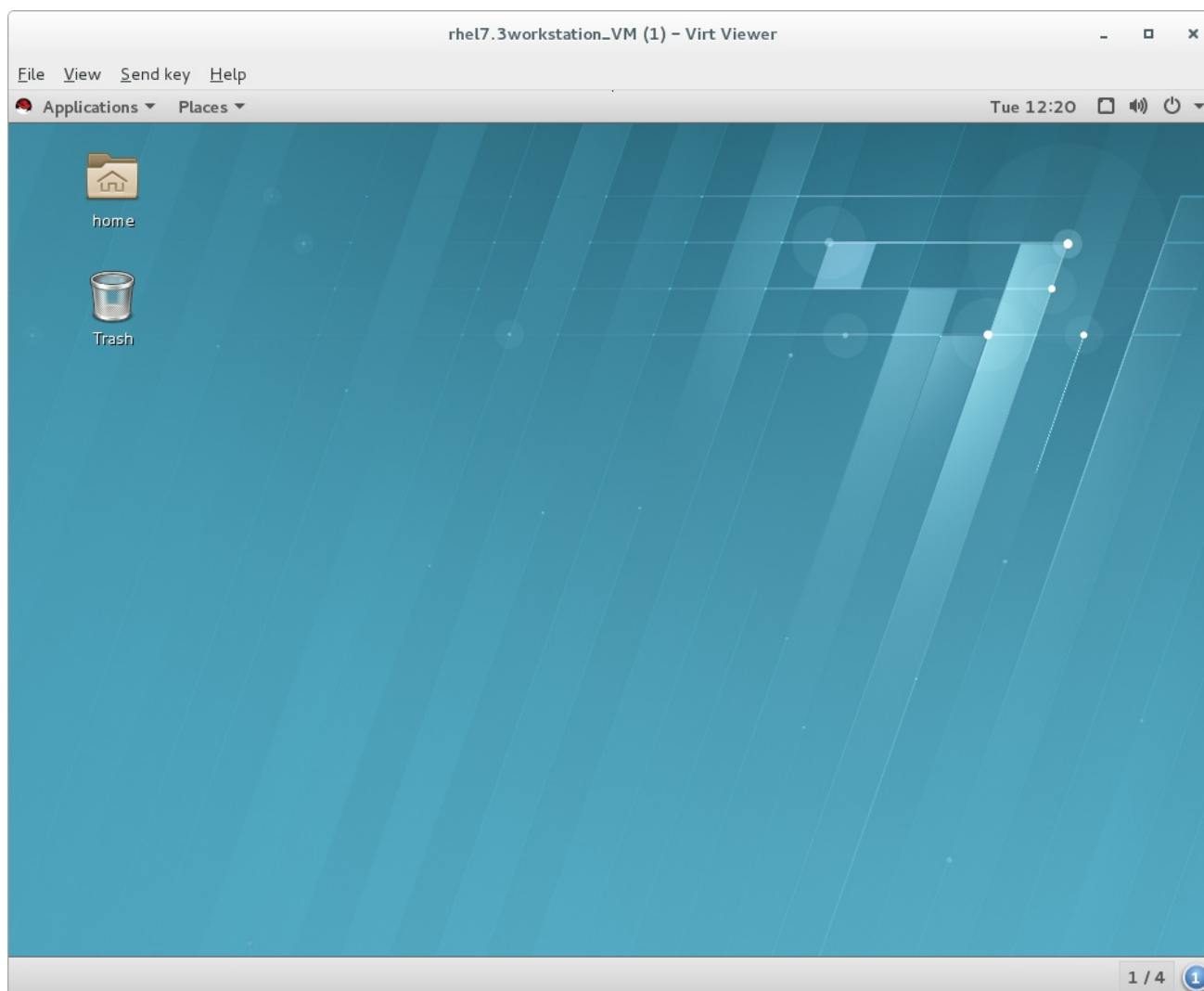


図23.1 virt-viewer インターフェースの例

### ホットキーの設定

**virt-viewer** セッションのカスタマイズされたキーボードのショートカット (またはホットキー) を作成するには、**--hotkeys** オプションを使用します。

```
# virt-viewer --hotkeys=action1=key-combination1[,action2=key-combination2] guest-name
```

以下のアクションをホットキーに割り当てることができます。

- toggle-fullscreen
- release-cursor
- smartcard-insert
- smartcard-remove

キー名の組み合わせホットキーは大/小文字を区別しません。ホットキーの設定は将来の **virt-viewer** セッションに引き継がれないことに注意してください。

### 例23.1 virt-viewer ホットキーの設定

**testquest** という KVM-QEMU ゲストに接続する際にフルスクリーンモードに切り換えるためのホットキーを追加するには、以下を実行します。

```
# virt-viewer --hotkeys=toggle-fullscreen=shift+f11 qemu:///system
testquest
```

### キオスクモード

キオスクモードでは、**virt-viewer** はユーザーの接続されたデスクトップとの対話のみを許可し、ゲストがシャットダウンされていない限り、ゲストの設定やホストシステムと対話するオプションを提供していません。これは、管理者がユーザーの動作範囲を指定されたゲストに制限する必要がある場合に役立ちます。

キオスクモードを使用するには、**-k** または **--kiosk** オプションを使用してゲストに接続します。

### 例23.2 キオスクモードでの virt-viewer の使用

マシンのシャットダウン後に終了する KVM-QEMU 仮想マシンにキオスクモードで接続するには、以下のコマンドを使用します。

```
# virt-viewer --connect qemu:///system guest-name --kiosk --kiosk-quit
on-disconnect
```

ただしキオスクモードのみでは、ゲストのシャットダウン後にユーザーがホストシステムやゲストの設定と対話しないことを確認することができないことに注意してください。これには、ホスト上で **Window Manager** を無効にするなどの追加のセキュリティ対策が必要になります。

## 23.2. REMOTE-VIEWER

**remote-viewer** は、SPICE および VNC をサポートする単純なリモートデスクトップディスプレイクライアントです。これが持つ機能および制限の大半は、**virt-viewer** と共通しています。

ただし、**virt-viewer** とは異なり、**remote-viewer** はリモートゲストのディスプレイに接続する際に **libvirt** を必要としません。そのため、**remote-viewer** は **libvirt** との対話または **SSH** 接続の使用に必要な権限を提供しないリモートホスト上の仮想マシンに接続する場合などに使用できます。

**remote-viewer** ユーティリティをインストールするには、以下を実行します。

```
# sudo yum install virt-viewer
```

### 構文

基本的な **remote-viewer** コマンドライン構文は以下のようになります。

```
# remote-viewer [OPTIONS] {guest-name|id|uuid}
```

**remote-viewer** で使用できるオプションの詳細の一覧については、**remote-viewer man** ページを参照してください。

### ゲスト仮想マシンへの接続

オプションを指定せずに使用される場合、**remote-viewer** はローカルシステムのデフォルトの URI で接続できるゲストを一覧表示します。

**remote-viewer** を使用して特定のゲストに接続するには、VNC/SPICE URI を使用します。URI の取得方法についての詳細は、「[グラフィカル表示に接続するための URI を表示](#)」を参照してください。

#### 例23.3 SPICE を使用したゲストディスプレイへの接続

以下を使用して、「**testquest**」というマシン上の SPICE サーバーに接続します。ここでは SPICE 通信にポート **5900** が使用されています。

```
# remote-viewer spice://testquest:5900
```

#### 例23.4 VNC を使用したゲストディスプレイへの接続

以下を使用して、**testquest2** というマシン上の VNC サーバーに接続します。ここでは VNC 通信にポート **5900** が使用されます。

```
# remote-viewer vnc://testquest2:5900
```

### インタフェース

デフォルトで、**remote-viewer** インターフェースはゲストとの対話に使用する基本的なツールのみを提供します。



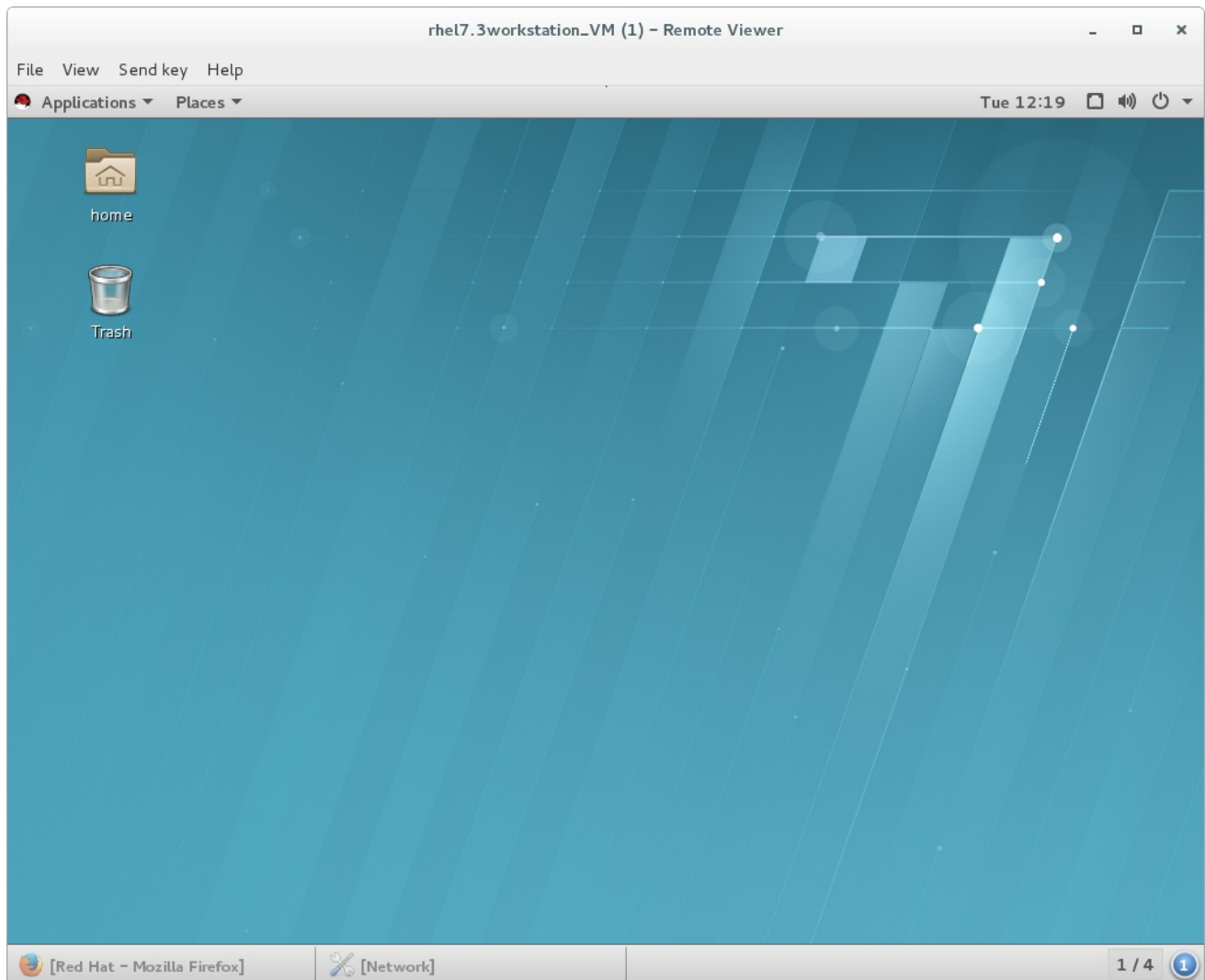


図23.2 remote-viewer インターフェースの例

## 23.3. GNOME BOXES

**Boxes** は仮想マシンおよびリモートシステムを表示し、これらにアクセスするために使用される簡易なグラフィカルデスクトップ仮想化ツールです。

**virt-viewer** および **remote-viewer** とは異なり、**Boxes** はゲスト仮想マシンの表示だけでなく、**virt-manager** と同様にそれらの作成および設定を可能にします。ただし、**virt-manager** と比較した場合に **Boxes** が提供する管理オプションと機能の数は少なくなりますが、それらの使用はより簡単になります。

**Boxes** をインストールするには、以下を実行します。

```
# sudo yum install gnome-boxes
```

アプリケーション ⇒ システムツール から **Boxes** を開きます。

メイン画面には、利用可能なゲスト仮想マシンが表示されます。画面の右側には 2 つのボタンがあります。

- 
 名前でゲスト仮想マシンを検索するための検索ボタン

-  選択ボタン

選択ボタンをクリックすると、1つ以上のゲスト仮想マシンを選択して、個別またはグループで操作を実行できます。選択可能な操作は、画面下部の操作バーに表示されます。



図23.3 操作バー

実行できる操作には以下の4つがあります。

- **Favorite:** 選択したゲスト仮想マシンにハートを追加し、これらをゲスト一覧の先頭に移動します。これはゲスト数が増大するほど便利になります。
- **Pause:** 選択したゲスト仮想マシンは実行を停止します。
- **Delete:** 選択したゲスト仮想マシンを削除します。
- **Properties:** 選択したゲスト仮想マシンのプロパティを表示します。

メイン画面の左側にある **New** ボタンを使って新規ゲスト仮想マシンを作成します。

### 手順23.1 Boxes を使用した新規ゲスト仮想マシンの作成

1. **New** をクリックします。  
これにより、**Introduction** 画面が開きます。**Continue** をクリックします。

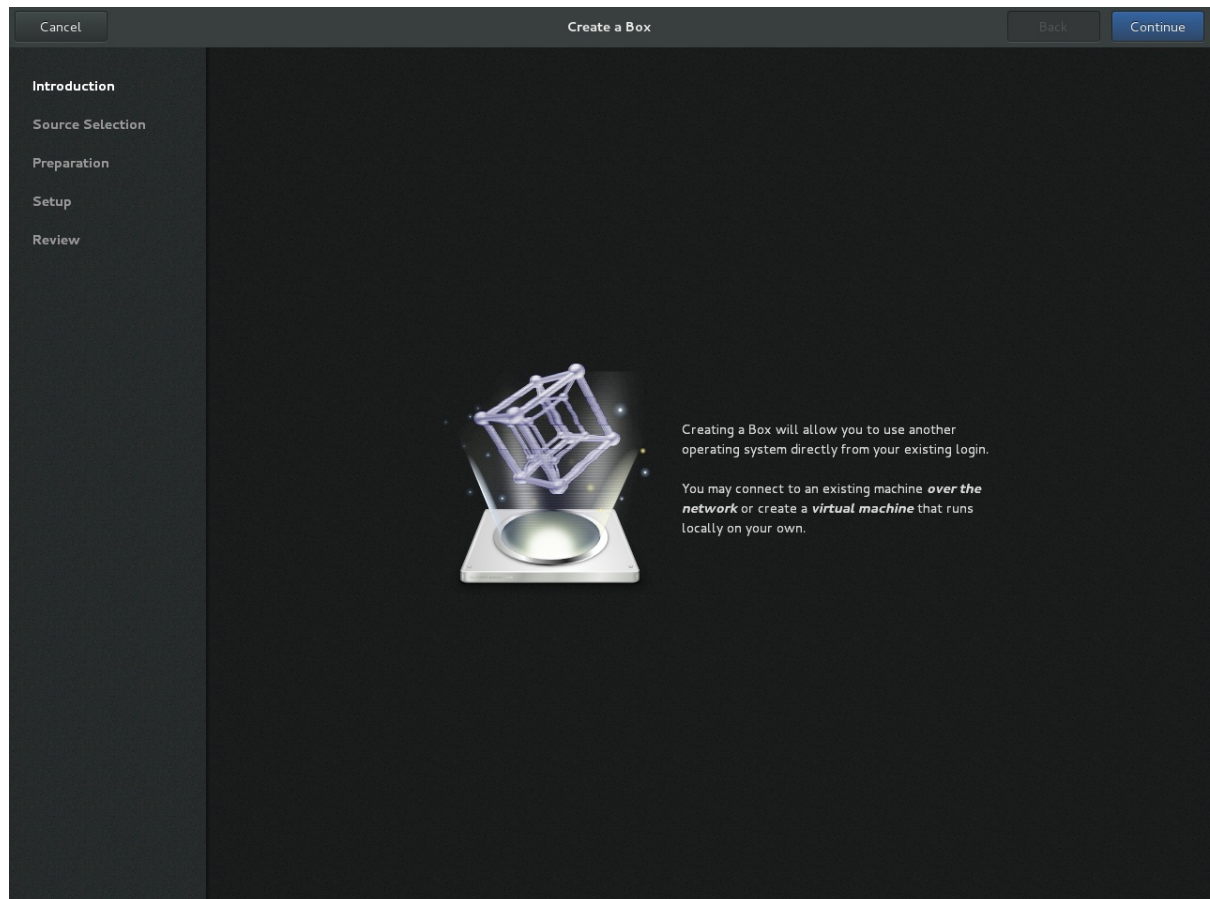


図23.4 Introduction 画面

## 2. ソースを選択します。

**Source Selection** 画面には 3 つのオプションがあります。

- **Available media:** すぐに利用できるインストールメディアがここに表示されます。これらのいずれかをクリックすると、**Review** 画面に直接移動します。
- **Enter a URL:** ローカル URI または ISO ファイルへのパスを指定するために URL を入力します。これはリモートマシンにアクセスするために使用することもできます。アドレスは ***protocol://IPaddress?port;*** のパターンに従う必要があります。以下が例になります。

```
spice://192.168.122.1?port=5906;
```

プロトコルは **spice://**、**qemu://**、または **vnc://** にすることができます。

- **Select a file:** インストールメディアを手動で検索するためにファイルディレクトリーを開きます。

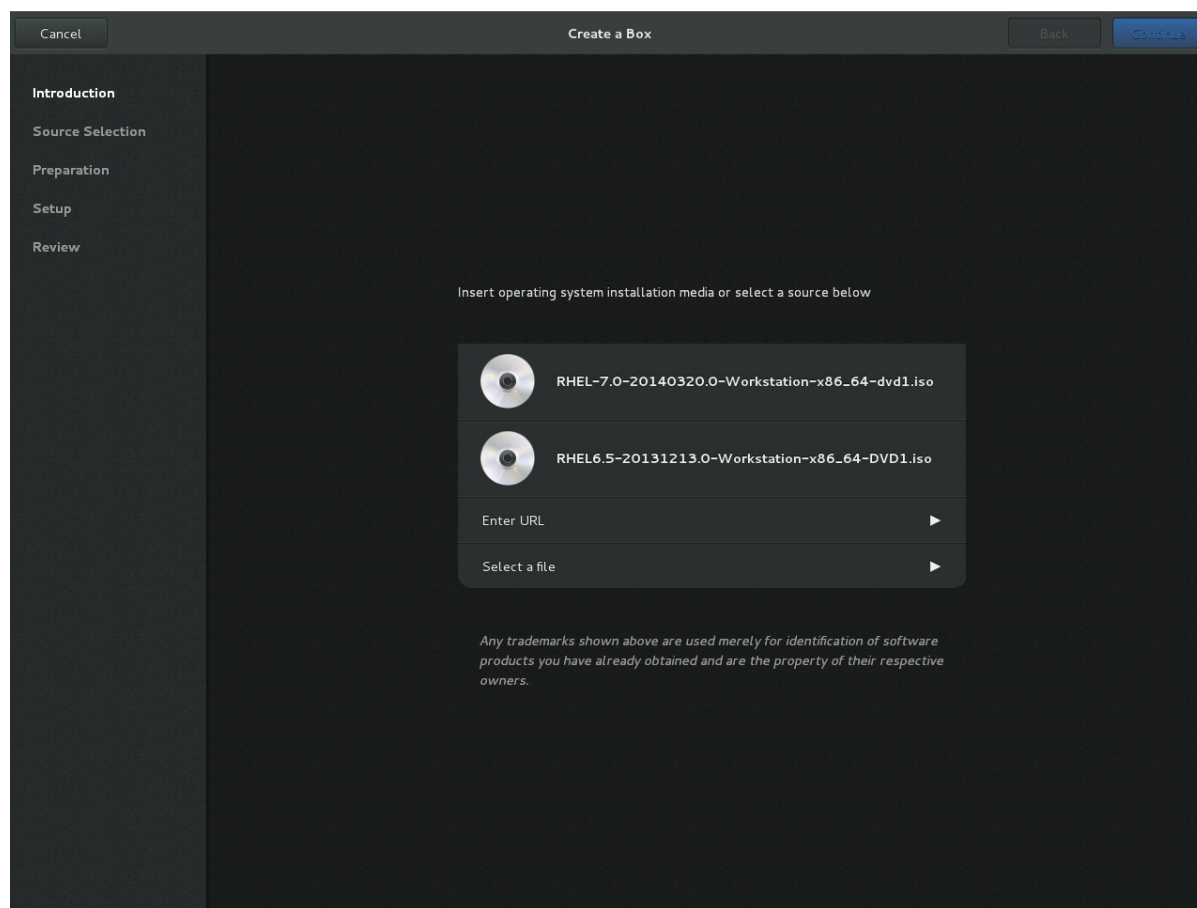


図23.5 ソース選択画面

### 3. 詳細を確認します

**Review** 画面は、ゲスト仮想マシンの詳細を表示します。

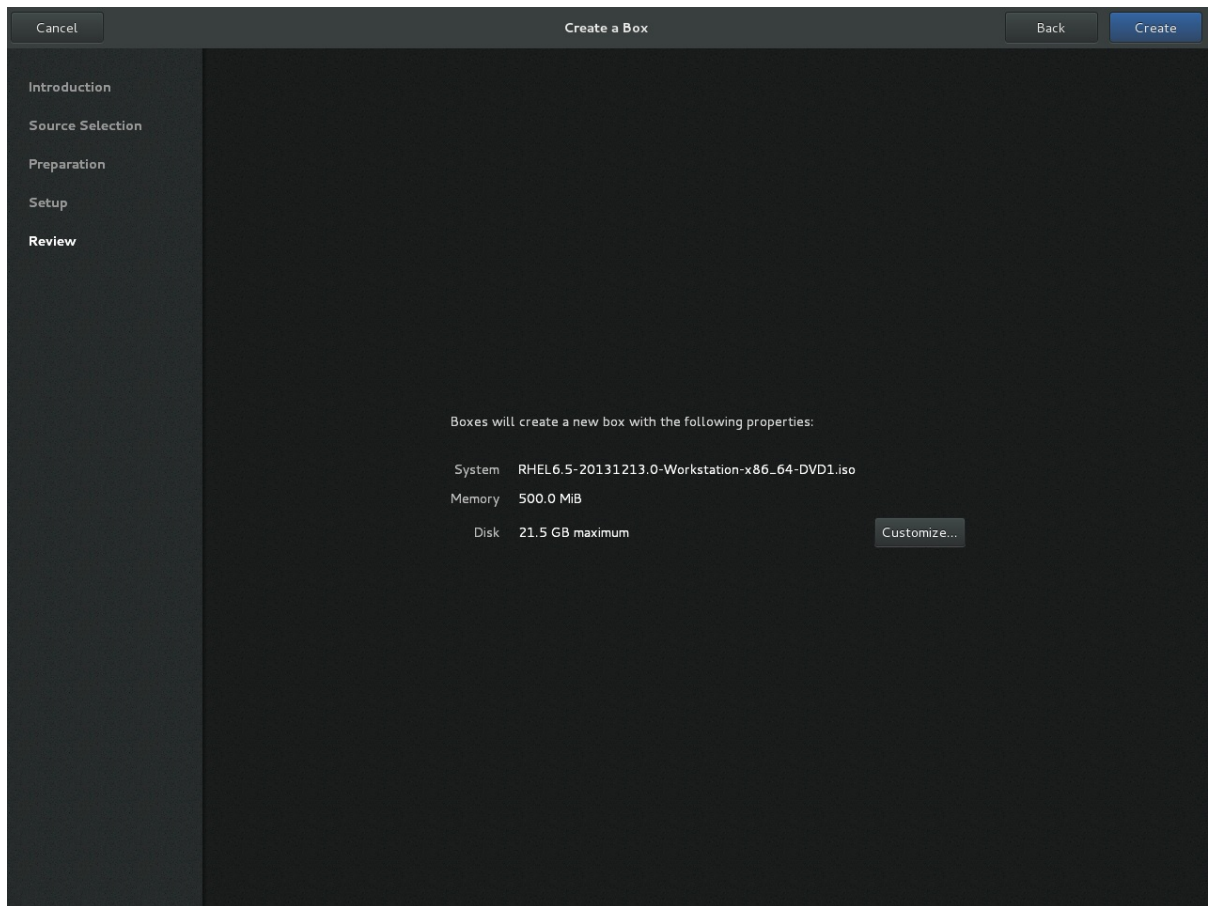


図23.6 確認画面

これらの詳細情報はそのまま残ります。この場合は最終ステップに進むか、または以下を実行します。

4. オプション: 詳細情報をカスタマイズします。

**Customize** をクリックすると、メモリーおよびディスクサイズなどの、ゲスト仮想マシンの設定を調整することができます。

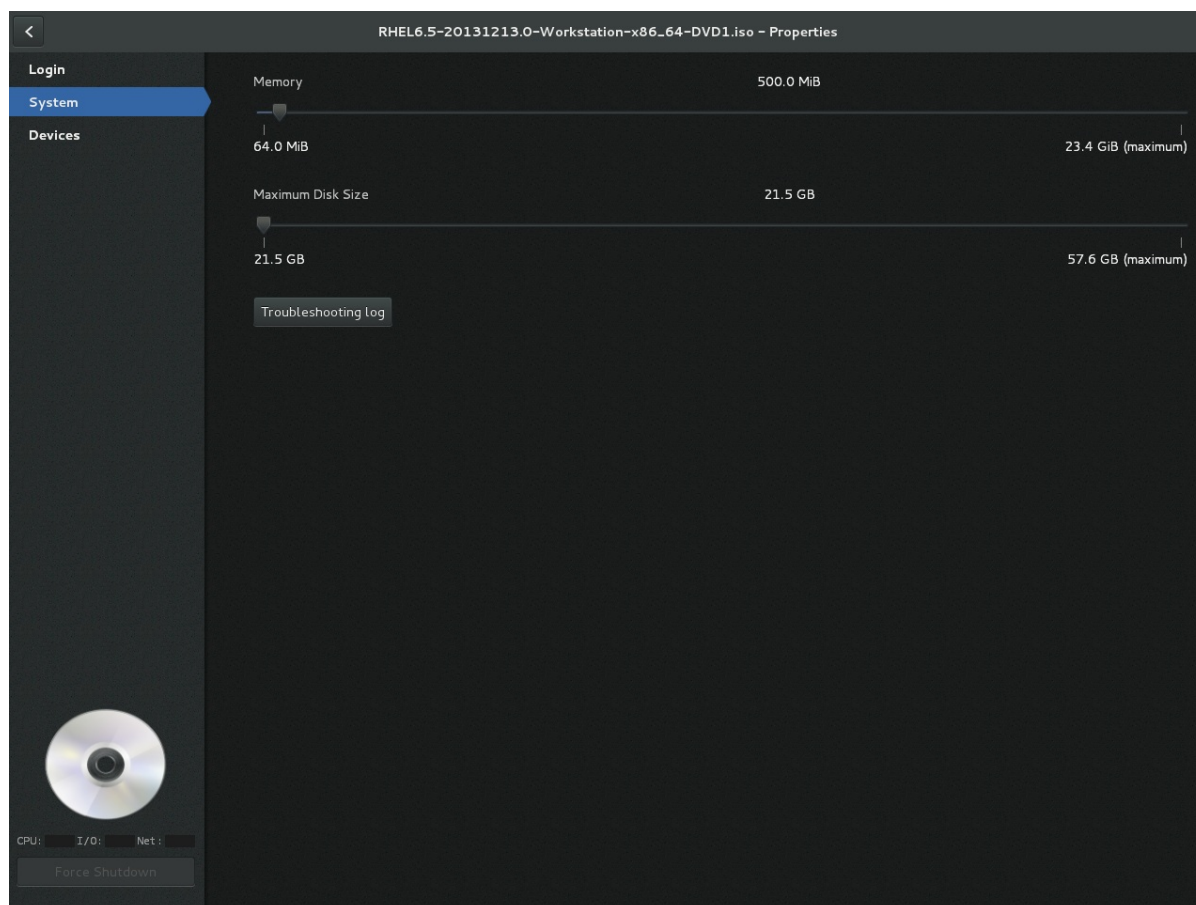


図23.7 カスタマイズ画面

## 5. 作成します。

**Create** をクリックします。新規のゲスト仮想マシンが開きます。



## 第24章 ドメイン XML の操作

この章では、ゲスト仮想マシン XML 設定ファイル(またはドメイン XML)のコンポーネントについて詳細に説明します。この章では、ドメインという用語は、すべてのゲスト仮想マシンに必要な `root` の `<domain>` 要素を指します。ドメイン XML には、`type` と `id` の 2 つの属性があります。`type` はドメインを実行するために使用されるハイパーバイザーを指定します。許可される値はドライバーに固有の値ですが、`KVM` およびその他の値が含まれます。`id` は実行中のゲスト仮想マシンの固有の整数識別子です。非アクティブなマシンには `id` 値がありません。本章のこのセクションではドメイン XML の各種コンポーネントについて説明します。本書の他の章は、ドメイン XML の操作に関連して本章に言及する場合があります。



### 重要

ドメイン XML ファイルのコンポーネントを編集するには、サポートされている管理インターフェース(`virsh` など)およびコマンド(`virsh edit` など)のみを使用します。`vim` または `gedit` などのテキストエディターでドメイン XML ファイルを直接開いたり、編集したりしないでください。

### 24.1. 一般的な情報およびメタデータ

該当する情報はドメイン XML の以下の部分にあります。

```
<domain type='kvm' id='3'>
  <name>fv0</name>
  <uuid>4dea22b31d52d8f32516782e98ab3fa0</uuid>
  <title>A short description - title - of the domain</title>
  <description>A human readable description</description>
  <metadata>
    <app1:foo xmlns:app1="http://app1.org/app1/">..</app1:foo>
    <app2:bar xmlns:app2="http://app1.org/app2/">..</app2:bar>
  </metadata>
  ...
</domain>
```

図24.1 ドメイン XML メタデータ

ドメイン XML のこのセクションを構成するコンポーネントは以下の通りです。

表24.1 一般的なメタデータの要素

要素	説明
<code>&lt;name&gt;</code>	仮想マシンの名前を割り当てます。この名前には英数字のみを使用する必要があり、単一ホスト物理マシン内で固有なものにする必要があります。多くの場合、永続的な設定ファイルを格納するためにファイル名を作成する際に使用されます。

要素	説明
<uuid>	仮想マシンのグローバルに固有となる識別子を割り当てます。形式は、たとえば <b>3e3fce45-4f53-4fa7-bb32-11f34168b82b</b> のように RFC 4122 に準拠している必要があります。新規マシンを削除/作成する場合にこれが省略される場合、ランダムな UUID が生成されます。さらに、 <b>sysinfo</b> 仕様で UUID を指定することもできます。
<title>	ドメインの簡単な説明用のスペースが作成されます。タイトルには改行を含めることができません。
<description>	<b>title</b> とは異なり、このデータは <b>libvirt</b> では全く使用されず、ユーザーが表示するよう選択するすべての情報を含めることができます。
<metadata>	カスタムメタデータを XML ノード/ツリーの形式で格納するためにアプリケーションで使用されます。アプリケーションは、XML ノード/ツリーのカスタム名前空間を使用する必要があります。この場合、1つの名前空間に1つのトップレベル要素のみがあります (アプリケーションに構造が必要な場合は、名前空間の要素にサブ要素を持たせる必要があります)。

24.2. オペレーティングシステムの起動

仮想マシンを起動する方法には多くの異なる方法があります。これには、BIOS ブートローダー、ホスト物理マシンブートローダー、直接のカーネルの起動、およびコンテナによる起動が含まれます。

24.2.1. BIOS ブートローダー

BIOS を起動する方法は、完全仮想化に対応するハイパーバイザーで利用できます。この場合、BIOS には起動順序の優先順位 (フロッピー、ハードディスク、CD-ROM、ネットワーク) があり、ブートイメージの検索方法が決定されます。ドメイン XML の <os> セクションには、以下のような情報が含まれます。

```
...
<os>
  <type>hvm</type>
  <boot dev='fd' />
  <boot dev='hd' />
  <boot dev='cdrom' />
  <boot dev='network' />
  <bootmenu enable='yes' />
  <smbios mode='sysinfo' />
  <bios useserial='yes' rebootTimeout='0' />
</os>
...
```

図24.2 BIOS ブートローダーのドメイン XML



ドメイン XML のこのセクションを構成するコンポーネントは以下の通りです。

表24.2 BIOS ブートローダーの各種要素

要素	説明
<b>&lt;type&gt;</b>	ゲスト仮想マシンで起動されるオペレーティングシステムのタイプを指定します。 <b>hvm</b> は、OS がベアメタルで実行されるように設計されたものであり、完全仮想化が必要であることを示します。 <b>linux</b> は KVM ハイパーバイザーのゲスト ABI に対応する OS を参照します。さらに、2つのオプション属性があり、 <b>arch</b> は仮想化に対する CPU アーキテクチャーを指定し、 <b>machine</b> はマシンタイプを指します。詳細は、『 <a href="#">libvirt アップストリームドキュメント</a> 』を参照してください。
<b>&lt;boot&gt;</b>	<b>fd</b> 、 <b>hd</b> 、 <b>cdrom</b> または <b>network</b> のいずれかの値を取る、考慮される次のブートデバイスを指定します。 <b>boot</b> 要素は、順番に試行するブートデバイスの優先順序の一覧をセットアップするために複数回繰り返すことができます。同じタイプの複数デバイスは、バスの順序を保持した状態で、それらのターゲットに基づいてソートされます。ドメインが定義された後に、libvirt によって <b>virDomainGetXMLDesc</b> から) 返される XML 設定は、ソートされた順序でデバイスを一覧表示します。ソート後は、最初のデバイスに <b>bootable</b> (起動可能) というマークが付けられます。詳細は、『 <a href="#">libvirt アップストリームドキュメント</a> 』を参照してください。
<b>&lt;bootmenu&gt;</b>	ゲスト仮想マシンの起動時のインタラクティブなブートメニューのプロンプトを有効にするかどうかを決定します。 <b>enable</b> 属性には <b>yes</b> または <b>no</b> のいずれかを使用できます。いずれも指定されない場合、ハイパーバイザーのデフォルトが使用されます。
<b>&lt;smbios&gt;</b>	SMBIOS 情報をゲスト仮想マシンで表示する方法を決定します。 <b>mode</b> 属性は、 <b>emulate</b> (ハイパーバイザーがすべての値を生成) か、または <b>host</b> (ホスト物理マシンの SMBIOS 値から、UUID を除くブロック 0 およびブロック 1 のすべての値をコピー。 <b>virConnectGetSysinfo</b> 呼び出しはコピーされた内容を表示するために使用できる)、または <b>sysinfo</b> ( <b>sysinfo</b> 要素の値を使用) のいずれかに指定される必要があります。いずれも指定されない場合、ハイパーバイザーのデフォルト設定が使用されます。

要素	説明
<bios>	この要素には、 <b>yes</b> または <b>no</b> の値が使用される可能性のある属性 <b>useserial</b> が含まれます。この属性は、ユーザーがシリアルポートに BIOS メッセージを表示することを可能にする <b>Serial Graphics Adapter</b> を有効/無効にします。そのため、シリアルポートを定義しておく必要があります。さらに、ブートが失敗した場合にゲスト仮想マシンが起動を再び開始するかどうかや、どの程度の時間が経過した後に再起動を開始するかなどを (BIOS に基づいて) 制御する <b>rebootTimeout</b> という別の属性もあります。値は、最大値が <b>65535</b> のミリ秒単位で設定され、値を <b>-1</b> に設定すると再起動が無効になります。

24.2.2. カーネルからの直接起動

新規ゲスト仮想マシンの OS をインストールする際に、ホスト物理マシン OS に格納されるカーネルと **initrd** から直接起動することが役立つ場合があります。これにより、コマンドライン引数がインストーラーに直接渡されます。この機能は通常、準仮想化および完全仮想化ゲスト仮想マシンで利用できます。

```
...
<os>
  <type>hvm</type>
  <kernel>/root/f8-i386-vmlinux</kernel>
  <initrd>/root/f8-i386-initrd</initrd>
  <cmdline>console=ttyS0 ks=http://example.com/f8-i386/os/</cmdline>
  <dtb>/root/ppc.dtb</dtb>
</os>
...
```

図24.3 カーネルからの直接起動

ドメイン XML のこのセクションを構成するコンポーネントは以下の通りです。

表24.3 カーネルから直接起動するための要素

要素	説明
<type>	BIOS ブートのセクションの説明と同様です。
<kernel>	ホスト物理マシン OS のカーネルイメージへの完全修飾パスを指定します。
<initrd>	ホスト物理マシン OS の (オプションの) ramdisk イメージへの完全修飾パスを指定します。

要素	説明
<code>&lt;cmdline&gt;</code>	起動時にカーネル(またはインストーラー)に渡される引数を指定します。これは、多くの場合、代替プライマリーコンソール(例: シリアルポート)、またはインストールメディアソース/キックスタートファイルを指定するために使用されます。

### 24.2.3. コンテナによる起動

カーネルまたはブートイメージの代わりにコンテナベースの仮想化を使用してドメインを起動する場合、**init** 要素を使用した **init** バイナリへのパスが必要です。デフォルトでは、これは引数を指定せずに起動されます。初期の **argv** を指定するには、**initarg** 要素を使用し、必要に応じて何度でも繰り返します。**cmdline** 要素は **/proc/cmdline** と同等の機能を提供しますが、**<initarg>** には影響を与えません。

```
...
<os>
  <type arch='x86_64'>exe</type>
  <init>/bin/systemd</init>
  <initarg>--unit</initarg>
  <initarg>emergency.service</initarg>
</os>
...
```

図24.4 コンテナによる起動

## 24.3. SMBIOS システム情報

一部のハイパーバイザーは、ゲスト仮想マシンにどのシステム情報を提供するかについての制御を可能にします(たとえば、SMBIOS フィールドにはハイパーバイザーによってデータが設定され、ゲスト仮想マシンの **dmidecode** コマンドで検査されます)。オプションの **sysinfo** 要素は、情報の以下のようなカテゴリーすべてを対象にします。

```
...
<os>
  <smbios mode='sysinfo' />
  ...
</os>
<sysinfo type='smbios'>
  <bios>
    <entry name='vendor'>LENOVO</entry>
  </bios>
  <system>
    <entry name='manufacturer'>Fedora</entry>
    <entry name='vendor'>Virt-Manager</entry>
  </system>
</sysinfo>
...
```

図24.5 SMBIOS システム情報

<sysinfo> 要素には、サブ要素のレイアウトを決める必須の属性 **type** があり、以下のように定義することができます。

- <mbios> - サブ要素は特定の SMBIOS 値を呼び出します。これは、<os> 要素の **mbios** サブ要素と共に使用されている場合にゲスト仮想マシンに影響を与えます。<sysinfo> のそれぞれのサブ要素は SMBIOS ブロックに名前を付け、それらの要素内にブロック内のフィールドを説明する **entry** 要素の一覧が入ることがあります。以下のブロックおよびエントリーが認識されます。
  - <bios> - これは SMBIOS のブロック 0 で、エントリー名は **vendor**、**version**、**date**、および **release** から取られます。
  - <system> - これは SMBIOS のブロック 1 で、エントリー名は **manufacturer**、**product**、**version**、**serial**、**uuid**、**sku**、および **family** から取られます。 **uuid** エントリーがトップレベルの **uuid** 要素と共に指定される場合、これらの 2 つの値は一致する必要があります。

## 24.4. CPU の割り当て

```
<domain>
...
<vcpu placement='static' cpuset="1-4,^3,6" current="1">2</vcpu>
...
</domain>
```

### 図24.6 CPU の割り当て

<vcpu> 要素は、ゲスト仮想マシン OS に割り当てられる仮想 CPU の最大数を定義します。この値は、1 からハイパーバイザーによってサポートされる最大数の間の値にする必要があります。この要素には、ドメインプロセスと仮想 CPU がデフォルトで固定される物理 CPU 数のコンマ区切りのリストである、オプションの **cpuset** 属性を含めることができます。

ドメインプロセスと仮想 CPU のピンニングポリシーは、**cputune** 属性を使用して別個に指定することができることに注意してください。<cputune> で指定される属性 **emulatorpin** が指定される場合、<vcpu> で指定される **cpuset** は無視されます。

同様に、**vcupin** の値を設定している仮想 CPU では、**cpuset** 設定は無視されます。**vcupin** が指定されていない仮想 CPU の場合、これは **cpuset** によって指定される物理 CPU に固定されます。**cpuset** 一覧の各要素は単一の CPU 番号、CPU 番号の範囲、キャレット (^) に続く直前範囲から除外される CPU 番号のいずれかになります。属性 **current** は、最大数より少ない数の仮想 CPU を有効にするかどうかを指定するために使用することができます。

オプションの属性 **placement** は、ドメインプロセスの CPU 配置モードを指定するために使用でき、その値は **static** または **auto** のいずれかにすることができます。これは、デフォルトで **placement** または **numatune** になるか、**cpuset** が指定されている場合は **static** になります。**auto** は、ドメインプロセスが **numad** の照会から返される **advisory** ノードセットに固定され、それが指定されている場合は属性 **cpuset** の値が無視されることを示します。**cpuset** と **placement** の両方が指定されていないか、または **placement** が **static** であるものの **cpuset** が指定されていない場合、ドメインプロセスはすべての利用可能な物理 CPU に固定されます。

## 24.5. CPU のチューニング

```
<domain>
...
<cputune>
  <vcupin vcpu="0" cpuset="1-4,^2"/>
  <vcupin vcpu="1" cpuset="0,1"/>
  <vcupin vcpu="2" cpuset="2,3"/>
  <vcupin vcpu="3" cpuset="0,4"/>
  <emulatorpin cpuset="1-3"/>
  <shares>2048</shares>
  <period>1000000</period>
  <quota>-1</quota>
  <emulator_period>1000000</emulator_period>
  <emulator_quota>-1</emulator_quota>
</cputune>
...
</domain>
```

図24.7 CPU のチューニング

すべてはオプションですが、ドメイン XML のこのセクションを構成するコンポーネントは以下のようになります。

表24.4 CPU チューニング要素

要素	説明
<cputune>	ドメインの CPU 調整可能パラメーターについての詳細を提供します。これはオプションです。
<vcupin>	ドメイン仮想 CPU が固定されるホスト物理マシンの物理 CPU を指定します。これが省略されていて、要素 <vcpu> の属性 cpuset が指定されていない場合、仮想 CPU はデフォルトですべての物理 CPU に固定されます。これには 2 つの属性が含まれ、属性 <vcpu> は id を指定し、属性 cpuset は、要素 <vcpu> の属性 cpuset と同じになります。
<emulatorpin>	ドメインのサブセットである「emulator」(vcpu を含まない)が固定されるホスト物理マシン CPU を指定します。これが省略されていて、要素 <vcpu> の属性 cpuset が指定されていない場合、「emulator」はデフォルトですべての物理 CPU に固定されます。これには、固定先となる物理 CPU を指定する 1 つの必須属性である cpuset が含まれます。要素 <vcpu> の属性 placement が auto の場合、emulatorpin は許可されません。

要素	説明
<b>&lt;shares&gt;</b>	ドメインの重み付け比例配分を指定します。これが省略されている場合、デフォルトは OS が指定するデフォルト値になります。値の単位がない場合、それは他の仮想ゲストマシンの設定との対比で計算されます。たとえば、ゲスト仮想マシンが <b>2048</b> の <b>&lt;shares&gt;</b> 値で設定されている場合、CPU 時間は <b>1024</b> の <b>&lt;shares&gt;</b> 値で設定されるゲスト仮想マシンの CPU 時間の <b>2</b> 倍になります。
<b>&lt;period&gt;</b>	施行間隔をマイクロ秒単位で指定します。 <b>&lt;period&gt;</b> を使用することにより、ドメインの <b>vcpu</b> のそれぞれには、実行時間に相当する割り当てられたクォータを超える量の消費が許可されません。この値は <b>1000 - 10000000</b> の範囲内にある必要があります。 <b>&lt;period&gt;</b> の後に値 <b>0</b> が来る場合は、値なしを意味します。
<b>&lt;quota&gt;</b>	マイクロ秒単位で許可される最大帯域幅を指定します。負の値の <b>&lt;quota&gt;</b> を持つドメインは、ドメインの帯域幅が無限であることを示し、つまりそれが帯域幅の制御が行われていないことを意味します。値は <b>1000 - 18446744073709551</b> の範囲内にするか、または <b>0</b> 未満にする必要があります。 <b>0</b> の値を持つ <b>quota</b> は値なしを意味します。この機能を使用して、すべての <b>vcpus</b> が同じ速度で実行されるようにすることができます。
<b>&lt;emulator_period&gt;</b>	施行間隔をマイクロ秒単位で指定します。 <b>&lt;emulator_period&gt;</b> 内で、ドメインのエミュレータスレッド (仮想 CPU を除く) は、実行時間に相当する <b>&lt;emulator_quota&gt;</b> を超える量を消費することが許可されません。 <b>&lt;emulator_period&gt;</b> 値は <b>1000 - 10000000</b> の範囲内にある必要があります。 <b>0</b> の値を持つ <b>&lt;emulator_period&gt;</b> は値なしを意味します。
<b>&lt;emulator_quota&gt;</b>	マイクロ秒単位でドメインのエミュレータスレッドに許可される最大帯域幅を指定します (仮想 CPU を除く)。負の値の <b>&lt;emulator_quota&gt;</b> を持つドメインは、ドメインがエミュレータスレッド (仮想 CPU を除く) の無限の帯域幅を持つことを示し、つまり、それが帯域幅の制御が行われていないことを意味します。値は <b>1000 - 18446744073709551</b> の範囲内にするか、または <b>0</b> 未満にする必要があります。 <b>0</b> の値を持つ <b>&lt;emulator_quota&gt;</b> は値なしを意味します。

## 24.6. メモリーのバッキング

メモリーのバッキングにより、ハイパーバイザーはゲスト仮想マシン内のラージページを適切に管理できます。

```
<domain>
...
<memoryBacking>
  <hugepages>
    <page size="1" unit="G" nodeset="0-3,5"/>
    <page size="2" unit="M" nodeset="4"/>
  </hugepages>
</memoryBacking>
...
</domain>
```

図24.8 メモリーのバッキング

`memoryBacking` 要素についての詳細は、[libvirt アップストリームドキュメント](#) を参照してください。

## 24.7. メモリーチューニング

```
<domain>
...
<memtune>
  <hard_limit unit='G'>1</hard_limit>
  <soft_limit unit='M'>128</soft_limit>
  <swap_hard_limit unit='G'>2</swap_hard_limit>
  <min_guarantee unit='bytes'>67108864</min_guarantee>
</memtune>
...
</domain>
```

図24.9 メモリーチューニング

`<memtune>` はオプションですが、ドメイン XML のこのセクションを構成するコンポーネントは以下のようになります。

表24.5 メモリーチューニング要素

要素	説明
----	----

要素	説明
<code>&lt;memtune&gt;</code>	ドメインのメモリー調整可能なパラメーターについての詳細を提供します。これが省略されている場合、デフォルトは OS が指定するデフォルト値になります。これらのパラメーターはプロセス全体に適用されます。そのため、制限を設定する場合は、ゲストマシンの RAM のゲスト仮想マシンのビデオ RAM への追加、およびいくらかのメモリーオーバーヘッドを可能にすることで値を決定します。それぞれの調整可能なパラメーターに対して <code>&lt;memory&gt;</code> の場合と同じ値を使用して入力の単位を指定することができます。後方互換の場合、出力は常に KiB 単位になります。
<code>&lt;hard_limit&gt;</code>	これは、ゲスト仮想マシンが使用できる最大メモリーです。この値は <b>キビバイト</b> (1024 バイトのブロック) で表わされます。
<code>&lt;soft_limit&gt;</code>	これは、メモリー競合中に強制するメモリーの制限です。この値はキビバイト (1024 バイトのブロック) で表わされます。
<code>&lt;swap_hard_limit&gt;</code>	これは、ゲスト仮想マシンが使用できる最大メモリーに <b>swap</b> を足したものです。この値はキビバイト (1024 バイトのブロック) で表わされます。これは、 <code>&lt;hard_limit&gt;</code> 値よりも大きくなければなりません。
<code>&lt;min_guarantee&gt;</code>	これは、ゲスト仮想マシンへの割り当てを保証できる最小メモリーです。この値はキビバイト (1024 バイトのブロック) で表わされます。

## 24.8. メモリーの割り当て

ゲスト仮想マシンがクラッシュする場合、オプションの属性 **`dumpCore`** を使用して、ゲスト仮想マシンのメモリーを生成されるコアダンプに含めるか (**`dumpCore='on'`**) か、または含めないか (**`dumpCore='off'`**) の制御を行うことができます。デフォルト設定は **`on`** になります。つまり、パラメーターが **`off`** に設定されていない限り、ゲスト仮想マシンのメモリーはコアダンプに含まれることになります。

**`currentMemory`** 属性でゲスト仮想マシンの実際のメモリー割り当てを確定します。ゲスト仮想マシンのメモリーバレーニングを随時許可するには、この値を最大割り当て値よりも小さくすることができます。この値の設定を省略すると、**`memory`** 要素と同じ値にデフォルト設定されます。単位の属性はメモリーの属性と同様に機能します。



```

<domain>

  <memory unit='KiB' dumpCore='off'>524288</memory>
  <!-- changes the memory unit to KiB and does not allow the guest virtual
machine's memory to be included in the generated coredump file -->
  <currentMemory unit='KiB'>524288</currentMemory>
  <!-- makes the current memory unit 524288 KiB -->
  ...
</domain>

```

図24.10 メモリー単位

## 24.9. NUMA ノードのチューニング

**virsh edit** を使用して NUMA ノードのチューニングが実行されると、以下のドメイン XML パラメーターが影響を受けます。

```

<domain>
  ...
  <numatune>
    <memory mode="strict" nodeset="1-4,^3"/>
  </numatune>
  ...
</domain>

```

図24.11 NUMA ノードのチューニング

すべてはオプションですが、ドメイン XML のこのセクションを構成するコンポーネントは以下のようになります。

表24.6 NUMA ノードのチューニング要素

要素	説明
<numatune>	ドメインプロセスの NUMA ポリシーを制御することによって NUMA ホスト物理マシンのパフォーマンスを調整する方法の詳細を提供します。

要素	説明
<memory>	NUMA ホスト物理マシンのドメインプロセスにメモリーを割り当てる方法を指定します。これには、複数のオプションの属性が含まれます。属性 <b>mode</b> は、 <b>interleave</b> 、 <b>strict</b> 、または <b>preferred</b> のいずれかになります。いずれの値も指定されない場合、デフォルトで <b>strict</b> になります。属性 <b>nodeset</b> 属性は、要素<vcpu>の属性 <b>cpuset</b> と同じ構文を使用して NUMA ノードを指定します。属性 <b>placement</b> は、ドメインプロセスのメモリー配置モードを指定するために使用できます。その値は <b>static</b> または <b>auto</b> のいずれかにすることができます。属性 <nodeset> が指定される場合、デフォルトで <vcpu> の<placement>になるか、または <b>static</b> になります。 <b>auto</b> は、numad の照会から返される advisory ノードセットからメモリーを割り当てるのみで、属性 <b>nodeset</b> の値は、それが指定されている場合は無視されます。 <b>vcpu</b> の属性 <b>placement</b> が <b>auto</b> であり、属性<numatune>が指定されていない場合、<placement> <b>auto</b> およびモード <b>strict</b> が指定されたデフォルトの<numatune> が暗黙的に追加されます。

24.10. ブロック I/O チューニング

```
<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>
```

図24.12 ブロック I/O チューニング

すべてはオプションですが、ドメイン XML のこのセクションを構成するコンポーネントは以下のようになります。

表24.7 ブロック I/O チューニング要素

要素	説明
<code>&lt;blkio tune&gt;</code>	このオプションの要素は、ドメインの <b>blkio cgroup</b> の調整可能パラメーターを調整する機能を提供します。これが省略されている場合、デフォルトは OS が指定するデフォルト値になります。
<code>&lt;weight&gt;</code>	このオプションの <b>weight</b> 要素はゲスト仮想マシンの全体の I/O ウェイト値になります。この値は 100 - 1000 の範囲内にある必要があります。
<code>&lt;device&gt;</code>	ドメインには、ドメインが使用するそれぞれのホスト物理マシンブロックデバイスのウェイトをさらに調整する複数の <b>&lt;device&gt;</b> 要素が含まれる場合があります。複数のゲスト仮想マシンディスクは単一のホスト物理マシンブロックデバイスを共有することに注意してください。さらに、それらが同じホスト物理マシンファイルシステム内のファイルでサポートされるため、このチューニングパラメーターは、ゲスト仮想マシンの各ディスクデバイスに関連付けられるのではなく、グローバルドメインレベルで実行されます (これは、単一 <b>&lt;disk&gt;</b> に適用される <b>&lt;iotune&gt;</b> 要素と対比できます)。それぞれの <b>device</b> 要素には、デバイスの絶対パスを記述する <b>&lt;path&gt;</b> と、許可される範囲が 100 から 1000 までのデバイスの相対的比率 (ウェイト) を指定する <b>&lt;weight&gt;</b> の 2 つの必須のサブ要素があります。

## 24.11. リソースパーティション作成

ハイパーバイザーでは、該当パーティションのネスト化によって、仮想マシンをリソースパーティションに配置できます。**<resource>** 要素は、リソースパーティション設定に関連する設定をグループ化し、現在ドメインを配置するリソースパーティションのパスを定義するコンテンツを持つ子要素をサポートしています。パーティションが一覧表示されない場合、ドメインはデフォルトのパーティションに配置されます。ゲスト仮想マシンを起動する前にパーティションを作成しておく必要があります。デフォルトでは (ハイパーバイザーに固有の) デフォルトパーティションのみが存在することが想定されます。

```
<resource>
  <partition>/virtualmachines/production</partition>
</resource>
```

### 図24.13 リソースパーティション作成

現在リソースパーティションは、パーティションパスをマウントされたすべてのコントローラー内の **cgroups** ディレクトリーにマップする **KVM** および **LXC** ドライバーによってサポートされています。

## 24.12. CPU モデルおよびトポロジー

このセクションでは、CPU モデルの要件について扱います。すべてのハイパーバイザーには、ゲストがデフォルトで表示する CPU 機能についての独自のポリシーがあることに注意してください。**KVM** によってゲストに提供される CPU 機能のセットは、ゲスト仮想マシン設定で選択される CPU モデルに

よって異なります。**qemu32** および **qemu64** は基本的な CPU モデルですが、他に利用できる他のモデル (追加の機能を含む) もあります。それぞれのモデルとそのトポロジーは、ドメイン XML の次の要素を使って指定されます。

```
<cpu match='exact'>
  <model fallback='allow'>core2duo</model>
  <vendor>Intel</vendor>
  <topology sockets='1' cores='2' threads='1' />
  <feature policy='disable' name='lahf_lm' />
</cpu>
```

図24.14 CPU モデルおよびトポロジーサンプル 1

```
<cpu mode='host-model'>
  <model fallback='forbid' />
  <topology sockets='1' cores='2' threads='1' />
</cpu>
```

図24.15 CPU モデルおよびトポロジーサンプル 2

```
<cpu mode='host-passthrough' />
```

図24.16 CPU モデルおよびトポロジーサンプル 3

CPU モデルまたはその機能への制限が設けられていない場合、以下のような単純な **<cpu>** 要素が使用される場合があります。

```
<cpu>
  <topology sockets='1' cores='2' threads='1' />
</cpu>
```

図24.17 CPU モデルおよびトポロジーサンプル 4

```
<cpu mode='custom'>
  <model>POWER8</model>
</cpu>
```

図24.18 PPC64/PSeries CPU モデルサンプル

```
<cpu mode='host-passthrough' />
```

図24.19 aarch64/virt CPU モデルサンプル

ドメイン XML のこのセクションを構成するコンポーネントは以下の通りです。

表24.8 CPU モデルおよびトポロジー要素

要素	説明
<b>&lt;cpu&gt;</b>	これは、ゲスト仮想マシンの CPU 要件について説明するための主なコンテナです。

要素	説明
<b>&lt;match&gt;</b>	<p>ゲスト仮想マシンに提供される仮想 CPU を各種要件にどの程度一致させるかを指定します。トポロジーが <b>&lt;cpu&gt;</b> 内の唯一の要素である場合、<b>match</b> 属性を省略することができます。<b>match</b> 属性に使用できる値は、以下になります。</p> <ul style="list-style-type: none"><li>• <b>minimum</b> - 指定した CPU モデルと機能は、必要最小限の CPU を記述します。</li><li>• <b>exact</b> - ゲスト仮想マシンに提供される仮想 CPU は仕様に完全に一致します。</li><li>• <b>strict</b> - ホスト物理マシンの CPU が仕様に完全に一致しない場合、ゲスト仮想マシンは作成されません。</li></ul> <p><b>match</b> 属性は省略でき、デフォルトで<b>exact</b> になることに注意してください。</p>

要素	説明
<mode>	<p>このオプションの属性は、ゲスト仮想マシン CPU をできるだけホスト物理マシン CPU に近づけるように設定しやすくするように使用できます。<b>mode</b> 属性に使用できる属性は以下になります。</p> <ul style="list-style-type: none"> <li> <b>custom - CPU</b> をゲスト仮想マシンに表示する方法を説明します。これは、<b>mode</b> 属性が指定されない場合のデフォルト設定です。このモードにより、ゲスト仮想マシンがどのホスト物理マシン上で起動されるのかにかかわらず、永続的なゲスト仮想マシンに同じハードウェアが表示されるようにします。 </li> <li> <b>host-model: capabilities XML</b> からドメイン XML にホスト物理マシンの CPU 定義をコピーするためのショートカットです。ドメインを起動する直前に CPU 定義がコピーされるので、同じ XML を異なる複数のホスト物理マシン上で使用することができます。その間、それぞれのホスト物理マシンがサポートする最適なゲスト仮想マシン CPU は依然として提供されます。<b>match</b> 属性も、いずれの機能要素もこのモードでは使用できません。詳細は、<a href="#">libvirt アップストリームの web サイト</a> を参照してください。 </li> <li> <b>host-passthrough:</b> このモードでは、ゲスト仮想マシンに表示される CPU は、libvirt 内でエラーを生じさせる要素を含め、ホスト物理マシン CPU の場合と全く同じになります。このモードの大きな短所は、ゲスト仮想マシン環境を異なるハードウェア上で再現できないことにあり、そのためこのモードは細心の注意を払って使用することが推奨されます。<b>model</b> または <b>feature</b> 要素もいずれもこのモードでは許可されません。 </li> <li> <b>host-model</b> と <b>host-passthrough</b> の両方のモードでは、現在のホスト物理マシンで使用される実際の (<b>host-passthrough</b> モードの値と近似する) CPU 定義は、<b>virDomainGetXMLDesc</b> API を呼び出す際に <b>VIR_DOMAIN_XML_UPDATE_CPU</b> フラグを指定することによって決定できます。異なるハードウェアが提供されると、オペレーティングシステムの再アクティブ化を引き起こす傾向があり、かつ異なる機能を持つ複数のホスト物理マシン間で移行するゲスト仮想マシンを実行する場合は、この出力を使用して、より堅牢な移行のために XML を <b>custom</b> モードに書き換えることができます。 </li> </ul>

要素	説明
<b>&lt;model&gt;</b>	ゲスト仮想マシンが要求する CPU モデルを指定します。利用可能な CPU モデルとそれらの定義の一覧は、libvirt のデータディレクトリーにインストールされた <b>cpu_map.xml</b> ファイルにあります。ハイパーバイザーが正確な CPU モデルを使用できない場合、libvirt は、CPU 機能の一覧を維持しつつ、ハイパーバイザーでサポートされる直近のモデルにフォールバックします。オプションの <b>fallback</b> 属性は、この動作を禁止するために使用できます。この場合、サポートされていない CPU モデルを要求するドメインを起動する試行は失敗します。 <b>fallback</b> 属性のサポートされている値は、 <b>allow</b> (デフォルト) と <b>forbid</b> です。オプションの <b>vendor_id</b> 属性は、ゲスト仮想マシンによって表示されるベンダー ID を設定するために使用できます。ID の長さは、ちょうど 12 文字分である必要があります。これが設定されていない場合、ホスト物理マシンのベンダー ID が使用されます。使用できる標準的な値は、 <b>AuthenticAMD</b> と <b>GenuineIntel</b> です。
<b>&lt;vendor&gt;</b>	ゲスト仮想マシンによって要求される CPU ベンダーを指定します。この要素がない場合、ゲスト仮想マシンは、そのベンダーの如何にかかわらず、指定された機能に一致する CPU 上で実行されます。サポートされるベンダーの一覧は <b>cpu_map.xml</b> にあります。
<b>&lt;topology&gt;</b>	ゲスト仮想マシンに提供される仮想 CPU の要求されるトポロジーを指定します。3 つのゼロ以外の値を、ソケット、コアおよびスレッドに指定する必要があります。それぞれは、CPU ソケットの合計数、1 ソケットあたりのコア数、1 コアあたりのスレッド数になります。

要素	説明
<feature>	<p>選択した CPU モデルによって提供される機能を微調整するために使用されるゼロまたは1つ以上の要素を含めることができます。既知の機能名の一覧は、<b>cpu_map.xml</b> ファイルにあります。それぞれの <b>feature</b> 要素の意味は、以下の値のいずれかに設定する必要のあるそのポリシー属性によって変わります。</p> <ul style="list-style-type: none"> <li>● <b>force</b> - 仮想マシンがホスト物理マシン CPU に実際にサポートされているかどうかにかかわらず、仮想マシンのサポートを強制します。</li> <li>● <b>require</b> - 機能がホスト物理マシンによってサポートされていない場合、ゲスト仮想マシンの作成が失敗するように指定します。これはデフォルトの設定です。</li> <li>● <b>optional</b> - この機能は仮想 CPU によってサポートされますが、サポートされるのは、ホスト物理マシン CPU によってサポートされている場合のみです。</li> <li>● <b>disable</b> - これは仮想 CPU によってサポートされません。</li> <li>● <b>forbid</b> - 機能がホスト物理マシン CPU によってサポートされている場合、ゲスト仮想マシンの作成は失敗します。</li> </ul>

### 24.12.1. 指定 CPU に設定された機能の変更

CPU モデルには継承された機能セットがありますが、個別の機能コンポーネントは機能ごとに機能上で許可するか、または禁止することができます。これにより、CPU のさらに個別化された設定が可能になります。

#### 手順24.1 CPU 機能の有効化および無効化

1. まず、ゲスト仮想マシンをシャットダウンします。
2. **virsh edit [domain]** コマンドを実行して、ゲスト仮想マシンの設定ファイルを開きます。
3. <feature> または <model> 内のパラメーターを、属性値 'allow' を組み込むように変更し、許可する機能を強制実行するか、または 'forbid' を組み込むように変更し、機能のサポートを否定します。



```

<!-- original feature set -->
<cpu mode='host-model'>
  <model fallback='allow'/>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>

<!--changed feature set-->
<cpu mode='host-model'>
  <model fallback='forbid'/>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>

```

図24.20 CPU 機能の有効化および無効化の例

```

<!--original feature set-->
<cpu match='exact'>
  <model fallback='allow'>core2duo</model>
  <vendor>Intel</vendor>
  <topology sockets='1' cores='2' threads='1'/>
  <feature policy='disable' name='lahf_lm'/>
</cpu>

<!--changed feature set-->
<cpu match='exact'>
  <model fallback='allow'>core2duo</model>
  <vendor>Intel</vendor>
  <topology sockets='1' cores='2' threads='1'/>
  <feature policy='enable' name='lahf_lm'/>
</cpu>

```

図24.21 CPU 機能の有効化および無効化の例 2

4. 変更が完了したら、設定ファイルを保存して、ゲスト仮想マシンを起動します。

### 24.12.2. ゲスト仮想マシンの NUMA トポロジー

ゲスト仮想マシンの NUMA トポロジーは、ドメイン XML の **<numa>** 要素を使用して指定できます。

```

<cpu>
  <numa>
    <cell cpus='0-3' memory='512000'/>
    <cell cpus='4-7' memory='512000'/>
  </numa>
</cpu>
...

```

図24.22 ゲスト仮想マシンの NUMA トポロジー

それぞれの **cell** 要素は NUMA セルまたは NUMA ノードを指定します。**cpus** は、ノードの一部である

CPU または CPU の範囲を指定します。**memory** はノードメモリーをキビバイト単位で指定します (例: 1024 バイトのブロック)。それぞれのセルまたはノードには、0 から始まる昇順で **cellid** または **nodeid** が割り当てられます。

## 24.13. イベント設定

ドメイン XML の以下のセクションを使用すると、各種のイベント用のデフォルトのアクションを上書きできます。

```
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<on_lockfailure>poweroff</on_lockfailure>
```

### 図24.23 イベント設定

以下の要素のコレクションを使用すると、ゲスト仮想マシン OS がライフサイクル操作をトリガーする際のアクションを指定することができます。一般的なユースケースには、初期の OS インストールの実行時に電源オフとして処理される再起動を強制する方法があります。これにより、VM をインストール後の最初のブート時に再設定することができます。

ドメイン XML のこのセクションを構成するコンポーネントは以下の通りです。

表24.9 イベント設定の要素

状態	説明
<b>&lt;on_poweroff&gt;</b>	<p>ゲスト仮想マシンが電源オフを要求する際に実行されるアクションを指定します。4 つの引数を使用できます。</p> <ul style="list-style-type: none"><li>● <b>destroy</b> - このアクションは、ドメインを完全に終了させ、すべてのリソースをリリースします。</li><li>● <b>restart</b> - このアクションは、ドメインを完全に終了させ、同じ設定でこれを再起動します。</li><li>● <b>preserve</b> - このアクションは、ドメインを完全に終了させますが、そのリソースは追加の分析ができるように保持されます。</li><li>● <b>rename-restart</b> - このアクションはドメインを完全に終了させてから、新しい名前前でこれを再起動します。</li></ul>

状態	説明
<b>&lt;on_reboot&gt;</b>	<p>ゲスト仮想マシンが再起動を要求する際に実行されるアクションを指定します。4つの引数を使用できます。</p> <ul style="list-style-type: none"> <li>● <b>destroy</b> - このアクションは、ドメインを完全に終了させ、すべてのリソースをリリースします。</li> <li>● <b>restart</b> - このアクションは、ドメインを完全に終了させ、同じ設定でこれを再起動します。</li> <li>● <b>preserve</b> - このアクションは、ドメインを完全に終了させますが、そのリソースは追加の分析ができるように保持されます。</li> <li>● <b>rename-restart</b> - このアクションはドメインを完全に終了させてから、新しい名前でこれを再起動します。</li> </ul>
<b>&lt;on_crash&gt;</b>	<p>ゲスト仮想マシンがクラッシュする際に実行されるアクションを指定します。さらに、これは以下の追加のアクションに対応します。</p> <ul style="list-style-type: none"> <li>● <b>coredump-destroy</b> - クラッシュしたドメインのコアはダンプされ、ドメインは完全に終了し、すべてのリソースはリリースされます。</li> <li>● <b>coredump-restart</b> - クラッシュしたドメインのコアはダンプされ、ドメインは同じ設定で再起動します。</li> </ul> <p>以下の4つの引数を使用できます。</p> <ul style="list-style-type: none"> <li>● <b>destroy</b> - このアクションは、ドメインを完全に終了させ、すべてのリソースをリリースします。</li> <li>● <b>restart</b> - このアクションは、ドメインを完全に終了させ、同じ設定でこれを再起動します。</li> <li>● <b>preserve</b> - このアクションは、ドメインを完全に終了させますが、そのリソースは追加の分析ができるように保持されます。</li> <li>● <b>rename-restart</b> - このアクションはドメインを完全に終了させてから、新しい名前でこれを再起動します。</li> </ul>

状態	説明
<code>&lt;on_lockfailure&gt;</code>	<p>ロックマネージャーがリソースロックを失う場合に取りべきアクションを指定します。以下のアクションは、それらすべてが個別のロックマネージャーによってサポートされる必要はないものの、<code>libvirt</code>によって認識されます。いずれのアクションも指定されない場合、各ロックマネージャーは、そのデフォルトアクションを取ります。以下の引数を使用することができます。</p> <ul style="list-style-type: none"> <li>• <b>poweroff</b> - 強制的にドメインの電源をオフにします。</li> <li>• <b>restart</b> - ロックを再取得するようにドメインを再起動します。</li> <li>• <b>pause</b> - ロックの問題が解決されるとドメインが手動で再開できるようにドメインを一時停止します。</li> <li>• <b>ignore</b> - 何も起こらない場合はドメインを実行したままにします。</li> </ul>

## 24.14. 電力管理

ドメイン XML の以下のセクションに影響を与える従来の管理ツールを使用して、ゲスト仮想マシン OS への BIOS の公開を強制的に有効にしたり、無効にしたりすることが可能です。

```
...
<pm>
  <suspend-to-disk enabled='no' />
  <suspend-to-mem enabled='yes' />
</pm>
...
```

図24.24 電力管理

`<pm>` 要素は、引数 **yes** を使用して有効にするか、または引数 **no** を使用して無効にすることができます。BIOS サポートは、引数の **suspend-to-disk** を使用する場合は **S3** に、引数 **suspend-to-mem** の ACP スリープ状態を使用する場合は **S4** に対して実行できます。何も指定されていない場合、ハイパーバイザーは、そのデフォルト値のままになります。

## 24.15. ハイパーバイザーの機能

ハイパーバイザーを使って、一部の CPU/マシン機能を有効にしたり (**state='on'**)、無効にしたり (**state='off'**) できます。

```
...
<features>
  <pae/>
  <acpi/>
  <apic/>
  <hap/>
  <privnet/>
  <hyperv>
    <relaxed state='on'/>
  </hyperv>
</features>
...
```

図24.25 ハイパーバイザーの機能

すべての機能は **<features>** 要素内に一覧表示されますが、**<state>** が指定されない場合はそれは無効にされます。使用可能な機能は、**capabilities XML** を呼び出して見つけることができますが、完全仮想化ドメインについての一般的なセットは以下のようになります。

表24.10 ハイパーバイザー機能の要素

状態	説明
<pae>	物理ドレスの拡張モードでは、32 ビットのゲスト仮想マシンが 4 GB を超えるメモリーに対応できます。
<acpi>	たとえば KVM ゲスト仮想マシンに対する電源管理に便利ですが、正常なシャットダウンを実行できる必要があります。
<apic>	プログラミング可能な IRQ 管理の使用を可能にします。この要素については、ゲスト仮想マシンの EOI (割り込み終点: End of Interrupt) の可用性を設定する値の <b>on</b> と <b>off</b> を含むオプションの属性 <b>eoi</b> があります。
<hap>	ハードウェア支援のページング (Hardware Assisted Paging) がハードウェアで利用可能な場合に、その使用を有効にします。

24.16. 時間管理

ゲスト仮想マシンのクロックは、通常ホスト物理マシンのクロックから初期化されます。大半のオペレーティングシステムは、ハードウェアのクロックをデフォルトの設定である UTC のままであることを予期します。

ゲスト仮想マシン上で時間を正確に管理することは、仮想化プラットフォームにおける主要な課題となります。複数のハイパーバイザーが様々な方法で時間管理の問題を処理しようとします。libvirt では、ドメインの XML 内で **<clock>** と **<timer>** の要素を使い、ハイパーバイザーとは独立した時間管

理の構成を提供します。ドメイン XML の編集は **virsh edit** コマンドを使って行います。さらに詳しくは、「[ゲスト仮想マシンの XML 設定の編集](#)」を参照してください。

```
...
<clock offset='localtime'>
  <timer name='rtc' tickpolicy='catchup' track='guest'>
    <catchup threshold='123' slew='120' limit='10000' />
  </timer>
  <timer name='pit' tickpolicy='delay' />
</clock>
...
```

図24.26 時間管理

ドメイン XML のこのセクションを構成するコンポーネントは以下の通りです。

表24.11 時間管理の要素

状態	説明
----	----

状態	説明
<clock>	<p>&lt;clock&gt; 要素は、ゲスト仮想マシンのクロックをホスト物理マシンのクロックに同期する方法を判別するために使用されます。<b>offset</b> 属性は 4 つの使用できる値を取り、ゲスト仮想マシンクロックをホスト物理マシンに同期する方法に対する詳細な制御を可能にします。ハイパーバイザーは、すべてのタイムソースに対するすべてのポリシーに対応する必要がないことに注意してください。</p> <ul style="list-style-type: none"> <li>• <b>utc</b> - クロックを起動時に UTC に同期します。<b>utc</b> モードは <b>variable</b> モードに変換でき、これは <b>adjustment</b> 属性を使用して制御できます。値が <b>reset</b> の場合、変換は実行されません。数値は、初期 <b>adjustment</b> としてその値を使用することで、<b>variable</b> モードへの変換を強制します。デフォルトの <b>adjustment</b> はハイパーバイザーに固有のものになります。</li> <li>• <b>localtime</b> - ゲスト仮想マシンクロックを、起動時にホスト物理マシンの設定タイムゾーンに同期します。<b>adjustment</b> 属性は <b>utc</b> モードと同様に機能します。</li> <li>• <b>timezone</b> - ゲスト仮想マシンクロックを、要求されるタイムゾーンに同期します。</li> <li>• <b>variable</b> - <b>basis</b> 属性に基づいて、ゲスト仮想マシンクロックに、UTC または <b>localtime</b> との対比で適用される任意のオフセットを指定します。UTC (または <b>localtime</b>) に対する差分は、<b>adjustment</b> 属性を使用して数秒単位で指定されます。ゲスト仮想マシンは、RTC (リアルタイムクロック) を自由に調整することができ、行われた調整は再起動後も維持されます。これは、RTC 調整が再起動のたびに失われる <b>utc</b> と <b>localtime</b> モード (オプション属性 <b>adjustment='reset'</b> を指定) とは対照的です。さらに、<b>basis</b> 属性には、<b>utc</b> (デフォルト) または <b>localtime</b> のいずれかを使用することができます。<b>clock</b> 要素には、ゼロまたは 1 以上の <b>&lt;timer&gt;</b> 要素を含めることができます。</li> </ul>
<timer>	注記を参照してください。
<present>	特定のタイマーがゲスト仮想マシンで利用できるかどうかを指定します。 <b>yes</b> または <b>no</b> に設定することができます。

## 注記

**<clock>** 要素には、子としてゼロ以上の**<timer>** 要素を持たせることができます。**<timer>** 要素は、ゲスト仮想マシンクロックの同期に使用されるタイムソースを指定します。

それぞれの**<timer>** 要素には、**name** のみが必要となり、その他すべての属性はオプションになります。

- **name** - 変更される **timer** を選択します。以下の値が受け入れ可能です。**kvmclock**、**pit**、または **rtc**。
- **track** - **timer track** を指定します。以下の値が受け入れ可能です。**boot**、**guest**、または **wall**。**track** は **name="rtc"** にのみ有効です。
- **tickpolicy** - ゲスト仮想マシンにティックを挿入するための期限に間に合わなかった場合にどうなるかを決定します。以下の値を割り当てることができます。
  - **delay** - 通常レートでのティックの配信を継続します。ゲスト仮想マシンの時間は、ティックの遅延により遅れます。
  - **catchup** - ティックの遅れを取り戻すために、高めのレートでティックを配信します。ゲスト仮想マシンの時間は、遅れが取り戻されると表示されなくなります。さらに、**threshold**、**slew**、および **limit** の 3 つのオプション属性があり、それぞれは正の整数になります。
  - **merge** - 遅れたティックを単一のティックにマージし、それらを挿入します。ゲスト仮想マシンの時間は、マージの実行方法により、遅れる可能性があります。
  - **discard** - 遅れたティックを破棄し、デフォルトの間隔設定で挿入を続行します。ゲスト仮想マシンの時間は、遅れたティックの処理についての明示的なステートメントがない場合に遅れる可能性があります。

## 注記

デフォルトでは、**utc** の値が仮想マシンのクロックオフセットとして設定されます。ただし、ゲスト仮想マシンのクロックが **localtime** の値を使って実行される場合、ゲスト仮想マシンのクロックをホスト物理マシンのクロックと同期させるために、クロックオフセットを別の値に変更しなければなりません。

### 例24.1 常に UTC に同期する

```
<clock offset="utc" />
```

### 例24.2 常にホスト物理マシンのタイムゾーンに同期する

```
<clock offset="localtime" />
```

### 例24.3 任意のタイムゾーンに同期する



```
<clock offset="timezone" timezone="Europe/Paris" />
```

#### 例24.4 UTC に同期させてから任意で秒数を増減させる

```
<clock offset="variable" adjustment="123456" />
```

## 24.17. TIMER 要素属性

**name** 要素には、使用されるタイムソースの名前が含まれます。これには、以下の値のいずれかを使用することができます。

表24.12 **name** 属性の値

値	説明
pit	Programmable Interval Timer - 定期的な割り込みが付いたタイマーです。この属性を使用すると、 <b>tickpolicy delay</b> がデフォルトの設定になります。
rtc	Real Time Clock - 継続的に実行するタイマーで、定期的な割り込みが付いています。この属性は <b>tickpolicy catchup</b> サブ要素をサポートします。
kvmclock	KVM クロック - KVM ゲスト仮想マシン用に推奨しているクロックソースです。KVM の <b>pvclock</b> または <b>kvm-clock</b> によりゲスト仮想マシンがホスト物理マシンのウォールクロックタイムを読み込みます。

**track** 属性は、タイマーで追跡する対象を指定します。**rtc** の **name** の値にのみ有効です。

表24.13 **track** 属性の値

値	説明
boot	旧オプションの <b>host physical machine</b> に相当します。これは未対応の追跡オプションです。
guest	RTC が常にゲスト仮想マシンの時間を追跡します。
wall	RTC が常にホストの時間を追跡します。

**tickpolicy** 属性とその値は、ゲスト仮想マシンにティックを渡すために使用されるポリシーを指定します。

表24.14 **tickpolicy** 属性の値

値	説明
delay	通常レートで配信を継続します (ティックが遅延する)。
catchup	遅れを取り戻すため高めレートで配信します。
merge	複数のティックを単一のティックにマージします。
discard	遅れたティックはすべて破棄します。

**present** 属性は、ゲスト仮想マシンに見えるデフォルトのタイマーセットを上書きします。**present** 属性は以下の値を取ります。

表24.15 **present** 属性の値

値	説明
はい	このタイマーをゲスト仮想マシンに表示するよう強制します。
いいえ	このタイマーをゲスト仮想マシンに非表示にするよう強制します。

## 24.18. DEVICES

この XML 要素のセットすべては、ゲスト仮想マシンのドメインに提供されるデバイスを説明するために使用されます。以下のデバイスのすべては、メイン **<devices>** 要素の子として示されます。

以下の仮想デバイスがサポートされています。

- **virtio-scsi-pci** - PCI バスストレージデバイス
- **virtio-9p-pci** - PCI バスストレージデバイス
- **virtio-blk-pci** - PCI バスストレージデバイス
- **virtio-net-pci** - PCI バスネットワークデバイス (**virtio-net** としても知られる)
- **virtio-serial-pci** - PCI バス入力デバイス
- **virtio-balloon-pci** - PCI バスメモリーバルーンデバイス
- **virtio-rng-pci** - PCI バス乱数ジェネレーターデバイス



## 重要

ベクターの番号が 33 以上の値に設定されている virtio デバイスが作成される場合、デバイスは Red Hat Enterprise Linux 7 ではなく、Red Hat Enterprise Linux 6 上のゼロの値に設定されているように動作します。結果として生じるベクター設定の不一致により、いずれかのプラットフォームの virtio デバイスのベクターの番号が 33 以上に設定されている場合は移行エラーが生じます。そのため、ベクター値を 33 以上に設定することは推奨されません。**virtio-balloon-pci** および **virtio-rng-pci** を例外とするすべての virtio デバイスは **vector** 引数を許可します。

```
...
<devices>
  <emulator>/usr/lib/kvm/bin/kvm-dm</emulator>
</devices>
...
```

図24.27 デバイス - 子要素

**<emulator>** 要素のコンテンツは、デバイスモデルエミュレーターのバイナリーへの完全修飾パスを指定します。この **capabilities XML** は、それぞれの特定ドメインタイプ、またはアーキテクチャーの組み合わせに対して使用するための推奨されるデフォルトエミュレーターを指定します。

### 24.18.1. ハードドライブ、フロッピーディスク、CD-ROM

ドメイン XML のこのセクションは、**<disk>** 要素でフロッピーディスク、ハードディスク、CD-ROM または準仮想化ドライバーが指定されている場合などの、ディスクに類するデバイスを指定します。

```
...
<devices>
  <disk type='file' snapshot='external'>
    <driver name="tap" type="aio" cache="default"/>
    <source file='/var/lib/xen/images/fv0' startupPolicy='optional'>
      <seclabel relabel='no'/>
    </source>
    <target dev='hda' bus='ide'/>
    <iotune>
      <total_bytes_sec>10000000</total_bytes_sec>
      <read_iops_sec>400000</read_iops_sec>
      <write_iops_sec>100000</write_iops_sec>
    </iotune>
    <boot order='2'/>
    <encryption type='...'>
      ...
    </encryption>
    <shareable/>
    <serial>
      ...
    </serial>
  </disk>
```

図24.28 デバイス - ハードドライブ、フロッピーディスク、CD-ROM

```
<disk type='network'>
  <driver name="qemu" type="raw" io="threads" ioeventfd="on"
event_idx="off"/>
  <source protocol="sheepdog" name="image_name">
    <host name="hostname" port="7000"/>
  </source>
  <target dev="hdb" bus="ide"/>
  <boot order='1' />
  <transient/>
  <address type='drive' controller='0' bus='1' unit='0' />
</disk>
```

図24.29 デバイス - ハードドライブ、フロッピーディスク、CD-ROM サンプル 2

```
<disk type='network'>
  <driver name="qemu" type="raw"/>
  <source protocol="rbd" name="image_name2">
    <host name="hostname" port="7000"/>
  </source>
  <target dev="hdd" bus="ide"/>
  <auth username='myuser'>
    <secret type='ceph' usage='mypassid' />
  </auth>
</disk>
```

図24.30 デバイス - ハードドライブ、フロッピーディスク、CD-ROM サンプル 3

```
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw' />
  <target dev='hdc' bus='ide' tray='open' />
  <readonly />
</disk>
<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw' />
  <source protocol="http" name="url_path">
    <host name="hostname" port="80" />
  </source>
  <target dev='hdc' bus='ide' tray='open' />
  <readonly />
</disk>
```

図24.31 デバイス - ハードドライブ、フロッピーディスク、CD-ROM サンプル 4

```

<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw' />
  <source protocol="https" name="url_path">
    <host name="hostname" port="443" />
  </source>
  <target dev='hdc' bus='ide' tray='open' />
  <readonly />
</disk>
<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw' />
  <source protocol="ftp" name="url_path">
    <host name="hostname" port="21" />
  </source>
  <target dev='hdc' bus='ide' tray='open' />
  <readonly />
</disk>

```

図24.32 デバイス - ハードドライブ、フロッピーディスク、CD-ROM サンプル 5

```

<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw' />
  <source protocol="ftps" name="url_path">
    <host name="hostname" port="990" />
  </source>
  <target dev='hdc' bus='ide' tray='open' />
  <readonly />
</disk>
<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw' />
  <source protocol="tftp" name="url_path">
    <host name="hostname" port="69" />
  </source>
  <target dev='hdc' bus='ide' tray='open' />
  <readonly />
</disk>
<disk type='block' device='lun'>
  <driver name='qemu' type='raw' />
  <source dev='/dev/sda' />
  <target dev='sda' bus='scsi' />
  <address type='drive' controller='0' bus='0' target='3' unit='0' />
</disk>

```

図24.33 デバイス - ハードドライブ、フロッピーディスク、CD-ROM サンプル 6

```

<disk type='block' device='disk'>
  <driver name='qemu' type='raw' />
  <source dev='/dev/sda' />
  <geometry cyls='16383' heads='16' secs='63' trans='lba' />
  <blockio logical_block_size='512' physical_block_size='4096' />
  <target dev='hda' bus='ide' />
</disk>
<disk type='volume' device='disk'>
  <driver name='qemu' type='raw' />
  <source pool='blk-pool0' volume='blk-pool0-vol0' />
  <target dev='hda' bus='ide' />
</disk>
<disk type='network' device='disk'>
  <driver name='qemu' type='raw' />
  <source protocol='iscsi' name='iqn.2013-07.com.example:iscsi-
nopol/2'>
    <host name='example.com' port='3260' />
  </source>
  <auth username='myuser'>
    <secret type='chap' usage='libvirtiscsi' />
  </auth>
  <target dev='vda' bus='virtio' />
</disk>

```

図24.34 デバイス - ハードドライブ、フロッピーディスク、CD-ROM サンプル 7

```

<disk type='network' device='lun'>
  <driver name='qemu' type='raw' />
  <source protocol='iscsi' name='iqn.2013-07.com.example:iscsi-
nopol/1'>
    iqn.2013-07.com.example:iscsi-pool
    <host name='example.com' port='3260' />
  </source>
  <auth username='myuser'>
    <secret type='chap' usage='libvirtiscsi' />
  </auth>
  <target dev='sda' bus='scsi' />
</disk>
<disk type='volume' device='disk'>
  <driver name='qemu' type='raw' />
  <source pool='iscsi-pool' volume='unit:0:0:1' mode='host' />
  <auth username='myuser'>
    <secret type='chap' usage='libvirtiscsi' />
  </auth>
  <target dev='vda' bus='virtio' />
</disk>

```

図24.35 デバイス - ハードドライブ、フロッピーディスク、CD-ROM サンプル 8

```

<disk type='volume' device='disk'>
  <driver name='qemu' type='raw' />
  <source pool='iscsi-pool' volume='unit:0:0:2' mode='direct' />
  <auth username='myuser'>
    <secret type='chap' usage='libvirtiscsi' />
  </auth>
  <target dev='vda' bus='virtio' />
</disk>
<disk type='file' device='disk'>
  <driver name='qemu' type='raw' cache='none' />
  <source file='/tmp/test.img' startupPolicy='optional' />
  <target dev='sdb' bus='scsi' />
  <readonly />
</disk>
<disk type='file' device='disk'>
  <driver name='qemu' type='raw' discard='unmap' />
  <source file='/var/lib/libvirt/images/discard1.img' />
  <target dev='vdb' bus='virtio' />
  <alias name='virtio-disk1' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x09'
function='0x0' />
</disk>
</devices>
...

```

図24.36 デバイス - ハードドライブ、フロッピーディスク、CD-ROM サンプル 9

#### 24.18.1.1. disk 要素

**<disk>** 要素は、ディスクを記述するためのメインコンテナです。属性の**type**は **<disk>** 要素と共に使用できます。以下のタイプが受け入れ可能です。

- **file**
- **block**
- **dir**
- **network**

詳細は、[libvirt アップストリームページ](#)を参照してください。

#### 24.18.1.2. ソース要素

ディスクソースを表します。ディスクソースは、以下のようにディスクタイプ属性に依存します。

- **<file>: file** 属性は、ディスクが置かれるファイルへの完全修飾パスを指定します。
- **<block>: dev** 属性は、ディスクとして機能するホストデバイスへの完全修飾パスを指定します。
- **<dir>: dir** 属性は、ディスクとして使用されるディレクトリーへの完全修飾パスを指定します。

- **<network>: protocol** 属性は、要求されるイメージにアクセスするために使用されるプロトコルを指定します。使用できる値は、**nbd**、**iscsi**、**rbd**、**sheepdog**、および **gluster** になります。
  - **protocol** 属性が **rbd**、**sheepdog**、または **gluster** の場合、追加属性 **name** が必須になります。この属性は使用されるボリュームおよびイメージを指定します。
  - **protocol** 属性が **nbd** の場合、**name** 属性はオプションになります。
  - **protocol** 属性が **iscsi** の場合、**name** 属性には、ターゲットの名前とスラッシュで区切られた論理ユニット番号が含まれる場合があります。たとえば、**iqn.2013-07.com.example:iscsi-pool/1** のようになります。指定されない場合、デフォルト LUN はゼロになります。
- **<volume>**: 基礎となるディスクソースは **pool** および **volume** 属性で表されます。
  - **<pool>**: ディスクソースがあるストレージプール (**libvirt** で管理される) の名前です。
  - **<volume>**: ディスクソースとして使用されるストレージボリューム (**libvirt** で管理される) の名前です。

**volume** 属性の値は、**virsh vol-list [pool-name]** の **Name** 列の出力になります。

ディスクタイプが **network** の場合、**source** には、接続するホスト物理マシンを指定するために使用されるゼロか、または1つ以上の **host** サブ要素が含まれます。これには、**type='dir'** および **type='network'** が含まれます。**CD-ROM** または **フロッピー** (デバイス属性) を表す **file** ディスクタイプの場合、ソースファイルにアクセスできない場合にディスクをどう処理するかについてのポリシーを定義することができます。これは、以下の値のいずれかと共に **startupPolicy** 属性を設定することによって実行されます。

- **mandatory** は、何らかの理由で欠落が生じる場合に失敗させます。これはデフォルトの設定です。
- **requisite** は、ブート時に欠落がある場合に失敗を生じさせ、移行/復元/復帰の時点で見つからない場合にドロップします。
- **optional** は、開始の試行時にドロップします。

#### 24.18.1.3. ミラー要素

この要素は、ハイパーバイザーが **BlockCopy** 操作を開始する場合に見られます。ここで、属性ファイルの **<mirror>** ロケーションには、最終的には、属性形式のファイル形式 (ソースの形式と異なる可能性がある) で、ソースと同じコンテンツが含まれることになります。**ready** 属性がある場合、ディスクではピボットが可能なことが認識されます。それ以外の場合、ディスクはおそらく依然としてコピーを続けます。現在のところ、この要素は出力でのみ有効であり、入力では無視されます。

#### 24.18.1.4. ターゲット要素

**<target>** 要素は、ディスクがゲスト仮想マシン OS に公開されるバス/デバイスを制御します。**dev** 属性は、論理デバイス名を示します。指定される実際のデバイス名は、ゲスト仮想マシン OS 内のデバイス名にマップされるとは保証されません。オプションのバス属性は、エミュレートするディスクデバイスのタイプを指定します。使用できる値は、以下の標準的な値を含む、ドライバー固有の値になります。**ide**、**scsi**、**virtio**、**kvm**、**usb** または **sata**。バスタイプは、省略される場合、デバイス名のスタイルから推定されます。たとえば、**'sda'** という名前のデバイスは、通常 **SCSI** バスを使用して



エクスポートされます。オプション属性の **tray** は、リムーバブルディスク (CDROM またはフロッピーディスクなど) のトレイの状態を示し、値は **open** または **closed** にすることができます。デフォルト設定は **closed** です。

#### 24.18.1.5. iotune 要素

オプションの **<iotune>** 要素は、デバイスごとに異なる可能性のある値と共に、追加のデバイスごとの I/O チューニングを提供する機能を提供します (ドメインにグローバルに適用される **blkiotune** 要素と比較)。この要素には、以下のオプションのサブ要素があります (全くサブ要素が指定されていないか、または **0** の値と共に指定されている場合は制限がないことを示すことに注意してください)。

- **<total\_bytes\_sec>** - 1 秒あたりのバイト単位の合計スループット制限です。この要素は、**<read\_bytes\_sec>** または **<write\_bytes\_sec>** と共に使用することはできません。
- **<read\_bytes\_sec>** - 1 秒あたりのバイト単位の読み込みスループット制限です。
- **<write\_bytes\_sec>** - 1 秒あたりのバイト単位の書き込みスループット制限です。
- **<total\_iops\_sec>** - 1 秒あたりの合計 I/O 操作回数です。この要素は、**<read\_iops\_sec>** または **<write\_iops\_sec>** と共に使用することはできません。
- **<read\_iops\_sec>** - 1 秒あたりの読み込み I/O 回数です。
- **<write\_iops\_sec>** - 1 秒あたりの書き込み I/O 操作回数です。

#### 24.18.1.6. ドライバー要素

オプションの **<driver>** 要素は、ディスクを提供するために使用されるハイパーバイザードライバーに関連する詳細を指定することを許可します。以下のオプションを使用することができます。

- ハイパーバイザーが複数のバックエンドドライバーをサポートする場合、**name** 属性は、プライマリバックエンドドライバーの名前を選択し、オプションの **type** 属性はサブタイプを提供します。
- オプションの **cache** 属性は、キャッシュメカニズムを制御します。使用できる値は以下の通りです。**default**、**none**、**writethrough**、**writeback**、**directsync** (**writethrough** に似ていますが、ホスト物理マシンのページキャッシュをバイパスします) および **unsafe** (ホスト物理マシンはすべてのディスク IO をキャッシュする可能性があり、ゲスト仮想マシンの同期要求は無視されます)。
- オプションの **error\_policy** 属性は、ディスクの読み込みまたは書き込みエラー時にハイパーバイザーがどのように動作するかを制御します。使用できる値は、**stop**、**report**、**ignore**、および **enospace** です。**error\_policy** のデフォルト設定は **report** です。さらに、読み取りエラーの動作のみを制御するオプションの **rerror\_policy** もあります。**rerror\_policy** が指定されていない場合、**error\_policy** が読み込みエラーと書き込みエラーの両方に使用されます。**rerror\_policy** が指定される場合、それは読み込みエラーの **error\_policy** を上書きします。また、**enospace** は読み込みエラーの有効なポリシーではないことに注意してください。そのため、**error\_policy** が **enospace** に設定され、**no rerror\_policy** が読み込みエラーに指定される場合、デフォルト設定の **report** が使用されます。
- オプションの **io** 属性は、I/O 上の特定のポリシーを制御します。**kvm** ゲスト仮想マシンは、**threads** および **native** をサポートします。オプションの **ioeventfd** 属性により、ユーザーは **virtio** ディスクデバイスのドメイン I/O の非同期処理を設定できます。デフォルトはハ

ハイパーバイザーによって設定されます。受け入れ可能な値は、**on** および **off** です。これを有効にすることにより、別のスレッドが I/O を処理する間に、ゲスト仮想マシンを実行させることができます。通常これは、I/O 時のシステム CUP の高い使用率を経験するゲスト仮想マシンの場合に役立ちます。なお、過負荷のホスト物理マシンは、ゲスト仮想マシンの I/O 待ち時間を増やす可能性があります。なお、デフォルトの設定を変更せず、ハイパーバイザーに設定を決定させるようにすることを推奨します。



### 注記

**ioeventfd** 属性は、**disk XML** セクションや **device XML** セクションの **<driver>** 要素に含まれます。前者の場合、virtIO ディスクに影響し、後者の場合は SCSI ディスクに影響を与えます。

- オプションの **event\_idx** 属性は、デバイスイベント処理のいくつかの側面を制御し、**on** または **off** のいずれかに設定できます。これが **on** の場合、割り込みの数が減少し、ゲスト仮想マシンに対して終了します。デフォルトはハイパーバイザーによって決定され、デフォルト設定は **on** になります。この動作が必要ない場合、**off** を設定することにより、この機能をオフに強制実行できます。ただし、デフォルトの設定を変更せずに、ハイパーバイザーに設定を決定させるようにすることを推奨します。
- オプションの **copy\_on\_read** 属性は、読み込みバッキングファイルをイメージファイルにコピーするかどうかを制御します。許可される値は、**on** または **<off>** のいずれかになります。**copy-on-read** は、同じバッキングファイルのセクターに繰り返しアクセスすることを防ぎ、バッキングファイルが速度の遅いネットワーク上にある場合に便利になります。デフォルトで、**copy-on-read** は **off** になります。
- **discard='unmap'** は破棄サポートを有効にするために設定できます。同じ行を **discard='ignore'** に入れ替えると無効にできます。**discard='ignore'** がデフォルト設定です。

#### 24.18.1.7. 追加のデバイス要素

以下の属性は、**device** 要素内で使用することができます。

- **<boot>** - ディスクが起動可能であると指定します。

#### 追加のブート値

- **<order>** - ブートシーケンス時にデバイスが試行される順序を決定します。
- **<per-device> boot** 要素は、BIOS ブートローダーセクションの一般的な **boot** 要素と共に使用することができません。
- **<encryption>** - ボリュームが暗号化される方法を指定します。
- **<readonly>** - デバイスがゲスト仮想マシンで変更できないことを示します。この設定は、**attribute <device='cdrom'>** の場合のディスクのデフォルトです。
- **<shareable>** デバイスがドメイン間で共有されることが予期されることを示します (ハイパーバイザーおよび OS がこれをサポートする場合)。**shareable** が使用される場合、**cache='no'** がそのデバイスに使用される必要があります。

- **<transient>**- ゲスト仮想マシンが終了する場合、デバイスコンテンツへの変更が自動的に元に戻されることを示します。一部のハイパーバイザーでは、ディスクを **transient** とマークすることにより、ドメインが移行またはスナップショットに加わることを防ぎます。
- **<serial>**- ゲスト仮想マシンのハードドライブのシリアル番号を指定します。たとえば、**<serial>WD-WMAP9A966149</serial>** のようになります。
- **<wwn>** - 仮想ハードディスクまたは CD-ROM ドライブの WWN (World Wide Name) を指定します。これは、16 進法の 16 桁番号で構成される必要があります。
- **<vendor>** - 仮想ハードディスクまたは CD-ROM デバイスのベンダーを指定します。この長さは、8 文字の印刷可能な文字を超えることはできません。
- **<product>** - 仮想ハードディスクまたは CD-ROM デバイスの製品を指定します。その長さは、16 文字の印刷可能な文字を超えることはできません。
- **<host>** - 以下の 4 つの属性をサポートします。**viz**、**name**、**port**、**transport** および **socket**。これらは、ホスト名、ポート番号、トランスポートタイプおよびソケットのパスをそれぞれ指定します。この要素の意味と要素の数は、以下に示される **protocol** 属性によって異なります。

#### 追加のホスト属性

- **nbd -nbd-server** を実行するサーバーを指定し、単一のホスト物理マシンにのみ使用することができます。
- **rbd** - RBD タイプのサーバーを監視し、1 つ以上のホスト物理マシンに使用することができます。
- **sheepdog: sheepdog** サーバーの 1 つを指定し (デフォルトは **localhost:7000**)、1 つのホスト物理マシンに使用できるか、またはいずれのホスト物理マシンにも使用できません。
- **gluster - glusterd** デーモンを実行するサーバーを指定し、単一のホスト物理マシンにのみ使用できます。**transport** 属性の有効な値は **tcp**、**rdma** または **unix** です。いずれの値も指定されていない場合は、**tcp** が想定されます。**transport** が **unix** の場合、**socket** 属性は、**unix** ソケットへのパスを指定します。
- **<address>** - ディスクをコントローラーの指定されたスロットに関連付けます。実際の **<controller>** デバイスを推定できることもありますが、これを明示的に指定することもできます。**type** 属性は必須であり、通常は **pci** または **drive** になります。**pci** コントローラーの場合、**bus**、**slot**、および **function** の属性が、オプションの **domain** および **multifunction** と共に存在する必要があります。**multifunction** はデフォルトで **off** になります。**drive** コントローラーの場合、追加属性の **controller**、**bus**、**target**、および **unit** を使用でき、それぞれのデフォルト設定は **0** になります。
- **auth:** ソースにアクセスするのに必要な認証資格情報を提供します。これには、認証時に使用するユーザー名を特定する必須属性 **username** と、必須属性 **type** を持つサブ要素 **secret** が含まれます。
- **geometry** - 配置設定をオーバーライドする機能を提供します。これは、ほとんどの場合 **S390 DASD** ディスクまたは古い **DOS** ディスクに対して役立ちます。これは以下のパラメーターを取ることができます。
  - **cyls** - シリンダーの数を指定します。

- **heads** - ヘッドの数を指定します。
- **secs** - トラックあたりのセクター数を指定します。
- **trans**: BIOS-Translation-Modus を指定し、**none**、**lba** または **auto** のいずれかを取るることができます。
- **blockio** - ブロックデバイスを、以下に記載されるブロックデバイスプロパティのいずれかで上書きできるようにします。

#### blockio オプション

- **logical\_block\_size** - ゲスト仮想マシン OS にレポートし、ディスク I/O の最小単位について記述します。
- **physical\_block\_size** - ゲスト仮想マシン OS にレポートし、ディスクデータの位置合わせに関連するディスクのハードウェアセクターサイズを記述します。

### 24.18.2. ファイルシステム

ゲスト仮想マシンから直接アクセスできるホスト物理マシンのファイルシステムディレクトリーです。

```
[...]
<devices>
  <filesystem type='template'>
    <source name='my-vm-template' />
    <target dir='/' />
  </filesystem>
  <filesystem type='mount' accessmode='passthrough'>
    <driver type='path' wrpolicy='immediate' />
    <source dir='/export/to/guest' />
    <target dir='/import/from/host' />
    <readonly />
  </filesystem>
  [...]
</devices>
[...]
```

図24.37 デバイス - ファイルシステム

**filesystem** 属性には、以下の値を使用できます。

- **type='mount'** - ゲスト仮想マシンにマウントするホスト物理マシンのディレクトリーを指定します。いずれの値も指定されていない場合は、これがデフォルトタイプになります。このモードには、属性の **type='path'** または **type='handle'** を持つオプションのサブ要素 **driver** があります。ドライバブロックには、ホスト物理マシンのページキャッシュをさらに制御するオプション属性の **wrpolicy** があり、この属性を省略すると、デフォルト設定に戻ります。一方、値 **immediate** を指定すると、ゲスト仮想マシンのファイル書き込み操作時に接触されるすべてのページに対して、ホスト物理マシンの書き戻しが即時にトリガーされます。
- **type='template'** - OpenVZ ファイルシステムテンプレートを指定し、OpenVZ ドライバーによってのみ使用されます。

- **type='file'** - ホスト物理マシンファイルがイメージとして処理され、ゲスト仮想マシンにマウントされることを指定します。このファイルシステム形式は自動検出され、LXC ドライバーによってのみ使用されます。
- **type='block'** - ゲスト仮想マシンでマウントされるホスト物理マシンのブロックデバイスを指定します。ファイルシステム形式は自動検出され、LXC ドライバーによってのみ使用されます。
- **type='ram'** - ホスト物理マシン OS からのメモリーを使用する、インメモリーファイルシステムが使用されることを指定します。**source** 要素には、キビバイト単位でメモリー使用制限を設ける単一属性の **usage** があり、LXC ドライバーによってのみ使用されます。
- **type='bind'** - ゲスト仮想マシン内の別のディレクトリーにバインドされるゲスト仮想マシン内のディレクトリーを指定します。この要素は、LXC ドライバーによってのみ使用されます。
- **accessmode** は、ソースのアクセス用にセキュリティーモードを指定します。現在、これは、KVM ドライバーに対して **type='mount'** と指定する場合にのみ機能します。使用できる値は以下の通りです。
  - **passthrough** - ゲスト仮想マシン内から設定されるユーザーの権限でソースがアクセスされることを指定します。これは、いずれも指定されていない場合にデフォルトの **accessmode** になります。
  - **mapped** - ソースがハイパーバイザーの権限でアクセスされることを指定します。
  - **squash** - '**passthrough**' に似ていますが、例外は、**chown** のような権限による操作の失敗が無視されることです。これにより、ハイパーバイザーを **root** 以外で実行するユーザーにとって、**passthrough** のようなモードを使いやすいものとしします。
- **source** - ゲスト仮想マシンでアクセスされるホスト物理マシン上のリソースを指定します。**name** 属性は、**<type='template'>** と共に使用され、**dir** 属性は **<type='mount'>** と共に使用される必要があります。**usage** 属性は、メモリーを KB 単位で設定するために **<type='ram'>** と共に使用する必要があります。
- **target** - ゲスト仮想マシン内のどこでソースドライバーがアクセスできるかを決定します。大半のドライバーの場合、これは自動的なマウントポイントになりますが、KVM の場合、これは、マウントする場所のヒントとしてゲスト仮想マシンにエクスポートされる任意の文字列タグでしかありません。
- **readonly** - ゲスト仮想マシンの読み取り専用マウントとしてファイルシステムのエクスポートを有効にします。デフォルトでは、**read-write** アクセスが指定されます。
- **space\_hard\_limit** - このゲスト仮想マシンのファイルシステムに利用可能な最大領域を指定します。
- **space\_soft\_limit** - このゲスト仮想マシンのファイルシステムで利用できる最大領域を指定します。コンテナは、猶予期間についてのソフト制限を超えることが許可されます。その後ハード制限が施行されます。

### 24.18.3. デバイスアドレス

多くのデバイスには、ゲスト仮想マシンに提示されるデバイスが仮想バス上のどこに配置されるかを説明するオプションの **<address>** サブ要素があります。アドレス (またはアドレス内のオプション属性) が入力で省略される場合、**libvirt** は適切なアドレスを生成しますが、レイアウトにより多くの制御が必

要な場合は明示的なアドレスが必要になります。**address** 要素を含むデバイス例について、以下を参照してください。

すべてのアドレスには、デバイスが置かれるバスを記述する必須の属性 **type** があります。指定されるデバイスに使用するアドレスを選択することは、デバイスおよびゲスト仮想マシンのアーキテクチャーによって部分的に制限されます。たとえば、ディスクデバイスは **type='disk'** を使用し、コンソールデバイスは、**i686** または **x86\_64** ゲスト仮想マシンで **type='pci'** を使用するか、または **PowerPC64 pseries** ゲスト仮想マシンで **type='spapr-vio'** を使用します。各アドレスの **<type>** には、デバイスが置かれるバス上の場所を制御するオプション属性があります。追加属性は以下のようになります。

- **type='pci'** - PCI アドレスには以下のような追加属性があります。
  - **domain** (2 バイトの 16 進整数。KVM によって現在使用されていません)
  - **bus** (0 から 0xffff までの 16 進値)
  - **slot** (0x0 から 0x1ff までの 16 進値)
  - **function** (0 から 7 までの値)
  - さらに、**multifunction** 属性も利用できます。これは、PCI コントロールレジスターの特定のスロット/機能のマルチファンクションビットをオンにするよう制御します。この **multifunction** 属性は、デフォルトで **'off'** になりますが、マルチファンクションが使用されるスロットのファンクション 0 では **'on'** に設定する必要があります。
- **type='drive'>** - **drive** アドレスには、以下の追加属性があります。
  - **controller** - (2 桁のコントローラー番号)
  - **bus** - (2 桁のバス番号)
  - **target** - (2 桁のバス番号)
  - **unit** - (バス上の 2 桁のユニット番号)
- **type='virtio-serial'** - 各 **virtio-serial** アドレスには、以下の追加属性があります。
  - **controller** - (2 桁のコントローラー番号)
  - **bus** - (2 桁のバス番号)
  - **slot** - (バス内の 2 桁のスロット)
- **type='ccid'** - スマートカードに使用される CCID アドレスには、以下の追加属性があります。
  - **bus** - (2 桁のバス番号)
  - **slot** - (バス内の 2 桁のスロット)
- **type='usb'** - USB アドレスには、以下の追加属性があります。
  - **bus** - (0 から 0xffff までの 16 進値)
  - **port** - (1.2 または 2.1.3.1 など最大 4 つのオクテットからなるドット区切りの表記)

- **type='spapr-vio'** - PowerPC pseries のゲスト仮想マシンで、デバイスは SPAPR-VIO バスに割り当てられます。これには、フラットな 64 ビットのアドレス空間があります。通常、デバイスは通常 0x1000 のゼロ以外の倍数で割り当てられますが、その他のアドレスも有効であり、libvirt によって許可されています。追加属性: **reg** (開始レジスターの 16 進値のアドレス) をこの属性に割り当てることができます。

#### 24.18.4. コントローラー

ゲスト仮想マシンのアーキテクチャーにより、多くの仮想デバイスを単一バスに割り当てることができます。通常、libvirt はバスに使用するコントローラーを自動的に推定できます。ただし、ゲスト仮想マシン XML に明示的な **<controller>** 要素を指定する必要がある場合があります。

```
...
<devices>
  <controller type='ide' index='0' />
  <controller type='virtio-serial' index='0' ports='16' vectors='4' />
  <controller type='virtio-serial' index='1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0a'
function='0x0' />
  <controller type='scsi' index='0' model='virtio-scsi'
num_queues='8' />
  </controller>
  ...
</devices>
...
```

図24.38 コントローラー要素

各コントローラーには、**"ide"**、**"fdc"**、**"scsi"**、**"sata"**、**"usb"**、**"ccid"**、または **"virtio-serial"** のいずれかにする必要のある必須属性 **type**、およびバスコントローラーが (**address** 要素のコントローラー属性で使用されるために) 表示される順序を記述する 10 進整数である、必須属性の **index** があります。**"virtio-serial"** コントローラーには、コントローラーで接続できるデバイスの数を制御する、2 つの追加のオプション属性である **ports** と **vectors** があります。

**<controller type='scsi'>** には、以下のいずれかになるオプション属性 **model** があります。**"auto"**、**"buslogic"**、**"ibmvscsi"**、**"lsilogic"**、**"lsias1068"**、**"virtio-scsi"** または **"vmpvscsi"**。**<controller type='scsi'>** には、指定されるキューの数に対してマルチキューサポートを有効にする属性 **num\_queues** があります。さらに、**ioeventfd** 属性を使用することもできます。これはコントローラーが SCSI ディスクで非同期処理を使用するかどうかを指定します。受け入れ可能な値は **"on"** および **"off"** です。

**"usb"** コントローラーには、以下のいずれかになるオプション属性の **model** があります。**"piix3-uhci"**、**"piix4-uhci"**、**"ehci"**、**"ich9-ehci1"**、**"ich9-uhci1"**、**"ich9-uhci2"**、**"ich9-uhci3"**、**"vt82c686b-uhci"**、**"pci-ohci"** または **"nec-xhci"**。さらに、USB バスがゲスト仮想マシンに対して明示的に無効にされる必要がある場合、**model='none'** を使用できます。PowerPC64 **"spapr-vio"** アドレスには、関連付けられたコントローラーがありません。

PCI または USB バス上のデバイスとなっているコントローラーの場合、オプションのサブ要素 **address** は、上記で指定される形式を使って、コントローラーのマスターバスとの正確な関係を指定できます。

USB コンパニオンコントローラーには、コンパニオンとマスターコントローラーの正確な関係を指定するためのオプションのサブ要素 **master** があります。コンパニオンコントローラーは、そのマスターと同じバス上にあるため、コンパニオンのインデックス値も等しくなければなりません。

```

...
<devices>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0' bus='0' slot='4' function='7' />
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <master startport='0' />
    <address type='pci' domain='0' bus='0' slot='4' function='0'
multifunction='on' />
  </controller>
  ...
</devices>
...

```

図24.39 デバイス - コントローラー - USB

### 24.18.5. デバイスのリース

ロックマネージャーを使用する場合、ゲスト仮想マシンに対してデバイスリースを記録するオプションがあります。ロックマネージャーは、ゲスト仮想マシンはリースが取得されない限り開始されないようにします。通常、管理ツールを使用して設定される場合、ドメイン XML の以下のセクションが影響を受けます。

```

...
<devices>
  ...
  <lease>
    <lockspace>somearea</lockspace>
    <key>somekey</key>
    <target path='/some/lease/path' offset='1024' />
  </lease>
  ...
</devices>
...

```

図24.40 デバイス - デバイスのリース

**lease** セクションには、以下の引数を含めることができます。

- **lockspace** - キーが保持される **lockspace** を特定する任意の文字列です。ロックマネージャーは、**lockspace** 名の形式または長さに対する追加の制限を設定できます。
- **key** - 取得されるリースを一意的に識別する任意の文字列です。ロックマネージャーは、キーの形式または長さに対して追加の制限を設定することができます。
- **target** - **lockspace** に関連付けられたファイルの完全修飾パスです。オフセットは、ファイル内のどこにリースが格納されるかを指定します。ロックマネージャーがオフセットを必要としない場合、この値を **0** に設定します。

### 24.18.6. ホスト物理マシンのデバイス割り当て



### 24.18.6.1. USB / PCI デバイス

ホスト物理マシンの USB および PCI デバイスは、管理ツールを使用してホスト物理マシンを変更することによって、**hostdev** 要素を使用してゲスト仮想マシンに渡すことができます。ドメイン XML ファイルの以下のセクションが設定されます。

```
...
<devices>
  <hostdev mode='subsystem' type='usb'>
    <source startupPolicy='optional'>
      <vendor id='0x1234' />
      <product id='0xbeef' />
    </source>
    <boot order='2' />
  </hostdev>
</devices>
...
```

図24.41 デバイス - ホスト物理マシンのデバイス割り当て

または、以下を実行することもできます。

```
...
<devices>
  <hostdev mode='subsystem' type='pci' managed='yes'>
    <source>
      <address bus='0x06' slot='0x02' function='0x0' />
    </source>
    <boot order='1' />
    <rom bar='on' file='/etc/fake/boot.bin' />
  </hostdev>
</devices>
...
```

図24.42 デバイス - ホスト物理マシン割り当ての代替

または、以下を実行することもできます。

```
...
<devices>
  <hostdev mode='subsystem' type='scsi'>
    <source>
      <adapter name='scsi_host0' />
      <address type='scsi' bus='0' target='0' unit='0' />
    </source>
    <readonly/>
    <address type='drive' controller='0' bus='0' target='0' unit='0' />
  </hostdev>
</devices>
..
```

図24.43 デバイス - ホスト物理マシン scsi デバイス割り当て

ドメイン XML のこのセクションを構成するコンポーネントは以下の通りです。

表24.16 ホスト物理マシンデバイス割り当て要素

パラメーター	説明
hostdev	<p>これは、ホスト物理マシンデバイスについて説明するための主な要素です。これは以下のオプションを受け入れます。</p> <ul style="list-style-type: none"><li>● <b>mode</b>: 値は常に <b>USB</b> および <b>PCI</b> デバイスの <b>subsystem</b> です。</li><li>● <b>type</b>: <b>USB</b> デバイスの場合は <b>usb</b>、<b>PCI</b> デバイスの場合は <b>pci</b> です。</li><li>● <b>managed</b>: デバイスの <i>Managed mode</i> (管理モード) を切り替えます。<ul style="list-style-type: none"><li>○ <b>PCI</b> デバイスについて <b>yes</b> に設定されると、ゲストマシンへの接続、ゲストマシンの接続解除、さらにホストマシンへの再接続が随時行われます。デバイス割り当ての一般的な使用の場合には <b>managed='yes'</b> が推奨されます。</li><li>○ <b>PCI</b> および <b>USB</b> デバイスについて <b>no</b> に設定されているか、または省略されている場合、デバイスはゲストに接続されたままになります。デバイスをホストが使用できる状態にするには、ユーザーは、ゲストの起動またはホットプラグを実行する前に、引数 <b>virNodeDeviceDetach</b> または <b>virsh nodedev-dettach</b> コマンドを使用する必要があります。さらに、ユーザーはデバイスのホットアンプラグまたはゲストの停止後に <b>virNodeDeviceReAttach</b> または <b>virsh nodedev-reattach</b> を使用する必要があります。特定のゲスト専用のデバイスには、<b>managed='no'</b> を設定することが主に推奨されています。</li></ul></li></ul>

パラメーター	説明
<b>source</b>	<p>ホスト物理マシンから表示されるデバイスについて説明します。USB デバイスは、<b>vendor</b> および <b>product</b> 要素を使用してベンダー/製品 ID によって処理されるか、または <b>address</b> 要素を使用してホスト物理マシン上のデバイスのアドレスによって処理されます。他方、PCI デバイスは、それらのアドレスによってのみ記述されます。USB デバイスのソース要素には、指定のホスト物理マシンの USB デバイスが見つからなかった場合の処理についてのルールを定義するために使用できる <b>startupPolicy</b> 属性が含まれる可能性があります。この属性は、以下の値を受け入れます。</p> <ul style="list-style-type: none"> <li>• <b>mandatory</b> - 何かの理由で見つからない場合に失敗します (デフォルト)</li> <li>• <b>requisite</b> - 起動時に見つからない場合に失敗し、移行/復元/復帰の時点で見つからない場合にドロップします。</li> <li>• <b>optional</b> - 開始の試行時に見つからない場合にドロップします。</li> </ul>
<b>vendor, product</b>	<p>これらの要素のそれぞれには、USB ベンダーおよび製品 ID を指定する <b>id</b> 属性があります。ID は、10 進数、16 進数 (0x で開始) または 8 進数 (0 で開始) の形式で指定できます。</p>
<b>boot</b>	<p>デバイスが起動可能であることを指定します。この属性の順序は、ブートシーケンス時にデバイスが試行される順序を決定します。デバイスごとの <b>boot</b> 要素は、BIOS ブートローダーセクションの一般的な <b>boot</b> 要素と共に使用することができません。</p>
<b>rom</b>	<p>PCI デバイスの ROM がゲスト仮想マシンに提示される方法を変更するために使用されます。オプションの <b>bar</b> 属性は、<b>on</b> または <b>off</b> に設定でき、デバイスの ROM がゲスト仮想マシンのメモリーマップに表示されるかどうかを決定します (PCI 資料によると、<b>rombar</b> 設定は、ROM の Base Address Register の表示を制御します)。rom bar が指定されない場合、デフォルト設定が使用されます。オプションの <b>file</b> 属性は、デバイスの ROM BIOS としてゲスト仮想マシンに提示されるバイナリーファイルをポイントするために使用されます。これは、たとえば SR-IOV 対応のイーサネットデバイス (VF のブート ROM を持たない) の仮想機能用に PXE ブート ROM を提供するのに便利です。</p>

パラメーター	説明
<b>address</b>	さらに、デバイスが表示されるホスト物理マシン上のUSBバスとデバイス番号を指定するための <b>bus</b> および <b>device</b> 属性があります。これらの属性の値は、10進数、16進数 (0x で開始) または 8進数 (0 で開始) の形式で指定できます。PCI デバイスの場合、要素には 3 つの属性が含まれ、デバイスを <b>lspci</b> または <b>virsh nodeudev-list</b> で検索できるように指定することができます。

#### 24.18.6.2. ブロック / キャラクターデバイス

ホスト物理マシンのブロック/キャラクターデバイスは、管理ツールを使用してドメイン xml の **hostdev** 要素を変更することで、ゲスト仮想マシンに渡すことができます。これは、コンテナベースの仮想化の場合にのみ可能であることに注意してください。

```
...
<hostdev mode='capabilities' type='storage'>
  <source>
    <block>/dev/sdf1</block>
  </source>
</hostdev>
...
```

#### 図24.44 デバイス - ホスト物理マシンデバイス割り当てブロックキャラクターデバイス

以下は代替アプローチです。

```
...
<hostdev mode='capabilities' type='misc'>
  <source>
    <char>/dev/input/event3</char>
  </source>
</hostdev>
...
```

#### 図24.45 デバイス - ホスト物理マシンデバイス割り当てブロックキャラクターデバイスの代替法 1

以下はもう 1 つの代替アプローチです。

```
...
<hostdev mode='capabilities' type='net'>
  <source>
    <interface>eth0</interface>
  </source>
</hostdev>
...
```

図24.46 デバイス - ホスト物理マシンデバイス割り当てブロックキャラクターデバイスの代替法 2

ドメイン XML のこのセクションを構成するコンポーネントは以下の通りです。

表24.17 ブロック / キャラクターデバイス要素

パラメーター	説明
<b>hostdev</b>	これは、ホスト物理マシンデバイスを記述するためのメインコンテナーです。ブロック/キャラクターデバイスのパススルーの場合 <b>mode</b> は常に <b>capabilities</b> で、ブロックデバイスの場合 <b>type</b> は <b>block</b> で、キャラクターデバイスの場合 <b>char</b> になります。
<b>source</b>	これは、ホスト物理マシンから表示されるデバイスについて記述します。ブロックデバイスの場合、ホスト物理マシン OS のブロックデバイスへのパスは、ネスト化された <b>block</b> 要素に指定され、キャラクターデバイスの場合は、 <b>char</b> 要素が使用されます。

24.18.7. リダイレクトされるデバイス

キャラクターデバイスによる USB デバイスのリダイレクトは、ドメイン XML の以下のセクションを変更して設定されます。

```
...
<devices>
  <redirdev bus='usb' type='tcp'>
    <source mode='connect' host='localhost' service='4000' />
    <boot order='1' />
  </redirdev>
  <redirfilter>
    <usbdev class='0x08' vendor='0x1234' product='0xbeef'
version='2.00' allow='yes' />
    <usbdev allow='no' />
  </redirfilter>
</devices>
...
```

図24.47 デバイス - リダイレクトされるデバイス

ドメイン XML のこのセクションを構成するコンポーネントは以下の通りです。

表24.18 リダイレクトされるデバイス要素

パラメーター	説明
<b>redirdev</b>	これは、リダイレクトされるデバイスを記述するためのメインコンテナーです。 <b>bus</b> は、USB デバイスの場合は <b>usb</b> にする必要があります。トンネルのホスト物理マシン側を記述するには、サポートされるシリアルデバイスの1つの一致する追加属性タイプが必要になります。それらの典型例は、 <b>type='tcp'</b> または <b>type='spicevmc'</b> (SPICE グラフィックスデバイスの <b>usbredir</b> チャンネルを使用) です。 <b>redirdev</b> 要素にはオプションのサブ要素 <b>address</b> があり、これは、デバイスを特定のコントローラーに関連付けます。さらに、 <b>source</b> などのサブ要素が、指定される <b>type</b> に基づいて必要になる場合があります。ただし、 <b>target</b> サブ要素は不要です (キャラクターデバイスのコンシューマーが、ゲスト仮想マシンに表示されるデバイスではなく、ハイパーバイザー自体であるため)。
<b>boot</b>	デバイスが起動可能であることを指定します。 <b>order</b> 属性は、ブートシーケンス時にデバイスが試行される順序を決定します。デバイスごとの <b>boot</b> 要素は、BIOS ブートローダーセクションの一般的な <b>boot</b> 要素と共に使用することができません。
<b>redirfilter</b>	これは、特定のデバイスをリダイレクトからフィルター処理するためのフィルタールールを作成するために使用されます。これは、各フィルタールールを定義するためにサブ要素 <b>usbdev</b> を使用します。 <b>class</b> 属性は USB クラスコードです。

### 24.18.8. スマートカードデバイス

仮想スマートカードは、**smartcard** 要素からゲスト仮想マシンに提供されます。ホスト物理マシン上の USB スマートカードリーダーデバイスは、ホスト物理マシンとゲスト仮想マシンの両方で使用できず、ゲスト仮想マシンから削除されるとホスト物理マシンのコンピューターをロックすることから、単純なデバイスパススルーではゲスト仮想マシンで使用することができません。したがって、一部のハイパーバイザーは、ゲスト仮想マシンにスマートカードインターフェースを提供できる特殊な仮想デバイスを提供します。その際、資格情報がホスト物理マシン、またはサードパーティーのスマートカードプロバイダーに対して作成されたチャンネルからどのように取得されるかを記述するために複数のモードが使用されます。

キャラクターデバイスによる USB デバイスのリダイレクトは、ドメイン XML の以下のセクションを変更する管理ツールで設定します。

```
...
<devices>
  <smartcard mode='host' />
  <smartcard mode='host-certificates'>
    <certificate>cert1</certificate>
    <certificate>cert2</certificate>
    <certificate>cert3</certificate>
    <database>/etc/pki/nssdb</database>
  </smartcard>
  <smartcard mode='passthrough' type='tcp'>
    <source mode='bind' host='127.0.0.1' service='2001' />
    <protocol type='raw' />
    <address type='ccid' controller='0' slot='0' />
  </smartcard>
  <smartcard mode='passthrough' type='spicevmc' />
</devices>
...
```

図24.48 デバイス - スマートカードデバイス

**smartcard** 要素には、必須属性の **mode** があります。それぞれのモードでは、ゲスト仮想マシンには、物理 USB CCID (Chip/Smart Card Interface Device) カードのように動作する USB バス上にデバイスが表示されます。

モード属性は以下ようになります。

表24.19 スマートカードの **mode** 要素

パラメーター	説明
<b>mode='host'</b>	このモードでは、ハイパーバイザーは、ゲスト仮想マシンから NSS 経由でホスト物理マシンのスマートカードに直接アクセスするためのすべての要求を中継します。これ以外の属性またはサブ要素は不要です。オプションの <b>address</b> サブ要素の使用については、以下を参照してください。
<b>mode='host-certificates'</b>	このモードにより、スマートカードをホスト物理マシンにプラグインする代わりに、ホスト物理マシン上のデータベースにある 3 つの NSS 証明書名を指定することができます。これらの証明書は、コマンドの <b>certutil -d /etc/pki/nssdb -x -t CT,CT,CT -S -s CN=cert1 -n cert1</b> , により生成され、結果として作成される 3 つの証明書名は、3 つの <b>certificate</b> サブ認証のそれぞれのコンテンツとして指定される必要があります。追加のサブ要素の <b>database</b> は代替ディレクトリーへの絶対パスを指定できます (証明書の作成時の <b>certutil</b> コマンドの <b>-d</b> フラグに一致)。これが表示されない場合、デフォルトは <b>/etc/pki/nssdb</b> になります。

パラメーター	説明
<code>mode='passthrough'</code>	<p>このモードを使用することにより、すべての要求を2次的なキャラクターデバイスを経由してサードパーティープロバイダーにトンネル化できます(これにより、ハイパーバイザーがホスト物理マシンと直接通信するのではなく、スマートカードと通信するか、または3つの認証ファイルを使用します。この操作モードでは、対応するシリアルデバイスタイプのいずれかに一致する追加属性の <b>type</b> が、トンネルのホスト物理マシン側を記述するために必要になります。 <b>type='tcp'</b> または <b>type='spicevmc'</b> (SPICE グラフィックスデバイスのスマートカードチャンネルを使用) などが一般的な例になります。さらに、<b>source</b> などのサブ要素は、指定されるタイプに応じて必要になります。ただし、<b>target</b> サブ要素は不要です(キャラクターデバイスのコンシューマーがゲスト仮想マシン内に表示されるデバイスではなく、ハイパーバイザー自体であるため)。</p>

各モードはオプションのサブ要素 **address** をサポートします。これは、スマートカードと ccid バスコントローラー間の相関関係を微調整します。詳細は、「[デバイスアドレス](#)」を参照してください。

#### 24.18.9. ネットワークインターフェース

ネットワークインターフェースデバイスは、ドメイン XML の以下の部分を設定する管理ツールを使用して変更します。

```
...
<devices>
  <interface type='direct' trustGuestRxFilters='yes'>
    <source dev='eth0' />
    <mac address='52:54:00:5d:c7:9e' />
    <boot order='1' />
    <rom bar='off' />
  </interface>
</devices>
...
```

図24.49 デバイス - ネットワークインターフェース

ゲスト仮想マシンのネットワークインターフェースの設定にはいくつかの方法が考えられます。これはインターフェース要素のタイプ属性に値を設定して実行されます。以下の値を使用できます。

- **"direct"**: ゲスト仮想マシンの NIC をホスト物理マシンの物理 NIC に割り当てます。詳細および具体例については、「[物理インターフェースへの直接割り当て](#)」を参照してください。
- **"network"**: これは動的またはワイヤレスのネットワーク設定を持つホスト物理マシン上の一般的なゲスト仮想マシンの接続に推奨される設定です。詳細および具体例については、「[仮想ネットワーク](#)」を参照してください。



- **"bridge"** - これは静的な有線ネットワーク設定を持つホスト物理マシン上の一般的なゲスト仮想マシンの接続に推奨される設定です。詳細は、「[LAN のブリッジ](#)」を参照してください。
- **"ethernet"**: ゲスト仮想マシンのネットワークを LAN に接続するために任意のスクリプトを実行する手段を管理者に提供します。詳細および具体例については、「[汎用イーサネット接続](#)」を参照してください。
- **"hostdev"**: PCI ネットワークデバイスが汎用デバイスパススルーを使用してゲスト仮想マシンに直接割り当てられるようにします。詳細および具体例については、「[PCI パススルー](#)」を参照してください。
- **"mcast"**: 仮想ネットワークを表すためにマルチキャストグループを使用できます。詳細および具体例については、「[マルチキャストトンネル](#)」を参照してください。
- **"user"**: **user** オプションを使用すると、外部への NAT が指定された仮想 LAN を提供するユーザー領域 **SLIRP** スタックパラメーターが設定されます。詳細および具体例については、「[ユーザー領域 SLIRP スタック](#)」を参照してください。
- **"server"**: サーバーオプションを使用して、仮想ネットワークを提供するための TCP クライアントサーバーアーキテクチャーを作成します。この仮想ネットワークでは、1 台のゲスト仮想マシンがネットワークのサーバーエンドを提供し、その他すべてのゲスト仮想マシンはクライアントとして設定されます。詳細および具体例については、「[TCP トンネル](#)」を参照してください。

これらのオプションには、それぞれ詳細についてのリンクがあります。さらに各 **<interface>** 要素は、ホスト物理マシンがゲスト仮想マシンから受信されるレポートを検出し、信頼することを可能にするオプションの **<trustGuestRxFilters>** 属性で定義できます。これらのレポートは、インターフェースがフィルターへの変更を受信するたびに送信されます。これには、プライマリー MAC アドレス、デバイスアドレスフィルター、または **vlan** 設定への変更が含まれます。**<trustGuestRxFilters>** 属性は、セキュリティ上の理由によりデフォルトで無効にされます。また、この属性のサポートは、ホスト物理マシンの接続タイプと共にゲストネットワークデバイスモデルによって異なることに注意してください。現時点では、これはホスト物理マシンの **virtio** デバイスモデルおよび **macvtap** 接続にのみサポートされています。オプションパラメーター **<trustGuestRxFilters>** を設定することが推奨される単純なケースには、ゲスト仮想マシンに対してホスト物理マシンのフィルターを制御する権限を付与する必要がある場合があります。この設定により、ゲスト仮想マシンのフィルターがホストでミラー化されるためです。

上記の属性のほかに、それぞれの **<interface>** 要素は、属性 **type='pci'** でインターフェースを特定の PCI スロットに関連付けるオプションの **<address>** サブ要素を取ります。詳細は、「[デバイスアドレス](#)」を参照してください。

#### 24.18.9.1. 仮想ネットワーク

仮想ネットワークは、動的/ワイヤレス設定を持つホスト物理マシン上の一般的なゲスト仮想マシンの接続（または、ホスト物理マシンのハードウェアの詳細が **<network>** 定義に別個に記載される複数ホスト物理マシン環境）について推奨される設定です。さらに、これは、名前付きネットワーク定義によって詳述される接続を提供します。仮想ネットワークの **forward mode** 設定に応じて、ネットワークは完全に分離するか (**<forward>** 要素の指定なし)、明示的なネットワークデバイスまたはデフォルトルートに対して NAT を実行するか (**forward mode='nat'**)、NAT なしで経路指定されるか (**forward mode='route' /**)、またはホスト物理マシンのネットワークインターフェースのいずれかに接続されるか (**macvtap** 経由)、またはブリッジデバイスに接続されます (**forward mode='bridge|private|vepa|passthrough' /**)。

ブリッジ、プライベート、**vepa**、およびパススルーの **forward** モードが指定されたネットワークの場合、ホスト物理マシンの必要な DNS および DHCP サービスが **libvirt** 外にすでにセットアップされてい

ることが想定されます。分離 (isolated)、nat および経路指定されるネットワークの場合、DHCP および DNS は、libvirt によって仮想ネットワーク上で提供され、IP 範囲は、`virsh net-dumpxml [networkname]` により仮想ネットワーク設定を検査して決定できます。デフォルトのルートに接続するために NAT を使用する既成の「デフォルト」仮想ネットワークがあり、この場合、IP 範囲は `192.168.122.0/255.255.255.0` になります。それぞれの仮想マシンには、`vnetN` の名前で作成された、関連付けられた `tun` デバイスがあり、これは、`<target>` 要素で上書きすることができます (「[ターゲット要素の上書き](#)」を参照してください)。

インターフェースのソースがネットワークの場合、ポートグループはネットワークの名前で指定することができます。1つのネットワークには複数のポートグループが定義される場合があります、それぞれのポートグループには、ネットワーク接続の異なるクラスについて若干異なる設定情報が含まれます。さらに、`<direct>` ネットワーク接続 (以下で説明) のように、タイプ `network` の接続が `<virtualport>` 要素を指定すると、`802.1Qbg` または `802.1Qbh` 互換の *Virtual Ethernet Port Aggregator* (VEPA) スイッチまたは *Open vSwitch* 仮想スイッチに設定データが転送されることがあります。

スイッチの実際のタイプは、ホスト物理マシン上の `<network>` の設定によって変わる場合があるため、`<virtualport type>` 属性を省略することができます。`<virtualport type>` は1回または複数回指定する必要があります。ドメインの起動時に、完全な `<virtualport>` 要素が、定義されるタイプと属性をマージして構成されます。これにより、仮想ポートが新たに構成されます。下位の仮想ポートからの属性は、上位の仮想ポートで定義された属性を変更することができないことに注意してください。インターフェースの優先順位が最も高くなり、ポートグループの優先順位が最も低くなります。

たとえば、`802.1Qbh` スイッチと *Open vSwitch* スイッチの両方を持つ、適切に動作するネットワークを作成するには、`no` タイプを指定する必要がある場合がありますが、`profileid` および `interfaceid` の両方を指定する必要があります。さらに、`managerid`、`typeid`、または `profileid` などの、仮想ポートから入力される他の属性はオプションになります。

ゲスト仮想マシンが特定タイプのスイッチにのみ接続するよう制限する場合、仮想ポートのタイプを指定でき、指定されたポートタイプのスイッチのみが接続されます。追加のパラメーターを指定すると、スイッチの接続はさらに制限されます。ポートが指定されており、ホスト物理マシンのネットワークに異なるタイプの仮想ポートがある場合、インターフェースの接続は失敗します。仮想ネットワークのパラメーターは、管理ツールを使用してドメイン XML の以下の部分を変更して定義されます。

```
...
<devices>
  <interface type='network'>
    <source network='default' />
  </interface>
  ...
  <interface type='network'>
    <source network='default' portgroup='engineering' />
    <target dev='vnet7' />
    <mac address='00:11:22:33:44:55' />
    <virtualport>
      <parameters instanceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f' />
    </virtualport>
  </interface>
</devices>
...
```

図24.50 デバイス - ネットワークインターフェース - 仮想ネットワーク

### 24.18.9.2. LAN のブリッジ

「ネットワークインターフェース」で言及されているように、まずこれが静的な有線ネットワーク設定を持つホスト物理マシン上の一般的なゲスト仮想マシンの接続に推奨される設定であることに注意してください。

LAN へのブリッジは、ゲスト仮想マシンから直接 LAN に接続するブリッジを提供します。これは、1 つ以上のホスト物理マシンの物理 NIC がスレーブ化されているホスト物理マシンにブリッジデバイスがあることが想定されます。ゲスト仮想マシンには、**<vnetN>** の名前で作成された、関連付けられた **tun** デバイスがあります。この名前は、**<target>** 要素で上書きできます（「ターゲット要素の上書き」を参照してください）。**<tun>** デバイスはブリッジに対してスレーブ化されます。IP 範囲/ネットワーク設定は、LAN で使用される任意の設定になります。これは、物理マシンのように、ゲスト仮想マシンに対して完全な着信および発信ネットワークアクセスを提供します。

Linux システムでは、ブリッジデバイスは、通常標準の Linux ホスト物理マシンブリッジです。Open vSwitch をサポートするホスト物理マシンでは、**virtualport type='openvswitch' /** をインターフェース定義に追加することによって Open vSwitch ブリッジデバイスに接続することもできます。Open vSwitch タイプの **virtualport** は、**parameters** 要素で以下の 2 つのパラメーターを受け入れます。1 つは、Open vSwitch へのこの特定のインターフェースを一意的に特定するために使用される標準 UUID になる **interfaceid** です（これを指定しない場合、ランダムな **interfaceid** が、インターフェースの初回定義時に生成されます）。もう 1 つはオプションの **profileid** であり、これはインターフェース **<port-profile>** として Open vSwitch に送信されます。LAN 設定へのブリッジを設定するには、管理ツールを使用してドメイン XML の以下の部分を設定します。

```
...
<devices>
  ...
  <interface type='bridge'>
    <source bridge='br0' />
  </interface>
  <interface type='bridge'>
    <source bridge='br1' />
    <target dev='vnet7' />
    <mac address='00:11:22:33:44:55' />
  </interface>
  <interface type='bridge'>
    <source bridge='ovsbr' />
    <virtualport type='openvswitch'>
      <parameters profileid='menial' interfaceid='09b11c53-8b5c-4eeb-
8f00-d84eaa0aaa4f' />
    </virtualport>
  </interface>
  ...
</devices>
```

図24.51 デバイス - ネットワークインターフェース - LAN のデバイス

### 24.18.9.3. ポートのマスカレード範囲の設定

ポートのマスカレード範囲を設定する必要がある場合に、ポートは以下のように設定できます。

```
<forward mode='nat'>
  <address start='1.2.3.4' end='1.2.3.10' />
</forward> ...
```

図24.52 ポートのマスカレード範囲

これらの値は、「[Network Address Translation](#)」に説明されるように **iptables** コマンドを使って設定される必要があります。

#### 24.18.9.4. ユーザー領域 SLIRP スタック

ユーザー領域 SLIRP スタックパラメーターを設定すると、外部への NAT が指定された仮想 LAN が提供されます。仮想ネットワークには DHCP および DNS サービスがあり、ゲスト仮想マシンに対して、10.0.2.15 から開始される IP アドレスを提供します。デフォルトのルーターは 10.0.2.2 であり、DNS サーバーは 10.0.2.3 です。このネットワーク設定は、ゲスト仮想マシンが発信アクセスを持つことを必要とする、権限のないユーザーにとって唯一のオプションです。

ユーザー領域 SLIRP スタックパラメーターは、ドメイン XML の以下の部分に定義されます。

```
...
<devices>
  <interface type='user' />
  ...
  <interface type='user'>
    <mac address="00:11:22:33:44:55" />
  </interface>
</devices>
...
```

図24.53 デバイス - ネットワークインターフェース - ユーザー領域 SLIRP スタック

#### 24.18.9.5. 汎用イーサネット接続

管理者が、ゲスト仮想マシンのネットワークを LAN に接続するための任意のスクリプトを実行するための手段を提供します。ゲスト仮想マシンには、**vnetN** という名前で作成される **<tun>** デバイスがあります。これは、**<target>** 要素で上書きすることができます。**tun** デバイスを作成した後に、シェルスクリプトが実行され、ホスト物理マシンのネットワーク統合に必要なすべてのことを実行することが予想されます。デフォルトでは、このスクリプトは **/etc/kvm-ifup** と呼ばれますが、これを上書きすることができます（「[ターゲット要素の上書き](#)」を参照してください）。

汎用イーサネット接続パラメーターは、ドメイン XML の以下の部分に定義されます。

```
...
<devices>
  <interface type='ethernet' />
  ...
  <interface type='ethernet'>
    <target dev='vnet7' />
    <script path='/etc/kvm-ifup-mynet' />
  </interface>
</devices>
...
```

図24.54 デバイス - ネットワークインターフェース - 汎用イーサネット接続

24.18.9.6. 物理インターフェースへの直接割り当て

これにより、物理インターフェースが指定されている場合、ゲスト仮想マシンの NIC がホスト物理マシンの物理インターフェースに割り当てられます。

これには、Linux macvtap ドライバーが利用可能である必要があります。**mode** 属性値 **vepa** (「Virtual Ethernet Port Aggregator」)、**bridge** または **private** のいずれかは、**macvtap** デバイスの操作モードとして選択できます。デフォルトモードは **vepa** です。

物理インターフェースへの直接割り当ての操作には、ドメイン XML の以下のセクションの次のようなパラメーターの設定を行うことが関係します。

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='vepa' />
  </interface>
</devices>
...
```

図24.55 デバイス - ネットワークインターフェース - 物理インターフェースへの直接割り当て

個々のモードにより、パケットの配信が [表24.20「物理インターフェース要素への直接割り当て」](#) に示されるような動作で行われます。

表24.20 物理インターフェース要素への直接割り当て

要素	説明
vepa	ゲスト仮想マシンのすべてのパケットは、外部ブリッジに送信されます。宛先がパケットの発信元と同じホスト物理マシン上にあるゲスト仮想マシンのパケットは、VEPA 対応ブリッジによって、ホスト物理マシンに送り戻されます (最近のブリッジは VEPA 対応でない場合が多い)。



要素	説明
<b>bridge</b>	宛先が発信元と同じホスト物理マシン上にあるパケットは、ターゲットの <b>macvtap</b> デバイスに直接配信されます。直接配信されるようにするには、発信元および宛先デバイスの両方を <b>bridge</b> モードにする必要があります。これらのいずれかが <b>vepa</b> モードの場合は、 <b>VEPA</b> 対応のブリッジが必要になります。
<b>private</b>	すべてのパケットは外部ブリッジに送信されます。それらが同じホスト物理マシンのターゲット VM に送信されるのは、それらが外部ルーターまたはゲートウェイ経由で送信され、そのデバイスがそれらをホスト物理マシンに送り戻す場合のみです。ソースまたは宛先デバイスのいずれかが <b>private</b> モードの場合に、この手順が実行されます。
<b>passthrough</b>	この機能は、 <b>SR-IOV</b> 対応の NIC の仮想機能を、移行機能を失わずにゲスト仮想マシンに直接割り当てます。すべてのパケットは、設定されたネットワークデバイスの <b>VF/IF</b> に送信されます。デバイスの機能に応じて、追加の前提条件または制限が適用される場合があります。たとえば、これにはカーネル <b>2.6.38</b> 以上が必要になります。

直接割り当てられる仮想マシンのネットワークアクセスは、ホスト物理マシンの物理インターフェースが接続されるハードウェアスイッチで管理できます。

スイッチが **IEEE 802.1Qbg** 標準に設定されている場合は、インターフェースに以下に示すように追加パラメータを持たせることができます。 **virtualport** エレメントのパラメータについては **IEEE 802.1Qbg** 標準の記載をご覧ください。その値についてはネットワーク固有となるため、ネットワーク管理者にお問い合わせください。 **802.1Qbg** では、 **VIS (Virtual Station Interface)** は仮想マシンの仮想インターフェースのことを指します。

**IEEE 802.1Qbg** の場合、 **VLAN ID** にゼロ以外の値が必要になります。

操作可能な追加の要素については [表24.21 「物理インターフェースの追加要素への直接割り当て」](#) に説明されています。

**表24.21 物理インターフェースの追加要素への直接割り当て**

要素	説明
<b>managerid</b>	<b>VSI Manager ID</b> は、 <b>VSI</b> タイプおよびインスタンス定義を含むデータベースを特定します。これは、整数値で、値 <b>0</b> は予約されています。
<b>typeid</b>	<b>VSI Type ID</b> でネットワークアクセスの特性を示す <b>VSI</b> タイプを識別します。 <b>VSI</b> タイプは一般的にはネットワーク管理者によって管理されます。整数の値になります。

要素	説明
typeidversion	VSI Type Version では VSI Type の複数のバージョンを許可します。整数の値になります。
instanceid	VSI Instance ID 識別子は、VSI インスタンス (つまり、仮想マシンの仮想インターフェース) の作成時に生成されます。これは、グローバルに固有な識別子です。
profileid	プロファイル ID には、このインターフェースに適用されるポートプロファイル名が含まれます。この名前は、ポートプロファイルのデータベースによってポートプロファイルからネットワークパラメーターに解決され、このネットワークパラメーターがこのインターフェースに適用されます。

ドメイン XML の追加パラメーターには以下が含まれます。

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0.2' mode='vepa' />
    <virtualport type="802.1Qbg">
      <parameters managerid="11" typeid="1193047" typeidversion="2"
instanceid="09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f" />
    </virtualport>
  </interface>
</devices>
...
```

図24.56 デバイス - ネットワークインターフェース-物理インターフェースの追加パラメーターへの直接割り当て

インターフェースには、スイッチが IEEE 802.1Qbh 標準に準拠している場合は、以下に示す追加のパラメーターを設定できます。値はネットワーク固有のものであるため、ネットワーク管理者にお問い合わせください。

ドメイン XML の追加パラメーターには以下が含まれます。

```

...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private' />
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance' />
    </virtualport>
  </interface>
</devices>
...

```

図24.57 デバイス - ネットワークインターフェース - 物理インターフェースの追加パラメーターへの直接割り当て

**profileid** 属性には、このインターフェースに適用されるポートプロファイルの名前が含まれます。この名前はポートプロファイルのデータベースによって、ポートプロファイルからネットワークパラメーターに解決され、それらのネットワークパラメーターはこのインターフェースに適用されます。

#### 24.18.9.7. PCI パススルー

PCI ネットワークデバイス (**source** 要素で指定される) は、最初にオプションでデバイスの **MAC** アドレスを設定済みの値に設定し、オプションで指定した **virtualport** 要素を使用してデバイスを **802.1Qbh** 対応のスイッチに関連付け (上記の仮想ポートの例にある **type='direct'** ネットワークデバイスを参照) た後に、汎用デバイスパススルーを使用してゲスト仮想マシンに直接割り当てます。標準の単一ポート **PCI** イーサネットカードドライバーの設計上の制限により、**SR-IOV** (単一 **Root I/O** 仮想化) 仮想化機能 (**VF**) デバイスのみについてこの方法で割り当てられることに注意してください。標準の単一ポート **PCI** または **PCIe** イーサネットカードをゲスト仮想マシンに割り当てるには、従来の **hostdev** デバイス定義を使用します。

ネットワークデバイスのこの「インテリジェントなパススルー」が標準の **hostdev** デバイスの機能に非常に似ていることに注意してください。相違点は、このメソッドではパススルーデバイスの **MAC** アドレスと **virtualport** を指定することを許可する点です。これらの機能が不要な場合で、**SR-IOV** をサポートしない (そのため、ゲスト仮想マシンのドメインに割り当てられた後のリセット時に設定済みの **MAC** アドレスが失われる) 標準の単一ポート **PCI**、**PCIe**、または **USB** ネットワークを持つ場合や、**0.9.11** より前の **libvirt** のバージョンを使用している場合、**interface type='hostdev' /** ではなく、標準の **hostdev** 定義を使用してデバイスをゲスト仮想マシンに割り当てする必要があります。



```

...
<devices>
  <interface type='hostdev'>
    <driver name='vfio' />
    <source>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07'
function='0x0' />
    </source>
    <mac address='52:54:00:6d:90:02'>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance' />
    </virtualport>
    </interface>
  </devices>
...

```

図24.58 デバイス - ネットワークインターフェース - PCI パススルー

#### 24.18.9.8. マルチキャストトンネル

マルチキャストグループは、仮想ネットワークを表示するために使用することができます。ネットワークデバイスが同じマルチキャストグループ内にあるゲスト仮想マシンは、それらが複数の物理的なホスト物理マシンにまたがって存在する場合でも相互に通信します。このモードは、権限を持たないユーザーとして使用することができます。デフォルトの DNS または DHCP サポートや、発信ネットワークアクセスはありません。発信ネットワークアクセスを提供するには、適切なルートを指定するため、ゲスト仮想マシンのいずれかに最初の 4 つのネットワークタイプのいずれかに接続される 2 番目の NIC がなければなりません。マルチキャストプロトコルは、**user mode linux** ゲスト仮想マシンによって使用されるプロトコルと互換性もあります。マルチキャストトンネルは、管理ツールを使用して **interface type** を操作し、これを **mcast** に設定/変更し、さらに **mac address** および **source address** を指定することによって作成されます。

```

...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
    <source address='230.0.0.1' port='5558' />
  </interface>
</devices>
...

```

図24.59 デバイス - ネットワークインターフェース - マルチキャストトンネル

#### 24.18.9.9. TCP トンネル

TCP クライアント/サーバーアーキテクチャを作成する方法は 1 つのゲスト仮想マシンがネットワークのサーバーエンドを提供し、その他のすべてのゲスト仮想マシンがクライアントとして設定される仮想ネットワークを提供するもう 1 つの方法になります。ゲスト仮想マシン間のすべてのネットワークトラフィックは、サーバーとして設定されるゲスト仮想マシンを経由で経路指定されます。このモデルも、権限のないユーザーが使用することができます。デフォルトの DNS や DHCP サポートはなく、発信ネットワークアクセスもありません。発信ネットワークアクセスを提供するには、ゲスト仮想マシンの 1 つに、最初の 4 つのネットワークタイプのいずれかに接続され、適切なルートを提供している 2 番

目のNICがある必要があります。TCPトンネルは、管理ツールを使用して **interface type** を操作し、これを **mcast** に設定/変更し、さらに **mac address** および **source address**などを指定して作成されます。

```
...
<devices>
  <interface type='server'>
    <mac address='52:54:00:22:c9:42'>
      <source address='192.168.0.1' port='5558' />
    </interface>
  ...
  <interface type='client'>
    <mac address='52:54:00:8b:c9:51'>
      <source address='192.168.0.1' port='5558' />
    </interface>
  </devices>
...
```

図24.60 デバイス - ネットワークインターフェース - TCP トンネル

24.18.9.10. NIC ドライバー固有オプションの設定

一部のNICには調整可能なドライバー固有のオプションがある場合があります。これらのオプションは、インターフェース定義の **driver** サブ要素の属性として設定されます。これらのオプションは、管理ツールでドメインXMLの以下のセクションを設定して設定されます。

```
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet1' />
    <model type='virtio' />
    <driver name='vhost' txmode='iothread' ioeventfd='on'
event_idx='off' />
  </interface>
</devices>
...
```

図24.61 デバイス - ネットワークインターフェース - NIC ドライバー固有オプションの設定

以下の属性が「virtio」NIC ドライバーに使用できます。

表24.22 virtio NIC ドライバー要素

パラメーター	説明
--------	----

パラメーター	説明
<b>name</b>	オプションの <b>name</b> 属性は、使用するバックエンドドライバのタイプを強制します。値は、 <b>kvm</b> (ユーザー空間バックエンド) または <b>vhost</b> (vhost モジュールをカーネルで指定するように要求するカーネルバックエンド) のいずれかになります。カーネルサポートなしに <b>vhost</b> ドライバーを要求する試みは拒否されます。 <b>vhost</b> ドライバーが存在する場合、デフォルト設定は <b>vhost</b> ですが、存在しない場合はサイレントに <b>kvm</b> にフォールバックします。
<b>txmode</b>	送信バッファ一杯の場合に、パケット送信の処理方法を指定します。値は、 <b>iothread</b> または <b>timer</b> のいずれかに設定できます。 <b>iothread</b> に設定される場合、 <b>packet tx</b> がドライバーの下半分の <b>iothread</b> ですべて実行されます (このオプションは、" <b>tx=bh</b> " を <b>kvm</b> コマンドラインの " <b>-device virtio-net-pci</b> " オプションに変換します)。 <b>timer</b> に設定される場合、 <b>tx</b> の作業は <b>KVM</b> で実行され、実行時間に送信できる以上の <b>tx</b> データがある場合、タイマーが <b>KVM</b> が他のタスクの実行に移行する前に設定されます。タイマーが切れると、さらに多くのデータを送信するための別の試行が行われます。通常、この値を変更しないことが推奨されています。
<b>ioeventfd</b>	インターフェースデバイスのドメイン I/O の非同期処理を設定します。デフォルトはハイパーバイザーに決定されます。許可される値は、 <b>on</b> と <b>off</b> です。このオプションを有効にすると、別のスレッドが I/O を処理する間、 <b>KVM</b> はゲスト仮想マシンを実行することができます。通常、I/O 時にシステム CPU の使用率が高くなるゲスト仮想マシンの場合に、この利点があります。他方、物理ホストマシンに過剰な負荷を与えると、ゲスト仮想マシンの I/O 待機時間も増えることになります。そのため、この値を変更することは推奨されません。
<b>event_idx</b>	<b>event_idx</b> 属性は、デバイスイベント処理の複数の側面を制御します。この値は <b>on</b> または <b>off</b> のいずれかにすることができます。 <b>on</b> はデフォルトで、ゲスト仮想マシンの割り込みおよび終了の数を減らします。この動作が最適化されていない状況では、この属性はこの機能をオフに強制実行する方法を提供します。この値を変更することは推奨されません。

#### 24.18.9.11. ターゲット要素の上書き

ターゲット要素を上書きするには、管理ツールを使用して、ドメイン XML に以下の変更を行います。

```
...
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet1' />
  </interface>
</devices>
...
```

図24.62 デバイス - ネットワークインターフェース - ターゲット要素の上書き

ターゲットが指定されない場合、一部のハイパーバイザーは、作成された **tun** デバイスの名前を自動的に生成します。この名前は手動で指定できますが、名前は **vnet** または **vif** のいずれでも開始することができません。これらは **libvirt** と一部のハイパーバイザーで予約されるプレフィックスであるためです。これらのプレフィックスを使用して手動で指定されたターゲットは無視されます。

#### 24.18.9.12. 起動順序の指定

起動順序を指定するには、管理ツールを使用してドメイン XML に以下の変更を行います。

```
...
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet1' />
    <boot order='1' />
  </interface>
</devices>
...
```

図24.63 起動順序の指定

これをサポートするハイパーバイザーの場合、ネットワークの起動に使用される特定の **NIC** を指定できます。属性の順序は、ブートシーケンス時にデバイスが試行される順序を決定します。デバイスごとの **boot** 要素は、**BIOS** ブートローダーセクションの一般的な **boot** 要素と共に使用することができないことに注意してください。

#### 24.18.9.13. インターフェース ROM BIOS 設定

ROM BIOS 設定を指定するには、管理ツールを使用してドメイン XML に以下のような変更を行います。

```

...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </interface>
</devices>
...

```

図24.64 インターフェース ROM BIOS 設定

これをサポートするハイパーバイザーの場合、PCI ネットワークデバイスの ROM をゲスト仮想マシンに提示する方法を変更することができます。**bar** 属性は **on** または **off** に設定することができ、デバイスの ROM がゲスト仮想マシンのメモリーマップに表示されるかどうかを決定します (PCI 資料によると、**rom bar** 設定は ROM の Base Address Register の提示を制御します)。**rom bar** が指定されない場合、KVM デフォルトが使用されます (古いバージョンの KVM はデフォルトの **off** を使用しましたが、新しい KVM ハイパーバイザーにはデフォルトの **on** が指定されます)。オプションの **file** 属性は、デバイスの ROM BIOS としてゲスト仮想マシンに提示されるバイナリファイルのポイントするために使用されます。これは、ネットワークデバイスの代替ブート ROM を指定する際に便利です。

#### 24.18.9.14. Quality of service (QoS)

Quality of service (QoS) を設定するために、着信および発信トラフィックは別個に形成できます。**bandwidth** 要素には、最大で1つの着信、および最大で1つの発信の子要素を持たせることができます。これらの子のいずれかを省略すると、そのトラフィック方向に QoS が適用されなくなります。そのため、ドメインの着信トラフィックのみを設定する場合には、着信のみを使用し、逆の場合も同じようにします。

これらの要素のいずれかには1つの必須属性 **average** (または下記の **floor**) があります。**Average** は、設定されるインターフェースの平均的なビットレートを指定します。次に、以下のような2つのオプション属性を持ちます。**peak**: この属性は、ブリッジが1秒あたりキロバイト単位でデータを送信できる最大レートを指定します。この実装の制限としては、着信要素のこの属性は、Linux のイングレスフィルターがこれを認識しないため、無視される点にあります。**burst** はピークの速度でバーストできるバイト量を指定します。属性に使用できる値は整数です。

**average** および **peak** 属性の単位は1秒あたりキロバイトとなり、**burst** はキロバイト単位でのみ設定されます。さらに、着信トラフィックには、オプションで **floor** 属性を持たせることができます。これは、形成されるインターフェースの最小スループットを保証します。**floor** の使用には、すべてのトラフィックが、QoS による決定が行われる1つのポイントを経由することが必要になります。そのため、これは、**interface type='network' /** に **forward** タイプの **route**、**nat**、または **no forward** が指定されている場合にのみ使用できます。仮想ネットワーク内で、すべての接続されたインターフェースには着信 QoS (少なくとも **average**) が設定される必要があるものの、**floor** 属性には **average** を指定する必要があることに注意する必要があります。ただし、**peak** および **burst** 属性には依然として **average** が必要です。現在、**ingress qdiscs** にはいずれのクラスも設定できないため、**floor** は発信トラフィックではなく、着信にのみ適用できます。

QoS 設定を指定するには、ドメイン XML に以下のような変更を行うために管理ツールを使用します。

```

...
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet0' />
    <bandwidth>
      <inbound average='1000' peak='5000' floor='200' burst='1024' />
      <outbound average='128' peak='256' burst='256' />
    </bandwidth>
  </interface>
</devices>
...

```

図24.65 QoS (Quality of Service)

#### 24.18.9.15. VLAN タグの設定 (サポートされるネットワークタイプでのみ)

VLAN 設定を指定するには、ドメイン XML に以下のような変更を行うために管理ツールを使用します。

```

...
<devices>
  <interface type='bridge'>
    <vlan>
      <tag id='42' />
    </vlan>
    <source bridge='ovsbr0' />
    <virtualport type='openvswitch'>
      <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f' />
    </virtualport>
  </interface>
</devices>
...

```

図24.66 VLAN タグの設定 (サポートされるネットワークタイプでのみ)

ゲスト仮想マシンが使用するネットワーク接続がゲスト仮想マシンに透過的な VLAN タグに対応している場合、オプションの **vlan** 要素は、ゲスト仮想マシンのネットワークトラフィックに適用される 1 つ以上の VLAN タグを指定できます。OpenvSwitch および **type='hostdev'** SR-IOV インターフェースのみがゲスト仮想マシンの透過的な VLAN タグに対応します。標準的な Linux ブリッジおよび libvirt の独自ネットワークを含む他のインターフェースはこれに対応しません。802.1Qbh (vn-link) および 802.1Qbg (VEPA) スイッチは独自のメソッド (libvirt 外) を提供し、ゲスト仮想マシントラフィックを特定の VLAN にタグ付けします。複数タグの指定を許可する場合 (VLAN とランキングの場合)、**tag** サブ要素は使用する VLAN タグを指定します (例: **tag id='42' />**)。インターフェースに複数の **vlan** 要素が定義されている場合、ユーザーは指定されるすべてのタグを使用して VLAN トランキングの実行を必要としていることが想定されます。単一タグを持つ VLAN トランキングが必要な場合、オプションの属性 **trunk='yes'** を上位の **vlan** 要素に追加することができます。

#### 24.18.9.16. 仮想リンク状態の変更

この要素は、仮想ネットワークリンクの状態を設定します。属性の **state** に使用できる値は、**up** および **down** です。**down** が値として指定される場合、インターフェースは、ネットワークケーブルが切断

されているかのように動作します。この要素が指定されていない場合のデフォルト動作では、リンク状態が **up** になります。

仮想リンク状態の設定を指定するには、ドメイン XML に以下のような変更を行うために管理ツールを使用します。

```
...
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet0' />
    <link state='down' />
  </interface>
</devices>
...
```

図24.67 仮想リンク状態の変更

#### 24.18.10. 入力デバイス

入力デバイスは、ゲスト仮想マシンのグラフィカルフレームバッファーとの対話を許可します。フレームバッファーを有効にする場合、入力デバイスが自動的に提供されます。絶対的なカーソル移動のためにグラフィックスタブレットを提供するなどの目的で、追加デバイスを明示的に追加することができます。

入力デバイスの設定を指定するには、ドメイン XML に以下のような変更を行うために管理ツールを使用します。

```
...
<devices>
  <input type='mouse' bus='usb' />
</devices>
...
```

図24.68 入力デバイス

**<input>** 要素には、必須属性の **type** があります。この属性は **mouse** または **tablet** に設定できます。**mouse** は相対的なカーソル移動を使用するのに対し、**tablet** は絶対的なカーソル移動を提供します。オプションの **bus** 属性は、正確なデバイスタイプを詳細化するために使用し、**kvm** (準仮想化)、**ps2**、および **usb** に設定できます。

**input** 要素には、上記のようにデバイスを特定の PCI スロットに関連付けることのできるオプションのサブ要素 **<address>** があります。

#### 24.18.11. ハブデバイス

ハブは、デバイスをホスト物理マシンシステムに接続するために利用できるポートが増えるよう、単一ポートを複数に拡張するデバイスです。

ハブデバイスの設定を指定するには、ドメイン XML に以下のような変更を行うために管理ツールを使用します。



```

...
<devices>
  <hub type='usb' />
</devices>
...

```

図24.69 ハブデバイス

**hub** 要素には1つの必須属性があり、**type** は **usb** のみに設定できます。**hub** 要素にはオプションのサブ要素 **address** があり、これにはデバイスを特定のコントローラーに関連付けることのできる **type='usb'** が付きます。

## 24.18.12. グラフィカルフレームバッファ

グラフィックスデバイスは、ゲスト仮想マシン OS とのグラフィカルな対話を可能にします。ゲスト仮想マシンは、通常はユーザーとの対話を可能にするために設定されるフレームバッファか、またはテキストコンソールのいずれかを持ちます。

グラフィカルフレームバッファデバイスの設定を指定するには、ドメイン XML に以下のような変更を行うために管理ツールを使用します。

```

...
<devices>
  <graphics type='sdl' display=':0.0' />
  <graphics type='vnc' port='5904'>
    <listen type='address' address='1.2.3.4' />
  </graphics>
  <graphics type='rdp' autoport='yes' multiUser='yes' />
  <graphics type='desktop' fullscreen='yes' />
  <graphics type='spice'>
    <listen type='network' network='rednet' />
  </graphics>
</devices>
...

```

図24.70 グラフィカルフレームバッファ

**graphics** 要素には、必須の **type** 属性があります。これは、以下に説明するように、値の **sdl**、**vnc**、**rdp**、**desktop** または **spice** を取ります。

表24.23 グラフィカルフレームバッファの主要な要素

パラメーター	説明
--------	----



パラメーター	説明
<b>sdl</b>	<p>これは、ホスト物理マシンデスクトップのウィンドウを表示します。また、以下のオプションの引数を受け入れます。</p> <ul style="list-style-type: none"> <li>• ディスプレイで使用する <b>display</b> 属性</li> <li>• 認証 ID の <b>xauth</b> 属性</li> <li>• 値 <b>yes</b> または <b>no</b> を受け入れるオプションの <b>fullscreen</b> 属性</li> </ul>
<b>vnc</b>	<p>VNC サーバーを起動します。</p> <ul style="list-style-type: none"> <li>• <b>port</b> 属性は TCP ポート番号を指定します (自動割り当てであることを示すレガシー構文として <b>-1</b> を指定する)。</li> <li>• <b>autoport</b> 属性は、使用する TCP ポートの自動割り当てを示すために推奨される構文です。</li> <li>• <b>listen</b> 属性は、サーバーがリッスンするために使用される IP アドレスです。</li> <li>• <b>passwd</b> 属性は、クリアテキストで VNC パスワードを提供します。</li> <li>• <b>keymap</b> 属性は、使用するキーマップを指定します。パスワードの有効性についての制限は、<b>timestamp passwdValidTo='2010-04-09T15:51:00'</b> を UTC に指定するなどして、設定することができます。</li> <li>• <b>connected</b> 属性は、パスワードの変更時に接続されたクライアントの制御を可能にします。VNC は <b>keep</b> 値のみを受け入れます。これはすべてのハイパーバイザーではサポートできないことに注意してください。</li> <li>• リッスン/ポートを使用する代わりに、KVM は UNIX ドメインソケットパスでリッスンするためのソケット属性をサポートします。</li> </ul>

パラメーター	説明
<b>spice</b>	<p>SPICE サーバーを起動します。</p> <ul style="list-style-type: none"> <li>● <b>port</b> 属性は TCP ポート番号を指定し (自動割り当てであることを示すレガシー構文として <b>-1</b> を指定する)、<b>tlsPort</b> は別のセキュアポート番号を指定します。</li> <li>● <b>autoport</b> 属性は、両方のポート番号の自動割り当てを示すための優先される新規の構文です。</li> <li>● <b>listen</b> 属性は、サーバーがリスンするために使用される IP アドレスです。</li> <li>● <b>passwd</b> 属性は、クリアテキストで SPICE パスワードを提供します。</li> <li>● <b>keymap</b> 属性は、使用するキーマップを指定します。パスワードの有効性についての制限は、<b>timestamp</b> <b>passwdValidTo='2010-04-09T15:51:00'</b> を UTC に指定するなどして、設定することができます。</li> <li>● <b>connected</b> 属性は、パスワードの変更時に接続されたクライアントの制御を可能にします。SPICE ではクライアントを接続した状態にするために <b>keep</b> を使用し、クライアントの接続を解除するには <b>disconnect</b>、またパスワードの変更の失敗時に <b>fail</b> を使用します。これはすべてのハイパーバイザーでサポートされている訳ではないことに注意してください。</li> <li>● <b>defaultMode</b> 属性は、デフォルトのチャネルセキュリティポリシーを設定し、有効な値は <b>secure</b>、<b>insecure</b> でデフォルトは <b>any</b> です。(可能な場合は <b>secure</b> ですが、セキュアパスが利用可能でない場合はエラーを出す代わりに <b>insecure</b> にフォールバックします)。</li> </ul>

SPICE に標準の、および TLS で保護された TCP ポートの両方が設定されている場合、各ポートでどのチャネルを実行できるかについて制限することが必要になる場合があります。これは、1つ以上の **channel** 要素をメイン **graphics** 要素内に追加することによって実行されます。有効なチャネル名には、**main**、**display**、**inputs**、**cursor**、**playback**、**record**、**smartcard**、および **usbredir** があります。

SPICE 設定を指定するには、ドメイン XML に以下のような変更を行うために管理ツールを使用します。

```
<graphics type='spice' port='-1' tlsPort='-1' autoport='yes'>
  <channel name='main' mode='secure' />
  <channel name='record' mode='insecure' />
  <image compression='auto_glz' />
  <streaming mode='filter' />
  <clipboard copypaste='no' />
  <mouse mode='client' />
</graphics>
```

図24.71 SPICE 設定の例

SPICE は、オーディオ、イメージおよびストリーミングの可変的な圧縮設定をサポートします。これらの設定は、以下の要素の **compression** 属性で行われます。

- イメージ圧縮を設定するための **image** (**auto\_glz**、**auto\_lz**、**quic**、**glz**、**lz**、**off** を許可)
- WAN 経由でのイメージ JPEG 圧縮用の **jpeg** (**auto**、**never**、**always** を許可)
- WAN イメージ圧縮を設定するための **zlib** (**auto**、**never**、**always** を許可) およびオーディオストリーム圧縮を有効にするための **playback** (**on** または **off** を許可)。

ストリーミングモードは **streaming** 要素によって設定され、その **mode** 属性は、**filter**、**all** または **off** のいずれかに設定されます。

さらに、コピーアンドペースト機能 (SPICE エージェント経由) は **clipboard** 要素によって設定されます。これはデフォルトで有効にされ、**copypaste** プロパティを **no** に設定することによって無効にできます。

マウスモードは **mouse** 要素によって設定され、その **mode** 属性は **server** または **client** のいずれかに設定されます。いずれのモードも指定されない場合、KVM のデフォルトが使用されます (**client** モード)。

追加の要素には以下が含まれます。

表24.24 追加のグラフィカルフレームバッファ要素

パラメーター	説明
--------	----

パラメーター	説明
<b>rdp</b>	<p>RDP サーバーを起動します。</p> <ul style="list-style-type: none"><li>● <b>port</b> 属性は TCP ポート番号を指定します (自動割り当てであることを示すレガシー構文として <b>-1</b> を指定する)。</li><li>● <b>autoport</b> 属性は、使用する TCP ポートの自動割り当てを示すために推奨される構文です。</li><li>● <b>replaceUser</b> 属性は、仮想マシンへの複数の同時接続が許可されるかどうかを決定するブール値です。</li><li>● <b>multiUser</b> は、新規クライアントが単一接続モードで接続される場合に、既存の接続がドロップされるかどうか、また新規接続が VRDP サーバーによって確立される必要があるかどうかを決定するために使用されます。</li></ul>
<b>desktop</b>	<p>この値は、現在 <b>VirtualBox</b> ドメイン用に保持されます。これは <b>SDL</b> と同様にホスト物理マシンのデスクトップ上にウィンドウを表示しますが、<b>VirtualBox</b> ビューアーを使用します。<b>SDL</b> の場合と同様に、これはオプションの属性の <b>display</b> および <b>fullscreen</b> を受け入れます。</p>

パラメーター	説明
<b>listen</b>	<p>グラフィックスタイプの <b>vnc</b> および <b>spice</b> 用にリスンするソケットをセットアップするために使用されるアドレス情報を入れる代わりに、<b>listen</b> 属性という <b>graphics</b> のサブ要素を指定できます (上記の例を参照)。<b>listen</b> は以下の属性を受け入れます。</p> <ul style="list-style-type: none"> <li>• <b>type</b> - アドレスまたはネットワークのいずれかに設定されます。これは、この <b>listen</b> 要素が直接使用されるアドレスを指定するか、またはネットワークに名前を付けて (それがリスンする適切なアドレスを決定するために使用される) 指定されるかどうかを指定します。</li> <li>• <b>address</b> - この属性には、リスンする IP アドレスまたはホスト名 (DNS 照会により IP アドレスに解決される) のいずれかが含まれます。実行中のドメインの「ライブ」XML の場合、この属性は、<b>type='network'</b> の場合でも、リスンするために使用される IP アドレスに設定されます。</li> <li>• <b>network - type='network'</b> の場合、<b>network</b> 属性には、設定済みネットワークの <b>libvirt</b> のリストにネットワークの名前が含まれます。名前付きのネットワーク設定は、適切なリスンアドレスを決定するために検査されます。たとえば、ネットワークがその設定に <b>IPv4</b> アドレスを持つ場合 (たとえば、ルートの <b>forward</b> タイプまたは <b>NAT</b>、または <b>no forward</b> タイプ (<b>isolated</b>) の場合)、ネットワークの設定にリストされる最初の <b>IPv4</b> アドレスが使用されます。ネットワークがホスト物理マシンのブリッジを記述する場合、そのブリッジデバイスに関連付けられた最初の <b>IPv4</b> アドレスが使用され、ネットワークが <b>'direct'</b> (<b>macvtap</b>) モードのいずれかを記述する場合には、最初の <b>forward dev</b> の最初の <b>IPv4</b> アドレスが使用されます。</li> </ul>

### 24.18.13. ビデオデバイス

ビデオデバイスの設定を指定するには、ドメイン XML に以下のような変更を行うために管理ツールを使用します。

```

...
<devices>
  <video>
    <model type='vga' vram='8192' heads='1'>
      <acceleration accel3d='yes' accel2d='yes' />
    </model>
  </video>
</devices>
...

```

図24.72 ビデオデバイス

**graphics** 要素には必須の **type** 属性があり、これは、以下に説明するように値 **"sdl"**、**"vnc"**、**"rdp"** または **"desktop"** を取ります。

表24.25 グラフィカルフレームバッファー要素

パラメーター	説明
<b>video</b>	<b>video</b> 要素は、ビデオデバイスを記述するためのコンテナです。下位互換性のために <b>video</b> が設定されていないものの、ドメイン XML に <b>graphics</b> 要素がある場合に、libvirt はゲスト仮想マシンタイプに基づいてデフォルトの <b>video</b> を追加します。 <b>"ram"</b> または <b>"vram"</b> が指定されていない場合、デフォルト値が使用されます。
<b>model</b>	これには必須の <b>type</b> 属性があり、この属性は利用可能なハイパーバイザー機能に応じて、値の <b>vga</b> 、 <b>cirrus</b> 、 <b>vmvga</b> 、 <b>kvm</b> 、 <b>vbox</b> 、または <b>qxl</b> を取ります。さらに、ビデオメモリーの容量は、ヘッドと共に <b>vram</b> および図の番号を使用してキビバイト単位 (1024 バイトのブロック) で指定できます。
<b>acceleration</b>	<b>acceleration</b> がサポートされる場合、 <b>acceleration</b> 要素で <b>accel3d</b> と <b>accel2d</b> 属性を使用してこれを有効にする必要があります。
<b>address</b>	オプションの <b>address</b> サブ要素は、ビデオデバイスを特定の PCI スロットに関連付けるために使用することができます。

#### 24.18.14. コンソール、シリアル、およびチャネルデバイス

キャラクターデバイスは、仮想マシンと対話する方法を提供します。準仮想化コンソール、シリアルポート、およびチャネルは、すべてキャラクターデバイスとして分類されるため、同じ構文を使用して表示されます。

コンソール、チャネルおよびその他のデバイス設定内容を指定するには、ドメイン XML に以下のような変更を行うために管理ツールを使用します。

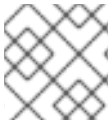
```

...
<devices>
  <serial type='pty'>
    <source path='/dev/pts/3' />
    <target port='0' />
  </serial>
  <console type='pty'>
    <source path='/dev/pts/4' />
    <target port='0' />
  </console>
  <channel type='unix'>
    <source mode='bind' path='/tmp/guestfwd' />
    <target type='guestfwd' address='10.0.2.1' port='4600' />
  </channel>
</devices>
...

```

図24.73 コンソール、シリアル、およびチャネルデバイス

これらのディレクティブのそれぞれで、トップレベルの要素名 (**serial**, **console**, **channel**) は、ゲスト仮想マシンにデバイスがどのように提示されるかを記述します。ゲスト仮想マシンのインターフェースは、**target** 要素によって設定されます。ホスト物理マシンに提示されるインターフェースは、トップレベル要素の **type** 属性で指定されます。ホスト物理マシンのインターフェースは、**source** 要素によって設定されます。**source** 要素には、ソケットパスでラベルが実行される方法を上書きするためのオプションの **seclabel** が含まれます。この要素がない場合、セキュリティーラベルはドメインごとの設定から継承されます。それぞれのキャラクターデバイス要素には、オプションのサブ要素 **address** が含まれ、これはデバイスを特定のコントローラーまたは PCI スロットに関連付けることができます。



#### 注記

パラレルポート、および **isa-parallel** デバイスはサポート対象外になりました。

### 24.18.15. ゲスト仮想マシンのインターフェース

キャラクターデバイスは、以下のタイプのいずれかとして、自らをゲスト仮想マシンに提示します。

シリアルポートを設定するには、管理ツールを使用して、ドメイン XML に以下の変更を行います。

```

...
<devices>
  <serial type='pty'>
    <source path='/dev/pts/3' />
    <target port='0' />
  </serial>
</devices>
...

```

図24.74 ゲスト仮想マシンインターフェースのシリアルポート

**<target>** には、**port** 属性を持たせることができます。これはポート番号を指定します。ポートは 0 から始まる番号を付けることができます。通常は、0、1 または 2 のシリアルポートがあります。さら

に、オプションの **type** 属性があり、これは、値を選択するのに **isa-serial** と **usb-serial** の2つのオプションがあります。**type** がない場合、**isa-serial** がデフォルトで使用されます。**usb-serial** の場合、オプションのサブ要素 **<address>** と **type='usb'** は、上記のようにデバイスを特定のコントローラーに関連付けることができます。

**<console>** 要素は、インタラクティブなコンソールを表示するために使用されます。使用されるゲスト仮想マシンのタイプによって、また以下のルールに応じて、コンソールは準仮想化デバイスであるか、またはシリアルデバイスのクローンになる可能性があります。

- **targetType** 属性が設定されていない場合、デフォルトデバイスの **type** はハイパーバイザーのルールに基づきます。デフォルトの **type** は、**libvirt** にフィードされる XML を再度照会する際に追加されます。完全仮想化ゲスト仮想マシンの場合、デフォルトのデバイスタイプは通常シリアルタイプになります。
- **targetType** 属性が **serial** の場合で、**<serial>** 要素が存在しない場合、**console** 要素は **<serial>** 要素にコピーされます。**<serial>** 要素がすでに存在する場合、**console** 要素は無視されます。
- **targetType** 属性が **serial** ではない場合、それは通常の方法で処理されます。
- 最初の **<console>** 要素のみが、**serial** の **targetType** を使用できます。2 番目のコンソールはすべて準仮想化する必要があります。
- **s390** では、**console** 要素は、**sclp** または **sclplm** (ラインモード) の **targetType** を使用できます。**SCLP** は **s390** のネイティブのコンソールタイプです。**SCLP** コンソールに関連付けられたコントローラーはありません。

以下の例では、**virtio** コンソールデバイスは **/dev/hvc[0-7]** としてゲスト仮想マシン内で公開されています (詳細は、[Fedora プロジェクトの virtio-serial ページ](#) を参照してください)。



```

...
<devices>
  <console type='pty'>
    <source path='/dev/pts/4' />
    <target port='0' />
  </console>

  <!-- KVM virtio console -->
  <console type='pty'>
    <source path='/dev/pts/5' />
    <target type='virtio' port='0' />
  </console>
</devices>
...

...
<devices>
  <!-- KVM s390 sclp console -->
  <console type='pty'>
    <source path='/dev/pts/1' />
    <target type='sclp' port='0' />
  </console>
</devices>
...

```

図24.75 ゲスト仮想マシンインターフェース - virtio コンソールデバイス

コンソールがシリアルポートとして表示される場合、**<target>** 要素には、シリアルポートの場合と同じ属性があります。通常、1つのコンソールのみが存在します。

#### 24.18.16. チャンネル

これは、ホスト物理マシンとゲスト仮想マシン間のプライベートな通信チャンネルを表し、管理ツールを使用してゲスト仮想マシンに変更を加えることによって操作できます。その結果、ドメイン XML の以下のセクションに変更が加わります。

```

...
<devices>
  <channel type='unix'>
    <source mode='bind' path='/tmp/guestfwd' />
    <target type='guestfwd' address='10.0.2.1' port='4600' />
  </channel>

  <!-- KVM virtio channel -->
  <channel type='pty'>
    <target type='virtio' name='arbitrary.virtio.serial.port.name' />
  </channel>
  <channel type='unix'>
    <source mode='bind' path='/var/lib/libvirt/kvm/f16x86_64.agent' />
    <target type='virtio' name='org.kvm.guest_agent.0' />
  </channel>
  <channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0' />
  </channel>
</devices>
...

```

図24.76 チャネル

これは、各種の方法で実装できます。**<channel>**の特定のタイプは**<target>**要素の**type**属性で指定されます。異なるチャネルタイプには、以下のようにそれぞれ異なるターゲット属性があります。

- **guestfwd** - 指定された IP アドレスに対してゲスト仮想マシンにより送信される TCP トラフィックを決定し、ポートはホスト物理マシン上のチャネルデバイスに転送されます。**target** 要素には、**address** と **port** 属性があります。
- **virtio** - 準仮想化された virtio チャネルです。**<channel>** は、ゲスト仮想マシンの **/dev/vport\*** の下に公開され、オプション要素の **name** が指定される場合は **/dev/virtio-ports/\$name** に表示されます (詳細は、[Fedora プロジェクトの virtio-serial ページ](#) を参照してください)。オプション要素の **address** は、上記のように、チャネルを特定の **type='virtio-serial'** コントローラーに関連付けます。KVM では、名前が **"org.kvm.guest\_agent.0"** の場合、libvirt は、ゲスト仮想マシンのシャットダウンやファイルシステムの休止などのアクションのために、ゲスト仮想マシンにインストールされたゲストエージェントと対話することができます。
- **spicevmc** - 準仮想化された SPICE チャネルです。ドメインには、グラフィックスデバイスとしての SPICE サーバーも必要です。ここで、ホスト物理マシンは、メインチャネル間のメッセージをピギーバックできます。**target** 要素は、属性 **type='virtio'**; と共に指定される必要があり、オプション属性 **name** はゲスト仮想マシンがチャネルにアクセスする方法を制御し、デフォルトで **name='com.redhat.spice.0'** に設定されます。オプションの **<address>** 要素は、チャネルを特定の **type='virtio-serial'** コントローラーに関連付けることができます。

#### 24.18.17. ホスト物理マシンインターフェース

キャラクターデバイスは、以下のタイプのいずれかとして、自らをホスト物理マシンに提示します。

表24.26 キャラクターデバイス要素

パラメーター	説明	XML スニペット
Domain logfile	キャラクターデバイスのすべての入力を無効にし、出力を仮想マシンのログファイルに送信します。	<pre>&lt;devices&gt;   &lt;console     type='stdio'&gt;     &lt;target       port='1' /&gt;     &lt;/console&gt; &lt;/devices&gt;</pre>
Device logfile	ファイルが開かれ、キャラクターデバイスに送信されたすべてのデータがファイルに書き込まれます。宛先ディレクトリーには、正常に起動するために、この設定を持つゲストの <b>virt_log_t SELinux</b> ラベルがなければなりません。ことに注意してください。	<pre>&lt;devices&gt;   &lt;serial     type="file"&gt;     &lt;source       path="/var/log/vm/vm-serial.log"/&gt;     &lt;target       port="1"/&gt;     &lt;/serial&gt; &lt;/devices&gt;</pre>
Virtual console	キャラクターデバイスを仮想コンソールのグラフィカルフレームバッファーに接続します。通常、これは "ctrl+alt+3" などの特殊ホットキーシーケンスでアクセスされます。	<pre>&lt;devices&gt;   &lt;serial     type='vc'&gt;     &lt;target       port="1"/&gt;     &lt;/serial&gt; &lt;/devices&gt;</pre>
Null device	キャラクターデバイスを <b>void</b> に接続します。データは入力に提供されません。書き込まれたすべてのデータは破棄されます。	<pre>&lt;devices&gt;   &lt;serial     type='null'&gt;     &lt;target       port="1"/&gt;     &lt;/serial&gt; &lt;/devices&gt;</pre>
Pseudo TTY	<b>Pseudo TTY</b> は、 <b>/dev/ptmx</b> を使用して割り当てられます。 <b>virsh console</b> などの適切なプロキシは、シリアルポートとローカルに対話することができます。	<pre>&lt;devices&gt;   &lt;serial     type="pty"&gt;     &lt;source       path="/dev/pts/3"/&gt;     &lt;target       port="1"/&gt;     &lt;/serial&gt; &lt;/devices&gt;</pre>

パラメーター	説明	XML スニペット
NB Special case	NB Special case。<console type='pty'> の場合に、TTY パスも、トップレベルの <console> タグ上の属性 tty='/dev/pts/3' として複製されます。これは、<console> タグの既存の構文との互換性を提供します。	
Host physical machine device proxy	キャラクターデバイスは、基礎となる物理キャラクターデバイスに渡されます。デバイスタイプ、たとえば、エミュレートされたシリアルポートは、ホスト物理マシンのシリアルポートにのみ接続される必要があります、シリアルポートはパラレルポートに接続しないでください。	<pre>&lt;devices&gt;   &lt;serial     type="dev"&gt;       &lt;source         path="/dev/ttyS0"/&gt;       &lt;target         port="1"/&gt;     &lt;/serial&gt;   &lt;/devices&gt;</pre>
Named pipe	キャラクターデバイスは出力を名前付きパイプに書き込みます。詳細は、 <a href="#">pipe(7) man ページ</a> を参照してください。	<pre>&lt;devices&gt;   &lt;serial     type="pipe"&gt;       &lt;source         path="/tmp/mypipe"/&gt;       &lt;target         port="1"/&gt;     &lt;/serial&gt;   &lt;/devices&gt;</pre>
TCP client-server	キャラクターデバイスは、リモートサーバーに接続する TCP クライアントとして機能します。	<pre>&lt;devices&gt;   &lt;serial     type="tcp"&gt;       &lt;source         mode="connect"         host="0.0.0.0"         service="2445"/&gt;       &lt;protocol         type="raw"/&gt;       &lt;target         port="1"/&gt;     &lt;/serial&gt;   &lt;/devices&gt;</pre> <p>または、クライアント接続を待機する TCP サーバーとして機能します。</p> <pre>&lt;devices&gt;</pre>

パラメーター	説明	<div>&lt;serial XML スニペット&gt;</div>
		<div><pre>&lt;source mode="bind" host="127.0.0.1" service="2445"/&gt;   &lt;protocol type="raw"/&gt;     &lt;target port="1"/&gt;   &lt;/serial&gt; &lt;/devices&gt;</pre></div> <div>または、raw TCP の代わりに telnet を使用することができます。さらに telnets (secure telnet) および tls を使用することもできます。</div> <div><pre>&lt;devices&gt;   &lt;serial type="tcp"&gt;     &lt;source mode="connect" host="0.0.0.0" service="2445"/&gt;     &lt;protocol type="telnet"/&gt;     &lt;target port="1"/&gt;   &lt;/serial&gt;   &lt;serial type="tcp"&gt;     &lt;source mode="bind" host="127.0.0.1" service="2445"/&gt;     &lt;protocol type="telnet"/&gt;     &lt;target port="1"/&gt;   &lt;/serial&gt; &lt;/devices&gt;</pre></div>

パラメーター	説明	XML スニペット
UDP network console	キャラクターデバイスは、UDP ネットコントロールサービスとして機能し、パケットの送受信を行います。これはロスの多いサービスです。	<pre> &lt;devices&gt;   &lt;serial     type="udp"&gt;     &lt;source       mode="bind"       host="0.0.0.0"       service="2445"/&gt;     &lt;source       mode="connect"       host="0.0.0.0"       service="2445"/&gt;     &lt;target       port="1"/&gt;     &lt;/serial&gt;   &lt;/devices&gt; </pre>
UNIX domain socket client-server	キャラクターデバイスは、UNIX ドメインソケットサーバーとして機能し、ローカルクライアントからの接続を受け入れます。	<pre> &lt;devices&gt;   &lt;serial     type="unix"&gt;     &lt;source       mode="bind"       path="/tmp/foo"/&gt;     &lt;target       port="1"/&gt;     &lt;/serial&gt;   &lt;/devices&gt; </pre>

### 24.18.18. サウンドデバイス

仮想サウンドカードは、**sound** 要素により、ホスト物理マシンに割り当てることができます。

```

...
<devices>
  <sound model='ac97' />
</devices>
...

```

図24.77 仮想サウンドカード

**sound** 要素には、必須属性の **model** があります。これは、エミュレートする実際のサウンドデバイスを指定します。有効な値は、典型的なオプションとして '**sb16**'、'**ac97**'、および '**ich6**' がありますが、基礎となるハイパーバイザーに特有のものとなります。さらに、'**ich6**' モデルを持つ **sound** 要素は、オーディオデバイスに各種の音声コーデックを割り当てるためのオプションのサブ要素 **codec** を持たせることができます。指定されない場合、デフォルトのコーデックが、再生や録音を可能にするために割り当てられます。有効な値は、'**duplex**' (line-in および line-out を提供) および '**micro**' (speaker および microphone を提供) です。

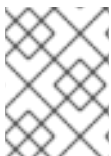
```

...
<devices>
  <sound model='ich6'>
    <codec type='micro' />
  </sound>
</devices>
...

```

図24.78 サウンドデバイス

それぞれの **sound** 要素にはオプションのサブ要素 **<address>** があり、これは、上記のようにデバイスを特定の PCI スロットに関連付けることができます。



## 注記

es1370 音声デバイスは Red Hat Enterprise Linux 7 ではサポートされません。代わりに ac97 を使用してください。 |

## 24.18.19. ウォッチドッグデバイス

仮想ハードウェアウォッチドッグデバイスは、**<watchdog>** 要素でゲスト仮想マシンに追加できます。ウォッチドッグデバイスは、ゲスト仮想マシン内に追加のドライバーおよび管理デーモンを必要とします。現在、ウォッチドッグが実行された場合の通知サポートはありません。

```

...
<devices>
  <watchdog model='i6300esb' />
</devices>
...

...
<devices>
  <watchdog model='i6300esb' action='poweroff' />
</devices>
...

```

図24.79 ウォッチドッグデバイス

以下の属性がこの XML で宣言されます。

- **model** - 必須の **model** 属性は、エミュレートする実際のウォッチドッグを指定します。有効な値は、基礎となるハイパーバイザーに固有です。
- **model** 属性は以下の値を取ることができます。
  - **i6300esb** – 推奨されるデバイスです。PCI Intel 6300ESB をエミュレートします。
  - **ib700** – ISA iBase IB700 をエミュレートします。
- **action** - オプションの **action** 属性は、ウォッチドッグの有効期限が切れると取られるアクションについて記述します。有効な値は基礎となるハイパーバイザーに固有です。**action** 属性には、以下の値を使用することができます。

- **reset** – デフォルト設定です。ゲスト仮想マシンを強制的にリセットします。
- **shutdown** – ゲスト仮想マシンを正常にシャットダウンします (推奨されません)
- **poweroff** – ゲスト仮想マシンの電源を強制的にオフにします。
- **pause** – ゲスト仮想マシンを一時停止します。
- **none** – 何も実行しません。
- **dump** – ゲスト仮想マシンを自動的にダンプします。

'shutdown' アクションでは、ゲスト仮想マシンが ACPI シグナルに反応することが必要になります。ウォッチドッグの有効期限が切れた状態では、ゲスト仮想マシンは通常 ACPI シグナルに反応することができません。そのため、'shutdown' の使用は推奨されません。さらに、ダンプファイルを保存するディレクトリーは、ファイル `/etc/libvirt/kvm.conf` の `auto_dump_path` で設定できます。

### 24.18.20. パニックデバイスの設定

Red Hat Enterprise Linux 7 ハイパーバイザーは、**pvpanic** メカニズムを使用して Linux ゲスト仮想マシンのカーネルパニックを検出することができます。起動されると、**pvpanic** はメッセージを **libvirtd** デーモンに送信し、事前に設定された反応が開始されます。

**pvpanic** デバイスを有効にするには、以下を実行します。

- ホストマシンの `/etc/libvirt/qemu.conf` ファイルに以下の行を追加、またはそのコメントを解除します。

```
auto_dump_path = "/var/lib/libvirt/qemu/dump"
```

- **virsh edit** コマンドを実行して指定されたゲストのドメイン XML ファイルを編集し、**panic** を **devices** 親要素に追加します。

```
<devices>
  <panic>
    <address type='isa' iobase='0x505' />
  </panic>
</devices>
```

要素 **<address>** 要素はパニックのアドレスを指定します。デフォルトの **ioport** は **0x505** です。ほとんどの場合、アドレスを指定する必要はありません。

**libvirtd** がクラッシュに反応する仕方は、ドメイン XML の **<on\_crash>** 要素で決定されます。実行できるアクションは以下になります。

- **coredump-destroy**: ゲスト仮想マシンのコアダンプをキャプチャーし、ゲストをシャットダウンします。
- **coredump-restart**: ゲスト仮想マシンのコアダンプをキャプチャーし、ゲストを再起動します。
- **preserve**: ゲスト仮想マシンを停止し、追加のアクションを待機します。





## 注記

**kdump** サービスが有効にされる場合、**<on\_crash>** 設定よりも優先され、選択されている **<on\_crash>** アクションは実行されません。

**pvpanic** についての詳細は、[関連のナレッジベースアール](#)を参照してください。

### 24.18.21. メモリーバルーンデバイス

仮想メモリーバルーンデバイスは、すべての KVM ゲスト仮想マシンに追加されます。これは、**<memballoon>** 要素として表示されます。これは、適切な場合に自動的に追加されるので、特定の PCI スロットを割り当てる必要がない限り、この要素をゲスト仮想マシン XML に明示的に設定する必要はありません。**<memballoon>** デバイスを明示的に無効にする必要がある場合は、**model='none'** を使用することができます。

以下の例では、memballoon デバイスを KVM で自動的に追加しています。

```
...
<devices>
  <memballoon model='virtio' />
</devices>
...
```

図24.80 メモリーバルーンデバイス

以下は、静的 PCI スロット 2 が要求された状態で、デバイスが手動で追加されている例です。

```
...
<devices>
  <memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02'
function='0x0' />
  </memballoon>
</devices>
...
```

図24.81 手動で追加されたメモリーバルーンデバイス

必須の **model** 属性は、提供されるバルーンデバイスのタイプを指定します。仮想化プラットフォームに有効な値は以下の通りです。KVM ハイパーバイザーの場合は、**'virtio'** がデフォルト設定になります。

## 24.19. ストレージプール

すべてのストレージプールのバックエンドは同じパブリック API と XML 形式を共有しますが、それらの機能のレベルはそれぞれ異なります。ボリュームの作成を許可するものもあれば、既存ボリュームの使用のみを許可するものもあります。ボリュームのサイズや位置に制約があるものもあります。

ストレージプールドキュメントの上位要素は **<pool>** です。これには、以下の値を取る単一属性 **type** があります。**dir**, **fs**, **netfs**, **disk**, **iscsi**, **logical**, **scsi**, **mpath**, **rbd**, **sheepdog**, または **gluster**。

### 24.19.1. ストレージプールのメタデータの提供

以下の XML サンプルは、ストレージプールに追加できるメタデータタグを示しています。この例では、プールは iSCSI ストレージプールになります。

```
<pool type="iscsi">
  <name>virtimages</name>
  <uuid>3e3fce45-4f53-4fa7-bb32-11f34168b82b</uuid>
  <allocation>10000000</allocation>
  <capacity>50000000</capacity>
  <available>40000000</available>
  ...
</pool>
```

図24.82 一般的なメタデータのタグ

この例で使用される要素は、表24.27「**virt-sysprep コマンド**」で説明されています。

表24.27 virt-sysprep コマンド

要素	説明
<b>&lt;name&gt;</b>	ホスト物理マシンに固有でなければならないストレージプールの名前を提供します。これは、ストレージプールを定義する際に必須となります。
<b>&lt;uuid&gt;</b>	グローバルに一意である必要のあるストレージプールの ID を提供します。UUID を指定することはオプションですが、UUID がストレージプールの作成時に提供されない場合、UUID は自動生成されます。
<b>&lt;allocation&gt;</b>	ストレージプールの合計ストレージ割り当てを提供します。これは、メタデータのオーバーヘッドにより、すべてのストレージボリューム間での合計割り当てよりも大きくなります。この値はバイト単位で表わされます。この要素は読み取り専用であり、値を変更することはできません。
<b>&lt;capacity&gt;</b>	プールの合計ストレージ容量を提供します。基礎となるデバイスの制約により、ストレージボリュームの完全容量を使用することができない場合があります。この値はバイト単位になります。この要素は読み取り専用であり、値を変更することはできません。
<b>&lt;available&gt;</b>	ストレージプールの新規ストレージボリュームを割り当てるために利用可能な空き領域を提供します。基礎となるデバイスの制約により、空き領域全体を単一ストレージボリュームに割り当てることができない場合があります。この要素は読み取り専用であり、値を変更することはできません。

24.19.2. source 要素

<pool> 要素内に、単一 <source> 要素がある場合があります (1つのみ)。<source> の子要素はストレージプールタイプに依存します。使用できる XML サンプルには以下が含まれます。

```
...
<source>
  <host name="iscsi.example.com"/>
  <device path="demo-target"/>
  <auth type='chap' username='myname'>
    <secret type='iscsi' usage='mycluster_myname' />
  </auth>
  <vendor name="Acme"/>
  <product name="model"/>
</source>
...
```

図24.83 source 要素オプション 1

```
...
<source>
  <adapter type='fc_host' parent='scsi_host5'
wwnn='20000000c9831b4b' wwpn='10000000c9831b4b' />
</source>
...
```

図24.84 source 要素オプション 2

<source> で許可される子要素は 表24.28 「source 子要素コマンド」 で説明されています。

表24.28 source 子要素コマンド

要素	説明
<device>	ホスト物理マシンデバイスをベースとするストレージプールのソース (<pool type=> に基づく (「 <a href="#">ストレージプール</a> 」に記載) を提供します。バックエンドドライバーによって複数回繰り返される可能性があります。ブロックデバイスノードへの完全修飾パスである単一属性 <b>path</b> が含まれます。
<dir>	ディレクトリーをベースとするストレージプールのソース (<pool type='dir'>) を提供します。またはオプションでファイルシステムに基づくストレージプール内のサブディレクトリー (<pool type='gluster'>) を選択します。この要素は (<pool>) ごとに1回のみ出現する可能性があります。この要素は、バックキングディレクトリーへの完全パスである単一属性 (<path>) を受け入れます。

要素	説明
<adapter>	<p>SCSI アダプターをベースとするストレージのソース (<b>&lt;pool type='scsi'&gt;</b>) を提供します。この要素は (<b>&lt;pool&gt;</b>) ごとに 1 回のみ出現する可能性があります。属性名は SCSI アダプター名です (例: "scsi_host1"。"host1" は後方互換性について依然としてサポートされますが、推奨はされていません。属性 <b>type</b> はアダプタータイプを指定します。有効な値は <b>'fc_host'   'scsi_host'</b>。これが省略され、<b>name</b> 属性が指定される場合、これはデフォルトで <b>type='scsi_host'</b> に設定されます。後方互換性を維持するには、属性 <b>type</b> は <b>type='scsi_host'</b> アダプターにはオプションになりますが、<b>type='fc_host'</b> アダプターには必須になります。属性 <b>wwnn</b> (Word Wide Node Name) および <b>wwpn</b> (Word Wide Port Name) は、ファイバーチャネルのストレージファブリック内のデバイスを一意に特定するために <b>type='fc_host'</b> アダプターによって使用されます (デバイスは HBA または vHBA のいずれかにすることができます)。<b>wwnn</b> および <b>wwpn</b> の両方を指定する必要があります ((v)HBA の <b>wwnn/wwpn</b> を取得する方法については、「<a href="#">デバイス設定の収集</a>」を参照してください)。オプションの属性 <b>parent</b> は、<b>type='fc_host'</b> アダプターの親デバイスを指定します。</p>
<host>	<p>リモートサーバーからストレージをベースとするストレージプールのソースを提供します (<b>type='netfs'   'iscsi'   'rbd'   'sheepdog'   'gluster'</b>)。この要素は、<b>&lt;directory&gt;</b> または <b>&lt;device&gt;</b> 要素と併用する必要があります。サーバーのホスト名または IP アドレスである属性 <b>name</b> が含まれます。オプションで、プロトコル固有のポート番号の <b>port</b> 属性が含まれる場合があります。</p>
<auth>	<p><b>&lt;auth&gt;</b> 要素は、<b>type</b> 属性 (<b>pool type='iscsi'   'rbd'</b>) の設定によりソースへのアクセスに必要な認証資格情報を提供します。<b>type</b> は <b>type='chap'</b> または <b>type='ceph'</b> のいずれかにする必要があります。Ceph RBD (Rados Block Device) ネットワークソースには "ceph" を使用し、CHAP (Challenge-Handshake Authentication Protocol) iSCSI ターゲットには "iscsi" を使用します。さらに必須の属性 <b>username</b> は、認証時に使用するユーザー名や、実際のパスワードまたは他の資格情報を保持する <b>libvirt</b> 秘密オブジェクトに関連付けるために必須の属性タイプの <b>secret</b> サブ要素を指定します。<b>secret</b> 要素には秘密オブジェクトの UUID を指定した <b>uuid</b> 属性か、または秘密オブジェクトに指定されたキーに一致する <b>usage</b> 属性のいずれかが必要になります。</p>

要素	説明
<b>&lt;name&gt;</b>	名前付き要素 <b>&lt;type&gt;</b> からストレージデバイスをベースとするストレージプールを提供します: <b>(type='logical' 'rbd' 'sheepdog','gluster')</b> 。
<b>&lt;format&gt;</b>	以下の値を取ることのできるストレージプール <b>&lt;type&gt;</b> の形式についての情報を提供します: <b>type='logical' 'disk' 'fs' 'netfs')</b> 。 この値はバックエンド固有の値であることに注意してください。通常これは、ファイルシステムタイプ、ネットワークファイルシステムのタイプ、パーティションテーブルのタイプ、または LVM メタデータタイプを示すために使用されます。すべてのドライバーがこれについてのデフォルト値を持つ必要があるため、この要素はオプションになります。
<b>&lt;vendor&gt;</b>	ストレージデバイスのベンダーについてのオプション情報を提供します。これには、値がバックエンド固有である単一属性の <b>&lt;name&gt;</b> が含まれます。
<b>&lt;product&gt;</b>	ストレージデバイスの製品名についてのオプション情報を提供します。これには、値がバックエンド固有である単一属性の <b>&lt;name&gt;</b> が含まれます。

### 24.19.3. target 要素の作成

単一 **<target>** 要素は、以下のタイプの上位 **<pool>** 要素内に含まれます。

**(type='dir'|'fs'|'netfs'|'logical'|'disk'|'iscsi'|'scsi'|'mpath')** このタグは、ストレージプールのホストファイルシステムへのマッピングを記述するために使用されます。これには、以下の子要素が含まれる可能性があります。

```
<pool>
  ...
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <owner>107</owner>
      <group>107</group>
      <mode>0744</mode>
      <label>virt_image_t</label>
    </permissions>
    <timestamps>
      <atime>1341933637.273190990</atime>
      <mtime>1341930622.047245868</mtime>
      <ctime>1341930622.047245868</ctime>
    </timestamps>
    <encryption type='...'>
      ...
    </encryption>
  </target>
</pool>
```

図24.85 target 要素 XML の例

表 (表24.29 「ターゲット子要素」) では親の `<target>` 要素に有効な子要素について説明しています。

表24.29 ターゲット子要素

要素	説明
<code>&lt;path&gt;</code>	ストレージプールがローカルファイルシステムの名前空間にマップされるロケーションを提供します。ファイルシステム/ディレクトリーベースのストレージプールの場合、これはストレージボリュームが作成されるディレクトリーの名前になります。デバイススペースのストレージプールの場合、これはデバイスのノードが存在するディレクトリーの名前になります。後者の場合、 <code>/dev/</code> が論理的なオプションのようになりますが、デバイスのノードの場合オンデマンドで割り当てられるため、それらが再起動時に常に安定しているとは限りません。そのため、 <code>/dev/disk/by-{path,id,uuid,label}</code> のいずれかのように安定した場所を使用の方が好ましいと言えます。

要素	説明
<b>&lt;permissions&gt;</b>	現在のところ、これはローカルファイルシステムの名前空間にディレクトリーとしてマップされるディレクトリーまたはファイルシステムベースのストレージプールの場合にのみ有効です。これは、ストレージプールが構築される最終ディレクトリーに使用する権限についての情報を提供します。 <b>&lt;mode&gt;</b> 要素には、8 進数の権限セットについての情報が含まれます。 <b>&lt;owner&gt;</b> 要素には、数値のユーザー ID が含まれます。 <b>&lt;group&gt;</b> 要素には、数値のグループ ID が含まれます。 <b>&lt;label&gt;</b> 要素には、MAC (例: SELinux) ラベル文字列が含まれます。
<b>&lt;timestamps&gt;</b>	ストレージプールについてのタイミング情報を提供します。最大 4 つのサブ要素が表示されます。ここで、 <b>timestamps='atime' 'btime' 'ctime' 'mtime'</b> は、既知の場合はストレージボリュームの <b>access</b> 、 <b>birth</b> 、 <b>change</b> 、および <b>modification time</b> を保持します。使用される時間形式は <b>&lt;seconds&gt;</b> です。 <b>&lt;nanoseconds&gt;</b> : エポックの開始 (1970 年 1 月 1 日) 以降。nanosecond 解決が 0 またはホストオペレーティングシステムまたはファイルシステムによってその他の方法でサポートされない場合、 <b>nanoseconds</b> の部分は省略されます。これは読み取り専用の属性であり、ストレージボリュームの作成時は無視されます。
<b>&lt;encryption&gt;</b>	(ある場合) ストレージボリュームが暗号化される方法を指定します。詳細は、 <a href="#">libvirt アップストリームのページ</a> を参照してください。

#### 24.19.4. デバイスエクステントの設定

ストレージプールがその基礎となる位置または割り当てスキームについての情報を公開する場合、**<source>** 要素内の **<device>** 要素には、その利用可能なエクステントについての情報が含まれる場合があります。一部のストレージプールには、ストレージボリュームが (ディスクパーティションプールなどのように) 完全な単一制約内で割り当てられる必要があるという制約があります。したがって、エクステント情報により、アプリケーションは新規ストレージボリュームに可能な最大サイズを判別することができます。

それぞれの **<device>** 要素内で、エクステントをサポートするストレージプールの場合、ゼロまたは 1 つ以上の **<freeExtent>** 要素があります。これらの要素にはそれぞれ、**<start>** および **<end>** の 2 つの属性が含まれます。これらの属性は、バイト単位で測定されるデバイス上のエクステントの範囲を提供します。

#### 24.20. ストレージボリューム

ストレージボリュームは標準的にはファイルまたはデバイスノードのいずれかになります。1.2.0 以降、オプションの **output-only** 属性タイプは実際のタイプ (**file**、**block**、**dir**、**network**、または **netdir**) を一覧表示します。

### 24.20.1. 一般的なメタデータ

**<volume>** 要素の上部セクションには、この XML サンプルに示されるメタデータとして知られる情報が含まれます。

```
...
<volume type='file'>
  <name>sparse.img</name>
  <key>/var/lib/xen/images/sparse.img</key>
  <allocation>0</allocation>
  <capacity unit="T">1</capacity>
  ...
</volume>
```

図24.86 ストレージボリュームの一般的なメタデータ

表 (表24.30 「ボリューム子要素」) は、親 **<volume>** 属性に有効な子要素について説明します。

表24.30 ボリューム子要素

要素	説明
<b>&lt;name&gt;</b>	ストレージプールに固有のストレージボリュームの名前を提供します。これはストレージボリュームを定義をする際に必須となります。
<b>&lt;key&gt;</b>	単一ストレージボリュームを識別するストレージボリュームの識別子を提供します。場合によっては、単一ストレージボリュームを識別する 2 つの一意のキーを持たせることができます。このフィールドは、常に生成されるので、ストレージボリュームの作成時に設定することはできません。
<b>&lt;allocation&gt;</b>	ストレージボリュームのストレージ割り当てを提供します。これは、ストレージボリュームがスパースに割り当てられている場合は論理容量よりも小さくなることがあります。ストレージボリュームのメタデータのオーバーヘッドが大きい場合は論理容量よりも大きくなることがあります。この値はバイト単位です。ストレージボリュームの作成時に省略されると、ストレージボリュームは作成時に完全に割り当てられます。容量よりも小さい値に設定されると、ストレージプールには、ストレージボリュームをスパースに割り当てるかどうかを決定するオプションが付きまします。ストレージプールのタイプにより、スパースストレージボリュームの処理方法がそれぞれ異なる場合があります。たとえば、論理プールは、満杯になるとストレージボリュームの割り当てを自動的に拡張しません。ユーザーは自動設定を行うために、これを設定するか、または <b>dmeventd</b> を設定します。 <a href="#">注記</a> を参照してください。



要素	説明
<capacity>	ストレージボリュームの論理ボリュームを提供します。この値はデフォルトではバイト単位ですが、<unit> 属性については、 <a href="#">注記</a> で説明されているように <allocation> の場合と同じセマンティクス (形式) で指定できます。これはストレージボリュームの作成時には必須になります。
<source>	ストレージボリュームの基礎となるストレージ割り当てについての情報を提供します。これは、一部のストレージプールタイプには利用できないことがあります。
<target>	ローカルホスト物理マシン上のストレージボリュームの表示についての情報を提供します。

注記

必要な場合は、オプション属性 **unit** を指定して、渡される値を調整することができます。この属性は、<allocation> 要素および <capacity> 要素と共に使用できます。属性 **unit** の受け入れられる値には、以下が含まれます。

- **B** または **bytes**: バイト
- **KB**: キロバイト
- **K** または **KiB**: キビバイト
- **MB**: メガバイト
- **M** または **MiB**: メビバイト
- **GB**: ギガバイト
- **G** または **GiB**: ギビバイト
- **TB**: テラバイト
- **T** または **TiB**: テビバイト
- **PB**: ペタバイト
- **P** または **PiB**: ペビバイト
- **EB**: エクサバイト
- **E** または **EiB**: エクスビバイト

24.20.2. target 要素の設定

<target> 要素は、<volume> の上位レベルの要素に配置することができます。これは、ストレージボリューム上でのホスト物理マシンファイルシステムへのマッピングを記述するために使用されます。こ

の要素は以下の子要素を取ることができます。

```
<target>
  <path>/var/lib/virt/images/sparse.img</path>
  <format type='qcow2' />
  <permissions>
    <owner>107</owner>
    <group>107</group>
    <mode>0744</mode>
    <label>virt_image_t</label>
  </permissions>
  <compat>1.1</compat>
  <features>
    <lazy_refcounts/>
  </features>
</target>
```

図24.87 ターゲット子要素

<target> の特定の子要素は [表24.31「ターゲット子要素」](#) で説明されています。

表24.31 ターゲット子要素

要素	説明
<path>	ローカルファイルシステム上で、絶対パスとしてストレージボリュームにアクセスできる位置を提供します。これは読み取り専用の属性で、ボリュームを作成する場合に指定することができません。
<format>	プール固有のボリューム形式についての情報を提供します。ディスクベースのストレージプールの場合、パーティションタイプを提供します。ファイルシステムまたはディレクトリベースのストレージプールの場合、ファイル形式のタイプ (cow、qcow、vmdk、raw など) を提供します。ストレージボリュームを作成する場合に省略されると、ストレージプールのデフォルト形式が使用されます。実際の形式は、 <b>type</b> 属性を使用して指定されます。有効な属性の一覧については、 <a href="#">13章ストレージプール</a> の特定のストレージプールのセクションを参照してください。

要素	説明
<b>&lt;permissions&gt;</b>	ストレージボリュームの作成時に使用するデフォルトのアクセス権についての情報を提供します。現在のところ、これは、割り当てられるストレージボリュームが単純なファイルであるディレクトリーまたはファイルシステムベースのストレージプールの場合にのみ役に立ちます。ストレージボリュームがデバイスノードであるストレージプールの場合、ホットプラグスクリプトがアクセス権を判別します。これには、4 つの子要素が含まれます。<b>mode</b> 要素には、8 進数のアクセス権セットが含まれます。<b>owner</b> 要素には数字のユーザー ID が含まれます。<b>group</b> 要素には、数字のグループ ID が含まれます。<b>label</b> 要素には、MAC (例: SELinux) ラベル文字列が含まれます。
<b>&lt;compat&gt;</b>	互換性レベルを指定します。現在のところ、これは <b>type='qcow2'</b> ボリュームにのみ使用されます。現在、イメージに互換性を持たせる QEMU バージョンを指定するための有効な値は、qcow2 (バージョン 2) には <b>compat>0.10</b>、および qcow2 (バージョン 3) には <b>compat>1.1</b> です。<b>feature</b> 要素がある場合、<b>compat>1.1</b> が使用されます。省略される場合はデフォルトの qemu-img が使用されます。
<b>&lt;features&gt;</b>	形式固有の機能です。現在は、<b>format type='qcow2' /> (バージョン 3) とのみ使用されます。有効なサブ要素には、<b>lazy_refcounts</b> が含まれます。これにより、メタデータの書き込みおよびフラッシュの量が削減されます。この改善は、とくに writethrough キャッシュモードで確認されますが、クラッシュ後のイメージを修復する必要があります。また、この改善により、遅延した参照カウンターの更新が可能になります。この機能は、実装時にスピードアップできることから、qcow2 (バージョン 3) と共に使用することをお勧めします。

### 24.20.3. バックイングストア要素の設定

単一の <b>backingStore</b> 要素は、トップレベルの <b>volume</b> 要素内に含まれます。このタグは、オプションのストレージボリュームの書き込みのコピー (copy-on-write) のバックイングストアを記述するために使用されます。これには、以下の子要素が含まれます。

```

<backingStore>
  <path>/var/lib/virt/images/master.img</path>
  <format type='raw' />
  <permissions>
    <owner>107</owner>
    <group>107</group>
    <mode>0744</mode>
    <label>virt_image_t</label>
  </permissions>
</backingStore>

```

図24.88 バッキングストアの子要素

表24.32 バッキングストアの子要素

要素	説明
<b>&lt;path&gt;</b>	絶対パスとしてローカルファイルシステム上のバッキングストアにアクセスできる位置を提供します。これが省略される場合、このストレージボリュームのバッキングストアはありません。
<b>&lt;format&gt;</b>	プールに固有のバッキングストア形式についての情報を提供します。ディスクベースのストレージプールの場合、パーティションタイプを提供します。フィルシステムまたはディレクトリーベースのストレージプールの場合、ファイル形式のタイプ (cow、qcow、vmdk、raw など) を提供します。実際の形式は、 <b>&lt;type&gt;</b> 属性を使って指定されます。有効な値の一覧については、プール固有のドキュメントを参照してください。ほとんどのファイル形式には、同じ形式のバッキングストアが必要になりますが、 <b>qcow2</b> 形式は異なるバッキングストア形式を許可します。
<b>&lt;permissions&gt;</b>	バッキングファイルの権限についての情報を提供します。これには、4 つの子要素が含まれます。 <b>&lt;owner&gt;</b> 要素には、数字のユーザー ID が含まれます。 <b>&lt;group&gt;</b> 要素には、数字のグループ ID が含まれます。 <b>&lt;label&gt;</b> 要素には、MAC (例: SELinux) ラベル文字列が含まれます。

## 24.21. セキュリティーラベル

**<seclabel>** 要素は、セキュリティードライバー上の制御を可能にします。以下のような 3 つの基本的な操作モードがあります。libvirt が固有のセキュリティーラベルを自動的に生成する場合の **'dynamic'**、アプリケーション/管理者がラベルを選択する場合の **'static'**、または限定を無効にする場合の **'none'**。動的なラベル生成では、libvirt が仮想マシンに関連付けられるリソースのラベルの付け直しを常に実行します。静的なラベル割り当てでは、デフォルトでは管理者またはアプリケーションがラベルがリソースに正しく設定されていることを確認する必要がありますが、自動の再ラベル化は必要な場合は有効にできます。

複数のセキュリティードライバーが **libvirt** によって使用されている場合、複数の **seclabel** タグを使用でき、各ドライバーに1つのタグを使用し、各タグで参照されるセキュリティードライバーは属性 **model** を使用して定義できます。トップレベルのセキュリティーラベルに有効な入力 XML 設定は以下になります。

```
<seclabel type='dynamic' model='selinux' />

<seclabel type='dynamic' model='selinux'>
  <baselabel>system_u:system_r:my_svirt_t:s0</baselabel>
</seclabel>

<seclabel type='static' model='selinux' relabel='no'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>

<seclabel type='static' model='selinux' relabel='yes'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>

<seclabel type='none' />
```

図24.89 セキュリティーラベル

'**type**' 属性が入力 XML に指定されていない場合、セキュリティードライバーのデフォルト設定が使用され、これは '**none**' または '**dynamic**' のいずれかになります。**<baselabel>** が設定されていても '**type**' が設定されていない場合、タイプは '**dynamic**' であると想定されます。リソースの自動再ラベル化がアクティブな状態の実行中のゲスト仮想マシンの XML を表示する場合、追加の XML 要素の **imagelabel** が組み込まれます。これは、出力専用の要素であるため、ユーザーが指定する XML 文書では無視されます。

以下の要素は、次のような値を使って操作することができます。

- **type** - **static**、**dynamic** または **none** のいずれかになります。libvirt が固有のセキュリティーラベルを自動生成するかどうかを定めます。
- **model** - 現在アクティブ化されているセキュリティーモデルに一致する有効なセキュリティーモデル名です。
- **relabel** - **yes** または **no** になります。これは、動的なラベルの割り当てが使用される場合、常に **yes** にする必要があります。静的なラベルの割り当てでは、デフォルトで **no** に設定されます。
- **<label>** - 静的なラベルが使用されている場合、これは、仮想ドメインに割り当てる完全なセキュリティーラベルを指定するはずです。コンテンツのフォーマットは、使用されているセキュリティードライバーによって異なります。
  - **SELinux**: SELinux コンテキストです。
  - **AppArmor**: AppArmor プロファイルです。
  - **DAC**: コロンで区切られた所有者またはグループです。それらは、ユーザー/グループ名または UID/GID として定義できます。ドライバーは、最初にこれらの値を名前として解析し、先頭の正の記号は、ドライバーに対してそれらを UID または GID として解析することを強制するために使用できます。

- **<baselabel>** - 動的なラベルが使用される場合、これは、ベースセキュリティーラベルを指定するためにオプションで使用できます。コンテンツのフォーマットは、使用されているセキュリティードライバによって異なります。
- **<imagelabel>** - これは出力のみの要素であり、仮想ドメインに関連付けられたリソースで使用するセキュリティーラベルを表示します。コンテンツの形式は、使用されるセキュリティードライバによって異なります。再ラベルが有効な場合、ラベルを無効にする (ファイルが NFS またはセキュリティーラベルのない他のファイルシステムにある場合は便利) か、または代替ラベルを要求する (管理アプリケーションがドメイン間のすべてではないが一部のリソースの共有を可能にするための特殊ラベルを作成する場合に便利) ことによって、特定ソースファイル名に実行されるラベルを微調整することもできます。**seclabel** 要素が、トップレベルのドメイン割り当てではなく、特定パスに割り当てられる場合、属性の **relabel** またはサブ要素の **label** のみがサポートされます。

## 24.22. サンプル設定ファイル

i686 上の KVM ハードウェアで加速されたゲスト仮想マシン:

```
<domain type='kvm'>
  <name>demo2</name>
  <uuid>4dea24b3-1d52-d8f3-2516-782e98a23fa0</uuid>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch="i686">hvm</type>
  </os>
  <clock sync="localtime"/>
  <devices>
    <emulator>/usr/bin/kvm-kvm</emulator>
    <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/demo2.img'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
      <mac address='24:42:53:21:52:45'/>
    </interface>
    <graphics type='vnc' port='-1' keymap='de'/>
  </devices>
</domain>
```

図24.90 ドメイン XML 設定例

## パート III. 付録

## 付録A トラブルシューティング

この章では、Red Hat Enterprise Linux 7 の仮想化における一般的な問題とその解決策について説明します。

この章をお読みになると仮想化技術に関連した一般的な問題の一部の理解を深めるのに役に立ちます。トラブルシューティングの技能を高めるには Red Hat Enterprise Linux 7 で仮想化を実験してテストすることをお勧めします。

このドキュメント内で解答を得られない場合は、仮想化コミュニティのサイトから解決案が見つかるかもしれません。「[オンラインリソース](#)」を参照して、Linux 仮想化の Web サイト一覧をご覧ください。

### A.1. デバッグおよびトラブルシューティングのツール

このセクションは、システム管理者用のアプリケーション、ネットワーク構築用のユーティリティー、デバッグ用のツールなどを簡単に説明します。こうした標準的なシステム管理ツールやログを活用してトラブルシューティングに役立ててください。

- **kvm\_stat**: KVM ランタイムの統計を取得します。詳細は、「[kvm\\_stat](#)」を参照してください。
- **ftrace**: カーネルイベントを取得します。詳細は、『[Red Hat Enterprise Linux 開発者ガイド](#)』を参照してください。
- **vmstat**: 仮想メモリの統計を表示します。詳細は、**man vmstat** コマンドを使用して確認してください。
- **iostat**: I/O ロードの統計を提供します。詳細は、[Red Hat Enterprise Linux パフォーマンスチューニングガイド](#) を参照してください。
- **lsuf**: 開いているファイルを一覧表示します。詳細は、**man lsuf** コマンドを使用して確認してください。
- **systemtap**: オペレーティングシステムを監視するためのスクリプトユーティリティーです。詳細は、[Red Hat Enterprise Linux 開発者ガイド](#) を参照してください。
- **crash**: カーネルクラッシュダンプデータまたはライブシステムを分析します。詳細は、[Red Hat Enterprise Linux カーネルクラッシュダンプガイド](#) を参照してください。
- **sysrq**: コンソールが反応しない場合でもカーネルが反応するキーの組み合わせです。詳細は、[Red Hat ナレッジベース](#) を参照してください。

仮想化ネットワーク関連の問題を解決する場合に次のようなネットワーキングユーティリティーが役立ちます。

- **ip addr**、**ip route**、および **ip monitor**
- **tcpdump**: ネットワーク上のパケットトラフィックを診断します。このコマンドは、ネットワークの異常やネットワーク認証に関する問題を見つけるのに役立ちます。**wireshark** という名前のグラフィック版の **tcpdump** があります。
- **brctl**: Linux カーネル内のイーサネットブリッジ構成を検査して設定するネットワーキングツールです。以下は例になります。



```

$ brctl show
bridge-name      bridge-id          STP   enabled  interfaces
-----
virtbr0          8000.feffffff      yes   eth0

$ brctl showmacs virtbr0
port-no          mac-addr           local?  aging
timer
1                fe:ff:ff:ff:ff:    yes     0.00
2                fe:ff:ff:fe:ff:    yes     0.00
$ brctl showstp virtbr0
virtbr0
bridge-id        8000.fefffffffffff
designated-root   8000.fefffffffffff
root-port        0                  path-cost        0
max-age          20.00             bridge-max-age
20.00
hello-time       2.00              bridge-hello-time
2.00
forward-delay    0.00              bridge-forward-delay
0.00
aging-time       300.01
hello-timer      1.43              tcn-timer
0.00
topology-change-timer 0.00             gc-timer
0.02

```

他にも仮想化に関する問題の解決に役立つコマンドの一覧を以下に示します。

- **strace** は、受信したり、別のプロセスで使用されたシステムコールやイベントのトレースを実行するコマンドです。
- **vncviewer** は、サーバーまたは仮想マシン上で実行している VNC サーバーへの接続を行います。**yum install vnc** コマンドを使用して、**vncviewer** をインストールします。
- **vncserver** は、サーバーでリモートのデスクトップを開始します。リモートセッションにより **virt-manager** などのグラフィカルユーザーインターフェースを使用することができるようになります。**yum install tigervnc-server** コマンドを使用して、**vncserver** をインストールします。

上記のすべてのコマンド以外にも、仮想化ログを参照すると便利です。詳細は、「[仮想化ログ](#)」を参照してください。

## A.2. 障害復旧に備える

天候やその他の理由により機器が危険にさらされる状況に備えることができます (可能な場合)。ホスト物理マシンの以下のファイルおよびディレクトリーのバックアップを実行することを強くお勧めします。

- **/etc/libvirt** ディレクトリーのすべてのファイル。
- **/var/lib/libvirt** ディレクトリーで以下の項目をバックアップします。
  - **/var/lib/libvirt/dnsmasq** にある現在の dnsmasq DHCP リース

- `/var/lib/libvirt/network`にある実行中の仮想ネットワーク設定ファイル
- ゲストの現在の状態を保存する際に **virt-manager** で作成されるゲスト仮想マシンファイル (ある場合)。これらは `/var/lib/libvirt/qemu/save/` ディレクトリーにあります。**virsh save** コマンドを使用して仮想マシンが作成される場合は、ファイルはユーザーにより **virsh save** に指定された場所に置かれます。
- **virt-manager** によって作成される場合に `*/var/lib/libvirt/qemu/snapsho*`にあるゲスト仮想マシンのスナップショットファイル。**qemu-img create** および **virsh snapshot-create** コマンドで作成される場合、それらはユーザーがそれらのコマンドに指定する場所に置かれます。
- **virt-manager** で作成されるゲスト仮想マシンのディスクイメージ (ある場合) で、`/var/lib/libvirt/images/` ディレクトリーにあります。**virsh pool-define** コマンドを使用して仮想ストレージが作成される場合、イメージファイルはユーザーが **virsh pool-define** に指定する場所にあります。ゲストイメージファイルをバックアップする方法については、[手順A.1「障害復旧に向けたゲスト仮想マシンのディスクイメージのバックアップ作成」](#) で説明されているステップを実行してください。
- ブリッジを使用している場合、`/etc/sysconfig/network-scripts/ifcfg-<bridge_name>`にあるファイルをバックアップする必要もあります。
- オプションとして、失敗の原因を分析するために使用される、`/var/lib/libvirt/qemu/dump`にあるゲスト仮想マシンコアダンプファイルもバックアップできます。ただし、これらのファイルは一部のシステムの場合に非常に大規模になることに注意してください。

### 手順A.1 障害復旧に向けたゲスト仮想マシンのディスクイメージのバックアップ作成

この手順では、複数の異なるディスクイメージタイプをバックアップする方法について説明します。

1. ゲスト仮想マシンのディスクイメージのみをバックアップするには、`/var/lib/libvirt/images`にあるファイルをバックアップします。ゲスト仮想マシンのディスクイメージを LVM 論理ボリュームと共にバックアップするには、以下のコマンドを入力します。

```
# lvcreate --snapshot --name snap --size 8G /dev/vg0/data
```

このコマンドは、**64G** ボリュームの一部として **8G** サイズの **snap** というスナップショットボリュームを作成します。

2. これに類するコマンドを使用してスナップショットのファイルを作成します。

```
# mkdir /mnt/virt.snapshot
```

3. 以下のコマンドを使用して、作成したディレクトリーとスナップショットボリュームをマウントします。

```
# mount /dev/vg0/snap /mnt/virt.snapshot
```

4. 以下のコマンドのいずれかを使用してボリュームをバックアップします。

```
a. # tar -pzc -f /mnt/backup/virt-snapshot-MM-DD-YYYY.tgz /mnt/virt.snapshot
```

```
b. # rsync -a /mnt/virt.snapshot/ /mnt/backup/virt-snapshot.MM-DD-YYYY/
```

### A.3. ダンプファイルの作成

ゲスト仮想マシンのコアのファイルへのダンプを要求して仮想マシンのエラーを診断することができます。

#### A.3.1. virsh ダンプファイルの作成

**virsh dump** コマンドを実行すると、ゲスト仮想マシンのコアをファイルにダンプする要求が送信されるので、仮想マシンのエラーを診断することができます。このコマンドの実行には、引数の **corefilepath** で指定されるファイルとパスに適切な権限があることを手動で確認する必要があります。**virsh dump** コマンドは、**coredump** (または **crash** ユーティリティー) と同様です。

詳細は、[ゲスト仮想マシンのコアのダンプファイルの作成](#)を参照してください。

#### A.3.2. Python スクリプトを使用したコアダンプの保存

**dump-guest-memory.py python** スクリプトは、**qemu-kvm** プロセスがホストでクラッシュした後にコアダンプからゲスト仮想マシンのメモリーを抽出し、保存する **GNU Debugger (GDB)** の拡張を実装します。ホスト側の **QEMU** プロセスのクラッシュがゲストのアクションに関連する場合、**QEMU** プロセスのクラッシュ時にゲストの状態を調べると役立ちます。

**python** スクリプトは **GDB** の拡張を実装します。これは **GDB** の新規コマンドです。元の (クラッシュした) **QEMU** プロセスのコアダンプファイルを **GDB** で開いた後に、**python** スクリプトを **GDB** にロードできます。次に、新規コマンドは **GDB** プロンプトから実行できます。これにより、**QEMU** コアダンプから新規ローカルファイルにゲストメモリーのダンプが抽出されます。

**dump-guest-memory.py python** スクリプトを使用するには、以下を実行します。

1. システムに **qemu-kvm[-rhev]-debuginfo** パッケージをインストールします。
2. **GDB** を起動し、クラッシュした **/usr/libexec/qemu-kvm** バイナリー用に保存されたコアダンプファイルを開きます。デバッグ記号が自動的にロードされます。
3. **GDB** での新規コマンドのロード:

```
# source /usr/share/qemu-kvm/dump-guest-memory.py
```



#### 注記

**python** スクリプトのロード後に、組み込み **GDB help** コマンドは **dump-guest-memory** 拡張についての詳細情報を提供できます。

4. **GDB** でコマンドを実行します。以下は例になります。

```
# dump-guest-memory /home/user/extracted-vmcore X86_64
```

5. ゲストカーネルの分析用に **crash** ユーティリティーで **/home/user/extracted-vmcore** を開きます。

**crash** ユーティリティーで使用するために **QEMU** コアファイルからゲスト仮想マシンのコアを抽出する方法についての詳細は、『[How to extract ELF cores from 'gcore' generated qemu core files for use with the 'crash' utility](#)』を参照してください。

## A.4. SYSTEMTAP フライトレコーダーを使用して継続的にトレースデータを取得する

**qemu-kvm** パッケージで提供される **systemtap** **initscript** を使用し、**QEMU** トレースデータを常時取得できます。このパッケージは、実行中のすべてのゲスト仮想マシンのトレースを実行し、結果をホスト上の固定サイズバッファに保存するために **System Tap** のフライトレコーダーを使用します。古いトレースエントリは、バッファがいっぱいになると新規エントリによって上書きされます。

### 手順A.2 systemtap の設定および実行

1. パッケージをインストールします。

以下のコマンドを実行して **systemtap-initscript** パッケージをインストールします。

```
# yum install systemtap-initscript
```

2. 設定ファイルをコピーします。

以下のコマンドを実行して **systemtap** スクリプトおよび設定ファイルを **systemtap** ディレクトリにコピーします。

```
# cp /usr/share/qemu-kvm/systemtap/script.d/qemu_kvm.stp
/etc/systemtap/script.d/
# cp /usr/share/qemu-kvm/systemtap/conf.d/qemu_kvm.conf
/etc/systemtap/conf.d/
```

有効にするトレースイベントのセットが **qemu\_kvm.stp** で提供されます。この **SystemTap** スクリプトは、**/usr/share/systemtap/tapset/qemu-kvm-simpletrace.stp** で提供されるトレースイベントの追加または削除を行えるようにカスタマイズできます。

**SystemTap** のカスタマイズは **qemu\_kvm.conf** に対して行われ、フライトレコーダーのバッファサイズの制御やトレースをメモリーのみに保存するか、またはディスクにも保存するかどうかの制御が行われます。

3. サービスを起動します。

以下のコマンドを実行して **systemtap** サービスを起動します。

```
# systemctl start systemtap qemu_kvm
```

4. 起動時に実行するよう **systemtap** を有効にします。

以下のコマンドを実行して **systemtap** サービスを起動時に実行できるようにします。

```
# systemctl enable systemtap qemu_kvm
```

5. サービスが実行中であることを確認します。

以下のコマンドを実行してサービスが機能していることを確認します。

```
# systemctl status systemtap qemu_kvm
qemu_kvm is running...
```

### 手順A.3 トレースバッファの検査

#### 1. トレースバッファのダンプファイルを作成します。

以下のコマンドを実行して **trace.log** というバッファードンプファイルを作成し、これを **tmp** ディレクトリーに置きます。

```
# staprun -A qemu_kvm >/tmp/trace.log
```

ファイル名と場所を他の名前に変更することができます。

#### 2. サービスを起動します。

直前のステップでサービスを停止したので、以下のコマンドを実行してサービスを再度起動します。

```
# systemctl start systemtap qemu_kvm
```

#### 3. トレース内容を読み取り可能な形式に変換します。

トレースファイルの内容を読みやすい形式に変換するには、以下のコマンドを実行します。

```
# /usr/share/qemu-kvm/simpletrace.py --no-header /usr/share/qemu-kvm/trace-events /tmp/trace.log
```

### 注記

以下の考慮点および制限に留意してください。

- **systemtap** サービスはデフォルトで無効にされます。
- サービスが有効にされる際に小規模なパフォーマンスペナルティーが発生しますが、これは全体的にどのイベントが有効にされているかによって変わります。
- **/usr/share/doc/qemu-kvm-\*/README.systemtap** には **README** ファイルがあります。

## A.5. KVM\_STAT

**kvm\_stat** コマンドは、**kvm** カーネルモジュールからランタイム統計を取り込む **python** スクリプトです。**kvm\_stat** コマンドは、**kvm** に見えているゲストの動作を診断するために使用できます。とくに、ゲストでのパフォーマンスに関する問題を診断します。現在、報告される統計はシステム全体のものがあり、すべての実行中のゲストの動作が報告されます。このスクリプトを実行するには、**qemu-kvm-tools** パッケージをインストールする必要があります。詳細は、「[既存の Red Hat Enterprise Linux システム上への仮想化パッケージのインストール](#)」を参照してください。

**kvm\_stat** コマンドは、**kvm** カーネルモジュールがロードされており、**debugfs** がマウントされている必要があります。これらの機能のどちらも有効でない場合は、このコマンドは、**debugfs** または **kvm** モジュールを有効にするために必要なステップを出力します。たとえば、以下のようになります。

```
# kvm_stat
Please mount debugfs ('mount -t debugfs debugfs /sys/kernel/debug')
and ensure the kvm modules are loaded
```

**debugfs** をマウントしていない場合はマウントします。

```
# mount -t debugfs debugfs /sys/kernel/debug
```

## kvm\_stat 出力

**kvm\_stat** コマンドは、すべてのゲストおよびホストの統計を出力します。この出力は、コマンドが停止されるまで更新されます (**Ctrl+c** または **q** キーの使用)。画面上に表示される出力はこれとは異なる場合があります。出力要素の説明については、定義にリンクする用語をクリックしてください。

```
# kvm_stat

kvm statistics
  kvm_exit                                17724          66
  Individual exit reasons follow, refer to kvm\_exit \(NAME\) for more
  information.
    kvm_exit(CLGI)                        0              0
    kvm_exit(CPUID)                       0              0
    kvm_exit(CR0_SEL_WRITE)               0              0
    kvm_exit(EXCP_BASE)                   0              0
    kvm_exit(FERR_FREEZE)                 0              0
    kvm_exit(GDTR_READ)                   0              0
    kvm_exit(GDTR_WRITE)                  0              0
    kvm_exit(HLT)                         11             11
    kvm_exit(ICEBP)                       0              0
    kvm_exit(IDTR_READ)                   0              0
    kvm_exit(IDTR_WRITE)                  0              0
    kvm_exit(INIT)                        0              0
    kvm_exit(INTR)                        0              0
    kvm_exit(INVD)                        0              0
    kvm_exit(INVLPG)                      0              0
    kvm_exit(INVLPGA)                     0              0
    kvm_exit(IOIO)                        0              0
    kvm_exit(IRET)                        0              0
    kvm_exit(LDTR_READ)                   0              0
    kvm_exit(LDTR_WRITE)                  0              0
    kvm_exit(MONITOR)                     0              0
    kvm_exit(MSR)                         40             40
    kvm_exit(MWAIT)                       0              0
    kvm_exit(MWAIT_COND)                  0              0
    kvm_exit(NMI)                         0              0
    kvm_exit(NPF)                         0              0
    kvm_exit(PAUSE)                       0              0
    kvm_exit(POPF)                        0              0
    kvm_exit(PUSHF)                       0              0
    kvm_exit(RDPMC)                       0              0
    kvm_exit(RDTSC)                       0              0
    kvm_exit(RDTSCP)                      0              0
    kvm_exit(READ_CR0)                    0              0
    kvm_exit(READ_CR3)                    0              0
    kvm_exit(READ_CR4)                    0              0
    kvm_exit(READ_CR8)                    0              0
    kvm_exit(READ_DR0)                    0              0
    kvm_exit(READ_DR1)                    0              0
    kvm_exit(READ_DR2)                    0              0
    kvm_exit(READ_DR3)                    0              0
    kvm_exit(READ_DR4)                    0              0
    kvm_exit(READ_DR5)                    0              0
```

kvm_exit(READ_DR6)	0	0
kvm_exit(READ_DR7)	0	0
kvm_exit(RSM)	0	0
kvm_exit(SHUTDOWN)	0	0
kvm_exit(SKINIT)	0	0
kvm_exit(SMI)	0	0
kvm_exit(STGI)	0	0
kvm_exit(SWINT)	0	0
kvm_exit(TASK_SWITCH)	0	0
kvm_exit(TR_READ)	0	0
kvm_exit(TR_WRITE)	0	0
kvm_exit(VINTR)	1	1
kvm_exit(VMLoad)	0	0
kvm_exit(VMMCALL)	0	0
kvm_exit(VMRUN)	0	0
kvm_exit(VMSAVE)	0	0
kvm_exit(WBINVD)	0	0
kvm_exit(WRITE_CR0)	2	2
kvm_exit(WRITE_CR3)	0	0
kvm_exit(WRITE_CR4)	0	0
kvm_exit(WRITE_CR8)	0	0
kvm_exit(WRITE_DR0)	0	0
kvm_exit(WRITE_DR1)	0	0
kvm_exit(WRITE_DR2)	0	0
kvm_exit(WRITE_DR3)	0	0
kvm_exit(WRITE_DR4)	0	0
kvm_exit(WRITE_DR5)	0	0
kvm_exit(WRITE_DR6)	0	0
kvm_exit(WRITE_DR7)	0	0
kvm_entry	17724	66
kvm_apic	13935	51
kvm_emulate_insn	13924	51
kvm_mmio	13897	50
var1-kvm_eoi	3222	12
kvm_inj_virq	3222	12
kvm_apic_accept_irq	3222	12
kvm_pv_eoi	3184	12
kvm_fpu	376	2
kvm_cr	177	1
kvm_apic_ipi	278	1
kvm_msi_set_irq	295	0
kvm_pio	79	0
kvm_userspace_exit	52	0
kvm_set_irq	50	0
kvm_pic_set_irq	50	0
kvm_ioapic_set_irq	50	0
kvm_ack_irq	25	0
kvm_cpuid	90	0
kvm_msr	12	0

#### 変数に関する説明:

- **kvm\_ack\_irq:** 割り込みコントローラー (PIC/IOAPIC) の割り込み確認回数です。
- **kvm\_age\_page:** メモリー管理ユニット (MMU) 通知機能によるページ経過時間の反復回数です。



- **kvm\_apic**: APIC レジスターアクセスの回数です。
- **kvm\_apic\_accept\_irq**: ローカル APIC に受け入れられた割り込みの数です。
- **kvm\_apic\_ipi**: プロセッサ間の割り込みの回数です。
- **kvm\_async\_pf\_completed**: 非同期ページフォルトの完了数です。
- **kvm\_async\_pf\_doublefault**: 非同期ページフォルトの失敗回数です。
- **kvm\_async\_pf\_not\_present**: 非同期ページフォルトの初期化の回数です。
- **kvm\_async\_pf\_ready**: 非同期ページフォルトの完了数です。
- **kvm\_cpuid**: CPUID インストラクションの実行回数です。
- **kvm\_cr**: トラップされ、エミュレートされた CR (コントロールレジスター) アクセス (CR0、CR3、CR4、CR8) の数です。
- **kvm\_emulate\_insn**: エミュレートされたインストラクションの数です。
- **kvm\_entry**: エミュレートされたインストラクションの数です。
- **kvm\_eoi**: Advanced Programmable Interrupt Controller (APIC) の割り込み終了 (EOI) 通知の数です。
- **kvm\_exit**: VM-exits の数です。
- **kvm\_exit (NAME)**: プロセッサ固有の個々の出口です。詳細は、プロセッサの資料を参照してください。
- **kvm\_fpu**: KVM 浮動小数点演算ユニット (FPU) の再読み込みの回数です。
- **kvm\_hv\_hypercall**: Hyper-V hypercall の回数です。
- **kvm\_hypercall**: Hyper-V 以外の hypercall の回数です。
- **kvm\_inj\_exception**: ゲストに注入された例外の数です。
- **kvm\_inj\_virq**: ゲストに注入された割り込みの数です。
- **kvm\_invlpga**: 傍受された INVLPGA インストラクションの数です。
- **kvm\_ioapic\_set\_irq**: 仮想 IOAPIC コントローラーへの割り込みレベルの変更回数です。
- **kvm\_mmio**: エミュレートされたメモリーマップド I/O (memory-mapped I/O)(MMIO) 操作の回数です。
- **kvm\_msi\_set\_irq**: メッセージシグナル割り込み (message-signaled interrupt)(MSI) の数です。
- **kvm\_msr**: モデル固有レジスター (MSR) アクセスの回数です。
- **kvm\_nested\_intercepts**: L1 ⇒ L2 のネスト化された SVM スイッチの数です。
- **kvm\_nested\_vmrunk**: L1 ⇒ L2 のネスト化された SVM スイッチの数です。



- **kvm\_nested\_intr\_vmexit**: 割り込みウィンドウによるネスト化された VM 出口インジェクションの数です。
- **kvm\_nested\_vmexit**: ネスト化された (L2) ゲストの実行中のハイパーバイザーへの出口です。
- **kvm\_nested\_vmexit\_inject**: L2 ⇒ L1 のネスト化されたスイッチの数です。
- **kvm\_page\_fault**: ハイパーバイザーによって処理されるページフォルトの数です。
- **kvm\_pic\_set\_irq**: 仮想プログラム割り込みコントローラー (PIC) への割り込みレベルの変更回数です。
- **kvm\_pio**: エミュレートされたプログラム I/O (programmed I/O)(PIO) 操作の回数です。
- **kvm\_pv\_eoi**: 準仮想 EOI (end of input) イベントの回数です。
- **kvm\_set\_irq**: 汎用 IRQ コントローラーレベルの割り込みレベルの変更回数です (カウント PIC、IOAPIC および MSI)。
- **kvm\_skinit**: SVM SKINIT 終了の回数です。
- **kvm\_track\_tsc**: タイムスタンプカウンタ (TSC) の書き込み回数です。
- **kvm\_try\_async\_get\_page**: 非同期ページフォルトの試行回数です。
- **kvm\_update\_master\_clock**: pvclock マスターロックの更新回数です。
- **kvm\_userspace\_exit**: ユーザー空間への出口の数です。
- **kvm\_write\_tsc\_offset**: TSC オフセットの書き込み回数です。
- **vcpu\_match\_mmio**: SPTE キャッシュされたメモリーマップド I/O (MMIO) ヒット数です。

**kvm\_stat** コマンドからの出力情報は、`/sys/kernel/debug/tracing/events/kvm/` ディレクトリー内に配置された擬似ファイルとして KVM ハイパーバイザーによってエクスポートされます。

## A.6. シリアルコンソールでのトラブルシューティング

Linux カーネルは、情報をシリアルポートに出力できます。これは、カーネルパニック、およびビデオデバイスまたはヘッドレスのサーバーでのハードウェア問題のデバッグに役に立ちます。このセクション内のサブセクションでは、KVM ハイパーバイザーを使用してホスト物理マシンのシリアルコンソールの出力を設定する方法を説明します。

このセクションでは、完全仮想化ゲストのシリアルコンソールの出力を有効にする方法を説明しています。

完全仮想化ゲストのシリアルコンソール出力は、**virsh console** コマンドを使用して表示できます。

完全仮想化ゲストのシリアルコンソールにはいくつかの制限があるため注意してください。現時点での制限は以下のようになります。

- 出力データが消えたり、出力が乱れることがあります。

シリアルポートは、Linux では **ttyS0** と呼ばれています。

仮想シリアルポートに情報を出力させる場合は、仮想化オペレーティングシステムを設定する必要があります。

完全仮想化 Linux ゲストからドメインにカーネル情報を出力するには、`/etc/default/grub` ファイルを修正します。**kernel** 行に以下を追記します:**`console=tty0 console=ttyS0,115200`**

```
title Red Hat Enterprise Linux Server (2.6.32-36.x86-64)
  root (hd0,0)
  kernel /vmlinuz-2.6.32-36.x86-64 ro root=/dev/volgroup00/logvol100 \
  console=tty0 console=ttyS0,115200
  initrd /initrd-2.6.32-36.x86-64.img
```

ゲスト内で以下のコマンドを実行します。

```
# grub2-mkconfig -o /etc/grub2.cfg
```



### 注記

UEFI ベースのホストを使用している場合、ターゲットファイルは `/etc/grub2-efi.cfg` であることに注意してください。

ゲストを再起動します。

ホスト上で、以下のコマンドを使用してシリアルコンソールにアクセスします。ここで、**GUEST** はゲスト仮想マシンの名前になります。

```
# virsh console GUEST
```

**virt-manager** を使って仮想テキストコンソールを表示することもできます。ゲストのコンソールウィンドウで、**表示** メニューから **テキストコンソール** で **Serial 1** を選択します。

## A.7. 仮想化ログ

以下の方法は、ハイパーバイザーおよびゲストでのイベントについてのログに記録されたデータにアクセスするために使用できます。これは、システム上で仮想化の問題を解決するのに役立ちます。

- 各ゲストにはログがあり、`/var/log/libvirt/qemu/` ディレクトリーに保存されます。ログには **GuestName.log** という名前が付けられ、サイズの上限に達すると定期的に圧縮されます。
- systemd Journal** で **libvirt** イベントを表示するには、以下のコマンドを使用します。

```
# journalctl _SYSTEMD_UNIT=libvirtd.service
```

- auvirt** コマンドは、ハイパーバイザーのゲストに関連する監査結果を表示します。表示されるデータの範囲は、特定のゲスト、時間枠、および情報の形式で絞り込むことができます。たとえば、以下のコマンドは、現在の日付で **testguest** 仮想マシンのイベントの要約を提供します。

```
# auvirt --start today --vm testguest --summary
Range of time for report:      Mon Sep  4 16:44 - Mon Sep  4 17:04
Number of guest starts:       2
Number of guest stops:        1
```

```

Number of resource assignments: 14
Number of related AVCs:        0
Number of related anomalies:    0
Number of host shutdowns:      0
Number of failed operations:    0

```

また、**auvirt** 情報を **systemd Journal** に自動的に組み込むよう **auvirt** 情報を設定することもできます。これを実行するには、**/etc/libvirt/libvirtd.conf** ファイルを編集し、**audit\_logging** パラメーターの値を **1** に設定します。

詳細は、**auvirt man** ページを参照してください。

- 仮想マシンマネージャーでエラーが出た場合は、**\$HOME/.virt-manager/** ディレクトリーの **virt-manager.log** ファイルで生成されたデータを確認することができます。
- ハイパーバイザーシステムの監査ログについては、**/var/log/audit/audit.log** ファイルを参照してください。
- ゲストオペレーティングシステムによっては、各種のシステムログファイルがゲストに保存される場合もあります。

Red Hat Enterprise Linux でのロギングについての詳細は、[システム管理者のガイド](#)を参照してください。

## A.8. ループデバイスエラー

ファイルベースのゲストイメージが使用される場合は、設定済みのループデバイスの数を増やす必要が生じる場合があります。デフォルトの設定では、最大**8**つのアクティブループデバイスを使用できます。**9**つ以上のファイルベースのゲストまたはループデバイスが必要な場合は、設定済みのループデバイスの数を **/etc/modprobe.d/** ディレクトリーで調整できます。以下の行を追加します。

```
options loop max_loop=64
```

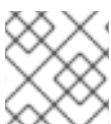
この例では **64** を使用していますが、別の値を最大ループ値に指定しても構いません。また、ループデバイスでバックアップされたゲストをシステムに実装する必要があるかもしれません。完全仮想化システムにループデバイスでバックアップされたゲストを使用する場合は、**phy: device** コマンドまたは **file: file** コマンドを使用します。

## A.9. ライブマイグレーションに関するエラー

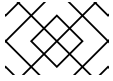
ゲストによるメモリーの変更速度が速すぎて、ライブマイグレーションプロセスで転送を何度も実行し、完了 (収束) に失敗する場合があります。

現在のライブマイグレーションの実装では、デフォルトの移行時間が **30** ミリ秒に設定されています。この値は、残りを転送するために移行の終わりにゲストをいったん停止する時間を定義します。値が高くなるほどライブマイグレーションが収束する確率が高くなります。

## A.10. BIOS での INTEL VT-X と AMD-V の仮想化ハードウェア拡張の有効化



注記



この点についてさらにお知りになりたい場合は、[Red Hat Virtualization \(RH318\)](#) トレーニングコースをご利用いただけます。

このセクションでは、ハードウェア仮想化拡張を特定し、拡張が無効になっている場合は BIOS でその拡張を有効にする方法について説明しています。

Intel VT-x 拡張は BIOS で無効にされている可能性があります。一部のラップトップベンダーは、デフォルトで CPU の Intel VT-x 拡張を無効にしています。

AMD-V の場合、仮想化拡張を BIOS で無効にすることはできません。

無効になっている仮想化拡張を有効にする方法については、次のセクションを参照してください。

仮想化拡張が BIOS で有効になっていることを確認します。Intel VT や AMD-V の BIOS 設定は、通常 **Chipset** または **Processor** のメニューで確認できます。メニュー名は本ガイドと異なる場合があります。仮想化拡張の設定は **Security Settings** で確認できるか、または別のメニュー名になっている場合もあります。

#### 手順A.4 BIOS での仮想化拡張の有効化

1. コンピューターを再起動して、システムの BIOS メニューを開きます。システムにより使用するキーは異なる場合がありますが、通常、**delete** キー、**F1** キー、**Alt** と **F4** キーの組み合わせなどのいずれかを押すと BIOS メニューを開くことができます。

#### 2. BIOS での仮想化拡張の有効化



##### 注記

以下に示す手順の多くは、ご使用のマザーボードやプロセッサの種類、チップセット、OEM などにより異なる場合があります。システム設定に関する正確な情報についてはそのシステムに付随するドキュメントを参照してください。

- a. **Processor** サブメニューを開きます。プロセッサの設定メニューは **Chipset** や **Advanced CPU Configuration**、**Northbridge** など非表示されている場合があります。
  - b. **Intel Virtualization Technology** (別名 Intel VT-x) を有効にします。**AMD-V** の拡張は BIOS では無効にできず、すでに有効にされているはずです。仮想化拡張機能には **Virtualization Extensions** や **Vanderpool** といったラベル名が付けられています。また、OEM とシステム BIOS によってはこれ以外の各種の名前が付けられていることもあります。
  - c. Intel VT-d または AMD IOMMU というオプションがあればそれを有効にします。Intel VT-d と AMD IOMMU は PCI デバイス割り当てに使用されます。
  - d. **Save & Exit** を選択します。
3. マシンを再起動します。
  4. マシンが起動したら、**grep -E "vmx|svm" /proc/cpuinfo** を実行します。 **--color** の指定はオプションですが、検索した単語を強調表示させたい場合は便利です。コマンドからの出力があれば、仮想化拡張が有効になっているということになります。出力がない場合は、システムに仮想化拡張がないか、または BIOS 設定が正しく有効にされていない可能性があります。

## A.11. RED HAT ENTERPRISE LINUX 7 ホスト上での RED HAT ENTERPRISE LINUX 6 ゲストのシャットダウン

**最小限のインストール (Minimal installation)** オプションを指定して Red Hat Enterprise Linux 6 ゲスト仮想マシンをインストールしても、**acpid (acpi daemon)** パッケージはインストールされません。このパッケージは **systemd** に引き継がれたので Red Hat Enterprise Linux 7 では不要です。ただし、Red Hat Enterprise Linux 7 ホスト上で実行される Red Hat Enterprise Linux 6 ゲスト仮想マシンにはこれが依然として必要になります。

**acpid** パッケージがないと、**virsh shutdown** コマンドを実行しても Red Hat Enterprise Linux 6 ゲスト仮想マシンがシャットダウンしません。**virsh shutdown** コマンドはゲスト仮想マシンを正常にシャットダウンするように設計されています。

**virsh shutdown** コマンドの使用はシステム管理上、より安全かつ簡単な方法となります。**virsh shutdown** コマンドで正常にシャットダウンできないと、システム管理者は手動でゲスト仮想マシンにログインするか、または **Ctrl-Alt-Del** のキー組み合わせを各ゲスト仮想マシンに送信しなければならなくなります。



### 注記

他の仮想化オペレーティングシステムもこの問題の影響を受ける場合があります。 **virsh shutdown** コマンドが正しく動作するには、ゲスト仮想マシンのオペレーティングシステムが **ACPI** シャットダウン要求を処理できるよう設定されている必要があります。 **ACPI** シャットダウン要求の受け入れを可能にするには、多くのオペレーティングシステムの場合、ゲスト仮想マシンのオペレーティングシステムで追加の設定が必要になります。

### 手順A.5 Red Hat Enterprise Linux 6 ゲストの回避策

1. **acpid** パッケージをインストールします。

**acpid** サービスが **ACPI** 要求をリッスンし、これを処理します。

ゲスト仮想マシンにログインし、ゲスト仮想マシンに **acpid** パッケージをインストールします。

```
# yum install acpid
```

2. ゲストで **acpid** サービスを有効にします。

**acpid** サービスがゲスト仮想マシンの起動シーケンスで起動され、サービスを起動するように設定します。

```
# chkconfig acpid on
# service acpid start
```

3. ゲストドメイン XML を準備します。

以下の要素を組み込むようにドメイン XML ファイルを編集します。**virtio** シリアルポートを **org.qemu.guest\_agent.0** に置き換え、表示されているものの代わりにゲストの名前を使用します。この例では、ゲストは **guest1** です。ファイルは必ず保存してください。

```
<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/guest1.agent' />
  <target type='virtio' name='org.qemu.guest_agent.0' />
</channel>
```

### 図A.1 ゲスト XML の置き換え

#### 4. QEMU ゲストエージェントをインストールします。

『[Red Hat Enterprise Linux 6 仮想化管理ガイド](#)』で説明されているように QEMU ゲストエージェント (QEMU-GA) をインストールし、サービスを起動します。

#### 5. ゲストをシャットダウンします。

- a. 既知のゲスト仮想マシンを一覧表示し、シャットダウンする必要がある仮想マシンの名前を検索できます。

```
# virsh list --all
   Id Name                               State
-----
  14 guest1                             running
```

- b. ゲスト仮想マシンをシャットダウンします。

```
# virsh shutdown guest1

guest virtual machine guest1 is being shutdown
```

- c. ゲスト仮想マシンがシャットダウンするまで数秒間待機します。それがシャットダウンされていることを確認します。

```
# virsh list --all
   Id Name                               State
-----
  14 guest1                             shut off
```

- d. 編集した XML ファイルを使用して、**guest1** という名前のゲスト仮想マシンを起動します。

```
# virsh start guest1
```

- e. **guest1** ゲスト仮想マシンで **acpi** をシャットダウンします。

```
# virsh shutdown --mode acpi guest1
```

- f. すべてのゲスト仮想マシンを再び一覧表示します。**guest1** が一覧に表示され、それが停止していることが表示されるはずです。

```
# virsh list --all
   Id Name                               State
-----
  14 guest1                             shut off
```

- g. 編集した XML ファイルを使用して、**guest1** という名前のゲスト仮想マシンを起動します。

```
# virsh start guest1
```

- h. **guest1** ゲスト仮想マシンのゲストエージェントをシャットダウンします。

```
# virsh shutdown --mode agent guest1
```

- i. ゲスト仮想マシンを一覧表示します。**guest1** が一覧に表示され、それが停止していることを確認できるはずです。

```
# virsh list --all
   Id Name                               State
-----
  guest1                               shut off
```

ゲスト仮想マシンは、連続するシャットダウンでは上記の回避策を使用せずに **virsh shutdown** コマンドを使ってシャットダウンします。

上記の方法以外にも、**libvirt-guests** サービスを停止してゲストを自動的にシャットダウンできます。この方法についてさらに詳しくは、「[正常なシャットダウンに向けたオプションの方法](#)」を参照してください。

## A.12. 正常なシャットダウンに向けたオプションの方法

**libvirt-guests** サービスには、ゲストの適切なシャットダウンを確実に実行するために使用できるパラメーター設定が含まれます。これは **libvirt** インストールの一部をなすパッケージであり、デフォルトでインストールされます。このサービスは、ホストがシャットダウンされる際にゲストをディスクに自動的に保存し、ホストの再起動時にシャットダウン前の状態にゲストを復元します。デフォルトでは、この設定はゲストをサスペンドするように設定されます。ゲストを正常にシャットダウンする必要がある場合、**libvirt-guests** 設定ファイルのパラメーターのいずれかを変更する必要があります。

### 手順A.6 ゲストの正常なシャットダウンに向けた **libvirt-guests** サービスパラメーターの変更

以下で説明される手順により、ホスト物理マシンが停止しているか、電源がオフになっているか、または再起動が必要な場合に、ゲスト仮想マシンを正常にシャットダウンすることができます。

#### 1. 設定ファイルを開きます。

設定ファイルは **/etc/sysconfig/libvirt-guests** にあります。ファイルを編集し、コメントマーク (#) を削除し、**ON\_SHUTDOWN=suspend** を **ON\_SHUTDOWN=shutdown** に変更します。変更は必ず保存してください。

```
$ vi /etc/sysconfig/libvirt-guests

# URIs to check for running guests
# example: URIS='default xen:/// vbox+tcp://host/system lxc:///'
```

**1**

```
#URIS=default

# action taken on host boot
# - start    all guests which were running on shutdown are started on
boot
```



```
#           regardless on their autostart settings
# - ignore  libvirt-guests init script won't start any guest on
boot, however,
#           guests marked as autostart will still be automatically
started by
#           libvirtd

2
#ON_BOOT=start

# Number of seconds to wait between each guest start. Set to 0 to

3
allow
# parallel startup.
#START_DELAY=0

# action taken on host shutdown
# - suspend  all running guests are suspended using virsh
managesave
# - shutdown all running guests are asked to shutdown. Please be
careful with
#           this settings since there is no way to distinguish
between a
#           guest which is stuck or ignores shutdown requests and
a guest
#           which just needs a long time to shutdown. When
setting
#           ON_SHUTDOWN=shutdown, you must also set
SHUTDOWN_TIMEOUT to a
#           value suitable for your guests.

4
ON_SHUTDOWN=shutdown

# If set to non-zero, shutdown will suspend guests concurrently.
Number of
# guests on shutdown at any time will not exceed number set in this

5
variable.
#PARALLEL_SHUTDOWN=0

# Number of seconds we're willing to wait for a guest to shut down.
If parallel
# shutdown is enabled, this timeout applies as a timeout for
shutting down all
# guests on a single URI defined in the variable URIS. If this is 0,
then there
# is no time out (use with caution, as guests might not respond to a
shutdown
# request). The default value is 300 seconds (5 minutes).

6
#SHUTDOWN_TIMEOUT=300
```



```
# If non-zero, try to bypass the file system cache when saving and
# restoring guests, even though this may give slower operation for
# some file systems.
```

7

```
#BYPASS_CACHE=0
```

- ❶ **URIS:** 実行中のゲストの指定された接続をチェックします。**Default** 設定は、明示的な URI が設定されていない場合に **virsh** と同じように機能します。さらに、URI は **/etc/libvirt/libvirt.conf** から明示的に設定できます。libvirt 設定ファイルのデフォルト設定を使用する場合、プローブは使用されないことに注意してください。
- ❷ **ON\_BOOT:** ホストの起動時にゲストに対して実行されるか、またはゲスト上で実行されるアクションを指定します。 **start** オプションは、**autostart** の設定にかかわらずシャットダウンの前に実行されているすべてのゲストを起動します。 **ignore** オプションは、起動時に正式に実行されているゲストを起動しませんが、**autostart** というマークが付けられたゲストは **libvirtd** によって自動的に起動されます。
- ❸ **START\_DELAY:** ゲストが起動する間の遅延期間を設定します。この期間は秒単位で設定されます。遅延がないようにし、かつすべてのゲストを同時に起動させるには時間設定を **0** にします。
- ❹ **ON\_SHUTDOWN:** ホストがシャットダウンする際に取られるアクションを指定します。設定できるオプションには、**virsh managedsave** を使用して実行中のすべてのゲストを一時停止する **suspend** と、実行中のすべてのゲストをシャットダウンする **shutdown** が含まれます。**shutdown** オプションの場合、停止した状態のゲストか、またはシャットダウン要求を無視するゲストか、またはシャットダウンにより長い時間がかかっているだけのゲストを区別する方法がないため、注意して使用してください。**ON\_SHUTDOWN=shutdown** を設定する際には、**SHUTDOWN\_TIMEOUT** をゲストに適した値に設定する必要もあります。
- ❺ **PARALLEL\_SHUTDOWN:** 任意に実行されるシャットダウン時のゲスト数がこの変数で設定される数を超えないようにし、かつゲストが同時にサスペンドするように指定します。**0** に設定される場合、ゲストは同時にシャットダウンしません。
- ❻ ゲストがシャットダウンするのを待機する秒数です。**SHUTDOWN\_TIMEOUT** が有効な場合、このタイムアウトが、変数 **URIS** で定義される単一 URI 上のすべてのゲストをシャットダウンするタイムアウトとして適用されます。**SHUTDOWN\_TIMEOUT** が **0** に設定される場合、タイムアウトはありません (ゲストがシャットダウン要求に応答しない可能性があるため注意して使用してください)。デフォルトの値は **300** 秒 (5 分) です。
- ❼ **BYPASS\_CACHE** には無効にするためには **0**、有効にするには **1** とする **2** つの値を使用することができます。有効にされている場合、ゲストが復元するとファイルシステムのキャッシュをバイパスします。この設定はパフォーマンスに影響を与え、一部のファイルシステムの操作の速度を低下させる可能性があることに注意してください。

## 2. libvirt-guests サービスを起動します。

サービスをまだ起動していない場合は、**libvirt-guests** サービスを起動します。サービスの再起動により、実行中のすべてのゲスト仮想マシンがシャットダウンする可能性があるため実行しないでください。

## A.13. KVM ネットワークのパフォーマンス

デフォルトでは仮想 **Realtek 8139 (rtl8139)** NIC (ネットワークインターフェースコントローラー) が KVM 仮想マシンに割り当てられます。

**rtl8139** 仮想化 NIC はほとんどの環境で正常に機能しますが、**10 Gigabit Ethernet** ネットワークなどの一部のネットワークでは、このデバイスがパフォーマンスの低下の影響を受ける場合があります。

パフォーマンスを改善するには、準仮想化ネットワークドライバーに切り替えます。



### 注記

エミュレートするドライバーの選択肢として、仮想化した **Intel PRO/1000 (e1000)** ドライバーもサポートされています。**e1000** ドライバーを使用する場合は、以下の手順にある **virtio** の部分を **e1000** に置き換えます。最善のパフォーマンスを得るには、**virtio** ドライバーの使用を推奨します。

## 手順A.7 virtio ドライバーへの切り替え

1. ゲストオペレーティングシステムをシャットダウンします。
2. **virsh** コマンドを使用してゲストの設定ファイルを編集します (**GUEST** はゲスト名)。

```
# virsh edit GUEST
```

**virsh edit** コマンドは **\$EDITOR** シェル変数を使用し、使用するエディターを判別します。

3. 設定のネットワークインターフェースのセクションを見つけます。以下にその抜粋した部分を示します。

```
<interface type='network'>
  [output truncated]
  <model type='rtl8139' />
</interface>
```

4. **model** 要素のタイプ属性を '**rtl8139**' から '**virtio**' に変更します。これによりドライバーが **rtl8139** から **virtio** に変更されます。

```
<interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

5. 変更を保存して、テキストエディターを終了します。
6. ゲストのオペレーティングシステムを再起動します。

## 他のネットワークドライバーを使用した新しいゲストの作成

別のネットワークドライバーで新規のゲストを作成することもできます。ネットワーク接続によるゲストのインストールが困難な場合に、この手段が必要になるかもしれません。この方法では、テンプレートとして使用するゲストが少なくとも1つすでに作成されていなければなりません (CD か DVD などからインストール)。

1. 既存のゲスト (この例では、**Guest1** という名前のゲスト) から XML テンプレートを作成します。

```
# virsh dumpxml Guest1 > /tmp/guest-template.xml
```

2. XML ファイルをコピーし、仮想マシンの名前、UUID、ディスクイメージ、MAC アドレス、その他固有のパラメーターなどの固有のフィールドを編集して更新を行います。UUID と MAC アドレスの行は削除して構いません。virsh により UUID と MAC アドレスが生成されます。

```
# cp /tmp/guest-template.xml /tmp/new-guest.xml
# vi /tmp/new-guest.xml
```

ネットワークインターフェースのセクションにモデルの行を追加します。

```
<interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

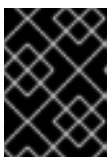
3. 新規の仮想マシンを作成します。

```
# virsh define /tmp/new-guest.xml
# virsh start new-guest
```

## A.14. LIBVIRT による外部スナップショットの作成方法

KVM ゲスト用のスナップショットには 2 つのクラスがあります。

- **内部スナップショット** は **qcow2** ファイル内に完全に格納されており、**libvirt** に完全にサポートされています。これにより、スナップショットの作成、削除および元に戻す操作が可能になります。これは、とりわけオプションが指定されていない場合にスナップショットを作成する際の、**libvirt** によって使用されるデフォルトの設定になります。このファイルのタイプではスナップショットを作成する場合に他のタイプにかかる時間よりも少し多くの時間がかかり、**qcow2** ディスクが必要になるという短所があります。



### 重要

内部スナップショットはアクティブに開発されていないため、Red Hat はそれらの使用をお勧めしません。

- 他方、**外部スナップショット** は元のディスクイメージのすべてのタイプで機能し、ゲストのダウンタイムなしに取得でき、より高い安定性および信頼性があります。そのため、外部スナップショットは KVM ゲスト仮想マシンでの使用に推奨されます。ただし、現在のところ、外部スナップショットは Red Hat Enterprise Linux 7 では完全に実装されておらず、**virt-manager** の使用時に利用できません。

外部スナップショットを作成するには、**snapshot-create-as** を **--diskspec vda,snapshot=external** オプションと共に使用するか、またはスナップショット XML ファイルで以下の **disk** 行を使用します。

```
<disk name='vda' snapshot='external'>
  <source file='/path/to,new' />
</disk>
```

現在のところ **libvirt** は外部スナップショットを作成できるものの、それ以外の動作を実行できないため、外部スナップショットには一方向の操作のみを実行できます。回避策については、[libvirt アップストリームページ](#) で説明されています。

## A.15. 日本語キーボードのゲストコンソールで欠如している文字

Red Hat Enterprise Linux 7 のホスト上で、日本語キーボードをマシンにローカルで接続すると、下線 ( \_ 記号) などの入力した文字がゲストコンソールでは正しく表示されないことがあります。これは、必要なキーマップがデフォルトでは正しく設定されていないために生じます。

Red Hat Enterprise Linux 6 および Red Hat Enterprise Linux 7 ゲストの場合、関連キーを押しても通常はエラーメッセージが生成されません。ただし、Red Hat Enterprise Linux 4 および Red Hat Enterprise Linux 5 ゲストは以下のようなエラーを表示することがあります。

```
atkdb.c: Unknown key pressed (translated set 2, code 0x0 on
isa0060/serio0).
atkdb.c: Use 'setkeycodes 00 <keycode>' to make it known.
```

**virt-manager** でこの問題を修復するには、以下のステップを実行します。

- **virt-manager** で影響を受けているゲストを開きます。
- 表示 → 詳細 をクリックします。
- 一覧内の **ディスプレイ VNC** を選択します。
- キー配列 プルダウンメニューの **自由** を **ja** に変更します。
- **適用** ボタンをクリックします。

または、ターゲットゲストで **virsh edit** コマンドを使ってこの問題を修復することもできます。

- **virsh edit guestname** を実行します。
- **keymap='ja'** の属性を **<graphics>** タグに追加します。

```
<graphics type='vnc' port='-1' autoport='yes' keymap='ja' />
```

## A.16. ゲスト仮想マシンのシャットダウンの失敗

通常 **virsh shutdown** コマンドを実行すると、電源ボタンの **ACPI** イベントが送信され、物理マシン上で誰かが電源ボタンを押す場合と同様のアクションがコピーされます。すべての物理マシン内で、このイベントを処理するのは **OS** になります。これまではオペレーティングシステムのサイレントインストールが実行されていましたが、現在の最も有効なアクションは、何を実行すべきかを尋ねるダイアログを表示させることです。一部のオペレーティングシステムは、とりわけいずれのユーザーもログインしていない場合にこのイベントを完全に無視します。このようなオペレーティングシステムがゲスト仮想マシンにインストールされていると、**virsh shutdown** を実行しても機能しません (このコマンドが無視されるか、ダイアログが仮想ディスプレイに表示されるかのいずれかです)。ただし、**qemu-guest-agent** チャンネルがゲスト仮想マシンに追加され、このエージェントがゲスト仮想マシンの **OS** 内で実行されている場合、**virsh shutdown** コマンドは、エージェントに対し、**ACPI** イベントを送信する代わりにゲスト **OS** をシャットダウンするように指示します。エージェントはゲスト仮想マシン **OS** の内部からシャットダウンを要求し、すべてのことが予想通りに機能します。

### 手順A.8 ゲスト仮想マシンのゲストエージェントチャンネルの設定

1. ゲスト仮想マシンを停止します。
2. ゲスト仮想マシン用のドメイン XML を開き、次にスニペットを追加します。

```
<channel type='unix'>
  <source mode='bind' />
  <target type='virtio' name='org.qemu.guest_agent.0' />
</channel>
```

#### 図A.2 ゲストエージェントチャネルの設定

3. **virsh start [domain]** を実行して、ゲスト仮想マシンを起動します。
4. ゲストの仮想マシンに **qemu-guest-agent** をインストール (**yum install qemu-guest-agent**) し、これをサービス (**qemu-guest-agent.service**) として毎回の起動時に自動的に実行させます。

## A.17. ゲスト仮想マシンの SMART ディスクモニタリングの無効化

仮想ディスクおよび物理的なストレージデバイスはホスト物理マシンで管理されるため、SMART ディスクモニタリングは安全に無効にすることができます。

```
# service smartd stop
# systemctl --del smartd
```

## A.18. LIBGUESTFS のトラブルシューティング

**libguestfs** が動作しているかどうかを確認できるテストツールがあります。**libguestfs** をインストールしてから次のコマンドを実行し (root 権限は不要)、通常の動作をテストします。

```
$ libguestfs-test-tool
```

このツールにより、**libguestfs** の動作テスト用のテキストが大量に出力されます。テストにパスすると、出力の最後の方で次のテキストが表示されます。

```
===== TEST FINISHED OK =====
```

## A.19. SR-IOV のトラブルシューティング

このセクションでは、SR-IOV に影響を与える可能性のある問題の解決方法を説明します。追加のヘルプが必要な場合は、「[SR-IOV 仮想機能のプールからの PCI デバイス割り当ての設定](#)」を参照してください。

### ゲストの起動時のエラー

設定済みの仮想マシンの起動時に、以下のようなエラーが発生する場合があります。

```
# virsh start test
error: Failed to start domain test
error: Requested operation is not valid: PCI device 0000:03:10.1 is in
use by domain rhel7
```

このエラーは多くの場合、すでに別のゲストか、またはホスト自体に割り当てられているデバイスによって発生します。

## ゲストの移行、保存またはダンプ時のエラー

仮想マシンの移行およびダンプ時には、以下のようなエラーが発生する場合があります。

```
# virsh dump rhel7/tmp/rhel7.dump

error: Failed to core dump domain rhel7 to /tmp/rhel7.dump
error: internal error: unable to execute QEMU command 'migrate': State
blocked by non-migratable device '0000:00:03.0/vfio-pci'
```

デバイスの割り当てでは仮想マシンが起動した特定のホスト上のハードウェアを使用するため、デバイス割り当ての使用時にはゲストの移行および保存はサポートされません。現在、同様の制限はゲストのコアダンプにも適用されます。これは将来変更される可能性があります。**QEMU** は現在、**-memory-only** オプションが指定されていない限り、**PCI** デバイスが接続されているゲスト仮想マシン上の移行、保存およびダンプ操作をサポートしていないことに注意してください。現在のところ、**USB** デバイスの場合に関してのみこれらの操作がサポートされています。現在、今後の改善に向けた作業が進行中です。

## A.20. 一般的な LIBVIRT エラーおよびトラブルシューティング

この付録では、**libvirt** 関連の一般的な問題とエラーおよびそれらの対処法について説明します。

以下の表でエラーを特定し、**解決法**にある対応するリンク先で詳細なトラブルシューティング情報を参照してください。

表A.1 一般的な libvirt エラー

エラー	問題の概要	解決法
<b>libvirtd failed to start</b>	<b>libvirt</b> デーモンの起動に失敗するが、 <b>/var/log/messages</b> にはこのエラーに関する情報が無い。	<a href="#">「libvirtd の起動に失敗」</a>
<b>Cannot read CA certificate</b>	URI がハイパーバイザー接続に失敗する際に発生するエラーの1つです。	<a href="#">「URI のハイパーバイザー接続に失敗する」</a>
<b>Other connectivity errors</b>	URI がハイパーバイザー接続に失敗する際に発生するその他のエラーです。	<a href="#">「URI のハイパーバイザー接続に失敗する」</a>
<b>Failed to create domain from vm.xml error: monitor socket did not show up.: Connection refused</b>	ゲスト仮想マシン (またはドメイン) の起動に失敗し、このエラーまたは同様のエラーを返す。	<a href="#">「ゲストの起動に失敗しエラーが発生: monitor socket did not show up」</a>

エラー	問題の概要	解決法
<b>Internal error cannot find character device (null)</b>	このエラーは、ゲストのコンソールに接続しようとする際に発生します。シリアルコンソールがゲスト仮想マシン用に設定されていないことを報告します。	「 <b>internal error cannot find character device (null)</b> 」
<b>No boot device</b>	既存のディスクイメージからゲスト仮想マシンを構築した後、ゲストの起動は停止するが、 <b>QEMU</b> コマンドを直接使うとゲストは正常に起動できる。	「ゲスト仮想マシンの起動が停止してエラーが発生: <b>No boot device</b> 」
<b>The virtual network "default" has not been started</b>	<b>default</b> ネットワーク (またはローカルで作成された別のネットワーク) を開始できないと、そのネットワークを接続に使用するように設定されたすべての仮想マシンも起動に失敗する。	「 <b>Virtual network default has not been started</b> 」
<b>PXE boot (or DHCP) on guest failed</b>	ゲスト仮想マシンは正常に起動するが、 <b>DHCP</b> から <b>IP</b> アドレスを取得できない、 <b>PXE</b> プロトコルを使用してブートできない、またはその両方。この原因は多くの場合、ブリッジの転送遅延時間が長く設定されているか、または <b>iptables</b> パッケージとカーネルがチェックサムの難号化 ( <b>mangle</b> ) 規則をサポートしないためです。	「ゲスト上の <b>PXE</b> ブート (または <b>DHCP</b> ) が失敗」
<b>Guest can reach outside network, but cannot reach host when using macvtap interface</b>	<p>ゲストは他のゲストと通信できるが、<b>macvtap</b> (または <b>type='direct'</b>) ネットワークインターフェースを使用するように設定した後はホストマシンに接続できない。</p> <p>この状況は実際にはエラーではなく、<b>macvtap</b> の定義済みの動作です。</p>	「ゲストは外部ネットワークにアクセスできるが、 <b>macvtap</b> インターフェースの使用時にはホストにアクセスできない」
<b>Could not add rule to fixup DHCP response checksums on network 'default'</b>	この警告メッセージはほとんどの場合、いずれの影響もありますが、間違って問題の証拠と見なされることが多くあります。	「 <b>Could not add rule to fixup DHCP response checksums on network 'default'</b> 」



エラー	問題の概要	解決法
Unable to add bridge br0 port vnet0: No such device	このエラーメッセージまたは同様の <b>Failed to add tap interface to bridge 'br0': No such device</b> というメッセージは、ゲストの(またはドメインの)<interface> 定義で指定されたブリッジデバイスが存在しないことを示しています。	「Unable to add bridge br0 port vnet0: No such device」
Warning: could not open /dev/net/tun: no virtual network emulation qemu-kvm: - netdev tap, script=/etc/my-qemu-ifup, id=hostnet0: Device 'tap' could not be initialized	ホストシステムの <b>type='ethernet'</b> (または「汎用イーサネット」) インターフェースの設定後にゲスト仮想マシンが起動しない。このエラーまたは同様のエラーが <b>libvirtd.log</b> または <b>/var/log/libvirt/qemu/name_of_guest.log</b> のどちらか、または両方に表示される。	「ゲストを起動できずエラーが発生: <b>warning: could not open /dev/net/tun</b> 」
Unable to resolve address <i>name_of_host</i> service '49155': Name or service not known	<b>QEMU</b> ゲストの移行が失敗し、このエラーメッセージが不明なホスト名と共に表示される。	「移行に失敗しエラーが発生 <b>Error: unable to resolve address</b> 」
Unable to allow access for disk path /var/lib/libvirt/images /qemu.img: No such file or directory	<b>libvirt</b> がディスクイメージにアクセスできないため、ゲスト仮想マシンを移行できない。	「移行に失敗しエラーが発生: <b>Unable to allow access for disk path: No such file or directory</b> 」
No guest virtual machines are present when libvirtd is started	<b>libvirt</b> デーモンは正常に起動したが、 <b>virsh list --all</b> を実行してもゲスト仮想マシンが表示されない	「 <b>libvirtd</b> の起動時にゲスト仮想マシンがない」
Common XML errors	<b>libvirt</b> は XML ドキュメントを使用して構造化データを保存します。XML ドキュメントのエラーのいくつかは、XML ドキュメントが API で <b>libvirt</b> に渡される際に発生します。このエントリーでは、ゲスト XML 定義の編集に関する指示と、XML 構文および設定における一般的なエラーの詳細を提供します。	「一般的な XML エラー」

### A.20.1. libvirtd の起動に失敗



## 現象

**libvirt** デーモンが自動的に起動しない。**libvirt** デーモンの手動による起動も失敗。

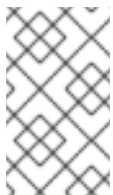
```
# systemctl start libvirtd.service
* Caching service dependencies ...
[ ok ]
* Starting libvirtd ...
/usr/sbin/libvirtd: error: Unable to initialize network sockets. Check
/var/log/messages or run without --daemon for more info.
* start-stop-daemon: failed to start `/usr/sbin/libvirtd'
[ !! ]
* ERROR: libvirtd failed to start
```

また、**/var/log/messages** にはこのエラーの詳細情報がない。

## 調査

以下の行を有効にし、**/etc/libvirt/libvirtd.conf** の **libvirt** のロギングを変更します。行の設定を有効にするには、テキストエディターで **/etc/libvirt/libvirtd.conf** ファイルを開き、以下の行の先頭からハッシュ (または #) 記号を削除して、変更を保存します。

```
log_outputs="3:syslog:libvirtd"
```



### 注記

この行は、**libvirt** が過剰なログメッセージを作成しないように、デフォルトではコメントアウトされています。問題の診断後には、**/etc/libvirt/libvirtd.conf** ファイルでこの行を再度コメントすることが推奨されます。

**libvirt** を再起動し、問題が解決されたかどうかを確認します。

**libvirtd** がまだ正常に開始されない場合には、以下と同様のエラーが表示されます。

```
# systemctl restart libvirtd
Job for libvirtd.service failed because the control process exited with
error code. See "systemctl status libvirtd.service" and "journalctl -xe"
for details.

Sep 19 16:06:02 jsrh libvirtd[30708]: 2017-09-19 14:06:02.097+0000:
30708: info : libvirt version: 3.7.0, package: 1.el7 (Unknown, 2017-09-
06-09:01:55, js
Sep 19 16:06:02 jsrh libvirtd[30708]: 2017-09-19 14:06:02.097+0000:
30708: info : hostname: jsrh
Sep 19 16:06:02 jsrh libvirtd[30708]: 2017-09-19 14:06:02.097+0000:
30708: error : daemonSetupNetworking:502 : unsupported configuration: No
server certif
Sep 19 16:06:02 jsrh systemd[1]: libvirtd.service: main process exited,
code=exited, status=6/NOTCONFIGURED
Sep 19 16:06:02 jsrh systemd[1]: Failed to start Virtualization daemon.

-- Subject: Unit libvirtd.service has failed
-- Defined-By: systemd
-- Support: http://lists.freedesktop.org/mailman/listinfo/systemd-devel
--
```

```
-- Unit libvirtd.service has failed.
--
-- The result is failed.
```

`libvirtd` man ページには、`libvirt` が **Listen for TCP/IP connections** モードで実行された際に、欠落している `cacert.pem` ファイルが TLS 認証として使用されることが示されます。つまり、`--listen` パラメーターが渡されます。

## 解決法

`libvirt` デモンを以下のいずれかの方法で設定します。

- CA 証明書をインストールする。



### 注記

CA 証明書およびシステム認証の設定に関する詳細は、『[Red Hat Enterprise Linux 7 ドメイン ID、認証、およびポリシーガイド](#)』の認証の設定についての章を参照してください。

- TLS を使わずにベア TCP を使用する。`/etc/libvirt/libvirtd.conf` で `listen_tls = 0` および `listen_tcp = 1` を設定します。デフォルト値は、`listen_tls = 1` と `listen_tcp = 0` です。
- `--listen` パラメーターを渡さない。`/etc/sysconfig/libvirtd.conf` で `LIBVIRT_ARGS` 変数を変更します。

## A.20.2. URI のハイパーバイザー接続に失敗する

サーバーへの接続時にいくつかの異なるエラーが発生する (例: `virsh` の実行時)。

### A.20.2.1. CA 証明書を読み込めない

#### 現象

コマンドの実行中に、以下のエラー (または同様のエラー) が表示される。

```
$ virsh -c qemu://$hostname/system_list
error: failed to connect to the hypervisor
error: Cannot read CA certificate '/etc/pki/CA/cacert.pem': No such file
or directory
```

#### 調査

このエラーメッセージは、実際の原因に関して誤解を招くものです。このエラーは、誤って指定された URI や未設定の接続などの様々な要素によって発生することがあります。

## 解決法

### 誤って指定された URI

`qemu://system` または `qemu://session` を接続 URI として指定すると、`virsh` はホスト名の `system` または `session` に接続しようとします。これは、`virsh` が 2 つ目のスラッシュの後のテキストをホストとして認識するためです。

スラッシュを 3 つ使ってローカルホストに接続します。たとえば、**qemu:///system** を指定すると、ローカルホスト上で **libvirtd** の **system** インスタンスに接続するよう **virsh** に指示します。

ホスト名が指定されていると、**QEMU** トランSPORTがデフォルトで**TLS** に設定されます。これで証明書が作成されます。

### 接続が未設定

URI が正しい (例: **qemu[+tls]://server/system**) が、証明書がマシン上で適切に設定されていない。TLS の設定に関する詳細は、[アップストリームの libvirt web サイト](#) を参照してください。

## A.20.2.2. 他の接続エラー

### Unable to connect to server at server : port: Connection refused

デーモンがサーバー上で動作していないか、**listen\_tcp** または **listen\_tls** 設定オプションを使用してリッスンしないように設定されている。

### End of file while reading data: nc: using stream socket: Input/output error

**ssh** トランSPORTが指定されている場合、デーモンはサーバー上で動作していない可能性があります。デーモンがサーバー上で実行しているかどうかを確認して、このエラーを解消します。

## A.20.3. ゲストの起動に失敗しエラーが発生:monitor socket did not show up

### 現象

ゲスト仮想マシン (またはドメイン) が起動に失敗し、以下のエラーメッセージが表示される。

```
# virsh -c qemu:///system create name_of_guest.xml error: Failed to
create domain from name_of_guest.xml error: monitor socket did not show
up.: Connection refused
```

### 調査

このエラーメッセージは以下の点を示しています。

1. **libvirt** が動作している
2. **QEMU** プロセスが起動に失敗した
3. **libvirt** は **QEMU** または **QEMU** エラーモニターソケットに接続しようとした際に終了した

エラーの詳細を確認するために、ゲストログを検証します。

```
# cat /var/log/libvirt/qemu/name_of_guest.log
LC_ALL=C PATH=/sbin:/usr/sbin:/bin:/usr/bin QEMU_AUDIO_DRV=none
/usr/bin/qemu-kvm -S -M pc -enable-kvm -m 768 -smp
1,sockets=1,cores=1,threads=1 -name name_of_guest -uuid ebfaadbe-e908-
ba92-fdb8-3fa2db557a42 -nodefaults -chardev
socket,id=monitor,path=/var/lib/libvirt/qemu/name_of_guest.monitor,serve
r,nowait -mon chardev=monitor,mode=readline -no-reboot -boot c -kernel
```

```

/var/lib/libvirt/boot/vmlinuz -initrd /var/lib/libvirt/boot/initrd.img -
append
method=http://www.example.com/pub/product/release/version/x86_64/os/ -
drive file=/var/lib/libvirt/images/name_of_guest.img,if=none,id=drive-
ide0-0-0,boot=on -device ide-drive,bus=ide.0,unit=0,drive=drive-ide0-0-
0,id=ide0-0-0 -device virtio-net-
pci,vlan=0,id=net0,mac=52:40:00:f4:f1:0a,bus=pci.0,addr=0x4 -net
tap,fd=42,vlan=0,name=hostnet0 -chardev pty,id=serial0 -device isa-
serial,chardev=serial0 -usb -vnc 127.0.0.1:0 -k en-gb -vga cirrus -
device virtio-balloon-pci,id=balloon0,bus=pci.0,
addr=0x3
char device redirected to /dev/pts/1
qemu: could not load kernel '/var/lib/libvirt/boot/vmlinuz':
Permission denied

```

## 解決法

ゲストログには、エラーの修正に必要な詳細が記載されています。

ゲストが **libvirt** の 0.9.5 よりも前のバージョンを実行中にホスト物理マシンがシャットダウンした場合、**libvirt** ゲストの **init** スクリプトはゲストの管理保存を実行しようとします。管理保存が不完全な場合 (たとえば、管理保存イメージがディスクにフラッシュされる前に電源が遮断される場合など)、保存イメージは破損し、**QEMU** はロードしません。古いバージョンの **libvirt** は破損を認識せず、問題は永続化します。この場合、ゲストログには、引数の1つとして **-incoming** の使用を試みたことが表示されます。これは、**libvirt** が保存状態のファイル内での移行により **QEMU** の起動を試みていることを意味します。

この問題は、**virsh managedsave-remove name\_of\_guest** を実行して破損した管理保存 (**managedsave**) イメージを削除することで修正できます。新しいバージョンの **libvirt** は最初の段階で破損を回避する手順を実行し、**virsh start --force-boot name\_of\_guest** を追加して管理保存 (**managedsave**) イメージを無視します。

## A.20.4.internal error cannot find character device (null)

### 現象

このエラーメッセージは、ゲスト仮想マシンのコンソールに接続を試みる際に表示されます。

```

# virsh console test2
Connected to domain test2
Escape character is ^]
error: internal error cannot find character device (null)

```

### 調査

このエラーメッセージは、ゲスト仮想マシン用にシリアルコンソールが設定されていないことを示しています。

## 解決法

ゲストの XML ファイルでシリアルコンソールを設定します。

### 手順A.9 ゲストの XML ファイルでのシリアルコンソールの設定

1. **virsh edit** を使用して、以下の XML をゲスト仮想マシンの XML に追加します。

-

```
<serial type='pty'>
  <target port='0' />
</serial>
<console type='pty'>
  <target type='serial' port='0' />
</console>
```

2. ゲストのカーネルコマンドラインでコンソールを設定します。

これを実行するには、ゲスト仮想マシンにログインして **/boot/grub2/grub.cfg** ファイルを直接編集するか、または **virt-edit** コマンドラインツールを使用します。ゲストのカーネルコマンドラインに以下を追加します。

```
console=ttyS0,115200
```

3. 以下のコマンドを実行します。

```
# virsh start vm && virsh console vm
```

### A.20.5. ゲスト仮想マシンの起動が停止してエラーが発生: **No boot device**

#### 現象

既存のディスクイメージからゲスト仮想マシンを作成した後、ゲストの起動が停止し、エラーメッセージ **No boot device** が表示される。ただし、**QEMU** コマンドを直接使うとゲスト仮想マシンは正常に起動する。

#### 調査

既存ディスクイメージをインポートするコマンドでディスクのバスの種類が指定されていない。

```
# virt-install \
--connect qemu:///system \
--ram 2048 -n rhel_64 \
--os-type=linux --os-variant=rhel5 \
--disk path=/root/RHEL-Server-5.8-64-
virtio.qcow2,device=disk,format=qcow2 \
--vcpus=2 --graphics spice --noautoconsole --import
```

ただし、**QEMU** を直接使用してゲスト仮想マシンを起動するために使用されるコマンドラインではバスの種類に **virtio** を使用することが示されます。

```
# ps -ef | grep qemu
/usr/libexec/qemu-kvm -monitor stdio -drive file=/root/RHEL-Server-5.8-
32-virtio.qcow2,index=0,if=virtio,media=disk,cache=none,format=qcow2 -
net nic,vlan=0,model=rtl8139,macaddr=00:30:91:aa:04:74 -net
tap,vlan=0,script=/etc/qemu-ifup,downscript=no -m 2048 -smp
2,cores=1,threads=1,sockets=2 -cpu qemu64,+sse2 -soundhw ac97 -rtc-td-
hack -M rhel5.6.0 -usbdevice tablet -vnc :10 -boot c -no-kvm-pit-
reinjection
```

インポートされたゲストについて、**libvirt** で生成されたゲストの XML に **bus=** があることに注意してください。

```
<domain type='qemu'>
  <name>rhel_64</name>
  <uuid>6cd34d52-59e3-5a42-29e4-1d173759f3e7</uuid>
  <memory>2097152</memory>
  <currentMemory>2097152</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='rhel5.4.0'>hvm</type>
    <boot dev='hd' />
  </os>
  <features>
    <acpi />
    <apic />
    <pae />
  </features>
  <clock offset='utc'>
    <timer name='pit' tickpolicy='delay' />
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' />
      <source file='/root/RHEL-Server-5.8-64-virtio.qcow2' />
      <emphasis role="bold"><target dev='hda' bus='ide' /></emphasis>
      <address type='drive' controller='0' bus='0' unit='0' />
    </disk>
    <controller type='ide' index='0' />
    <interface type='bridge'>
      <mac address='54:52:00:08:3e:8c' />
      <source bridge='br0' />
    </interface>
    <serial type='pty'>
      <target port='0' />
    </serial>
    <console type='pty'>
      <target port='0' />
    </console>
    <input type='mouse' bus='ps2' />
    <graphics type='vnc' port='-1' autoport='yes' keymap='en-us' />
    <video>
      <model type='cirrus' vram='9216' heads='1' />
    </video>
  </devices>
</domain>
```

ディスクのバスの種類は **ide** に設定されており、これは **libvirt** によって設定されるデフォルト値です。これは間違ったバスの種類で、インポートされたゲストが正常に起動できない原因となっています。

## 解決法

### 手順A.10 ディスクのバスの種類の訂正

1. インポートされたゲスト仮想マシンの定義を解除し、以下のように**bus=virtio**を使って再度インポートします。

```
# virsh destroy rhel_64
# virsh undefine rhel_64
# virt-install \
--connect qemu:///system \
--ram 1024 -n rhel_64 -r 2048 \
--os-type=linux --os-variant=rhel5 \
--disk path=/root/RHEL-Server-5.8-64-
virtio.qcow2,device=disk,bus=virtio,format=qcow2 \
--vcpus=2 --graphics spice --noautoconsole --import
```

2. **virsh edit** を使ってインポートされたゲストの XML を編集し、ディスクのバスの種類を訂正します。

## A.20.6. Virtual network *default* has not been started

### 現象

通常、**libvirt** パッケージの一部として **default** という名前の仮想ネットワークの設定がインストールされており、**libvirtd** の起動時に自動起動ように設定されています。

**default** ネットワーク (または他のローカルで作成されたネットワーク) が起動しない場合、接続にそのネットワークを使うように設定されている仮想マシンも起動に失敗し、以下のエラーメッセージが表示されます。

```
Virtual network default has not been started
```

### 調査

**libvirt** 仮想ネットワークの起動の失敗についての一般的な理由の1つは、そのネットワーク上でクライアントからのDHCP および DNS 要求に対応するために必要な **dnsmasq** インスタンスの起動に失敗したためです。

これが理由であることを確かめるには、**root** シェルから **virsh net-start default** を実行し、**default** 仮想ネットワークを起動します。

この方法で仮想ネットワークが正常に起動しない場合は、**/var/log/libvirt/libvirtd.log** を開いて詳細のエラーログメッセージを表示します。

以下と同様のメッセージが表示される場合、問題は **libvirt** のブリッジ上ですでにリッスンしているシステム全体の **dnsmasq** インスタンスに関係する可能性が高くなります。このインスタンスは、**libvirt** 独自の **dnsmasq** イスタンスがリッスンすることを妨げています。エラーメッセージで留意すべき最も重要な部分は、**dnsmasq** と **exit status 2** になります。

```
Could not start virtual network default: internal error
Child process (/usr/sbin/dnsmasq --strict-order --bind-interfaces
--pid-file=/var/run/libvirt/network/default.pid --conf-file=
--except-interface lo --listen-address 192.168.122.1
--dhcp-range 192.168.122.2,192.168.122.254
--dhcp-leasefile=/var/lib/libvirt/dnsmasq/default.leases
--dhcp-lease-max=253 --dhcp-no-override) status unexpected: exit status
2
```

## 解決法

物理ネットワークで DHCP の応答にマシンが `dnsmasq` を使用していない場合は、`dnsmasq` を完全に無効にします。

物理ネットワークの DHCP への応答に `dnsmasq` の実行が必要な場合は、`/etc/dnsmasq.conf` ファイルを編集します。最初の行と続く 2 行のうちのいずれかにコメントマークの追加または削除を実行します。これらの 3 行すべてからコメントを追加したり、削除したりしないでください。

```
bind-interfaces
interface=name_of_physical_interface
listen-address=chosen_IP_address
```

この変更を加えてファイルを保存した後、システム全体の `dnsmasq` サービスを再起動します。

次に **`virsh net-start default`** コマンドを使用して *default* ネットワークを起動します。

仮想マシンを起動します。

## A.20.7. ゲスト上の PXE ブート (または DHCP) が失敗

### 現象

ゲスト仮想マシンは正常に起動するが、DHCP から IP アドレスを取得できないか、PXE プロトコルを使用してブートを実行できない、またはその両方。このエラーには、ブリッジの転送遅延時間が長く設定されている場合と `iptables` パッケージとカーネルがチェックサムの難号化 (mangle) 規則をサポートしない場合という 2 つの一般的な原因があります。

### ブリッジの転送遅延時間が長い

#### 調査

これは、このエラーの最も一般的な原因になります。ゲストのネットワークインターフェースが STP (Spanning Tree Protocol) 対応のブリッジデバイスに接続しており、かつ長時間の転送遅延時間が設定されていると、ゲストがブリッジに接続してからその転送遅延時間が経過してからでなければゲスト仮想マシンからブリッジにネットワークパケットを転送しません。この遅延により、インターフェースからのトラフィックの監視、背後での MAC アドレスの決定、ネットワークトポロジー内の転送ループ防止がブリッジ時間で可能になります。

転送遅延がゲストの PXE や DHCP クライアントのタイムアウトよりも長い場合、クライアントの操作は失敗し、ゲストは (PXE の場合) 起動に失敗するか、または (DHCP の場合) IP アドレスの取得に失敗します。

## 解決法

これが原因の場合は、ブリッジにおける転送遅延を 0 に変更するか、ブリッジの STP を無効にするか、またはこの両方を実行します。



### 注記

この解決法は、ブリッジが複数のネットワーク接続に使用されておらず、複数のエンドポイントを単一のネットワーク接続のみに使用されている場合にのみ適用されます (`libvirt` で使用される最も一般的なブリッジのユースケース)。

ゲストが `libvirt` が管理する仮想ネットワークに接続するインターフェースを備えている場合、そのネットワークの定義を編集し、再起動します。たとえば、以下のコマンドでデフォルトのネットワークを編集します。



```
# virsh net-edit default
```

<bridge> 要素に以下の属性を追加します。

```
<name_of_bridge='virbr0' delay='0' stp='on' />
```



### 注記

**delay='0'** と **stp='on'** は仮想ネットワークのデフォルト設定なので、このステップは設定がデフォルトから変更されている場合にのみ必要となります。

ゲストインターフェースが **libvirt** 外で設定されているホストブリッジに接続されている場合は、遅延設定を変更します。

**/etc/sysconfig/network-scripts/ifcfg-name\_of\_bridge** ファイルで以下の行を追加または編集して、STP を on にし、遅延を 0 秒にします。

```
STP=on DELAY=0
```

設定ファイルの変更にブリッジデバイスを再起動します。

```
/usr/sbin/ifdown name_of_bridge
/usr/sbin/ifup name_of_bridge
```



### 注記

**name\_of\_bridge** がネットワークの **root** ブリッジでない場合、そのブリッジの遅延時間は最終的には **root** ブリッジに設定されている遅延時間にリセットされます。この場合の唯一の解決法は、**name\_of\_bridge** で STP を完全に無効にすることです。

## iptables パッケージとカーネルがチェックサムの難号化 (mangle) 規則をサポートしない調査

このメッセージは、以下の 4 つの条件すべてが該当する場合にのみ問題となります。

- ゲストが **virtio** ネットワークデバイスを使用している。

その場合、設定ファイルに **model type='virtio'** が含まれます。

- ホストに **vhost-net** モジュールがロードされている。

これは、**ls /dev/vhost-net** が空の結果を返さない場合に該当します。

- ゲストがホスト上で直接実行されている DHCP サーバーから IP アドレスを取得しようとしている。
- ホスト上の **iptables** バージョンが 1.4.10 よりも古いバージョンである。

**iptables 1.4.10** は **libxt\_CHECKSUM** 拡張を追加する最初のバージョンでした。**libvirtd** ログに以下のメッセージが表示される場合は、これに該当します。

```
warning: Could not add rule to fixup DHCP response checksums
on network default
warning: May need to update iptables package and kernel to
support CHECKSUM rule.
```



### 重要

この一覧の最初の 3 つの条件すべてが該当していなければ上記の警告メッセージは問題を示すものではなく、無視することができます。

これらの条件が当てはまる場合、ホストからゲストへ送信される UDP パケットには未算出のチェックサムがあります。これにより、ホストの UDP パケットがゲストのネットワークスタックには無効のように表示されます。

### 解決法

この問題を解決するには、上記の 4 つの条件のいずれかを無効にします。最善の解決法は、可能であればホストの **iptables** およびカーネルを **iptables-1.4.10** 以降に更新することです。または、この特定のゲストの **vhost-net** ドライバーを無効にすることが最も具体的な修正方法になります。これを実行するには、以下のコマンドでゲストの設定を編集します。

```
virsh edit name_of_guest
```

**<interface>** セクションの **<driver>** 行を変更するか、またはこれを追加します。

```
<interface type='network'>
  <model type='virtio' />
  <driver name='qemu' />
  ...
</interface>
```

変更を保存し、ゲストをシャットダウンしてから再起動します。

この問題が解決されない場合、**firewalld** とデフォルトの **libvirt** ネットワーク間に競合がある可能性があることが原因として考えられます。

これを修正するには、**service firewalld stop** コマンドで **firewalld** を停止し、**service libvirtd restart** コマンドで **libvirt** を再起動します。

## A.20.8. ゲストは外部ネットワークにアクセスできるが、**macvtap** インターフェースの使用時にはホストにアクセスできない

### 現象

ゲストは他のゲストと通信できるが、**macvtap** (または **type='direct'**) ネットワークインターフェースを使用するように設定された後にはホストマシンに接続できない。

### 調査

Virtual Ethernet Port Aggregator (VEPA) や VN-Link 対応スイッチに接続していない場合でも、**macvtap** インターフェースは役に立ちます。インターフェースのモードを **bridge** に設定すると、

非常に簡単な方法でゲストを物理ネットワークに直接接続することができます。この場合、従来のホストブリッジデバイスの使用に伴う設定の問題（または、**NetworkManager** の非互換性）がありません。

ただし、ゲスト仮想マシンが **macvtap** などの **type='direct'** ネットワークインターフェースを使用するように設定されていると、ネットワーク上で他のゲストや他の外部ホストと通信する機能がありながら、ゲストは自らのホストと通信できなくなります。

この状況は、実際にはエラーではなく **macvtap** の定義済みの動作です。ホストの物理イーサネットが **macvtap** ブリッジに割り当てられている方法が原因で、物理インターフェースに転送されるゲストからそのブリッジへのトラフィックは、ホストの IP スタックに送り返されることはありません。さらに、物理インターフェースに送信されたホストの IP スタックからのトラフィックも **macvtap** ブリッジに送り返されず、ゲストに転送することができません。

## 解決法

**libvirt** を使って、分離されたネットワークと、このネットワークに接続する各ゲスト仮想マシンの 2 つ目のインターフェースを作成します。これでホストとゲストはこの分離されたネットワーク上で直接通信でき、**NetworkManager** との互換性も維持できます。

### 手順A.11 libvirt による分離ネットワークの作成

1. 以下の XML を **/tmp/isolated.xml** ファイルに追加して保存します。**192.168.254.0/24** がネットワーク上ですでに使用されている場合、別のネットワークを選択できます。

```
...
<network>
  <name>isolated</name>
  <ip address='192.168.254.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.254.2' end='192.168.254.254' />
    </dhcp>
  </ip>
</network>
...
```

図A.3 分離されたネットワーク XML

2. コマンド **virsh net-define /tmp/isolated.xml** を使ってネットワークを作成します。
3. **virsh net-autostart isolated** コマンドを使ってネットワークの自動起動を設定します。
4. **virsh net-start isolated** コマンドを使ってネットワークを起動します。
5. **virsh edit name\_of\_guest** を使って、ネットワーク接続に **macvtap** を使用する各ゲストの設定を編集し、以下と同様の新たな **<interface>** を **<devices>** セクションに追加します。**(<model type='virtio' /> 行を含めるかはオプション)**

```
...
<interface type='network' trustGuestRxFilters='yes'>
  <source network='isolated' />
  <model type='virtio' />
</interface>
```

#### 図A.4 インターフェースデバイス XML

6. 各ゲストをシャットダウンし、再起動します。

これで、ゲストは **192.168.254.1** アドレスにあるホストにアクセスでき、ホストは各ゲストが DHCP から取得した IP アドレスでゲストにアクセスできます (または、ゲストに手動で IP アドレスを設定することもできます)。この新たなネットワークはこのホストとゲスト専用として分離されているので、ゲストからの他の通信はすべて **macvtap** インターフェースを使用することになります。詳細は、「[ネットワークインターフェース](#)」を参照してください。

### A.20.9. Could not add rule to fixup DHCP response checksums on network 'default'

#### 現象

以下のメッセージが表示されます。

```
Could not add rule to fixup DHCP response checksums on network 'default'
```

#### 調査

このメッセージはエラーに見えますが、ほとんどの場合問題ありません。

#### 解決法

ゲスト仮想マシンが DHCP から IP アドレスを取得できないという問題が発生していない限り、このメッセージは無視してかまいません。

この問題が発生している場合は、この状況の詳細について「[ゲスト上の PXE ブート \(または DHCP\) が失敗](#)」を参照してください。

### A.20.10. Unable to add bridge br0 port vnet0: No such device

#### 現象

以下のエラーメッセージが表示されます。

```
Unable to add bridge name_of_bridge port vnet0: No such device
```

たとえばブリッジ名が **br0** の場合、エラーメッセージは以下のようになります。

```
Unable to add bridge br0 port vnet0: No such device
```

**libvirt** のバージョン **0.9.6** 以前では、以下のようなメッセージになります。

```
Failed to add tap interface to bridge name_of_bridge: No such device
```

たとえばブリッジ名が **br0** の場合、エラーメッセージは以下のようになります。

```
Failed to add tap interface to bridge 'br0': No such device
```

## 調査

いずれのエラーメッセージも、ゲストの(またはドメインの) **<interface>** 定義で指定されたブリッジデバイスが存在しないことを示しています。

エラーメッセージで表示されたブリッジデバイスが存在しないことを確認するには、シェルコマンド **ip addr showbr0** を使います。

以下のようなメッセージが表示されると、その名前のブリッジが存在しないことが確認できます。

```
br0: error fetching interface information: Device not found
```

存在しない場合は、解決法に進みます。

ただし、メッセージが以下のようであれば、問題は別に存在します。

```
br0          Link encap:Ethernet  HWaddr 00:00:5A:11:70:48
              inet addr:10.22.1.5   Bcast:10.255.255.255  Mask:255.0.0.0
              UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
              RX packets:249841 errors:0 dropped:0 overruns:0 frame:0
              TX packets:281948 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:0
              RX bytes:106327234 (101.4 MiB)  TX bytes:21182634 (20.2 MiB)
```

## 解決法

### virshで既存のブリッジを編集または新規ブリッジを作成

**virsh** を使用して既存のブリッジまたはネットワークの設定を編集するか、またはブリッジデバイスをホストシステム設定に追加します。

#### virshを使った既存ブリッジ設定の編集

**virsh edit name\_of\_guest** を使って既存のブリッジまたはネットワークを使用するための **<interface>** 定義を変更します。

たとえば、**type='bridge'** を **type='network'** に、**<source bridge='br0' />** を **<source network='default' />** に変更します。

#### virshを使ったホストブリッジの作成

**libvirt** のバージョン 0.9.8 以降では、**virsh iface-bridge** コマンドを使用してブリッジデバイスを作成できます。これにより、**eth0** のあるブリッジデバイス **br0** が作成され、ブリッジの一部として設定された物理ネットワークインターフェースが割り当てられます。

```
virsh iface-bridge eth0 br0
```

オプション: このブリッジを削除して、以下のコマンドで元の **eth0** 設定を復元することもできます。

```
virsh iface-unbridge br0
```

## ホストブリッジを手動作成

**libvirt** の古いバージョンでは、ホスト上にブリッジデバイスを手動で作成することができます。作成方法については「[libvirt を使用したブリッジネットワーク](#)」を参照してください。

### A.20.11. ゲストを起動できずエラーが発生:warning: could not open /dev/net/tun

#### 現象

ホストシステムの **type='ethernet'** (または汎用イーサネット) インターフェースの設定後にゲスト仮想マシンが起動しない。以下と同様のエラーメッセージが **libvirtd.log** または **/var/log/libvirt/qemu/name\_of\_guest.log** のどちらか、または両方に表示される。

```
warning: could not open /dev/net/tun: no virtual network emulation qemu-kvm: -netdev tap,script=/etc/my-qemu-ifup,id=hostnet0: Device 'tap' could not be initialized
```

#### 調査

汎用イーサネットのインターフェースタイプ(<**interface type='ethernet'**>)の使用は推奨されません。これを使用するには、**QEMU** およびゲストにおけるホストの潜在的なセキュリティの不備に対する保護レベルを下げる必要があるからです。しかし、このタイプのインターフェースを使用して **libvirt** で直接サポートされていない別の機能を活用することが必要になることもあります。たとえば、**openvswitch** は **libvirt** では **libvirt-0.9.11** までサポートされていなかったの、**libvirt** の古いバージョンでは <**interface type='ethernet'**> がゲストを **openvswitch** ブリッジに接続する唯一の方法でした。

しかし、ホストシステムに他の変更を加えることなく <**interface type='ethernet'**> インターフェースを設定すると、ゲスト仮想マシンは正常に起動しなくなります。

この失敗の原因は、この種類のインターフェースでは、**QEMU** に呼び出されるスクリプトはタップデバイスを操作する必要があることによります。しかし、**type='ethernet'** が設定されていると、**QEMU** をロックダウンする目的で、**libvirt** と **SELinux** はこれを防ぐためのチェックをいくつか実施します(通常、**libvirt** はタップデバイス作成および操作のすべてを実行し、タップデバイスのオープンファイル記述子を **QEMU** に渡します)。

#### 解決法

ホストシステムが汎用イーサネットインターフェースと互換性を持つように再設定します。

#### 手順A.12 ホストシステムが汎用イーサネットインターフェースを使用するように再設定

1. **/etc/selinux/config** で **SELINUX=permissive** を設定し、SELinux to permissive に設定します。

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of
#                  enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=permissive
# SELINUXTYPE= can take one of these two values:
#     targeted - Targeted processes are protected,
#     mls - Multi Level Security protection.
```

```
SELINUXTYPE=targeted
```

2. root シェルから **setenforce permissive** コマンドを実行します。

3. **/etc/libvirt/qemu.conf** で、以下の行を追加または編集します。

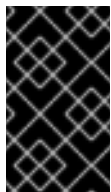
```
clear_emulator_capabilities = 0

user = "root"

group = "root"

cgroup_device_acl = [
    "/dev/null", "/dev/full", "/dev/zero",
    "/dev/random", "/dev/urandom",
    "/dev/ptmx", "/dev/kvm", "/dev/kqemu",
    "/dev/rtc", "/dev/hpet", "/dev/net/tun",
```

4. **libvirtd** を再起動します。



### 重要

上記のステップはそれぞれ **QEMU** ゲストドメインに対するホストのセキュリティー保護を大幅に低下させるので、この設定は **<interface type='ethernet'>** を使用する以外に方法がない場合にのみ使用してください。



### 注記

SELinux に関する詳細は、『[Red Hat Enterprise Linux 7 SELinux ユーザーおよび管理者のガイド](#)』を参照してください。

## A.20.12. 移行に失敗しエラーが発生 **Error: unable to resolve address**

### 現象

**QEMU** ゲストの移行が失敗し、以下のエラーメッセージが表示される。

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
error: Unable to resolve address name_of_host service '49155': Name or
service not known
```

たとえば、移行先のホスト名が **"newyork"** の場合、エラーメッセージは以下のようになります。

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
error: Unable to resolve address 'newyork' service '49155': Name or
service not known
```

しかし、ホスト名「**newyork**」をどこにも使用していないため、このエラーは不明メッセージのように思われるかもしれません。



## 調査

移行時に、移行先ホスト上で稼働している **libvirt** は移行データの受信が予想されるアドレスおよびポートから **URI** を作成し、これを移行元ホスト上で稼働している **libvirt** に送信します。

上記の例では、移行先ホスト (**192.168.122.12**) は名前を **'newyork'** に設定しています。何らかの理由で、そのホスト上で実行中の **libvirt** は IP アドレスに対して名前を解決できず、送信されるべきものが有効なままとなっています。このため、移行元の **libvirt** が名前を解決することを予期してホスト名 **'newyork'** が返されました。DNS が適切に設定されていなかったり、**/etc/hosts** にローカルループバックアドレス (**127.0.0.1**) に関連するホスト名がある場合にこのようなことが発生します。

移行データに使用されるアドレスは、移行先 **libvirt** への接続に使用されるアドレス (例: **qemu+tcp://192.168.122.12/system**) から自動的に決定されることはありません。これは、移行先 **libvirt** と通信するために、移行元の **libvirt** は (別のマシンで実行中かもしれない) **virsh** が必要とするネットワークインフラストラクチャーとは別のものを使用する必要がある場合があるためです。

## 解決法

最善の解決法として、DNS を正しく設定し、移行に関連するすべてのホストがすべてのホスト名を解決できるようにすることができます。

DNS をこのように設定できない場合は、各ホスト上で、移行に使用するすべてのホストのリストを **/etc/hosts** ファイルに手動で追加することができます。ただし、動的な環境ではこのようなリストの一貫性を維持することは容易ではありません。

どの方法でもホスト名を解決可能にできない場合、**virsh migrate** は移行ホストの特定をサポートします。

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
tcp://192.168.122.12
```

移行先 **libvirt** は **tcp://192.168.122.12** URI を取得し、自動生成されたポート番号を追加します。この番号が望ましくない場合は (たとえば、ファイアウォール設定との関連により適切でない場合)、ポート番号は以下のコマンドで指定できます。

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
tcp://192.168.122.12:12345
```

別のオプションとして、トンネル化した移行を使用することもできます。トンネル化した移行では移行データ用に別の接続を作成しませんが、その代わりに移行先 **libvirt** との通信で使用する接続でデータをトンネルします (例: **qemu+tcp://192.168.122.12/system**)。

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system --p2p --tunnelled
```

### A.20.13. 移行に失敗しエラーが発生: Unable to allow access for disk path: No such file or directory

#### 現象

**libvirt** がディスクイメージにアクセスできないため、ゲスト仮想マシン (またはドメイン) を移行できない。



```
# virsh migrate qemu qemu+tcp://name_of_host/system
error: Unable to allow access for disk path
/var/lib/libvirt/images/qemu.img: No such file or directory
```

たとえば、移行先のホスト名が「**newyork**」の場合、エラーメッセージは以下のようになります。

```
# virsh migrate qemu qemu+tcp://newyork/system
error: Unable to allow access for disk path
/var/lib/libvirt/images/qemu.img: No such file or directory
```

## 調査

デフォルトでは、移行で移動するのは実行中のゲストのメモリー内の状態のみです (メモリーまたは CPU 状態など)。移行中にはディスクイメージは移動しませんが、両方のホストから同じパスでアクセスできる状態である必要があります。

## 解決法

両方のホストの同じ位置に共有ストレージをセットアップし、マウントします。最も簡単な方法として、**NFS** を使用することができます。

### 手順A.13 共有ストレージのセットアップ

1. ホスト上に共有ストレージとして機能する **NFS** サーバーをセットアップします。関連するすべてのホストが **NFS** で共有ストレージにアクセスしている限り、**NFS** サーバーには移行関連のいずれかのホストを使用できます。

```
# mkdir -p /exports/images
# cat >>/etc/exports <<EOF
/exports/images    192.168.122.0/24(rw,no_root_squash)
EOF
```

2. **libvirt** を実行しているすべてのホスト上の共通の場所にエクスポートされたディレクトリーをマウントします。たとえば、**NFS** サーバーの IP アドレスが **192.168.122.1** の場合は、以下のコマンドでディレクトリーをマウントします。

```
# cat >>/etc/fstab <<EOF
192.168.122.1:/exports/images /var/lib/libvirt/images nfs auto
0 0
EOF
# mount /var/lib/libvirt/images
```

## 注記

**NFS** を使ってホストからローカルディレクトリーをエクスポートし、別のホストの同じパスにマウントすることはできません。ディスクイメージの保存に使われるディレクトリーは、両方のホストで共有ストレージからマウントされる必要があります。これが正確に設定されていない場合、ゲスト仮想マシンは移行時にそのディスクイメージへのアクセスを失う可能性があります。これは、ゲストを移行先に正常に移行した後に、移行元ホストの **libvirt** デーモンがディスクイメージ上の所有者、権限および SELinux ラベルを変更する可能性があるからです。

**libvirt** は、ディスクイメージが共有ストレージの場所からマウントされたことを検出すると、これらの変更を実施しません。

## A.20.14. libvirtd の起動時にゲスト仮想マシンがない

### 現象

**libvirt** デーモンは正常に起動したが、ゲスト仮想マシンが見当たらない

```
# virsh list --all
 Id      Name                               State
-----
```

### 調査

この問題の原因としていくつもの原因が考えられます。以下のテストを実施して原因を特定します。

#### KVM カーネルモジュールの確認

KVM カーネルモジュールがカーネルに挿入されていることを確認します。

```
# lsmod | grep kvm
kvm_intel          121346  0
kvm                328927  1 kvm_intel
```

AMD マシンを使用している場合は、**root** シェルで同様の **lsmod | grep kvm\_amd** コマンドを使用してカーネルに **kvm\_amd** カーネルモジュールが挿入されていることを確認します。

モジュールが確認されない場合は、**modprobe <modulename>** コマンドを使用して挿入します。



#### 注記

一般的ではありませんが、KVM 仮想化サポートがカーネルにコンパイルされている場合もあります。その場合は、モジュールは必要ありません。

#### 仮想化拡張機能の確認

仮想化拡張機能がホスト上でサポートされ、有効にされていることを確認します。

```
# egrep "(vmx|svm)" /proc/cpuinfo
flags  : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv svm_lock
nrip_save
flags  : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv svm_lock
nrip_save
```

BIOS 設定内のファームウェア設定で仮想化拡張機能を有効にします。詳細は、お使いのハードウェアの資料を参照してください。

#### クライアント URI 設定の確認

クライアントの URI が適切に設定されていることを確認します。

```
# virsh uri
vbox:///system
```

たとえば、このメッセージは URI が **QEMU** ではなく **VirtualBox** ハイパーバイザーに接続されていることを示し、本来は **QEMU** ハイパーバイザーに接続するように設定されているはずの URI

の設定エラーを示しています。URIが**QEMU**に正常に接続している場合は、メッセージは以下のようになります。

```
# virsh uri
qemu:///system
```

他のハイパーバイザーが存在し、**libvirt**がこのハイパーバイザーとデフォルトで通信する場合に、この状況が発生します。

## 解決法

これらのテスト実行後に、以下のコマンドでゲスト仮想マシンの一覧を表示します。

```
# virsh list --all
```

### A.20.15. unable to connect to server at 'host:16509': Connection refused ... error: failed to connect to the hypervisor

#### 現象

**libvirtd**がTCPポートをリッスンしている間に、接続が失敗する。

```
# virsh -c qemu+tcp://host/system
error: failed to connect to the hypervisor
error: unable to connect to server at 'host:16509': Connection refused
```

**/etc/libvirt/libvirtd.conf**で設定を変更した後でも、**libvirt**デーモンがTCPポートでリッスンしない。

```
# grep listen_ /etc/libvirt/libvirtd.conf
listen_tls = 1
listen_tcp = 1
listen_addr = "0.0.0.0"
```

ただし、設定の変更後も**libvirt**のTCPポートは開いていない。

```
# netstat -lntp | grep libvirtd
#
```

#### 調査

**libvirt**デーモンが**--listen**オプションなしに起動したことを以下のコマンドを実行して確認します。

```
# ps aux | grep libvirtd
root      10749  0.1  0.2 558276 18280 ?        Ssl  23:21   0:00
/usr/sbin/libvirtd
```

出力には**--listen**オプションが含まれていません。

## 解決法

**--listen**オプションを使用してデーモンを起動します。

これを実行するには、`/etc/sysconfig/libvirtd` ファイルを編集し、以下の行のコメント解除を行います。

```
# LIBVIRT_ARGS="--listen"
```

次に、以下のコマンドで **libvirtd** サービスを再起動します。

```
# /bin/systemctl restart libvirtd.service
```

## A.20.16. 一般的な XML エラー

**libvirt** ツールは XML ドキュメントを使用して構造化データを保存します。XML ドキュメントの様々なエラーは、XML ドキュメントが API で **libvirt** に渡される際に発生します。一般的な XML エラー (誤りのある XML タグや不適切な値、要素の欠如を含む) のいくつかを以下で詳述します。

### A.20.16.1. ドメイン定義の編集

これは推奨されませんが、ゲスト仮想マシン (またはドメイン) の XML ファイルを手動で編集することが必要になることがあります。編集の目的でゲストの XML にアクセスするには、以下のコマンドを使用します。

```
# virsh edit name_of_guest.xml
```

このコマンドで、ゲスト仮想マシンの現在の定義によるファイルがテキストエディターで開かれます。編集して変更を保存した後、XML はリロードされ、**libvirt** で解析されます。XML が正しければ、以下のメッセージが表示されます。

```
# virsh edit name_of_guest.xml

Domain name_of_guest.xml XML configuration edited.
```



#### 重要

**virsh** で **edit** コマンドを使用して XML ドキュメントを編集する際は、エディターを終了する前にすべての変更を保存します。

XML ファイルの保存後に、**xmllint** コマンドで XML が整形形式かどうかを確認します。または、**virt-xml-validate** コマンドで使用法の問題を確認します。

```
# xmllint --noout config.xml
```

```
# virt-xml-validate config.xml
```

エラーが表示されない場合、XML 記述は整形形式で **libvirt** スキーマに一致していることになります。スキーマはすべての制約要素をキャッチする訳ではありませんが、レポートされたエラーを修正することでトラブルシューティングが促進されます。

#### **libvirt** で保存される XML ドキュメント

これらのドキュメントには、ゲストの状態や設定の定義が含まれます。ドキュメントは自動生成され、手動での編集することはできません。ドキュメントにあるエラーには、破損したドキュメント

のファイル名が含まれています。このファイル名は、URI によって定義されたホストマシン上でのみ有効で、コマンドが実行されたマシンを参照する場合があります。

**libvirt** が作成したファイルのエラーはまれにしかありません。エラーの原因となり得るのは **libvirt** のダウングレードです。新しいバージョンの **libvirt** は、古いバージョンが生成した XML を常に取り込み込めるのに対して、古いバージョンの **libvirt** は、新しいバージョンで追加された XML 要素によって混乱する可能性があります。

### A.20.16.2. XML 構文エラー

構文エラーは、XML パーサーがキャッチします。エラーメッセージには、問題特定のための情報が含まれています。

以下の例では、XML パーサーからのエラーメッセージは 3 行で構成されています。最初の行はエラーメッセージを表示し、残りの 2 行にはエラーを含む XML コードのコンテキストと場所が含まれています。3 行目には、上の行のどこにエラーがあるかのおおよその位置を示す表示が含まれています。

```
error: (name_of_guest.xml):6: StartTag: invalid element name
<vcpu>2</vcpu><
-----^
```

このメッセージに含まれている情報

**(name\_of\_guest.xml)**

これは、エラーを含むドキュメントのファイル名です。括弧内のファイル名は、メモリーから解析された XML ドキュメントを表現するシンボリック名で、ディスク上のファイルに直接対応するものではありません。括弧内に含まれていないファイル名は、接続先に配置されているローカルファイルです。

**6**

これは、XML ファイル内でのエラーを含む行番号です。

**StartTag: invalid element name**

これは **libxml2** パーサーからのエラーメッセージで、特定の XML エラーを記述するものです。

#### A.20.16.2.1. ドキュメント内の不要な <

現象

以下のエラーが発生する。

```
error: (name_of_guest.xml):6: StartTag: invalid element name
<vcpu>2</vcpu><
-----^
```

調査

このエラーメッセージは、ゲストの XML ファイルの 6 行目にある < 記号の後に、パーサーが新たな要素名を予測していることを示しています。

テキストエディターの行番号の表示が有効になっていることを確認します。XML ファイルを開いて、6 行目のテキストを見つけます。

```
<domain type='kvm'>
  <name>name_of_guest</name>
  <memory>524288</memory>
  <vcpu>2</vcpu><
```

ゲストの XML ファイルのこの抜粋には、ドキュメントに余分な `<` が含まれています。

## 解決法

余分な `<` を削除するか、または新たな要素を終了させます。

### A.20.16.2.2. 未終了の属性

#### 現象

以下のエラーが発生する。

```
error: (name_of_guest.xml):2: Unescaped '<' not allowed in attributes
values
<name>name_of_guest</name>
--^
```

#### 調査

ゲストの XML ファイルのこの抜粋には、未終了要素の属性値が含まれています。

```
<domain type='kvm>
<name>name_of_guest</name>
```

このケースでは、`'kvm'` を閉じる引用符が足りません。XML の開始および終了タグと同様に、引用符やアポストロフィーなどの属性値の文字列は開いたり、閉じたりする必要があります。

## 解決法

すべての属性値の文字列を正しく開いて、閉じます。

### A.20.16.2.3. 開始および終了タグの不一致

#### 現象

以下のエラーが発生する。

```
error: (name_of_guest.xml):61: Opening and ending tag mismatch: clock
line 16 and domain
</domain>
-----^
```

#### 調査

上記のエラーメッセージには、問題となっているタグを特定する 3 つのヒントがあります。

最後のコロンの後のメッセージである **clock line 16 and domain** は、ドキュメントの 16 行目の **<clock>** にタグの不一致があることを示しています。最後のヒントはメッセージのコンテキスト部分にあるポインターで、これは問題となっている 2 つ目のタグを特定しています。

ペアになっていないタグは `</>` で閉じる必要があります。以下の抜粋はこのルールを守っていないので、上記のエラーメッセージが表示されました。

```
<domain type='kvm'>
...
<clock offset='utc'>
```

このエラーはファイル内の XML タグの不一致が原因で発生します。すべての XML タグには、一致する開始および終了タグがなければなりません。

#### XML タグの不一致の他の例

以下の例では同様のエラーメッセージが生成され、XML タグの不一致の例が示されています。

この抜粋には、終了タグ (`</name>`) がいないために `<features>` の不一致エラーが含まれています。

```
<domain type='kvm'>
...
<features>
  <acpi/>
  <pae/>
...
</domain>
```

以下の例では、`</name>` の終了タグのみがあり、対応する開始タグがありません。

```
<domain type='kvm'>
  </name>
...
</domain>
```

#### 解決法

すべての XML タグが正しく開始し、終了していることを確認します。

#### A.20.16.2.4. よくあるタグのエラー

##### 現象

以下のエラーメッセージが表示されます。

```
error: (name_of_guest.xml):1: Specification mandate value for attribute
ty
<domain ty pe='kvm'>
-----^
```

##### 調査

XML エラーは、簡単な誤字脱字で発生します。このエラーメッセージは、XML エラー (このケースでは **type** という単語の中に余分な空白) をポインターで示しています。

```
<domain ty pe='kvm'>
```

以下の XML 例は、特別文字が足りなかったり余分な文字があったりするエラーが理由で正確に解析されません。

```
<domain type 'kvm'>
```

```
<dom#ain type='kvm'>
```

## 解決法

問題のあるタグを特定するには、ファイルのコンテキストのエラーメッセージを読んで、ポインターでエラーを見つけます。XML を修正し、変更を保存します。

### A.20.16.3. 論理および設定エラー

適切にフォーマットされた XML ドキュメントには、構文が正しくても **libvirt** が解析できないエラーが含まれる場合があります。これらのエラーの多くは、以下で説明する 2 つのケースに当てはまります。

#### A.20.16.3.1. 部分的な消滅

##### 現象

ドメインの編集または定義後に、変更箇所が表示されず、その効果も見られない。**define** コマンドや **edit** コマンドは機能するが、XML を再度ダンプすると変更が消えてしまう。

##### 調査

このエラーの原因となり得るのは、構築または構文が破損していて、**libvirt** が解析できないという場合です。**libvirt** は一般的に、認識できる構築のみを探し、他のものはすべて無視するので、**libvirt** が入力を解析した後に XML 変更が消滅する場合があります。

##### 解決法

XML 入力を **edit** コマンドや **define** コマンドに渡す前に確認します。**libvirt** 開発者は、**libvirt** とバンドルされている XML スキーマ式を維持します。これは、**libvirt** が使用する XML ドキュメントで許可されるコンストラクトの大半を定義するものです。

以下のコマンドを使って、**libvirt** XML ファイルを検証します。

```
# virt-xml-validate libvirt.xml
```

このコマンドが渡されると、**libvirt** が XML からのすべてのコンストラクトを理解します。ただし、特定のハイパーバイザーにのみ有効なオプションをスキーマが検出できない場合は例外です。たとえば、**virsh dump** コマンドの結果として **libvirt** が生成した XML は、エラーなしで確認されます。

#### A.20.16.3.2. ドライブデバイスの種類の誤り

##### 現象

CD-ROM 仮想ドライブ用のソースイメージの定義を追加したにもかかわらず、見当たらない。

```
# virsh dumpxml domain
<domain type='kvm'>
  ...
```



```
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw' />
  <target dev='hdc' bus='ide' />
  <readonly />
</disk>
...
</domain>
```

## 解決法

以下のように、見つからない **<source>** パラメーターを追加して XML を訂正します。

```
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw' />
  <source file='/path/to/image.iso' />
  <target dev='hdc' bus='ide' />
  <readonly />
</disk>
```

**type='block'** ディスクデバイスはソースが物理デバイスであることを予想します。イメージファイルのディスクを使用するには、**type='file'** を代わりに使用します。

## 付録B 仮想化の制限

この付録では、Red Hat Enterprise Linux 7 の仮想化パッケージの追加サポートおよび製品の各種制限について説明します。

### B.1. サポート制限

Red Hat Enterprise Linux 7 では、Windows ゲスト仮想マシンは、Advanced Mission Critical (AMC) などの特定のサブスクリプションプログラムにおいてのみサポートされています。お使いのサブスクリプションモデルに Windows ゲストのサポートが含まれるかどうかは不明な場合は、カスタマーサポートにお問い合わせください。

Red Hat Enterprise Linux 7 の Windows ゲスト仮想マシンについての詳細は、[Windows Guest Virtual Machines on Red Hat Enterprise Linux 7 Knowledgebase article](#) を参照してください。

### B.2. KVM の制限

Red Hat Enterprise Linux に含まれるハイパーバイザーパッケージは `qemu-kvm` です。これは、Red Hat Virtualization (RHV) および Red Hat OpenStack (RHOS) 製品に含まれる `qemu-kvm-rhev` パッケージとは異なります。`qemu-kvm` に適用される制限の多くは、`qemu-kvm-rhev` には適用されません。

`qemu-kvm` と `qemu-kvm-rhev` パッケージ間の違いについての詳細は、[What are the differences between qemu-kvm and qemu-kvm-rhev and all sub-packages?](#) を参照してください。

Red Hat Enterprise Linux に含まれる KVM ハイパーバイザーには、以下の制限が適用されます。

#### ゲストあたりの最大仮想 CPU (vCPU) 数

Red Hat Enterprise Linux 7.2 以降では、Red Hat Enterprise Linux 7.0 での 160 に対して、1 ゲストあたり 240 の仮想 CPU をサポートします。

#### 仮想化のネスト

Red Hat Enterprise Linux 7.2 以降ではネスト化した仮想化機能をテクノロジープレビューとして提供しています。この機能により、KVM はハイパーバイザーとして動作するゲストを起動し、それ自体のゲストを作成できるようになります。

#### Tiny Code Generator サポート

QEMU および `libvirt` には、QEMU Tiny Code Generator (TCG) を使用する動的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サポートは不要です。ただし、TCG は Red Hat ではサポートされていません。

仮想マシンでネスト化されたゲストを作成するために `qemu-kvm` パッケージが使用される場合、ネスト化された仮想化が親仮想マシンで有効にされていない限り、これは TCG を使用します。仮想化のネスト化がテクノロジープレビューであることに注意してください。詳細は、[12章 仮想化のネスト](#) を参照してください。

仮想ハードウェアの詳細ビューの「概要」ペインで、`virt-manager` は、KVM または QEMU TCG の仮想マシンのタイプを表示します。仮想ハードウェアの詳細ビューについての詳細は、「[仮想ハードウェアの詳細ウィンドウ](#)」を参照してください。

TCG ベースの仮想マシンは、ドメイン XML ファイルの以下の行でも確認できます。

```
<domain type='qemu'>
```

## 不変タイムスタンプカウンター (TSC) ビット

不変 TSC を搭載していないシステムは、追加設定が必要です。不変 TSC の搭載の有無を確認するには、また関連する問題を解決するための設定手順の詳細は [8章 KVM ゲストのタイミング管理](#) を参照してください。

## エミュレートした SCSI アダプター

SCSI デバイスエミュレーションは、`virtio-scsi` 準仮想化ホストバスアダプター (HBA) でのみサポートされます。Red Hat Enterprise Linux では、エミュレートした SCSI HBA は KVM でサポートされません。

## エミュレートされた IDE デバイス

KVM では、仮想マシン 1 台あたりの仮想化 (エミュレートされた) IDE デバイスは最大 4 つまでに制限されています。

## 準仮想化デバイス

準仮想化デバイスは、VirtIO デバイスとしても知られています。これらは、仮想マシン内で最適に機能するように設計された純粋な仮想デバイスです。

Red Hat Enterprise Linux 7 は、仮想マシンバス 1 つあたりの 32 の PCI デバイススロットと、デバイススロット 1 つあたり 8 つの PCI 機能をサポートします。これにより、マルチファンクション機能が仮想マシンで有効にされ、PCI ブリッジが使用されると、理論上はバスあたり最大 256 の PCI 機能が提供されることになります。それぞれの PCI ブリッジは新規バスを追加し、追加で 256 のデバイスアドレスを有効にする可能性があります。ただし、一部のバスでは、`root` バスがスロットを占有するいくつかの組み込みデバイスを持つ場合など、256 デバイスアドレスすべてをユーザーに対して利用可能にしない場合があります。

デバイスについての詳細は、[17章 ゲスト仮想マシンデバイスの設定](#) を、PCI ブリッジの詳細については、「[PCI ブリッジの作成](#)」を参照してください。

## 移行の制限

デバイス割り当ては、仮想マシンの排他的使用のために、仮想マシンに公開されている物理デバイスを参照します。デバイス割り当てでは、仮想マシンが実行される特定ホスト上のハードウェアを使用するため、デバイス割り当てが使用されている場合には移行および保存/復元はサポートされません。ゲストオペレーティングシステムがホットプラグをサポートしている場合は、移行または保存/復元操作の前に割り当てデバイスを削除してこの機能を有効にできます。

ライブマイグレーションは、CPU タイプが同一のホスト間でのみ可能です (つまり、Intel から Intel、または AMD から AMD のみ)。

ライブマイグレーションの場合、両方のホストが `on` または `off` のいずれかの、NX (No eXecution) ビットの同一値セットを備えている必要があります。

移行を機能させるには、書き込みモードで開かれたすべてのブロックデバイスで `cache=none` を指定しておく必要があります。

**警告**

**cache=none** オプションを加えない場合、ディスクの破損につながる恐れがあります。

## ストレージの制限

ディスクやブロックデバイス全体 (**/dev/sdb** など) への書き込みアクセスをゲスト仮想マシンに与えることには、リスクが伴います。ゲスト仮想マシンにブロックデバイス全体へのアクセスがあると、ホストマシンとボリュームラベルやパーティションテーブルを共有できます。しかし、ホストシステムのパーティション認識コードに不具合があると、セキュリティーリスクが発生する可能性があります。このリスクは、ゲスト仮想マシンに割り当てたデバイスを無視するようにホストマシンを設定することで回避できます。

**警告**

ストレージ制限に従わない場合、セキュリティー関連のリスクが発生する可能性があります。

## ライブスナップショット

Red Hat Enterprise Linux における KVM のバックアップ/リストア API は、ライブスナップショットをサポートしません。

## ストリーミング、ミラーリング、およびライブマージ

ストリーミング、ミラーリング、およびライブマージはサポートされません。これにより、ブロックジョブを防止できます。

## I/O スロットリング

Red Hat Enterprise Linux は、仮想ディスクでの操作についての最大入力/出力レベルの設定をサポートしていません。

## I/O スレッド

Red Hat Enterprise Linux は、VirtIO インターフェースでのディスクの入出力操作についての別々のスレッドの作成をサポートしていません。

## メモリーのホットプラグとホットアンプラグ

Red Hat Enterprise Linux は、仮想マシンからのメモリーのホットプラグまたはホットアンプラグをサポートしていません。

## vhost-user

Red Hat Enterprise Linux は、ユーザー空間 vhost インターフェースの実装をサポートしていません。

## I/O スレッド

Red Hat Enterprise Linux は、仮想マシンからの CPU のホットアンプラグをサポートしていません。

## PCIe の NUMA ゲストの局所性 (ローカリティー)

Red Hat Enterprise Linux は、仮想 PCIe デバイスの特定の NUMA ノードへのバインドをサポートしていません。

## コアダンプの制限

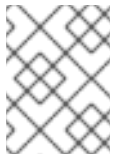
現在コアダンプは移行プロセスの一部として実装されていないため、デバイス割り当てが使用中の場合はサポートされません。

## リアルタイムカーネル

KVM は現在リアルタイムカーネルをサポートしていないため、Red Hat Enterprise Linux for Real Time でこれを使用することはできません。

## IBM Power Systems

Red Hat Enterprise Linux with KVM は AMD64 および Intel 64 のシステムではサポートされていますが IBM Power System ではサポートされていません。現在、IBM Power System に対しては Red Hat Virtualization で POWER8 ベースのソリューションを提供しています。



### 注記

バージョンサポートおよびインストール手順については、[関連するナレッジベースの記事](#)を参照してください。

## B.3. アプリケーションの制限

仮想化には、特定の種類のアプリケーションを不安定にする側面があります。

I/O スループット要件の高いアプリケーションでは、完全仮想化ゲスト用に KVM の準仮想化ドライバー (virtio ドライバー) の使用をお勧めします。virtio ドライバーがないと、特定のアプリケーションは I/O 負荷が高い場合に予想できない動きをする場合があります。

以下のアプリケーションは、I/O 要件が高い場合は回避することをお勧めします。

- **kdump** サーバー
- **netdump** サーバー

I/O を過剰に使用したり、リアルタイムのパフォーマンスが要求されるアプリケーションやツールは、慎重に評価することをお勧めします。I/O パフォーマンスを高めるには、virtio ドライバーまたは PCI デバイス割り当ての使用を検討してください。完全仮想化ゲスト用の virtio ドライバーの詳細は、[5 章 KVM 準仮想化 \(virtio\) ドライバー](#)を参照してください。また、PCI デバイス割り当ての詳細は[17 章 ゲスト仮想マシンデバイスの設定](#)を参照してください。

仮想化環境でアプリケーションを実行すると、パフォーマンスが若干低下します。仮想化でより新しく速いハードウェアに統合することでパフォーマンスにどのような利点をもたらされるかを判断するには、仮想化の使用に関連するアプリケーションの潜在的なパフォーマンス問題を対象にして評価することをお勧めします。

## B.4. その他の制限

仮想化に影響するその他すべての制限および問題点についての一覧は、『Red Hat Enterprise Linux 7 リリースノート』に記載されています。『Red Hat Enterprise Linux 7 リリースノート』では、現時点での新機能と更新または発見された既知の問題および制限が説明されています。

## B.5. ストレージ対応

仮想マシンのサポートされるストレージメソッドは以下になります。

- ローカルストレージ上のファイル
- 物理ディスクパーティション
- ローカル接続の物理 LUN
- LVM パーティション
- NFS 共有ファイルシステム
- iSCSI
- クラスター化した GFS2 ファイルシステム
- ファイバーチャネルベースの LUN
- イーサネット上ファイバーチャネル (FCoE)

## B.6. USB 3 / xHCI サポート

USB 3 (xHCI) USB ホストアダプターエミュレーションは Red Hat Enterprise Linux 7.2 以降で対応しています。あらゆる USB 速度がサポートされています。つまり、どの世代の USB デバイスも xHCI バスにプラグインできます。また (USB 1 デバイスの) コンパニオンコントローラーは一切不要です。ただし、USB 3 バルクストリームはサポートされないことに注意してください。

xHCI の利点:

- 仮想化と互換性があるハードウェア設計になっています。つまり、xHCI エミュレーションでは、ポーリングのオーバーヘッドが削減されたため、以前のバージョンよりも必要な CPU リソースが少なくなっています。
- USB 3 デバイスの USB パススルーを利用できます。

xHCI の制限:

- Red Hat Enterprise Linux 5 ゲストではサポートされません。

xHCI デバイスのドメイン XML のサンプルについては、[図17.19「USB3/xHCI デバイスに使用されるドメイン XML の例」](#)を参照してください。

## 付録C その他のリソース

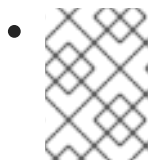
仮想化および Red Hat Enterprise Linux に関する詳細については次のリソースを参照してください。

### C.1. オンラインリソース

- <http://www.libvirt.org/> は **libvirt** 仮想化 API の公式のアップストリーム web サイトになります。
- <https://virt-manager.org/> は、仮想マシン管理用のグラフィカルなアプリケーション、**仮想マシンマネージャー (virt-manager)** のプロジェクト web サイトになります。
- Red Hat Virtualization - <http://www.redhat.com/products/cloud-computing/virtualization/>
- Red Hat 製品ドキュメンテーション - <https://access.redhat.com/documentation/en/>
- 仮想化テクノロジーの概要 - <http://virt.kernelnewbies.org>

### C.2. インストールされているドキュメント

- **man virsh** および **/usr/share/doc/libvirt-version-number** – 仮想マシン管理ユーティリティのサブコマンドおよびオプションが記載されているほか、**libvirt** 仮想化ライブラリー API に関して総合的に解説しています。
- **/usr/share/doc/gnome-applet-vm-version-number** – ローカルに実行している仮想マシンの監視および管理を行う **GNOME** のグラフィカルなパネルアプレットに関する記載です。
- **/usr/share/doc/libvirt-python-version-number** – **libvirt** ライブラリーの Python バインディングに関して詳細に説明されています。**libvirt-python** パッケージを使用することで、**python** の開発者は **libvirt** 仮想化管理ライブラリーとのインターフェースとなるプログラムを作成できるようになります。
- **/usr/share/doc/virt-install-version-number** – **virt-install** コマンドに関するドキュメントです。仮想マシン内で **Fedora** や **Red Hat Enterprise Linux** 関連のディストリビューションのインストールを開始する場合に役立ちます。
- **/usr/share/doc/virt-manager-version-number** – 仮想マシンマネージャーに関するドキュメントです。仮想マシンの管理に使用できるグラフィカルなツールについて解説されています。



#### 注記

他の Red Hat Enterprise Linux コンポーネントに関する詳細は、該当する **man** ページまたは **usr/share/doc/** のファイルを参照してください。



## 付録D IOMMU グループの使用<sup>[1]</sup>

Red Hat Enterprise Linux 7 で導入された **VFIO** (Virtual Function I/O) は、ユーザー空間のドライバフレームワークを提供する一連の Linux カーネルモジュールです。このフレームワークにより、ユーザー空間ドライバのセキュアなデバイスアクセスを有効にするために **IOMMU** (Input-output memory management unit) 保護が使用されます。VFIO は、**DPDK** (Data Plane Development Kit) やより一般的な **PCI デバイス割り当て**などのユーザー空間ドライバを有効にします。

VFIO は IOMMU グループを使用してデバイスを分離し、ホストとゲストの機能に影響を与えかねない同じホスト物理マシン上で実行される 2 つのデバイス間の意図しない **DMA** (Direct Memory Access/直接メモリアクセス) を防ぎます。Red Hat Enterprise Linux 7 で利用できる IOMMU グループは、Red Hat Enterprise Linux 6 で利用可能なレガシー KVM デバイス割り当てと比較すると大幅な改善をもたらします。この付録ではとくに以下の点について説明します。

- IOMMU グループの概要
- デバイス分離の重要性
- VFIO の利点

### D.1. IOMMU の概要

IOMMU はデバイスの仮想アドレス空間を作成します。ここで、各 **I/O Virtual Address (IOVA)** は物理システムメモリの異なるアドレスに変換されます。この変換が完了すると、デバイスは物理システムのメモリー内の異なるアドレスに接続されます。IOMMU がいない場合、すべてのデバイスは、メモリアドレス変換がないために物理メモリの共有されたフラットビューを持ちます。IOMMU がある場合、デバイスは、デバイスの割り当てに役立つ新規のアドレス空間である **IOVA 空間**を受信します。

IOMMU が異なると機能レベルも異なります。過去において、IOMMU は変換のみを提供することに制限され、アドレス空間の小規模な枠にのみ使用されることが多くありました。たとえば、IOMMU は複数デバイスで共有されるローメモリー (**low memory**) の **IOVA 空間**の小規模な枠 (**1GB 以下**)のみを予約しました。**AMD GART** (**graphics address remapping table**) が汎用 IOMMU として使用される場合がこのモデルの例になります。これらの従来の IOMMU は、**バウンスバッファ** (**bounce buffer**) および **アドレスコアレスシング** (**address coalescing**) という 2 つの機能を主に提供してきました。

- **バウンスバッファ**は、デバイスのアドレス機能がプラットフォームの同機能よりも小さい場合に必要になります。たとえば、デバイスのアドレス空間がメモリーの **4GB (32 ビット)** に制限され、ドライバが **4GB** を超えるバッファに割り当てようとする場合に、デバイスはバッファに直接アクセスできなくなります。このような状況ではバウンスバッファを使用する必要があります。ローメモリー (**low memory**) にあるバッファ領域では、デバイスが **DMA 操作**を実行できます。バッファのデータは操作の完了時にドライバの割り当て済みバッファにのみコピーされます。つまり、バッファはローメモリー (**low memory**) アドレスからハイメモリー (**high memory**) アドレスにバウンスされます。IOMMU は、デバイスのアドレス空間内で **IOVA 変換**を提供してバウンスバッファを防止します。これにより、デバイスは、バッファがデバイスの物理アドレス空間を超えても **DMA 操作**を直接バッファで実行します。これまでこの IOMMU 機能は、IOMMU での使用に制限されてきましたが、**PCI-Express (PCIe)** の導入により、**4GB** を超えるアドレスのサポート機能がレガシー以外のすべてのエンドポイントで必要になりました。
- 従来のメモリー割り当てでは、メモリーのブロックはアプリケーションのニーズに応じて割り当てや解放が実行されます。この方法を使用することにより、物理アドレス空間全体に分散するメモリーギャップが生成されます。メモリーギャップの**コアレスシング**、つまり簡単に言えばメモリーギャップが1つにまとめられることにより、メモリーをより効率的に使用できるようになることが望ましいと言えます。IOMMU は、**scatter-gather** 一覧と呼ばれることもある分散したメモリーの一覧の**コアレスシング**を **IOVA 空間全体**で実行します。これにより、IOMMU は連続的な **DMA 操作**を作成し、最終的には **I/O パフォーマンス**の効率を高めます。最も簡単



な例として、ドライバーは物理メモリー領域の連続しない2つの4KBバッファを割り当てることができます。IOMMUは、これらのバッファの連続する範囲を割り当て、I/O デバイスが2つの4 KB DMAではなく単一の8KB DMAを実行できるようにします。

メモリーコアレスシングおよびバウンスバッファはホストの高パフォーマンス I/O において重要ですが、仮想化環境に不可欠な IOMMU 機能は最新 IOMMU の分離機能です。従来の PCI は要求側のデバイスの ID (リクエスト ID) でトランザクションにタグ付けしないため、PCI-Express 以前には分離を広範囲に実行することはできませんでした。PCI-X にはある程度のリクエスト ID が含まれていましたが、デバイスを相互に接続し、トランザクションの所有権を持たせるルールでは、デバイスを分離するための完全なサポートは提供されませんでした。

PCIe では、各デバイスのトランザクションは、デバイスに固有のリクエスト ID (BDF と省略されることの多い PCI バス/デバイス/機能番号) でタグ付けされ、これはデバイスの固有の IOVA テーブルを参照するために使用されます。分離が可能になることにより、IOVA 空間は接続できないメモリーのオフロードやメモリーコアレスシングなどの変換操作のみならず、デバイスから DMA アクセスを制限するためにも使用できるようになりました。これにより、デバイスを相互に分離し、メモリー領域の重複した割り当てを防ぐことができます。これは適切なゲスト仮想マシンデバイス管理に不可欠です。これらの機能をゲスト仮想マシンで使用するにより、割り当て済みデバイスの IOVA 空間には仮想マシンについてのゲスト物理およびホスト物理メモリーのマッピングが設定されます。これがいったん実行されると、デバイスはゲスト仮想マシンのアドレス空間で DMA 操作を透過的に実行します。

## D.2. IOMMU グループの詳細

IOMMU グループは、IOMMU の観点から分離されているとみなされる最も小さなデバイスセットとして定義されます。分離を実行するために必要な最初の手順は、詳細度 (granularity) の使用です。IOMMU がデバイスを別個の IOVA 空間に区分できない場合、それらは分離されません。たとえば、複数のデバイスが同一 IOVA 空間のエイリアスになる場合、IOMMU はそれらを区別することができません。これは、通常の x86 PC がすべての従来型の PCI デバイスに同一リクエスト ID に対するエイリアスを設定してそれらを1つにまとめる方法になります (PCIe-to-PCI ブリッジ)。レガシー KVM デバイスの割り当てでは、ユーザーはこれらの従来型 PCI デバイスを別々に割り当てるため、IOMMU はデバイス間を区別できずに設定は失敗します。VFIO は IOMMU グループで管理されるため、VFIO は IOMMU の詳細度の使用というこの最も基本的な要件を満たさない設定を許可しません。

次の手順として、デバイスからのトランザクションが IOMMU に実際に到達するかどうかを判別する必要があります。PCIe 仕様では、トランザクションを相互接続ファブリック内に再度ルート指定できます。PCIe ダウンストリームポートは、あるダウンストリームデバイスから別のデバイスへのトランザクションを再度ルート指定できます。PCIe スイッチのダウンストリームポートは相互に接続し、1つのポートから別のポートへの再ルート指定を可能にします。マルチファンクションエンドポイントデバイス内でも、1つの機能からのトランザクションは別のファンクションに直接送信できます。これらの1つのデバイスから別のデバイスへのトランザクションはピアツーピアトランザクションと呼ばれ、別々の IOV 空間で稼働しているデバイスの分離を破棄する可能性があります。たとえば、ゲスト仮想マシンに割り当てられているネットワークインターフェースカードが DMA の書き込み操作を独自の IOV 空間内の仮想アドレスに試行するとします。ただし、物理空間ではその同じアドレスはホストによって所有されているピアディスクコントローラーに属します。この場合、デバイスについての物理変換に対する IOVA は IOMMU のみで実行されるため、トランザクションのデータパスの最適化を試行する相互接続により、ディスクへの DMA 書き込み操作が、変換用に IOMMU に到達する前に間違っリダイレクトされる可能性があります。

この問題を解決するために、PCI Express 仕様には、これらのリダイレクトの可視性とコントロールを提供する PCIe Access Control Services (ACS) のサポートが含まれます。これは相互接続やマルチファンクションエンドポイントに欠落していることの多い、デバイスを相互に分離するために必要なコンポーネントです。デバイスから IOMMU へのすべてのレベルで ACS サポートがない場合も、リダイレクトが生じることを想定する必要があります。このため、ACS サポートがない場合、PCI トポロジーのすべてのデバイスの分離に影響があります。PCI 環境の IOMMU グループはこの分離を考慮に入れて、変換されないピアツーピア DMA が可能な複数のデバイスを1つにグループ化します。

要約すると、IOMMU グループは、IOMMU が可視性を持ち、他のグループから分離される最小単位のデバイスセットを表します。VFIO はこの情報を使用してユーザー空間についてデバイスの安全な所有権を確保します。ブリッジ、root ポートおよびスイッチ (相互接続ファブリックの例すべて) の例外を除き、IOMMU グループ内のすべてのデバイスは VFIO デバイスドライバーにバインドされるか、または安全なスタブドライバーとして認識されます。PCI の場合、これらのドライバーは **vfio-pci** と **pci-stub** です。**pci-stub** は、ホストがこのドライバー経由でデバイスと対話しないと認識されているので許可されます<sup>[2]</sup>。VFIO の実行時にグループが実行不能であることを示すエラーが出される場合、このグループ内のすべてのデバイスが適切なホストドライバーにバインドする必要があることを意味します。**virsh nodedev-dumpxml** を使用して、IOMMU グループの構成を公開し、**virsh nodedev-detach** を使用してデバイスを VFIO と互換性のあるドライバーにバインドすることにより、このような問題を解決できます。

### D.3. IOMMU グループの特定および割り当て方法

この例では、ターゲットシステムにある PCI デバイスを特定し、割り当てる方法を示しています。追加の例および情報については、「[GPU デバイスの割り当て](#)」を参照してください。

#### 手順D.1 IOMMU グループ

##### 1. デバイスの一覧表示

**virsh nodedev-list device-type** コマンドを実行することにより、システムのデバイスを特定します。この例では、PCI デバイスを見つける方法を説明します。出力は簡潔にするために変更されています。

```
# virsh nodedev-list pci

pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
[...]
pci_0000_00_1c_0
pci_0000_00_1c_4
[...]
pci_0000_01_00_0
pci_0000_01_00_1
[...]
pci_0000_03_00_0
pci_0000_03_00_1
pci_0000_04_00_0
pci_0000_05_00_0
pci_0000_06_0d_0
```

##### 2. デバイスの IOMMU グループの特定

一覧表示される各デバイスについては、**virsh nodedev-dumpxml name-of-device** コマンドを使用すると、IOMMU グループを含むデバイスについての追加情報を参照できます。たとえば、IOMMU グループで **pci\_0000\_04\_00\_0** という名前の PCI デバイス (PCI アドレス **0000:04:00.0**) を検索する場合、以下のコマンドを使用します。

```
# virsh nodedev-dumpxml pci_0000_04_00_0
```

このコマンドは、以下に示すのと同様の XML ダンプを生成します。

```

<device>
  <name>pci_0000_04_00_0</name>
  <path>/sys/devices/pci0000:00/0000:00:1c.0/0000:04:00.0</path>
  <parent>pci_0000_00_1c_0</parent>
  <capability type='pci'>
    <domain>0</domain>
    <bus>4</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10d3'>82574L Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='8'>    <!--This is the element block you
will need to use-->
      <address domain='0x0000' bus='0x00' slot='0x1c'
function='0x0' />
      <address domain='0x0000' bus='0x00' slot='0x1c'
function='0x4' />
      <address domain='0x0000' bus='0x04' slot='0x00'
function='0x0' />
      <address domain='0x0000' bus='0x05' slot='0x00'
function='0x0' />
    </iommuGroup>
    <pci-express>
      <link validity='cap' port='0' speed='2.5' width='1' />
      <link validity='sta' speed='2.5' width='1' />
    </pci-express>
  </capability>
</device>

```

## 図D.1 IOMMU グループ XML

### 3. PCI データの表示

上記で収集される出力では、4つのデバイスを含む1つのIOMMUグループがあります。これは、ACS サポートのないマルチファンクション PCIe root ポートの例です。スロット 0x1c の2つの機能は PCIe root ポートであり、これらは (pciutils パッケージにある) **lspci** コマンドを実行することによって特定できます。

```

# lspci -s 1c

00:1c.0 PCI bridge: Intel Corporation 82801JI (ICH10 Family) PCI
Express Root Port 1
00:1c.4 PCI bridge: Intel Corporation 82801JI (ICH10 Family) PCI
Express Root Port 5

```

エンドデバイスであるバス 0x04 および 0x05 の2つの PCIe デバイスについてこの手順を繰り返します。

```

# lspci -s 4
04:00.0 Ethernet controller: Intel Corporation 82574L Gigabit
Network Connection これは次のステップで使用され、04:00.0 と呼ばれます。
# lspci -s 5 これは次のステップで使用され、05:00.0 と呼ばれます。
05:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5755
Gigabit Ethernet PCI Express (rev 02)

```

#### 4. エンドポイントのゲスト仮想マシンへの割り当て

エンドポイントのいずれかを仮想マシンに割り当てるには、現時点で割り当てていないエンドポイントを VFIO 互換ドライバーにバインドして、IOMMU グループがユーザーおよびホストドライバー間で分離されないようにします。たとえば、上記で受信される出力では **04:00.0** のみを持つ仮想マシンを設定しようとしませんが、マシンは **05:00.0** がホストドライバーから分離されない限り、起動に失敗します。**05:00.0** の割り当てを解除するには、**root** で **virsh nodedev-detach** コマンドを実行します。

```
# virsh nodedev-detach pci_0000_05_00_0
Device pci_0000_05_00_0 detached
```

両方のエンドポイントを仮想マシンに割り当てることは、この問題を解決するための別のオプションになります。libvirt は、**<hostdev>** 要素内の **managed** 属性に **yes** 値を使用する場合には、割り当てられたデバイスに対してこの操作を自動的に実行します。例: **<hostdev mode='subsystem' type='pci' managed='yes'>**。詳細については、[注記](#) を参照してください。

#### 注記

libvirt には、PCI デバイスを処理するための 2 つの方法があります。それらは管理対象にも非管理対象にもなります。これは、**<hostdev>** 要素内の **managed** 属性に指定される値によって決まります。デバイスが管理対象になる場合、libvirt は既存ドライバーからデバイスの割り当てを自動的に解除し、起動時にこれを **vfio-pci** にバインドしてこれを仮想マシンに割り当てます (仮想マシンの場合)。仮想マシンがシャットダウンされるか、または削除される場合、または PCI デバイスの割り当てが仮想マシンから解除される場合、libvirt はデバイスを **vfio-pci** のバインドから解除し、これを元のドライバーに再びバインドします。デバイスが非管理対象の場合、libvirt はこのプロセスを自動化せず、ここで説明されている管理的な側面すべてが実行されていることを、デバイスの仮想マシンへの割り当て前に各自で確認する必要があります。デバイスが仮想マシンで使用されなくなった後には、デバイスの割り当てを再度実行する必要もあります。管理されていないデバイスでこれらを実行しないと、仮想マシンは失敗します。そのため、libvirt でデバイスを管理するのがより容易な方法と言えます。

### D.4. IOMMU ストラテジーおよびユースケース

必要以上のデバイスが含まれる IOMMU グループを処理する方法は多数あります。プラグインカードの場合、最初のオプションとして、カードを異なるスロットにインストールして必要なグループが生成されるかどうかを判別します。通常の Intel チップセットでは、PCIe ルートポートはプロセッサと PCH (Platform Controller Hub) の両方で提供されます。これらのルートポートの各種機能はそれぞれ非常に異なる場合があります。Red Hat Enterprise Linux 7 には、PCH ルートポートの多くにネイティブの PCIe ACS サポートがない場合でも、数多くの PCH ルートポートの分離の公開をサポートします。そのため、これらのルートポートは小規模な IOMMU グループを作成する上での適切なターゲットになります。Intel® Xeon® class プロセッサ (E5 シリーズ以降) および「High End Desktop Processor」の場合、通常プロセッサベースの PCIe ルートポートは PCIe ACS のネイティブサポートを提供しますが、Core™ i3、i5、および i7 および Xeon E3 プロセッサなどのローエンドのクライアントプロセッサはこれを提供しません。これらのシステムの場合、通常 PCH ルートポートは最も柔軟な分離設定を提供します。

別のオプションとして、ハードウェアベンダーと連携し、分離が存在するかどうかを判別し、カーネルにこの分離を認識させる方法もあります。これは通常は、機能間の内部のピアツーピアが可能かどうか、またはダウンストリームポートの場合であればダイレクトが可能かどうかを判別することに関係します。Red Hat Enterprise Linux 7 カーネルには、これらのデバイスに対応する代替方法が数多く含まれます。Red Hat カスタマーサポートはハードウェアと連携して ACS 対応の分離が可能かどうか、および類似のデバイス固有の方法をカーネルに組み込んでこの分離を公開する最善の方法を判別するお

手伝いをいたします。ハードウェアベンダーについては、ピアツーピアをサポートしないマルチファンクションエンドポイントが設定領域の単一の静的 ACS テーブルを使用してこれを公開しますが、各種機能は公開しないことに注意してください。この機能をハードウェアに追加すると、カーネルは機能を分離されているものと自動的に検知するので、ハードウェアのすべてのユーザーに関わる問題を排除できます。

上記の提案を実行できない場合には、共通の対応として、カーネルがユーザーが指定する特定デバイスまたは特定タイプのデバイスについての分離チェックを無効にするオプションを提供するはずです。以前のテクノロジーにより分離がこの程度まで実行されず、すべてが「正常に機能」しているかのような状態で引数が作成されることがよくあります。しかし、これらの分離機能をバイパスすると環境がサポート不可能になります。この分離が存在することを認識しないとは、デバイスが実際に分離されているかどうかを認識していないことを意味するので、障害が発生する前に分離について確認するのが最善の策と言えます。デバイスの分離機能におけるギャップをトリガーすることは極めて困難な場合があり、原因追求のためにデバイスの分離を追跡するのはさらに困難です。VFIO の主要なジョブは、ユーザー所有のデバイスからホストカーネルを保護することであり、IOMMU グループは VFIO が分離を確保するために使用するメカニズムです。

要約すると、IOMMU グループの上部にビルドされる VFIO は、レガシー KVM デバイス割り当てによって実行されたデバイス間のセキュリティー保護と分離のレベルをさらに強化します。この分離は Linux カーネルのレベルで実行されるようになったため、カーネルはそれ自体を保護すると共に、ユーザーにとって危険な設定を防ぐことができます。さらに、ハードウェアベンダーは、マルチファンクションエンドポイントデバイスだけではなく、チップセットおよび相互接続デバイスで PCIe ACS サポートに対応するように奨励されています。このサポートのない既存デバイスの場合、Red Hat はハードウェアベンダーと連携して、分離が可能かどうかを判別し、Linux カーネルサポートを追加してこの分離を公開する場合があります。

---

[1] この付録の元の内容はプリンシパルソフトウェアエンジニアの Alex Williamson によって提供されました。

[2] 例外となるのは、レガシー KVM デバイス割り当てであり、pci-stub ドライバーにバインドされている場合にデバイスとの対話が行われることがよくあります。Red Hat Enterprise Linux 7 にはレガシー KVM デバイス割り当てが含まれないため、この相互対話と潜在的な衝突は回避されます。そのため、同一 IOMMU グループ内で VFIO とレガシー KVM デバイス割り当てを同時に使用することは推奨されていません。

## 付録E 改訂履歴

<b>改訂 2-37.2</b> 翻訳ファイルを XML ソースバージョン 2-37 と同期	<b>Mon Mar 5 2018</b>	<b>Terry Chuang</b>
<b>改訂 2-37.1</b> 翻訳ファイルを XML ソースバージョン 2-37 と同期	<b>Wed Nov 22 2017</b>	<b>Terry Chuang</b>
<b>改訂 2-37</b> 7.4 GA 公開用の各種のコンテンツの更新	<b>Mon Sep 26 2017</b>	<b>Jiri Herrmann</b>
<b>改訂 2-35</b> 7.4 GA 公開用バージョン	<b>Thu Aug 01 2017</b>	<b>Jiri Herrmann</b>
<b>改訂 2-29</b> 7.3 GA 公開用バージョン	<b>Mon Oct 17 2016</b>	<b>Jiri Herrmann</b>
<b>改訂 2-24</b> 複数の問題の修正後の本ガイドの再発行	<b>Thu Dec 17 2015</b>	<b>Laura Novich</b>
<b>改訂 2-23</b> 本ガイドの再発行	<b>Sun Nov 22 2015</b>	<b>Laura Novich</b>
<b>改訂 2-21</b> 7.2 についての内容の複数の更新	<b>Thu Nov 12 2015</b>	<b>Laura Novich</b>
<b>改訂 2-19</b> 改訂履歴の整理	<b>Thu Oct 08 2015</b>	<b>Jiri Herrmann</b>
<b>改訂 2-17</b> 7.2 ペータリリリース用の更新	<b>Thu Aug 27 2015</b>	<b>Dayle Parker</b>