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

Laura Novich
Scott Radvan
Dayle Parker
Tahlia Richardson

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

Laura Novich
Red Hat Customer Content Services
lnovich@redhat.com

Scott Radvan
Red Hat Customer Content Services
sradvan@redhat.com

Dayle Parker
Red Hat Customer Content Services
dayleparker@redhat.com

Tahlia Richardson
Red Hat Customer Content Services
trichard@redhat.com

法律上の通知

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、複数の異なるディストリビューションで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。

目次

パート I. デプロイメント

第1章 システム要件

この章では、Red Hat Enterprise Linux 7 上の VM と呼ばれる仮想マシンを正常に実行するためのシステム要件を示します。仮想化は、Intel 64 および AMD64 アーキテクチャー上の Red Hat Enterprise Linux 7 で利用可能です。

KVM ハイパーバイザーは、Red Hat Enterprise Linux 7 で提供されています。

仮想化パッケージのインストールに関する詳細は、 *4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。

最小システム要件

- 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

- ゲスト仮想マシン内の最大数の仮想化 CPU に 1 つのプロセッサーコアまたはハイパースレッド、およびホストに 1 つのプロセッサーコアまたはハイパースレッド
- 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、および各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が、各ゲストに必要な追加のストレージ領域はイメージ形式によって異なります。

  raw イメージを使用するゲスト仮想マシンでは、ゲストが必要とする領域の合計 **(raw 形式の合計)** は、ゲストの raw イメージファイル **(images)** で必要な領域、ホストオペレーティングシステム **(host)** に必要な 6GB の領域、およびゲストが必要とするスワップ領域**(swap)** 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で必要な領域の計算**
  >
  > total for raw format = images + host + 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で必要な領域の計算**
  >
  > total for qcow format = (expected maximum guest storage * 1.01) + host + swap

ゲスト仮想マシン要件は、さらに 11章*KVM でのオーバーコミット* で説明されています。

1.1. スワップ領域の計算

スワップ領域を使用することで、利用可能な物理メモリー以外のメモリーを追加で提供することが可能になります。メモリーの性能 (速度) を上げるために使用率の低いメモリーをハードドライブにスワップする際に使用するのがスワップパーティションです。スワップパーティションのデフォルトサイズはホストの物理 RAM から計算します。

Red Hat ナレッジベースには、スワップパーティションのサイズを安全かつ効率的に確定する方法についての記事が掲載されています (https://access.redhat.com/site/solutions/15244)。

1.2. KVM 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 拡張機能を備えた Intel プロセッサー
- AMD-V および AMD64 拡張機能を備えたAMD プロセッサー

ご使用のプロセッサーが仮想化拡張機能を備えているかを判別するには、「仮想化拡張機能の確認」 を参照してください。

1.3. ストレージ対応

ゲスト仮想マシンのストレージ方法は、以下の通りです。

- 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)

1.4. 仮想化拡張機能の確認

ご使用のシステムにハードウェアの仮想化拡張機能が備わっているかどうかを確認するには、このセクションを使用してください。完全仮想化の場合には仮想化拡張機能 (Intel VT-x または AMD-V) が必要になります。

1. 次のコマンドを実行して利用できる 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。

- 以下の出力には **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 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 xtpr lahf_lm
```

- 次の出力には **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 fxsr_opt lm 3dnowext 3dnow
pni cx16
lahf_lm cmp_legacy svm cr8legacy ts fid vid ttp tm stc
```

いずれかの出力が表示されていれば、そのプロセッサーにはハードウェアの仮想化拡張機能が備わっていることになります。ただし、メーカー側が BIOS で仮想化拡張機能を無効にしていることがあります。

システム上の各ハイパースレッド、コア、または CPU に対して「**flags:**」出力の内容が複数回表示されることがあります。

仮想化拡張機能は BIOS で無効にされている場合があります。拡張機能が表示されないか、または完全仮想化が機能しない場合は、手順A.3「BIOS 内で仮想化拡張を有効にする」を参照してください。

3. **KVM サブシステムがロードされていることを確認します。**

   追加のチェック事項として、**kvm** モジュールがカーネルにロードさ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ロードされていることを示すため、システムが要件を満たしていることになります。

注記

libvirt パッケージをインストールしている場合は、**virsh** コマンドは仮想化システムの機能の全一覧を出力することができます。root で **virsh capabilities** を実行して全一覧を表示させます。

第2章 KVM ゲスト仮想マシンの互換性

お使いのプロセッサーが仮想化拡張機能に対応しているかどうかを確認し、仮想化拡張機能を有効にする方法についての情報を参照するには、「仮想化拡張機能の確認」をご覧ください。

2.1. Red Hat Enterprise Linux 7 におけるサポート制限

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

Red Hat Enterprise Linux のプロセッサー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/articles/rhel-kvm-limits を参照してください。

対応しているオペレーティングシステムとホストおよびゲストの組み合わせの詳細は、以下の URL をご覧ください。

- https://access.redhat.com/certified-hypervisors

2.2. 対応 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。

注記

対応 CPU モデルの詳細一覧は、「ゲスト CPU モデルの一覧表示」にあるように**`virsh cpu-models`** コマンドを使用して確認することもできます。追加情報は「CPU モデルおよびトポロジー」にも記載されています。ホストのモデルは、必要に応じて特定の機能セット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。

2.2.1. ゲスト CPU モデルの一覧表示

アーキテクチャーの特定の種類に対応する CPU を確認するには、**`virsh cpu-models <arch>`** 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
```

対応 CPU モデルおよび各種機能の詳細一覧を表示するには、該当情報を含む XML ファイルを開く必要があります。このファイルのタイトルは **cpu_map.xml** で、**/usr/share/libvirt/** にあります。内容を確認するには、**cat /usr/share/libvirt/cpu_map.xml** を実行します。ゲストの CPU モデルおよび各種機能を変更するには、ドメイン XML ファイルの **<cpu>** セクションを参照する必要があります (「CPU モデルおよびトポロジー」を参照してください)。

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

gedit の代わりに他のエディターを使用することもできます。開かれるファイルは図2.1「cpu_map.xml ファイルの部分的な内容」のようになりますが、内容はこれよりもはるかに長くなります。

```
<!-- This is only a partial file, only containing the CPU models. The
XML file has more information (including supported features per model)
which you can see when you open the file yourself -->
<cpus>
  <arch name='x86'>
...

    <!-- Intel-based QEMU generic CPU models -->
    <model name='pentium'>
      <model name='486'/>
     </model>

    <model name='pentium2'>
      <model name='pentium'/>
    </model>

    <model name='pentium3'>
```

```
      <model name='pentium2'/>
    </model>

    <model name='pentiumpro'>
    </model>

    <model name='coreduo'>
      <model name='pentiumpro'/>
      <vendor name='Intel'/>
    </model>

    <model name='n270'>
      <model name='coreduo'/>
    </model>

    <model name='core2duo'>
      <model name='n270'/>
    </model>

    <!-- Generic QEMU CPU models -->
    <model name='qemu32'>
      <model name='pentiumpro'/>
    </model>

    <model name='kvm32'>
      <model name='qemu32'/>
    </model>

    <model name='cpu64-rhel5'>
      <model name='kvm32'/>
    </model>

    <model name='cpu64-rhel6'>
      <model name='cpu64-rhel5'/>
    </model>

    <model name='kvm64'>
      <model name='cpu64-rhel5'/>
    </model>

    <model name='qemu64'>
      <model name='kvm64'/>
    </model>

    <!-- Intel CPU models -->
    <model name='Conroe'>
      <model name='pentiumpro'/>
      <vendor name='Intel'/>
    </model>

    <model name='Penryn'>
      <model name='Conroe'/>
    </model>

    <model name='Nehalem'>
      <model name='Penryn'/>
```

```
    </model>

    <model name='Westmere'>
      <model name='Nehalem'/>
      <feature name='aes'/>
    </model>

    <model name='SandyBridge'>
      <model name='Westmere'/>
    </model>

    <model name='Haswell'>
      <model name='SandyBridge'/>
    </model>

    <!-- AMD CPUs -->
    <model name='athlon'>
      <model name='pentiumpro'/>
      <vendor name='AMD'/>
     </model>

    <model name='phenom'>
      <model name='cpu64-rhel5'/>
      <vendor name='AMD'/>
    </model>

    <model name='Opteron_G1'>
      <model name='cpu64-rhel5'/>
      <vendor name='AMD'/>
    </model>

    <model name='Opteron_G2'>
      <model name='Opteron_G1'/>
    </model>

    <model name='Opteron_G3'>
      <model name='Opteron_G2'/>
    </model>

    <model name='Opteron_G4'>
      <model name='Opteron_G2'/>
    </model>

    <model name='Opteron_G5'>
      <model name='Opteron_G4'/>
    </model>
  </arch>
</cpus>
```

図2.1 cpu_map.xml ファイルの部分的な内容

2.3. 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 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。

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:

- Windows 8 よりも前の Windows ゲストと一緒に使用できません。
- Red Hat Enterprise Linux 5 ゲストには対応しません。

XHCI デバイスのドメイン XML のサンプルについては、図20.15「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。

第3章 仮想化の制限

この章では、Red Hat Enterprise Linux 7 の仮想化パッケージにおける追加サポートと製品の制限について説明します。

3.1. KVM の制限

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 vCPU 数

Red Hat Enterprise Linux 7.0 では、ゲスト仮想マシンは最大 160 個の仮想 CPU に対応します。Red Hat Enterprise Linux 7.1 のリリース後は、この制限が 240 に引き上げられます。

仮想化のネスト

Red Hat Enterprise Linux 7 では、仮想化のネストがデフォルトで無効にされています。Red Hat は、ゲスト仮想マシン内でゲスト仮想マシンを起動することに対応していません。

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 12章*KVM ゲストのタイミング管理* を参照してください。

メモリーのオーバーコミット

KVM は メモリーのオーバーコミットをサポートし、ゲスト仮想マシンのメモリーをスワップに保存できます。メモリーのスワップが頻繁に行われると仮想マシンの実行速度が遅くなります。スワップパーティションのサイズを安全かつ効率的に決定する方法は、Red Hat ナレッジベースの https://access.redhat.com/site/solutions/15244 に記載されています。KSM がメモリーのオーバーコミットに使用される場合は、スワップサイズがこの記事で説明されている推奨サイズに従っていることを確認してください。

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 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はサポートされません。

CPU のオーバーコミット

物理プロセッサーコア 1 つあたりに 11 個以上の仮想 CPU を割り当てることは推奨されません。CPU オーバーコミット率の決定するにはキャパシティープラニングツールを使用することが推奨されます。割合はそれぞれのワークロードによって大きく異なるため、理想的な割合を見積もることは容易ではありません。たとえば、あるユースケースではゲスト仮想マシンが CPU を 100% 消費しますが、別のケースでは複数のゲストが完全にアイドル状態である場合もあります。

Red Hat は、システム上に存在する物理コアの全体数よりも多くの vCPU を単一ゲストで実行することをサポートしていません。ハイパースレッドはコアとみなすことはできますが、そのパフォーマンスは使用シナリオによって異なるため、通常のコアと同等のパフォーマンスを期待することはできません。

CPU のオーバーコミットに関するヒントや推奨事項については、「仮想化 CPU のオーバーコミット」を参照してください。

仮想化 SCSI デバイス

Red Hat Enterprise Linux の KVM では、SCSI エミュレーションはサポートされていません。

仮想化 IDE デバイス

KVM では、ゲスト仮想マシン 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 1 台あたり 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。つまりマルチファンクション機能が有効にされ、PCI ブリッジが使用されると、理論上はゲストあたり最大 256 の PCI 機能が提供されることになります。PCI ブリッジの詳細は、「PCI ブリッジの作成」を、およびデバイスについての詳細は、*20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。

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、デバイス割り当ての使用中の移行および保存/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。

SR-IOV の制限

SR-IOV は、以下のデバイスとのみ十分なテストが行われています (他の SR-IOV デバイスも機能する可能性がありますが、リリース時のテストは実施されていません)。

- 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)

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。

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 PCI-e 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。

パラレル/レガシー PCI では、PCI ブリッジの内側の単一デバイスのみがサポートされています。

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、 PCI 設定領域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。

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には、割り込み再マッピングのプラットフォームサポートが必要です。このサポートがないと、ホストが悪意のあるゲストからの割り込みインジェクション攻撃に対して脆弱になる可能性があります。ゲストが信頼されている環境では、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依然として許可することを選択するかもしれません。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、同じことを実行するために sysfs エントリーを動的に使用することもできます。

```
# echo 1 >
/sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts
```

3.2. 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期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準仮想化ドライバーの詳細は、9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20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が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。

3.3. 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。

第4章 仮想化パッケージのインストール

仮想化を使用する前に、コンピューターに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 を使用してホスト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。

4.1. 仮想化ホストインストールの設定

このセクションでは、新たな Red Hat Enterprise Linux インストールの一部としての仮想化ツールおよび仮想化パッケージのインストールについて説明します。

注記

https://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ からご利用いただける『Red Hat Enterprise Linux インストールガイド』は、Red Hat Enterprise Linux のインストールについて詳細に説明しています。

手順4.1 仮想化パッケージグループのインストール

1. **Red Hat Enterprise Linux 7 インストールプログラムを起動します。**

   Red Hat Enterprise Linux インストール CD-ROM、DVD または PXE から対話形式の Red Hat Enterprise Linux 7 インストールを開始します。

2. **ソフトウェアの選択までインストールを進めます**

   ソフトウェア選択のステップまでの他のステップを完了します。**インストール概要** 画面は、対応の必要なステップを完了するように、ユーザーに対してプロンプトを出します。

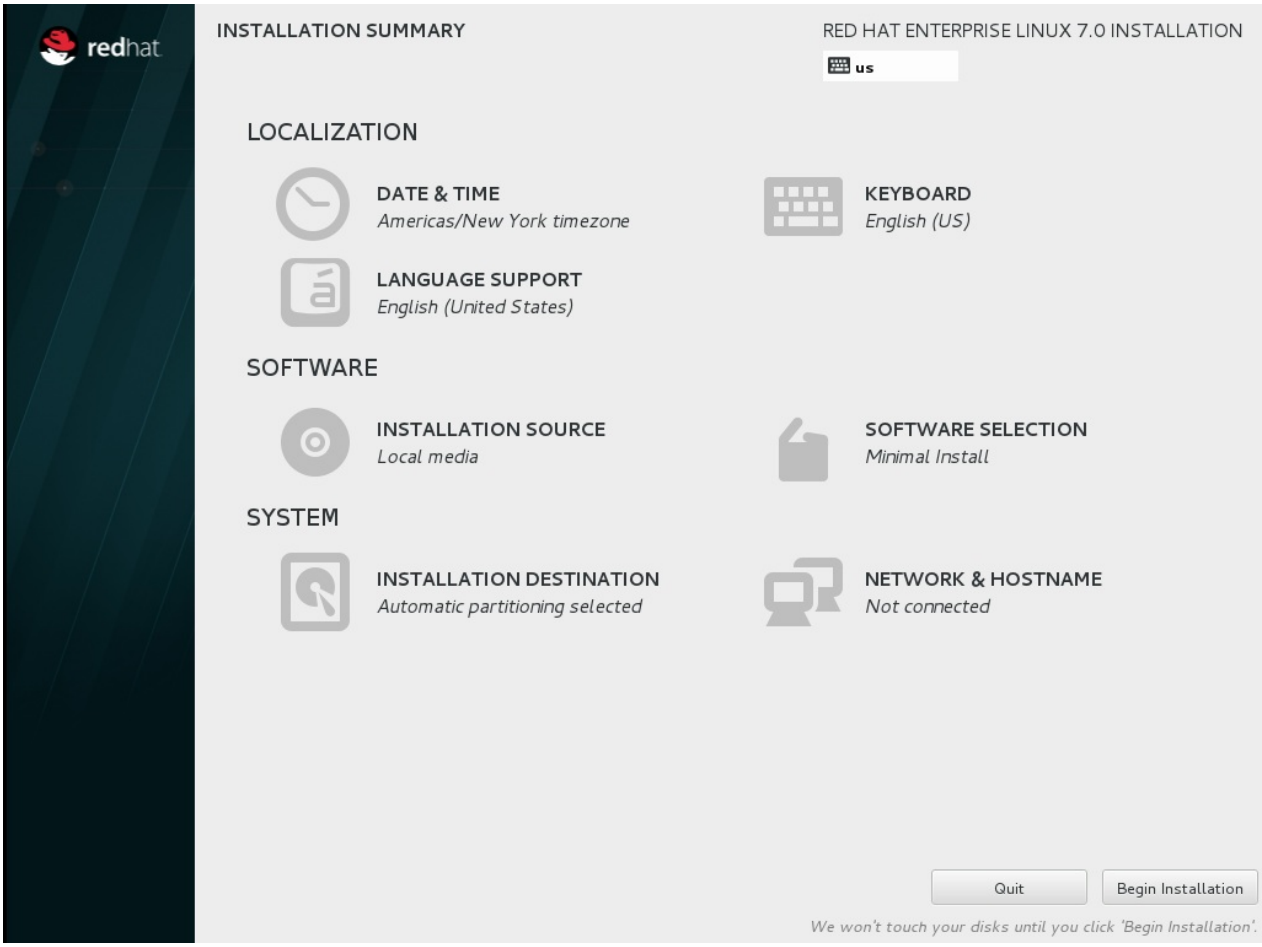


図4.1 「インストール概要」画面

ソフトウェアの選択 はデフォルトで「Minimal Install」に設定されます。**ソフトウェアの選択** 画面を開いて、仮想化パッケージを選択します。

3. **サーバータイプとパッケージグループを選択します。**

   Red Hat Enterprise Linux 7 には、仮想化ホストをインストールするための 2 つの選択可能なオプションがあります。1 つ目は、基本パッケージのみがインストールされた最小仮想化ホスト (ステップ 3.a) であり、2 つ目はグラフィカルユーザーインターフェースでのゲストの管理を可能にするパッケージがインストールされた仮想化ホスト (ステップ 3.b) です。

   a.

   最小仮想化ホストの選択

   ベース環境 の下の **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** の下の **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。

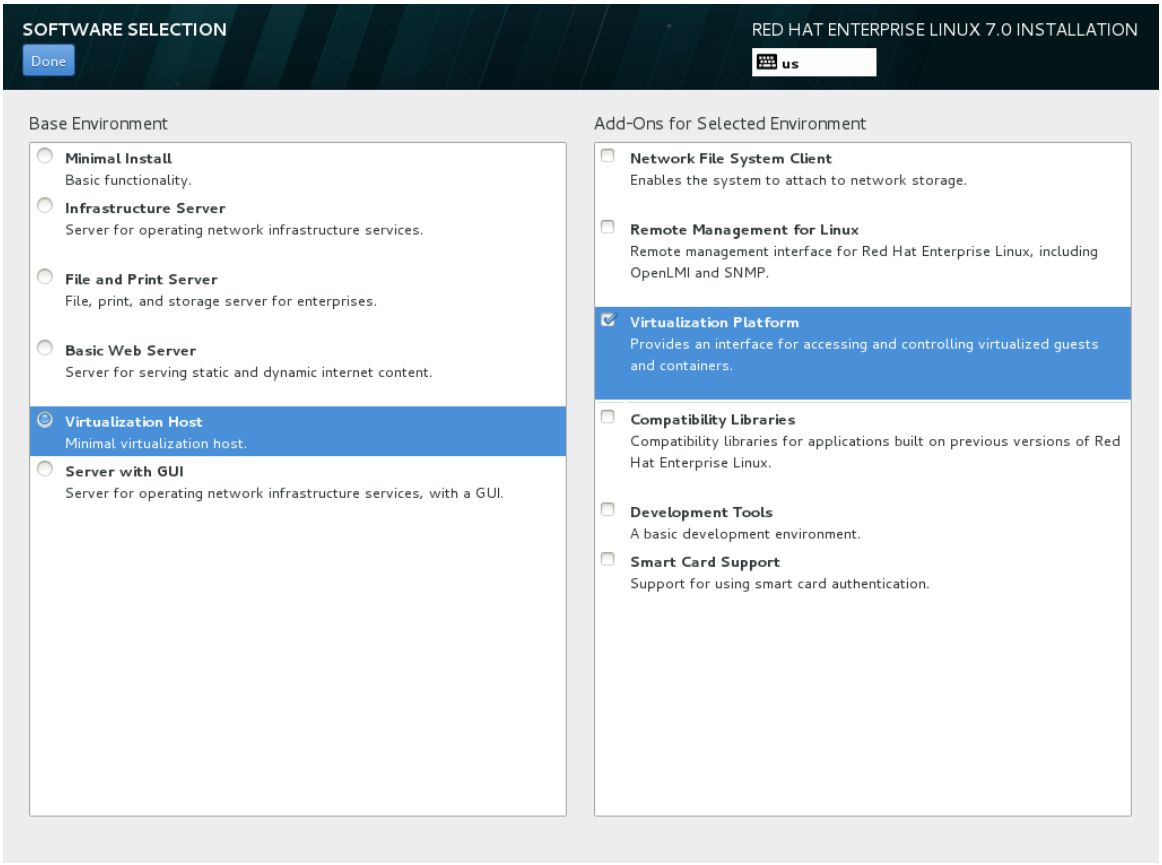


図4.2 「ソフトウェアの選択」画面で選択される仮想化ホスト

b.

グラフィカルユーザーインターフェースのある仮想化ホストの選択

ベース環境 の下の **Server with GUI** ラジオボタンと、**選択した環境のアドオン** の下にある **仮想化クライアント**、**Virtualization Hypervisor**、および **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。

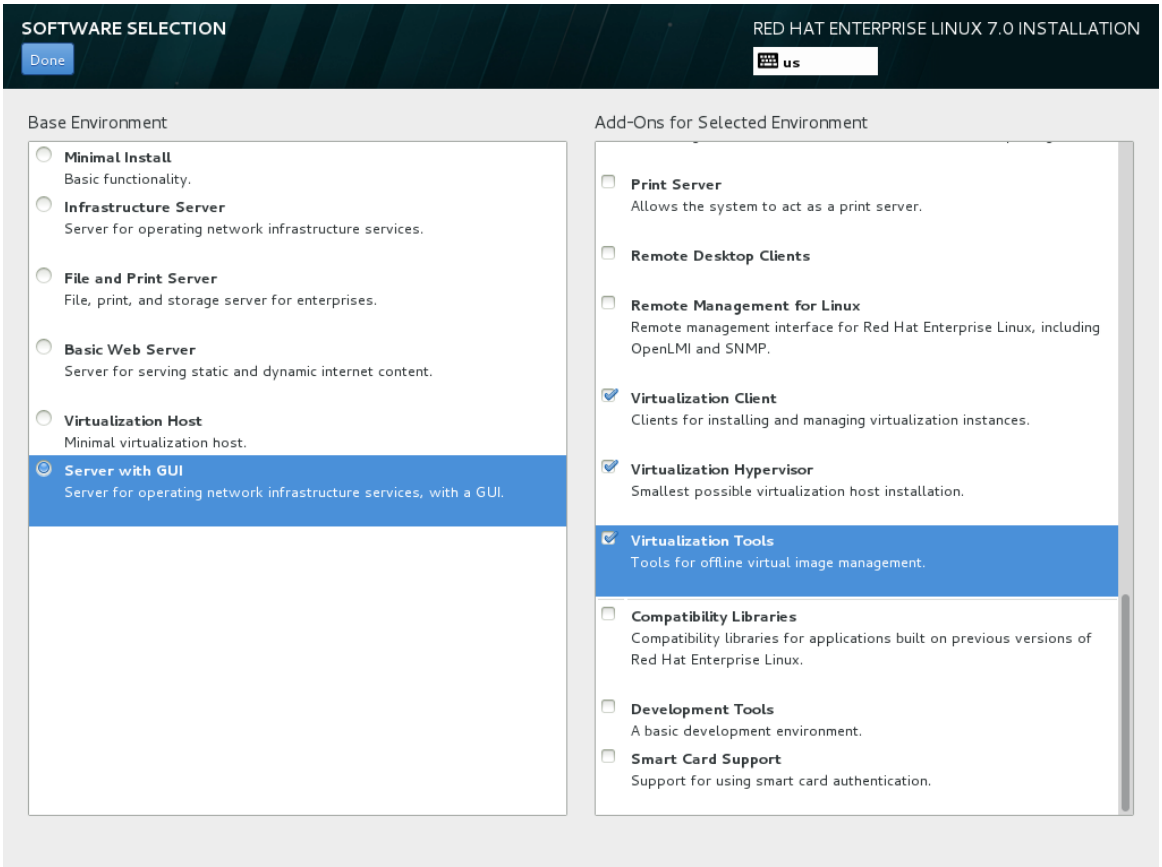


図4.3 「ソフトウェアの選択」画面で選択される Server with GUI

4. **インストールを終了します。**

「インストール概要」画面で必要なステップを完了し、**インストールの開始** をクリックします。

インストールが完了したら、システムを再起動します。

重要

仮想化パッケージの更新を受信するには、有効な仮想化エンタイトルメントが必要です。

キックスタートファイルによる KVM パッケージのインストール

キックスタートファイルを使用すると、ユーザーが手作業で個別のホストシステムをインストールすることなく、自動化された大規模インストールが可能になります。このセクションでは、キックスタートファイルを作成し、これを使用して、仮想化パッケージと共に Red Hat Enterprise Linux をインストールする方法について説明します。

キックスタートファイルの **%packages** セクションでは、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
```

キックスタートファイルについてさらに詳しくは、https://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ からご利用いただける『Red Hat Enterprise Linux インストールガイド』 を参照してください。

4.2. 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、動作中の Red Hat Enterprise Linux 7 またはそれ以降に 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、マシンを登録しておく必要があります。Red Hat サブスクリプションマネージャーで登録するには、**subscription-manager register** コマンドを実行してプロンプトに従います。

有効な Red Hat サブスクリプションをお持ちでない場合は、Red Hat オンラインストアで取得してください。

yum を使用した仮想化パッケージのインストール

Red Hat Enterprise Linux 上で仮想化を使用するには、少なくとも **qemu-kvm** と **qemu-img** パッケージが必要になります。これらのパッケージは、ホストの Red Hat Enterprise Linux システム上にユーザーレベルの KVM エミュレーターとディスクイメージマネージャーを提供します。

qemu-kvm と **qemu-img** パッケージをインストールするには、以下のコマンドを実行します。

```
# yum install qemu-kvm qemu-img
```

以下の仮想化管理パッケージも利用可能です。

推奨される仮想化パッケージ

virt-install

仮想マシン作成用の **virt-install** コマンドを提供します。

libvirt

libvirt 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。*libvirt* パッケージは、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する **libvirtd** デーモンを提供します。

libvirt-python

libvirt-python パッケージには、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

virt-manager は**仮想マシンマネージャー**とも知られ、仮想マシンを管理するグラフィカルツールを提供します。これは、管理 API として *libvirt-client* ライブラリーを使用します。

libvirt-client

libvirt-client 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-manager libvirt libvirt-python python-virtinst
libvirt-client
```

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。

注記

qemu-img パッケージは、システム上ですでにインストールされていない場合に**仮想化** パッケージグループに依存する形でインストールされます。また、前述の **yum install qemu-img** コマンドを使って手動でインストールすることも可能です。

表4.1 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
Virtualization Hypervisor	仮想化ホストの最小インストール	libvirt, qemu-kvm	qemu-kvm-tools
仮想化クライアント	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	virt-top、libguestfs-tools、libguestfs-tools-c
仮想化プラットフォーム	仮想マシンおよびコンテナー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	fence-virtd-libvirt、fence-virtd-multicast、fence-virtd-serial、libvirt-cim、libvirt-java、libvirt-snmp、perl-Sys-Virt
仮想化ツール	オフラインの仮想イメージを管理するためのツール	libguestfs	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 *groupname*** コマンドを実行します。たとえば、**仮想化ツール** パッケージグループをインストールするには、**yum groupinstall "Virtualization Tools"** コマンドを実行します。

第5章 ゲスト仮想マシンのインストール概要

仮想化パッケージをホストシステムにインストールすると、ゲストオペレーティングシステムの作成が可能になります。この章では、仮想マシンにゲストオペレーティングシステムをインストールする全般的なプロセスを説明します。ゲスト仮想マシンの作成は、**virt-manager** の **新規作成 (New)** ボタンか、または **virt-install** コマンドラインインターフェースを使用します。この章では、どちらの方法についても説明します。

Red Hat Enterprise Linux および Microsoft Windows の特定バージョンの詳細なインストール方法は、この後に続く章で説明しています。

5.1. ゲスト仮想マシンの前提条件および留意事項

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検討するだけでなく、定期的な継続モニタリングと (負荷やクライアント数などの) 様々な要素に基づく評価の実行をお勧めします。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、ストレージやより高速なディスクタイプへの優先度の高いアクセスを必要とするものもあります。ストレージのデプロイとメンテナンス時には、ゲストが使用するストレージ容量も定期的に監視して考慮することをお勧めします。

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、ディスクデバイスパラメーターが以下の例のように **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'>
```

5.2. virt-install を使用したゲストの作成

コマンドラインから **virt-install** コマンドを使用してゲスト仮想マシンを作成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されます。**virt-install** をキックスタートファイルと共に使用すると、仮想マシンの無人インストールが可能になります。

virt-install ツールは、コマンドラインで使用可能な数多くのオプションを提供します。オプションの詳細一覧を表示するには、以下のコマンドを実行します。

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

virt-install コマンドを正しく完了するには、root 権限が必要となります。**virt-install** の man ページも、各コマンドのオプションと重要な変数について記載しています。

qemu-img は、**virt-install** の前にストレージオプションを設定するために使用できる関連コマンドです。

--graphics は重要なオプションで、仮想マシンのグラフィカルインストールを可能にします。

以下の例では、個別に作成する必要のあるネットワークブリッジを使用します。ネットワークブリッジの作成方法についての詳細は、「libvirt を使用したブリッジネットワーキング」を参照してください。

例5.1 virt-install を使用した Red Hat Enterprise Linux 6 のゲスト仮想マシンのインストール

この例では、Red Hat Enterprise Linux 6 ゲストを作成します。

```
virt-install \
   --name=guest1-rhel6-64 \
   --disk path=/var/lib/libvirt/images/guest1-rhel6-
64.dsk,size=8,sparse=false,cache=none \
   --graphics spice \
   --vcpus=2 --ram=2048 \
   --location=http://example1.com/installation_tree/RHEL6.4-Server-
x86_64/os \
   --network bridge=br0 \
   --os-type=linux \
   --os-variant=rhel6
```

Red Hat Enterprise Linux 7 では、virtio-scsi コントローラーをゲストで使用することができます。ホストとゲストの両方が virtio-scsi をサポートする場合、以下のようにこれを使用することができます。

例5.2 virt-install を使用した virtio-scsi コントローラーによるゲスト仮想マシンのインストール

virtio-scsi コントローラーを使用するには、標準インインストールに加えて太字で表示されている項目が必要になります。

```
virt-install \
   --name=guest1-rhel6-64 \
   --controller type=scsi,model=virtio-scsi \
   --disk path=/var/lib/libvirt/images/guest1-rhel6-
64.dsk,size=8,sparse=false,cache=none,bus=scsi \
   --graphics spice \
   --vcpus=2 --ram=2048 \
   --location=http://example1.com/installation_tree/RHEL6.4-Server-
```

```
x86_64/os \
    --network bridge=br0 \
    --os-type=linux \
    --os-variant=rhel6
```

このコマンドを実行する際は、オペレーティングシステムについて正しい ***os-type*** を選択していることを確認してください。

追加の例については、**man virt-install** を参照してください。

注記

virt-install で Windows ゲストをインストールする場合は、**--os-type=*windows*** オプションが推奨されます。このオプションは、インストール手順を実行中の再起動時に CD-ROM の切断を回避します。**--os-variant** オプションは、特定のゲストオペレーティングシステムの設定をさらに最適化します。

5.3. virt-manager を使用したゲスト作成

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

手順5.1 virt-manager を使ったゲスト仮想マシンの作成

1. **virt-manager を開く**

   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。

2. **オプション: リモートハイパーバイザーを開く**

   ハイパーバイザーを選択し、**接続** ボタンを押してリモートハイパーバイザーに接続します。

3. **新規の仮想マシンを作成**

   virt-manager ウィンドウで新規の仮想マシンを作成できます。**新しい仮想マシンを作成** ボタン (図5.1「仮想マシンマネージャーのウィンドウ」) をクリックして **新しい仮想マシン** ウィザードを開きます。

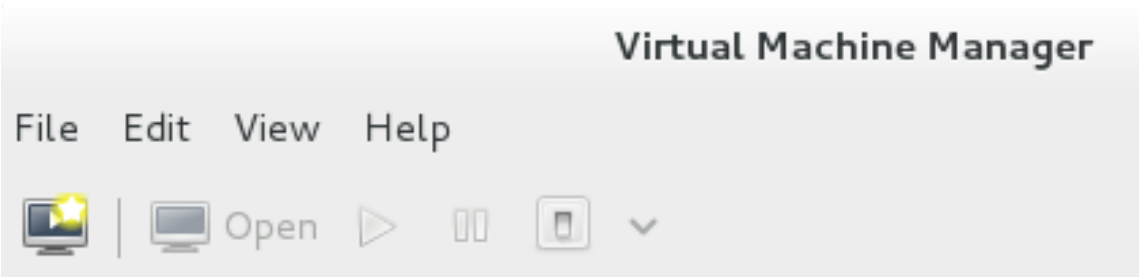


図5.1 仮想マシンマネージャーのウィンドウ

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。

a. ゲスト仮想マシンの名前の入力とインストール方法の選択

b. インストールメディアの場所の選択と設定

c. メモリーと CPU オプションの設定

d. 仮想マシンのストレージの設定

e. 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) 。

4. **名前とインストール方法の特定**

最初にゲスト仮想マシンに名前を付け、インストール方法を選択します。仮想マシンの名前にはアンダースコア (_) 、ピリオド (.) 、ハイフン (-) を使用することができます。

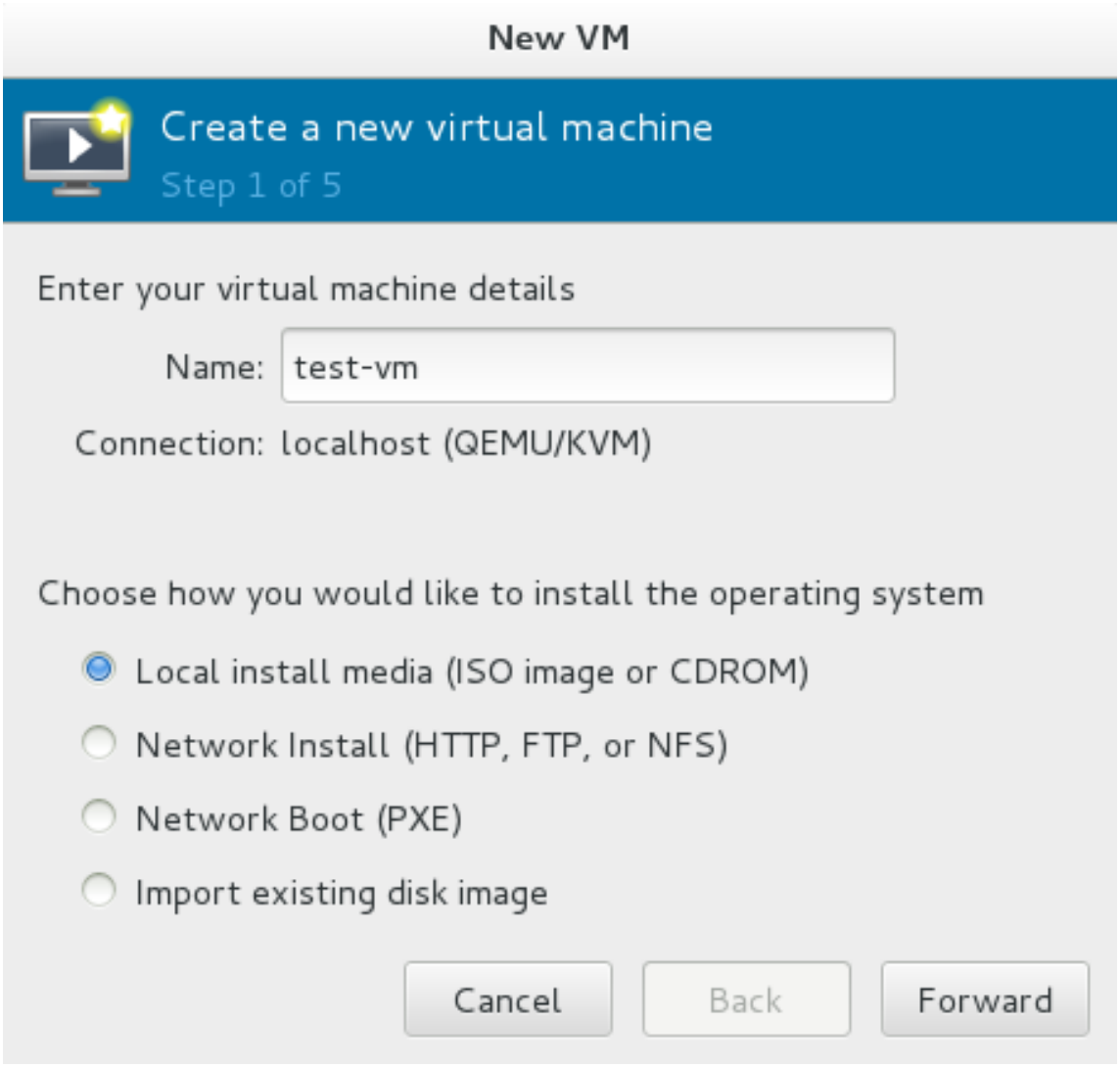


図5.2 仮想マシンに名前を付けてインストール方法を選択

仮想マシンの名前を入力し、インストール方法を選択します。

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 、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、『導入ガイド』 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。PXE インストールのネットワーク設定要件についての情報は、「PXE を使用したゲスト仮想マシンのインストール」 を参照してください。

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。

進む をクリックし、続けます。

5. **インストール設定**

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 OS の種類を選択することを確認してください。インストール方法によっては、インストールの URL または既存のストレージパスを指定します。

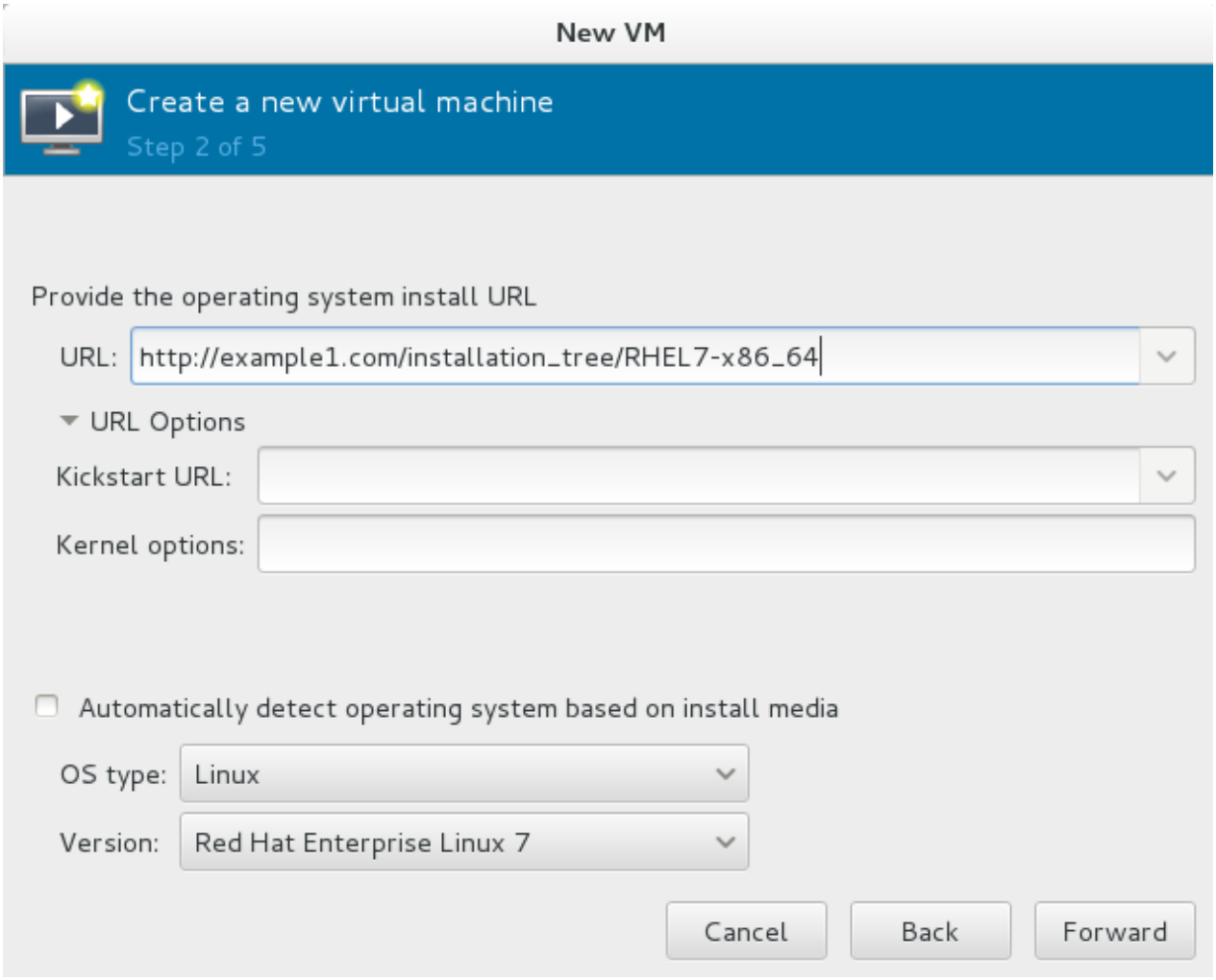


図5.3 リモートインストールの URL

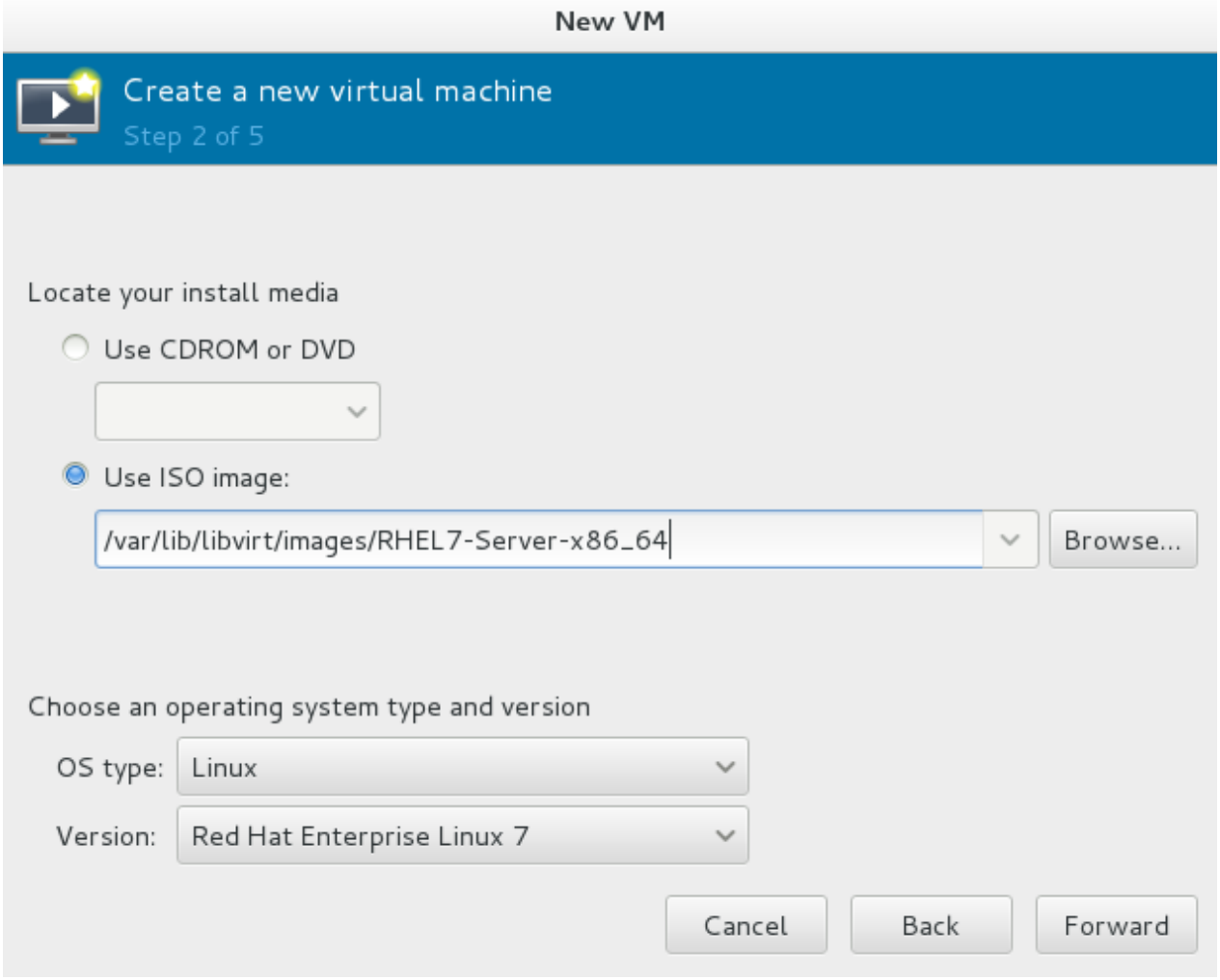


図5.4 ローカル ISO イメージのインストール

6. **CPU およびメモリーの設定**

   次に、仮想マシンに割り当てる CPU の数とメモリー量を設定します。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。これらを設定して、**進む** をクリックします。

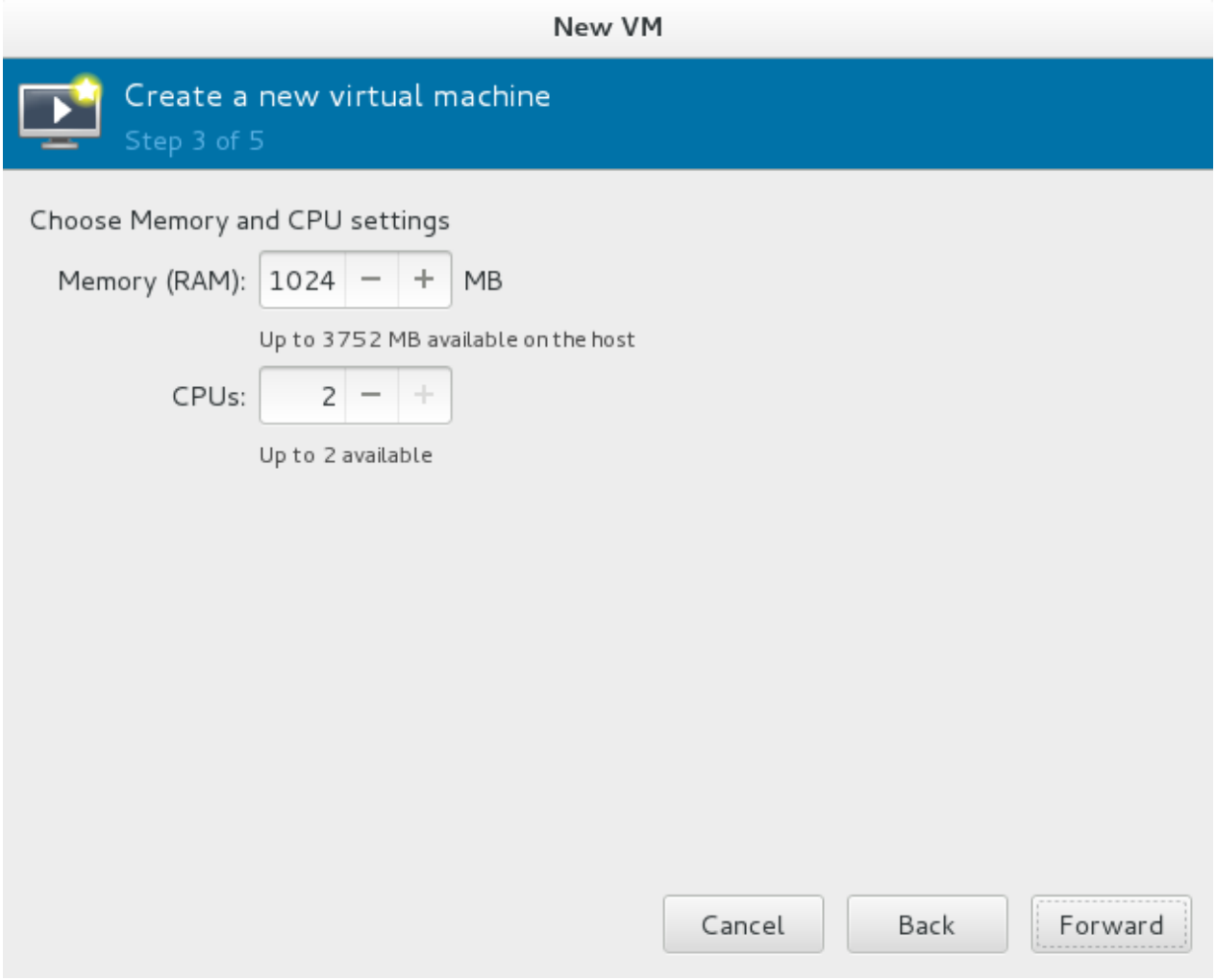


図5.5 CPU およびメモリーの設定

7. **ストレージの設定**

   ゲスト仮想マシンにストレージを割り当てます。

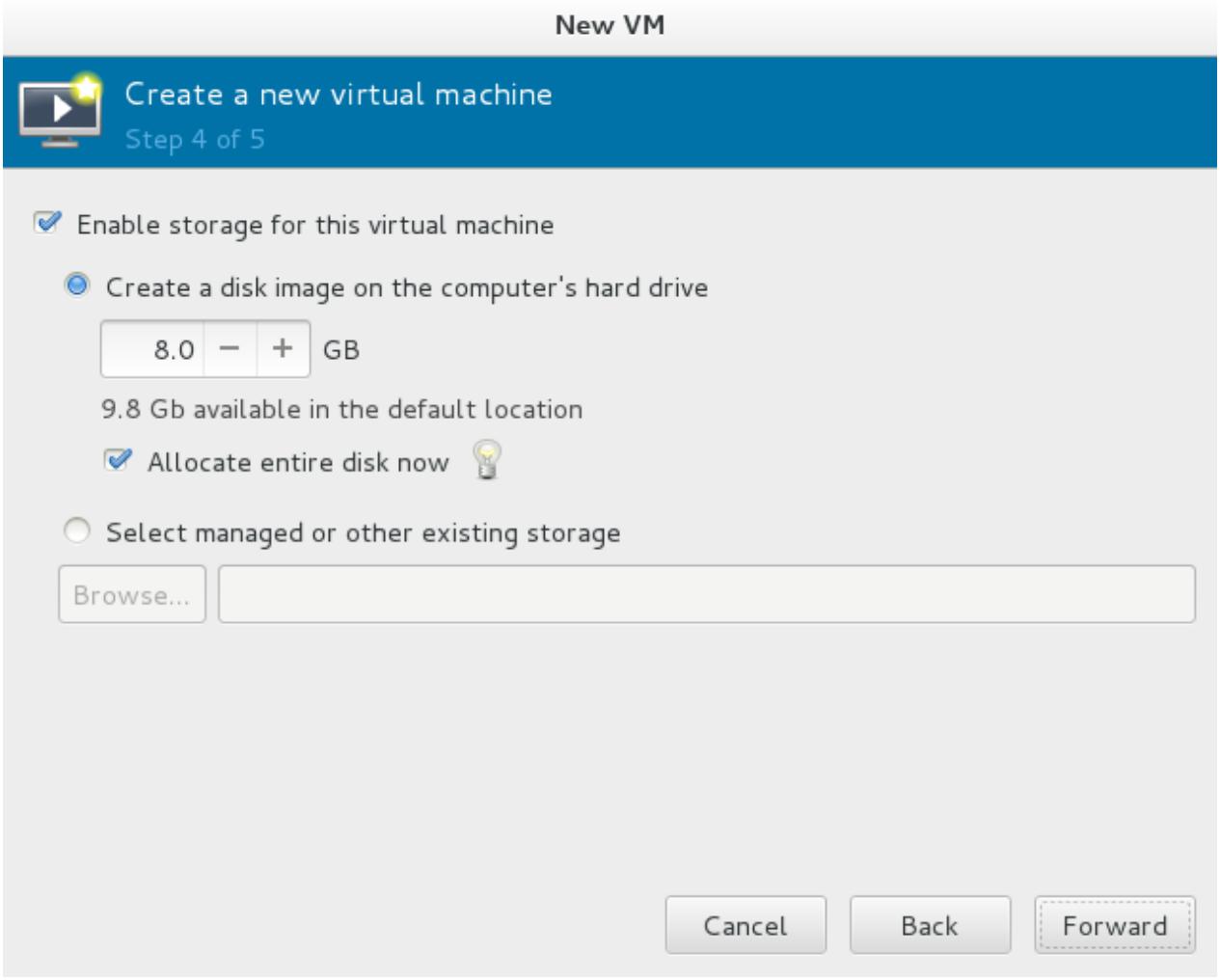


図5.6 仮想マシンのストレージの設定

最初のステップで既存のディスクイメージのインポートを選択した場合は、**virt-manager** はこのステップを飛ばします。

仮想マシンと必要なアプリケーションに十分な領域を割り当て、**進む** をクリックして次に進みます。

8. **最終設定**

注記

タイプが Red Hat Enterprise Linux 5 または Red Hat Enterprise Linux 4 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。

仮想マシンの設定を確認し、問題がなければ**完了** をクリックします。デフォルトのネットワーク設定、仮想化の種類、アーキテクチャーで仮想マシンが作成されます。

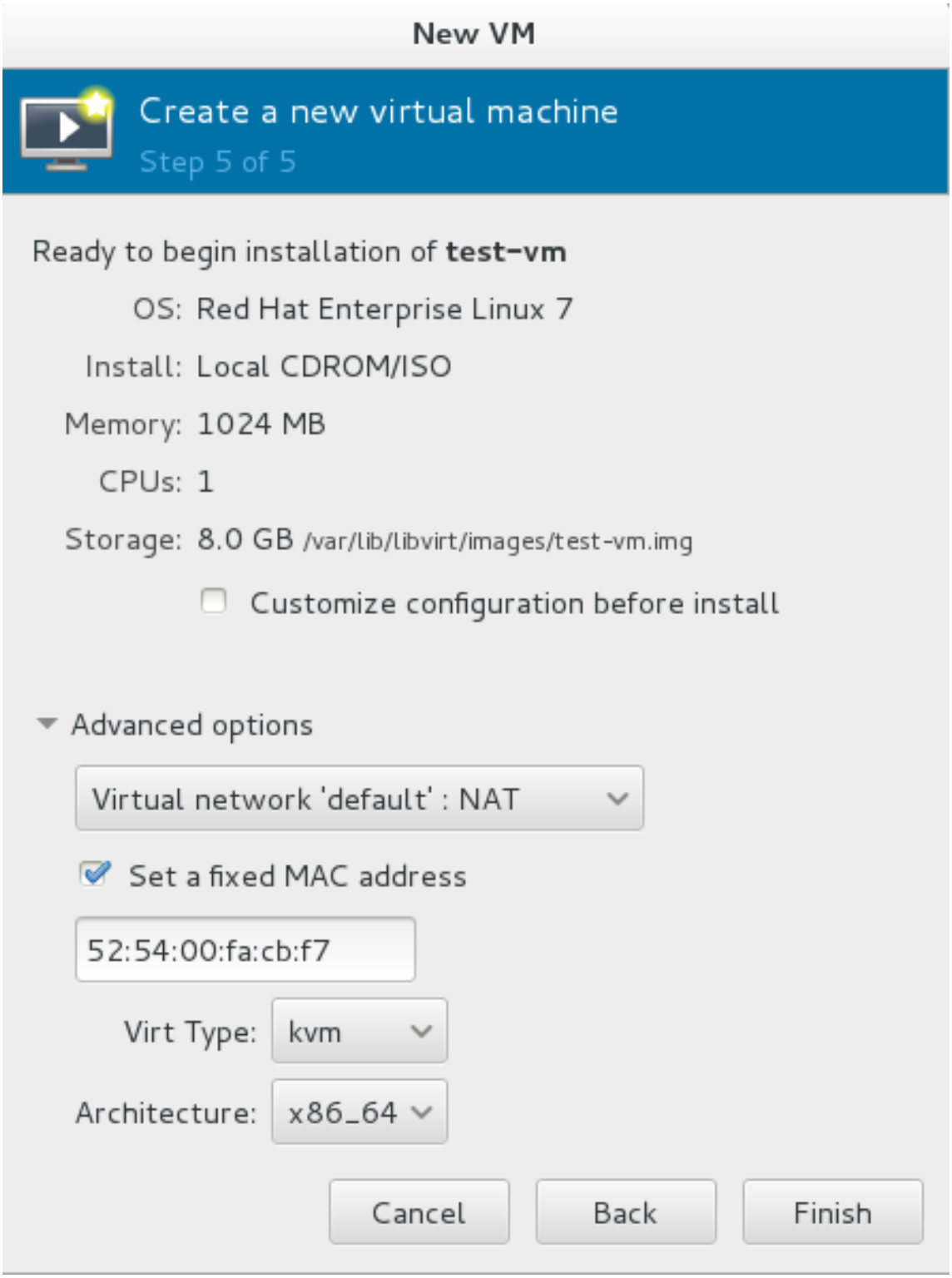


図5.7 設定の確認

仮想マシンのハードウェアをさらに設定したい場合は、**完了** をクリックする前に **インストールの前に設定をカスタマイズする** ボックスにチェックを入れます。新たなウィザードが開き、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。

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。

5.4. PXE を使用したゲスト仮想マシンのインストール

PXE ゲストインストールでは、インストールするゲスト仮想マシンと同じサブネット上で PXE サーバーが実行されている必要があります。この実行方法は、仮想マシンのネットワーク接続方法によって異なります。PXE サーバーの設定でヘルプが必要な場合は、サポートに連絡してください。

5.4.1. virt-install を使用した PXE インストール

virt-install を使った PXE インストールでは、***installation*** がブリッジ名となる **--network=bridge:installation** パラメーターと、***--pxe*** パラメーターの両方が必要です。

デフォルトでは、ネットワークが見つからない場合、ゲスト仮想マシンは別の起動可能なデバイスから起動を試みます。起動可能なデバイスが見つからない場合は、ゲスト仮想マシンは一時停止します。起動可能なデバイスが見つからない場合は、**qemu-kvm** の起動パラメーター ***reboot-timeout*** を使ってゲストの起動を再度試すことができます。

```
# qemu-kvm -boot reboot-timeout=1000
```

例5.3 virt-install を使用した完全仮想化 PXE インストール

```
# virt-install --hvm --connect qemu:///system \
--network=bridge:installation --pxe --graphics spice \
--name rhel6-machine --ram=756 --vcpus=4 \
--os-type=linux --os-variant=rhel6 \
--disk path=/var/lib/libvirt/images/rhel6-machine.img,size=10
```

上記のコマンドは、テキストのみの環境では実行できないことに注意してください。完全仮想化 (***--hvm***) ゲストは、***--graphics spice*** パラメーターの代わりに ***--location*** と ***--extra-args "console=console_type"*** が指定されている場合にのみ、テキストのみの環境でインストールできます。

5.4.2. virt-manager を使用した PXE インストール

手順5.2 virt-manager を使用した PXE インストール

1.

PXE の選択

インストール方法として PXE を選択し、続くステップで OS の種類やメモリー、CPU およびストレージを設定します。

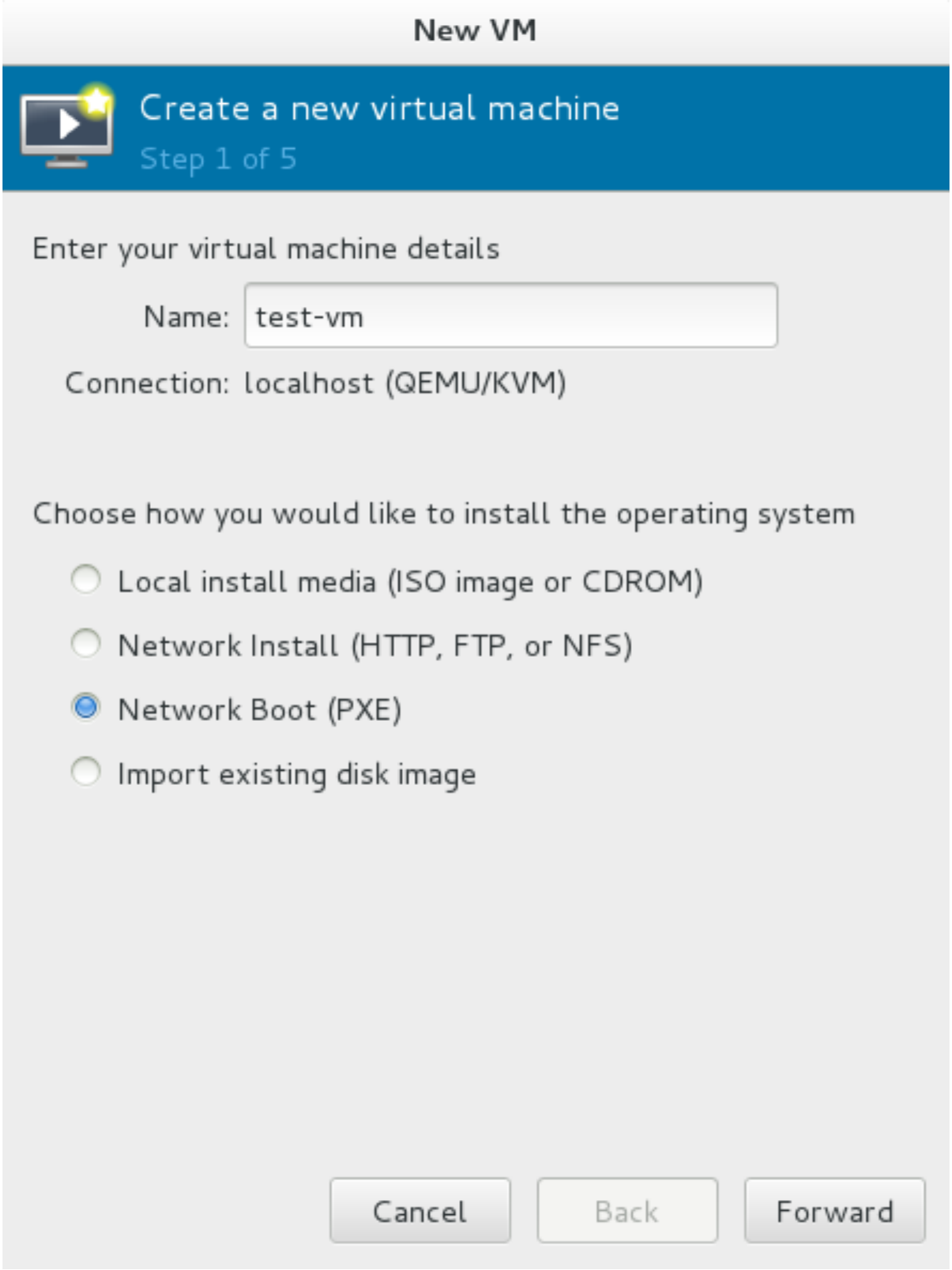


図5.8 インストール方法の選択

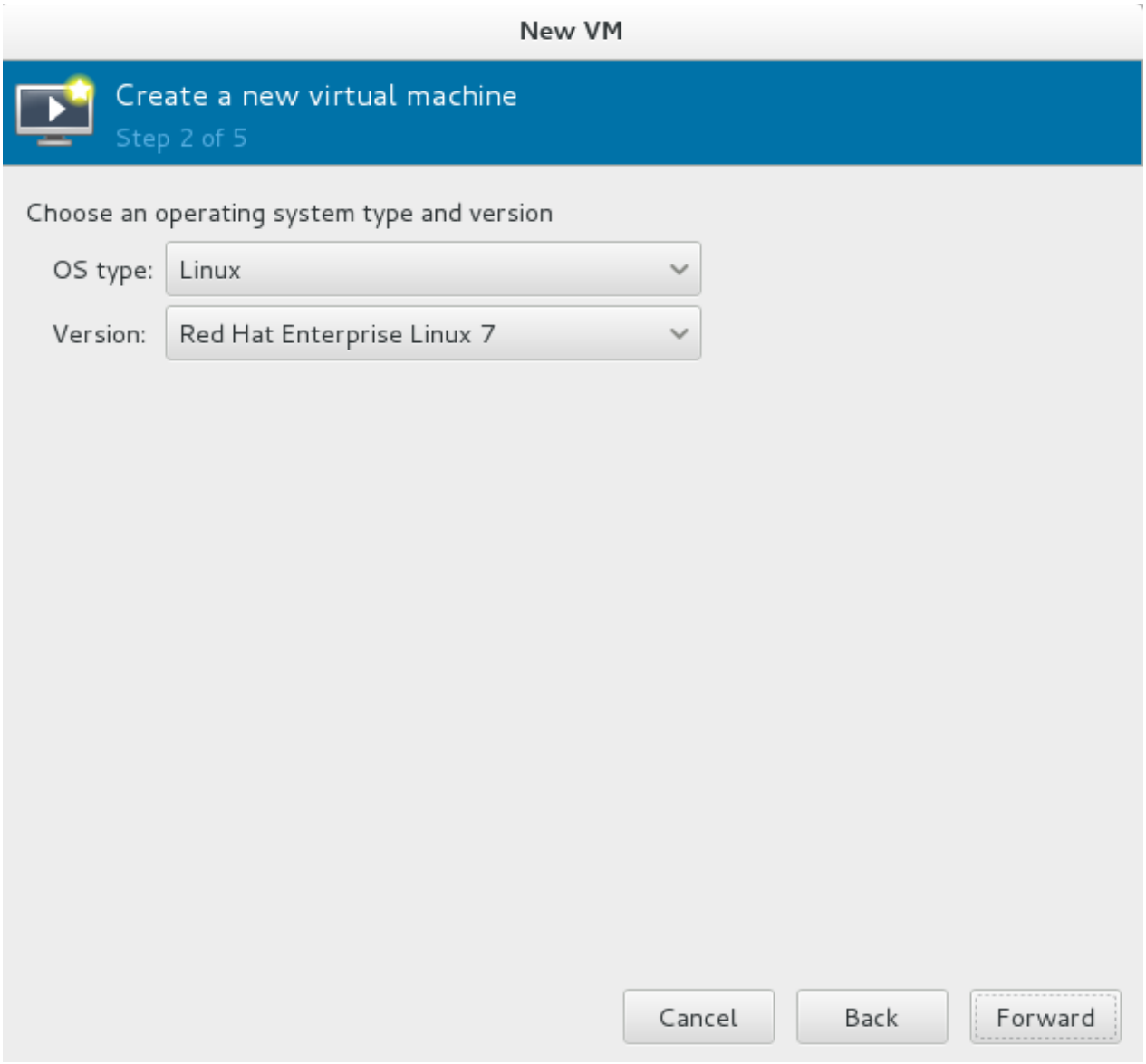


図5.9 インストールする OS の種類の選択

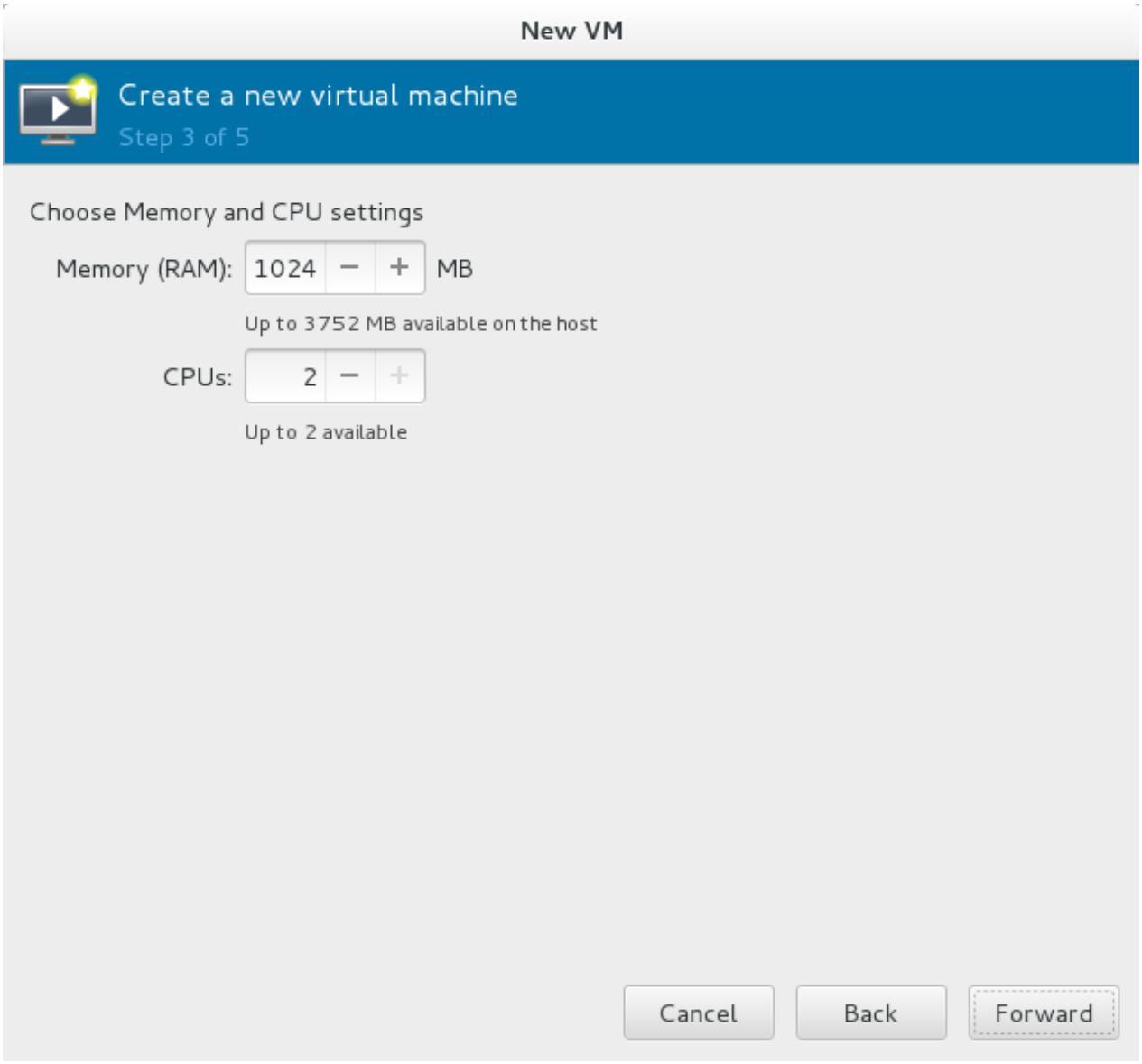


図5.10 仮想化ハードウェア詳細の指定

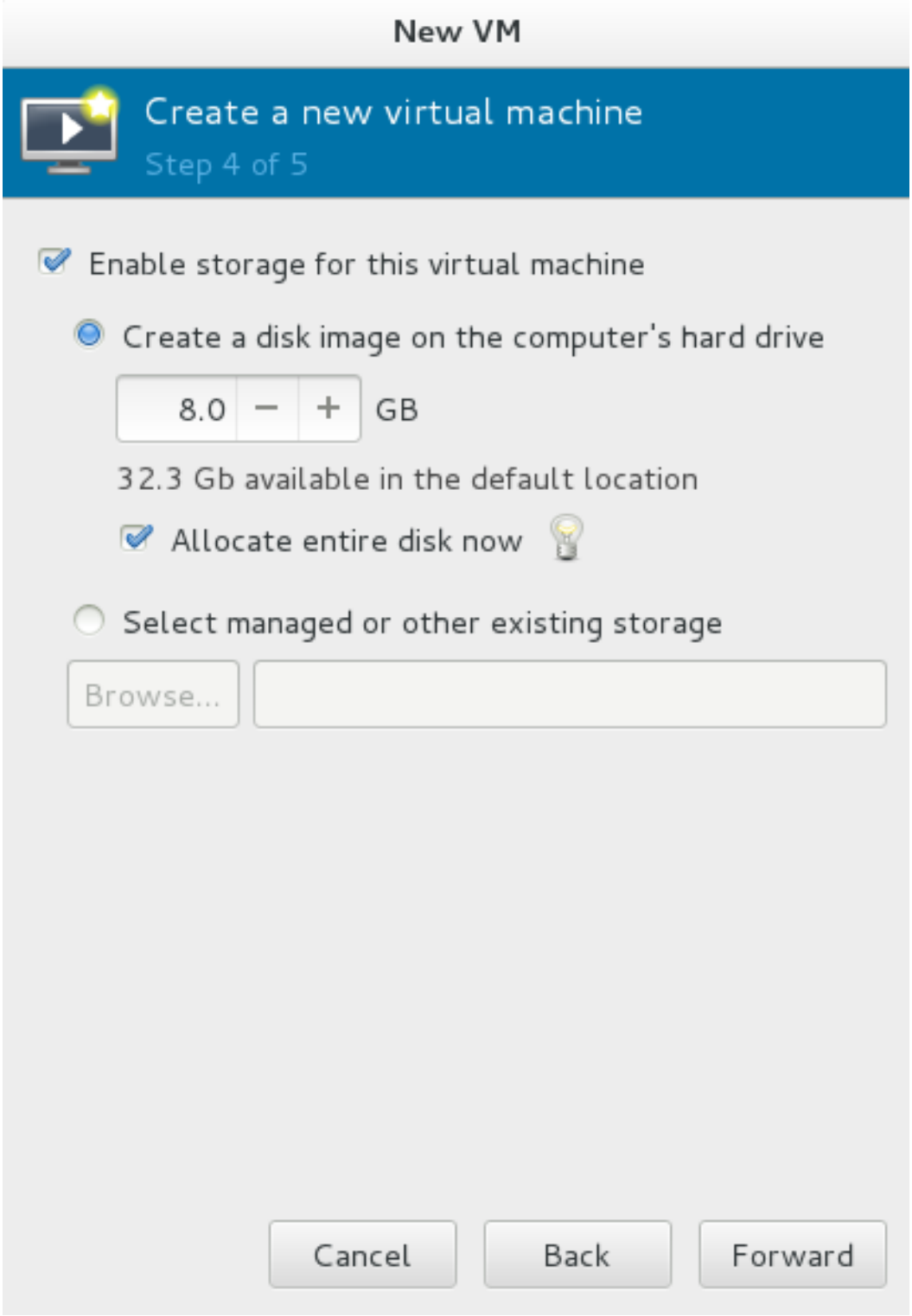


図5.11 ストレージ詳細の指定

2.

インストールの開始

インストールを開始する準備ができました。

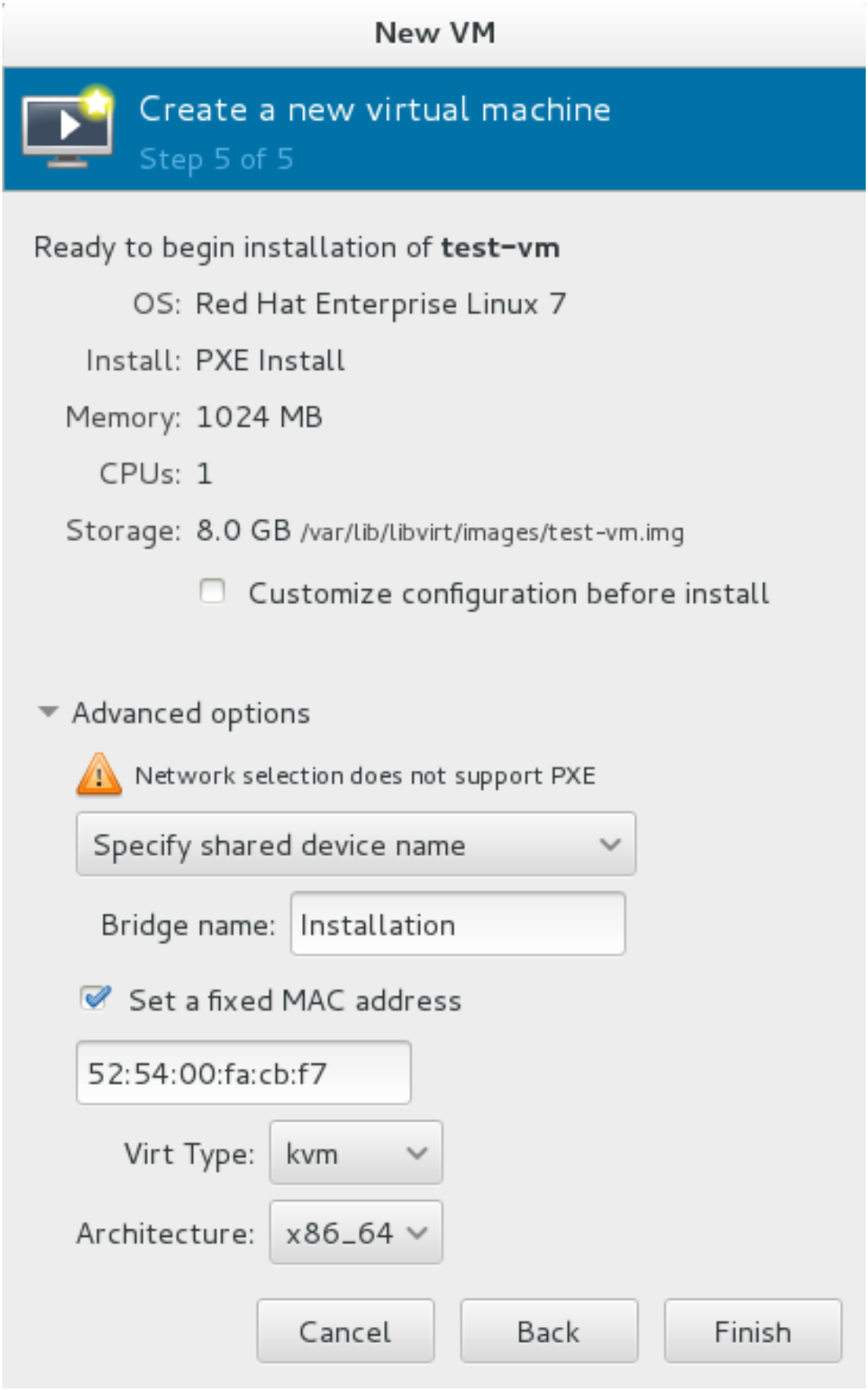


図5.12 仮想マシン詳細の確定

DHCP リクエストが送信され、有効な PXE サーバーが見つかるとゲスト仮想マシンのインストールプロセスが開始されます。

第6章 Red Hat Enterprise Linux 7 ホスト上での Red Hat Enterprise Linux 7 ゲスト仮想マシンのインストール

この章では、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。

注記

仮想化パッケージのインストールに関する詳細は、*4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。

6.1. ローカルのインストールメディアを使用した 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 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、http://access.redhat.com で入手できます。

手順6.1 virt-manager 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、*16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 Ha
   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。

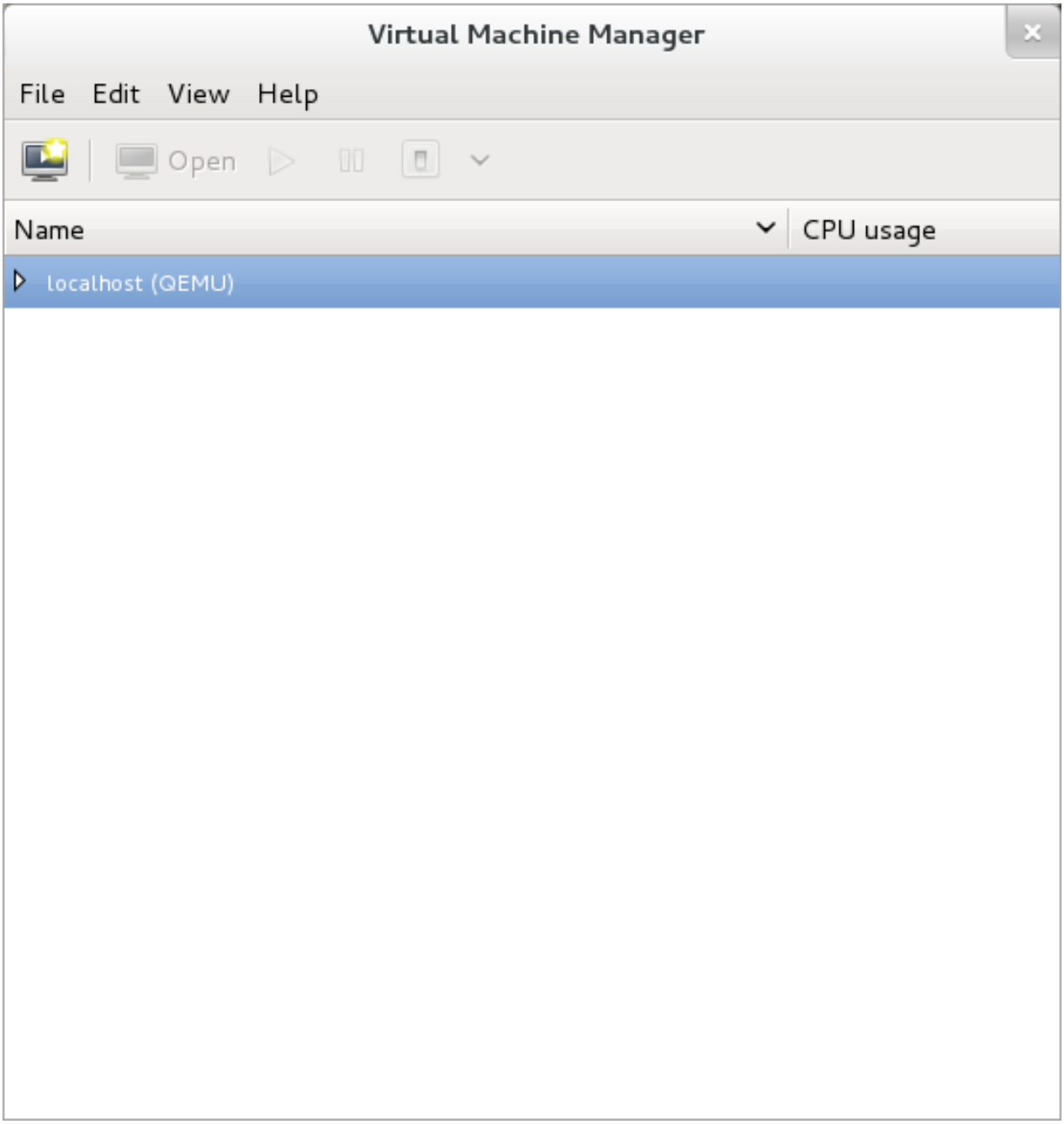


図6.1 仮想マシンマネージャーのウィンドウ

新しい仮想マシンを作成 ボタンをクリックして、新しい仮想マシンのウィザードを開始します。

図6.2 新しい仮想マシンを作成ボタン

新しい仮想マシン ウィンドウが開きます。

3. **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 (-) 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ず、数字のみの名前は使用できません。

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) ラジオボタンを選択します。

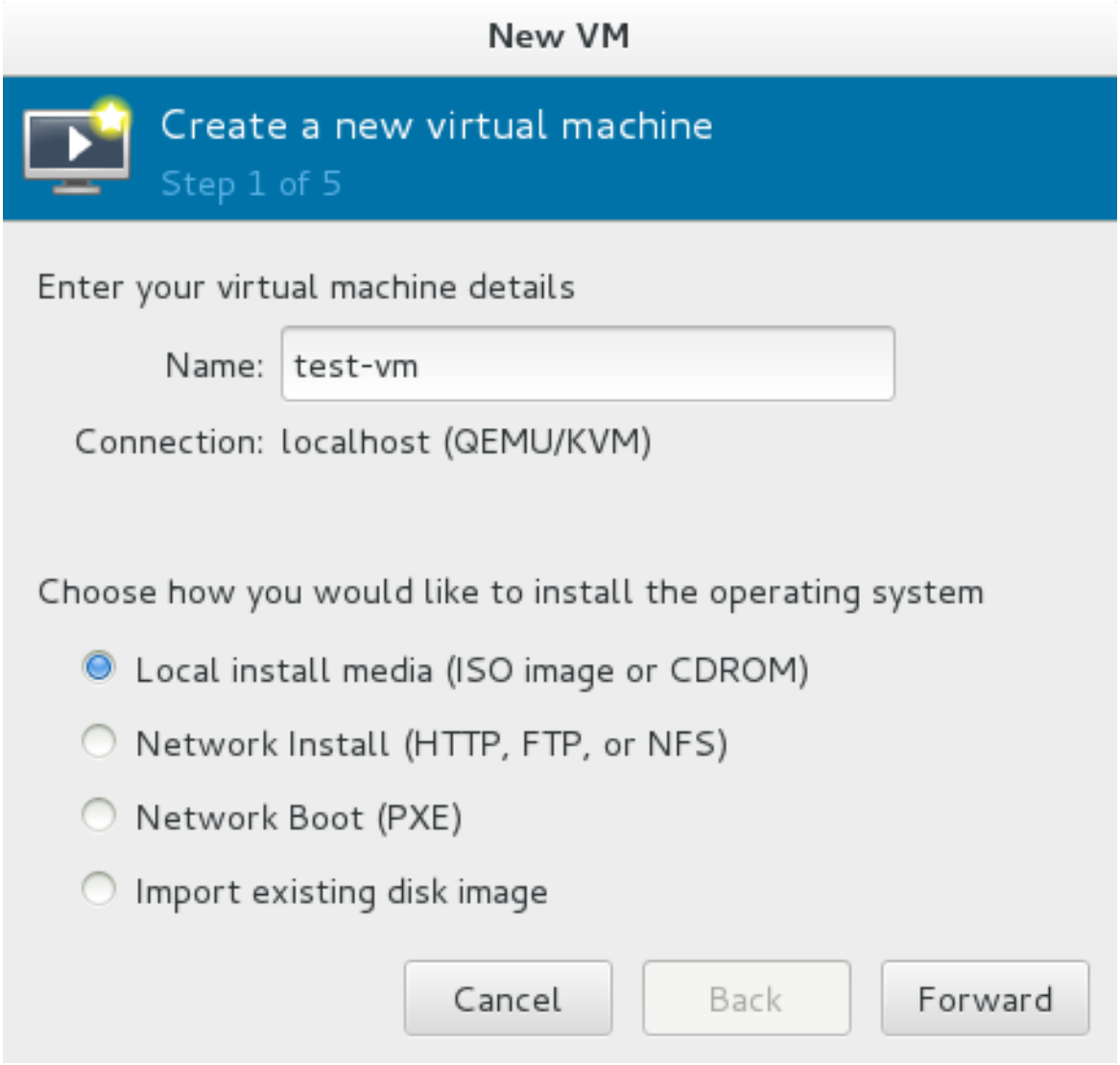


図6.3 「新しい仮想マシン」ウィンドウ - ステップ 1

進む をクリックして次に進みます。

4. **インストールメディアを選択します。**

   該当するインストールメディアのラジオボタンを選択します。

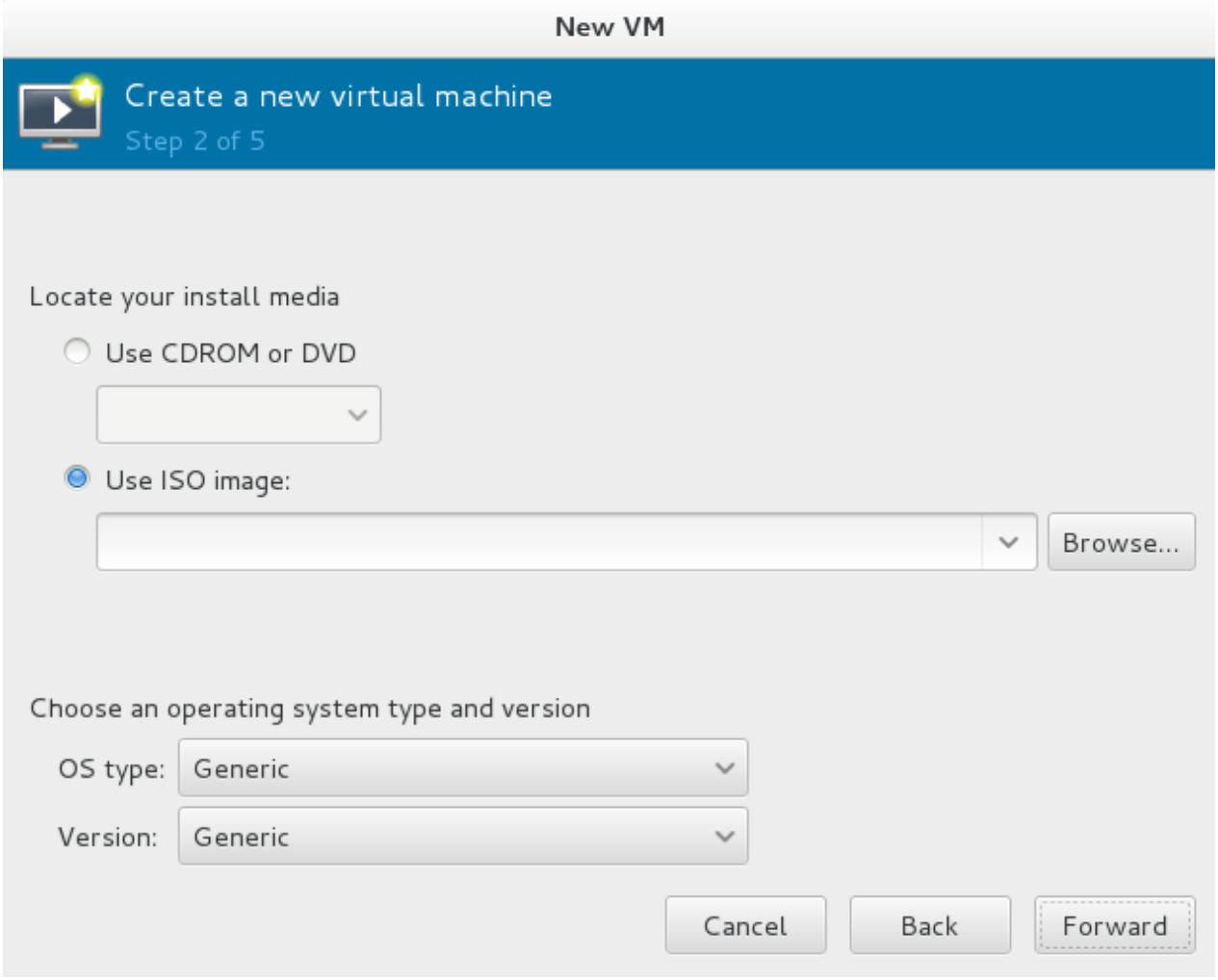


図6.4 インストールメディアの場所

A. 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。

B. 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、**ボリュームを選択** 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 による新たな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 Virtualization Security Guide』 または 『Red Hat Enterprise Linux SELinux User's and Administrator's Guide』 を参照してください。

選択したインストールメディアに一致するオペレーティングシステムの種類とバージョンを選択します。

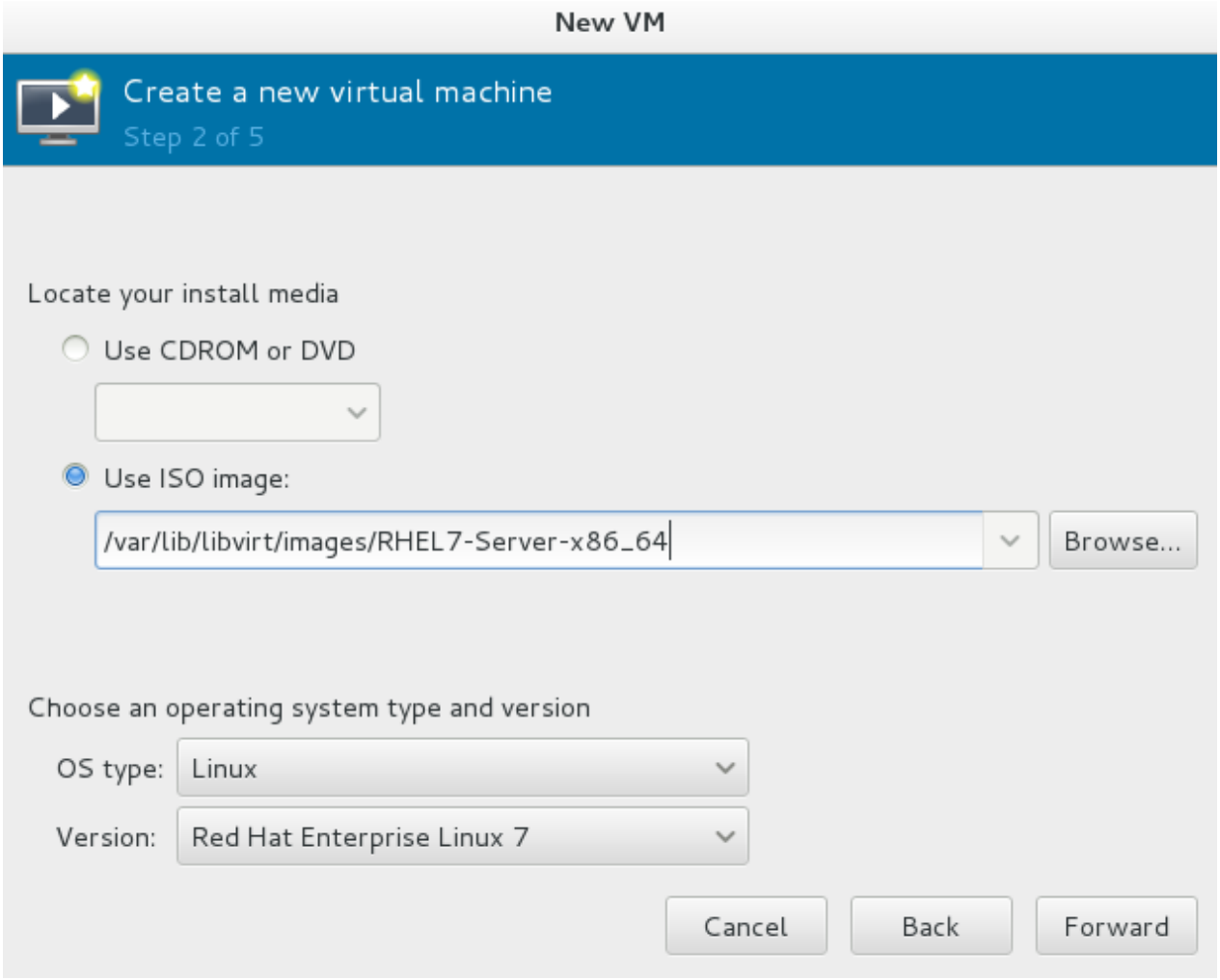


図6.5 「新しい仮想マシン」ウィンドウ - ステップ 2

進む をクリックして次に進みます。

5.

RAM および 仮想 CPU を設定します。

仮想 CPU と 割り当てるRAM の適切な値を選択します。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。メモリーと仮想 CPU のオーバーコミットが可能です。オーバーコミットに関する詳細は、*11章KVM でのオーバーコミット* を参照してください。

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、ゲスト仮想マシンが効果的に実行できるように必要な数の仮想 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ー (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。

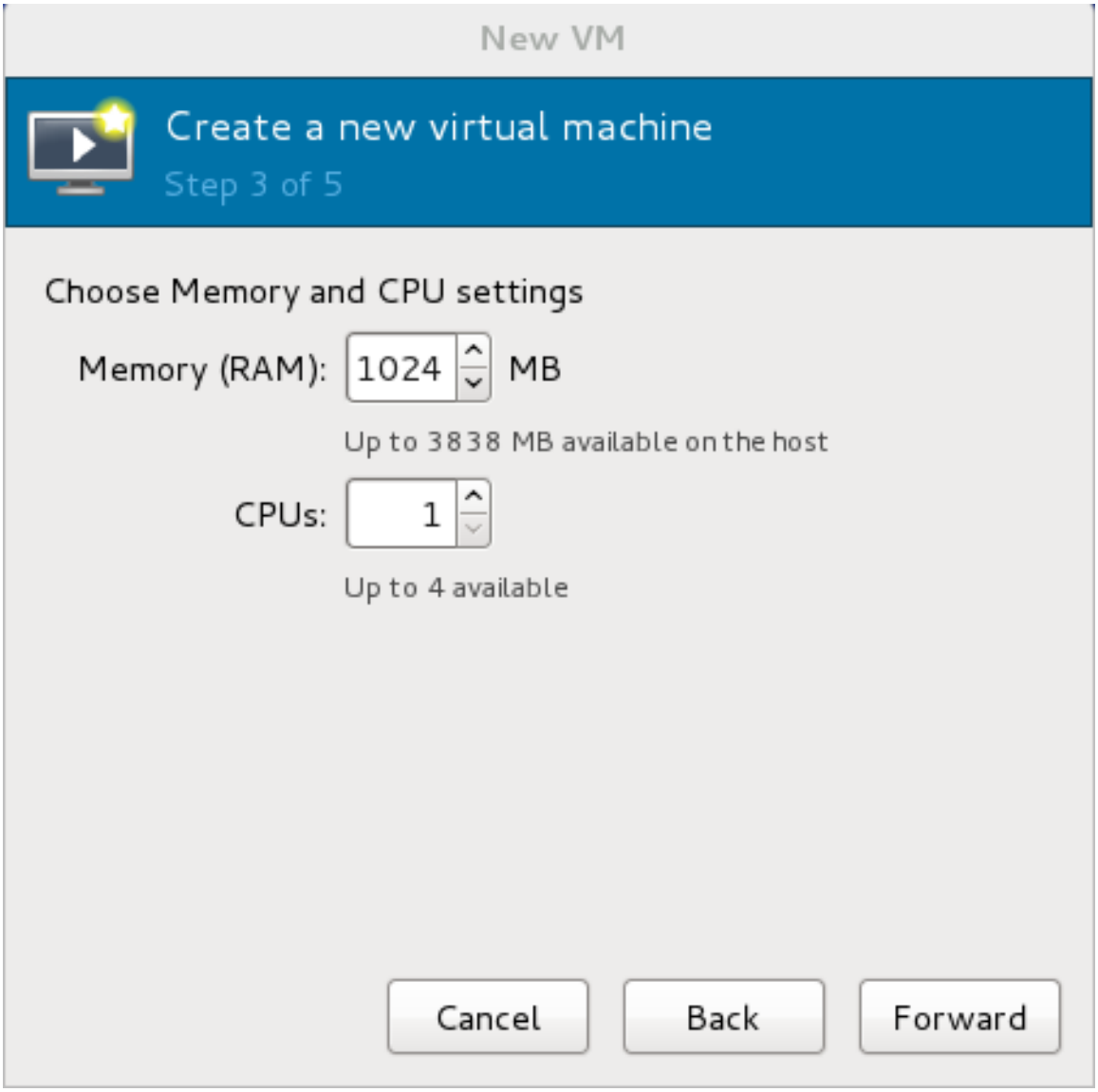


図6.6 「新しい仮想マシン」ウィンドウ - ステップ 3

進む をクリックして次に進みます。

6. **ストレージ**

ストレージを有効にして Red Hat Enterprise Linux 7 ゲスト仮想マシンに割り当てます。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。

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 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座に作成されます。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ーであるものの、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 Exabytes
- 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ー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、スケーラビリティー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。

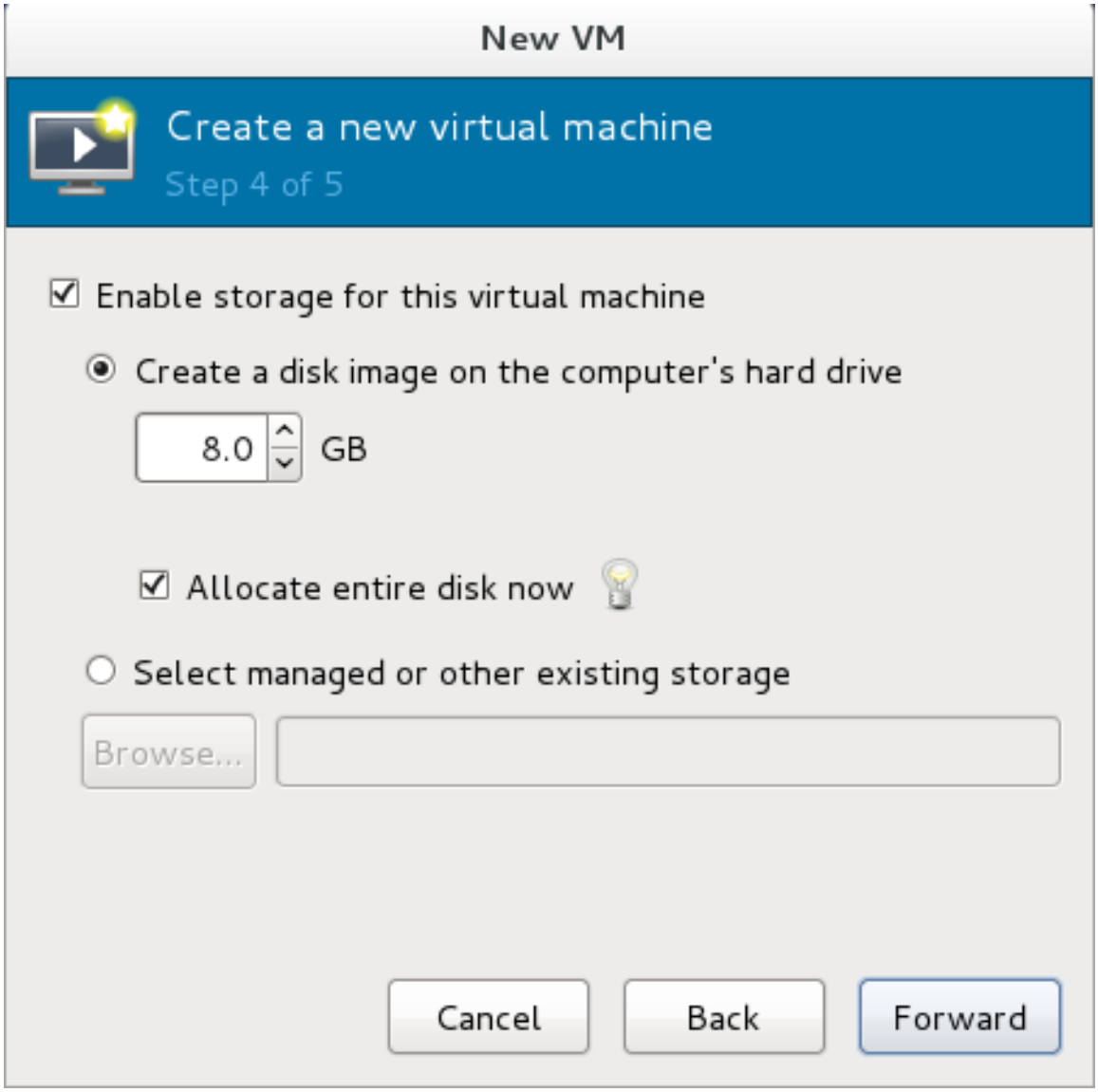


図6.7 「新しい仮想マシン」ウィンドウ - ステップ 4

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. **ストレージプールを使用**

前のステップでストレージプールの使用に **管理しているか、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** ウィンドウが表示されます。

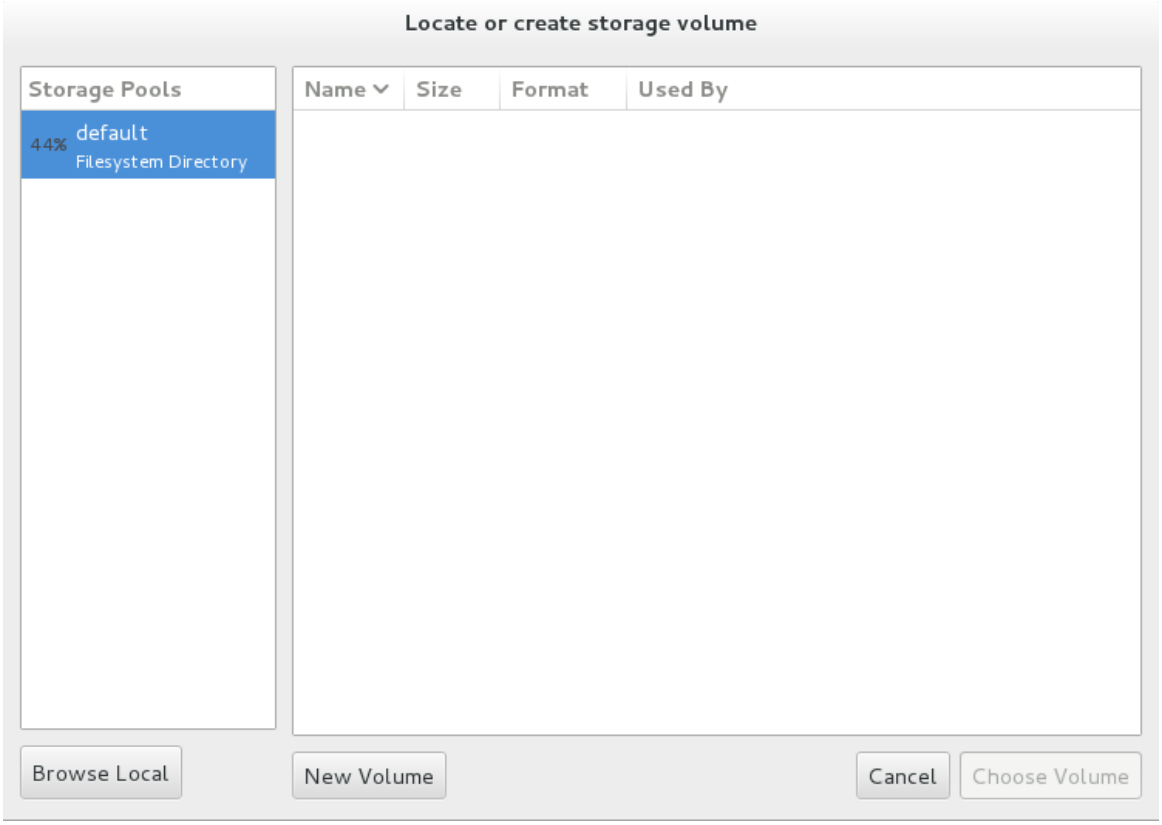


図6.8 「ストレージボリュームの検索または作成」ウィンドウ

i. **Storage Pools** 一覧からストレージプールを選択します。

ii. オプション: **新規ボリューム** ボタ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追加** 画面が表示されます。新規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、cow、qcow、qcow2、vmdk および vpc 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、「ターゲット要素の設定」 を参照してください。

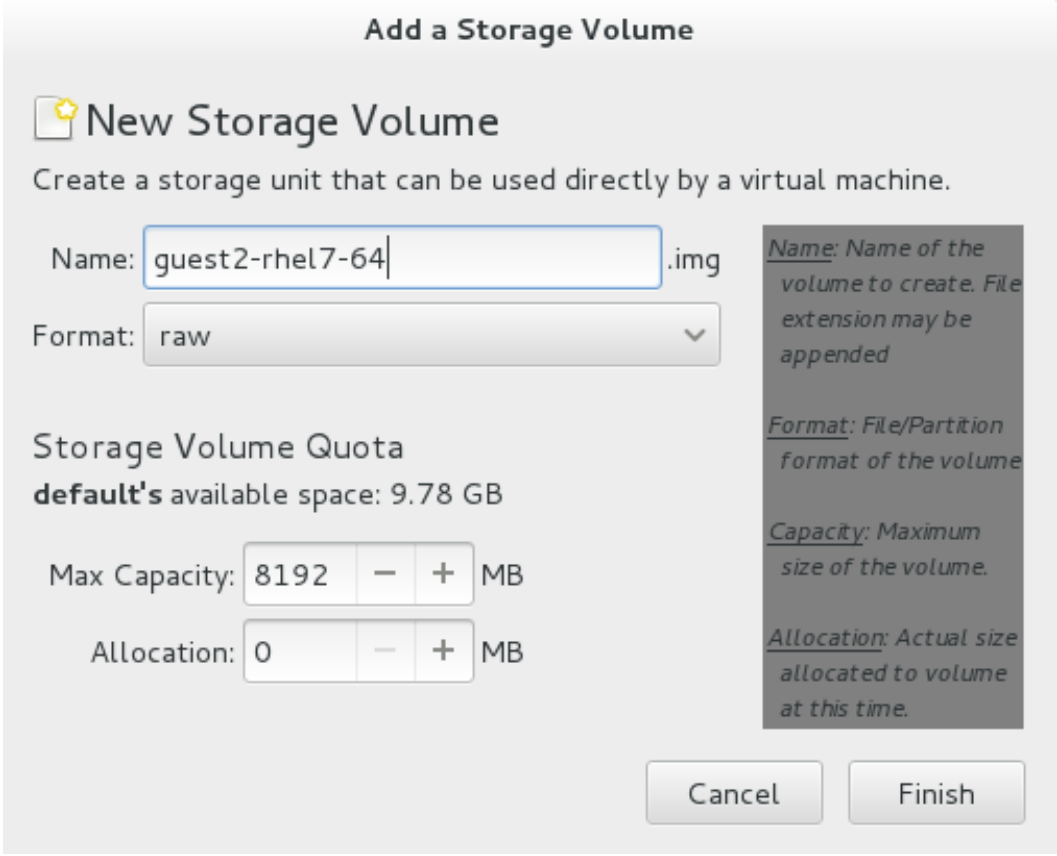


図6.9 「ストレージボリュームを追加」ウィンドウ

完了 ボタンをクリックして、次に進みます。

7. **確認し、終了します。**

   ウィザードにエラーがなく、すべての情報が予想どおりに表示されていることを確認します。

   インストールの前に設定をカスタマイズする チェックボックスを選択し、ゲストのストレージまたはネットワークデバイスを変更するか、準仮想化 (virtio) ドライバーを使用するか、またはデバイスを新たに追加します。

   詳細なオプション の下向き矢印キーをクリックし、詳細オプションを確認し、変更します。標準的な Red Hat Enterprise Linux 7 インストールの場合、これらのオプションを変更する必要はありません。

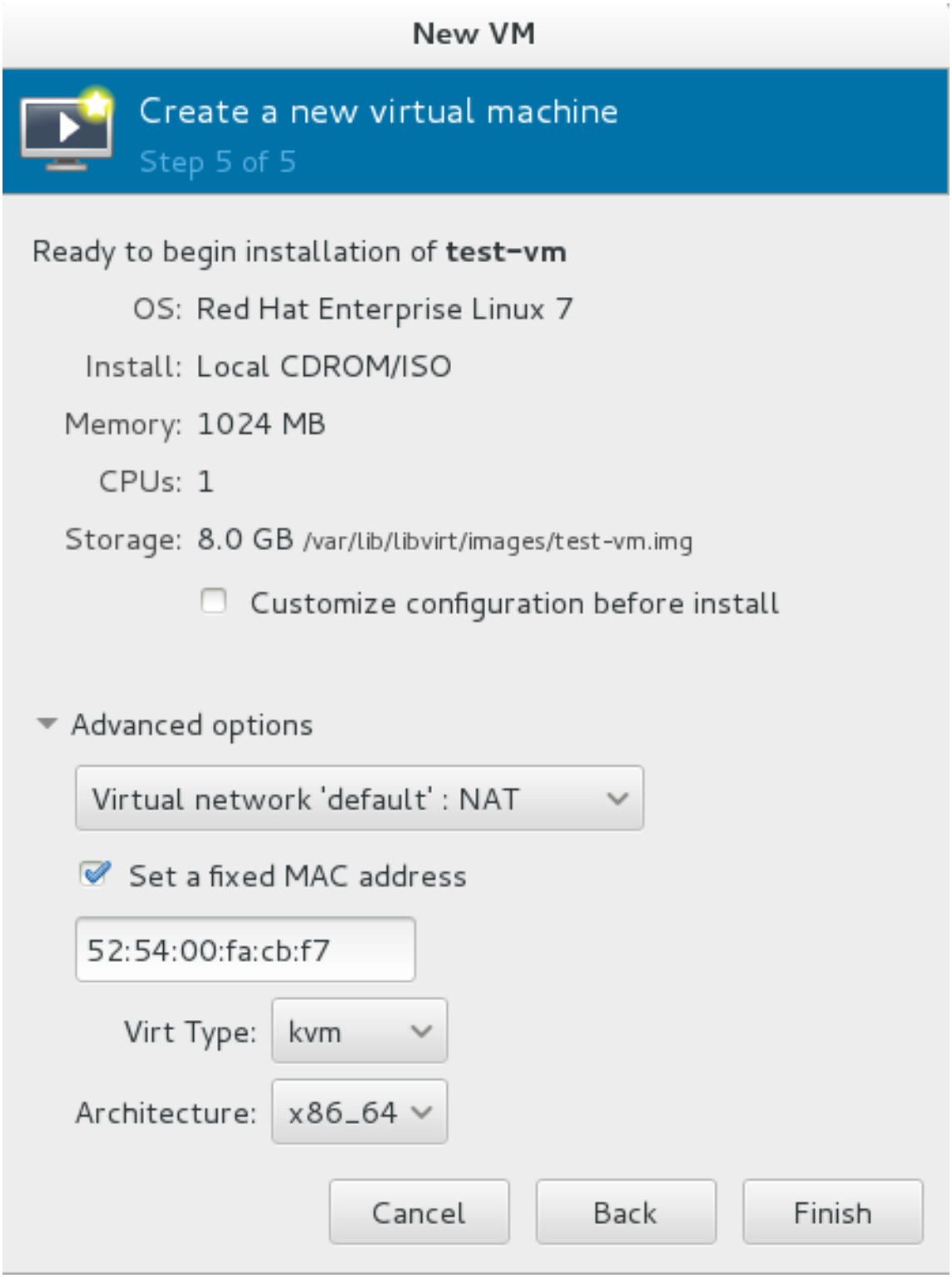


図6.10 「新しい仮想マシン」ウィンドウ - ローカルストレージ

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 Installation Guide』 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。

6.2. ネットワークインストールツリーを使用した Red Hat Enterprise Linux 7 ゲストの作成

手順6.2 virt-manager を使用した Red Hat Enterprise Linux 7 ゲストの作成

1. **オプション: 準備**

   ゲスト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、16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 Ha 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。

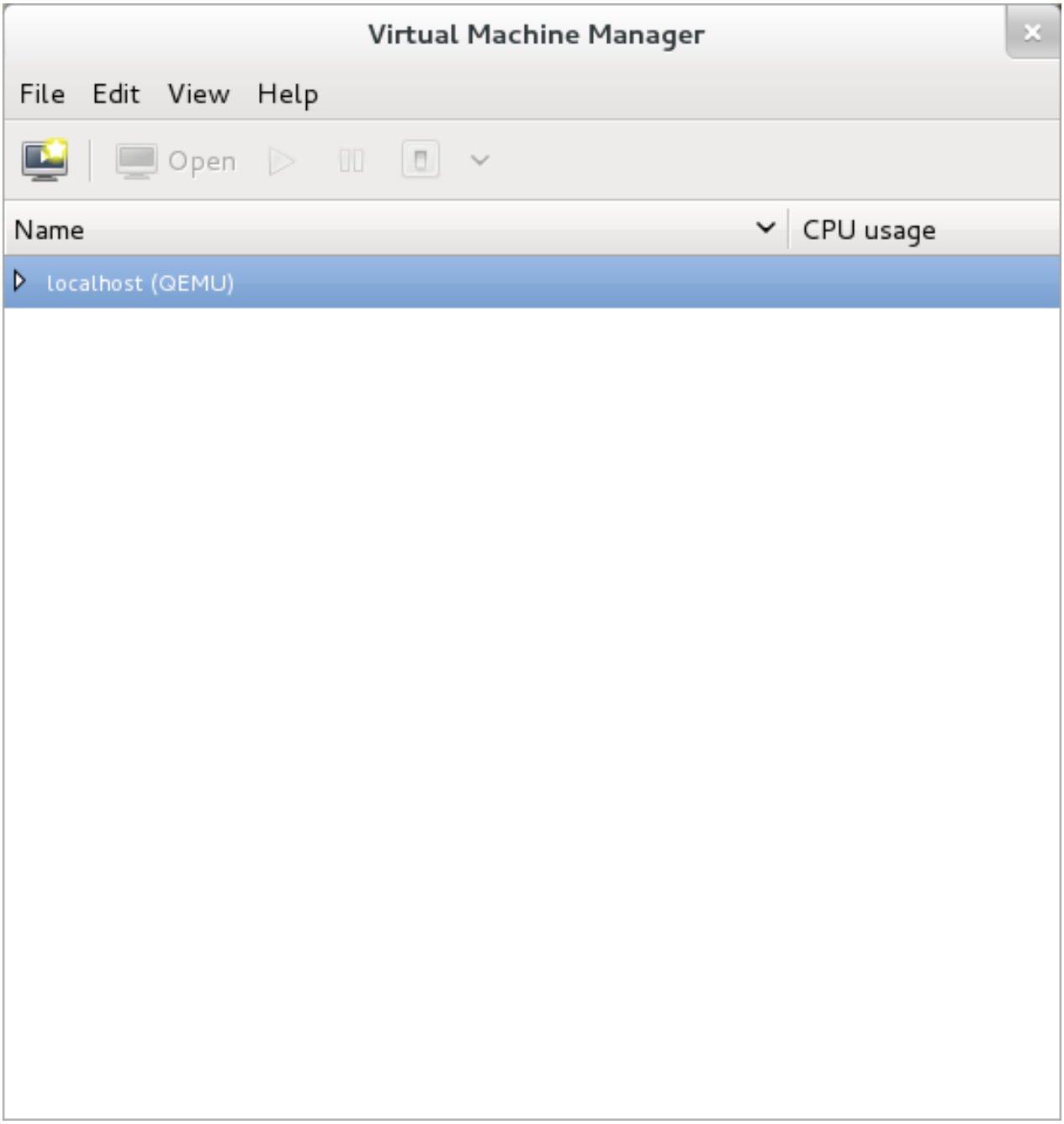


図6.11 virt-manager のメインウィンドウ

新しい仮想マシンを作成 ボタンをクリックして、新しい仮想マシンのウィザードを開始します。

図6.12 新しい仮想マシンを作成ボタン

新しい仮想マシンを作成 ウィンドウが表示されます。

3. **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 (-) 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ず、数字のみの名前は使用できません。

ラジオボタンの一覧からインストール方法を選択します。

図6.13 「新しい仮想マシン」ウィンドウ - ステップ 1

進む をクリックして次に進みます。

4. インストール URL を指定します。必要に応じて、キックスタート URL およびカーネルオプションを指定します。

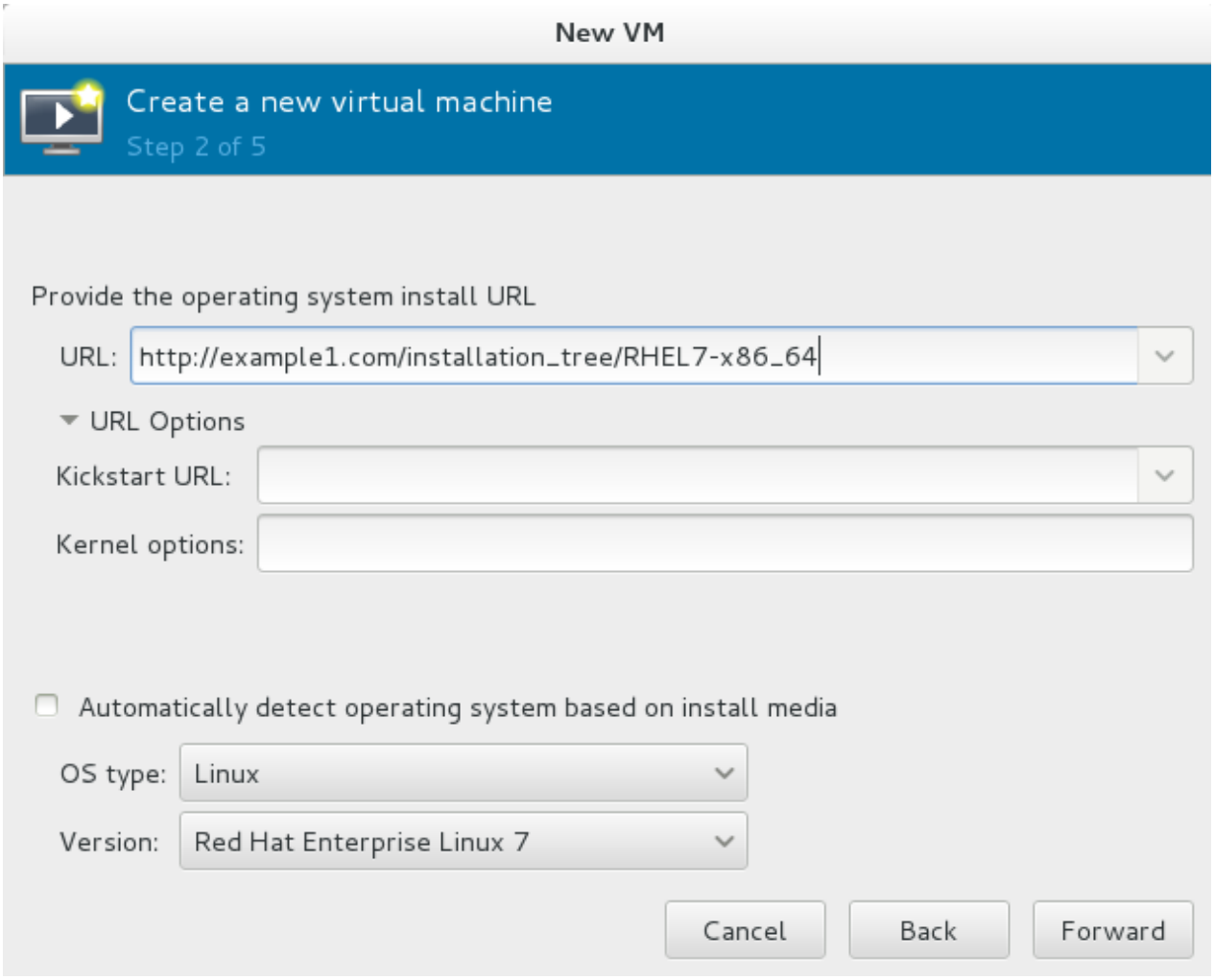


図6.14 「新しい仮想マシン」ウィンドウ - ステップ 2

進む をクリックして次に進みます。

5. 残りのステップは、ISO インストール手順と同じです。ISO インストール手順の ステップ 5 以降を実行します。

6.3. PXE を使用した Red Hat Enterprise Linux 7 ゲストの作成

手順6.3 virt-manager を使用した Red Hat Enterprise Linux 7 ゲストの作成

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、16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 Ha 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。

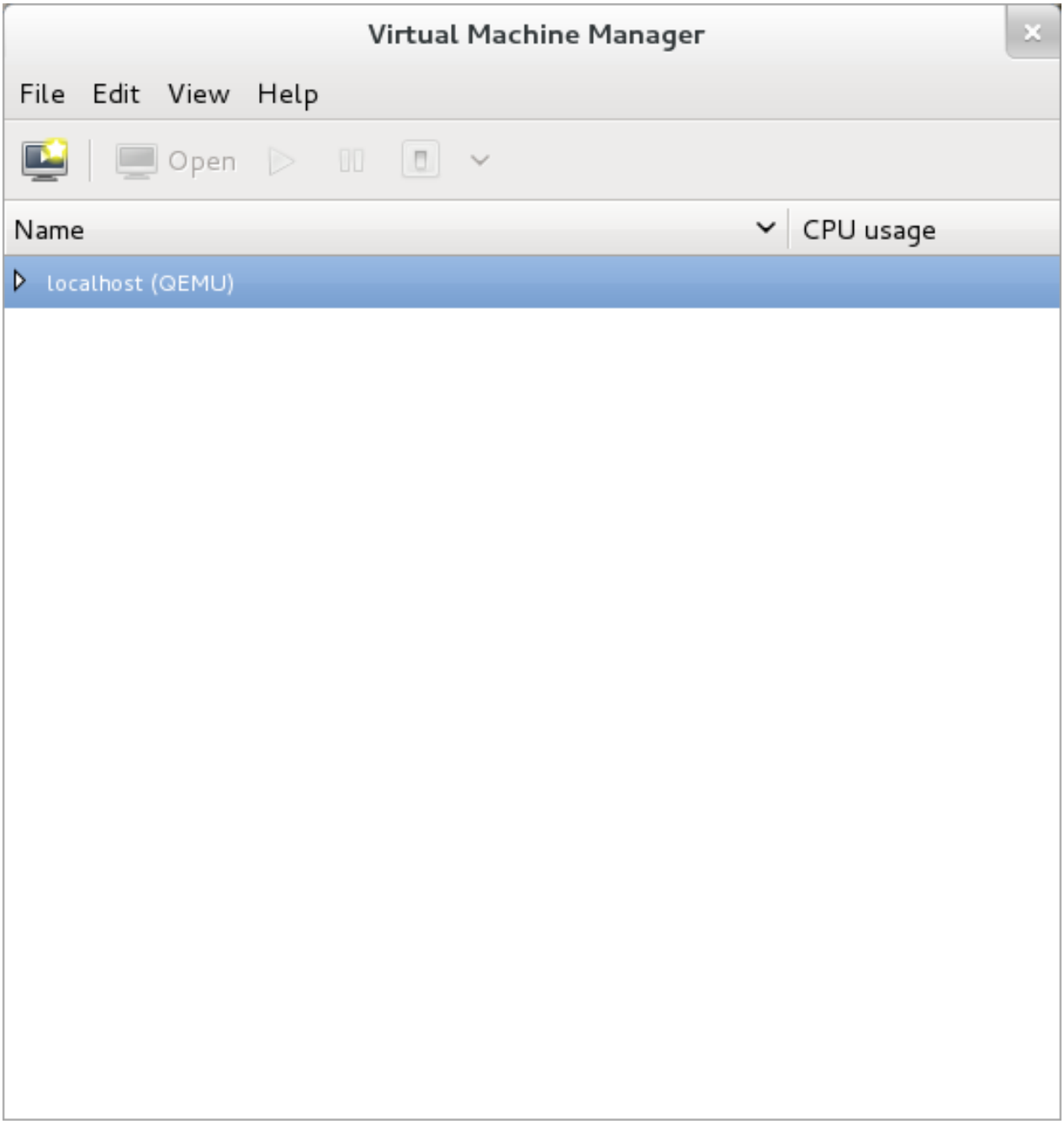


図6.15 virt-manager のメインウィンドウ

新しい仮想マシンを作成 ボタンをクリックして、新しい仮想ゲストのウィザードを開始します。

図6.16 「新しい仮想マシンを作成」ボタン

新しい仮想マシン ウィンドウが開きます。

3. **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 (-) 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ず、数字のみの名前は使用できません。

   ラジオボタンの一覧からインストール方法を選択します。

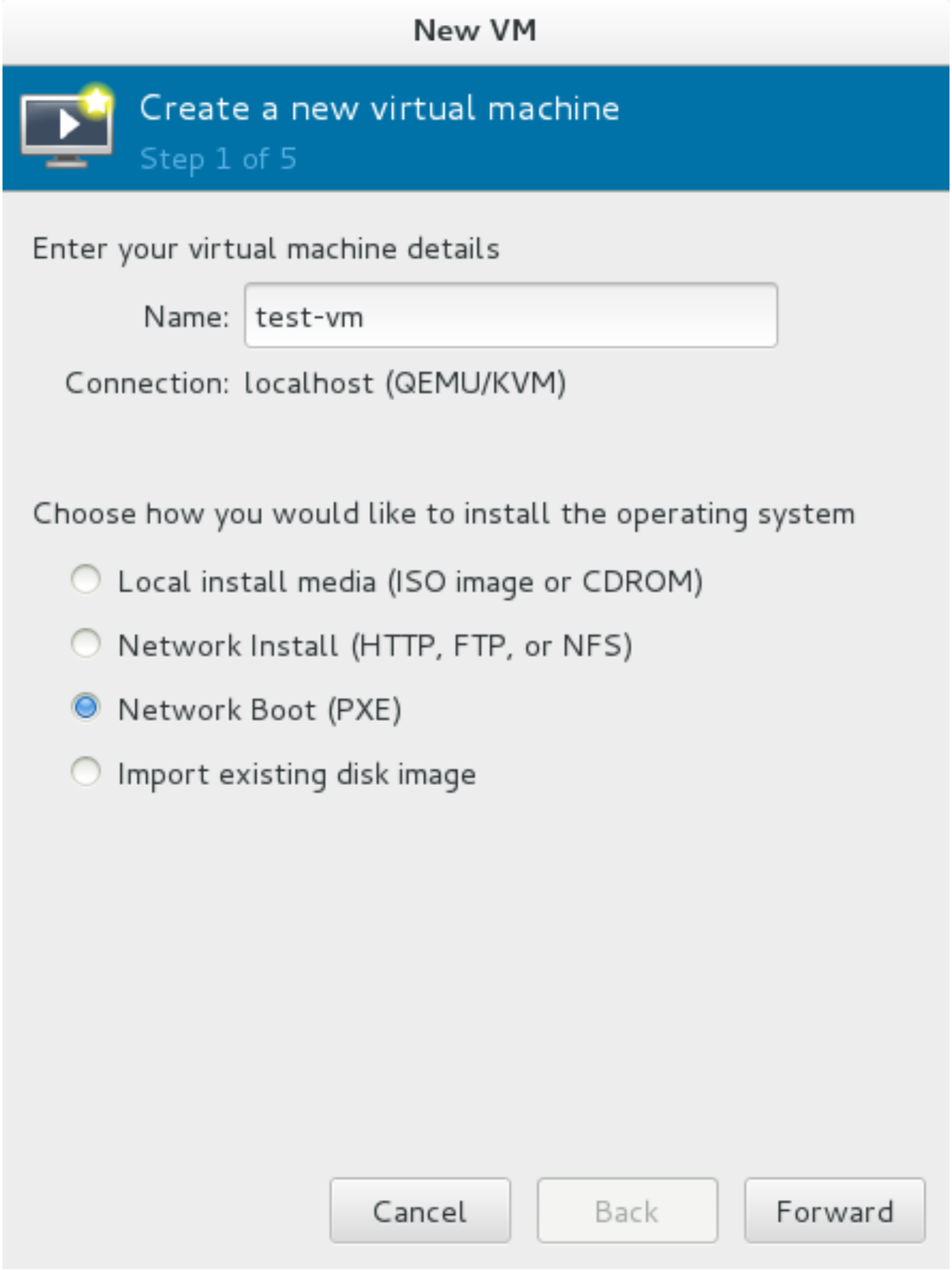


図6.17 「新しい仮想マシン」ウィンドウ - ステップ 1

進む をクリックして次に進みます。

4. 残りのステップは、ISO インストール手順と同じです。ISO インストール手順の ステップ 5 以降を実行します。これ以降の PXE 手順が ISO 手順と異なるのは、最後の **新しい仮想マシン** 画面で **インストール： PXE インストール** フィールドがある点のみです。

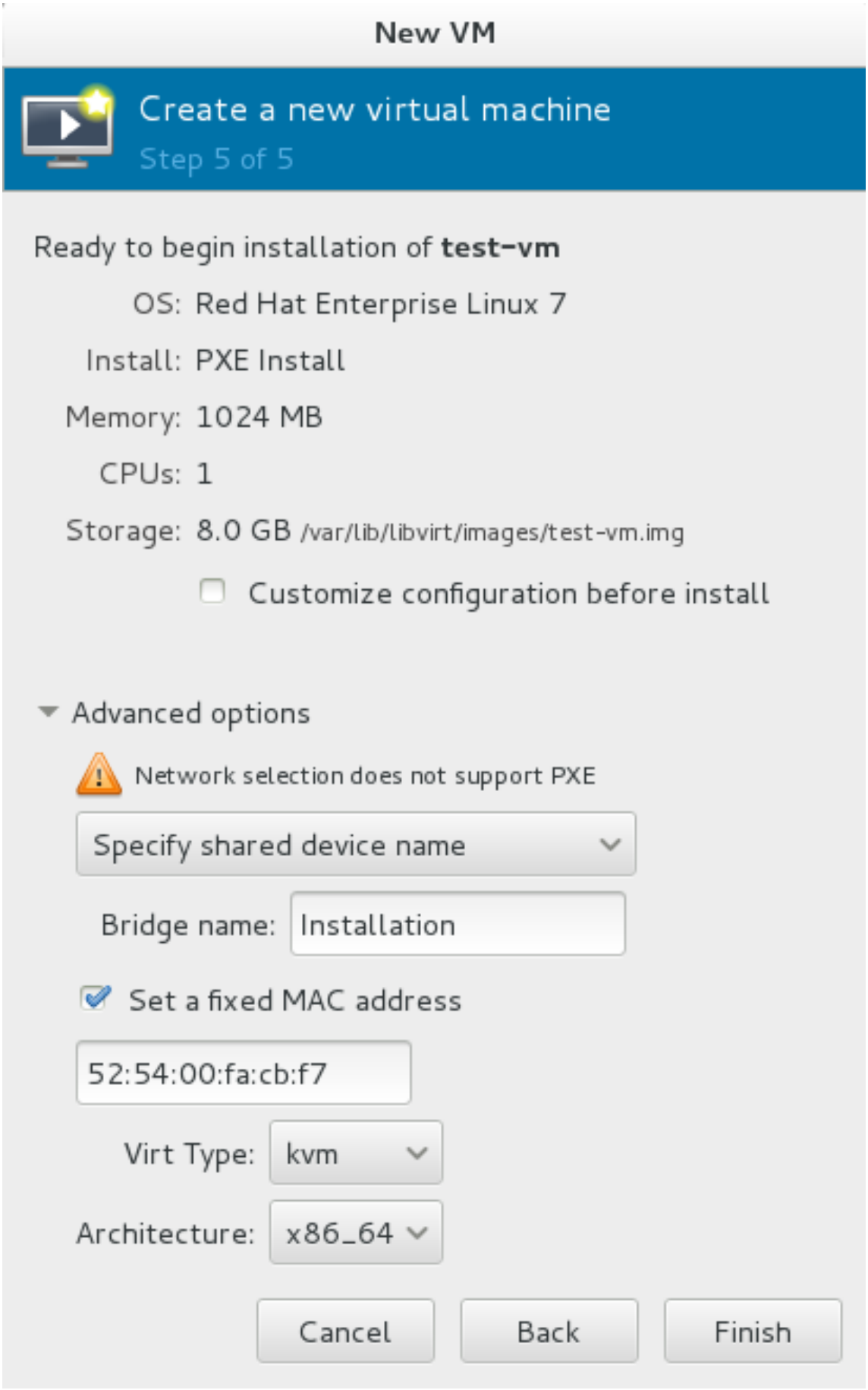


図6.18 「新しい仮想マシン」ウィンドウ - ステップ 5 - PXE インストール

第7章 他のプラットフォーム上での Red Hat Enterprise Linux の仮想化

この章では、他の仮想化ホスト上で Red Hat Enterprise Linux 7 を仮想化オペレーティングシステムとして実行する際に役立つ参考資料について記載しています。

7.1. VMware ESX

Red Hat Enterprise Linux 7 は以下のドライバーを提供しています。

- **vmw_balloon** - これは、VMware ホスト上で Red Hat Enterprise Linux を実行する際に使用される準仮想化メモリーバルーニングドライバーです。このドライバーについての詳細は、http://kb.VMware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=1002586 を参照してください。
- **vmmouse_drv** - これは、VMware ホスト上で Red Hat Enterprise Linux を実行する際に使用される準仮想化マウスドライバーです。このドライバーについての詳細は、http://kb.VMware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=5739104 を参照してください。
- **vmware_drv** - これは、VMware ホスト上で Red Hat Enterprise Linux を実行する際に使用される準仮想化ビデオドライバーです。このドライバーについての詳細は、http://kb.VMware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=1033557 を参照してください。
- **vmxnet3** - これは、VMware ホスト上で Red Hat Enterprise Linux を実行する際に使用される準仮想化ネットワークアダプターです。このドライバーについての詳細は、http://kb.VMware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1001805 を参照してください。
- **vmw_pvscsi** - これは、VMware ホスト上で Red Hat Enterprise Linux を実行する際に使用される準仮想化 SCSI アダプターです。このドライバーについての詳細は、http://kb.VMware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1010398 を参照してください。

7.2. Hyper-V

Red Hat Enterprise Linux 7 には、Microsoft の Linux Integration Services が同梱されています。これは、対応する仮想化オペレーティングシステムで統合デバイスサポートを有効にするドライバーセットです。このドライバーについての詳細は、http://technet.microsoft.com/en-us/library/dn531030.aspx を参照してください。

以下の機能強化により、Hyper-V ハイパーバイザーへの Red Hat Enterprise Linux ゲスト仮想マシンの導入と管理がより簡単になりました。

- アップグレードされた VMBUS プロトコル - VMBUS プロトコルが Windows 8 レベルにアップグレードされました。この機能強化の一環として、VMBUS 割り込みがゲストの利用可能なすべての仮想 CPU で処理できるようになりました。さらに、Red Hat Enterprise Linux ゲスト仮想マシンと Windows ホスト物理マシン間のシグナルプロトコルが最適化されています。
- 統合フレームバッファードライバー - グラフィックスパフォーマンスを強化し、Red Hat Enterprise Linux デスクトップユーザーに対して優れた解像度を提供します。
- ライブ仮想マシンバックアップサポート - ライブ Red Hat Enterprise Linux ゲスト仮想マシンの中断なしのバックアップサポートを提供します。

- 固定サイズの Linux VHDX を動的に拡張 - ライブのマウントされている固定サイズの Red Hat Enterprise Linux VHDX の拡張を可能にします。
- UEFI を使用した起動 - UEFI (Unified Extensible Firmware Interface) を使用して Hyper-V 2012 R2 ホスト上で仮想マシンを起動できるようになりました。

詳細は、以下の記事を参照してください。 Enabling Linux Support on Windows Server 2012 R2 Hyper-V

第8章 完全仮想化 Windows ゲストのインストール

この章では、コマンドライン (**virt-install**) を使用して完全仮想化 Windows ゲストを作成する方法、ゲスト内でオペレーティングシステムのインストーラーを起動する方法、および **virt-viewer** でインストーラーにアクセスする方法を説明します。

ゲストに Windows オペレーティングシステムをインストールするには、**virt-viewer** ツールを使用します。このツールは、仮想マシンのグラフィカルコンソールを (VNC プロトコル経由で) 表示します。これにより、**virt-viewer** はオペレーティングシステムのインストーラー (例: Windows 8 インストーラー) で完全仮想化ゲストのオペレーティングシステムをインストールできるようになります。

Windows オペレーティングシステムのインストールには、2 つのステップがあります。

1. **virt-install** または **virt-manager** を使用してゲスト仮想マシンを作成する。
2. **virt-viewer** を使用してゲスト仮想マシン上に Windows オペレーティングシステムをインストールする。

virt-install または **virt-manager** を使用したゲスト仮想マシンの作成方法については、*5章ゲスト仮想マシンのインストール概要* を参照してください。

この章では、完全仮想化ゲスト上での Windows オペレーティングシステムのインストール方法を説明するのではなく、ゲストの作成方法とゲスト内でインストーラーを起動する方法を説明します。Windows オペレーティングシステムのインストール方法については、関連する Microsoft のインストールについての説明書を参照してください。

8.1. virt-install を使用したゲストの作成

virt-install コマンドを使うと、たとえば GUI を使わずにターミナルから完全仮想化ゲストを作成することができます。

重要

ゲストの作成前に、まずゲストで KVM Windows 準仮想化 (virtio) ドライバーを使用する必要があるかどうかを検討してください。必要がある場合は、Windows オペレーティングシステムのゲストへの **インストール時** か、または **インストール後** にこのドライバーが使用可能であることに留意してください。virtio ドライバーについての詳細は、*9章KVM 準仮想化 (virtio) ドライバー* を参照してください。

KVM virtio ドライバーのインストール方法については、「KVM Windows virtio ドライバーのインストール」 を参照してください。

1 つのコマンドで完全仮想化ゲストを作成することも可能です。これを実行するには、以下のプログラムを実行します (必要に応じて値を変更します) 。

```
# virt-install \
   --name=guest-name \
   --os-type=windows \
   --network network=default \
   --disk path=path-to-disk,size=disk-size \
   --cdrom=path-to-install-disk \
   --graphics spice --ram=1024
```

path-to-disk はデバイス (例 **/dev/sda3**) またはイメージファイル (**/var/lib/libvirt/images/*name*.img**) である必要があります。***disk-size*** に対応する十分な空き領域も必要になります。

path-to-install-disk は、ISO イメージへのパスか、または最小限の起動用 ISO イメージのアクセス先となる URL である必要があります。

重要

すべてのイメージファイルは、デフォルトで **/var/lib/libvirt/images/** に保存されます。ファイルベースのイメージは他のディレクトリーでも保存可能ですが、SELinux 設定が必要な場合もあります。SELinux を強制 (enforcing) モードで実行する場合の詳細は、『Red Hat Enterprise Linux SELinux User's and Administrator's Guide』 を参照してください。

完全仮想化ゲストが作成されると、**virt-viewer** がゲストを起動し、オペレーティングシステムのインストーラーを実行します。オペレーティングシステムのインストール方法については、関連する Microsoft のインストールについての説明書を参照してください。

8.2. Windows ゲスト仮想マシンの使用における効率改善のヒント

Windows ゲスト仮想マシンを効率的に機能させるには、以下のフラグを *libvirt* で設定する必要があります。

- hv_relaxed
- hv_spinlocks=0x1fff
- hv_vapic
- hv_time

8.2.1. Hyper-V クロックフラグの設定

Hyper-V クロックフラグを設定するには、以下を組み込むように Windows ゲスト仮想マシン XML を拡張します。

```
<domain type='kvm'>
...
  <clock offset='utc'>
    <timer name='hypervclock' present='yes'/>
  </clock>
...
</domain>
```

図8.1 クロック要素 XML

このアクションは、ゲスト仮想マシンが実行中の場合は実行することができません。ゲスト仮想マシンをシャットダウンし、XML ファイルを変更してから、ゲスト仮想マシンを再起動します。

第9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 Windows 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 4.8 およびそれ以降
- 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 カーネルおよびそれ以降の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。

Red Hat Enterprise Linux 3 (3.9 およびそれ以降) では、手動のインストールが必要になります。

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。割り当てデバイス使用時の追加の制限については、20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。

KVM virtio ドライバーを使用すると、以下の Microsoft Windows バージョンがベアメタルベースのシステムと同様の動作をすることが予想されます。

- Windows XP Service Pack 3 およびそれ以降 (32 ビットのみ)
- Windows Server 2003 (32 ビットおよび 64 ビット)
- Windows Server 2008 (32 ビットおよび 64 ビット)
- Windows Server 2008 R2 (64 ビットのみ)
- Windows 7 (32 ビットおよび 64 ビット)
- Windows Server 2012 (64 ビットのみ)
- Windows Server 2012 R2 (64 ビットのみ)
- Windows 8 (32 ビットおよび 64 ビットバージョン)
- Windows 8.1 (32 ビットおよび 64 ビットバージョン)

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。

9.1. KVM Windows virtio ドライバーのインストール

このセクションでは、KVM Windows virtio ドライバーのインストールプロセスを説明します。KVM virtio ドライバーは、Windows のインストール時にロードしたり、ゲストのインストール後にインストールしたりすることができます。

virtio ドライバーは、以下のいずれかの方法でゲスト仮想マシンにインストールできます。

- 仮想マシンにアクセス可能なネットワーク上のインストールファイルをホスト
- ドライバーインストールディスク .iso ファイルの仮想化 CD-ROM デバイスを使用
- CD-ROM に使用するのと同じ (提供される) .ISO ファイルをマウントして USB ドライブを使用
- 起動時に仮想化フロッピーデバイスを使用してドライバーをインストール (XP/2003 にのみ必須および推奨)

本書では、仮想化 CD-ROM デバイスを使用した準仮想化インストールディスクからのインストールについて説明します。

1.

ドライバーのダウンロード

virtio-win パッケージには、サポート対象のすべての Windows ゲスト仮想マシン用の virtio ブロックおよびネットワークドライバーが含まれます。

注記

virtio-win パッケージは、https://access.redhat.com/downloads/content/rhel---7/x86_64/2476/virtio-win/1.7.1-1.el7/noarch/fd431d51/package にあります。以下のいずれかのチャンネルへのアクセスが必要になります。

- RHEL Client Supplementary (v. 7)
- RHEL Server Supplementary (v. 7)
- RHEL Workstation Supplementary (v. 7)

yum コマンドを実行して、*virtio-win* パッケージをダウンロードし、これをインストールします。

```
# yum install virtio-win
```

Windows オペレーティングシステム上でサポートされる virtio-win パッケージと現行の認証済みパッケージバージョンの一覧は、windowsservercatalog.com で確認できます。

Red Hat Enterprise Virtualization Hypervisor と Red Hat Enterprise Linux は同じコードベース上で作成されているため、同じバージョンのドライバー (例: Red Hat Enterprise Virtualization Hypervisor 3.3 と Red Hat Enterprise Linux 6.5) は両方の環境でサポートされます。

virtio-win パッケージは、**/usr/share/virtio-win/** ディレクトリーに CD-ROM イメージの **virtio-win.iso** をインストールします。

2.

virtio ドライバーのインストール

virtio-win デバイスを使用する Windows ゲストを起動する場合、関連する virtio-win デバイスドライバーをこのゲストにインストールしておく必要があります。virtio-win ドライバーは Microsoft Windows インストールキットのインボックスドライバーとして提供されていないため、virtio-win ストレージデバイス (viostor/virtio-scsi) に Windows ゲストをインストールするには、インストール**時** に、**virtio-win.iso** から直接か、または提供される仮想フロッピーイメージ**virtio-win<*version*>.vfd** のいずれかにより適切なドライバーを指定する必要があります。

9.2. インストール済み Windows ゲスト仮想マシンへのドライバーのインストール

以下の手順では、Windows のインストール後に仮想化 CD-ROM を使用して virtio ドライバーをインストールする方法を説明します。

virt-manager を使って CD-ROM イメージを追加してからドライバーをインストールする方法については、以下の手順にしたがってください。

手順9.1 virt-manager を使用してドライバー CD-ROM イメージからインストールする

1. **virt-manager を開き、ゲスト仮想マシンを開始します。**

   virt-manager を開き、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. **ハードウェアウィンドウを開きます。**

   ウィンドウ最上部のツールバーにある電球のアイコンをクリックして仮想ハードウェアの詳細を表示します。

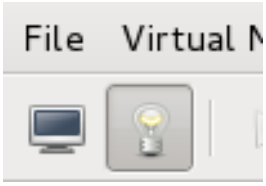


図9.1 仮想ハードウェア詳細ボタン

新たなビューの最下部にある**ハードウェアを追加** ボタンをクリックします。

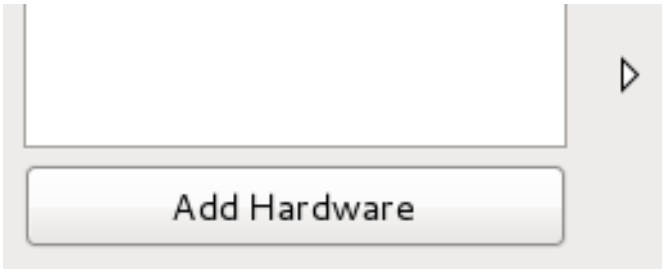


図9.2 仮想マシンのハードウェア情報ウィンドウ

新規デバイスを追加するためのウィザードが開きます。

3. **ISO ファイルを選択します。**

   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 ラジオボタンを選択し、virtio ドライバーの **.iso** イメージファイルを参照します。デフォルトでは、ドライバーの最新バージョンは、**/usr/share/virtio-win/virtio-win.iso** にあります。

   デバイスの種類 を **IDE cdrom** に変更し、 **進む** ボタンをクリックして次に進みます。

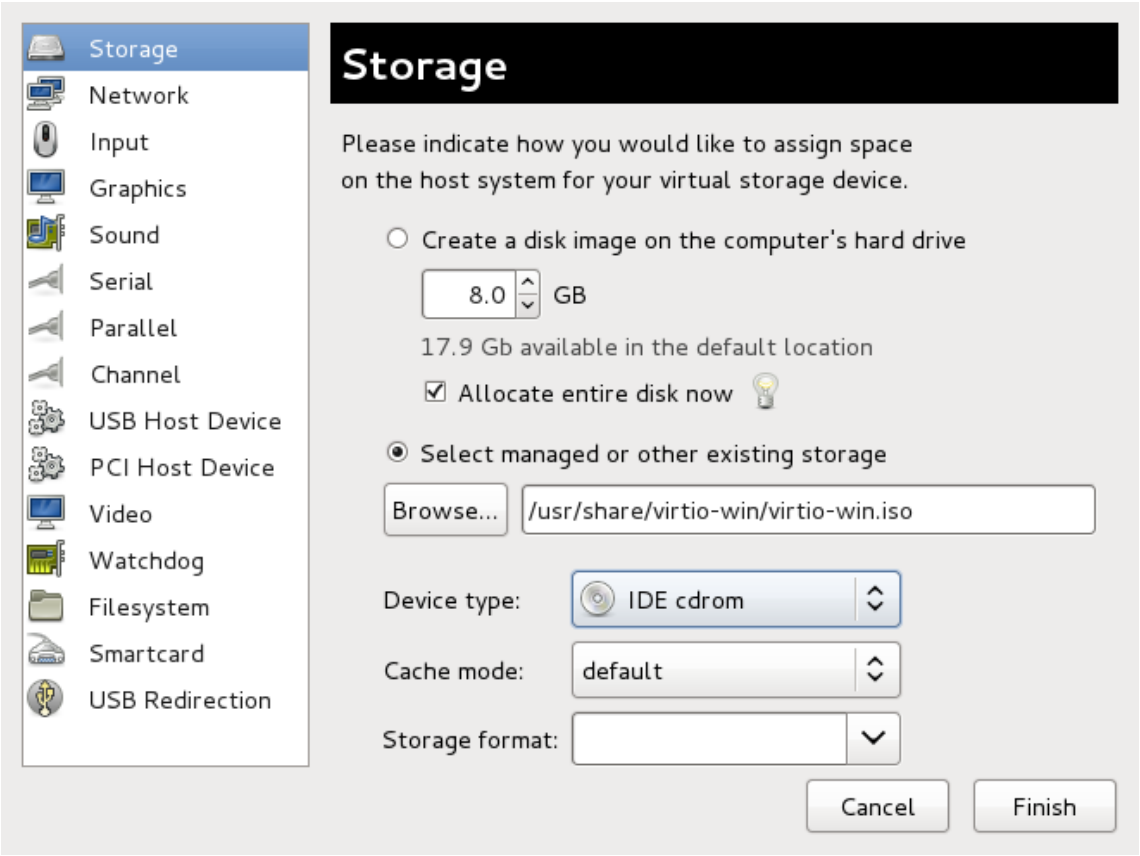


図9.3 新たなハードウェア追加ウィザード

4. **再起動します。**

   ドライバーディスクを使用して仮想マシンを開始または再起動します。仮想マシンが新たなデバイスを認識できるようにするには、仮想化 IDE デバイスの再起動が必要です。

ドライバーのCD-ROM が割り当てられ、仮想マシンが開始したら、手順9.2「Windows 7 仮想マシン上への Windows インストール」 に進みます。

手順9.2 Windows 7 仮想マシン上への Windows インストール

この手順では、ドライバーを Windows 7 仮想マシンにインストールする例を取り挙げます。ご自分の Windows ゲストのバージョンに合わせて Windows のインストール指示にしたがってください。

1. **コンピューターの管理ウィンドウを開きます。**

   Windows 仮想マシンのデスクトップ上で、画面下部左端にある **Windows** アイコンをクリックし、スタートメニューを開きます。

   コンピューター を右クリックし、ポップアップメニューから **管理** を選択します。

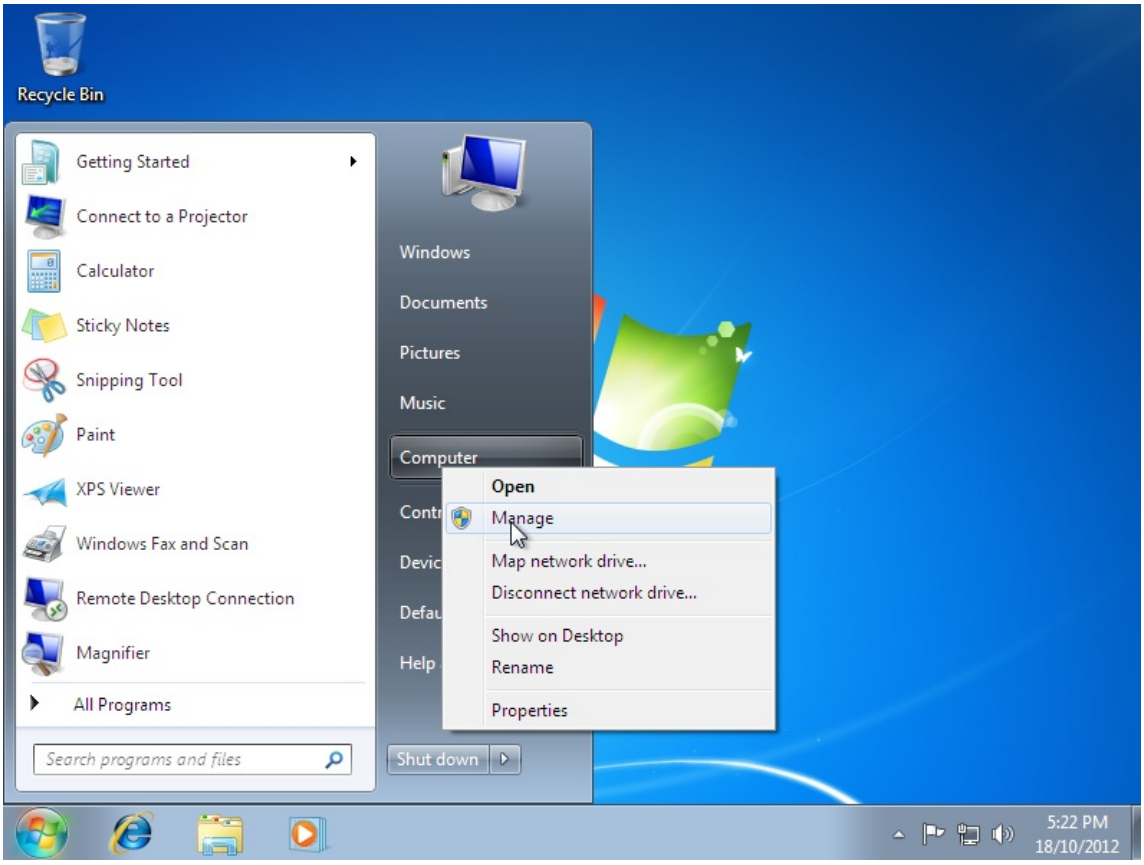


図9.4 コンピューターの管理ウィンドウ

2. **デバイスマネージャーを開く**

   左端のペインから **デバイスマネージャー** を選択します。これは、**コンピューターの管理** > **システムツール** にあります。

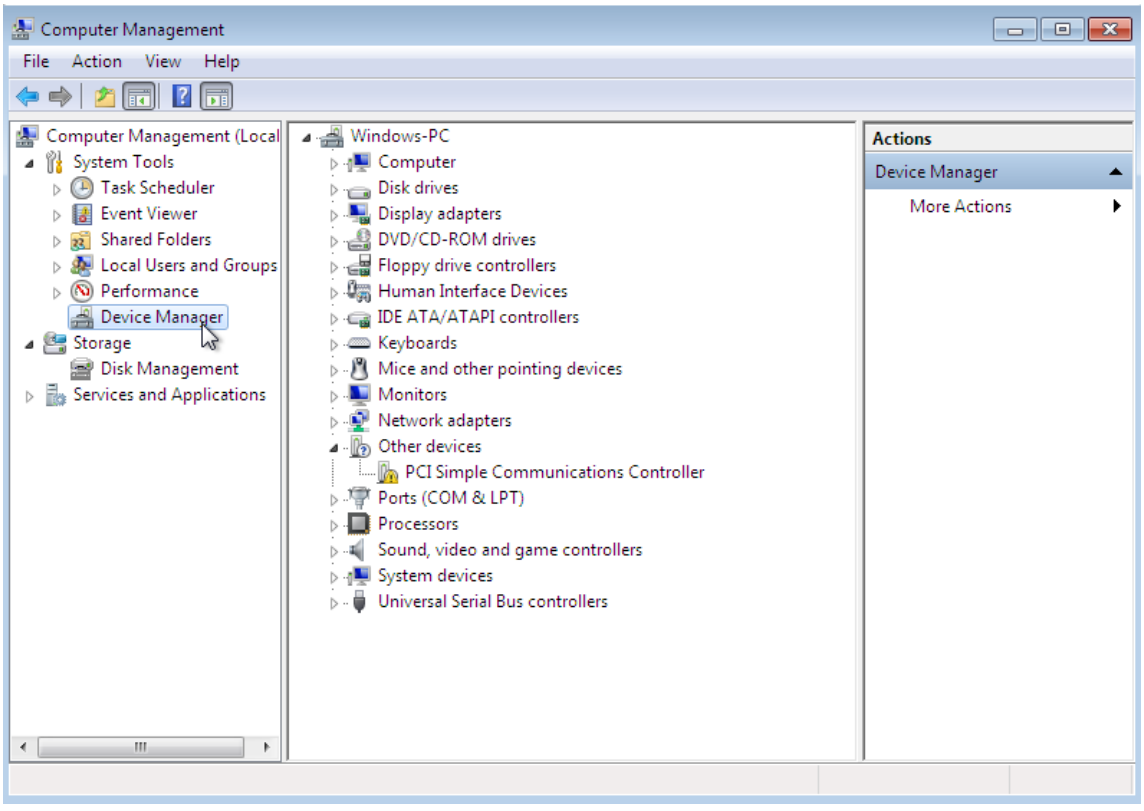


図9.5 コンピューターの管理ウィンドウ

3. **ドライバーアップデートウィザードの開始**

   a. **利用可能なシステムデバイスの表示**

   システムデバイス の左にある矢印をクリックして展開します。

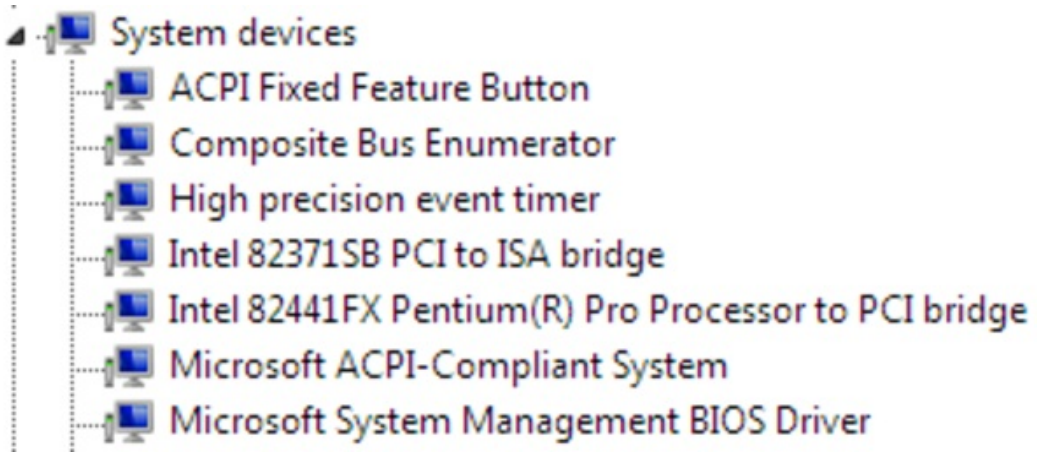


図9.6 コンピューターの管理ウィンドウで利用可能なシステムデバイスを表示

   b. **適切なデバイスの特定**

   最大で 以下の4 種類のドライバーが利用可能です。バルーンドライバー、シリアルドライバー、ネットワークドライバー、ブロックドライバー。

   - **Balloon**: このバルーンドライバーは**システムデバイス** グループの**PCI 標準 RAM コントローラー** に影響します。
   - **vioserial**: このシリアルドライバーは、**システムデバイス** グループの**PCI シンプル通信コントローラー** に影響します。
   - **NetKVM**: このネットワークドライバーは、**ネットワークアダプター** グループに影響します。このドライバーは、virtio NIC が設定されている場合のみ利用可能です。このドライバーの設定可能パラメーターは、付録C *NetKVM ドライバーパラメーター*で説明されています。
   - **viostor**: このブロックドライバーは、**ディスクドライブ** グループに影響します。このドライバーは、virtio ディスクが設定されている場合のみ、利用可能です。

   アップデートするドライバーのデバイスを右クリックし、ポップアップメニューから **ドライバーソフトウェアの更新...** を選択します。

   この例ではバルーンドライバーをインストールするので、**PCI 標準 RAM コントローラー** を右クリックします。

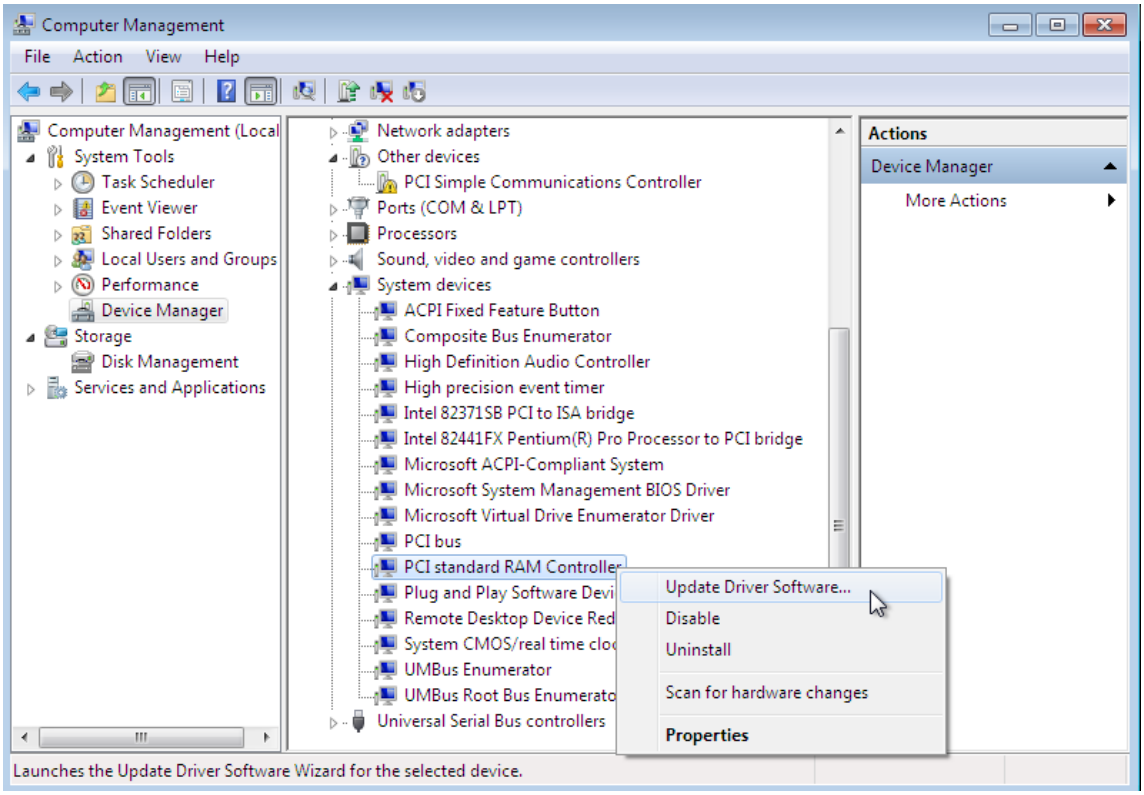


図9.7 コンピューターの管理ウィンドウ

c. **ドライバー更新ウィザードの開始**

ドロップダウンメニューから**ドライバーソフトウェアの更新 . . .** を選択し、ドライバー更新ウィザードにアクセスします。

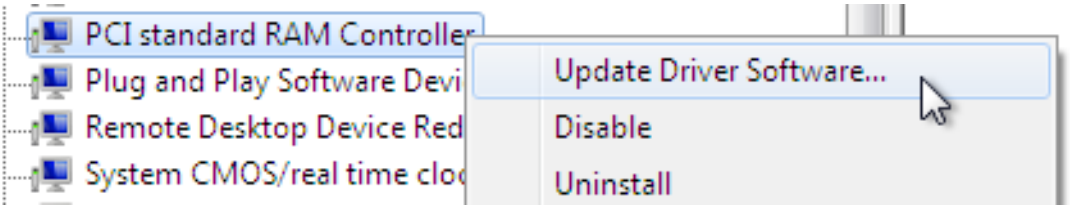


図9.8 ドライバー更新ウィザードの開始

4. **ドライバー検索方法の指定**

ドライバー更新ウィザードの最初のページでは、ドライバーソフトウェアの検索方法を聞かれます。2 つ目のオプション、**コンピューターを参照してドライバーソフトウェアを検索します**をクリックします。

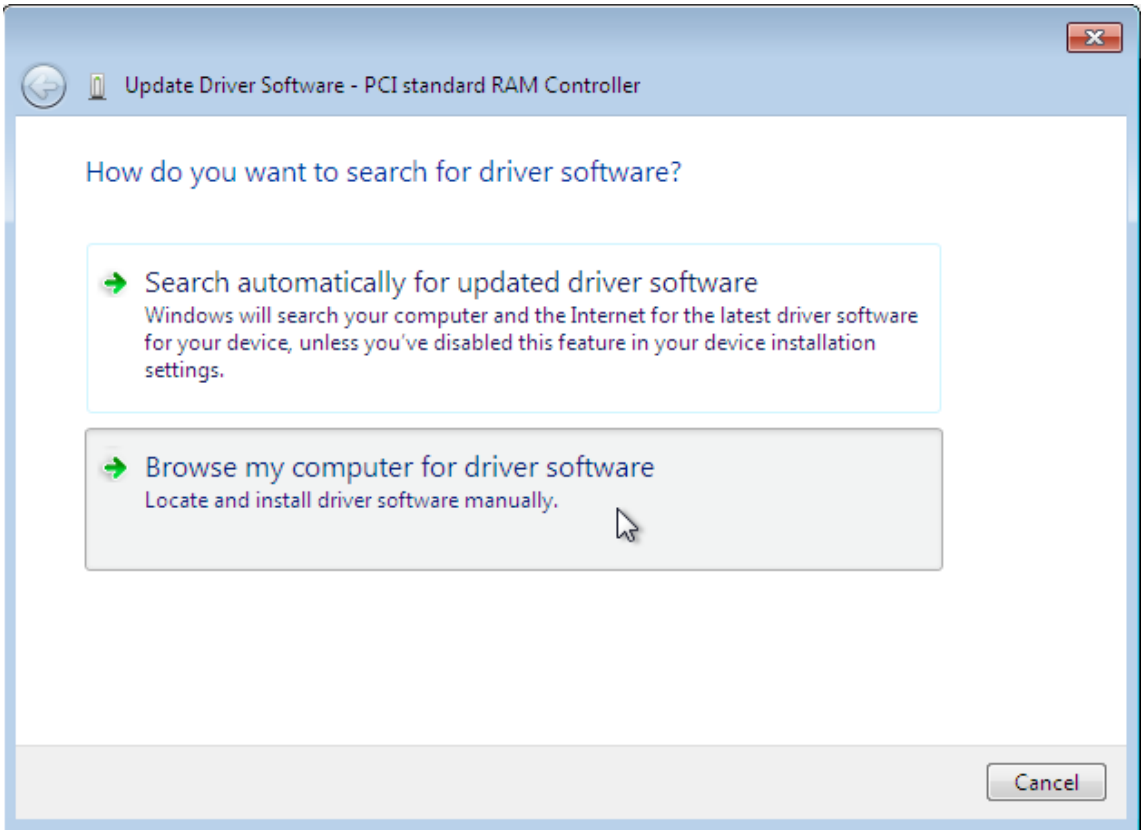


図9.9 ドライバー更新のウィザード

5. **インストールするドライバーの選択**

   a. **ファイルのブラウザーを開きます。**

      参照 . . . をクリックします。

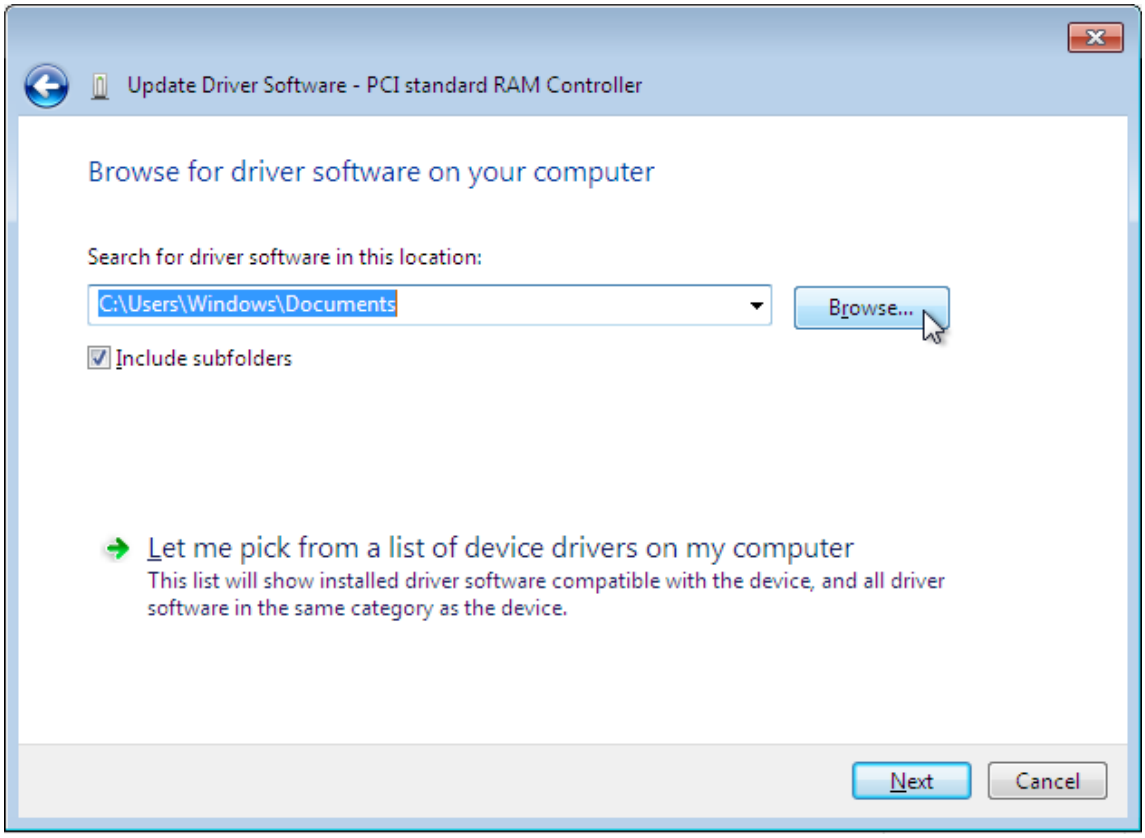


図9.10 ドライバー更新のウィザード

b. **ドライバーの場所を参照**

オペレーティングシステムとアーキテクチャーの組み合わせごとに、別のドライバーが提供されます。ドライバーは以下のように、ドライバータイプ、オペレーティングシステム、インストールされるアーキテクチャーという階層で配置されます。***driver_type/os/arch/*** 。例えば、 x86 (32 ビット) アーキテクチャーの Windows 7 オペレーティングシステム用バルーンドライバーは、**Balloon/w7/x86** ディレクトリーに配置されます。

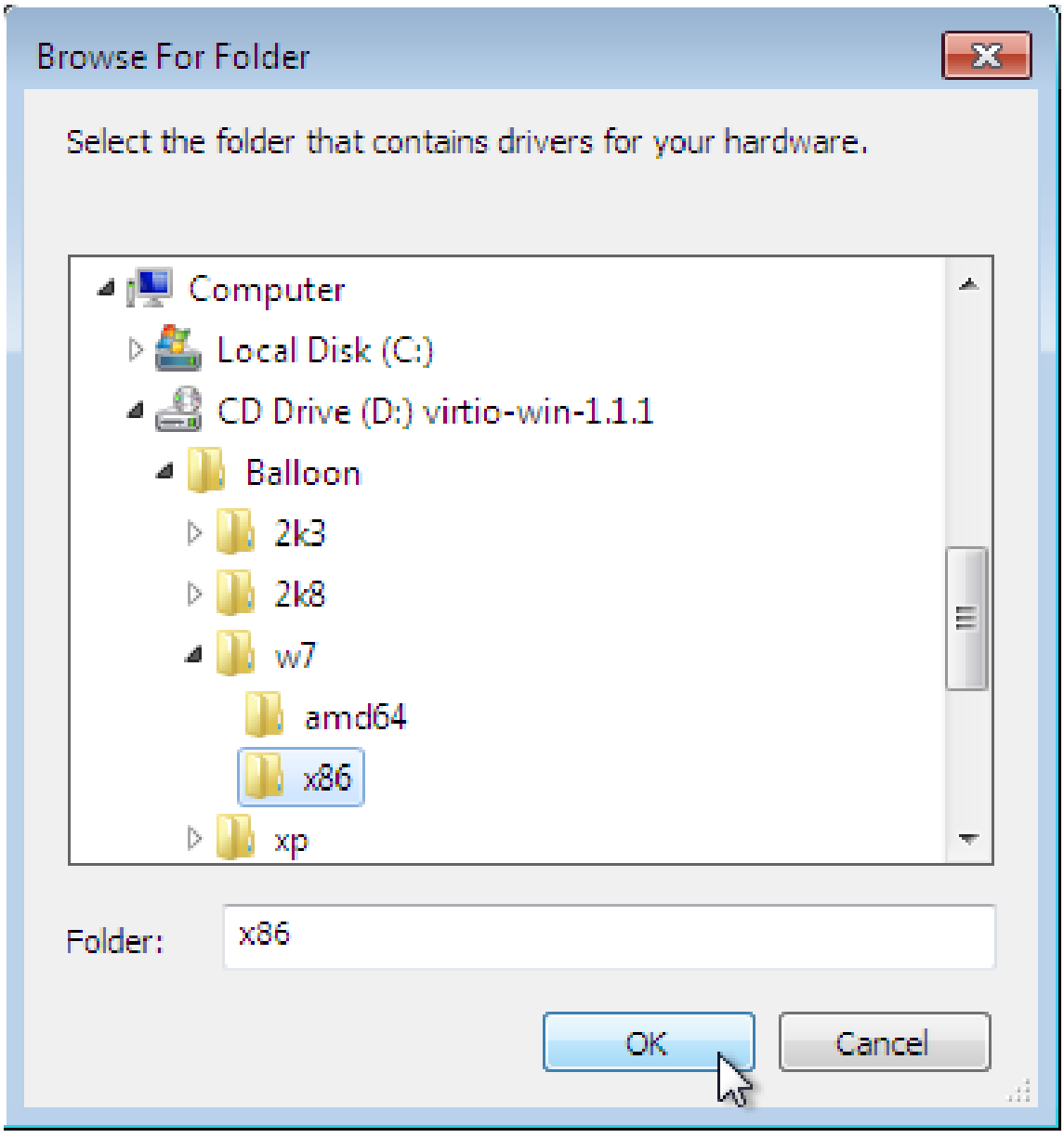


図9.11 ドライバーソフトウェア参照のポップアップウィンドウ

正しい場所まで移動したら、**OK** をクリックします。

c. **次へ をクリックして続ける**

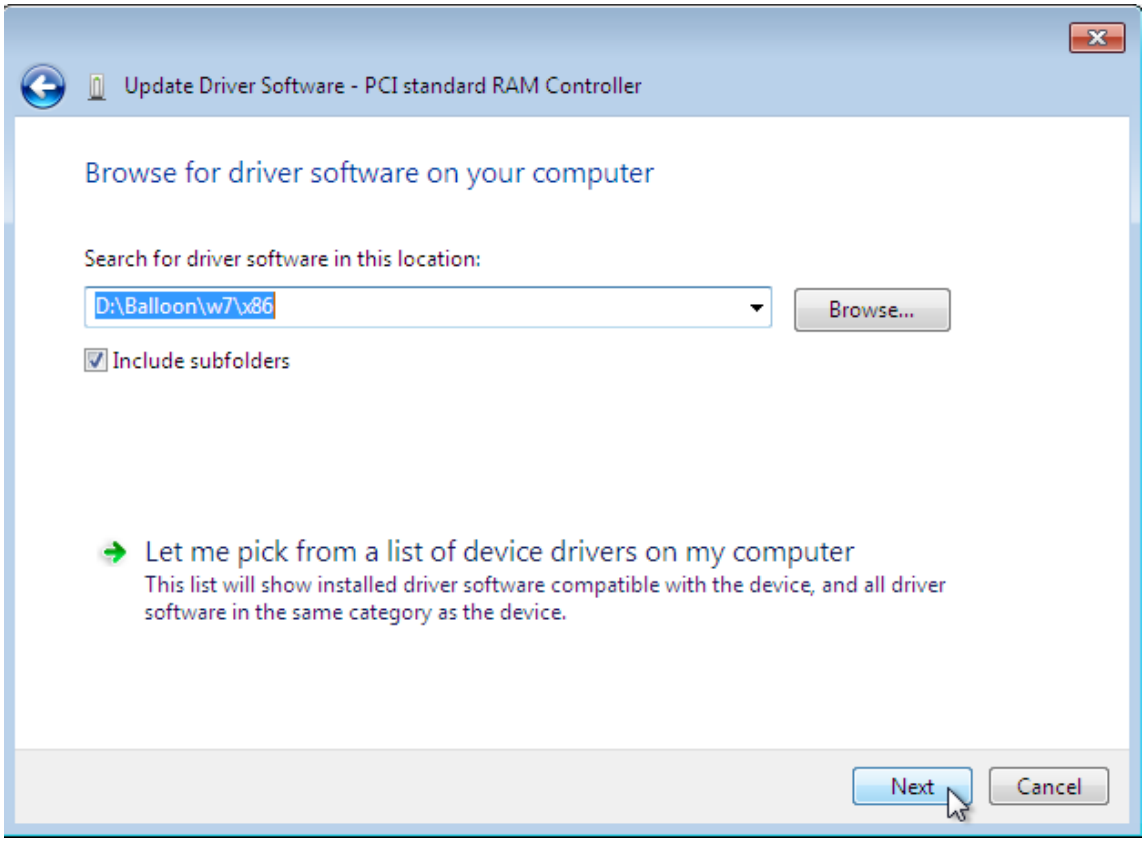


図9.12 ドライバーソフトウェア更新ウィザード

ドライバーのインストール中に以下の画面が表示されます。

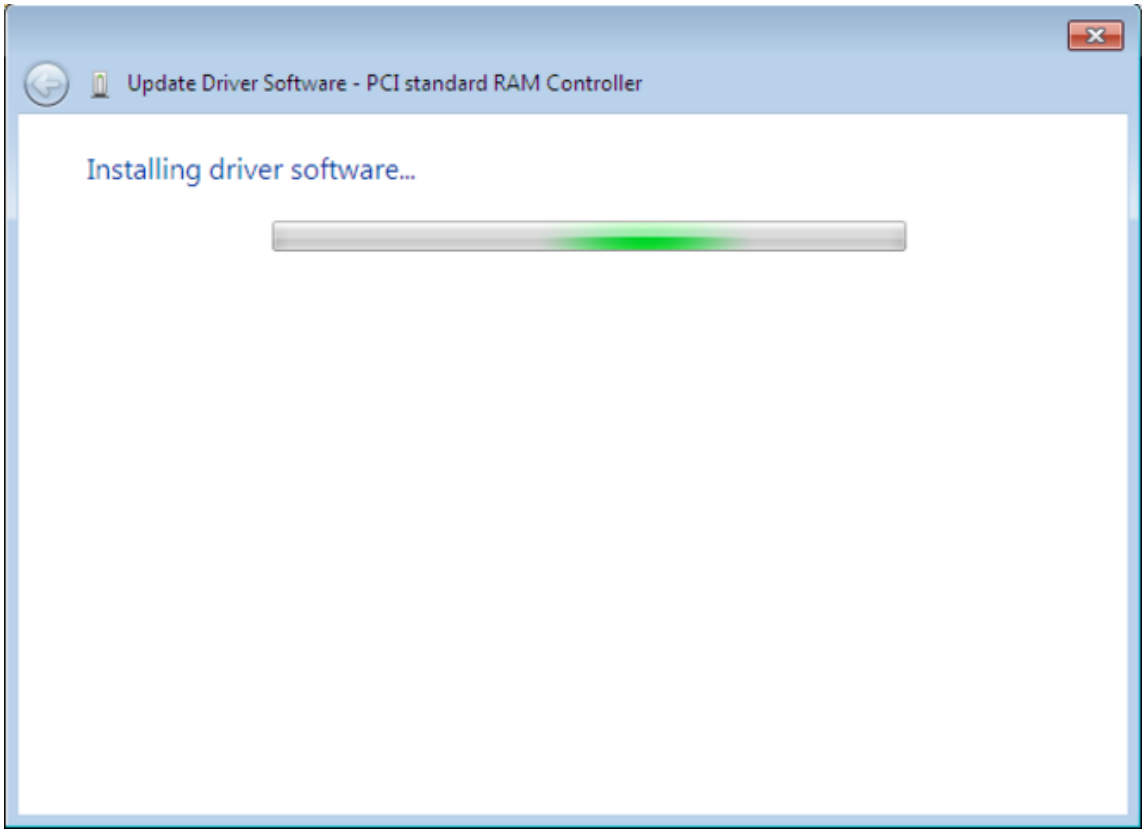


図9.13 ドライバーソフトウェア更新ウィザード

6. **インストーラーを閉じる**

   インストールが完了すると、以下の画面が表示されます。

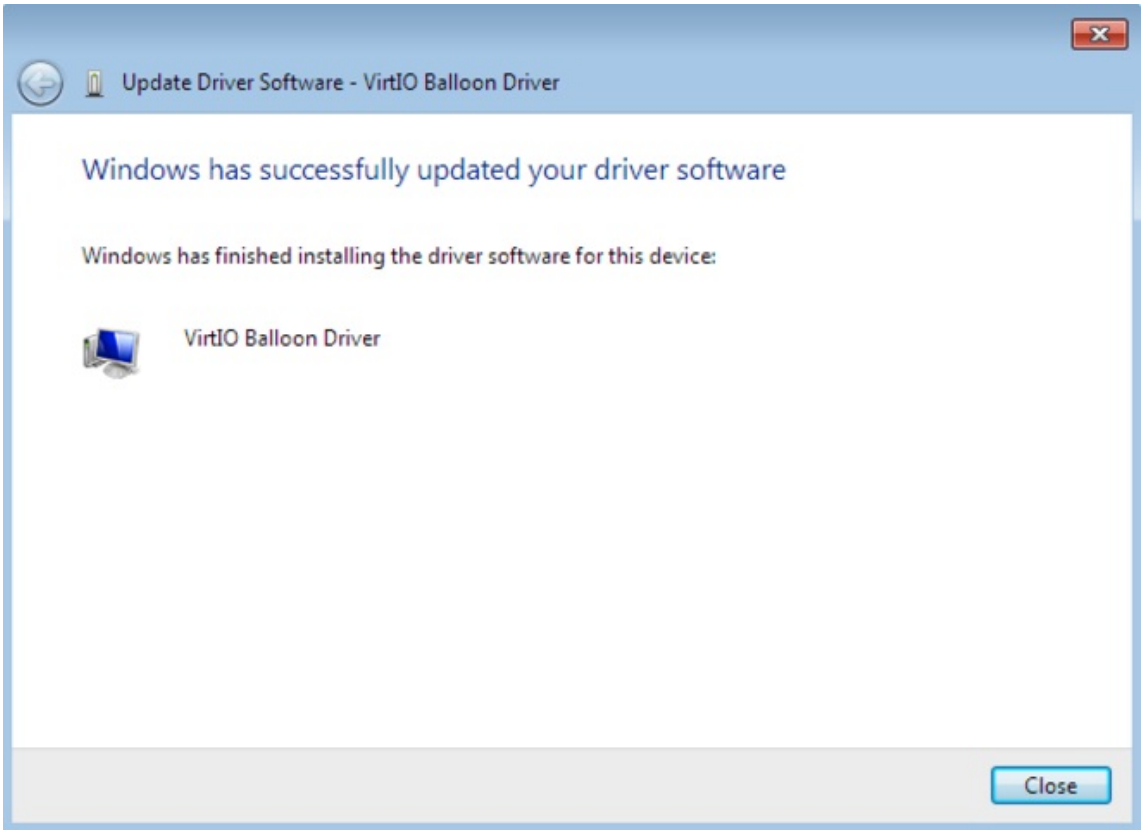


図9.14 ドライバーソフトウェア更新ウィザード

閉じる ボタンをクリックして、インストーラーを閉じます。

7. **再起動します。**

   仮想マシンを再起動してドライバーのインストールを完了します。

9.3. Windows インストール中のドライバーのインストール

ここでは、Windows インストール中に virtio ドライバーをインストールする方法を説明します。

この方法では、Windows ゲスト仮想マシンは virtio ドライバーをデフォルトのストレージデバイスに使用できます。

手順9.3 Windows インストール時の virtio ドライバーのインストール

1. virtio-win パッケージをインストールします。

```
# yum install virtio-win
```

注記

virtio-win パッケージは、https://access.redhat.com/downloads/content/rhel---7/x86_64/2476/virtio-win/1.7.1-1.el7/noarch/fd431d51/package にあります。以下のいずれかのチャンネルへのアクセスが必要になります。

- RHEL Client Supplementary (v. 7)
- RHEL Server Supplementary (v. 7)
- RHEL Workstation Supplementary (v. 7)

2. **ゲスト仮想マシンの作成**

重要

仮想マシンを開始せずに、通常どおりに仮想マシンを作成します。以下のいずれかの手順にしたがいます。

以下のゲスト作成方法のうち、どれか **一つ** を選び、指示にしたがいます。

a. **virsh を使ったゲスト仮想マシンの作成**

この方法では、インストール **前に** virtio ドライバーフロッピーディスクを Windows ゲストに割り当てます。

virsh で XML 定義ファイルから仮想マシンが作成された場合、**virsh create** コマンドではなく、**virsh define** コマンドを使用してください。

i. 仮想マシンを作成します。ただし、仮想マシンを開始しないでください。**virsh** コマンドを使用した仮想マシンの作成に関する詳細は、『Red Hat Enterprise Linux 仮想化管理ガイド』 を参照してください。

ii. **virsh** コマンドでドライバーディスクを仮想化フロッピーディスクとして追加します。ゲスト仮想マシンに他の仮想化フロッピーディスクが割り当てられていない場合に、この例を適用することができます。*vm_name* は仮想マシン名と置き換えることに注意してください。

```
# virsh attach-disk vm_name /usr/share/virtio-
win/virtio-win.vfd fda --type floppy
```

ステップ 3 に進みます。

b. **virt-manager を使用してゲスト仮想マシンを作成し、ディスクの種類を変更する**

i. virt-manager ゲスト作成ウィザードの最後のステップで、**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。

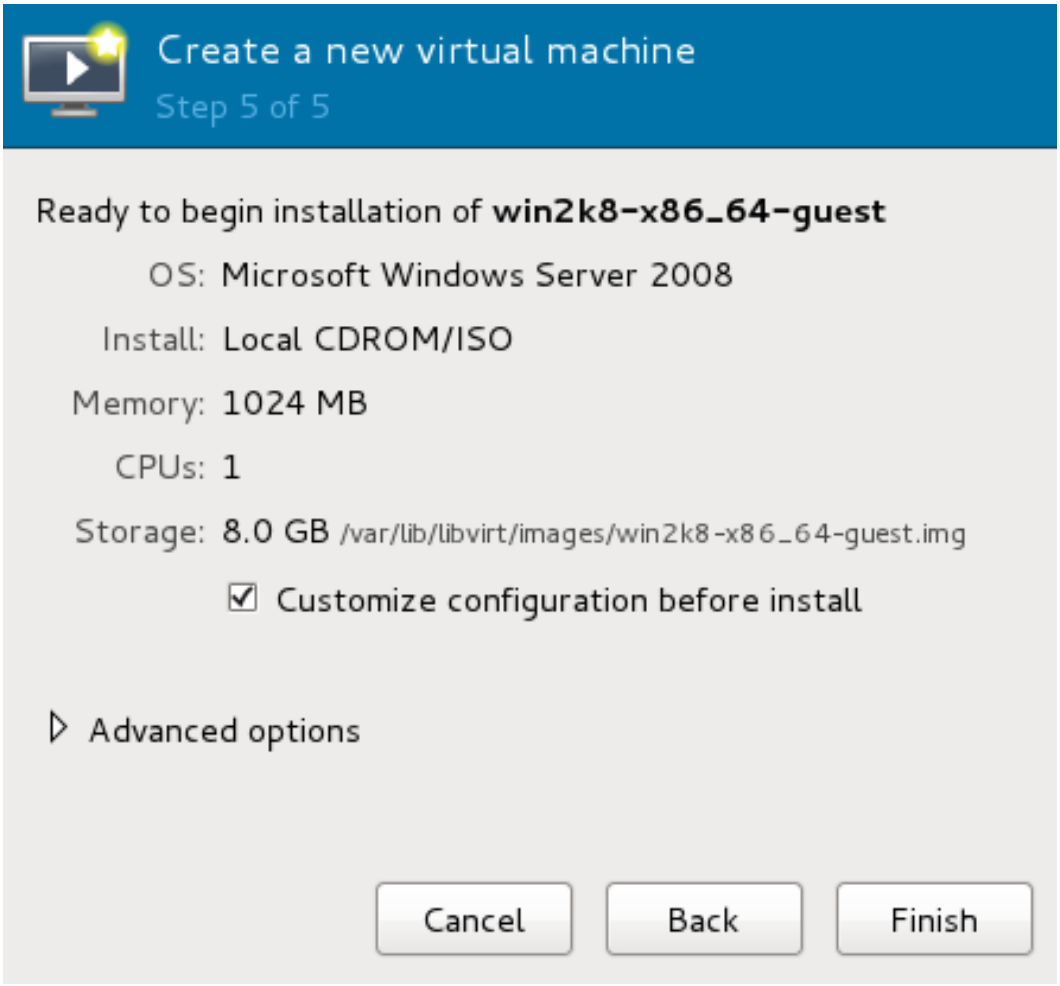


図9.15 virt-manager でのゲスト作成ウィザード

完了 ボタンをクリックし、続けます。

ii. **ハードウェア追加ウィザード**

新しいパネルの左下にある **ハードウェアを追加** ボタンをクリックします。

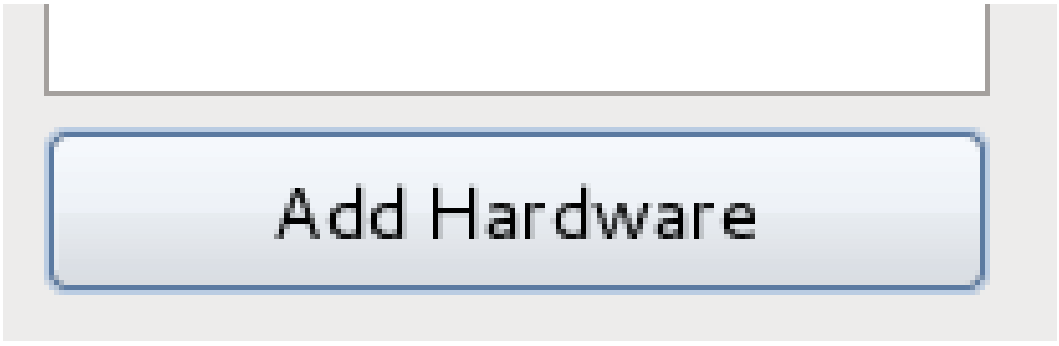


図9.16 ハードウェアを追加ボタン

iii. **ストレージデバイスの選択**

ハードウェアのタイプ のリストでは、**ストレージ** がデフォルトで選ばれています。

図9.17 新たなハードウェア追加ウィザード

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 ラジオボタンを選択します。**参照...** をクリックします。

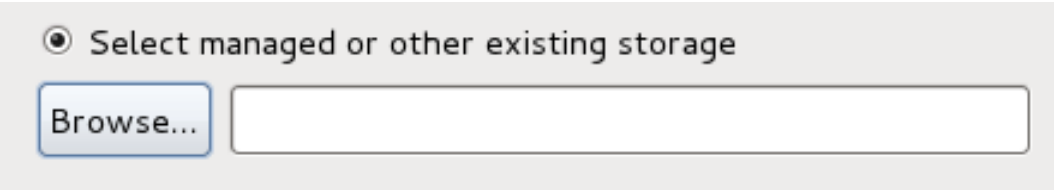


図9.18 管理しているストレージか、他の既存のストレージを選択する

新しいウィンドウが開くので、**ローカルを参照** をクリックします。**/usr/share/virtio-win/virtio-win.vfd** を選択し、**開く** をクリックして確定します。

デバイスの種類 を **Floppy disk** に変更し、**完了** をクリックして続けます。

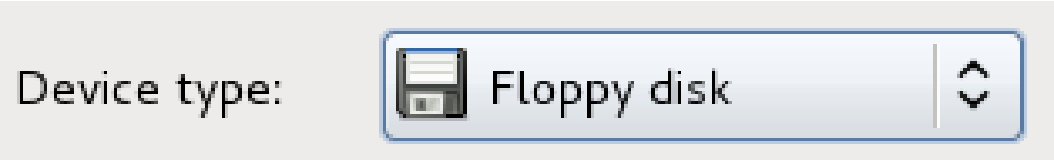


図9.19 デバイスの種類の変更

iv. **設定の確認**

デバイス設定を確認します。

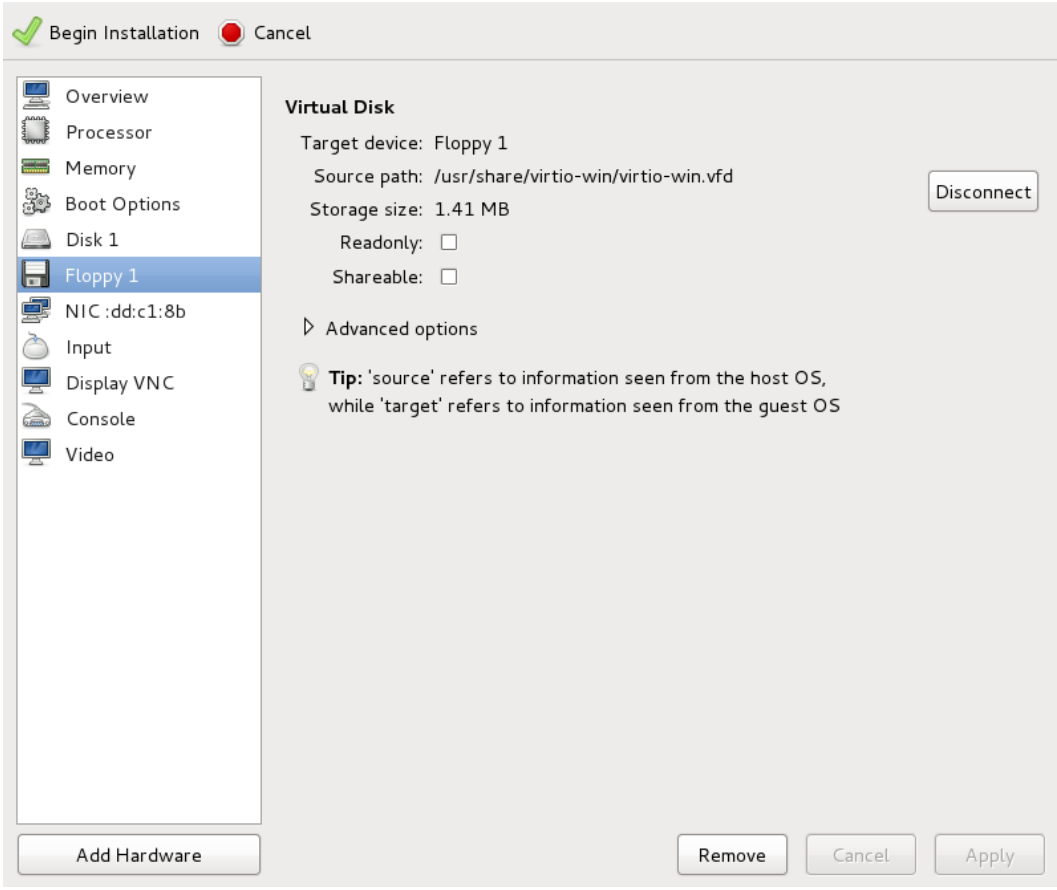


図9.20 仮想マシンのハードウェア情報ウィンドウ

これで、仮想マシンがアクセス可能なリムーバブルデバイスが作成されました。

v. **ハードディスクの種類の変更**

ハードディスクの種類を *IDE Disk* から *Virtio Disk* に変更するには、まず既存のハードディスクである Disk 1 を削除する必要があります。ディスクを選択し、**削除** ボタンをクリックします。

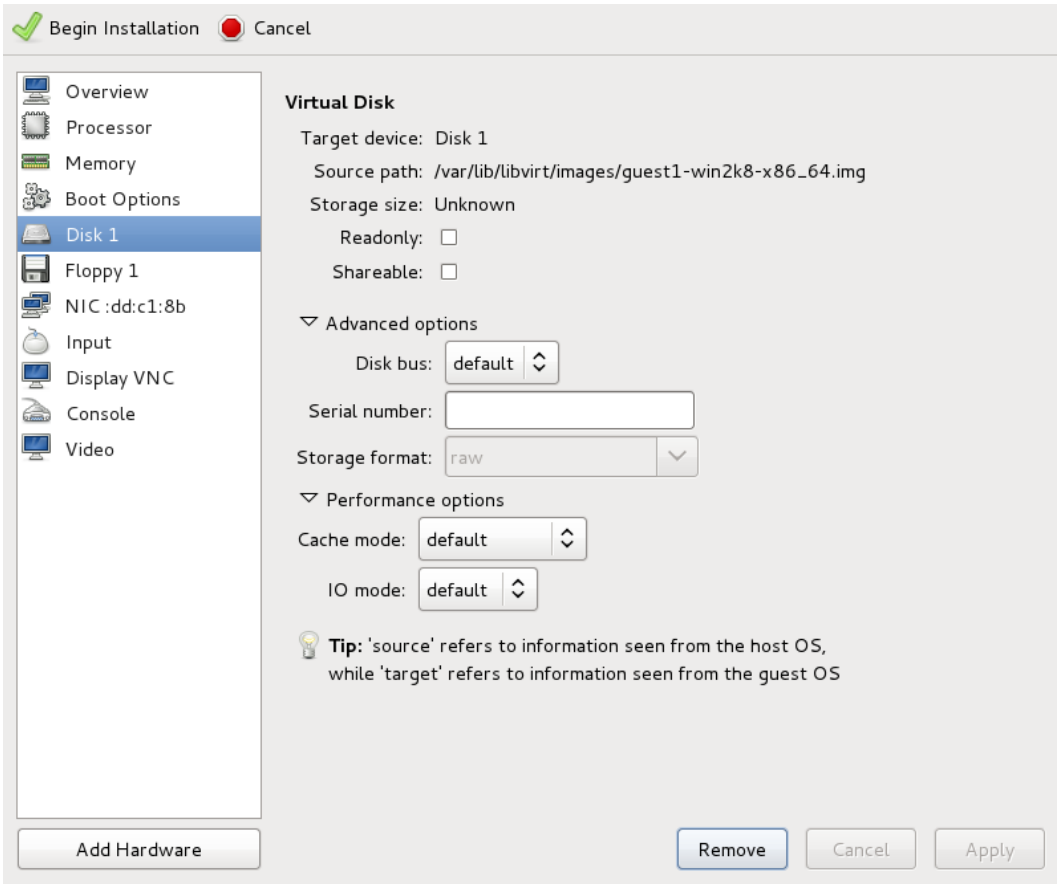


図9.21 仮想マシンのハードウェア情報ウィンドウ

ハードウェアを追加 をクリックして、新たな仮想ストレージデバイスを追加します。次に、**デバイスの種類** を *IDE disk* から *Virtio Disk* に変更します。**完了** をクリックして確定します。

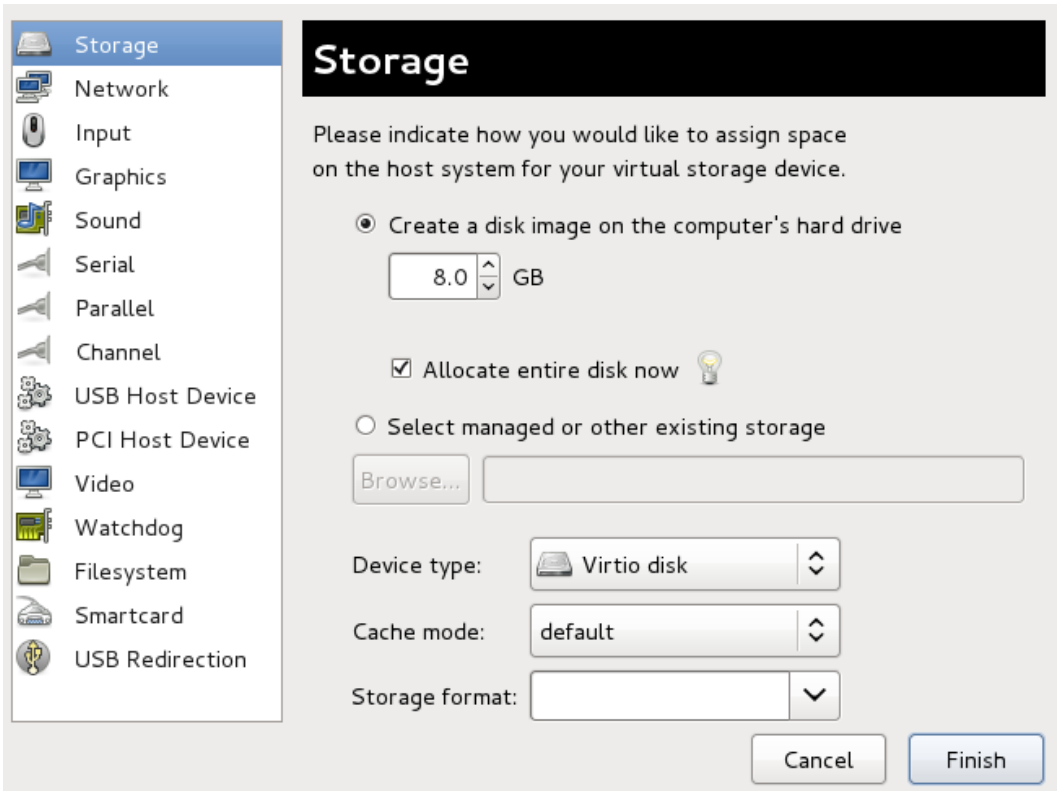


図9.22 仮想マシンのハードウェア情報ウィンドウ

vi. **設定の確認**

VirtIO Disk 1 の設定を確認します。

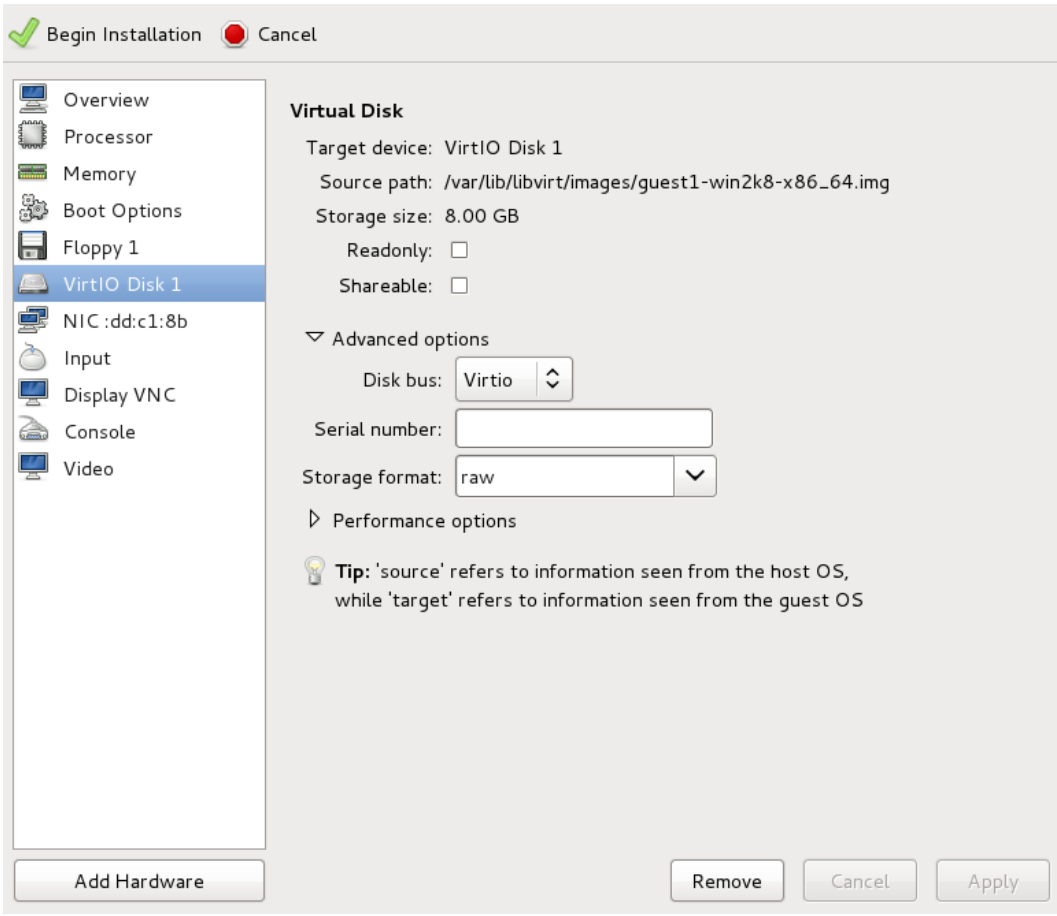


図9.23 仮想マシンのハードウェア情報ウィンドウ

設定詳細が正しければ、**インストールの開始** ボタンをクリックします。

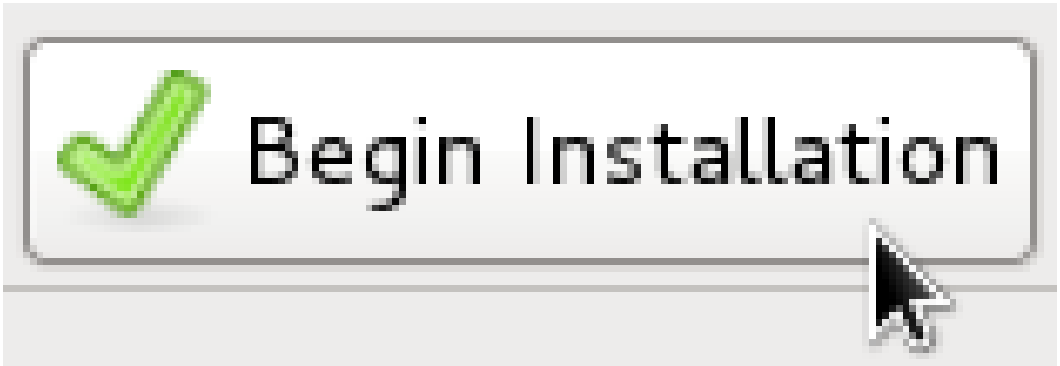


図9.24 インストールの開始ボタン

ステップ 3 に進みます。

c. **virt-install を使ったゲスト仮想マシンの作成**

virt-install コマンドでのインストールにドライバーディスクを追加するには、以下のパラメーターを正確に追加します。

```
--disk path=/usr/share/virtio-win/virtio-
win.vfd,device=floppy
```

重要

追加したいデバイスが **disk** の場合 (つまり、**floppy** や **cdrom** でない場合) 、***--disk*** パラメーターの最後に **bus=virtio** オプションを加える必要もあるので、以下のようになります。

```
--disk path=/usr/share/virtio-win/virtio-
win.vfd,device=disk,bus=virtio
```

インストールする Windows のバージョンによって、**virt-install** コマンドに以下のオプションのいずれかを追加します。

```
--os-variant winxp
```

```
--os-variant win2k3
```

```
--os-variant win7
```

ステップ 3 に進みます。

3.

ドライバーインストールの追加ステップ

インストール中に、 Windows ゲストの種類によってはドライバーをインストールする追加ステップが必要になります。

a.

Windows Server 2003 および Windows XP

サードパーティードライバーの場合、インストールのブルースクリーンで **F6** を繰り返し押します。

図9.25 Windows セットアップ画面

新たなデバイスドライバーをインストールするには、**S** を押します。

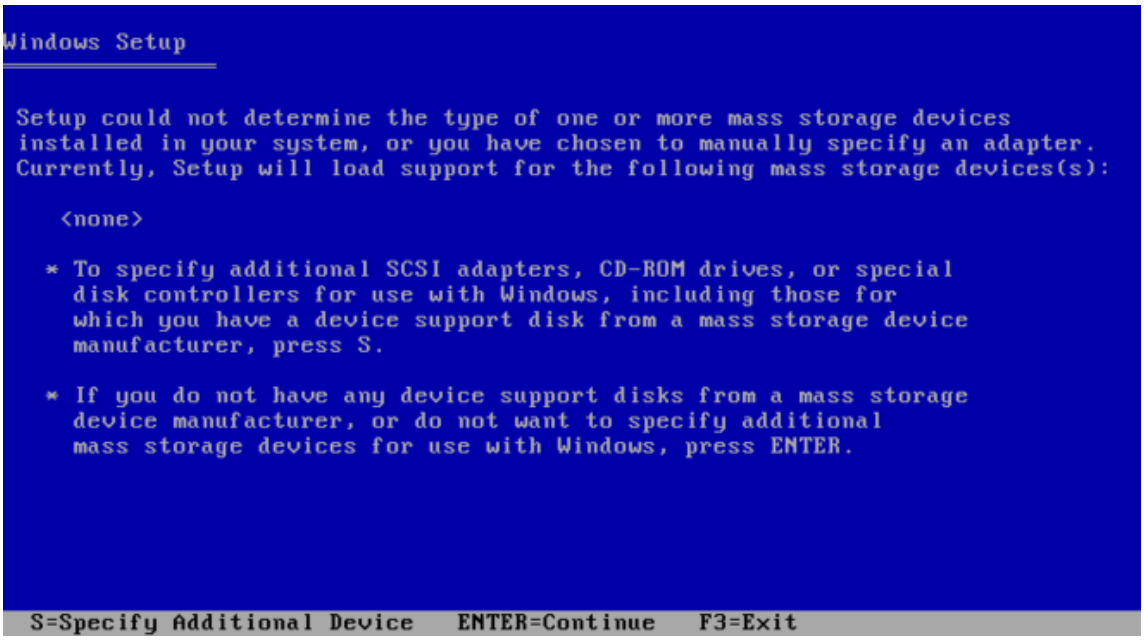


図9.26 Windows セットアップ画面

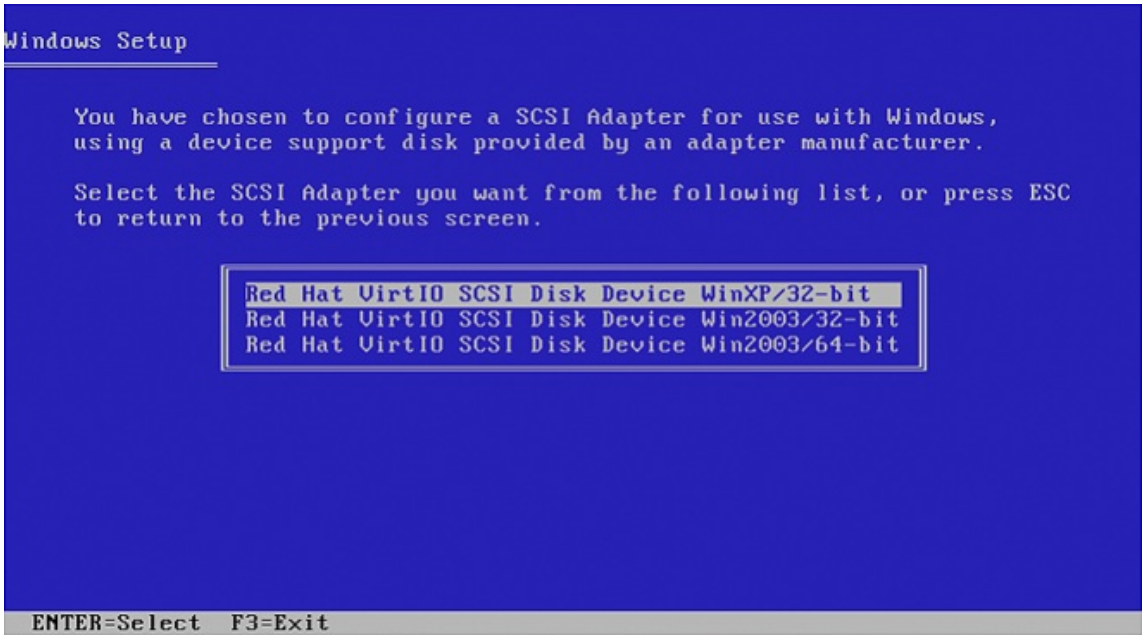


図9.27 Windows セットアップ画面

Enter を押して、インストールを続行します。

b.

Windows Server 2008

Windows Server 2003 と同じ手順ですが、インストーラーがドライバーを表示した時には、**Load Driver** をクリックし、インストーラーを**Drive A:** に向けて、自分のゲストオペレーティングシステムとアーキテクチャーに適切なドライバーを選択します。

9.4. 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 ドライブの代わりに **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、ゲストが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9.4 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、「KVM Windows virtio ドライバーのインストール」で説明された適切なドライバー (**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。例えば、**virsh edit guest1** といったようにです。設定ファイルは、**/etc/libvirt/qemu** 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トリーです。

```
<disk type='file' device='disk'>
   <source file='/var/lib/libvirt/images/disk1.img'/>
   <target dev='hda' bus='ide'/>
</disk>
```

4. virtio デバイスを使用するために、**bus=** エントリーを **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つことになります。**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'/>
</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。仮想マシンの次回スタート時に、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 使用方法の詳細については、libvirt のウェブサイト http://www.linux-kvm.org/page/Virtio を参照してください。

9.5. 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、**virsh attach-disk** または **virsh attach-interface** コマンドで virtio ドライバーを使用して、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、Windows ゲストに準仮想化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9.5 virtio ストレージドライバーを使用してストレージデバイスを追加する

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. **電球** ボタンをクリックして **仮想ハードウェアの詳細表示** タブを開きます。

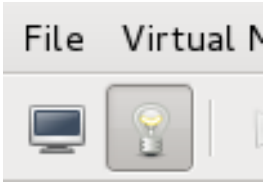


図9.28 仮想ハードウェアの詳細表示タブ

3. **仮想ハードウェアの詳細表示** 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**

   ストレージ で **ハードウェアの種類** を選びます。

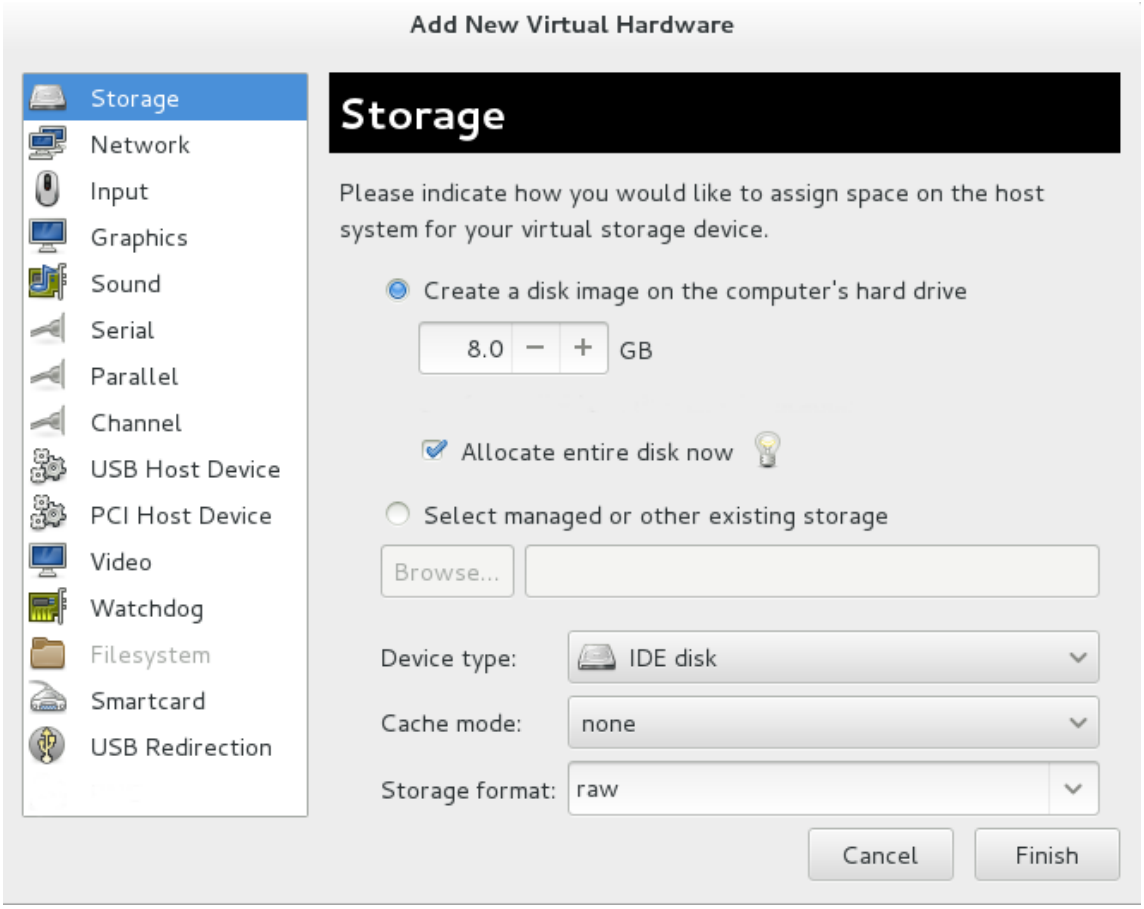


図9.29 新たなハードウェア追加ウィザード

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 **`Virtio disk`** に設定して virtio ドライバーを使用します。

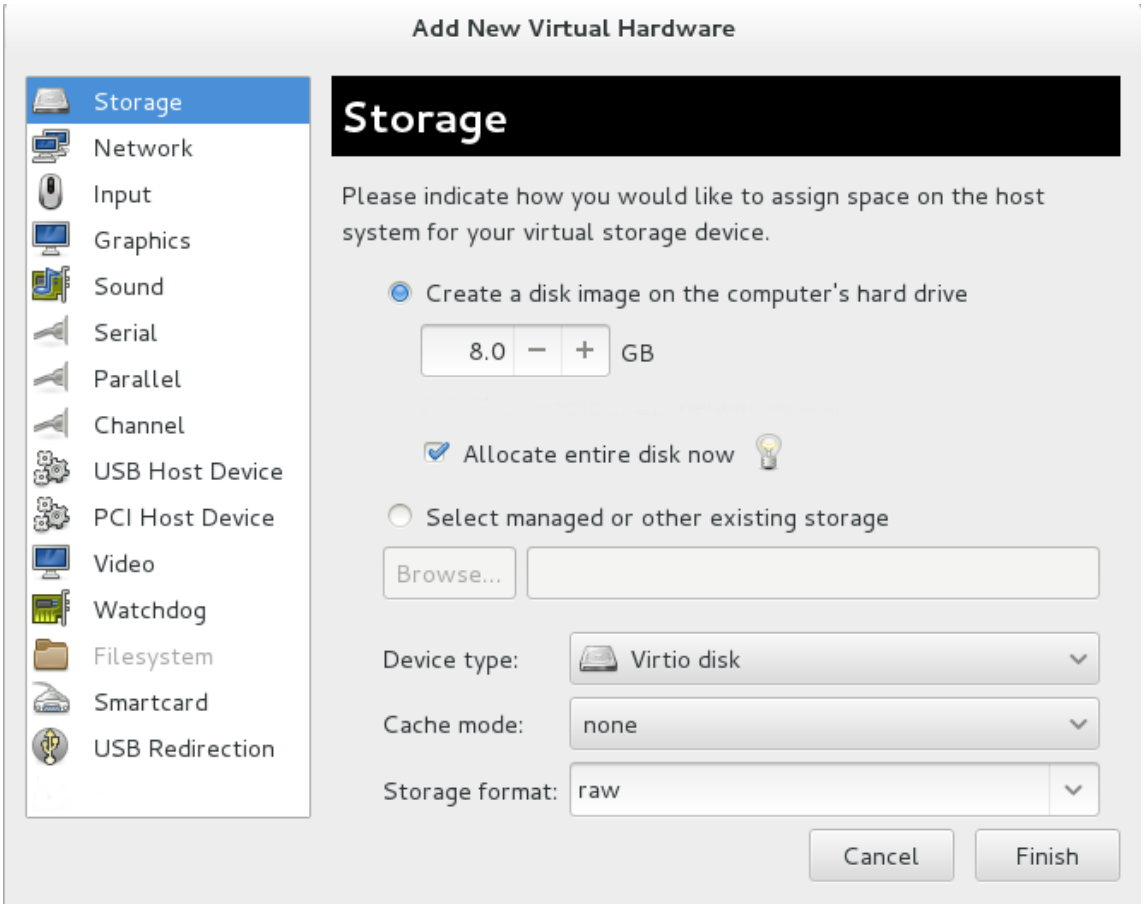


図9.30 新たなハードウェア追加ウィザード

完了 をクリックして終了します。

手順9.6 virtio ネットワークドライバーを使用してネットワークデバイスを追加する

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. **電球** ボタンをクリックして**仮想ハードウェアの詳細表示** タブを開きます。

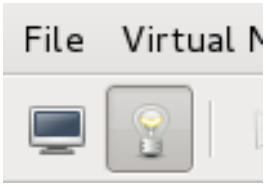


図9.31 仮想ハードウェアの詳細表示タブ

3. **仮想ハードウェアの詳細表示** 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**

   ハードウェアの種類 で **ネットワーク** を選びます。

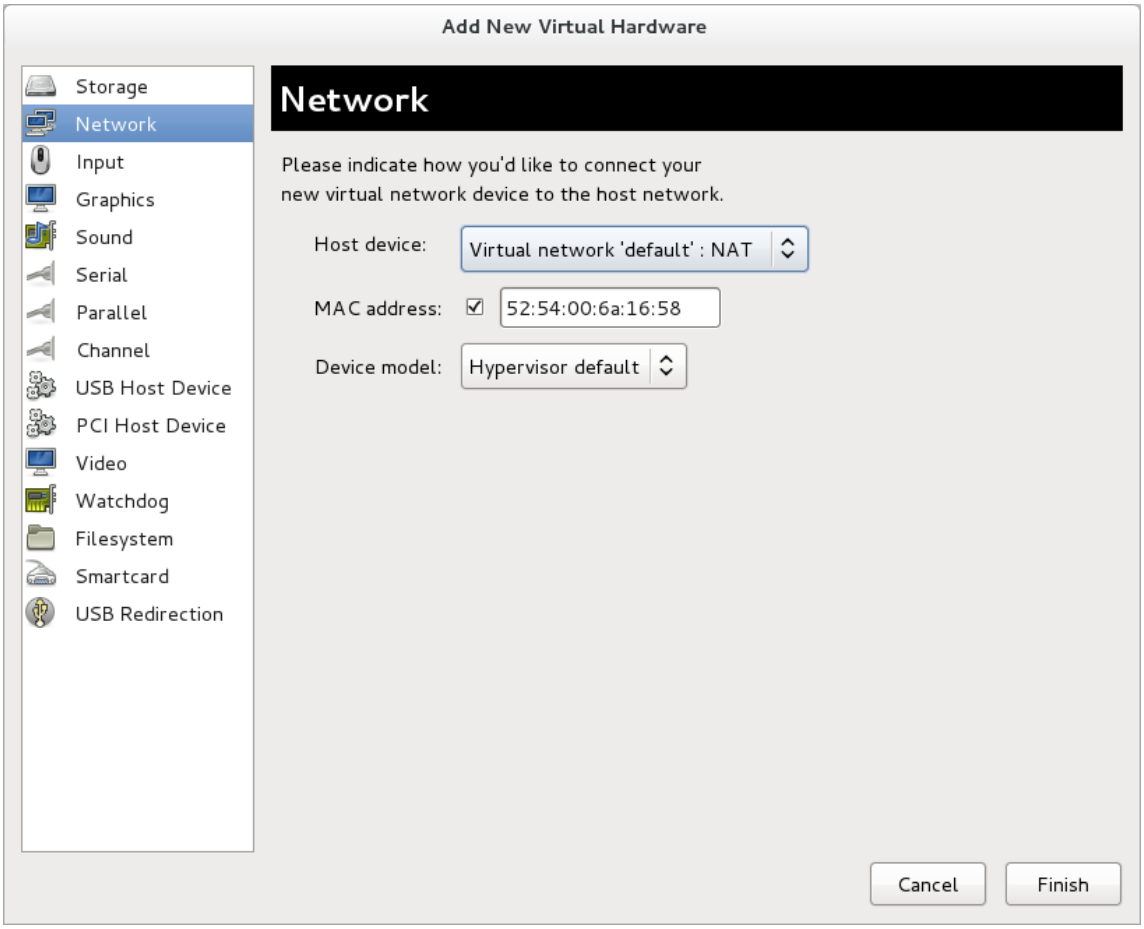


図9.32 新たなハードウェア追加ウィザード

5. **ネットワークデバイスとドライバーの選択**

   `Device model` 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。

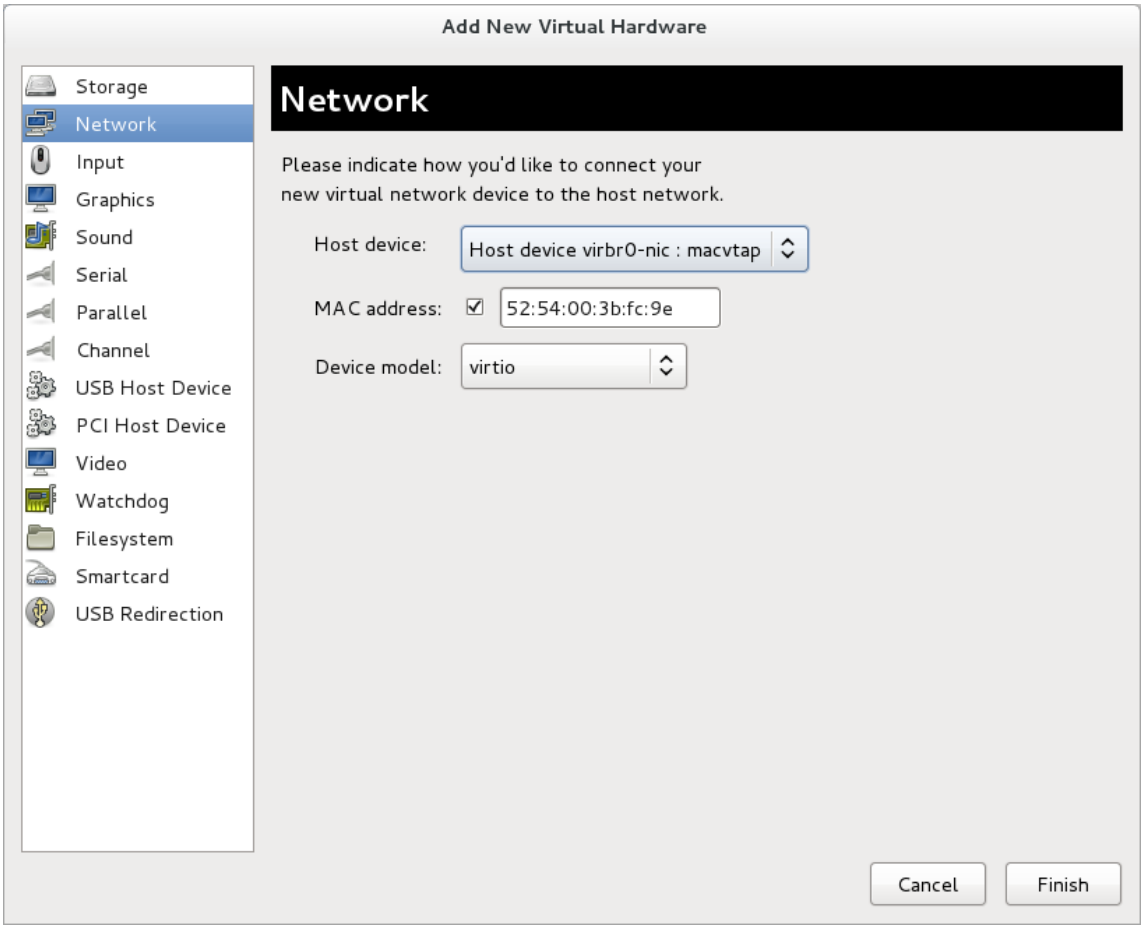


図9.33 新たなハードウェア追加ウィザード

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。Windows 仮想マシンは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。

第10章 ネットワークの設定

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。詳細は、http://libvirt.org/archnetwork.html にある libvirt ネットワークアーキテクチャーについての資料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。

10.1. 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。

ホスト設定

スタンダードの **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で 仮想マシンへのNAT ベースの接続を提供します。**virsh net-list --all** コマンドで利用可能であること確認します。

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

これが存在しない場合、以下がゲスト用に XML 設定ファイル (/etc/libvirtd/qemu/myguest.xml など) で使用される可能性があります。

```
# 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あることが分かります。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          8000.000000000000       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可能になります。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 (**/etc/libvirtd/qemu/myguest.xml** などの) XML 設定ファイル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、自分の環境内での一貫性や容易なリファレンスの維持、または非常に低い可能性の競合の回避に役立つ場合があります。

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

10.2. 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) に動かすことで仮想化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 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ー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を QEMU ユーザー領域に強制するので、そのインスタンスで vhost-net を事実上無効にします。

10.3. vhost-net zero-copy の有効化

Red Hat Enterprise Linux 7 では、vhost-net ゼロコピー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、以下を実行できます。

```
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 つ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。これを使って有効にすることもできますが、この変更は永続化されません。

これが有効になったことを確認するには、**cat /sys/module/vhost_net/parameters/experimental_zcopytx** の出力をチェックします。以下のように表示されるはずです。

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

10.4. ブリッジネットワーク

ブリッジネットワーキング (別名は仮想ネットワークスイッチ) は、仮想マシン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明なセットアップを提供します。

ブリッジは、*virt-manager* または *libvirt* 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らについて説明します。

また、仮想化管理ツールを使わずにブリッジネットワークを設定することもできます。このような設定は、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、またはホストの仮想ネットワークインターフェースである場合により適切になる可能性があります。

注記

仮想化ツール外でネットワークブリッジを設定する方法については、『Red Hat Enterprise Linux 7 Networking Guide』 (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Networking_Guide/ch-Configure_Network_Bridging.html) を参照してください。

10.4.1. virt-manager を使用したブリッジネットワーキング

手順10.1 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 **Connection Details** 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. ウィンドウの最下部にある **+** 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** をクリックして継続します。

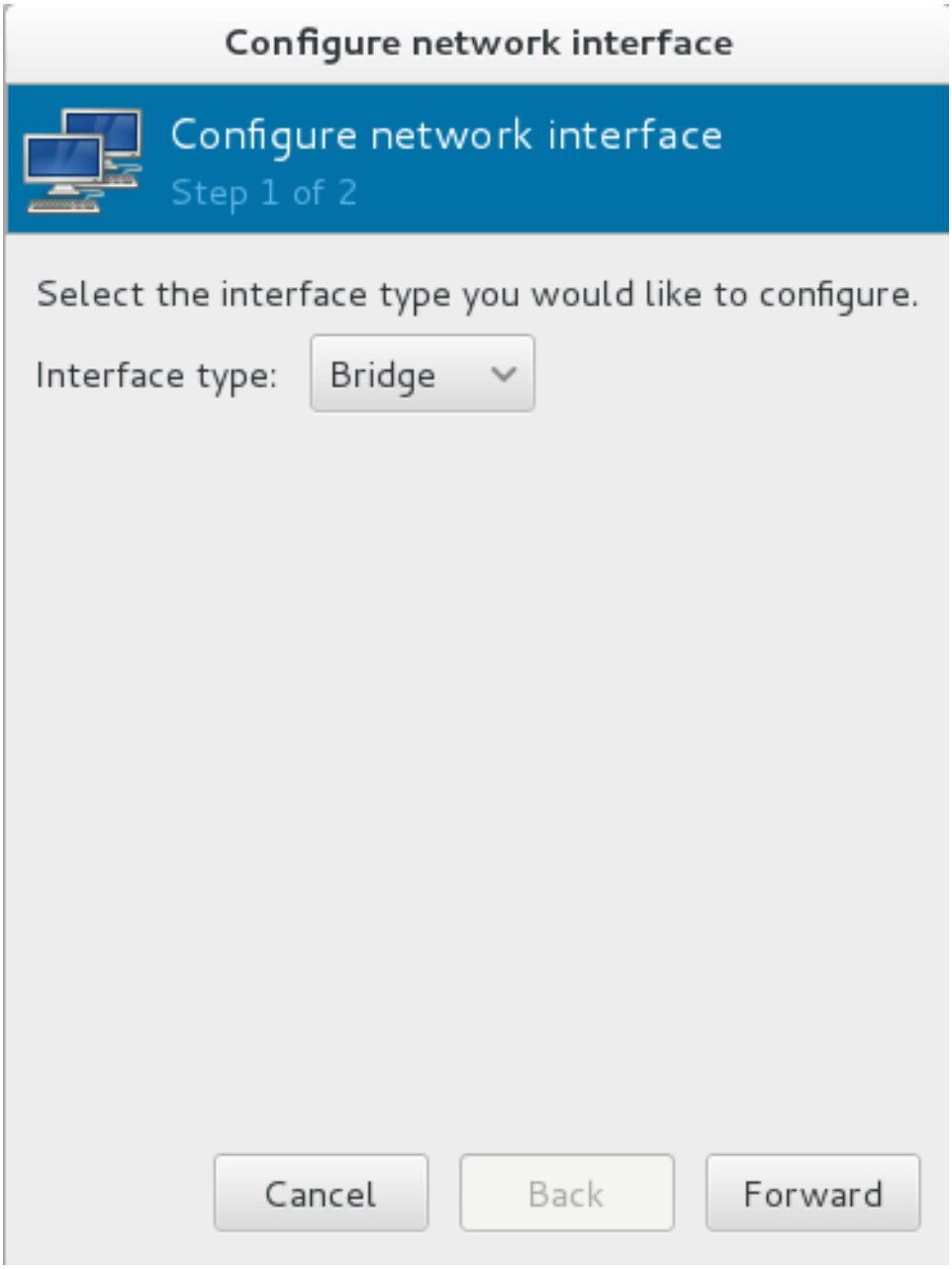


図10.1 ブリッジの追加

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最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化します。

   d. IP またはブリッジのいずれかを設定する場合、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. ブリッジを設定する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。インターフェースを停止するには **Stop (停止)** キーをクリックします。インターフェースを削除するには、**Delete (削除)** キーをクリックします。

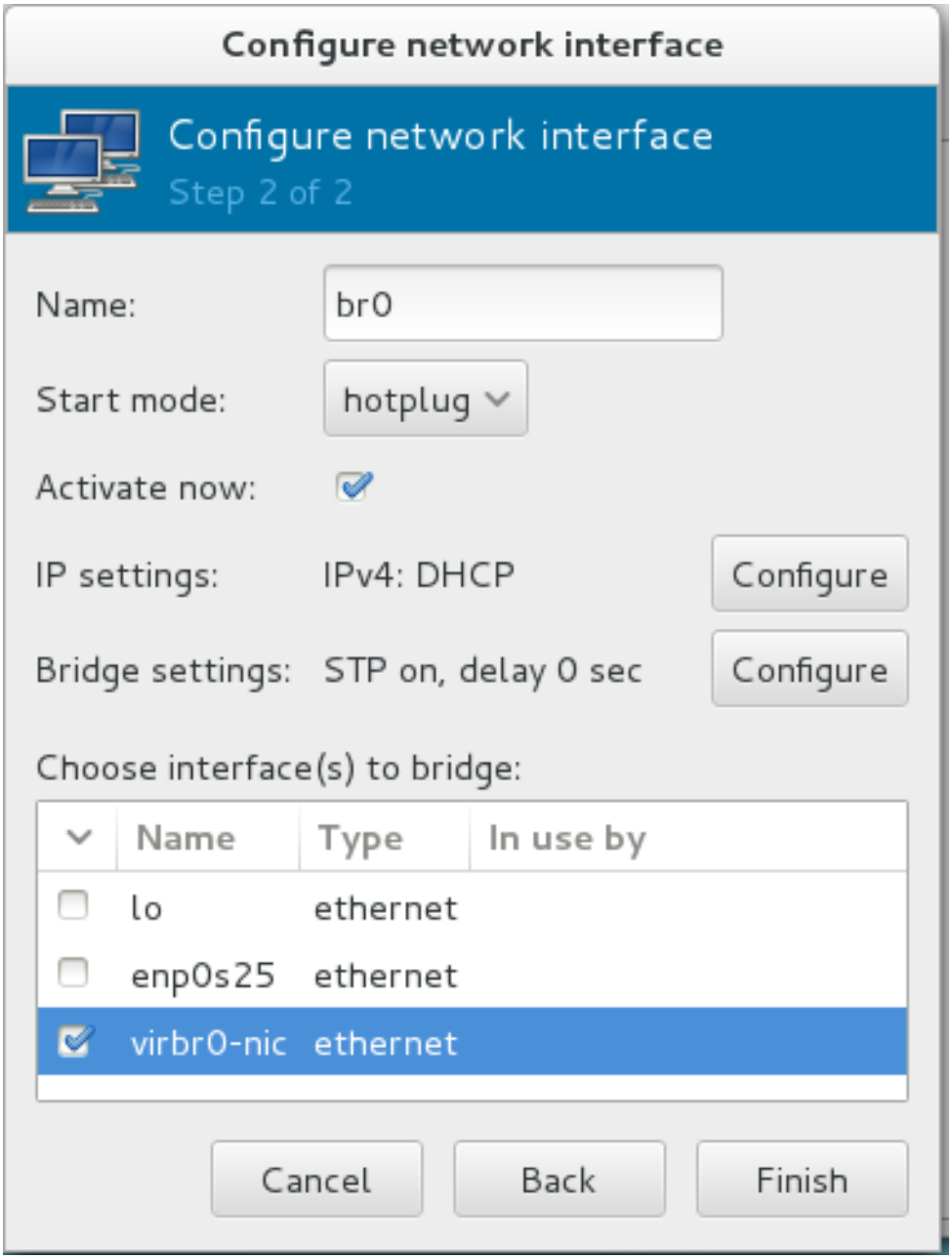


図10.2 ブリッジを追加する - ステップ 2

10.4.2. libvirt を使用したブリッジネットワーキング

このセクションでは、*libvirt* を使用してホスト物理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。virt-manager を使用してブリッジを作成する方法は、「virt-manager を使用したブリッジネットワーキング」 を参照してください。

eth0 インターフェースベースのブリッジ (**br0**) を作成するには、ホスト上で以下のコマンドを実行します。

```
# virsh iface-bridge eth0 br0
```

第11章 KVM でのオーバーコミット

11.1. はじめに

KVM ハイパーバイザーは、 CPU およびメモリーのオーバーコミットに対応しています。 システム上に実際に存在する物理的なリソースの容量を超える仮想化 CPU または仮想化メモリーを割り当てるのがオーバーコミットです。CPU のオーバーコミットを行うことで、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サーバー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。

ほとんどのプロセスで割り当てられたメモリーを常に 100% 必要とすることはありません。KVM ではこの性質を利用することで、ホスト物理マシンに実際にある物理的なメモリー以上のメモリーをゲスト仮想マシンに割り当てることができます。これをリソースのオーバーコミットと呼びます。

11.2. メモリーのオーバーコミット

重要

オーバーコミットがあらゆるメモリー関連の問題に対する理想的な解決策になる訳ではありません。メモリー不足に対処するための推奨される方法は、すべてのゲストのメモリー (およびホスト OS の 4GB) の合計がホスト物理マシンの物理的なメモリーがより少なくなるよう、ゲストに少なめのメモリーを割り当てることです。ゲスト仮想マシンにさらに多くのメモリーが必要な場合は、ゲスト仮想マシンのスワップ領域の割り当てを拡張します。それでもなお、オーバーコミットを採用する場合には十分なテストをしてから行なうようにしてください。

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、専用の物理的な RAM ブロックが割り当てられていません。その代わりに、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、ホスト物理マシンのメモリー管理機能により、物理メモリーとスワップ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。そのため、オーバーコミットを採用する際は、すべてのゲスト仮想マシンに対応するだけでなく、ホスト物理マシンのプロセスに必要なメモリーを確保できるよう、ホスト物理マシン上に十分なスワップ領域を配分する必要があります。基本的に、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 のスワップ領域が必要になります。詳細は、例11.1「メモリーのオーバーコミット例」 を参照してください。

Red Hat ナレッジベース には、スワップパーティションのサイズを安全かつ効率的に確定する方法について記載されています。

注記

以下の例は、スワップを設定する方法のみを説明しています。記載されている設定が実際の環境には適さない場合もありますので注意してください。

例11.1 メモリーのオーバーコミット例

ExampleServer1 に物理 RAM が 32GB あるとします。このシステムは 50 ゲスト仮想マシンを実行するように設定されています。各ゲストには 1GB の仮想化メモリーが必要になります。前述のように、ホスト物理マシンのシステム自体にも最大 4 GB のメモリー (ゲスト仮想マシンのメモリーとは別) とさらに最小でも 4GB 以上のスワップ領域が必要になります。

スワップ領域は以下のように計算します。

- すべてのゲスト仮想マシンに必要なメモリーの合計量を計算します。この例では、50 ゲスト仮想マシン * 1GB メモリー (1 ゲスト仮想マシンあたり) = 50GB になります。
- ホスト物理マシンの OS およびホスト物理マシンの最小スワップ領域に必要なメモリー量に、ゲスト仮想マシンのメモリー量を加えます。この例では、50GB (ゲスト仮想マシンメモリー) + 4GB (ホスト物理マシンの OS) + 4GB (スワップ最小サイズ) = 58GB になります。
- この値からシステム上にある物理 RAM 量を引き算します。この例の場合: 58GB - 32GB = 26GB
- 算出される数値は、割り当てる必要のあるスワップ領域の量になります。この例の場合: 26GB

注記

オーバーコミットはすべてのゲスト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。スワップやメモリーのオーバーコミットの設定は、それぞれの環境やセットアップが異なるため、プラグインのように簡単に採用できるものではなく、設定の際に定形がある訳でもありません。設定を変更する前に慎重に検討し、ご使用の環境や設定を完全に理解した上で変更を行なってください。

KSM およびオーバーコミットに関する詳細は、 24章*KSM* を参照してください。

11.3. 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 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 (仮想化 CPU) 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。

各ゲスト仮想マシンが単一の VCPU を持つ状態が、仮想化 CPU のオーバーコミットを行う最適な状態と言えます。このタイプの負荷については Linux スケジューラーの使用が非常に効率的です。KVM は、VCPU は 5 つまでの、また負荷が 100% 未満となるゲスト仮想マシンを安全にサポートするはずです。VCPU が 1 つのゲスト仮想マシンのオーバーコミットは問題になりません。

物理的なプロセッシングコア数を超えた状態の完全対称型マルチプロセッシングのゲスト仮想マシンはオーバーコミットできません。たとえば、VCPU 数が 4 つのゲスト仮想マシンは、デュアルコアプロセッサーのホスト物理マシンでは実行すべきではありません。実際の物理プロセッシングコア数を超えた状態の完全対称型マルチプロセッシングのゲスト仮想マシンをオーバーコミットすると、パフォーマンスが大幅に低下する原因となります。

ゲスト仮想マシンに割り当てる VCPU 数は、最大でも物理的なコア数と同数にするのが適切であり、その場合は予想どおりに動作します。たとえば、クアッドコア (4 つのプロセッサーコア) のホストでは VCPU 数が 4 つのゲスト仮想マシンを実行できます。この場合、負荷が 100% 未満であれば、ゲスト仮想マシンはこのセットアップで効率的に機能するはずです。

重要

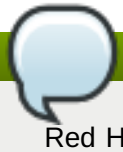
十分な検証を行っていない状態で、 実稼働環境でのメモリーや CPU のオーバコミットは実施しないでください。オーバーコミットしている環境では、メモリーやプロセッシングリソースを 100% 使用するアプリケーションは不安定になる可能性があります。十分なテストを行ってから導入してください。

第12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タイミング管理に内在する各種の課題があります。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、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ー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、正確なタイミング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、時間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ーを実行する前には、タイミングをテストすることが依然として重要です。



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** なしで実行するためには特別な設定が必要になるため、この方法は推奨されていません。

重要

Network Time Protocol (NTP) デーモンがホストとゲスト仮想マシンで 稼働している必要があります。必ず *ntp* をインストールし、**ntpd** サービスを有効にしてください。

ntpd サービスを有効にし、デフォルトの起動シーケンスに追加します。

```
# systemctl enable ntpd
```

サービスを起動します。

```
# systemctl start ntpd
```

クロックの実行速度と参照クロックソースとの違いが 0.05% 未満の場合、**ntpd**サービスはクロックスキューの影響を修正します。**ntp** 起動スクリプトは、必要に応じて起動時のシステムクロックを調整することで、参照時間からのクロックのオフセットを調整します。

不変タイムスタンプカウンター (TSC)

最新の Intel と AMD の CPU は、不変タイムスタンプカウンター (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フラグがある場合は CPU には一定のタイムスタンプカウンターが存在します。CPU に**constant_tsc**フラグがあるかどうかを確認するには次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、以下の説明にしたがってください。

不変 TSC (タイムスタンプカウンター) がないホストの設定

不変 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間管理が妨げられるので、KVM を使用して仮想マシンで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 (BZ#513138)。各システムには、時間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ません。これは、**cpufreq** の変化や Deep C 状態、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。Deep 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。カーネルによる Deep C 状態の使用を回避するには、カーネルブートに **processor.max_cstate=1** を追加します。この変更を永続化するには、 **/etc/default/grub file** の **GRUB_CMDLINE_LINUX_DEFAULT** キーの値を変更します。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_DEFAULT="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_DEFAULT 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** (**systemctl enable cpupower.service**) を有効にします。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_frequencies** 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 Power Management Guide』 を参照してください。

12.1. Red Hat Enterprise Linux ゲストで必要なパラメーター

一部の Red Hat Enterprise Linux ゲスト仮想マシンには、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 **/boot/grub/grub.conf** ファイルの **/kernel** の行の末尾に追記することで設定できます。

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12.1 カーネルパラメーター要件

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
準仮想化クロック搭載の 7.0 AMD64/Intel 64	追加のパラメーターは必要ありません
準仮想化クロック搭載の 6.1 以降の AMD64/Intel 64	追加のパラメーターは必要ありません
準仮想化クロック搭載の 6.0 AMD64/Intel 64	追加のパラメーターは必要ありません
準仮想化クロックを搭載していない 6.0 AMD64/Intel 64	notsc lpj=*n*
準仮想化クロック搭載の 5.5 AMD64/Intel 64	追加のパラメーターは必要ありません
準仮想化クロックを搭載していない 5.5 AMD64/Intel 64	notsc lpj=*n*
準仮想化クロック搭載の 5.5 x86	追加のパラメーターは必要ありません
準仮想化クロックを搭載していない 5.5 x86	clocksource=acpi_pm lpj=*n*
5.4 AMD64/Intel 64	notsc
5.4 x86	clocksource=acpi_pm
5.3 AMD64/Intel 64	notsc
5.3 x86	clocksource=acpi_pm
4.8 AMD64/Intel 64	notsc
4.8 x86	clock=pmtmr
3.9 AMD64/Intel 64	追加のパラメーターは必要ありません

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 ÷ 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。

12.2. Windows Server 2003 および Windows XP ゲストでのリアルタイムクロックの使用

Windows は、リアルタイムクロック (RTC) とタイムスタンプカウンター (TSC) の両方を使用します。Windows ゲスト仮想マシンでは、TSC の代わりに RTC をすべてのタイムリソースに対して使用することができます。これによって、ゲストのタイミング関連の問題が解決されます。

PMTIMER クロックソース (**PMTIMER** は通常 TSC を使用) に対して RTC を有効にするには、Windows の起動設定に以下のオプションを追加します。Windows の起動設定は、boot.ini ファイルに保存されています。**boot.ini** ファイル内の Windows のブート行の最後に以下のオプションを追加してください。

```
/usepmtimer
```

Windows の起動設定および usepmtimer オプションについての詳細は、Windows XP および Windows Server 2003 の Boot.ini ファイルで使用可能なスイッチ オプション を参照してください。

12.3. Windows Server 2008、Windows Server 2008 R2、および Windows 7 ゲストでのリアルタイムクロックの使用

Windows は、リアルタイムクロック (RTC) とタイムスタンプカウンター (TSC) の両方を使用します。Windows ゲスト仮想マシンでは、TSC の代わりに RTC をすべてのタイムリソースに対して使用することができます。これによって、ゲストのタイミング関連の問題が解決されます。

Windows Server 2008 以降では、**boot.ini** ファイルは使用されなくなりました。Windows Server 2008 と Windows Server 2008 R2 および Windows 7 では、**hypervisor-present** ビットが設定されていると、TSC をタイムソースとして使用しません。Red Hat Enterprise Linux 7 KVM ハイパーバイザーは、この CPUID ビットをデフォルトで有効にしているので、**Boot Configuration Data Editor** (**bcdedit.exe**) を使用して Windows 起動パラメーターを変更する必要がなくなりました。

1. Windows ゲスト仮想マシンを開きます。
2. **スタート** メニューの **アクセサリ** を開きます。**コマンドプロンプト** を右クリックして、**管理者として実行** を選択します。
3. セキュリティー例外が出てきた場合は、これを確認します。
4. ブートマネージャーでプラットホームクロックを使用するように設定します。これにより、Windows はプライマリークロックソースに PM タイマーを使用するように指示されます。システム UUID (以下の例では、***{default}***) がデフォルトの起動デバイスと異なる場合は、システム UUID を変更します。

   ```
   C:\Windows\system32>bcdedit /set {default} USEPLATFORMCLOCK on
   The operation completed successfully
   ```

この修正で、Windows Server 2008 および Windows 7 のゲストの時間管理が改善されます。

12.4. 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す。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 ***st*** として報告されます。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、スイッチオフにすることはできません。

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13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。

13.1. 起動サーバーの準備

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

13.1.1. 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13.1 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/tftp** に配置します。
2. 以下のコマンドを実行します。

```
# 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/tftp' />
   <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)。

13.2. 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。

13.2.1. ブリッジネットワークの使用

手順13.2 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>** 内に **<boot dev='network'/>** 要素があることを確認します。

```
<os>
   <type arch='x86_64' machine='rhel6.2.0'>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>
```

13.2.2. プライベート libvirt ネットワークの使用

手順13.3 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>** 内に **<boot dev='network'/>** 要素があることを確認します。

```
<os>
   <type arch='x86_64' machine='rhel6.2.0'>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14章 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。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に応答します。この章では、libvirt コマンドおよびゲストエージェントが利用できる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、ゲストエージェントを使用するのが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組み込まれた安全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協調を要求します。

Linux および Windows ゲスト上の QEMU ゲストアージェントのサポートと共に、CPU のホットプラグおよびホットアンプラグがでサポートされていることに注意してください。CPU はゲストの実行中に有効/無効にすることができるので、ホットプラグ機能を実装でき、かつアンプラグ機能と同じ動作を想定することができます。詳細は、「仮想 CPU 数の設定」を参照してください。

14.1. 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 シリアル接続でゲストエージェントと通信します。VirtIO シリアルチャネルは、キャラクターデバイスドライバー (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ネルでリッスンします。以下の手順では、ゲストエージェントの使用に備えてホストおよびゲストマシンを設定する方法を説明します。

注記

Windows ゲストに QEMU ゲストエージェントをセットアップする方法については、こちら をご覧いただくか、または「Windows ゲスト上での QEMU ゲストエージェントの実行」を参照してください。

手順14.1 ゲストエージェントとホスト間の通信設定

1. **ゲスト仮想マシンのドメイン XML を開きます。**

   QEMU ゲストエージェント設定を使ってゲスト仮想マシンのドメイン XML を開きます。ファイルを開くにはドメイン名が必要です。ホスト物理マシン上でコマンド **# virsh list** を使用して、これが認識できるドメインを一覧表示します。この例では、ドメインの名前は以下に示されるように *rhel7* になります。

   ```
   # virsh edit rhel7
   ```

2. **ドメイン XML ファイルを編集します。**

   以下の要素を XML ファイルに追加し、変更を保存します。

```
<channel type='unix'>
   <source mode='bind' path='/var/lib/libvirt/qemu/rhel7.agent'/>
   <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

図14.1 QEMU ゲストエージェントを設定するためのドメイン XML の編集

3. **ゲスト内の QEMU ゲストエージェントを起動します。**

   まだ実行していない場合は、**`yum install qemu-guest-agent`** を使用してゲスト仮想マシン内にゲストエージェントをダウンロードし、これをインストールします。いったんインストールしたら、以下のようにサービスを開始します。

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

これで設定されたキャラクターデバイスドライバーで有効な libvirt コマンドが送信され、ゲストと通信することができます。

14.2. QEMU ゲスト仮想マシンエージェントの使用

QEMU ゲスト仮想マシンのエージェントプロトコル (QEMU GA) パッケージの *qemu-guest-agent* は Red Hat Enterprise Linux 7 で完全にサポートされています。isa-serial/virtio-serial トランスポートに関する問題がいくつかあるため、以下のような警告が出されています。

- *qemu-guest-agent* は、クライアントがチャネルに接続しているかどうかを検出できません。
- クライアントからは、*qemu-guest-agent* がバックエンドとの接続を切断したか、または再接続したかを検出する方法がありません。
- virtio-serial デバイスがリセットされ、その結果として *qemu-guest-agent*> がチャネルに接続されなくなった場合 (通常は再起動またはホットプラグによって引き起こされる)、クライアントからのデータはドロップされます。
- virtio-serial デバイスがリセットされてから *qemu-guest-agent* がチャネルに接続されると、*qemu-guest-agent* がまだ実行中か、または接続されているかどうかにかかわらず、クライアントからのデータはキューに入れられます (さらに使用可能なバッファーが消費されると、スロットリングされます)。

14.2.1. libvirt を使用した QEMU ゲストエージェントの更新

QEMU ゲストエージェントをインストールすると、他のさまざまな 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`** はクリーンな状態で協調的なゲストを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、ゲストの協調を可能にすることにより、ディスクコンテンツの高い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。

14.2.2. ゲスト仮想マシンディスクバックアップの作成

libvirt は *qemu-ga* 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。Red Hat Enterprise Linux 7 に改善が加えられたことにより、ファイルとアプリケーションの両レベルの同期 (フラッシュ) が確実に実行されるようになりました。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-ga* は主なフックスクリプト (*qemu-ga* 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。

ファイルシステムがフリーズされる前に、以下のアクションが発生します。

- 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ー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-ga* 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。

snapshot-create-as を使用するために必要なコマンドの詳細は、「現在のドメインのスナップショットの作成」 に説明されています。

注記

アプリケーション固有のフックスクリプトが正常に実行されるには、各種の SELinux パーミッションが必要になる場合があります。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14.1「QEMU ゲストエージェントのパッケージコンテンツ」 の **/etc/qemu-ga/fsfreeze-hook.d/** というラベルが付けられた表中の行にある **restorecon -FvvR** コマンドを実行した後に、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するはずです。

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14.1 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
/usr/lib/systemd/system/qemu-guest-agent.service	QEMU GA のサービス制御スクリプト (起動/停止)。

ファイル名	説明
/etc/sysconfig/qemu-ga	**/usr/lib/systemd/system/qemu-guest-agent.service** 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。
/usr/bin/qemu-ga	QEMU GA バイナリーファイル。
/etc/qemu-ga	フックスクリプトの root ディレクトリー。
/etc/qemu-ga/fsfreeze-hook	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。
/etc/qemu-ga/fsfreeze-hook.d	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 **restorecon -FvvR** を実行する必要があります。
/usr/share/qemu-kvm/qemu-ga/	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。

メインフックスクリプトの **/etc/qemu-ga/fsfreeze-hook** は、アプリケーション固有スクリプトの標準出力とエラーメッセージと共に、独自のメッセージのログを **/var/log/qemu-ga/fsfreeze-hook.log** のログファイルに記録します。詳細は、wiki.qemu.org、または libvirt.org の qemu-guest-agent wiki ページを参照してください。

14.3. Windows ゲスト上での QEMU ゲストエージェントの実行

以下の説明は、Red Hat Enterprise Linux ホストの物理マシンのみで実行される Windows ゲストを対象にしています。

注記

Windows ゲスト仮想マシンは、Windows 用の QEMU GA である *qemu-guest-agent-win* を使用します。このエージェントは、Red Hat Enterprise Linux 上で実行される Windows ゲスト仮想マシンへの VSS (Volume Shadow Copy Service) サポートに必要です。詳細は、こちら をご覧ください。

手順14.2 Windows ゲスト上で QEMU ゲストエージェントを設定する

1. **Red Hat Enterprise Linux ホスト物理マシンの準備**

   以下のパッケージが Red Hat Enterprise Linux ホスト物理マシン上に実行されていることを確認します。

   - *virtio-win*: **/usr/share/virtio-win/** にあります。
   - *spice* および *qxl* ドライバー: どちらも **/usr/share/virtio-win/drivers/** にあります。

   Windows ゲストにドライバーをコピーするには、以下のコマンドを使用して **qxl** ドライバーの ***.iso** ファイルを作成します。

```
# mkisofs -o /var/lib/libvirt/images/virtiowin.iso
/usr/share/virtio-win/drivers
```

2. **Windows ゲストの準備**

   ドライバーを更新するために、***.iso** を Windows ゲストにマウントして virtio-serial および spice+qxl ドライバーをゲストにインストールします。必ずゲストを起動してから、以下のように driver .iso ファイルをゲストに割り当てます。

   ```
   # virsh attach-disk guest /var/lib/libvirt/images/virtiowin.iso
   vdb
   ```

   Windows の **Control Panel** を使用してドライバーをインストールするには、以下のメニューに移動します。

   - qxl-win ドライバーのインストール: **Hardware and Sound** > **device manager** > **display adapter** を選択し、spice+qxl でドライバーを更新します。
   - virtio-win ドライバーのインストール: **Hardware and Sound** > **device manager** > **virtio-serial driver** を選択します。

3. **Windows ゲスト XML 設定ファイルの更新**

   Windows ゲストのドメイン XML ファイルは Red Hat Enterprise Linux ホスト物理マシン上にあります。このファイルにアクセスするには、Windows ゲストドメイン名が必要です。マシンが認識できるようにドメインを一覧表示するには、ホスト物理マシン上でコマンド **# virsh list** を使用します。この例では、ドメインの名前は以下に示すように win7x86 になります。

   ```
   # virsh edit win7x86
   ```

4. **ドメイン XML ファイルを編集します。**

   以下の要素を XML ファイルに追加し、変更を保存します。

   ```
         <channel type='unix'>
            <source mode='bind'
   path='/var/lib/libvirt/qemu/rhelnew.agent'/>
            <target type='virtio' name='org.qemu.guest_agent.0'/>
            <address type='virtio-serial' controller='0' bus='0'
   port='1'/>
         </channel>
         <channel type='spicevmc'>
            <target type='virtio' name='com.redhat.spice.0'/>
            <address type='virtio-serial' controller='0' bus='0'
   port='2'/>
         </channel>
   ```

   図14.2 QEMU ゲストエージェントを設定するための Windows ゲストのドメイン XML の編集

5. **Windows ゲストでの qemu-ga の準備**

Windows ゲストでゲストエージェントを準備するには、以下を実行します。

a. **最新の *virtio-win* パッケージをインストールします。**

インストールするファイルを見つけるには、Red Hat Enterprise Linux ホスト物理マシンのターミナルウィンドウ上で以下のコマンドを実行します。以下に示されるファイルは、お使いのシステムで検索されるファイル名と全く同じであるとは限りませんが、最新の正式バージョンの名前であるはずです。

```
# rpm -qa|grep virtio-win
virtio-win-1.6.8-5.el7.noarch

# rpm -iv virtio-win-1.6.8-5.el7.noarch
```

b. **インストールが完了したことを確認します。**

virtio-win パッケージがインストールを終了した後に、**/usr/share/virtio-win/guest-agent/** フォルダーをチェックすると、以下に示すように qemu-ga-x64.msi または qemu-ga-x86.msi というファイルが見つかります。

```
# ls -l /usr/share/virtio-win/guest-agent/

total 1544

-rw-r--r--. 1 root root 856064 Oct 23 04:58 qemu-ga-x64.msi

-rw-r--r--. 1 root root 724992 Oct 23 04:58 qemu-ga-x86.msi
```

c. **.msi ファイルのインストール**

Windows ゲスト (例: win7x86) から、ファイルをダブルクリックして qemu-ga-x64.msi または qemu-ga-x86.msi をインストールします。いったんインストールされると、システムマネージャー内の Windows ゲストに *qemu-ga* サービスとして表示されます。サービスのステータスをモニタリングするためにこの同じマネージャーを使用できます。

注記

現在 Windows ゲストの qemu-ga でサポートされるのは以下のコマンドのみです。これらのコマンドは、Red Hat Enterprise Linux コマンドと同様に機能し、汎用性があります。詳細情報は、パッケージのインストール時にダウンロードされる **README.TXT** ファイルを参照してください。この情報はすべて参照していただくことをお勧めします。さらに詳しくは、qemu-devel list を参照してください。

- **guest-info**
- **guest-ping**
- **guest-sync-delimited**
- **guest-sync**
- **guest-shutdown**
- **guest-suspend-disk**
- **guest-suspend-ram**
- **guest-fsfreeze-status**
- **guest-fsfreeze-freeze**
- **guest-fsfreeze-thaw**

パート II. 管理

第15章 ホスト物理マシンのセキュリティー保護およびパフォーマンスの強化

Red Hat Enterprise Linux ホストのパフォーマンスを強化するには次のようなタスクやヒントが役立ちます。

- 強制 (enforcing) モードで SELinux を実行します。**setenforce** コマンドを使って SELinux を強制 (enforcing) モードで実行するように設定します。

```
# setenforce 1
```

- すべての不必要なサービスを削除するか、または無効にします (**AutoFS**、**NFS**、**FTP**、**HTTP**、**NIS**、**telnetd**、**sendmail** など)。
- サーバー上にはプラットフォームの管理に必要な最低限のユーザーアカウントのみを追加します。不必要なユーザーアカウントは削除してください。
- ホストでは不必要なアプリケーションは実行しないようにしてください。ホストでアプリケーションを実行すると仮想マシンのパフォーマンスに影響を与えるため、その影響がサーバーの安定性に及ぶ可能性があります。サーバーをクラッシュさせる可能性のあるアプリケーションは、サーバー上のすべての仮想マシンをダウンさせてしまう原因ともなります。
- 仮想マシンのインストールおよびイメージには集中管理できる場所を使用します。仮想マシンのイメージは **/var/lib/libvirt/images/** に格納してください。仮想マシンのイメージをこれ以外のディレクトリーに格納する場合は、そのディレクトリーを SELinux ポリシーに追加し、インストールを開始する前にラベルの再設定を必ず行ってください。集中管理ができる共有可能なネットワークストレージの使用を強くお勧めします。

注記

パフォーマンスに関するヒントの詳細は、『Red Hat Enterprise Linux Virtualization Tuning and Optimization Guide』 を参照してください。

セキュリティーに関するヒントの詳細は、『Red Hat Enterprise Linux Virtualization Security Guide』 を参照してください。

これらのガイドはすべて https://access.redhat.com/site/documentation/ からご利用いただけます。

15.1. セキュリティー導入計画

仮想化技術を導入する際には、ホスト物理マシンとそのオペレーティングシステムが攻撃されないことを確認する必要があります。この場合、ホスト物理マシンとは、システム、デバイス、メモリー、ネットワークのほかにすべてのゲスト仮想マシンを管理する Red Hat Enterprise Linux システムのことです。ホスト物理マシンが保護されていないと、システム内のすべてのゲスト仮想マシンが脆弱になります。仮想化を使用してシステムのセキュリティーを強化する方法はいくつかあります。担当者または担当者の企業は導入計画を作成する必要があります。この導入計画には、以下が含まれている必要があります。

- 動作仕様
- ご使用のゲスト仮想マシンで必要なサービスの指定

- ホスト物理サーバーとこれらのサービスに必要なサポートの指定

導入計画を作成する際に、考慮に入れる必要のあるいくつかのセキュリティー関連の問題があります。

- ホスト物理マシン上では必要となるサービスのみを実行する。ホスト物理マシン上で実行されているプロセスやサービスが少ないほど、セキュリティーのレベルとパフォーマンスが高くなります。
- ハイパーバイザーで SELinux を有効にする。SELinux と仮想化の使用方法についての詳細は、『Red Hat Enterprise Linux Virtualization Security Guide』 を参照してください。
- ファイアーウォールを使用してホスト物理マシンへのトラフィックを制限する。攻撃からホスト物理マシンを保護するデフォルト拒否ルールでファイアウォールをセットアップできます。また、ネットワークを介するサービスを制限することも重要です。
- 標準ユーザーのホストのオペレーティングシステムに対するアクセスを許可しない。ホストのオペレーティングシステムの特権アカウントが設定されている場合、特権のないアカウントにアクセスを許可すると、セキュリティーが危険にさらされる可能性があります。

15.2. クライアントアクセス制御

libvirt のクライアントアクセス制御フレームワークでは、システム管理者は複数のクライアントユーザー、管理オブジェクト、および API 操作に対する細かいアクセス権のルールをセットアップすることができます。これにより、クライアントの接続を最小限の特権セットに制限することができます。

デフォルト設定では、*libvirtd* デーモンには 3 つのレベルのアクセス制御があります。最初は、すべての接続が非認証状態で行われます。この状態では、認証を完了するために必要な API 操作のみが許可されます。認証が成功した後は、クライアント接続に使用されたソケットによって、接続はすべての API 呼び出しへの完全な無制限アクセスを持つか、または「読み取り専用」操作に制限されます。アクセス制御フレームワークでは、管理者が、認証された接続に細かいアクセス権ルールを定義することができます。libvirt のすべての API 呼び出しには、使用されるオブジェクトに対して検証される一連のアクセス権があります。さらに、特定のフラグが API 呼び出しに設定されているかについてアクセス権のチェックが行われます。API 呼び出しに渡されるオブジェクトのチェックのほかにも、一部のメソッドは結果をフィルタリングします。

15.2.1. アクセス制御ドライバー

アクセス制御フレームワークは、今後追加される任意のアクセス制御技術との統合を可能にするためにプラグ可能なシステムとして設計されています。デフォルトでは、ドライバーは使用されません。そのため、アクセス制御のチェックは全く行われません。*libvirt* は polkit を実際のアクセス制御ドライバーとして使用するためのサポートと共に出荷されます。polkit アクセス制御ドライバーの使用方法については、設定ドキュメント を参照してください。

アクセス制御ドライバーは、***access_drivers*** パラメーターを使用して *libvirtd.conf* 設定ファイルで設定されます。このパラメーターは、数々のアクセス制御ドライバー名を受け入れます。複数のアクセス制御ドライバーが必要とされる場合は、すべてのアクセス制御ドライバーのアクセスが付与されるようにそれらが処理される必要があります。「polkit」をドライバーとして有効にするには、以下のコマンドを実行します。

```
# augtool -s set '/files/etc/libvirt/libvirtd.conf/access_drivers[1]'
polkit
```

ドライバーをデフォルト (アクセス制御なし) に戻すには、以下のコマンドを実行します。

```
# augtool -s rm /files/etc/libvirt/libvirtd.conf/access_drivers
```

libvirtd.conf に変更を加えると、*libvirtd* デーモンの再起動が必要になることに注意してください。

15.2.2. オブジェクトおよびアクセス権

libvirt は、アクセス制御を、その API の主なオブジェクトタイプすべてに適用します。それぞれのオブジェクトタイプには、それぞれ一連のアクセス権が定義されます。特定の API 呼び出しについてチェックされるアクセス権を判別するには、該当 API の API 参照マニュアル文書を参照してください。オブジェクトとアクセス権の詳細の一覧は、 libvirt.org を参照してください。

第16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が、管理者の中には、ストレージプールやボリュームを使用せずに独自のストレージを管理し、ストレージ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。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。

この時点で、従来のシステム管理ツールに対し、ストレージプールとボリュームの値が不明確な場合は、*libvirt* の機能にリモートプロトコルが含まれることに留意してください。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ことになります。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ーであるものの、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上のディスクのサイズを超えることはできません。以下が最大サイズです。

- 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ー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、スケーラビリティー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16.1 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で定義します (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。

16.1. 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。

16.1.1. virsh を使用してディスクベースのストレージプールを作成

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16.2 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. **デバイスを接続します。**

   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
```

4. **ストレージプールを起動します。**

   virsh pool-start コマンドを使用してストレージプールを起動します。**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 guest_images_disk
Pool guest_images_disk started
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_disk    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使用できるようになりました。

16.1.2. virsh を使用してストレージプールを削除する

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
   ```

16.2. 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 ext4 でフォーマットされた 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。

16.2.1. virt-manager を使用してパーティションベースのストレージプールを作成する

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16.1 virt-manager を使用してパーティションベースのストレージプールを作成する

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。

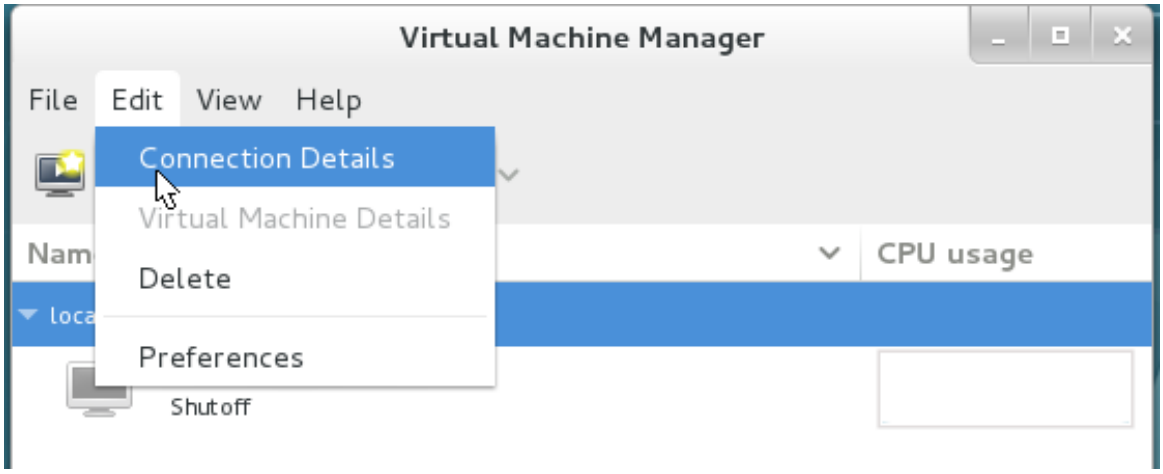


図16.1 接続の詳細

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。

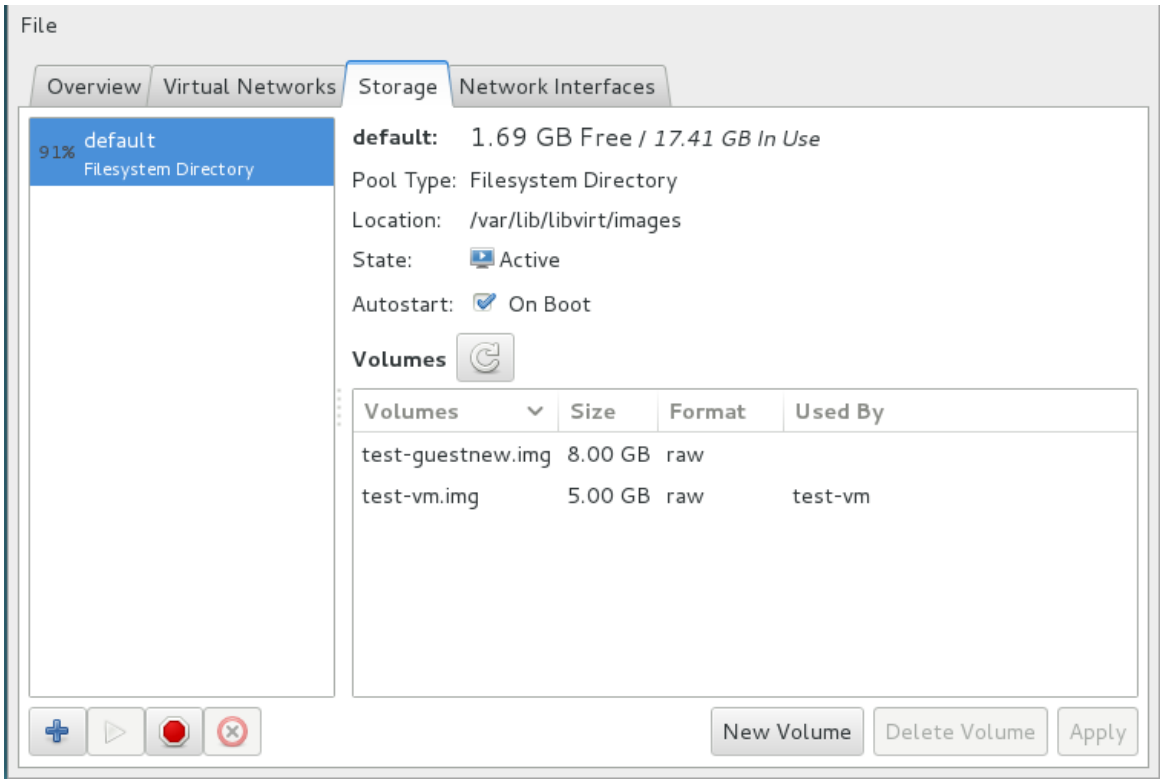


図16.2 ストレージタブ

2. **新規ストレージプールを作成します。**

a. **新規プールの追加 (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。

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: Pre-Formatted Block Device`** に変更します。

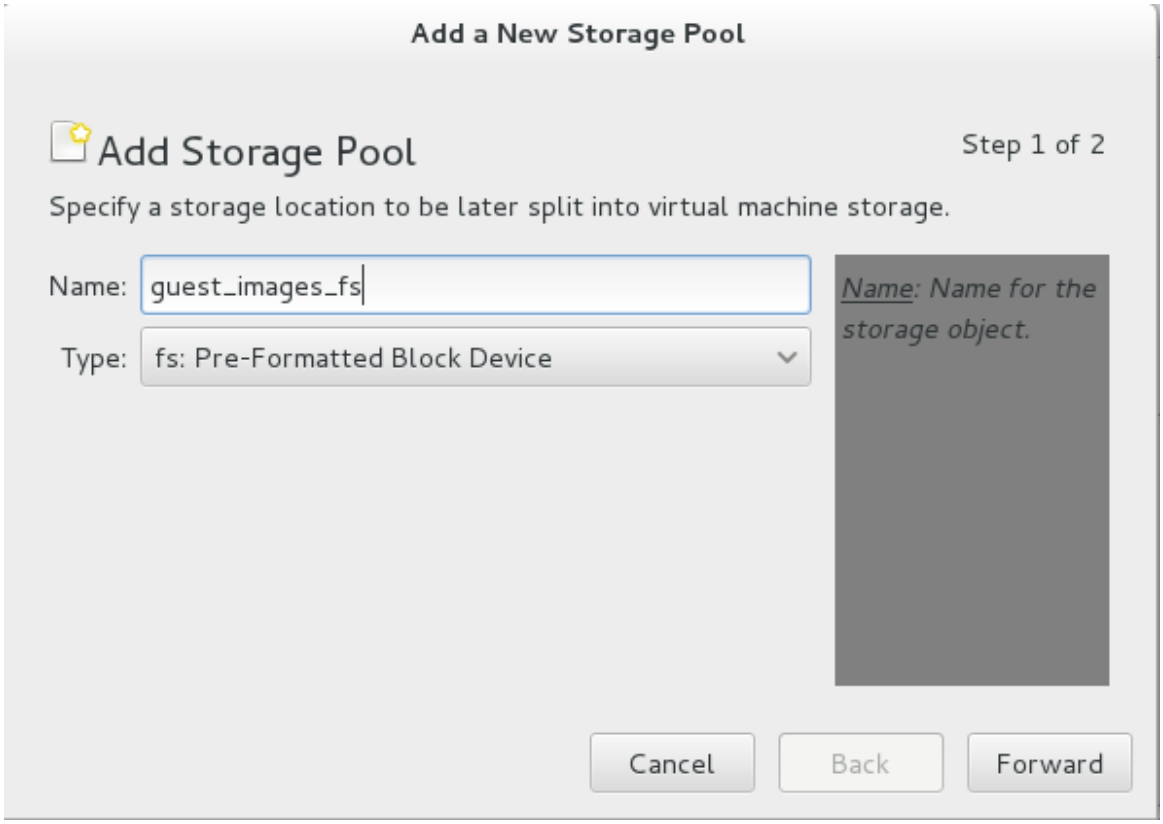


図16.3 ストレージプールの名前と種類

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、**フォーマット**、**ソースパス** の各フィールドを変更します。

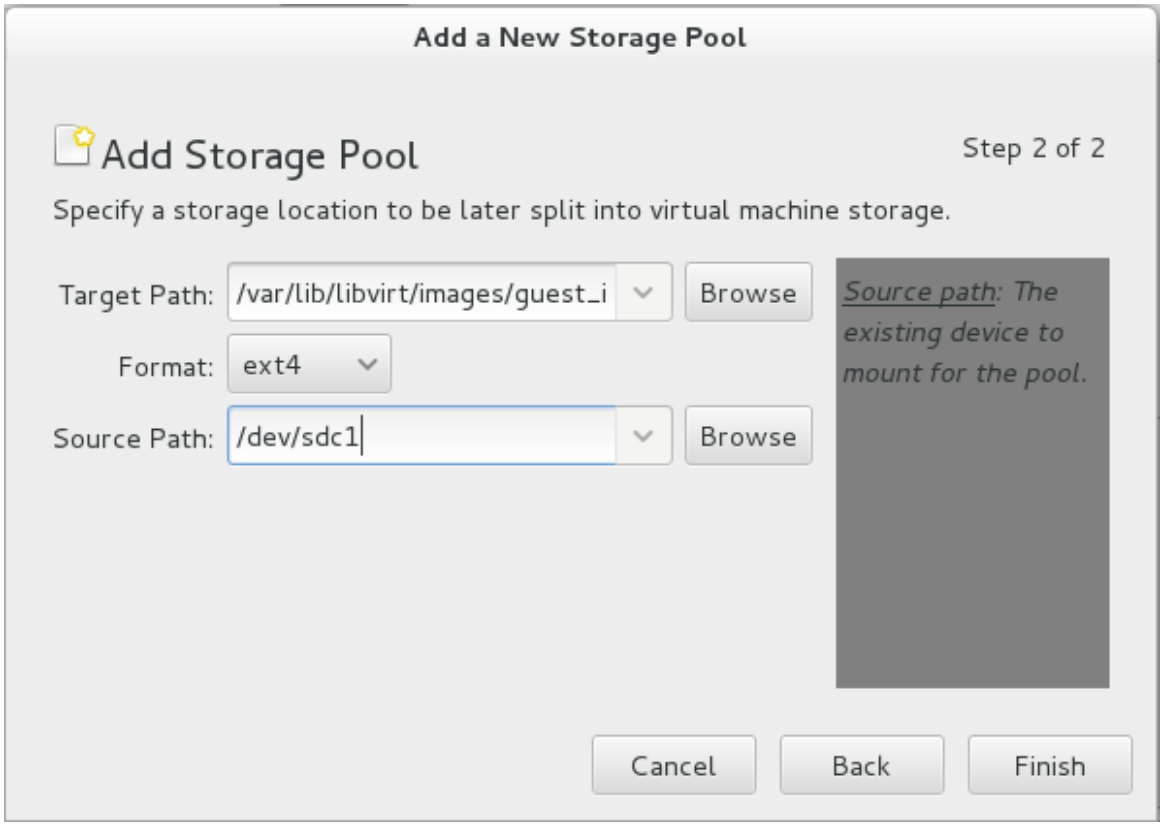


図16.4 ストレージプールのパスとフォーマット

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。

フォーマット

フォーマット 一覧からフォーマットを選択します。デバイスはこの選択したフォーマットを使ってフォーマットされます。

この例では、*ext4* ファイルシステムを使用しています。これはデフォルトの Red Hat Enterprise Linux ファイルシステムです。

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。

3. **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、サイズは *458.20 GB Free* です。**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。

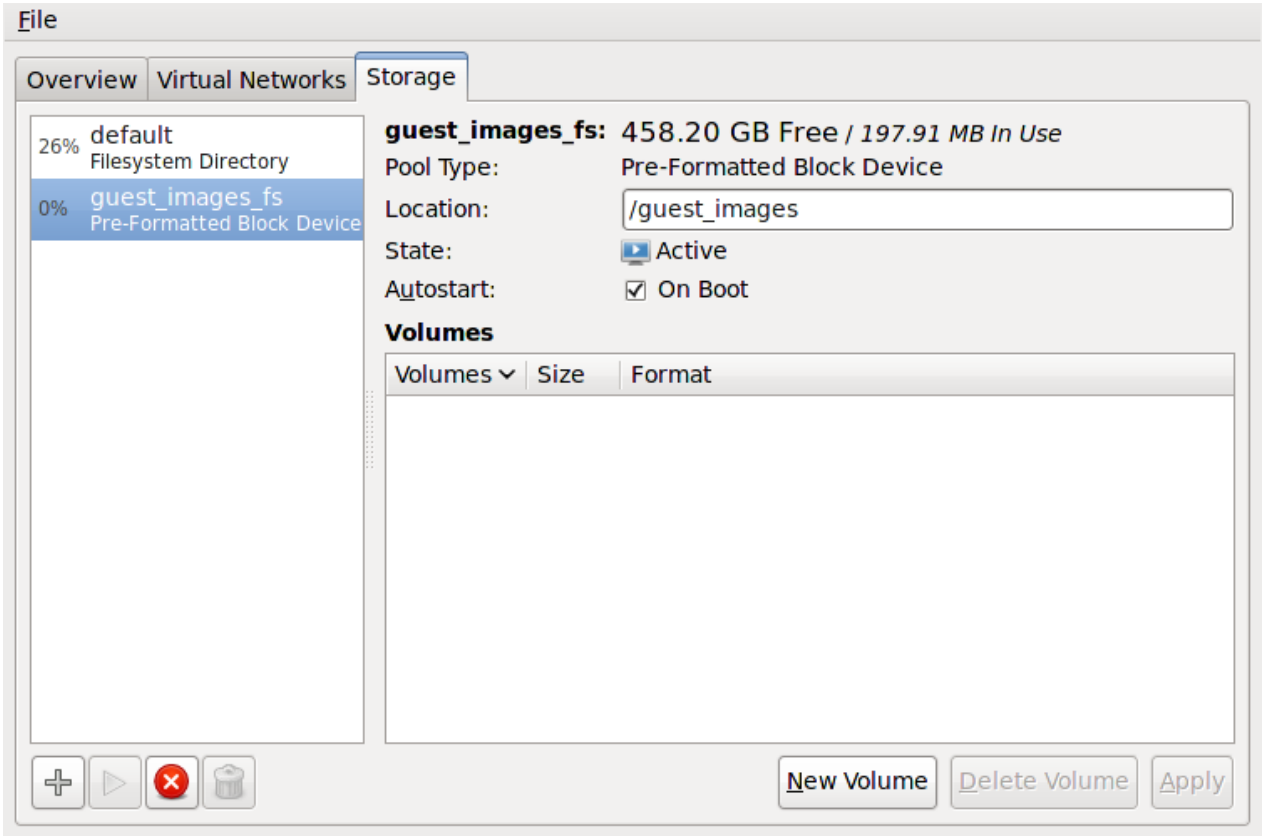


図16.5 ストレージ一覧の確認

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。

16.2.2. virt-manager を使用してストレージプールを削除する

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赤い X アイコンをクリックします。

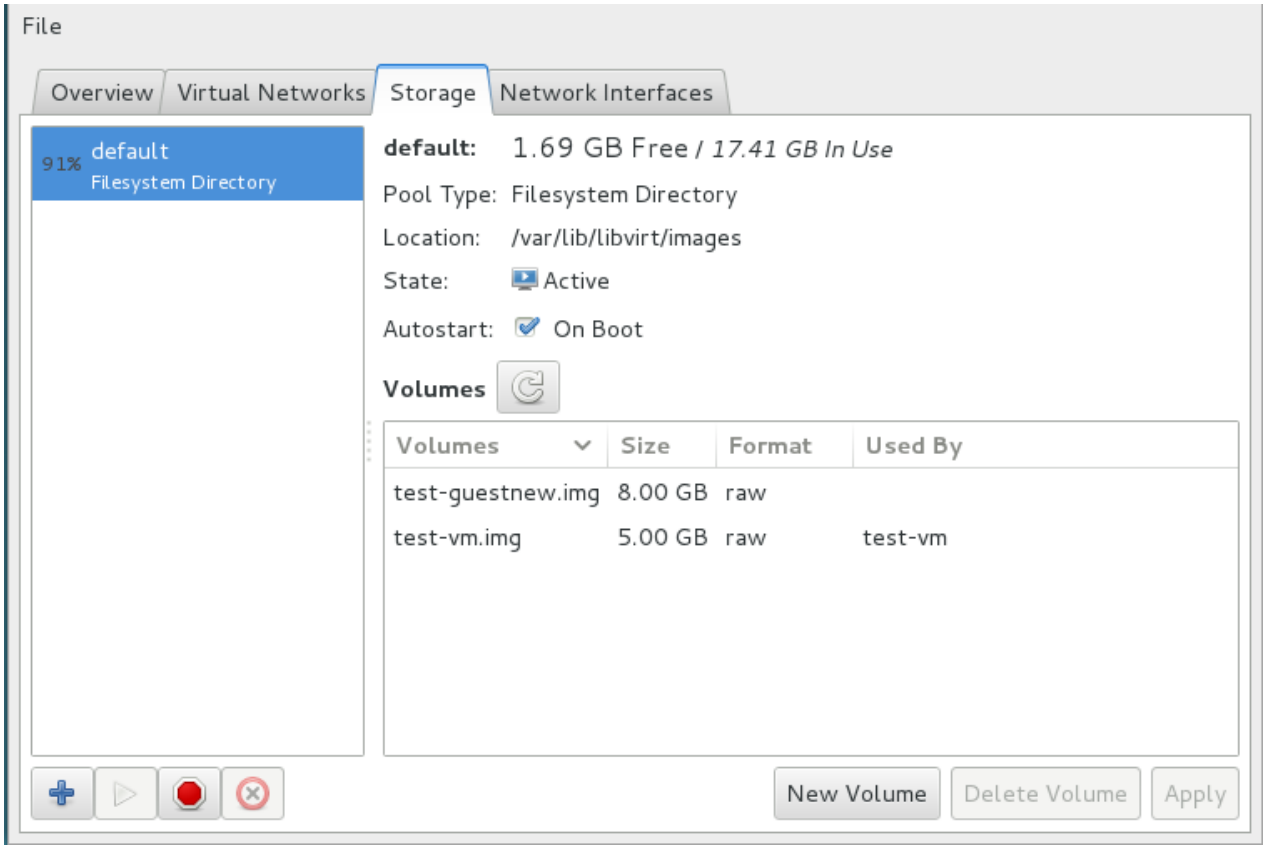


図16.6 停止アイコン

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。

16.2.3. virsh を使用してパーティションベースのストレージプールを作成する

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はゲストに与えないようにしてください。この手順はパーティション (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16.2 virsh を使用して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を作成する

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ーが存在しない場合は、**virsh** コマンドがその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_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_fs
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、およびサイズが予想通り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。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
```

16.2.4. virsh を使用してストレージプールを削除する

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
```

16.3. ディレクトリ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。

16.3.1. virt-manager を使用してディレクトリーベースのストレージプールを作成する

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。**

   ホスト物理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 virtual machine_images* という名前のディレクトリーを作成しています。

```
# 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。ティレクトリーは 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. **ディレクトリーのアクセス権を設定します。**

ディレクトリーのファイルアクセス権を変更します。

```
# chmod 700 /guest_images
```

d. **変更を確認します。**

アクセス権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。

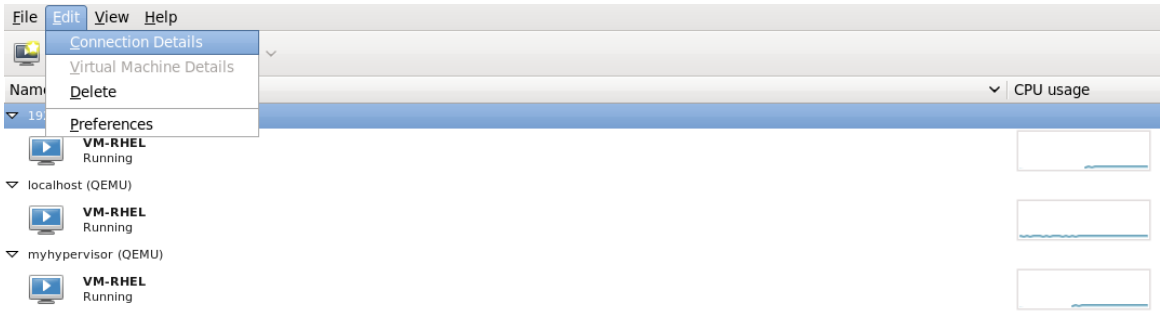


図16.7 接続の詳細ウィンドウ

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

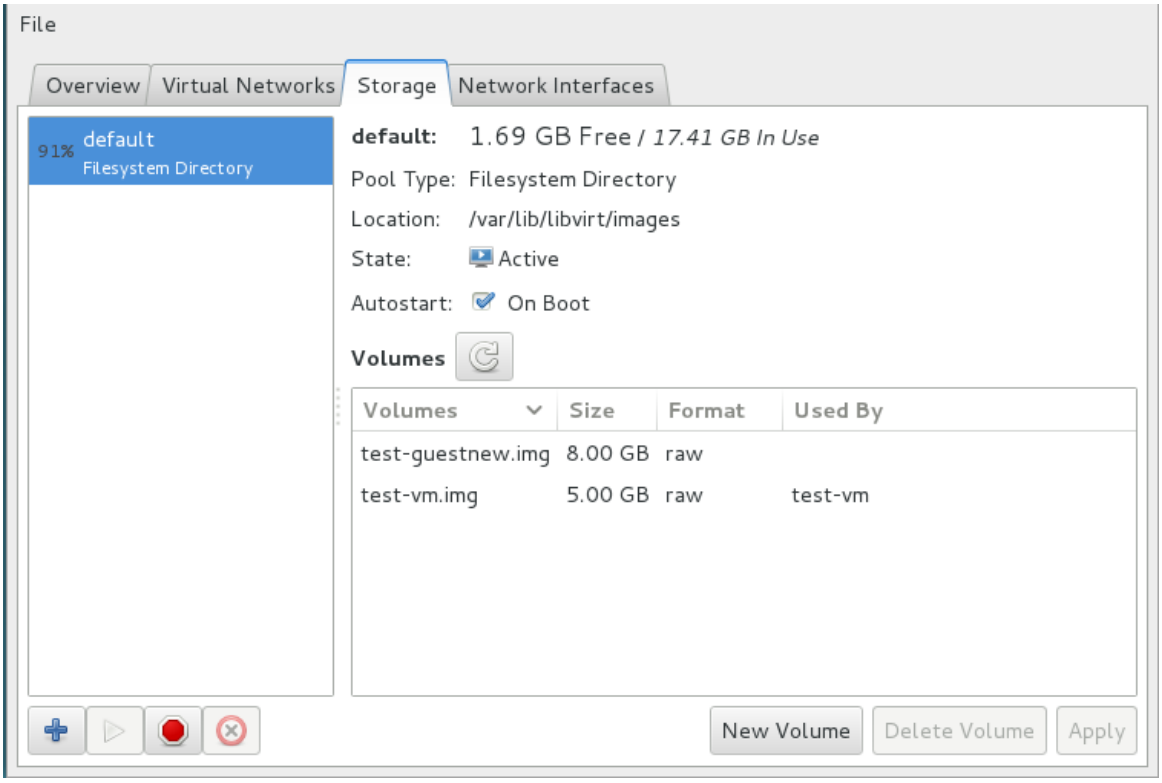


図16.8 ストレージタブ

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** の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。

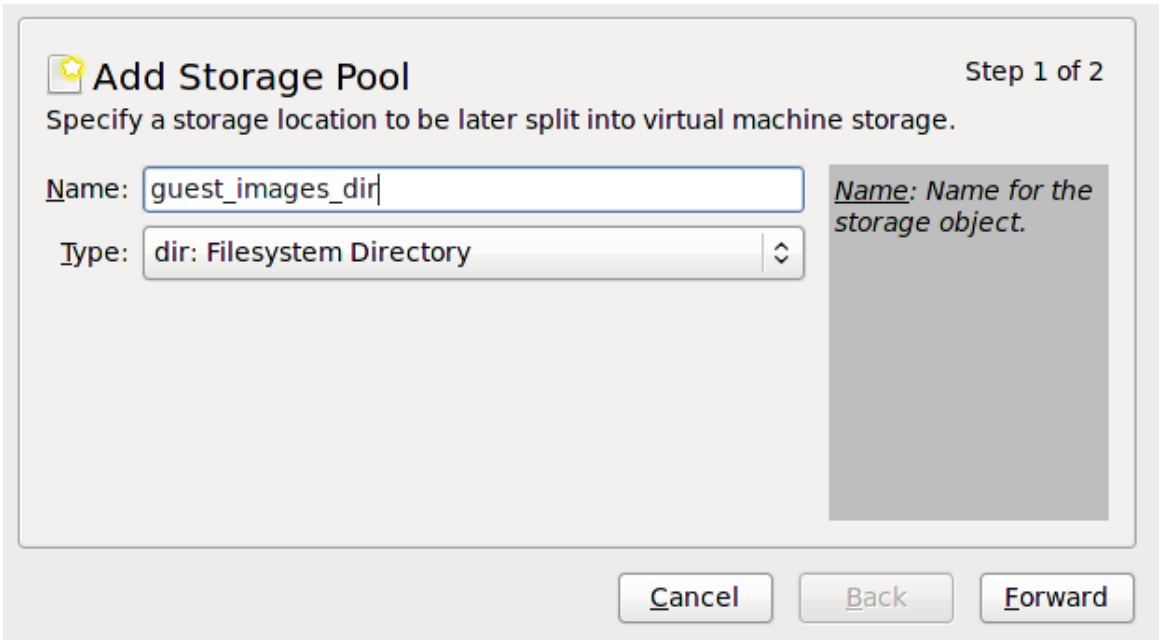


図16.9 ストレージプールに名前を付ける

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。

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。サイズが予測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、*36.41 GB Free* と表示されています。**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。

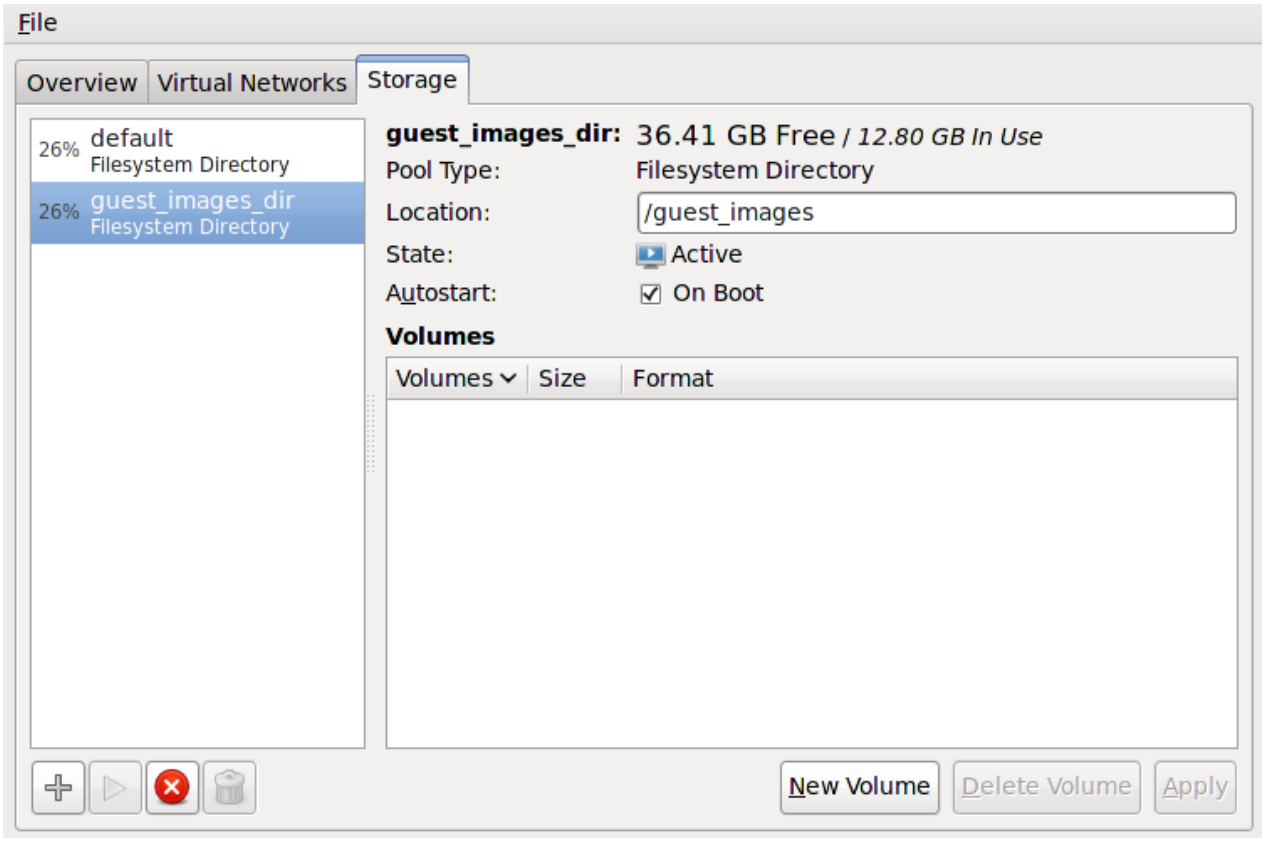


図16.10 ストレージプール情報の確認

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。

16.3.2. virt-manager を使用してストレージプールを削除する

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赤い X アイコンをクリックします。

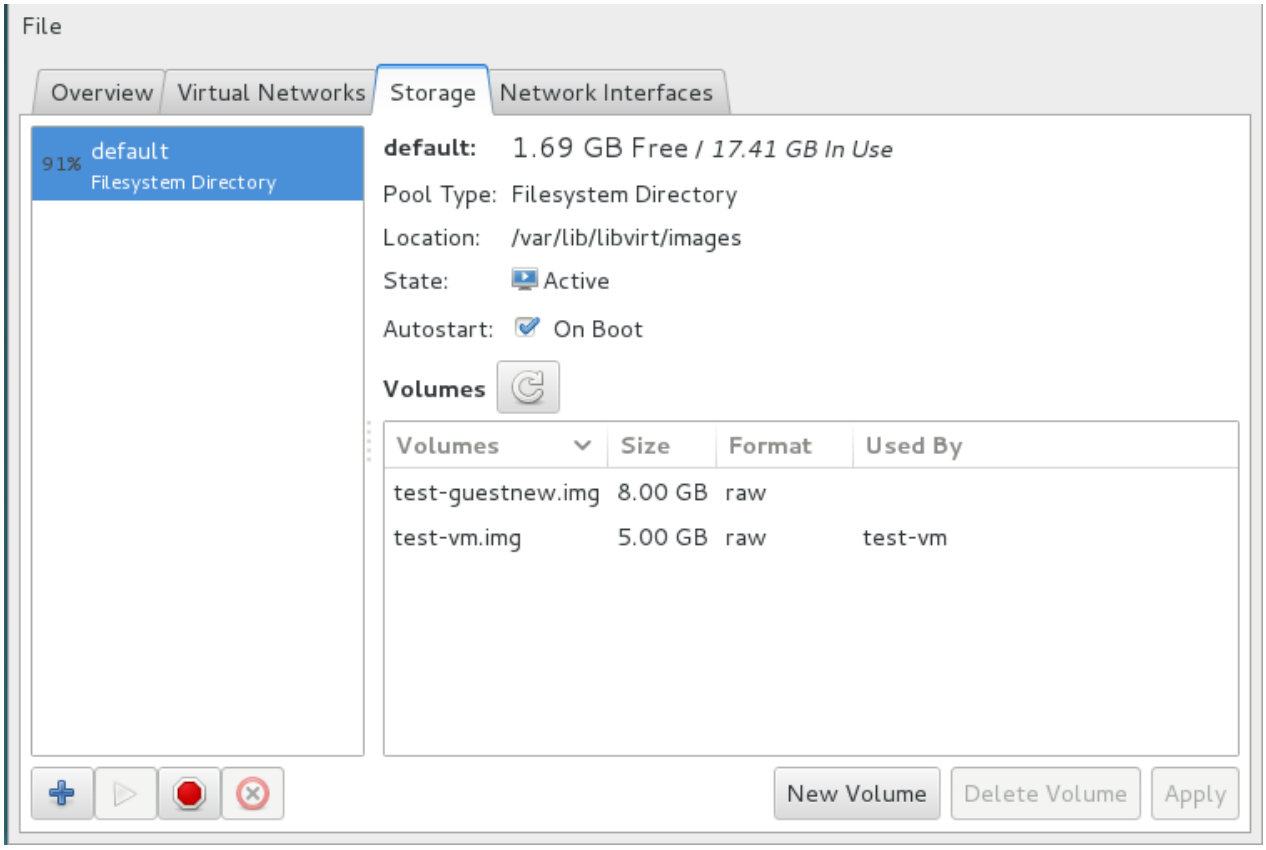


図16.11 停止アイコン

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。

16.3.3. virsh を使用してディレクトリーベースのストレージプールを作成する

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。ディレクトリー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。この例のこれ以降で使用される **`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。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。

16.3.4. virsh を使用してストレージプールを削除する

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
```

16.4. LVM ベースのストレージプール

この章では、 LVM ボリュームグループをストレージプールとして使用する方法について説明しています。

LVM ベースのストレージプールでは、 LVM の柔軟性を十分に利用することができます。

注記

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ご利用できません。

注記

LVM の詳細は、『Red Hat Enterprise Linux ストレージ管理ガイド』 を参照して下さい。

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にする場合、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をお勧めします。

16.4.1. virt-manager を使用して LVM ベースのストレージプールを作成する

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して使用することができます。

1. **オプション: LVM ボリューム用に新規のパーティションを作成します**

   以下のステップは、新しいハードディスクドライブ上に新規のパーティションまたは LVM ボリュームグループを作成する方法を説明しています。

   警告

   以下の手順は選択したストレージデバイスから全てのデータを抹消します。

   a. **新規のパーティションを作成します。**

      fdisk コマンドを使用してコマンドラインから新しいディスクパーティションを作成します。以下の例では、ストレージデバイス **/dev/sdb** 上でディスク全体を使用する新しいパーティションを作成します。

      ```
      # fdisk /dev/sdb
      Command (m for help):
      ```

      n を押して新規のパーティションを作成します。

   b. ***p*** を押してプライマリーパーティションを選択します。

      ```
      Command action
         e   extended
         p   primary partition (1-4)
      ```

c. 選択可能な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
```

d. ***Enter*** を押して、デフォルトの 1 番目のシリンダーを入力します。

```
First cylinder (1-400, default 1):
```

e. パーティションのサイズを選択します。この例では、***Enter*** を押した後に、ディスク全域が割り当てられています。

```
Last cylinder or +size or +sizeM or +sizeK (2-400, default
400):
```

f. ***t*** を押してパーティションのタイプを設定します。

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

g. 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は ***1*** です。

```
Partition number (1-4): 1
```

h. Linux LVM パーティションの ***8e*** を記入します。

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

i. 変更をディスクに書き込んで終了します。

```
Command (m for help): w
Command (m for help): q
```

j. **新規 LVM ボリュームグループを作成します。**

vgcreate コマンドで新しい LVMボリュームグループを作成します。この例では、*guest_images_lvm* という名前のボリュームグループを作成しています。

```
# vgcreate guest_images_lvm /dev/sdb1
  Physical volume "/dev/vdb1" successfully created
  Volume group "guest_images_lvm" successfully created
```

これで新規の LVM ボリュームグループ *guest_images_lvm* が LVM ベースのストレージプールで使用できるようになりました。

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。

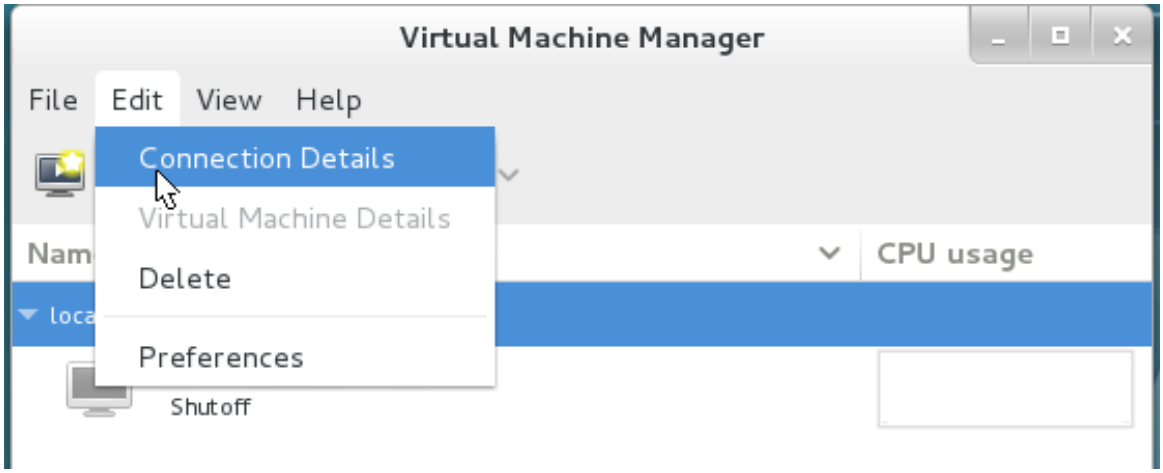


図16.12 接続の詳細

b. **ストレージ** タブをクリックします。

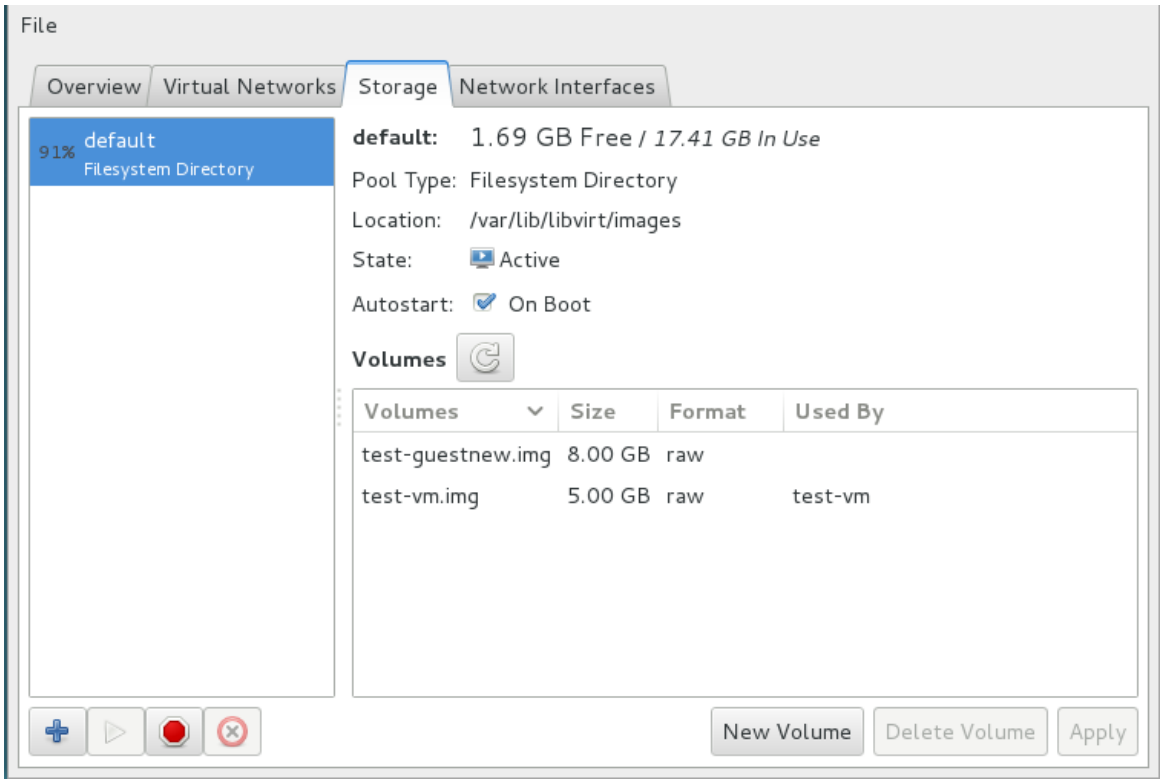


図16.13 ストレージタブ

3. **新規ストレージプールを作成します。**

a. **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の追加** ウィザードが出現します。

ストレージプール用に **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。

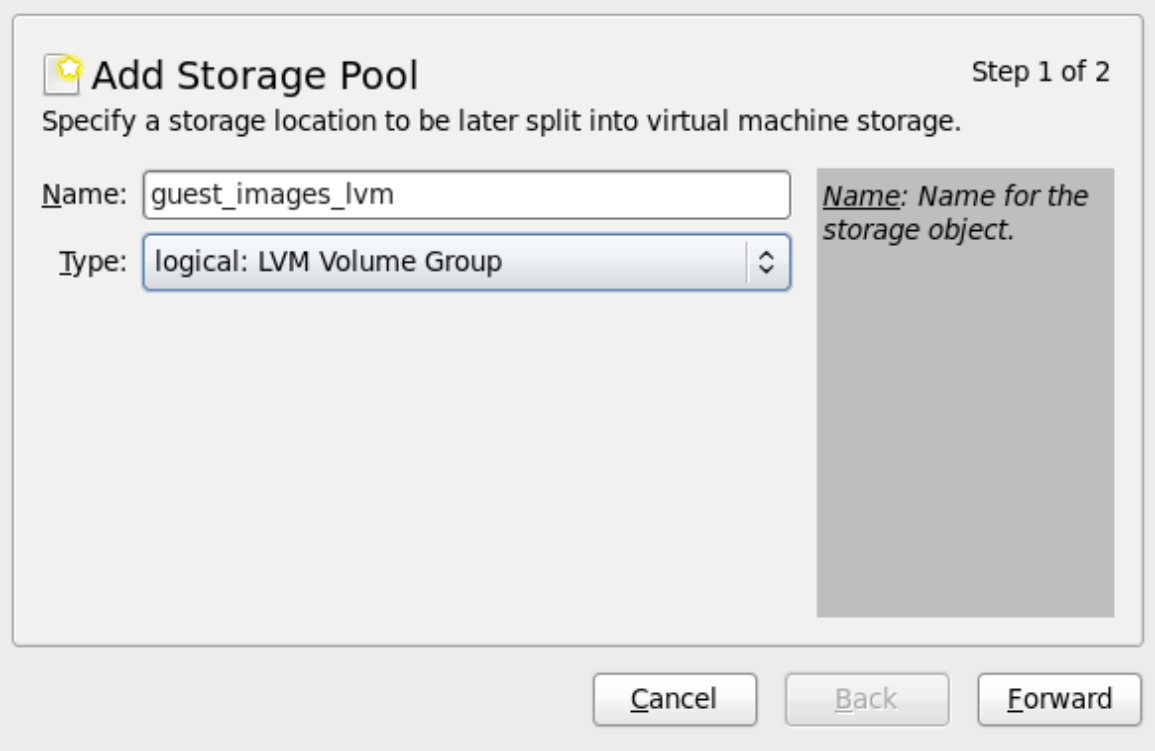


図16.14 LVM ストレージプールの追加

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 フィールドを変更します。この例では、*/guest_images* を使用しています。

次に **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** の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、**/dev/***storage_pool_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** の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** の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** のチェックボックスを選択しておく必要があります。

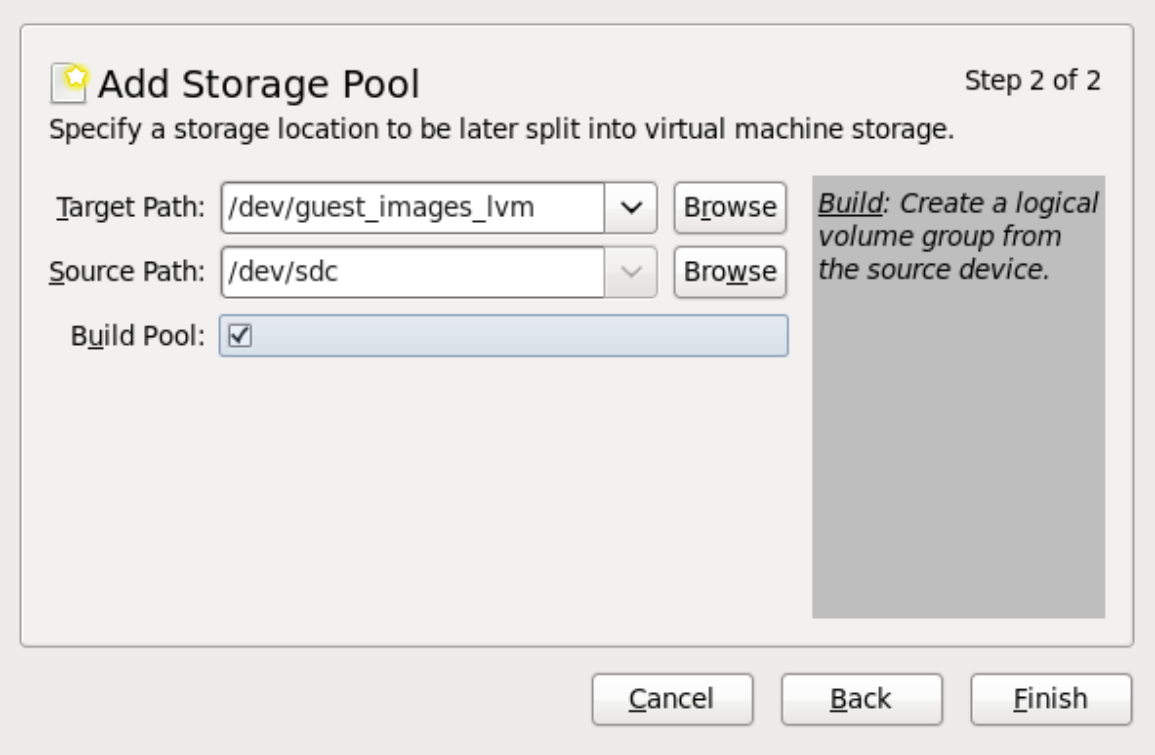


図16.15 ターゲットとソースの追加

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。

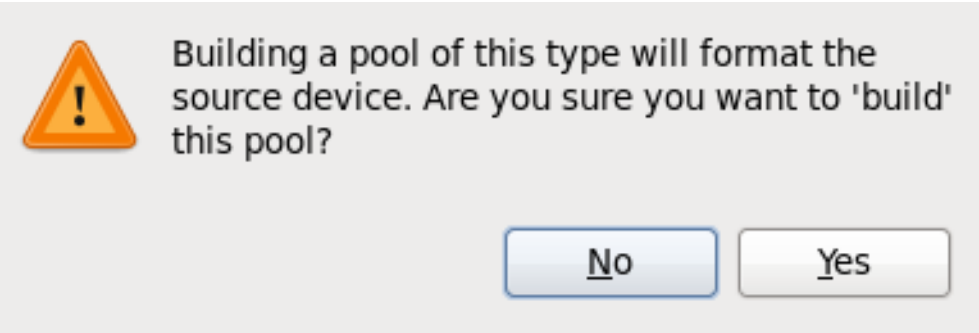


図16.16 警告メッセージ

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。

4. **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、*465.76 GB Free* 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、libvirtd 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と便利です。

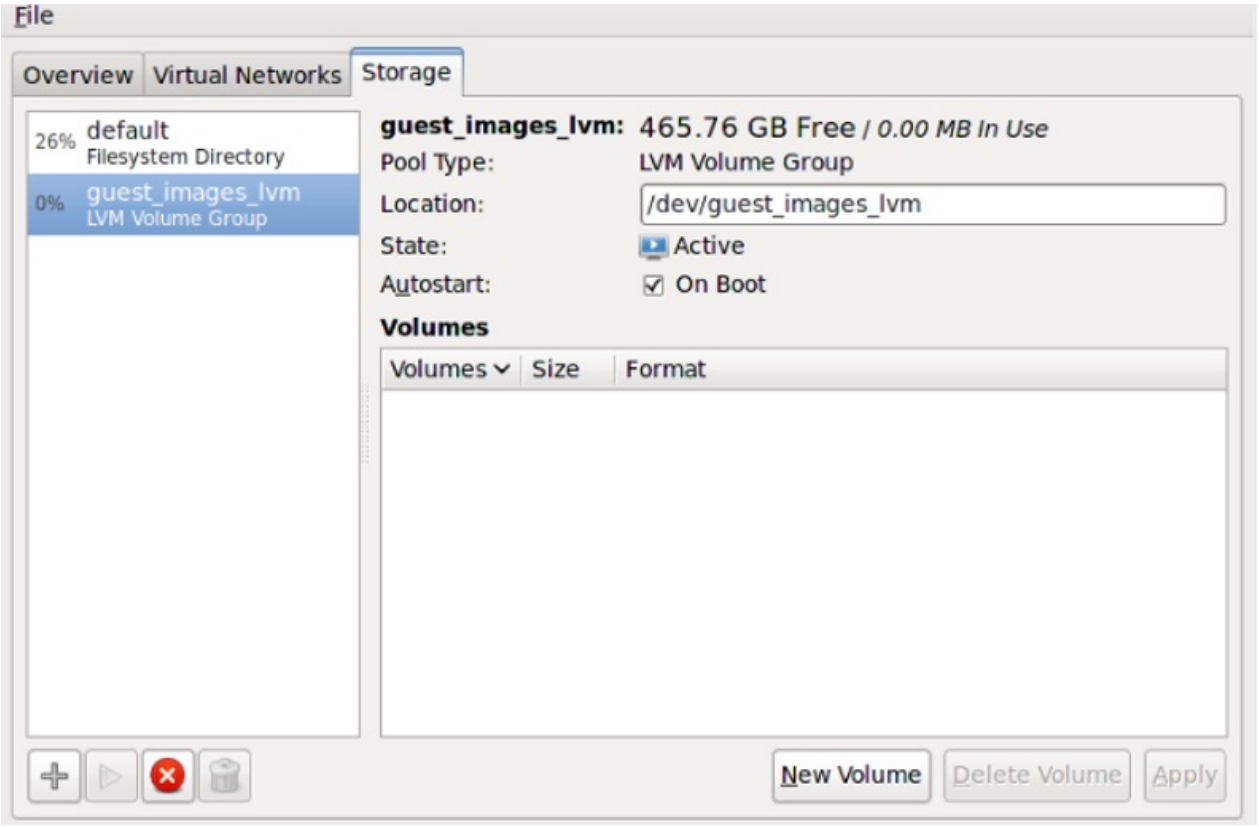


図16.17 LVM ストレージプールの詳細確認

これで作業は完了です。ホストの詳細ダイアログを閉じます。

16.4.2. virt-manager を使用してストレージプールを削除する

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、ストレージウィンドウの下部にある赤い X アイコンをクリックします。

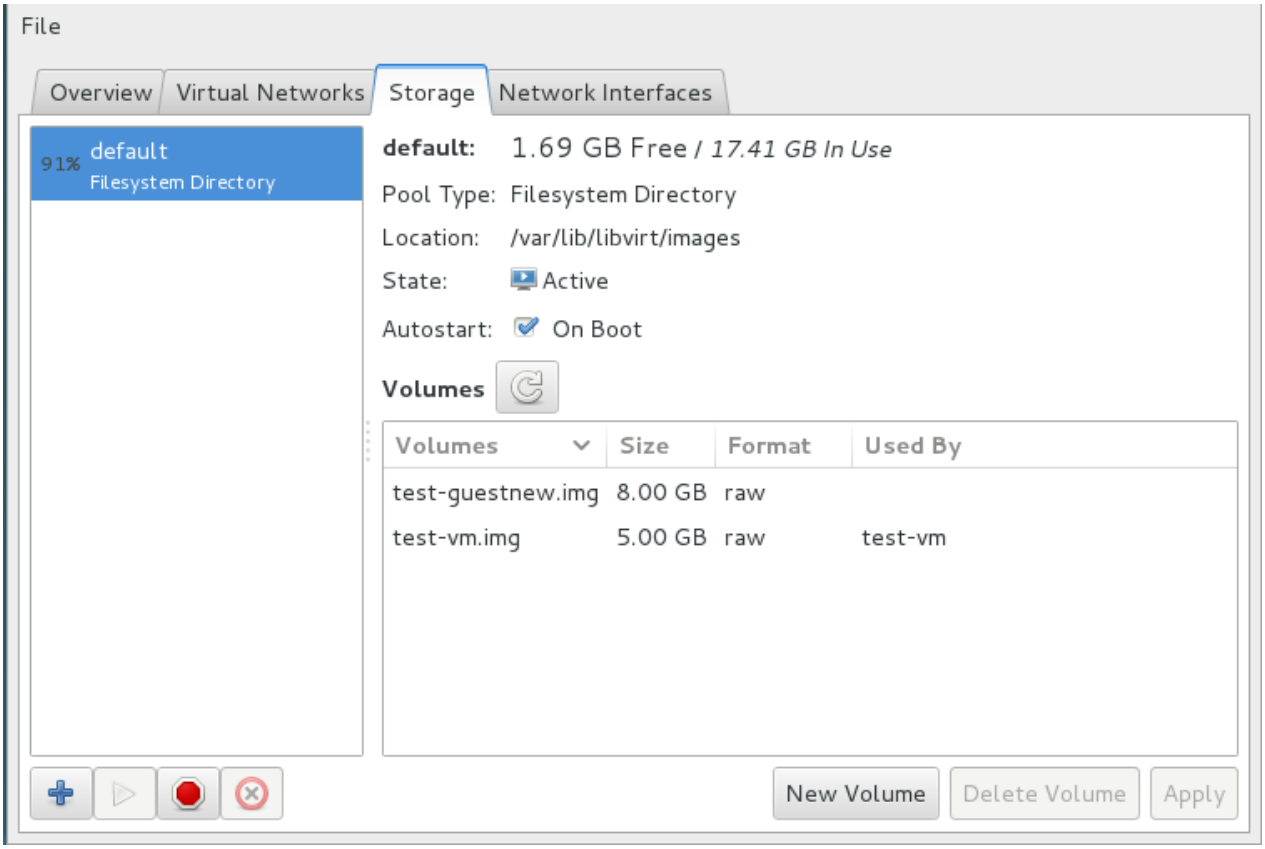


図16.18 停止アイコン

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。

16.4.3. virsh を使用して LVM ベースのストレージプールを作成する

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、単一ドライブ (**/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、説明を行なうための単なる例ですので、実際の設定を行なう場合には適した値に置き換えてください。

手順16.3 virsh を使用して LVM ベースのストレージプールを作成する

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. 指定した名前に応じ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
```

16.4.4. virsh を使用してストレージプールを削除する

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
```

16.5. 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。

16.5.1. ソフトウェア iSCSI ターゲットを設定する

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16.4 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、ステップ 3 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** ディレクトリーか、またはたとえば、**iscsi>iqn.iqn.2010-05.com.example.server1:iscsirhel7guest** 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- tgp1
.....................................................................
..............[enabled, auth]
   o-
```

```
acls.....................................................
.....................[0 ACL]
   o-
luns.....................................................
.....................[3 LUNs]
   | o-
lun0.....................................................
...........[ramdisk/ramdisk1]
   | o-
lun1.....................................................
.....[block/block1 (dev/vdb1)]
   | o-
lun2.....................................................
......[fileio/file1 (foo.img)]
   o-
portals..................................................
.....................[1 Portal]
     o- 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。これは、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、次回の起動時に保存された設定を適用します。

手順16.5 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 のバッキング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 導入ガイド』を参照してください。

   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も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. デバイスを切り離します。

```
# 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。

16.5.2. virt-manager で iSCSI ストレージプールを作成する

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16.6 iSCSI デバイスを virt-manager に追加する

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。

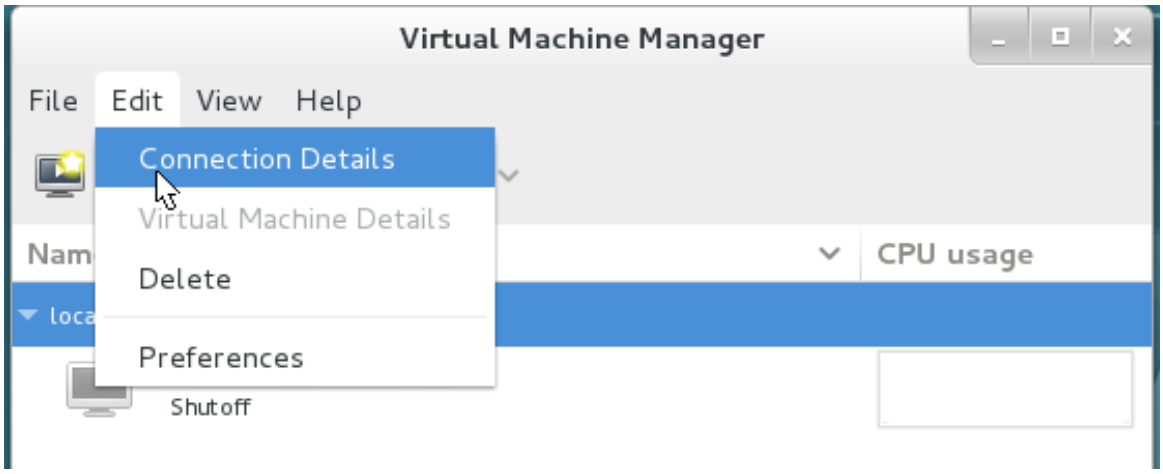


図16.19 接続の詳細

   b. **ストレージ** タブをクリックします。

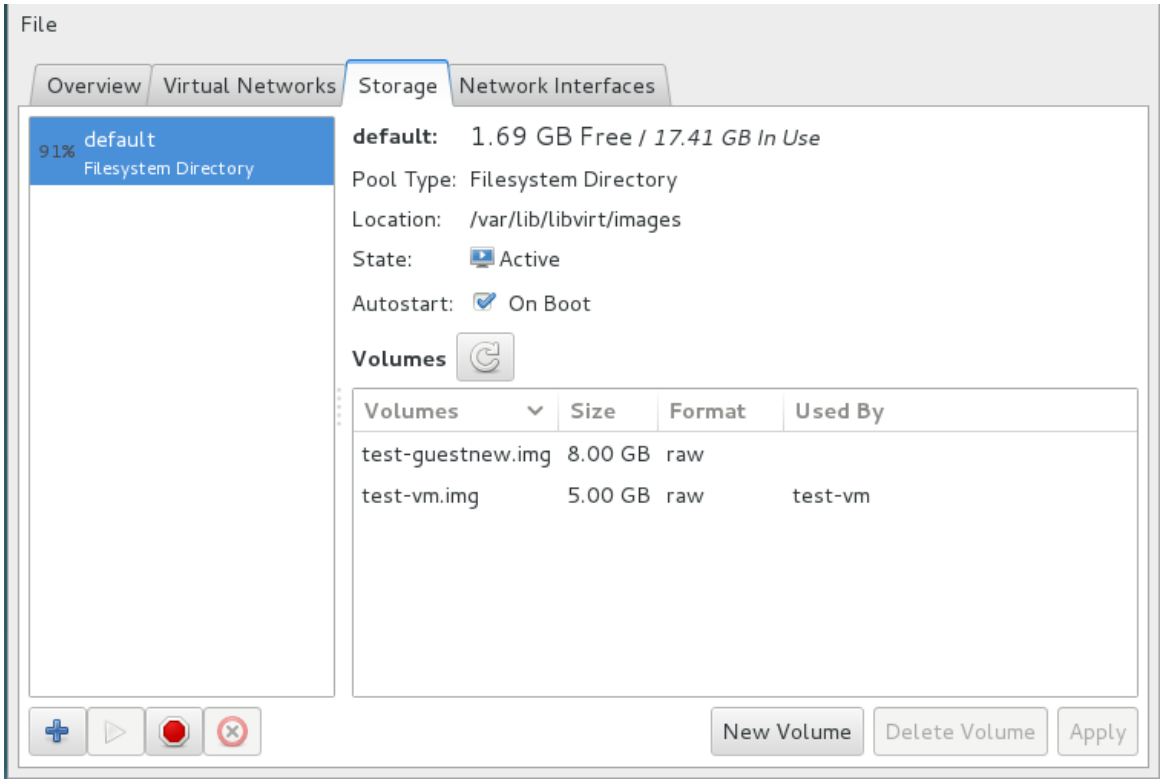


図16.20 ストレージメニュー

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の追加** ウィザードが表示されます。

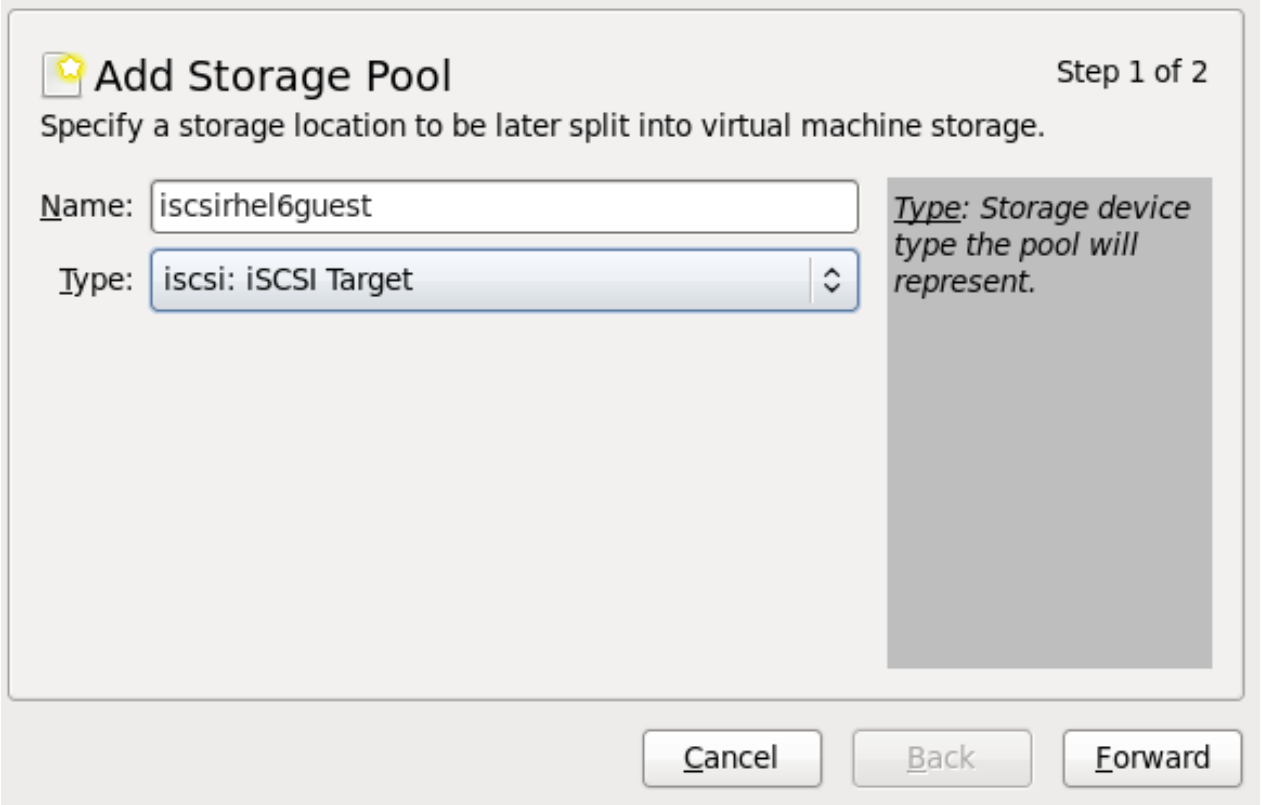


図16.21 iSCSI ストレージプールの名前とタイプを追加します。

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** を押して先に進みます。

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」および ステップ 6 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. **ソースパス** 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」の ステップ 6 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.example.server1:iscsirhel7guest** を使用しています。

   d. **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:iscsirhel7** 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。

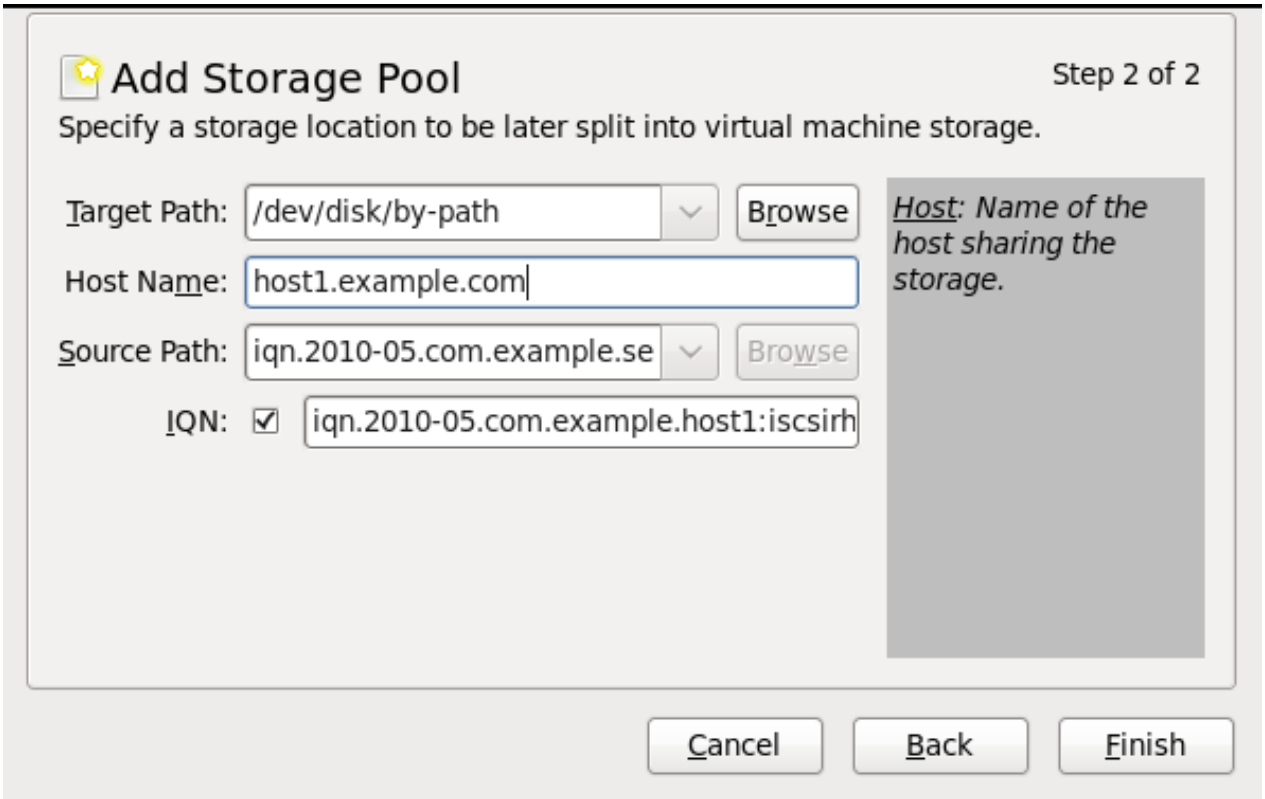


図16.22 iSCSI ストレージプールの作成

16.5.3. virt-manager を使用してストレージプールを削除する

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赤い X アイコンをクリックします。

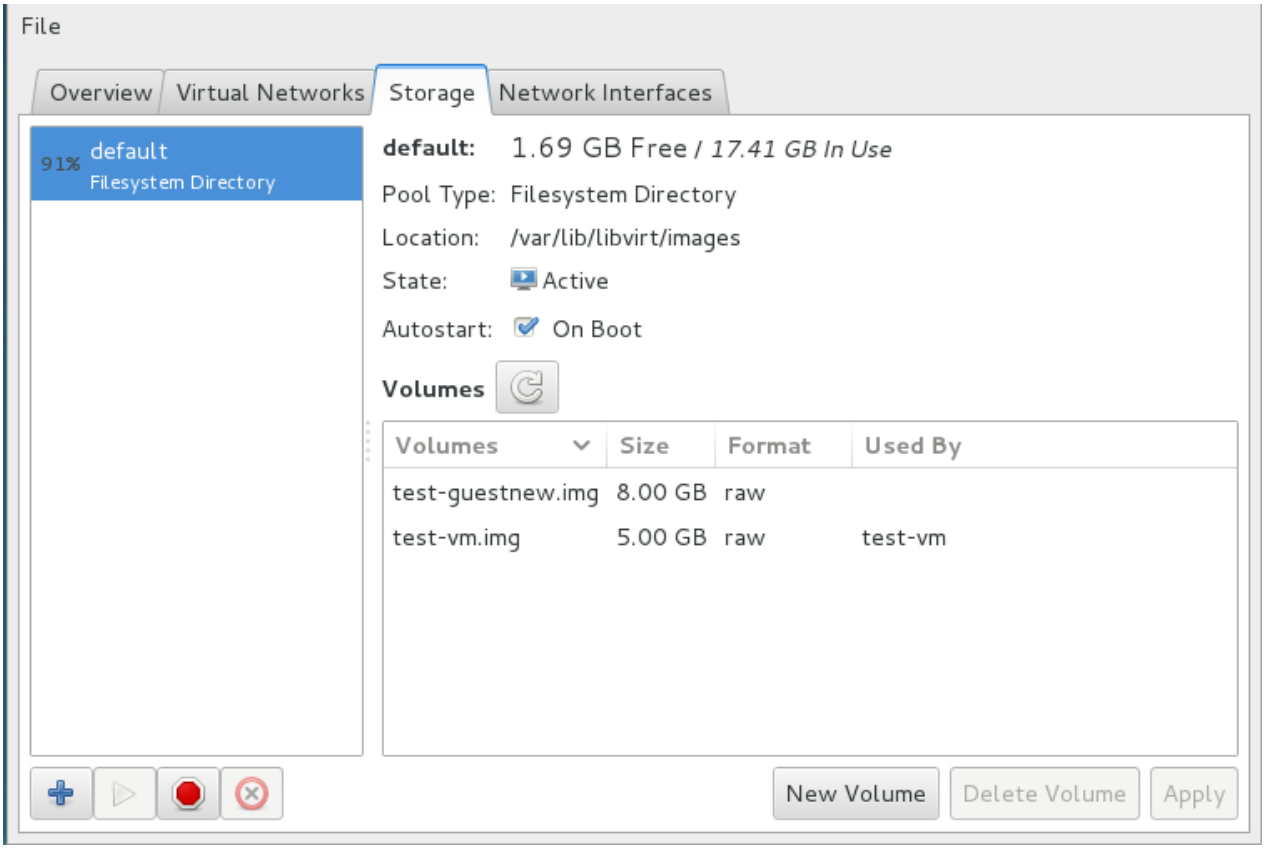


図16.23 停止アイコン

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。

16.5.4. virsh を使用して iSCSI ベースのストレージプールを作成する

1. **オプション: ストレージプールのセキュリティーを保護します。**

   必要な場合は、「iSCSI ストレージプールのセキュリティー保護」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、以下の形式で使用します。

```
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

ホストマシン上に 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'/>
      <!--you can change mode to mode='host' for a connection to
the host physical machine-->
```

```
      <target dev='vda' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06'
function='0x0'/>
   </disk>
   ...
```

図16.24 ディスクタイプ要素 XML の例

ステップ 9 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。

16.5.5. 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16.7 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 を使用して iSCSI ベースのストレージプールを作成する」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 を使用して iSCSI ベースのストレージプールを作成する」にあるステップに従ってストレージを定義し、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
```

16.5.6. virsh を使用してストレージプールを削除する

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
```

16.6. NFS ベースのストレージプール

以下の手順では、**virt-manager** で NFS マウントポイントを持つストレージプールを作成します。

16.6.1. virt-manager を使用して NFS ベースのストレージプールを作成する

1. **ホスト物理マシンのストレージタブを開きます**

   ホストの詳細 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。

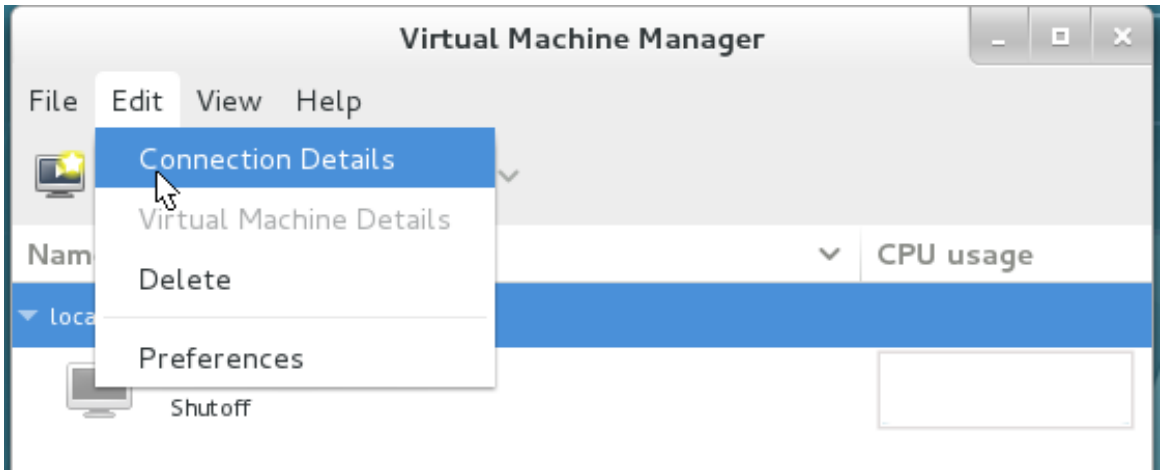


図16.25 接続の詳細

   c. ストレージタブをクリックします。

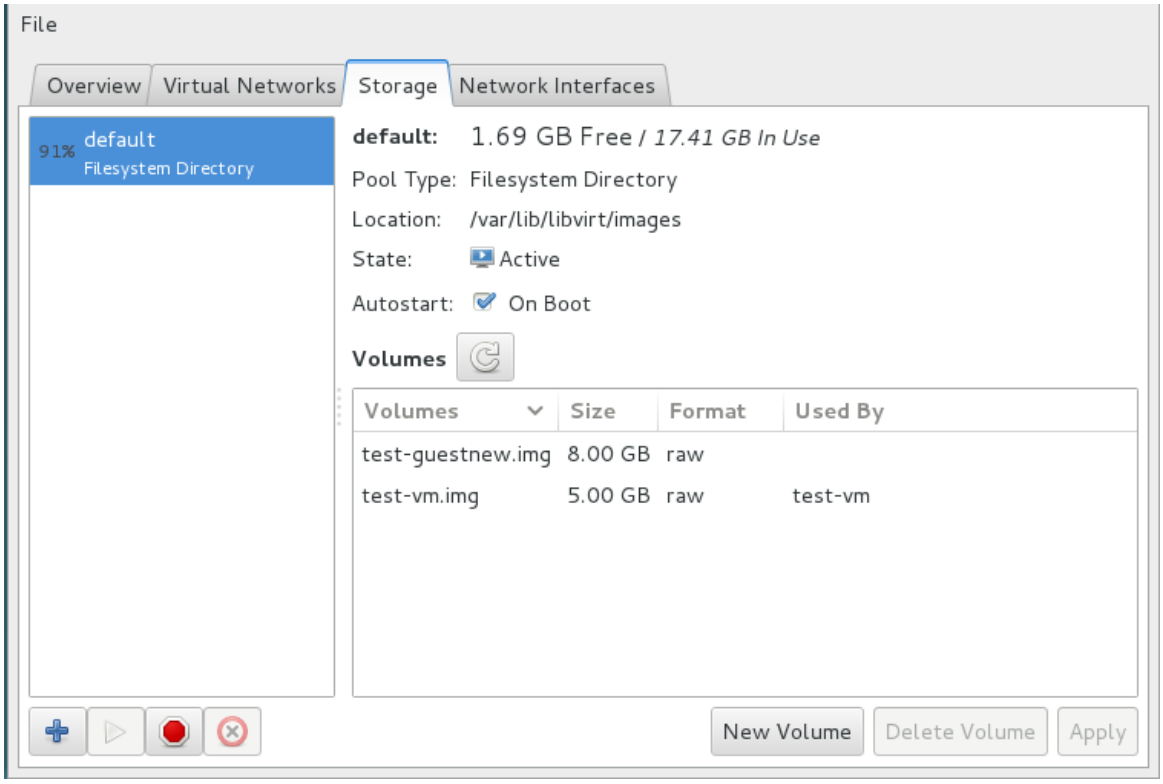


図16.26 ストレージタブ

2. **新規のプールを作成します (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の追加** ウィザードが表示されます。

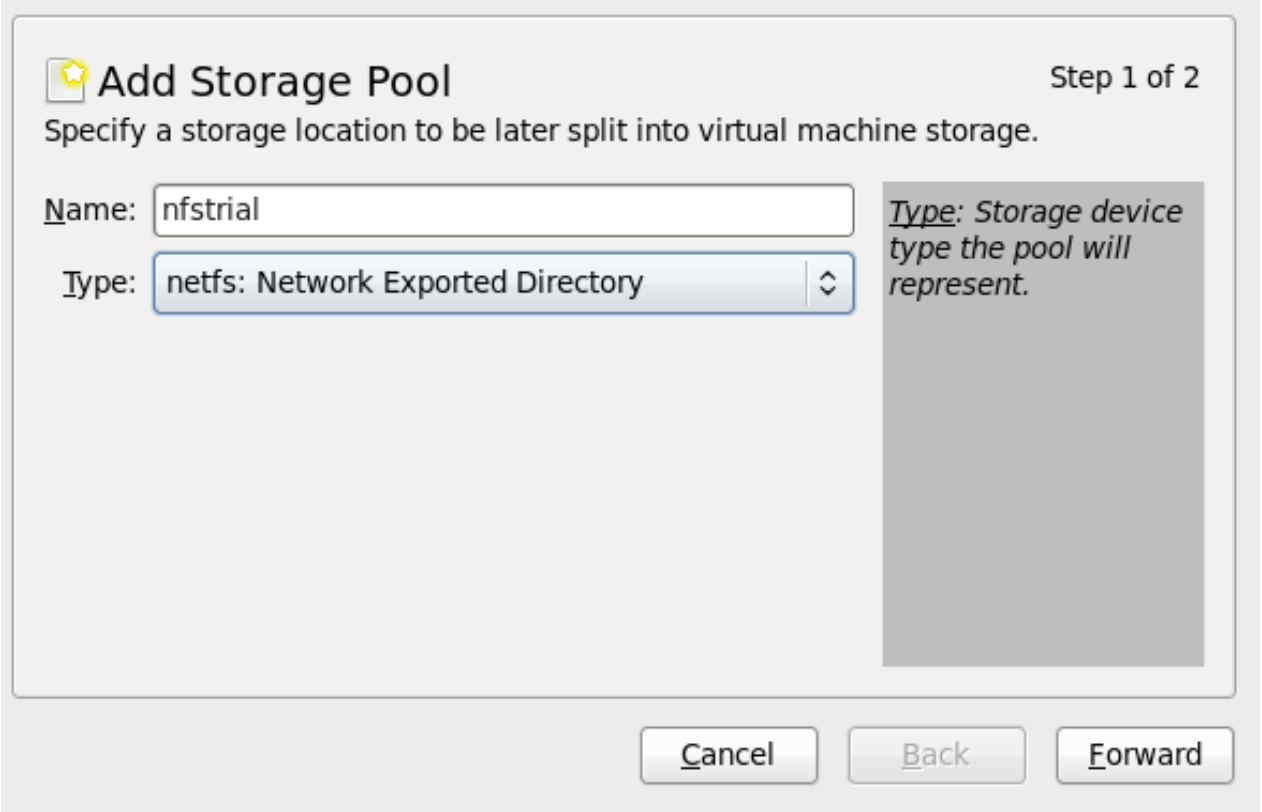


図16.27 NFS の名前と種類を追加

ストレージプールの名前を選択して、**進む** を押して先に進みます。

3. **新規のプールを作成します (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。

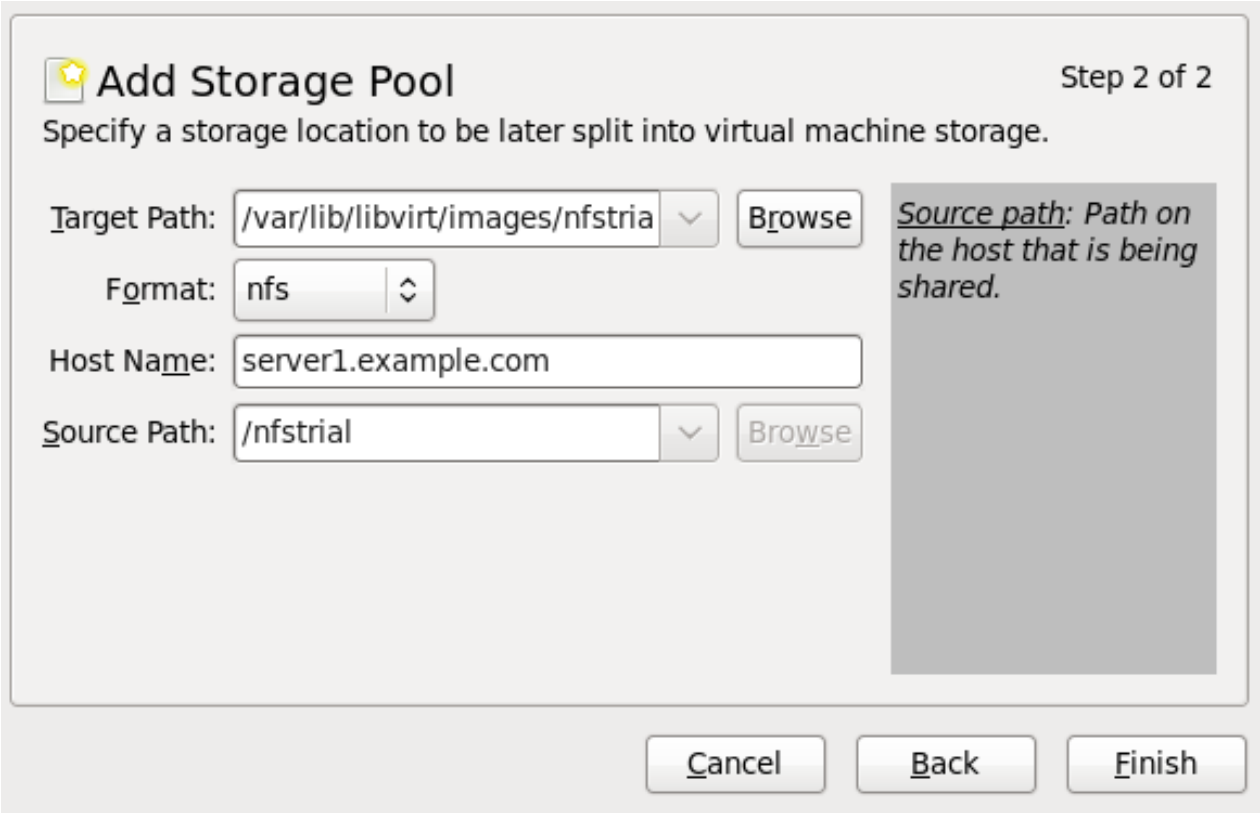


図16.28 NFS ストレージプールの作成

完了 を押すと、 新規のストレージプールが作成されます。

16.6.2. virt-manager を使用してストレージプールを削除する

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との問題を避けるため、 ストレージプールを停止して使用中のリソースをすべて解放するのが最適です。 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赤い X アイコンをクリックします。

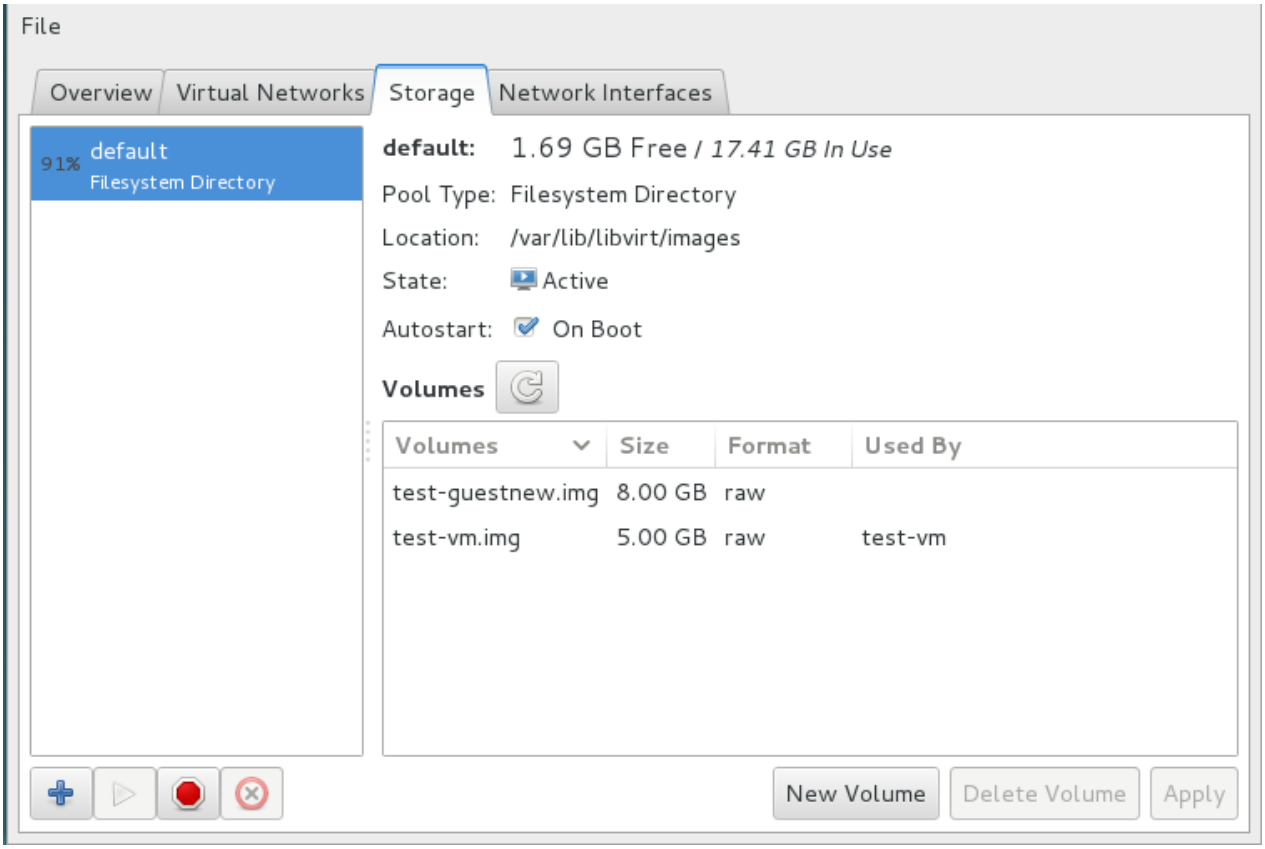


図16.29 停止アイコン

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。

16.7. SCSI デバイスで NPIV 仮想アダプター (vHBA) を使用する

NPIV (N_Port ID Virtualization) とは、単一の物理的なファイバーチャネル HBA を複数の仮想ポートと共有するために使用されるファイバーチャネル技術です。「仮想ホストバスアダプター」または vHBA としても知られる仮想ポートは、それぞれ独自の WWPN (World Wide Port Name) および WWNN (World Wide Node Name) で特定されます。仮想化では、vHBA は仮想マシンの LUN を制御します。

16.7.1. ホストシステム上で HBA を識別する

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_host4
scsi_host5
```

`virsh nodedev-dumpxml` コマンドからの XML 出力は、フィールドの**`<name>`**、**`<wwnn>`**、および **`<wwpn>`** 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。

以下の例は、HBA の XML のレイアウトを示しています。

```
# virsh nodedev-dumpxml scsi_host5
  <device>
    <name>scsi_host5</name>
    <parent>pci_0000_04_00_1</parent>
    <capability type='scsi_host'>
      <host>5</host>
      <capability type='fc_host'>
        <wwnn>2001001b32a9da4e</wwnn>
        <wwpn>2101001b32a9da4e</wwpn>
        <fabric_wwn>2001000dec9877c1</fabric_wwn>
      </capability>
      <capability type='vport_ops'>
        <max_vports>164</max_vports>
        <vports>5</vports>
      </capability>
    </capability>
  </device>
```

この例では、**<max_vports>** 値は HBA 設定で使用できる合計 164 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。

16.7.2. ノードデバイスドライバーを使用した vHBA の作成

ノードデバイスドライバーを使って vHBA を作成するには、利用可能な **vport** 領域を持つ HBA を選択し、HBA **<name>** フィールドを以下の XML の **<parent>** フィールドとして使用します。

```
  <device>
    <parent>scsi_host5</parent>
    <capability type='scsi_host'>
      <capability type='fc_host'>
      </capability>
    </capability>
  </device>
```

次に **virsh nodedev-create** コマンドで vHBA を作成します (上記の XML ファイルの名前が **vhba.xml** であるとします)。

```
  # virsh nodedev-create vhba.xml
  Node device scsi_host6 created from vhba.xml
```

注記

vHBA のカスタム **name**、**wwpn** または **wwnn** を指定することはできません。カーネルはまだ使用されていない順序で次の SCSI ホスト名を自動的に選択します。**wwpn** および **wwnn** 値は libvirt で自動生成されます。

生成された vHBA XML を表示するには、以下のように **virsh nodedev-dumpxml** コマンドを使用します。

```
# virsh nodedev-dumpxml scsi_host6
   <device>
     <name>scsi_host6</name>
     <parent>scsi_host5</parent>
     <capability type='scsi_host'>
       <capability type='fc_host'>
         <wwnn>2001001b32a9da5e</wwnn>
         <wwpn>2101001b32a9da5e</wwpn>
       </capability>
     </capability>
   </device>
```

重要

この vHBA は、ホストが再起動されない場合にのみ定義されます。永続的な vHBA を作成するには、「libvirt ストレージプールを使用した vHBA の作成」で説明されている libvirt ストレージプールを使用します。

16.7.3. libvirt ストレージプールを使用した vHBA の作成

ノードデバイスドライバーで管理される vHBA はホストの再起動時に永続化されません。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をお勧めします。

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、「vHBA LUN を使用するために仮想マシンを設定する」を参照してください。

永続的な vHBA 設定を作成するには、以下のように XML を使用して libvirt **'scsi'** ストレージプールを最初に作成します。

```
  <pool type='scsi'>
      <name>poolvhba0</name>
      <source>
        <adapter type='fc_host' wwnn='20000000c9831b4b'
wwpn='10000000c9831b4b'/>
      </source>
      <target>
        <path>/dev/disk/by-path</path>
        <permissions>
          <mode>0700</mode>
          <owner>0</owner>
          <group>0</group>
        </permissions>
      </target>
    </pool>
```

重要

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。**wwnn** および **wwpn** 属性は、作成される vHBA の固有 ID として指定されます。

オプションの **parent** 属性は **adapter** パラメーター内で、vHBA を作成するために HBA の名前を指定するために使用できます。この値には、ノードデバイスドライバーがダンプする値と一致している必要があります (例: *scsi_host5*)。これが指定されない場合、libvirt はサポートされる vport の最大数を超えない最初の NPIV 対応 HBA を選択します。

HBA の **'fc_host'** ソースアダプタータイプで SCSI プールを作成する場合、**parent** 属性は不要になります。

vHBA に使用する親 HBA を選択する場合は、以下のようにソースアダプター XML に **parent**、**wwnn**、および **wwpn** を指定する必要があります。

```
    <source>
      <adapter type='fc_host' parent='scsi_host5' wwnn='20000000c9831b4b'
wwpn='10000000c9831b4b'/>
    </source>
```

ストレージプール (この例では *poolvhba0.xml* 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 poolvhba0.xml
```

プールの開始:

```
# virsh pool-start poolvhba0
```

プールの破棄:

```
# virsh pool-destroy poolvhba0
```

プールを起動する際に、libvirt は同じ **wwpn:wwnn** を持つ vHBA がすでに存在するかどうかを検査します。vHBA が存在しない場合、**wwpn:wwnn** が指定された新規 vHBA が作成されます。これに応じて、プールを破棄する際には、libvirt は vHBA も破棄します。

最後に、後続ホストの起動により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には、以下のようにストレージプールの autostart 機能を設定できます (この例では、プールの名前は *poolvhba0* です)。

```
# virsh pool-autostart poolvhba0
```

16.7.4. vHBA 上で LUN を特定する

16.7.4.1. ストレージプールを使用して作成される vHBA の LUN を使用する

vHBA のストレージプールが作成された後は、vHBA 上で利用可能な LUN の一覧を生成するために **virsh vol-list** コマンドを使用します。以下にようになります。

```
   # virsh vol-list poolvhba0 --details
   Name        Path
   -----------------------------------------------------------------------
   unit:0:2:0 /dev/disk/by-path/pci-0000:04:00.1-fc-0x203500a0b85ad1d7-
lun-0 block
```

表示される LUN 名の一覧は、仮想マシン設定内のディスクボリュームとして使用できます。

16.7.4.2. ノードデバイスドライバーとして作成される vHBA の LUN を使用する

ノードデバイスドライバーを使用して作成される vHBA から利用できる LUN を検索する方法には以下の 2 つがあります。1 つ目は **virsh nodedev-list** コマンドを使用する方法で、2 つ目はホストのファイルシステムを手動で検索する方法です。

virsh nodedev-list --tree | more を使用して、vHBA が設定された親 HBA を検索します。以下の例は、*scsi_host5* HBA のツリーの関連部分を一覧表示しています。

```
          +- scsi_host5
              |
              +- scsi_host7
              +- scsi_target5_0_0
              |   |
              |   +- scsi_5_0_0_0
              |
              +- scsi_target5_0_1
              |   |
              |   +- scsi_5_0_1_0
              |
              +- scsi_target5_0_2
              |   |
              |   +- scsi_5_0_2_0
              |       |
              |       +- block_sdb_3600a0b80005adb0b0000ab2d4cae9254
              |
              +- scsi_target5_0_3
                  |
                  +- scsi_5_0_3_0
```

block_ は、これがブロックデバイスであることを示します。**sdb**_ は **/dev/sdb** の短いデバイスパスを表すために使用されます。短いデバイスパスまたは番号は、たとえば特定 LUN の **/dev/disk/by-{id,path,uuid,label}/** 名前空間を名前で検索するために使用できます。

```
  # ls /dev/disk/by-id/ | grep 3600a0b80005adb0b0000ab2d4cae9254
  scsi-3600a0b80005adb0b0000ab2d4cae9254

  # ls /dev/disk/by-path/ -l | grep sdb
  lrwxrwxrwx. 1 root root  9 Sep 16 05:58 pci-0000:04:00.1-fc-
0x203500a0b85ad1d7-lun-0 -> ../../sdb
```

virsh nodedev-list コマンドを使用する以外にも、以下のステップを使用して、LUN を見つけるために **/sys/bus/scsi/device** および **/dev/disk/by-path** ディレクトリーツリーを手動で検索することができます。

手順16.8 vHBA LUN を検索する

1. **/sys/bus/scsi/devices** ツリー以下に、vHBA の SCSI ホスト番号で始まるディレクトリーを表示します。

   たとえば SCSI ホスト番号が 6 の場合、以下のコマンドを使用します。

   ```
   # ls /sys/bus/scsi/devices/6:* -d
   /sys/bus/scsi/devices/6:0:0:0  /sys/bus/scsi/devices/6:0:1:0
   /sys/bus/scsi/devices/6:0:2:0  /sys/bus/scsi/devices/6:0:3:0
   ```

2. 以下のように SCSI ホストに属するすべてのエントリーの **block** 名を一覧表示します。

   ```
   # ls /sys/bus/scsi/devices/6:*/block/
   /sys/bus/scsi/devices/6:0:2:0/block/:
   sdc
   /sys/bus/scsi/devices/6:0:3:0/block/:
   sdd
   ```

   これは **scsi_host6** に 2 つの LUN があることを示し、1 つは*6:0:2:0* に割り当てられている短いデバイス名が *sdc* の LUN で、もう 1 つは *6:0:3:0* に割り当てられている短いデバイス名が*sdd* の LUN です。

3. LUN への安定したパスを判別します。

   sdc などのデバイス名には libvirt で使用する場合の安定性がありません。安定したパスを取得するには、**ls -l /dev/disk/by-path** を使用し、*sdc* パスを検索します。

   ```
      # ls -l /dev/disk/by-path/ | grep sdc
      lrwxrwxrwx. 1 root root  9 Sep 10 22:28 pci-0000:08:00.1-fc-
   0x205800a4085a3127-lun-0 -> ../../sdc
   ```

   これは、**/dev/disk/by-path/pci-0000:08:00.1-fc-0x205800a4085a3127-lun-0** がアドレス *6:0:2:0* に割り当てられた LUN の安定したパスであり、仮想マシンの設定で使用されることを示しています。

16.7.4.3. vHBA LUN を使用するために仮想マシンを設定する

このセクションでは、vHBA LUN を仮想マシン設定に追加する方法を詳しく説明します。これは、**virsh edit** コマンドを使用して仮想マシン XML を変更することによって設定されます。

16.7.4.3.1. ストレージプールを使用して作成される vHBA の LUN を使用する

vHBA LUN の仮想マシンへの追加は、以下の XML サンプルを使って仮想マシン上にディスクボリュームを作成することによって実行されます。

```
<disk type='volume' device='disk'>
  <driver name='qemu' type='raw'/>
  <source pool='poolvhba0' volume='unit:0:2:0'/>
  <target dev='hda' bus='ide'/>
</disk>
```

とくに、ストレージプールと短いボリューム名を一覧表示する **pool** および **volume** 属性と共に **<source>** パラメーターが使用されていることに注意してください。

16.7.4.3.2. ノードデバイスドライバーを使用して作成される vHBA の LUN を使用する

仮想マシンでの vHBA の設定は、安定したパス (**{by-id|by-path|by-uuid|by-label}** のパス) を使用して実行できます。以下は、直接の LUN パスの XML サンプルです。

```
   <disk type='volume' device='disk'>
     <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'/>
   </disk>
```

device='disk' および長い **<source>** デバイス名が使用されていることに注意してください。この例では、*by-path* オプションを使用しています。コロンは区切り文字として認識される可能性があるため、コロンの前にはバックスラッシュが必要です。

LUN をパススルーデバイスとして設定するには、以下の XML サンプルを参照してください。

ノードデバイスドライバーを使用して作成された vHBA の場合:

```
   <disk type='volume' device='lun'>
     <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'/>
   </disk>
```

device='lun' とここでも長い **<source>** デバイス名が使用されていることに注意してください。また、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。

ストレージプールによって作成される vHBA の場合:

```
   <disk type='volume' device='disk'>
     <driver name='qemu' type='raw'/>
     <source pool='poolvhba0' volume='unit:0:2:0'/>
     <target dev='hda' bus='ide'/>
   </disk>
```

LUN のパスをストレージプールによって作成される vHBA のディスクソースとして使用することはできますが、libvirt ストレージプールとストレージボリュームを代わりに使用することをお勧めします。

16.7.4.4. vHBA の破棄

ストレージプールによって作成される vHBA は、以下のように **virsh pool-destroy** コマンドで破棄することができます。

```
# virsh pool-destroy poolvhba0
```

ストレージプールが永続的な場合、vHBA もストレージプールを破棄する際に libvirt によって削除されることに注意してください。

ノードデバイスドライバーを使用して作成される vHBA は **`virsh nodedev-destroy`** コマンドなどによって破棄できます (先に示されるように *scsi_host6* がすでに作成されていることを前提とします)。

```
# virsh nodedev-destroy scsi_host6
```

vHBA を破棄すると、再起動時と同様に vHBA の削除も実行されます。ノードデバイスドライバーが永続的な接続をサポートしないためです。

第17章 ストレージボリューム

17.1. 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。

17.1.1. 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使用される場合、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。

17.2. 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
```

17.3. 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。残りのコマンドの部分は、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
```

17.4. ゲストにストレージデバイスを追加する

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。「ゲストにファイルベースのストレージを追加する」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)。「ゲストにハードドライブと他のブロックデバイスを追加する」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。

17.4.1. ゲストにファイルベースのストレージを追加する

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17.1 ファイルベースのストレージを追加する

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ーをソースとする 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なわれます。

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を設定する

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選択します。

```
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ーを作成して、ゲスト上にディスクをマウントします。この例では、ディレクトリーは *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
```

17.4.2. ゲストにハードドライブと他のブロックデバイスを追加する

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17.2 ゲストに物理ブロックデバイスを追加する

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. 次のいずれかを行ないます。

   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:** 用のデバイスベースの追加ストレージコンテナー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、デバイスがハードディスクの場合には、 ***--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** など (ゲスト仮想マシン OS の選択によって異なる)、または Windows の場合は **D: ドライブ** という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 17.1「ファイルベースのストレージを追加する」 および ステップ 6 を参照してください。

警告

ホスト物理マシンでは、ファイルシステムを特定するために、**fstab** ファイルや、**initrd** ファイルまたはカーネルコマンドラインなどでファイルシステムのラベルを使用しないようにしてください。ゲスト仮想マシンなど特権を持たないユーザーがパーティションや LVM ボリューム全体への書き込みアクセスを持つ場合、ファイルシステムのラベルを使用するとセキュリティー上のリスクが発生します。ゲスト仮想マシンが、ホスト物理マシンに属するファイルシステムのラベルを独自のブロックデバイスストレージに書き込む可能性があるためです。これにより、ホスト物理マシンの再起動時に、ホスト物理マシンがこのゲスト仮想マシンのディスクをシステムディスクとして誤って使用してしまう可能性があり、ホスト物理マシンシステムが危険にさらされる可能性があります。

fstab ファイル、**initrd** ファイルまたはカーネルコマンドラインなどで使用する場合、デバイスの識別にはその UUID を使用した方がよいでしょう。それでも、特定のファイルシステムでは UUID の使用が完全に安全であるとは言えませんが、UUID を使用した場合には同様のセキュリティー侵害の可能性は確実に低くなります。

重要

ゲスト仮想マシンには、ディスク全域またはブロックデバイス全域 (例: **/dev/sdb**) に書き込みアクセスを付与することはできません。ブロックデバイス全域にアクセスを持つゲスト仮想マシンはボリュームレベルを修正できる場合があり、これがホスト物理マシンシステムの攻撃に使用される可能性があります。パーティション (例: **/dev/sdb1**) または LVM ボリュームを使用して、この問題を回避することができます。LVM ボリュームの詳細は、https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Logical_Volume_Manager_Administration/lv_overview.html を参照してください。**/dev/sdb1** または **/dev/sdb** などの raw ディスクなどのパーティションへの raw アクセスを使用している場合、**global_filter** 設定を使用して安全なディスクのみをスキャンできるように LVM を設定する必要があります。**global_filter** コマンドを使用した LVM 設定スクリプトのサンプルについては、https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Logical_Volume_Manager_Administration/lvmconf_file.html を参照してください。

17.4.3. 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17.3 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、それを、以下のステップを実行して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17.2「ゲストに物理ブロックデバイスを追加する」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、「ゲストにファイルベースのストレージを追加する」および「ゲストにハードドライブと他のブロックデバイスを追加する」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 SCSI コントローラーは 7 つのディスクをサポートしますが、ディスクの合計メモリーはホスト物理マシンのメモリー合計を超えることはできません。

   ```
   # 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 ストレージ管理ガイド』 のステップに従ってください。

17.5. ボリュームの消去と削除

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。

第18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。

18.1. 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

整合性チェックに対応するのは、*qcow2*、*qcow2 version3*、および *vdi* 形式のみになります。

18.2. イメージへの変更のコミット

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 qcow2] [-t cache] imgname
```

18.3. 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 のみであるのにもかかわらず、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。

18.4. 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、それ自体とベースファイル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キング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。

18.5. 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
```

18.6. イメージのバッキングファイルの再設定

qemu-img rebase はイメージのバッキング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
```

バッキングファイルが *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)、バッキングファイルの形式は *backing_format* に変更されます。

注記

バッキングファイルの変更 (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。新しいバッキングファイルは古いバッキング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* の古いバッキングファイル間で異なるクラスターは、バッキング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* のバッキング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。新しいバッキング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。

このモードはバッキング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、古いバッキング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、バッキング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。

18.7. 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と比較してサイズ変更を行うこともできます。サイズを大きくする場合はバイト数の前に **+** を付けます。小さくする場合はバイト数の前に**-** を付けます。単位のサフィックスを付けると、キロバイト (K)、メガバイト (M)、ギガバイト (G)、テラバイト (T) というようにそれぞれの単位でイメージサイズを変更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。

18.8. 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。

18.9. 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 か ext3、Windows の場合は NTFS)、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 (スナップショットなどは不可)。

- **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 (Windows 上の NTFS 以外のファイルシステム)。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、「ターゲット要素の設定」 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 Lunux 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。この形式は Windows では機能しません。
- **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。
- **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。
- **vmdk** - VMware 3 および 4 互換のイメージ形式です。
- **vpc** - Windows 仮想 PC (Windows Virtual PC) のディスクイメージ形式です。**vhd**、または Microsoft の仮想ハードディスクのイメージ形式とも呼ばれます。
- **vvfat** - Virtual VFAT のディスクイメージ形式です。

第19章 KVM のライブマイグレーション

本章では、あるホスト物理マシンで実行中のゲスト仮想マシンを別のホスト物理マシンに移行する方法について説明します。どちらのインスタンスでも、ホスト物理マシンは KVM ハイパーバイザーを実行しています。

移行とは、ゲスト仮想マシンをあるホスト物理マシンから別のホスト物理マシンに移行するプロセスです。移行が可能であるのは、ゲスト仮想マシンは直接ハードウェア上で実行されているではなく、仮想化環境内で実行されているためです。次の点を考慮すると移行が有用であると言えます。

- 負荷分散: ホスト物理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、別のホスト物理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して負荷分散を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安全に別のホスト物理マシンに移動することができます。つまり、テ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: ゲスト仮想マシンを別のホスト物理マシンに再配分できるので、電源をオフにして節電することで使用量の低い時間帯にコストを節約することができます。
- 地理的な移行: ゲスト仮想マシンは、待ち時間を短縮するため、または重大な状況が発生した場合に別の場所に移動することができます。

ゲスト仮想マシンのメモリー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なわれます。移行するゲスト仮想マシンのイメージは、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をお勧めします。

移行はライブでもライブでなくても行なうことができます。

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物理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、ゲスト仮想マシン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仮想マシン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。

注記

virtio デバイスが搭載されているゲスト仮想マシンを移行する場合は、重要 に説明されている警告に従ってください。

19.1. ライブ移行の要件

ゲスト仮想マシンの移行には以下が必要になります。

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
  - FCoE
  - 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* サービスが有効にされていることを確認します。

```
# systemctl enable libvirtd
```

- *libvirtd* サービスが実行中であることを確認します。

```
# systemctl restart libvirtd
```

  さらに、効果的に移行する機能は **/etc/libvirt/libvirtd.conf** 設定ファイルのパラメーター設定に依存することに留意してください。

- 表19.1「ライブマイグレーションの互換性」の表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。
- 両方のシステムで適切な TCP/IP ポートを開いておく必要があります。ファイアウォールが使用される場合には、ポートの詳細について、『Red Hat Enterprise Linux Virtualization Security Guide』 を参照してください。
- 共有ストレージメディアをエクスポートするシステムが別途必要になります。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します。マウントするディレクトリー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、推奨されていません。virt-manager を使って移行を行なう予定の場合、パス名は同一である必要があります。ただし、virsh を使って移行を行なう予定の場合は、移行を行なう際に --xml オプションや pre-hooks 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ーを使用することができます (ライブマイグレーションの制限 を参照してください)。**prehooks** についてさらに詳しくは、libvirt.org を参照してください。また、XML オプションについては、29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、移行元ホスト物理マシンと移行先ホスト物理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。

注記

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。
- ゲスト仮想マシンの電源がオフの場合、Red Hat Enterprise Linux 7 で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。ストレージのライブマイグレーション機能は Red Hat Enterprise 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。

手順19.1 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
```

19.2. 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

以下の表 表19.1「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19.1 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ため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。

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 の例を利用します。

19.3. 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よいでしょう。設定の詳細は、「iSCSI ベースのストレージプール」 を参照してください。

また、このセクションの内容は 『Red Hat Linux ストレージ管理ガイド』 に記載されている詳細な説明を置き換えるものではありません。NFS の設定方法、 IP テーブルの開き方、ファイアウォールの設定方法などの詳細については、 必ず 『Red Hat Linux 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)
   ```

   環境に応じてホスト名のパラメーターを変更してください。

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 restart nfs-server
   ```

3. **移動先に共有ストレージをマウントします。**

   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。ディレクトリーは同一でない場合、virt-manager を使用する移行は失敗します。

19.4. 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。一方のサイドが他方のサイド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。

本例では、 共有ストレージが正しく設定され前提条件はすべて満たしていると仮定しています (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 は、TLS/SSL、UNIX ソケット、SSH、および暗号化されていない TCP など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は *23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 **`virsh migrate`** では行えません。稼働していないゲスト仮想マシンを移行するには、次のスクリプトを使用してください。

```
 virsh -c qemu+ssh://<target-system-FQDN> migrate --offline --
persistent
```

19.4.1. 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、各移行インスタンスは移行元と移行先でそれぞれ 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19.1「libvirtd.conf の設定」の手順を参照してください。

1. 手順19.1「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 = 20


# 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 1 つのワーカー、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 つの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なう際は、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ーが起動すると、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な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。

19.4.2. 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** - ダイレクト移行に使用します。
- **--p2p** - ピアツーピア移行に使用します。
- **--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を定義しません。
- **--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な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** - 移行を実行中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*** - 接続先のハイパーバイザーが着信移行に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続行されます。
- **--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します。

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 **domjobingfo** 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、ライブマイグレーションの制限か、または virsh の man ページを参照してください。

19.5. 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. **virt-manager を開きます。**

   virt-manager を開きます。 メインメニューバーで**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** の順に選択し **virt-manager** を起動します。

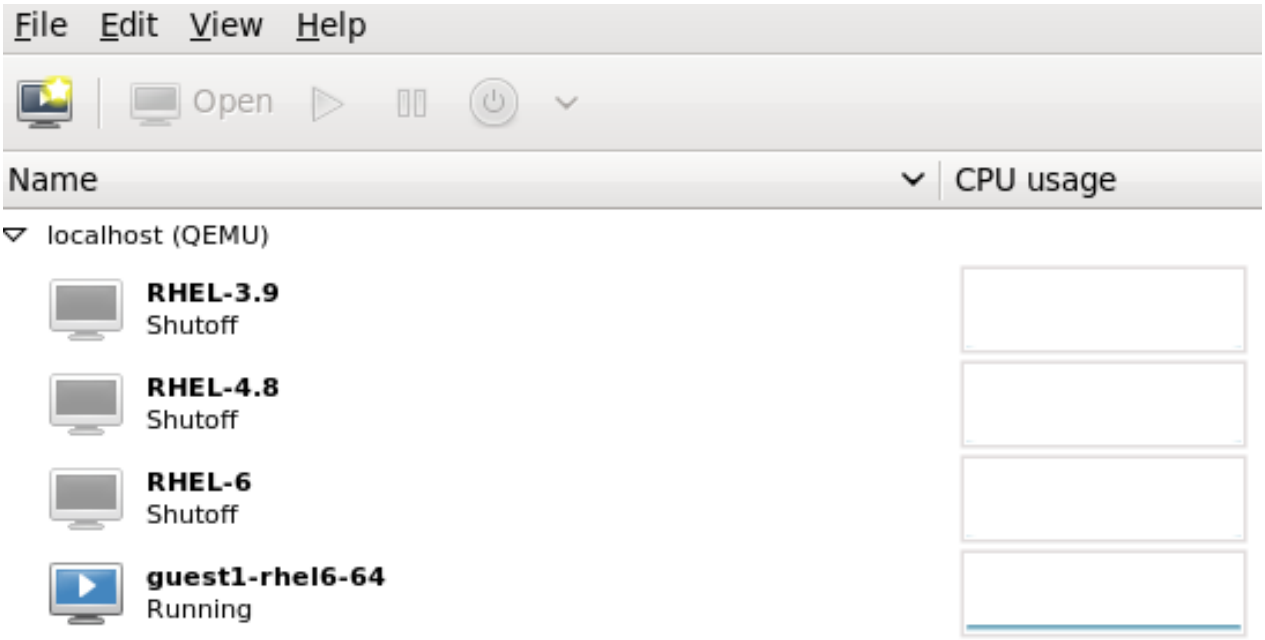


図19.1 Virt-Manager のメインメニュー

2. **移行先のホスト物理マシンに接続します。**

   ファイル メニューをクリックし、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。

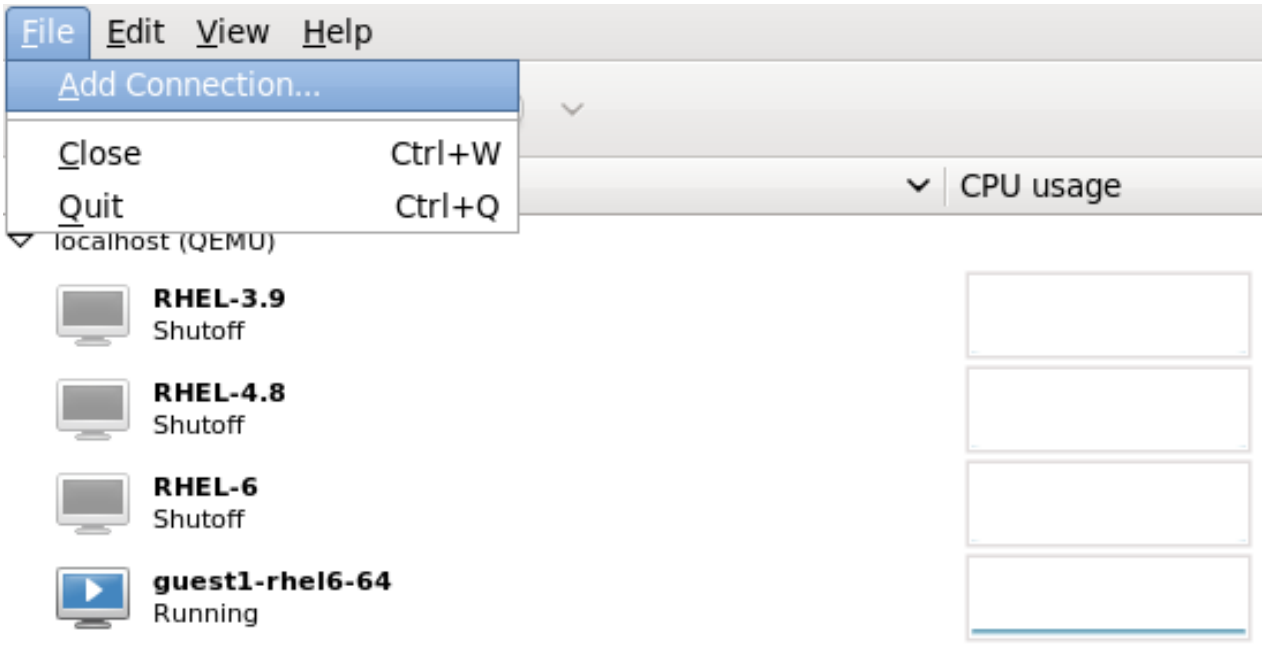


図19.2 「接続を追加」ウィンドウ

3. **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。

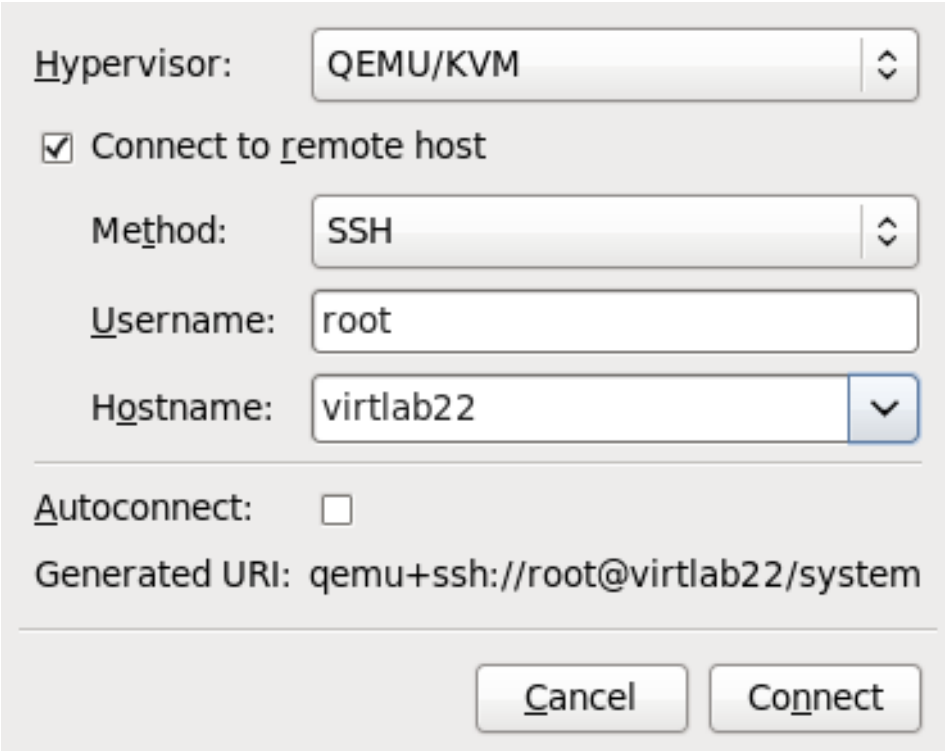


図19.3 移行先ホスト物理マシンへの接続を追加する

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。

接続 ボタンをクリックします。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。

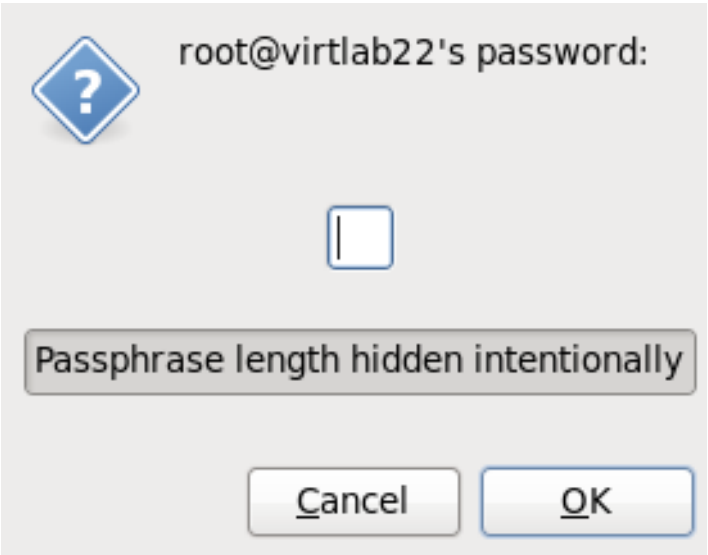


図19.4 パスワードの入力

4. **ゲスト仮想マシンを移行します。**

   移行元のホスト物理マシン内にあるゲストの一覧を開き (ホスト名の左側にある小さい三角をクリックし)、移行するゲスト (この例では **guest1-rhel6-64**) を右クリックしてから **マイグレーション** をクリックします。

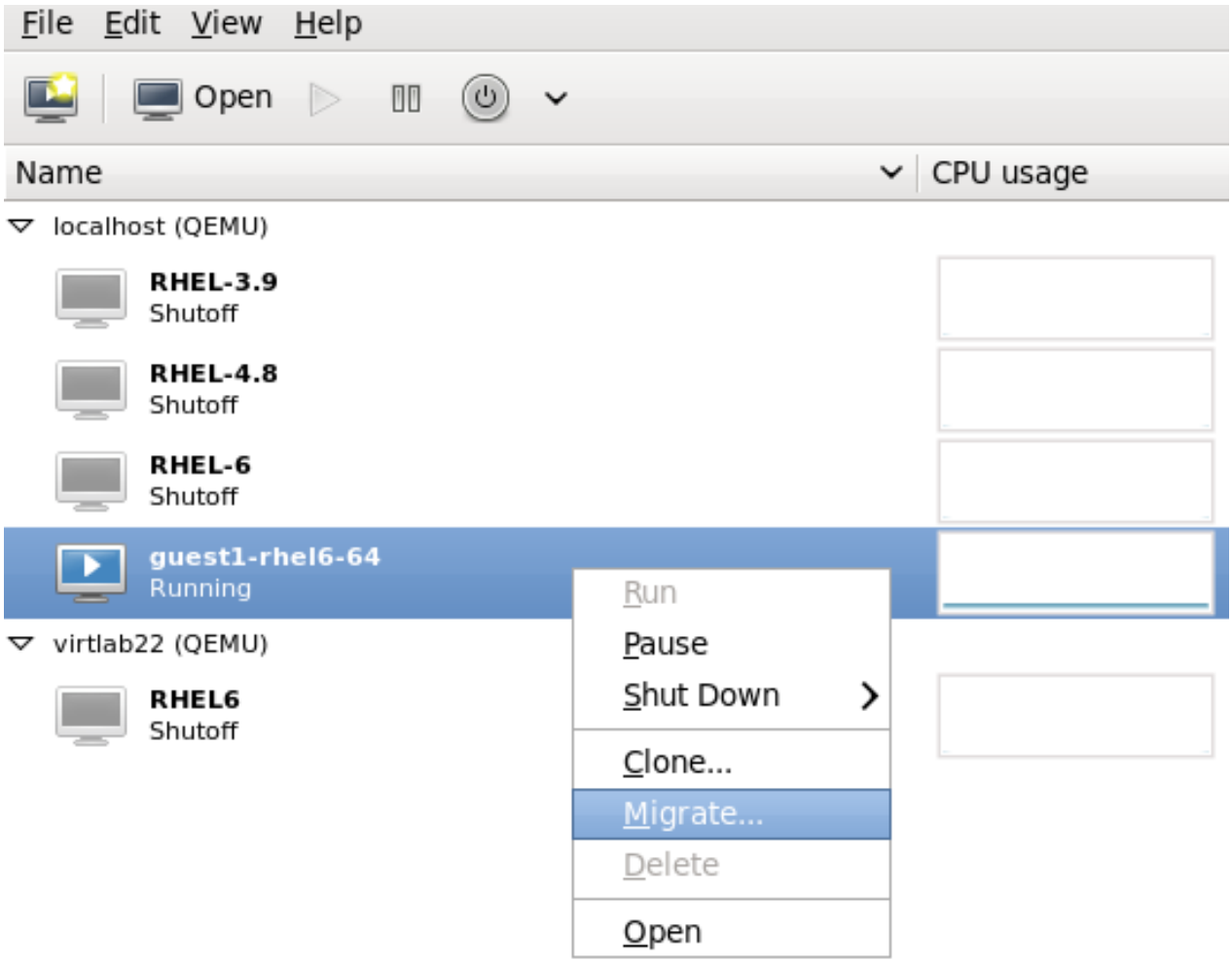


図19.5 移行するゲストの選択

新しいホスト 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。

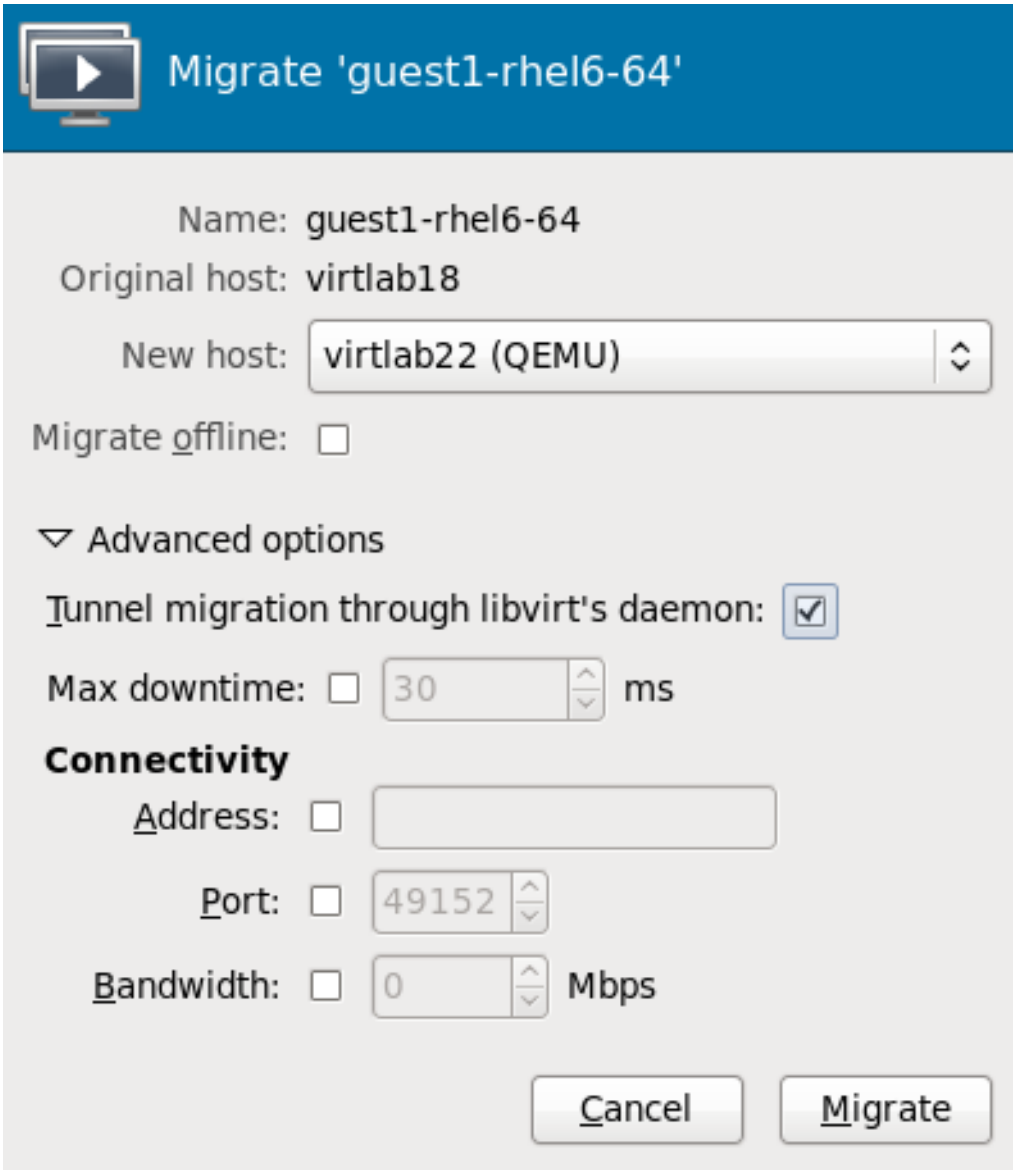


図19.6 移行先ホスト物理マシンの選択と移行プロセスの開始

進捗ウィンドウが表示されます。

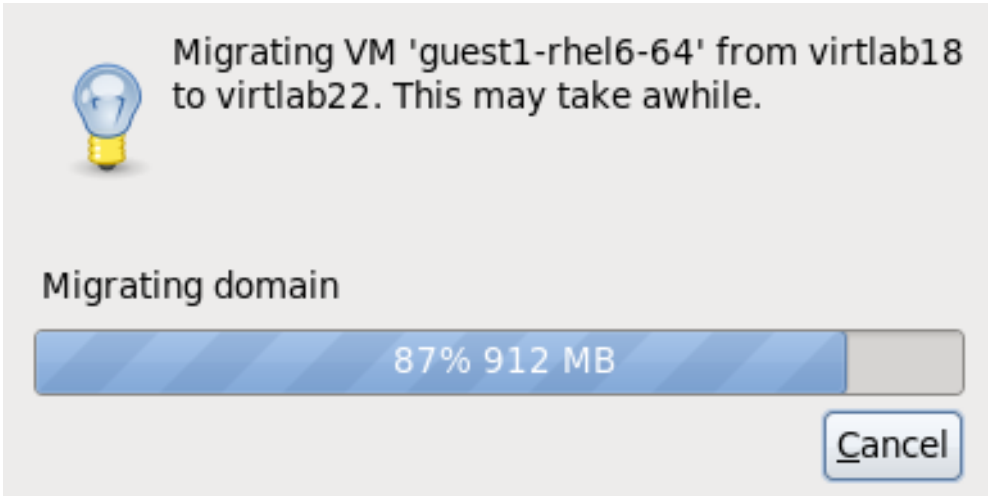


図19.7 進捗ウィンドウ

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。移行元のホスト物理マシンで実行されていたゲスト仮想マシンは「停止中 (Shutoff)」状態で表示されます。

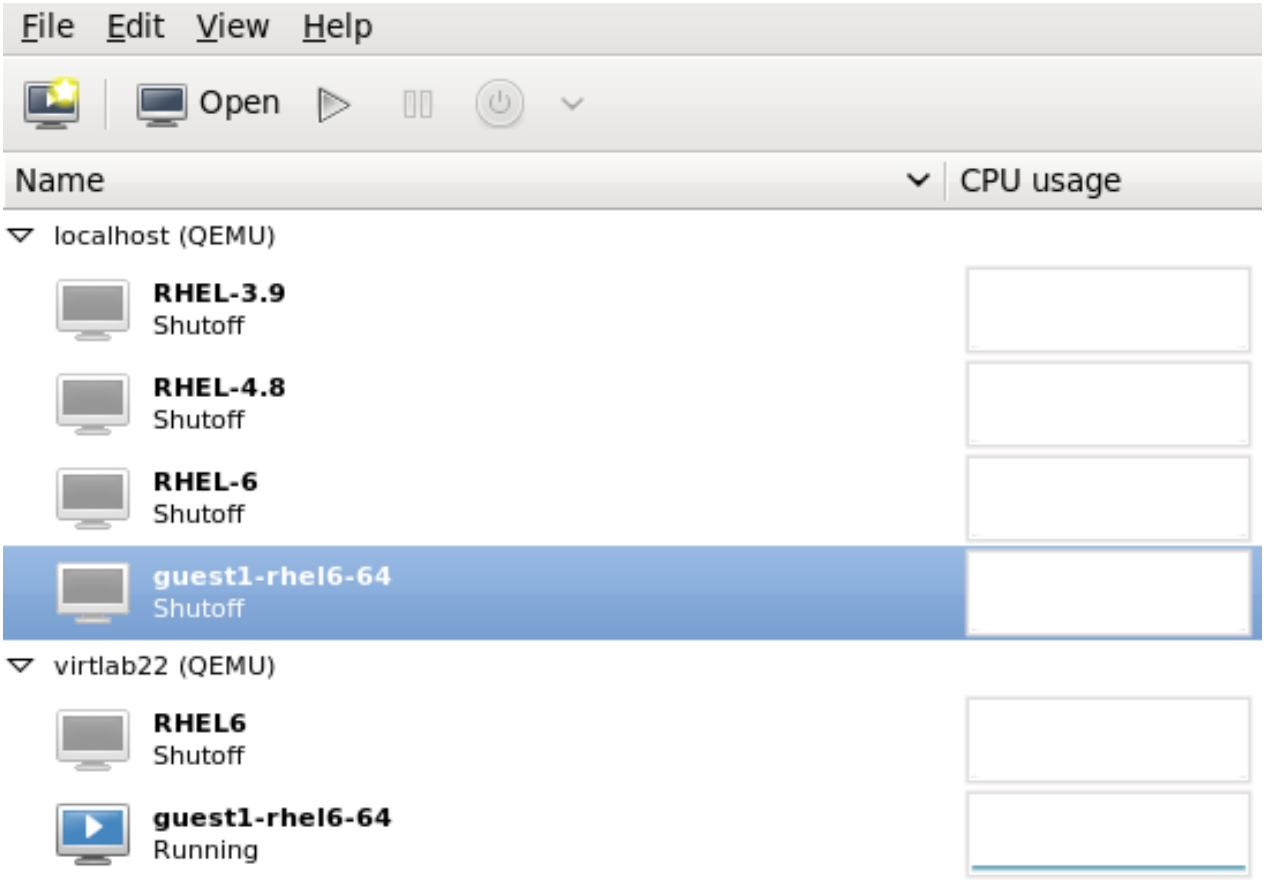


図19.8 移行先ホスト物理マシンで実行される移行済みゲスト仮想マシン

5. **オプション - ホスト物理マシンのストレージの詳細を表示します。**

   編集 メニューの **接続の詳細** をクリックすると、「接続の詳細」ウィンドウが表示されます。

   ストレージ タブをクリックします。移行先ホスト物理マシンの iSCSI ターゲットの詳細が表示されます。移行済みゲスト仮想マシンがストレージを使用しているマシンとして一覧表示されることに注意してください。

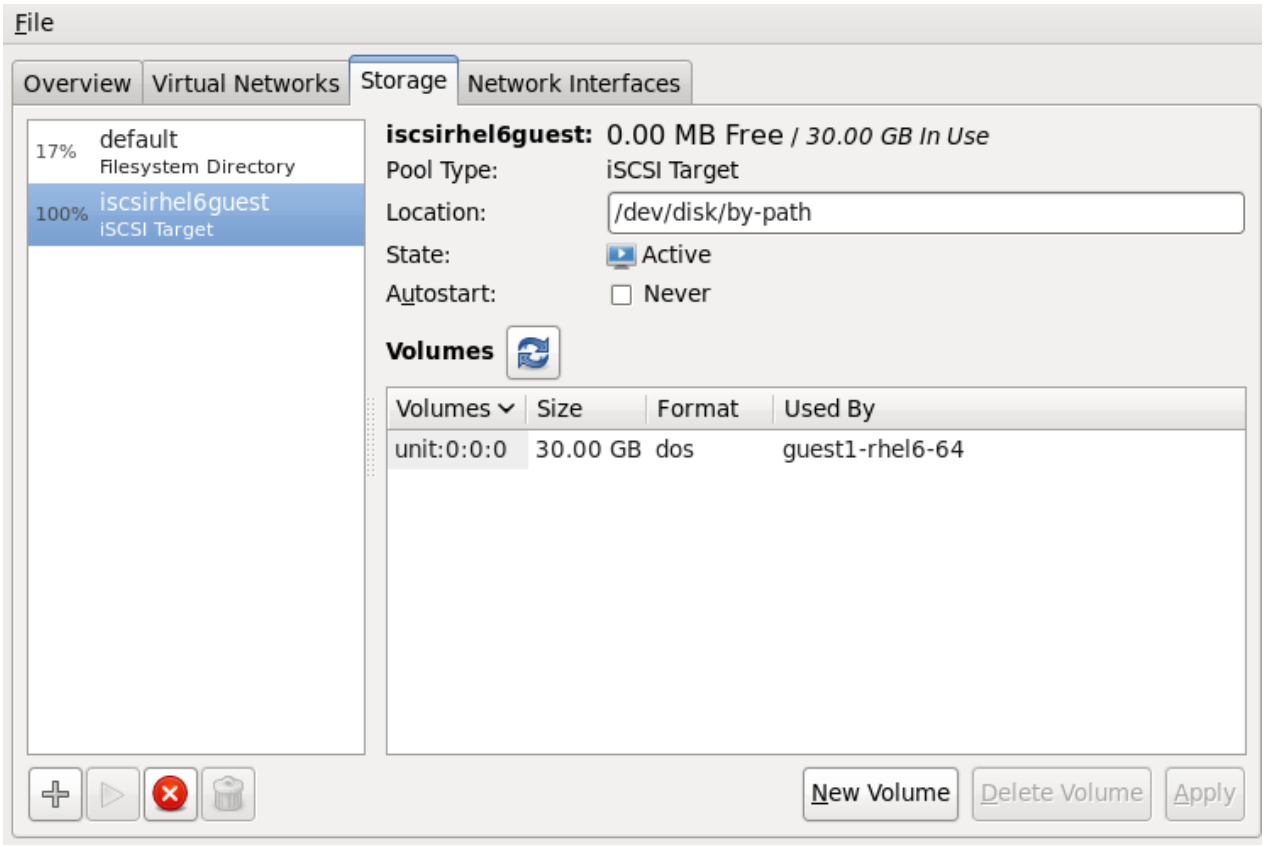


図19.9 ストレージの詳細

このホストは以下の XML 設定で定義されました。

```
<pool type='iscsi'>
    <name>iscsirhel6guest</name>
    <source>
        <host name='virtlab22.example.com.'/>
        <device path='iqn.2001-05.com.iscsivendor:0-8a0906-
fbab74a06-a700000017a4cc89-rhevh'/>
    </source>
    <target>
        <path>/dev/disk/by-path</path>
    </target>
</pool>
  ...
```

図19.10 移行先ホスト物理マシンの XML 設定

第20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仮想デバイスです。変更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ー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。Red Hat Enterprise Linux 7 は、最高 216 の virtio デバイスに対応します。
- *Virtio デバイス*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ー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 Enterprise 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にこれらのドライバーを自動的にインストールします。Red Hat Enterprise Linux 7 は最高 700 の scsi ディスクに対応します。
- *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、一部のグラフィックスデバイスを含め PCIe デバイス上でサポートされています。Nvidia K シリーズ Quadro、GRID、および Tesla グラフィックスカード GPU 機能が Red Hat Enterprise Linux 7 のデバイス割り当てでサポートされるようになりました。パラレル PCI デバイスは割り当てデバイスとしてサポートされますが、セキュリティーとシステム設定の競合により厳しい制限があります。

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。

特定デバイスの詳細および制限の詳細は、「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依然として許可することを選択するかもしれません。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
```

20.1. 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 BIOS で有効にされている必要があります。

手順20.1 Intel システムでの PCI デバイス割り当て準備

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*** 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。

   以下の修正例は、**grub** ファイルで Intel VT-d をアクティブにしたものです。

   ```
   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"
   ```

3. **設定ファイルを再生成します。**

   以下を実行して /boot/grub2/grub.cfg を再生成します。

   ```
   grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

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20.2 AMD システムでの PCI デバイス割り当て準備

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=on を **/etc/sysconfig/grub** の GRUB_CMDLINX_LINUX 行の終わり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 /boot/grub2/grub.cfg を再生成します。

   ```
   grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

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。

20.1.1. 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います。

手順20.3 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 nodedev-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 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
```

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>
```

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の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。**

   仮想マシン名を特定して **virsh edit** を実行し、**<source>** セクションにデバイスエントリーを追加して 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>
```

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。

20.1.2. 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20.4 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。

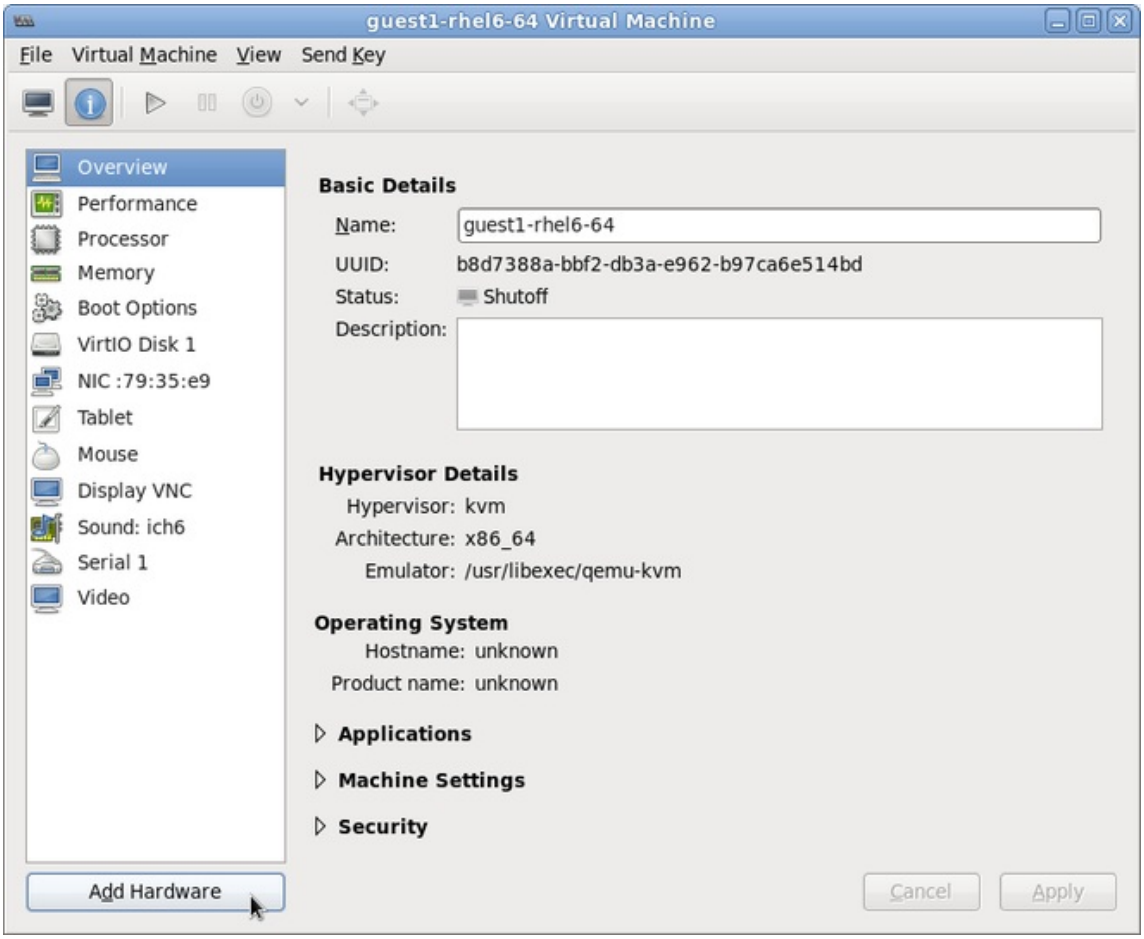


図20.1 仮想マシンのハードウェア情報ウィンドウ

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 Host Device** 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 82576 ネットワークデバイスが使用されます。**完了** をクリックしてセットアップを終了します。

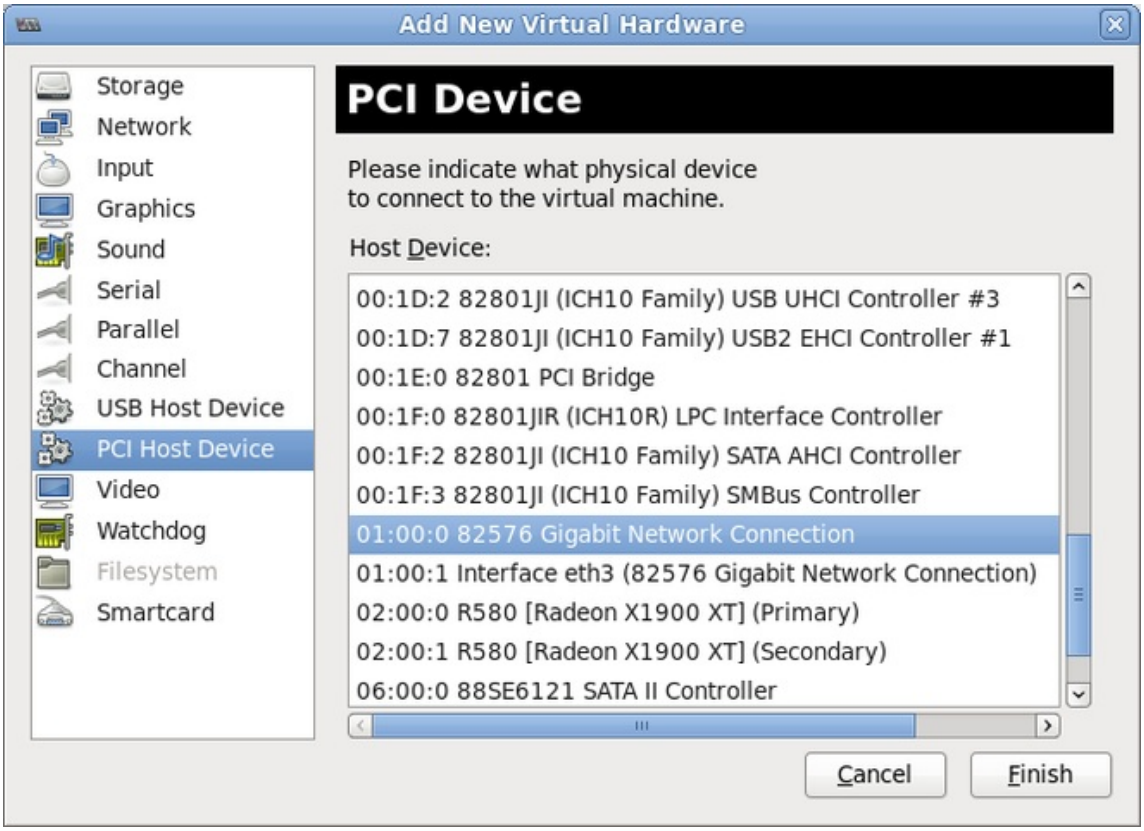


図20.2 新たな仮想ハードウェア追加ウィザード

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。

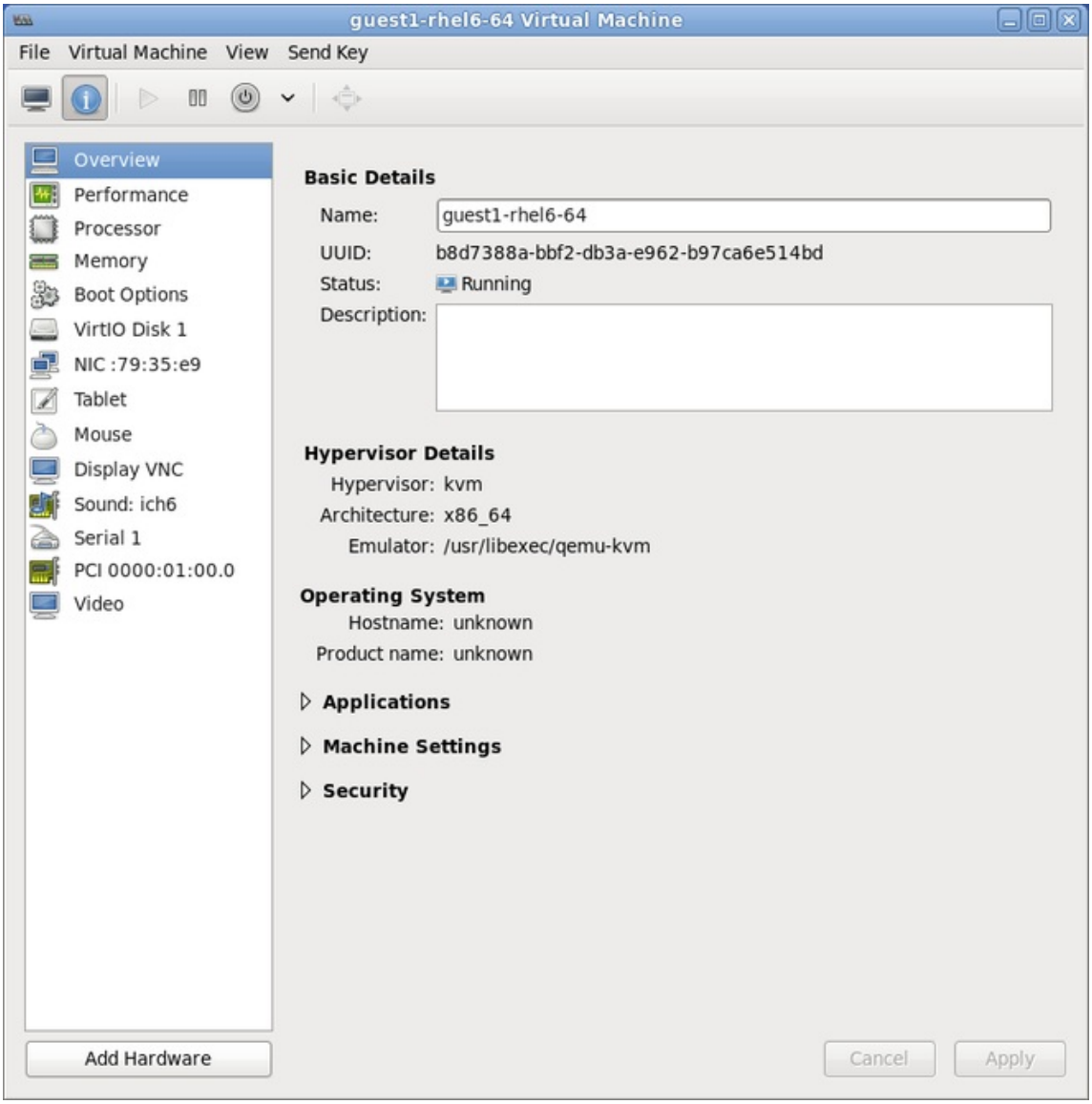


図20.3 仮想マシンのハードウェア情報ウィンドウ

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。

20.1.3. virt-install を使用した PCI デバイス割り当て

virt-install を使用して PCI デバイスを割り当てるには、***--host-device*** パラメーターを使います。

手順20.5 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、以下のコマンドを実行します。

```
# 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>
```

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 \
--nonsparse --graphics spice \
--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。

20.1.4. 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。このセクションでは、**virsh** または **virt-manager** を使ってデバイスをゲストから切り離す方法を説明します。これによって、ホストがデバイスを使えるようになります。

手順20.6 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。デバイスは自動的にホストに戻ります。

   デバイスが ***managed*** モードでない場合は、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20.7 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。**仮想ハードウェアの詳細表示** 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20.4 仮想ハードウェア詳細ボタン

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。

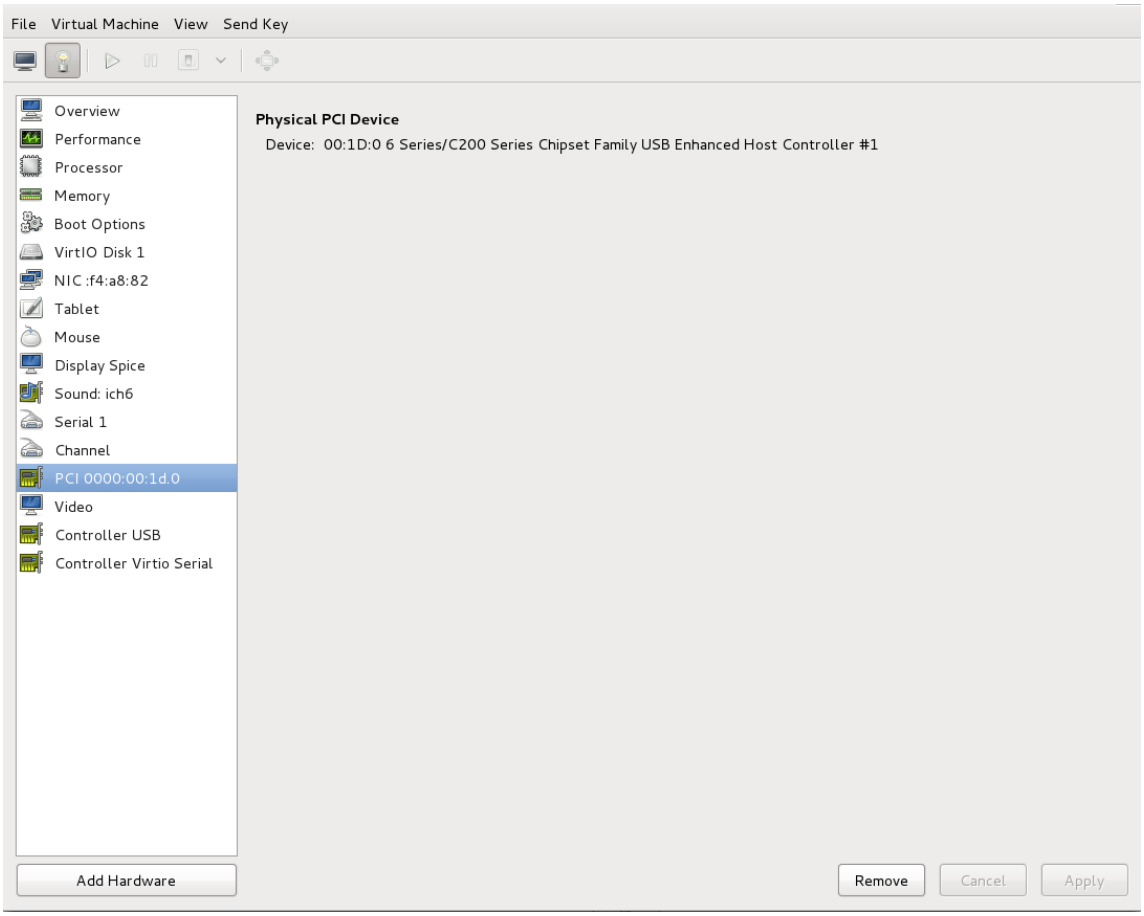


図20.5 接続解除する PCI デバイスの選択

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。

20.1.5. 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。

20.1.5.1. 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

20.1.6. 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、汎用デバイス*パススルー*を使用してゲストに直接割り当てられます。これは、まず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、デバイスをオプションで指定した **<virtualport>** 要素を使用して 802.1Qbh 対応スイッチに関連付けた後に行なわれます (type='direct' ネットワークデバイスに指定された virtualport の例を参照)。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、SR-IOV (シングルルート I/O 仮想化) 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。

従来のレガシーの KVM デバイス割り当てではなく、VFIO デバイス割り当てを使用するには (VFIO は、UEFI Secure Boot と互換性のあるデバイス割り当ての新しい方法です)、**<type='hostdev'>** インターフェースに、name 属性を「vfio」に設定したオプションの driver サブ要素を持たせることができます (または、現在 **<driver='kvm'>** がデフォルトになっているため、**<driver>** 要素を単純に省略することもできます)。

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」は標準の **<hostdev>** デバイスの機能に非常によく似ていることに注意してください。相違点は、この方法では パススルーデバイスの MAC アドレスと **<virtualport>** を指定できる点にあります。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 ネットワークの場合や、0.9.11 より前の libvirt のバージョンを使用している場合は、デバイスをゲストに割り当てるには **<interface type='hostdev'/>** の代わりに標準の **<hostdev>** を使用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20.6 PCI デバイス割り当ての XML サンプル

20.1.7. SR-IOV デバイスの場合の PCI 割り当て (パススルー) の設定

このセクションは、SR-IOV デバイスのみを対象とします。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ても、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、要素 **<hostdev>** 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。この問題を修復するには、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、以下で説明されている手順を参照してください: 手順20.8「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」

手順20.8 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

まず、**<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アイテムに使用することはできないことに留意してください。その理由は、**<mac>**、**<vlan>**、および **<virtualport>** 要素は **<hostdev>** の有効な子ではないからです。それらは **<interface>** では有効であるため、新規インターフェースタイプのサポートが追加されました (**<interface type='hostdev'>**)。この新規インターフェースデバイスタイプは**<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. **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 (「Red Hat Enterprise Linux 7 ホスト上の Red Hat Enterprise Linux 6 ゲストのシャットダウン」 を参照) を使用して、*guestVM* という名前のゲスト仮想マシンをシャットダウンします。

   ```
   # virsh shutdown guestVM
   ```

2. **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。

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、# **virsh save-image-edit** コマンドを実行します (詳細は、「ドメイン XML 設定ファイルの編集」 を参照してください)。ゲスト仮想マシンを以前の実行状態に戻したい場合は、**--running** を使用できます。この例の設定ファイルの名前は、ゲスト仮想マシンの名前が *guestVM* なので *guestVM.xml* になります。

   ```
   # 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* が開きます。

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20.7 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、802.11Qgh ハードウェアスイッチ (802.11Qbg (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。**

   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、「定義されたドメインの起動」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。

20.1.8. 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、これは各 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)。
- ゲスト仮想マシンを別のホスト物理マシンに移動する場合、そのホスト物理マシンには、PCI バス上の同じ場所にあるものと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のプールに戻ります。

手順20.9 デバイスプールの作成

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 (「Red Hat Enterprise Linux 7 ホスト上の Red Hat Enterprise Linux 6 ゲストのシャットダウン」 を参照) を使用して、*guestVM* という名前のゲスト仮想マシンをシャットダウンします。

   ```
   # 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) 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20.8 ネットワーク定義ドメインの サンプル 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、以下のコマンドを実行します。

   ```
   # 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、「定義されたドメインの起動」 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20.9 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20.10 XML ダンプファイル *passthrough* の内容

20.2. 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。

20.2.1. 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26.1「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、これをシンクライアント上で実行されているゲスト仮想マシンで使用することができます。*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。

20.2.2. USB デバイスのリダイレクトに制限を設ける

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20.1 Windows 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. Windows 7 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。

20.3. 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20.11 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の数を制御する 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。

コントローラー自体が 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、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20.12 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 コントローラーは自動的に追加され、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20.13 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20.14 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='0x0f'
```

```
function='0x0'/>
    </controller>
  </devices>
    ...
```

図20.15 USB3/XHCI デバイスに使用されるドメイン XML の例

20.4. 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 デバイスサンプルについては、図20.6「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20.1 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
type='pci'	PCI アドレスには以下の追加属性が含まれます。 - domain (2 バイト 16 進数整数。現在 qemu で使用されていません) - bus (0 から 0xff までの 16 進数値) - slot (0x0 から 0x1f までの 16 進数値) - function (0 から 7 までの値) - multifunction 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 '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'	ドライブアドレスには、以下の追加属性が含まれます。 - 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'	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。 - 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)

アドレスタイプ	説明
type='ccid'	スマートカードに使用される CCID アドレスには、以下の追加属性が含まれます。 - bus (2 桁のバス番号) - slot 属性 (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含まれます。 - bus (0 から 0xfff までの 16 進数) - port (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)
type='isa'	ISA アドレスには以下の追加属性が含まれます。 - iobase - irq

20.5. 乱数ジェネレーターデバイス

virtio-rng は、要求されるとゲストに新規エントロピーを提供できる仮想ハードウェアの乱数ジェネレーターデバイスです。ドライバーはデータをゲスト仮想マシンの OS にフィードし直します。

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が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、エントロピーのゲスト仮想マシンオペレーティングシステムへ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* を使用して実行できます。*virt-manager* の説明については、手順20.10「 Virtualzation Manager を使用した virtio-rng の実装」を参照し、コマンドラインの説明については、手順20.11「コマンドラインツールを使用した virtio-rng の実装」 を参照してください。

手順20.10 Virtualzation Manager 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。

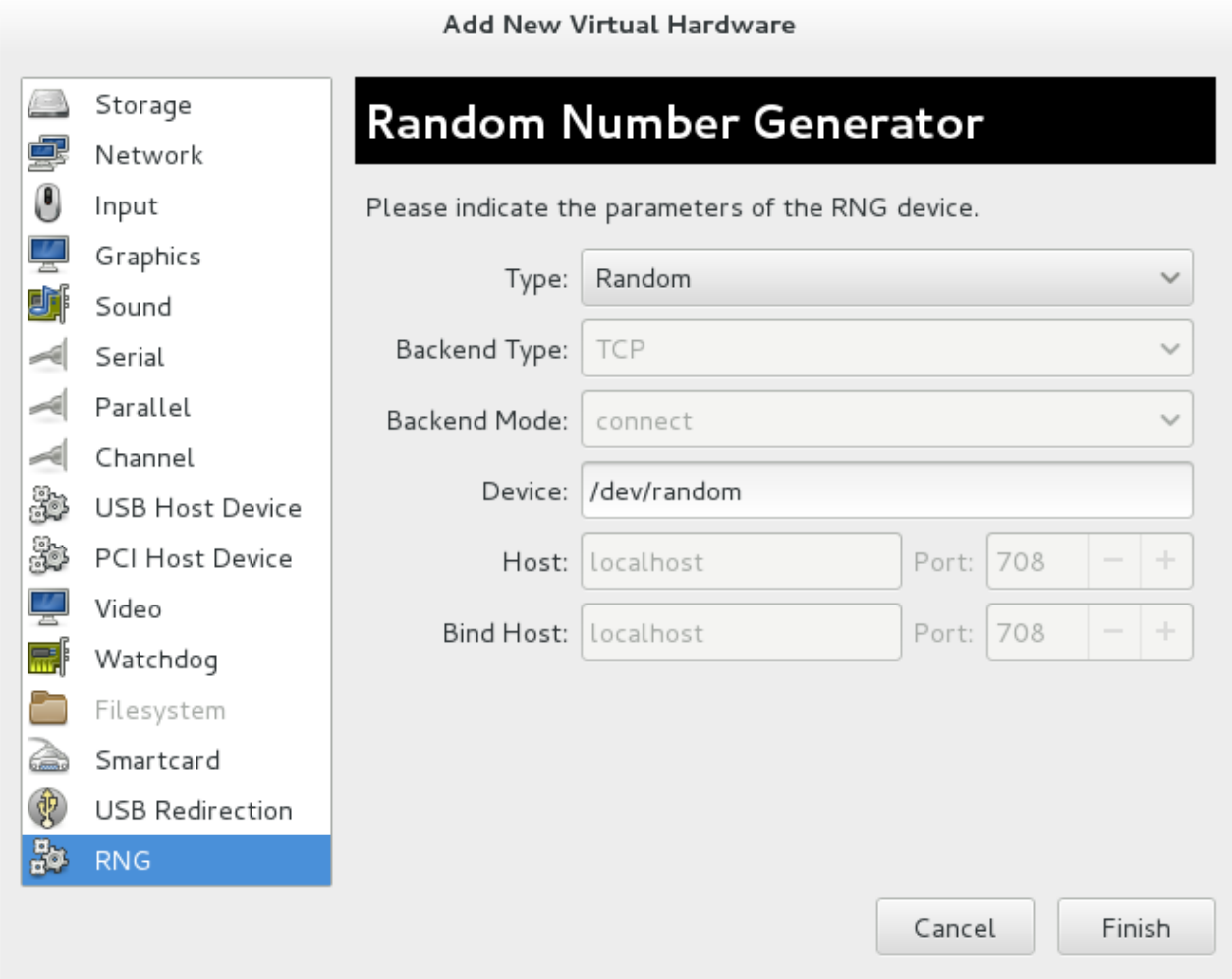


図20.16 乱数ジェネレーターデバイスウィンドウ

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20.11 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 physical machine='1.2.3.4'
service='1234'>
```

```
      </backend>
    </rng>
  </devices>
  ...
```

図20.17 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。**'virtio'**
- **<backend>** - **<backend>** 要素は、ドメイン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'** — **/dev/random** (デフォルト設定) またはソースと同様のデバイス、および EGD プロトコルのバックエンドを設定する **'egd'** が含まれます。
- **backend type='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 type='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。

20.6. GPU デバイスの割り当て

RHEL7 は、NVIDIA K シリーズの Quadro、GRID、および Tesla の PCI デバイス割り当てを VGA 以外のグラフィックスデバイスとしてサポートします。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、1 つの以上の GPU を VM 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ーがロードされると、NVIDIA GPU に引き継がれます。

この手順では、簡単に説明すると、lspci からデバイスを特定し、この割り当てをホスト物理マシンから解除してから、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*** 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=on*** 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_CMDLINX_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"
   ```

2. **ブートローダー設定を再生成します。**

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20.12 GPU デバイスを、ホスト物理マシンドライバーへのバインドから除外する

GPU 割り当ての場合、ホストドライバー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ー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ーがこの GPU デバイスを使用することを防ぎます。**

   ネイティブのホスト物理マシンドライバーが 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を再生成します。**

   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
```

手順20.13 PCI デバイスの変換およびホスト物理マシンからの切り離し

1. **現在の設定 XML を開きます。**

   virsh nodedev-dumpxml コマンドを実行して、現在の設定ファイルをダウンロードし、これを開きます。このファイルを 手順20.12「GPU デバイスを、ホスト物理マシンドライバーへのバインドから除外する」 で取得した PCI アドレスとして保存します。ただし、PCI バスアドレスは、「pci_」をファイル名に追加し、区切り文字をアンダースコアに変換することにより、libvirt と互換性のある形式に変換する必要があります。この場合、libvirt アドレスは pci_0000_02_00_0 になります。完全なコマンドを実行すると、設定がダウンロードされ、以下のように libvirt が認識できるファイル名に保存されます。

   ```
   # virsh nodedev-dumpxml pci_0000_02_00_0
   ```

2. **新規の設定ファイルを開き、これを編集します。**

   nodedev-dumpxml オプションを実行すると、ファイルが開かれます。デバイスについての以下の追加情報と類似する情報を指定してファイルを編集します。

```
  ...
<device>
  <name>pci_0000_02_00_0</name>
  <path>/sys/devices/pci0000:00/0000:00:03.0/0000:02:00.0</path>
  <parent>pci_0000_00_03_0</parent>
  <driver>
    <name>pci-stub</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>2</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x11fa'>GK106GL [Quadro K4000]</product>
    <vendor id='0x10de'>NVIDIA Corporation</vendor>
    <iommuGroup number='13'><!--LOOK HERE-->
      <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20.18 ゲスト GPU 割り当て用のドメイン XML

とりわけ重要な部分 (図20.18「ゲスト GPU 割り当て用のドメイン XML」) は強調表示されている iommuGroup 一覧になります (--LOOK HERE-- が表示されている部分を確認してください)。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ると見なされるデバイスセットを示します。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の iommuGroup 内のすべてのエンドポイントデバイスは、ゲスト仮想マシンに割り当てるため、ネイティブのホスト物理マシンドライバーからのバインドを解除する必要があります。図20.18「ゲスト GPU 割り当て用のドメイン XML」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。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 をゲスト仮想マシンに割り当てるには、最初に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** コマンドを使用してデバイスの PCI ID を検索し、これを 手順20.12「GPU デバイスを、ホスト物理マシンドライバーへのバインドから除外する」 に示されるように pci-stub.ids オプションに追加して実行されます。

手順20.14 GPU デバイスをゲスト仮想マシンに割り当てる

1. **ゲスト仮想マシンの GPU 設定内容についての XML ファイルを作成します。**

   ホスト物理マシンから切り離されると、XML 設定ファイルをゲスト仮想マシンに割り当てることにより、virsh を使用して GPU をゲスト仮想マシンに割り当てることができます。最初に実行できる点として、割り当てるスニペットを作成できます。テキストエディターを使用して、以下と同様のファイルを作成し、これを保存します。以下の例では、guestVM1 には作成された XML ファイルが含まれ、このファイルのタイトルは gpu.xml になります。

```
  ...
<hostdev mode='subsystem' type='pci' managed='yes'>
  <driver name='vfio'/>
  <source>
    <address domain='0x0000' bus='0x02' slot='0x00'
function='0x0'/>
  </source>
</hostdev>
...
```

   図20.19 GPU デバイスの割り当て

2. **ゲストの設定を変更します。**

   virsh attach-device [domain] [file] --persistent を実行して XML をゲスト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 が追加されます。割り当てられた GPU は、ゲスト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。直前のステップで提供された例を使用して、以下のコマンドを実行できます。

```
virsh attach-device guestVM1 gpu.xml --persistent
```

3. **ゲスト仮想マシンを再起動します。**

   変更を即時に実行するには、**virsh reboot [domain]** を使用してゲストを再起動します。

```
# virsh reboot guestVM1
```

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 ゲスト仮想マシンの場合、nouveau ドライバーが、ゲストのインストール時にカーネルコマンドライン内でオプションの modprobe.blacklist=nouveau を使用することでブラックリスト化されないことに注意してください。

Linux 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。たとえば、ゲスト内で GPU の PCI バスアドレスを判別するには (これはホストアドレスとは異なります)、以下のコマンドを実行します。

```
# 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 は以下のエントリーを加えるように変更されます。BudID を読み取る行に注意してください。

```
Section "Device"
    Identifier     "Device0"
    Driver         "nvidia"
    VendorName     "NVIDIA Corporation"
    BusID          "PCI:0:9:0"
EndSection
```

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ーは、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 が物理ディスプレー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21章 SR-IOV

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。

注記

Xeon E3-1200 シリーズチップセット を使用する仮想マシンは SR-IOV に対応していません。詳細情報は、Intel の web サイトまたはこの 記事 を参照してください。

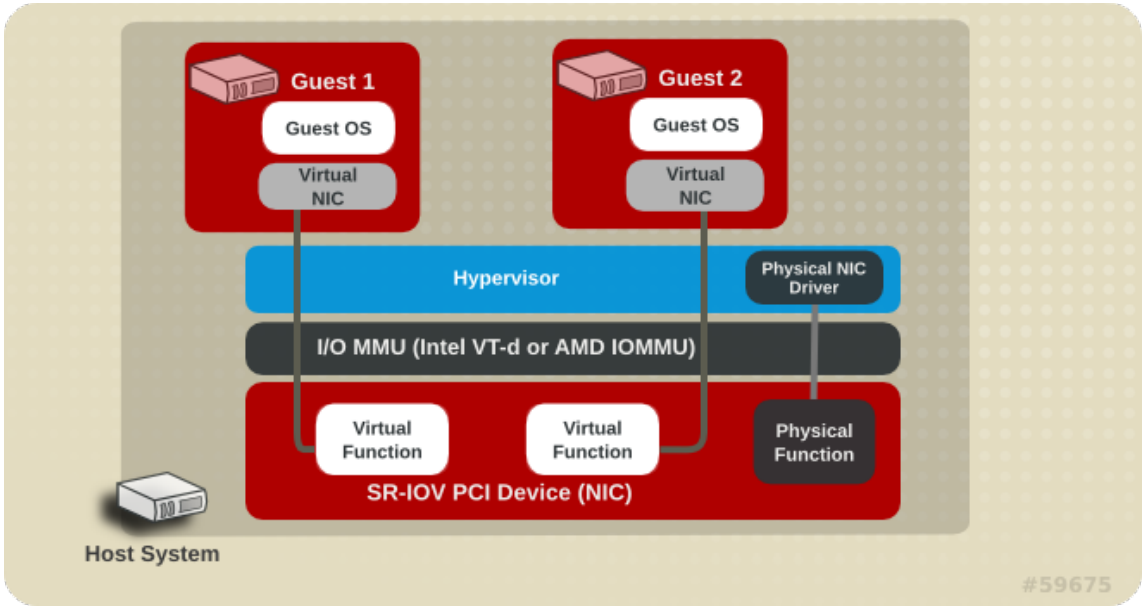


図21.1 SR-IOVのしくみ

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マッピングできます。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マッピングされます。

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 つのゲストにのみマッピングできます。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。

21.1. 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。

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。

21.2. 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ー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-0.9.10 以降には、***`<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)。その後に、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21.1 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20.1「Intel システムでの PCI デバイス割り当て準備」 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20.2「AMD システムでの PCI デバイス割り当て準備」 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. **SR-IOV カーネルモジュールを開始します。**

   デバイスが対応していれば、ドライバーカーネルモジュールがカーネルによって自動的にロードされます。オプションのパラメーターは、**`modprobe`** コマンドを使ってモジュールに渡すことができます。Intel 82576 ネットワークインタフェースカードは、**`igb`** ドライバーカーネルモジュールを使用します。

   ```
   # modprobe igb [<option>=<VAL1>,<VAL2>,]
   # lsmod |grep igb
   igb     87592  0
   dca     6708    1 igb
   ```

4. **仮想機能をアクティブにします。**

   `igb` モジュールの ***`max_vfs`*** パラメーターは、最大数の仮想機能を割り当てます。***`max_vfs`*** パラメーターにより、ドライバーは最大パラメーターの値までの仮想機能を発生させます。このカードでは、有効な範囲は ***`0`*** から ***`7`*** です。

   モジュールを削除して、変数を変更します。

   ```
   # modprobe -r igb
   ```

   `max_vfs` を ***`7`*** または使用しているデバイスがサポートする最大数の仮想機能数に設定し、モジュールを再起動します。

   ```
   # modprobe igb max_vfs=7
   ```

5. **仮想機能を永続化します。**

   /etc/modprobe.d にある任意のファイルに **options igb max_vfs=7** の行を追加し、仮想機能を永続化します。たとえば、以下のようになります。

```
# echo "options igb max_vfs=7" >>/etc/modprobe.d/igb.conf
```

6. **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。

7. **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、すべての句読点とセミコロン (**;**) 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シリアル番号が表示されます。

8. **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、両方のデバイスの詳細な出力を表示します。

```
# virsh nodedev-dumpxml pci_0000_0b_00_0
<device>
   <name>pci_0000_0b_00_0</name>
   <parent>pci_0000_00_01_0</parent>
   <driver>
      <name>igb</name>
   </driver>
   <capability type='pci'>
      <domain>0</domain>
      <bus>11</bus>
      <slot>0</slot>
      <function>0</function>
      <product id='0x10c9'>Intel Corporation</product>
      <vendor id='0x8086'>82576 Gigabit Network Connection</vendor>
   </capability>
</device>
```

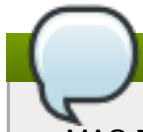
```
# virsh nodedev-dumpxml pci_0000_0b_10_0
<device>
   <name>pci_0000_0b_10_0</name>
   <parent>pci_0000_00_01_0</parent>
   <driver>
      <name>igbvf</name>
   </driver>
   <capability type='pci'>
```

```
      <domain>0</domain>
      <bus>11</bus>
      <slot>16</slot>
      <function>0</function>
      <product id='0x10ca'>Intel Corporation</product>
      <vendor id='0x8086'>82576 Virtual Function</vendor>
   </capability>
</device>
```

この例では、仮想機能 **pci_0000_0b_10_0** を ステップ 9 で仮想マシンに追加します。仮想機能の **bus**、**slot**、**function** パラメーターは、デバイスを追加するために必要になります。

/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='0' bus='11' slot='16'
function='0'/>
     </source>
   </interface>
```



注記

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、802.11Qbh ハードウェアスウィ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。

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>
```

9.

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。

21.3. 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、付録A *トラブルシューティング* および「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ホスト自体に割り当てられているデバイスが引き起こすものです。

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'
```

デバイス割り当ては、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中は、ゲストの移行および保存が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、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、今後の改善に向けて作業が進行中です。

第22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、変更などを行な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。

22.1. 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。

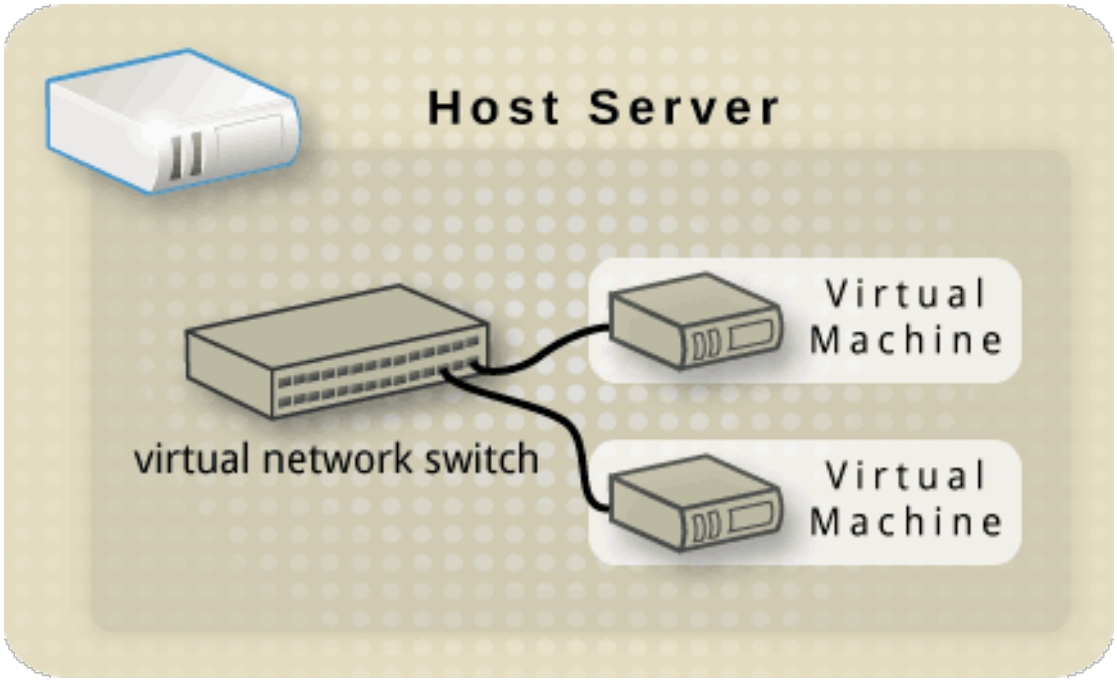


図22.1 ゲストを 2 つ持つ仮想ネットワークの例

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。libvirtd 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。

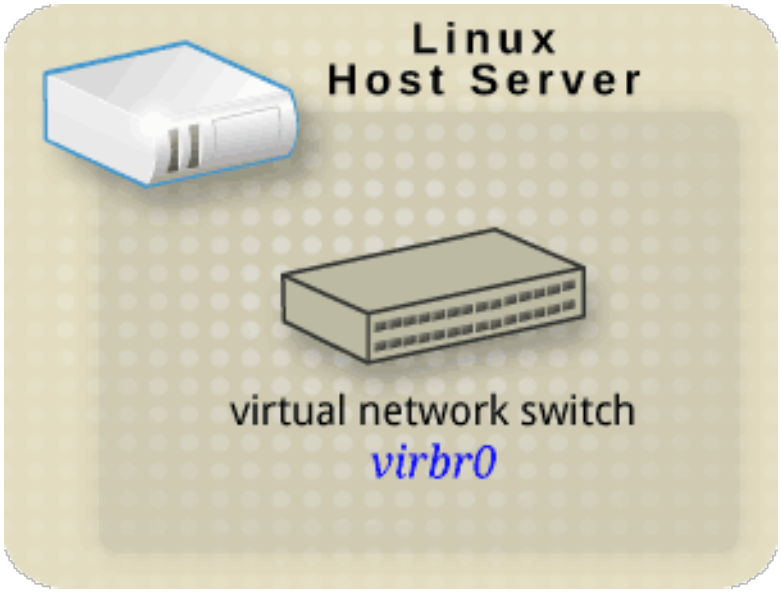


図22.2 仮想ネットワークスイッチへのインターフェースを持つ Linux ホスト物理マシン

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**ifconfig** コマンドや**ip** コマンドで表示させることができます。

```
$ ifconfig virbr0
 virbr0    Link encap:Ethernet  HWaddr 1B:C4:94:CF:FD:17
           inet addr:192.168.122.1  Bcast:192.168.122.255
Mask:255.255.255.0
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
           TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:0
           RX bytes:0 (0.0 b)  TX bytes:3097 (3.0 KiB)
```

```
 $ 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
```

22.2. 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。SNAT (Source-NAT) や DNAT (Destination-NAT) 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。

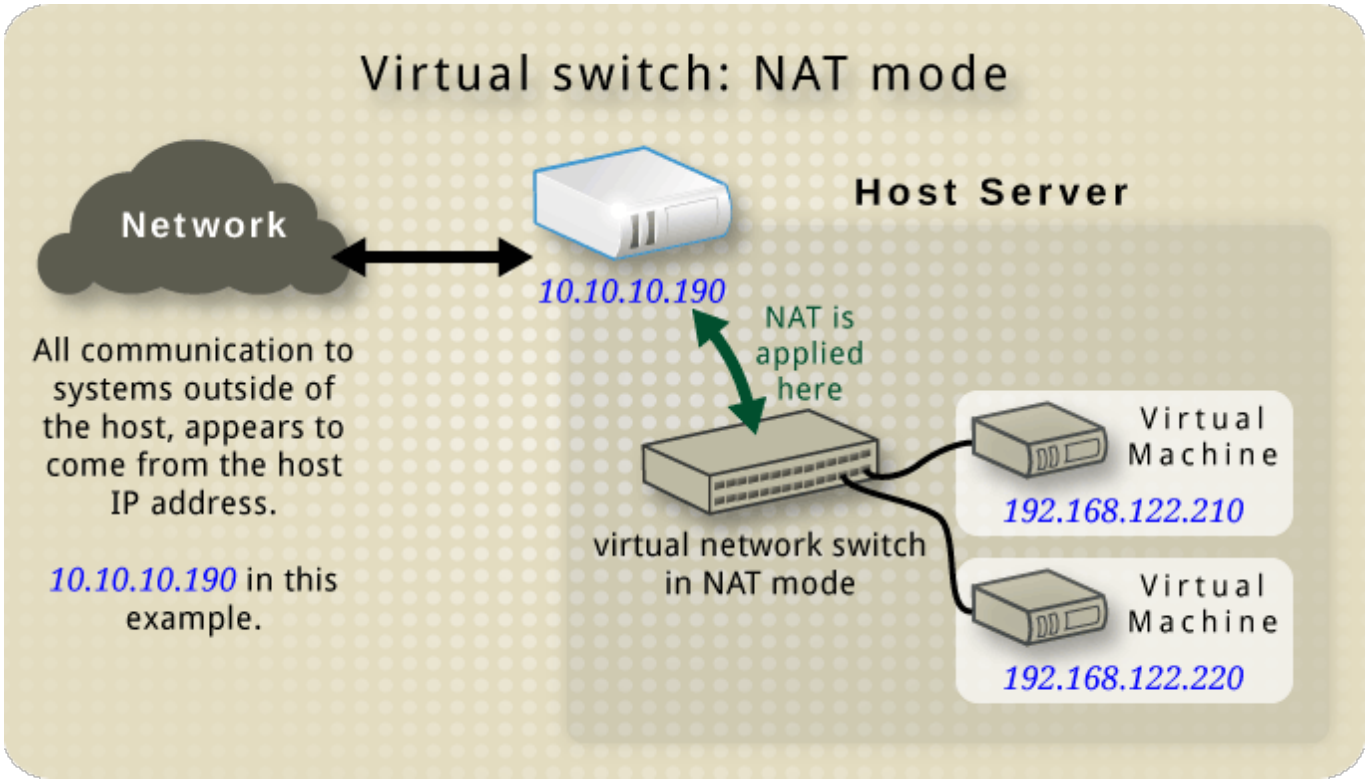


図22.3 ゲストを 2 つ持つ仮想ネットワークスイッチで NAT を使用している例

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]
```

22.3. ネットワークプロトコル

次のセクションでは、各ネットワークプロトコルについて、また各プロトコルの libvirt での使い方について説明します。

22.3.1. 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。

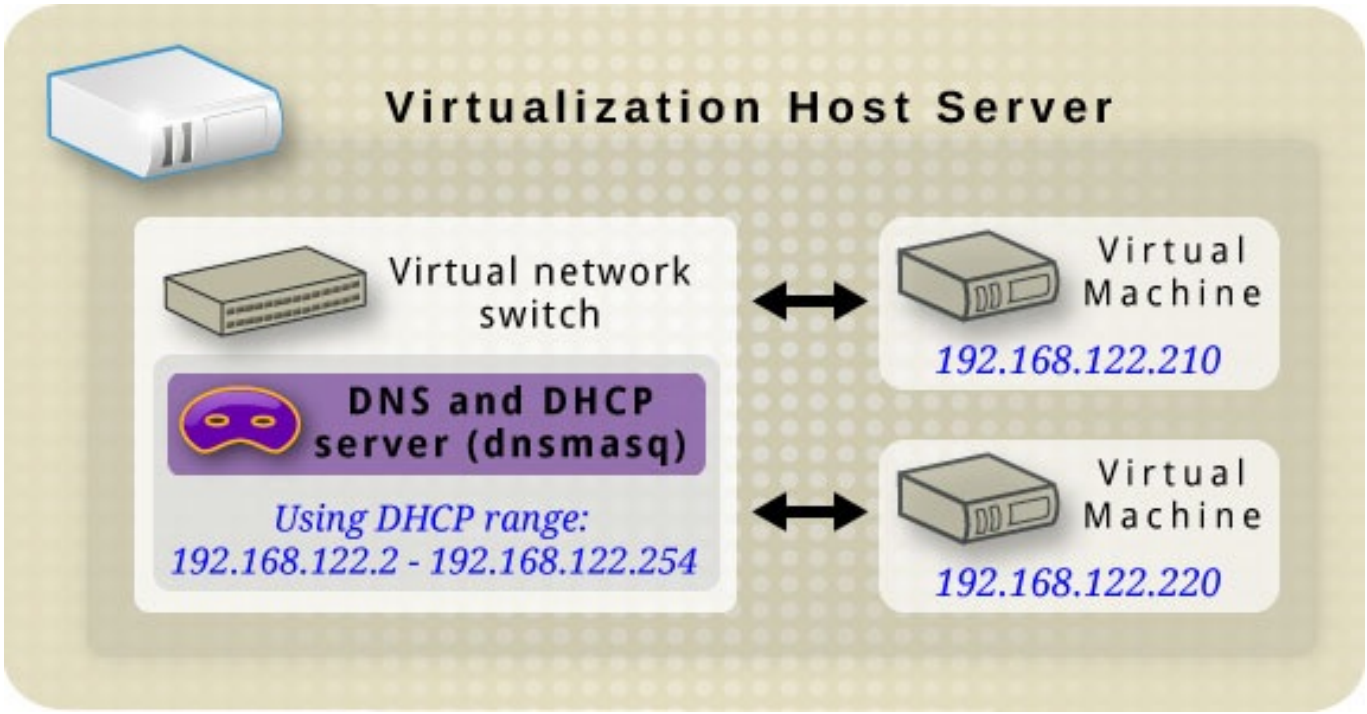


図22.4 dnsmasq を実行している仮想ネットワークスイッチ

22.3.2. 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。

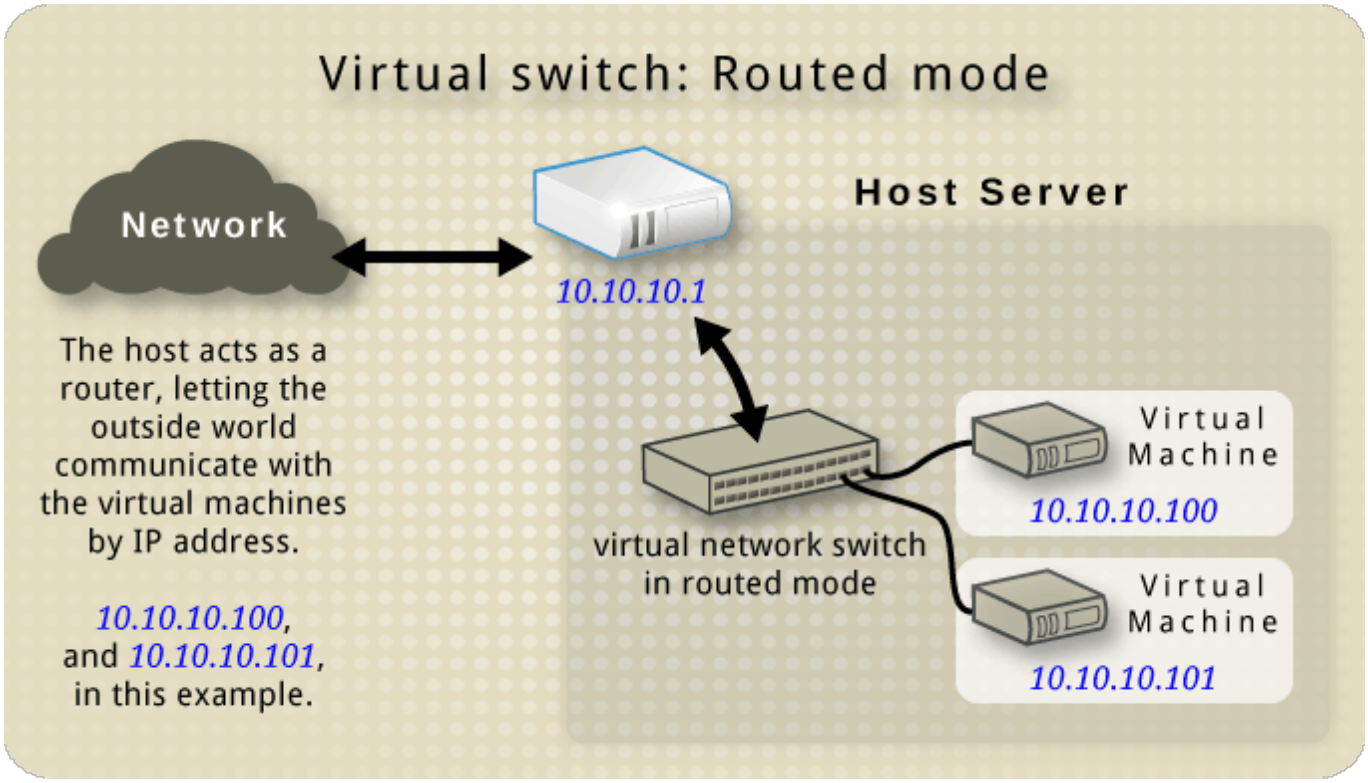


図22.5 ルーティングモードの仮想ネットワークスイッチ

22.3.3. 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。

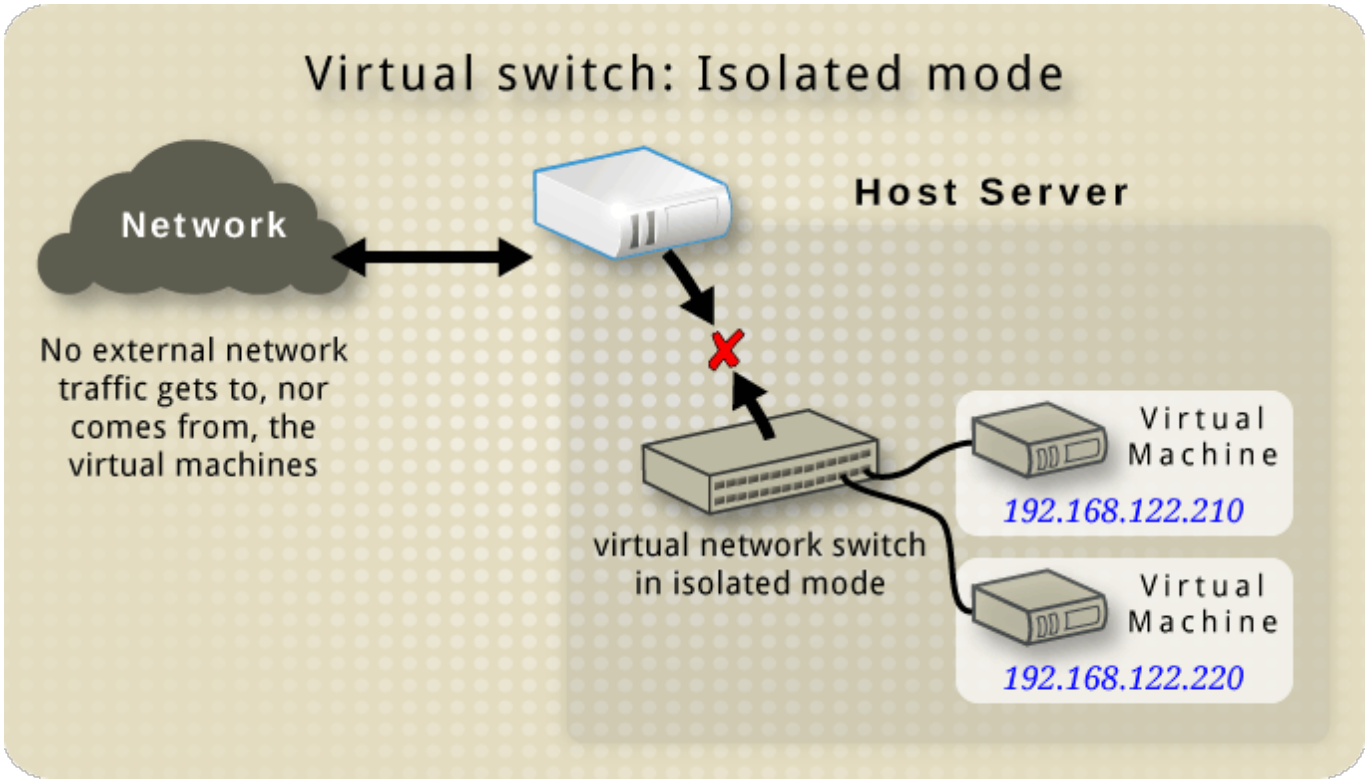


図22.6 隔離モードの仮想ネットワークスイッチ

22.4. デフォルト設定

libvirtd デーモン (**libvirtd**) 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 **libvirtd** のデフォルト設定を表しています。

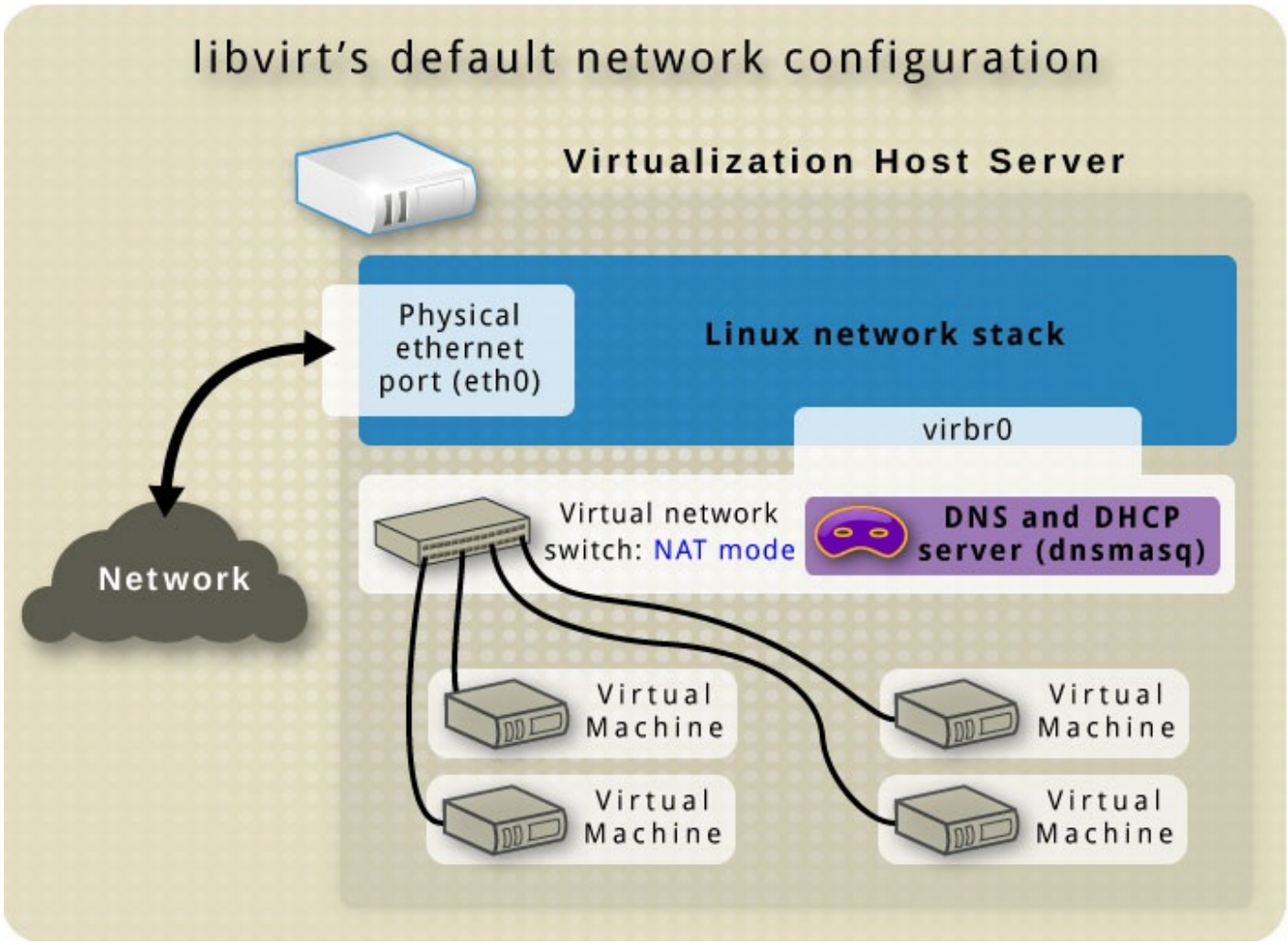


図22.7 libvirt のデフォルトネットワーク設定

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。

22.5. 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。

22.5.1. 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てみます。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。

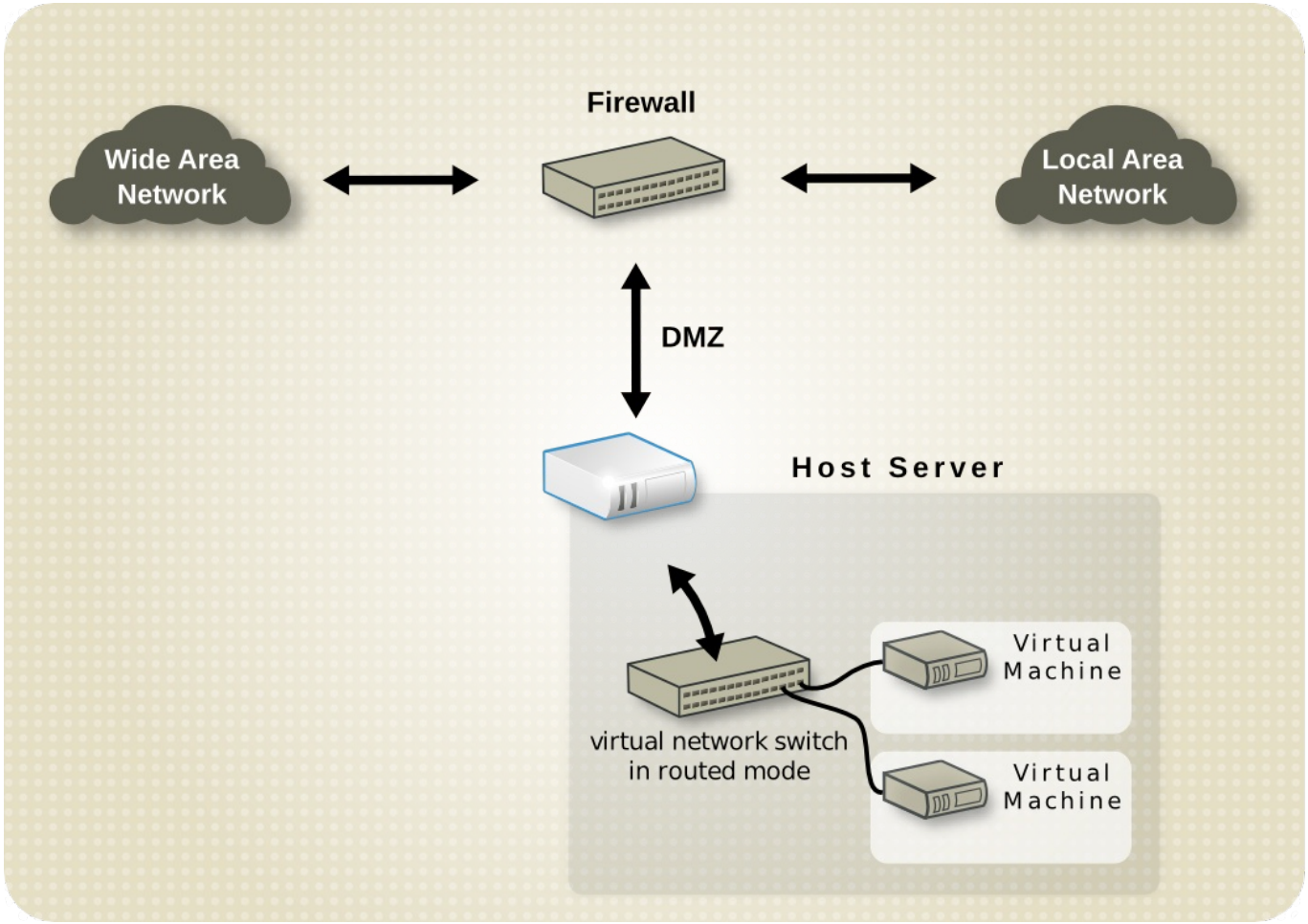


図22.8 DMZ の設定例

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ティングに使用し、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。

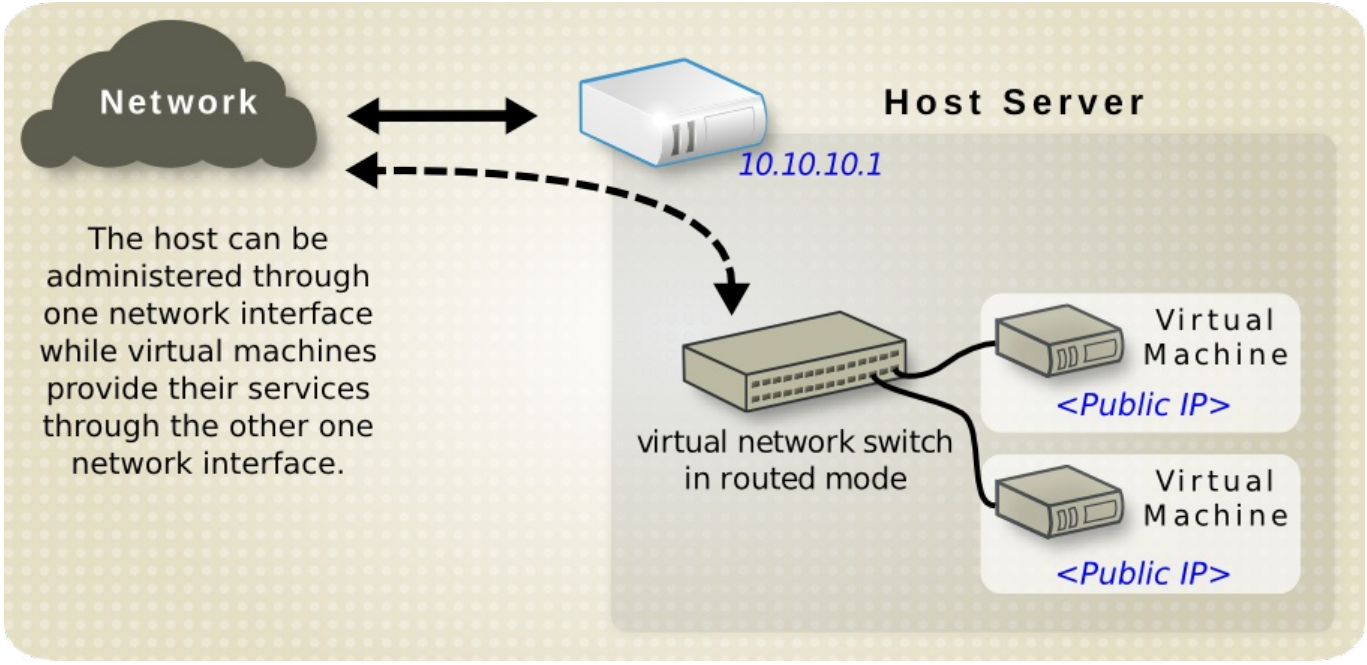


図22.9 仮想サーバーホスティングの設定例

ホスト物理マシンにパブリック IP アドレス、仮想マシンに静的なパブリック IP アドレスを持たせる場合、ブリッジしているネットワーク設定は使用できません。プロバイダーはパブリックのホスト物理マシンの MAC アドレスからしかパケットを受け取れないためです。以下の図はこれを説明しています。

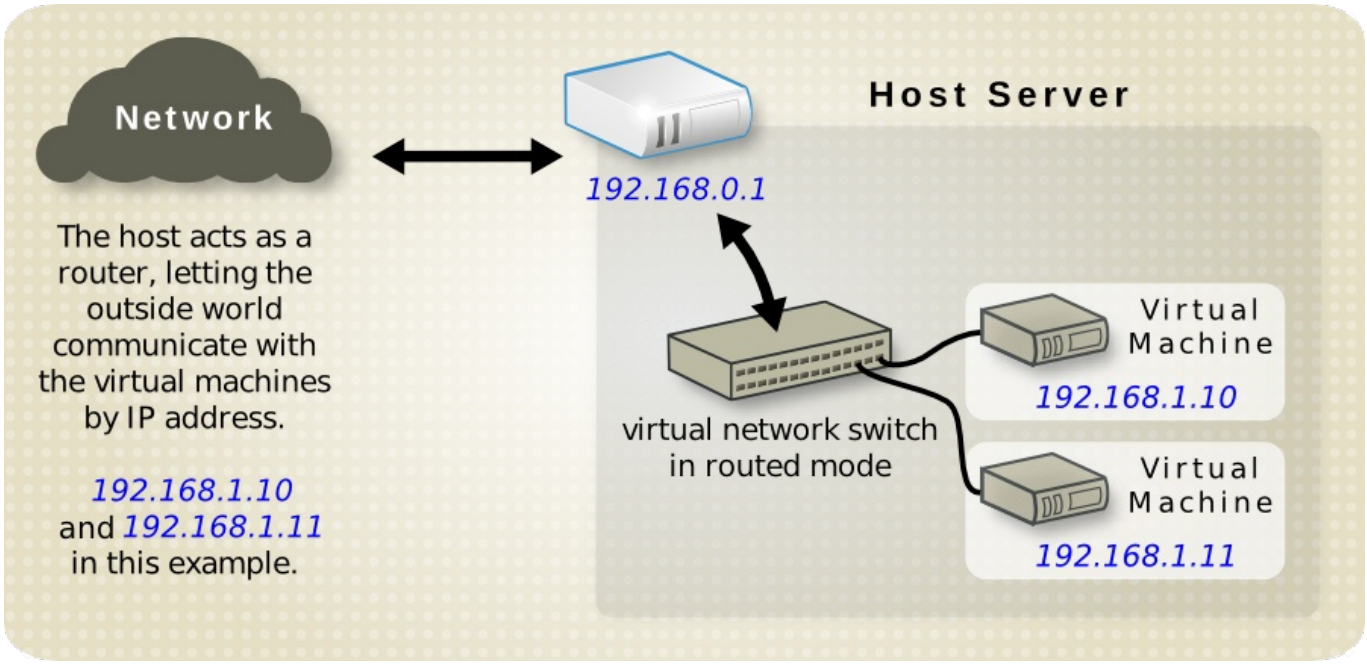


図22.10 静的 IP アドレスを使用する仮想サーバー

22.5.2. 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。

22.5.3. 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。

22.6. 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。

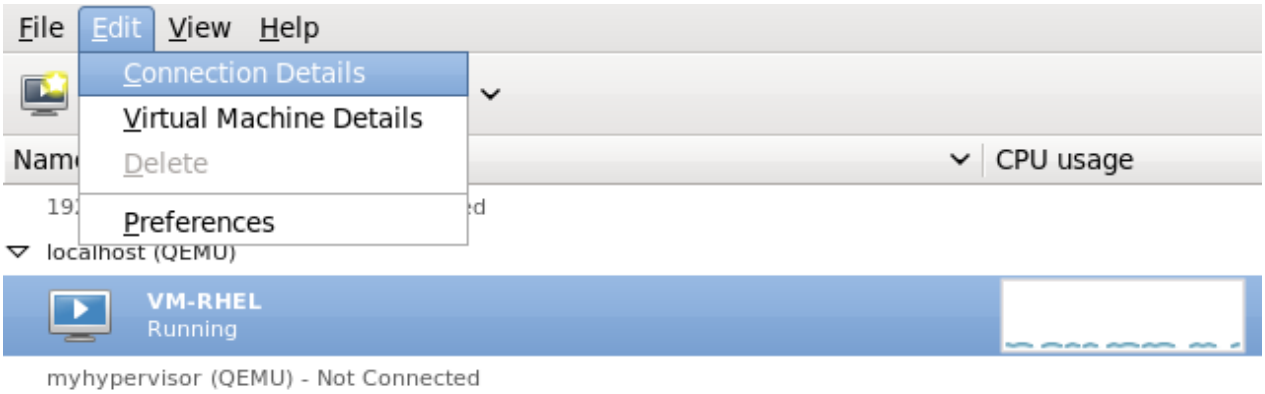


図22.11 ホスト物理マシンの詳細の選択

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。

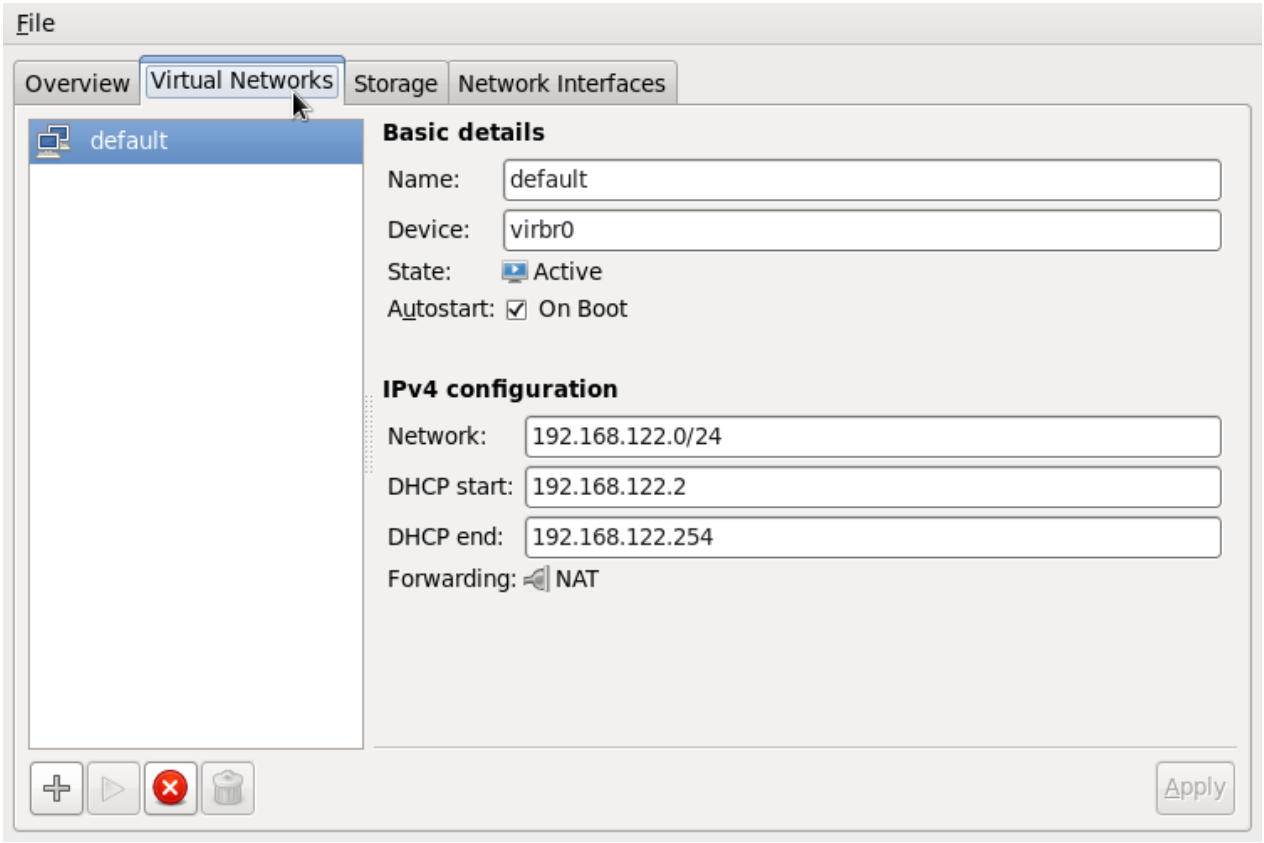


図22.12 仮想ネットワークの設定

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。

22.7. 仮想ネットワークの作成

システムで仮想ネットワークを作成する

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。

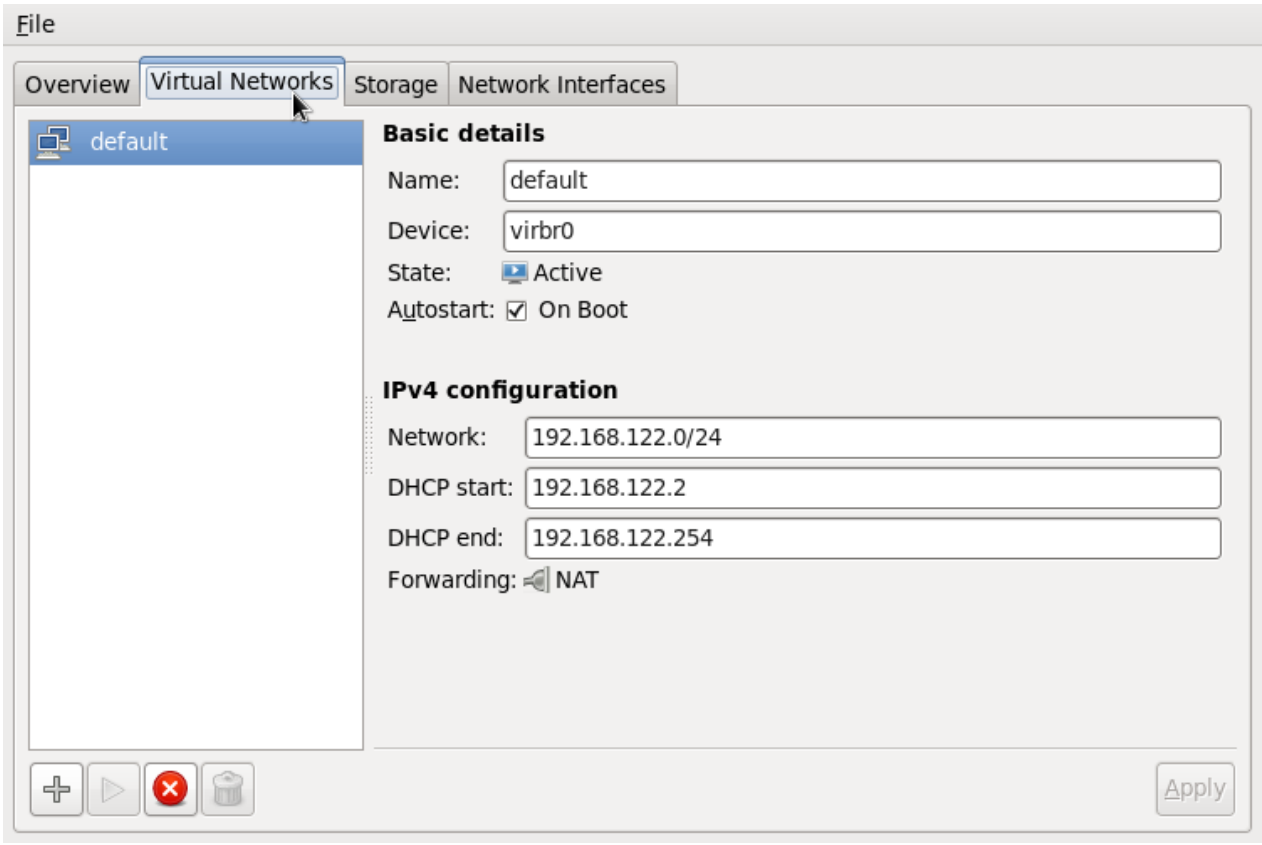


図22.13 仮想ネットワークの設定

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。

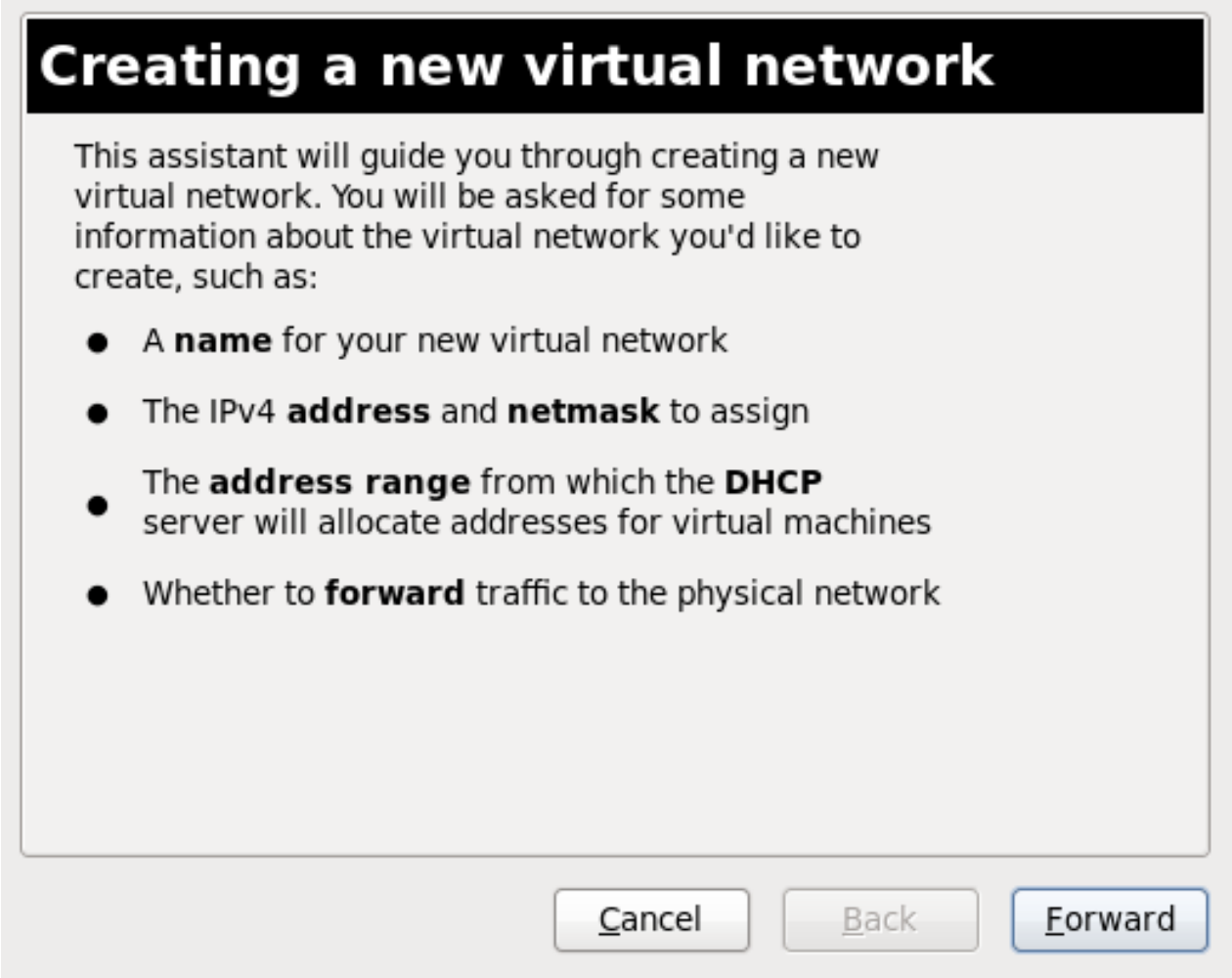


図22.14 新しい仮想ネットワークの作成

2. 仮想マシンの名前を入力して **進む** をクリックします。

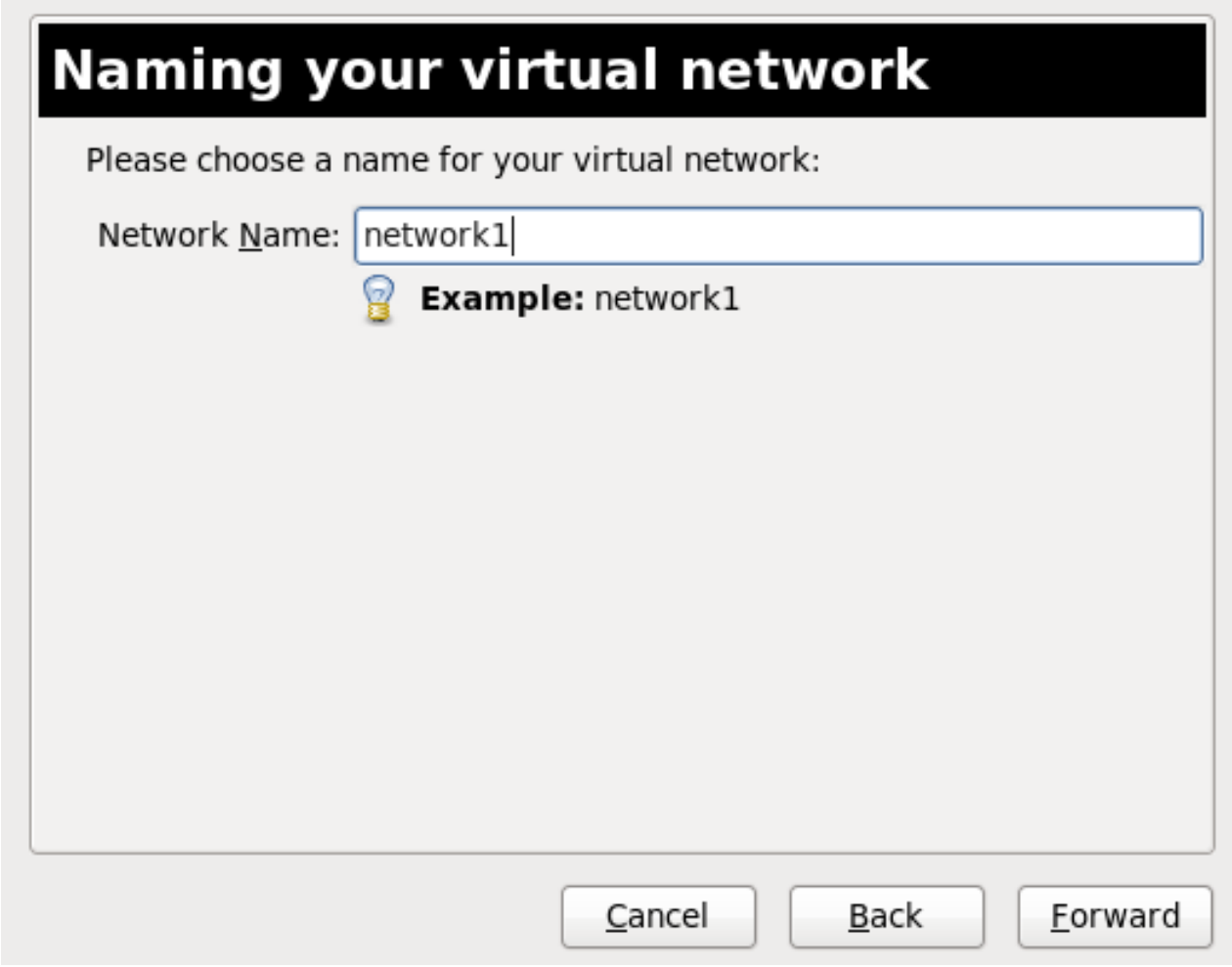


図22.15 仮想ネットワークの名前の入力

3. 仮想ネットワーク用の IPv4 アドレス領域を入力し **進む** をクリックします。

図22.16 IPv4 アドレス領域の選択

4. 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。**進む** をクリックして先に進みます。

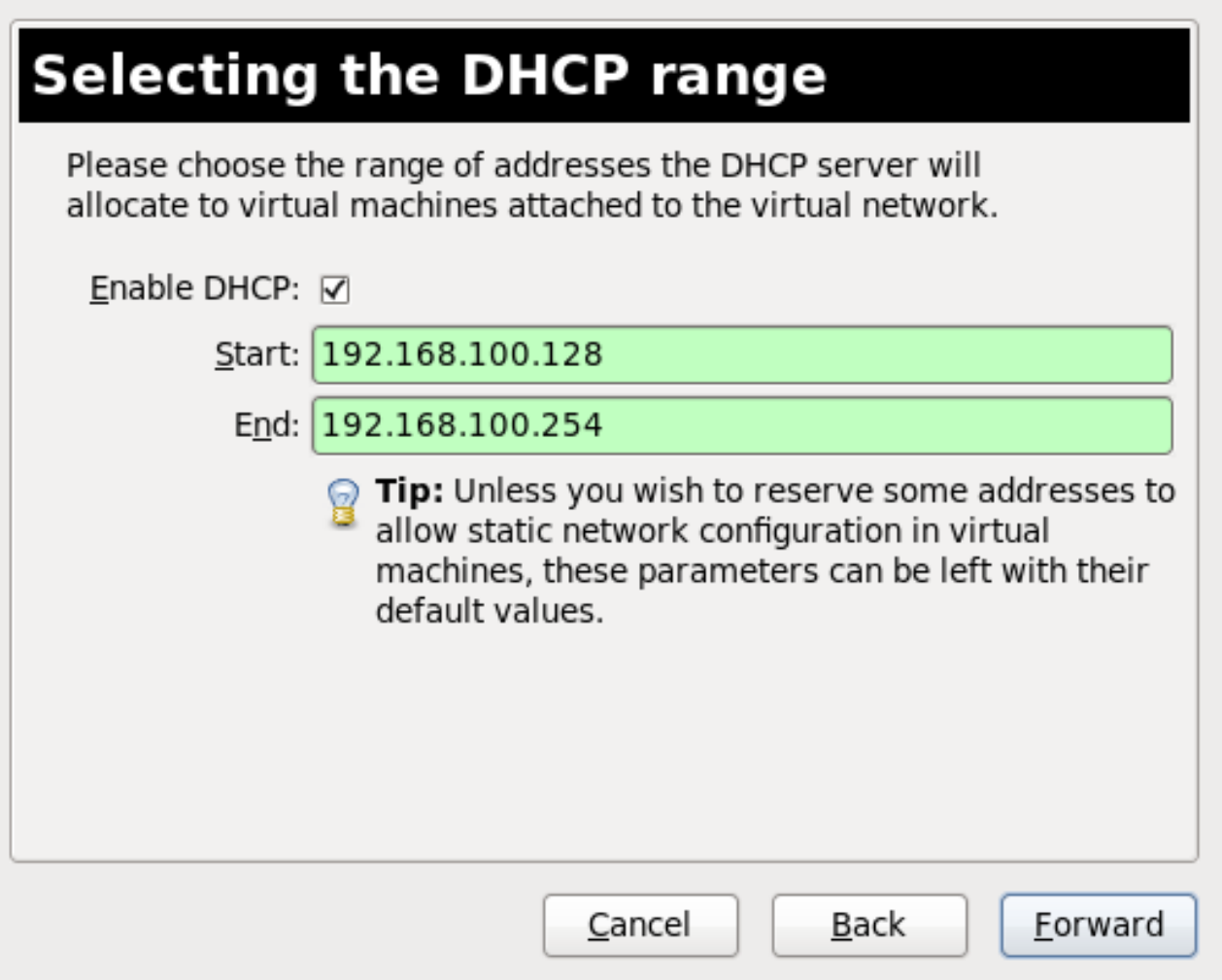


図22.17 DHCP 範囲の選択

5. 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。

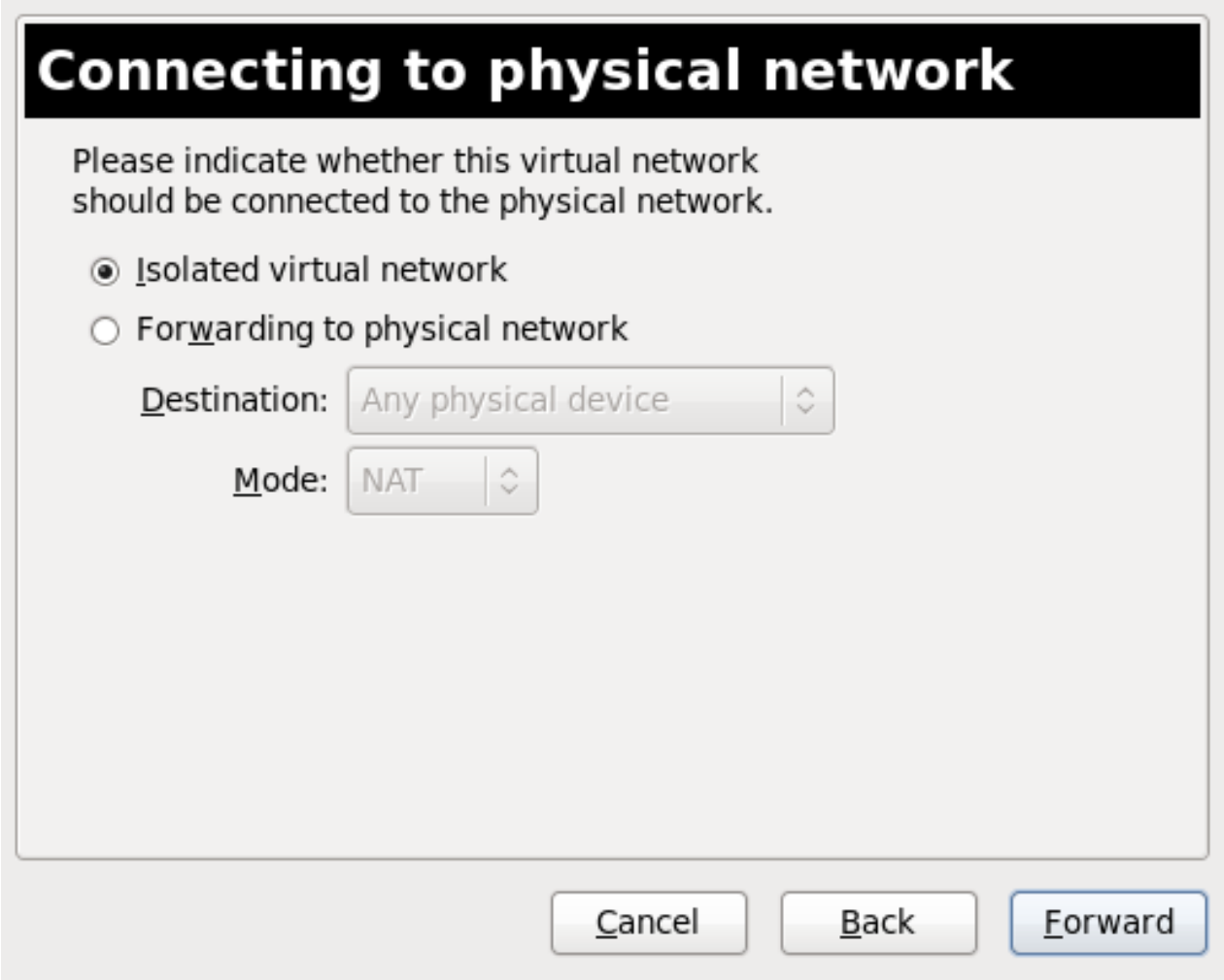


図22.18 物理ネットワークへの接続

物理ネットワークにフォワード を選択する場合は、**宛先** に **いずれかの物理デバイス** か特定の物理デバイスを選択します。また、**モード** には **NAT** か **ルーティング** のいずれかを選択します。

進む をクリックして先に進みます。

6. これでネットワーク作成の準備が整いました。 ネットワークの設定を確認して、**完了** をクリックします。

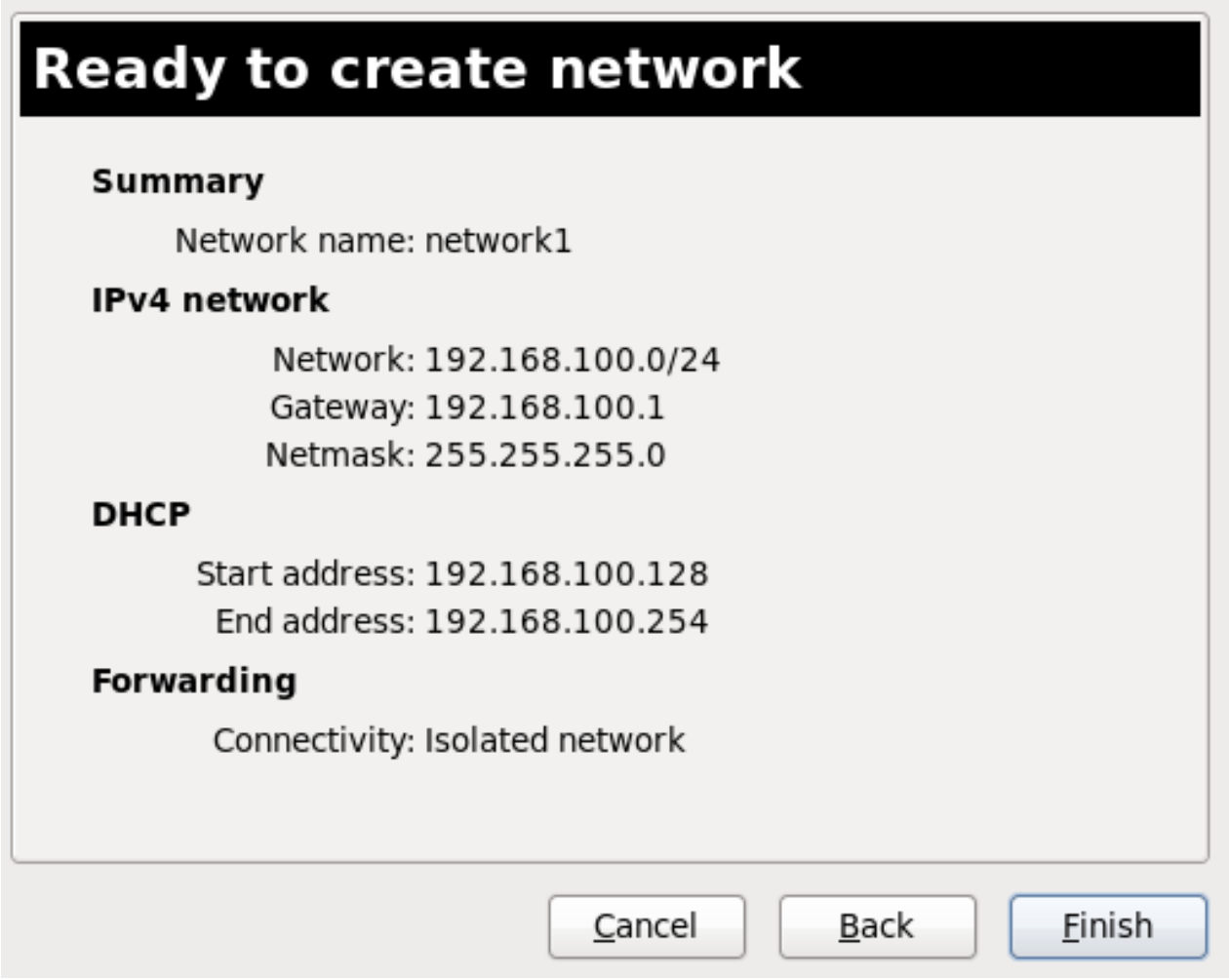


図22.19 ネットワーク作成の準備完了

7. 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。

22.8. 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。

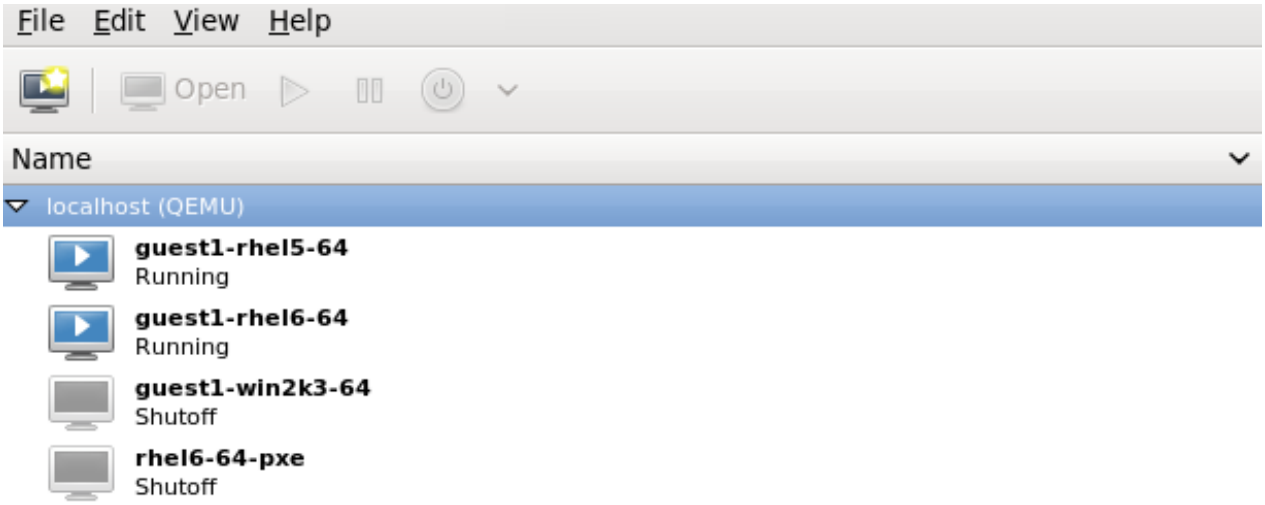


図22.20 表示する仮想マシンの選択

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。

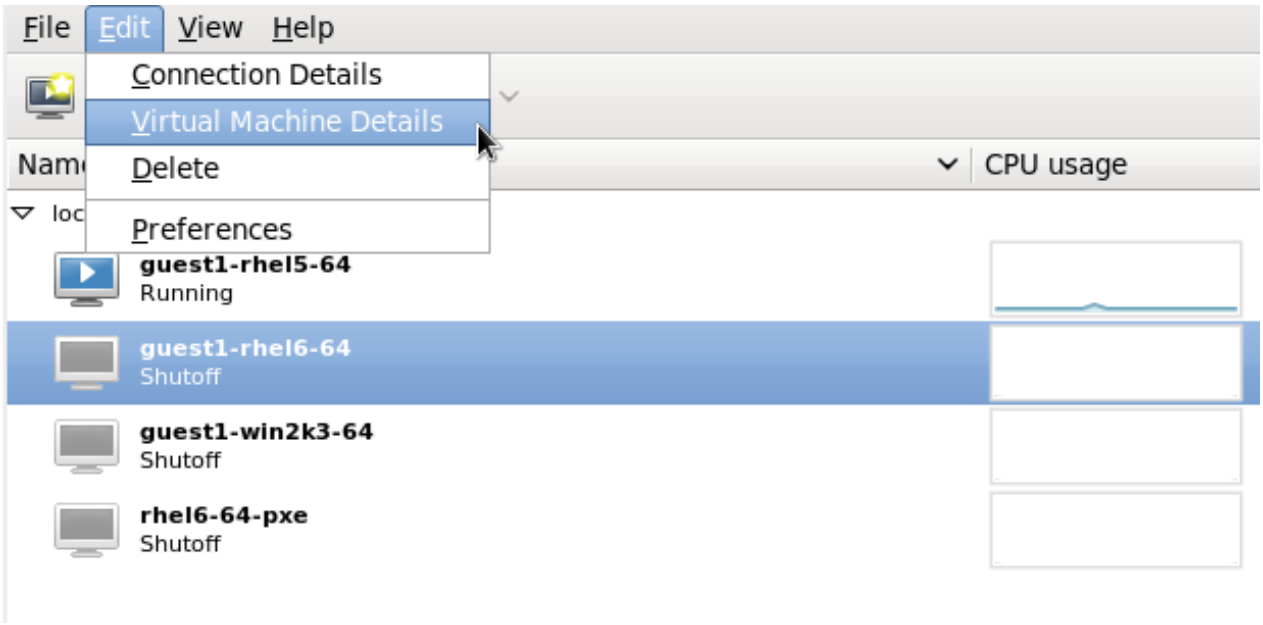


図22.21 仮想マシンの詳細を表示

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。

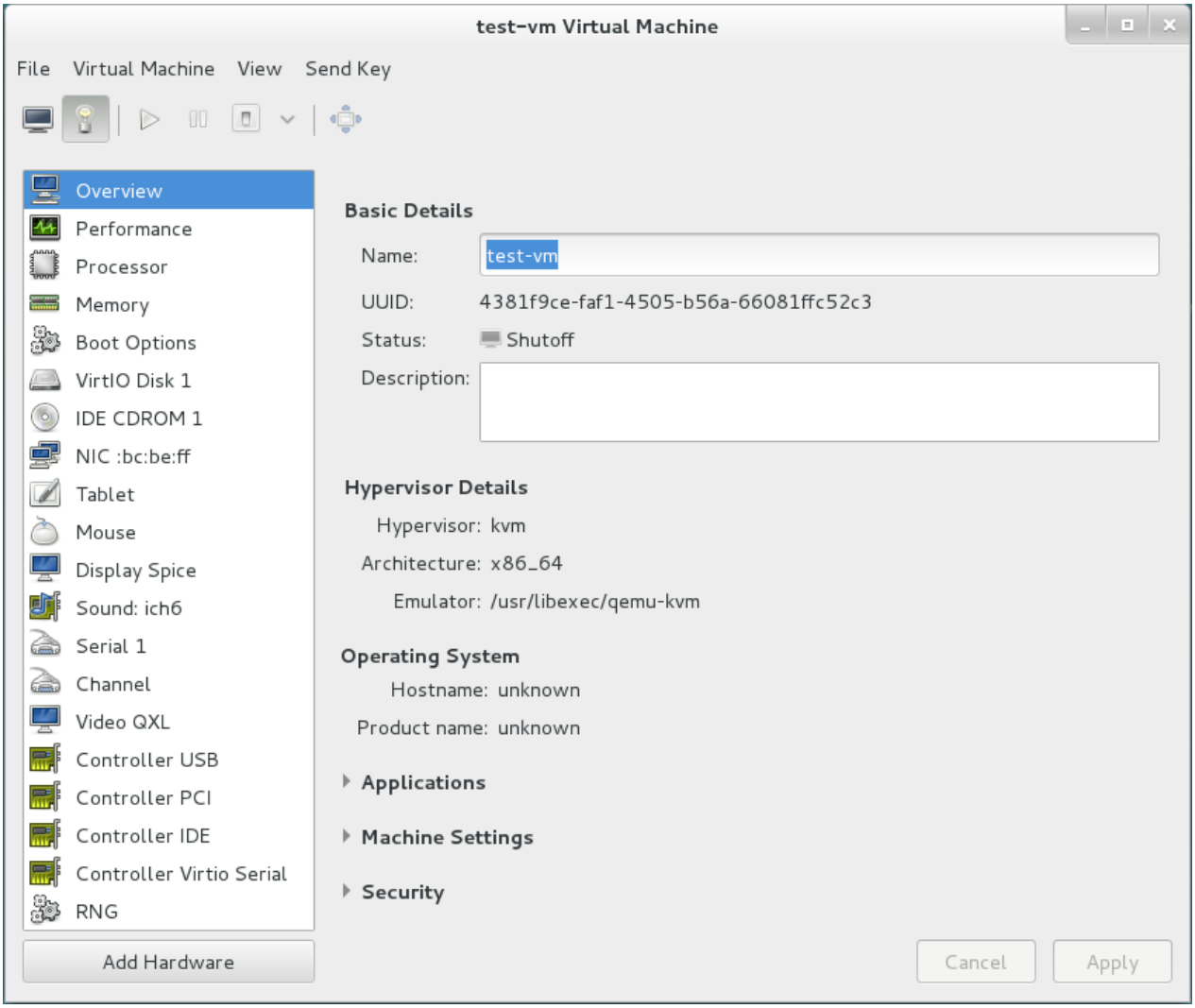


図22.22 仮想マシンの詳細ウィンドウ

4. **新規の仮想ハードウェアの追加 (Add new virtual hardware)** ウィンドウ内の左側のペインから **ネットワーク (Network)** を選択し、**ホストデバイス** メニューでネットワーク名 (この例では **network1**) を選択して **完了** をクリックします。

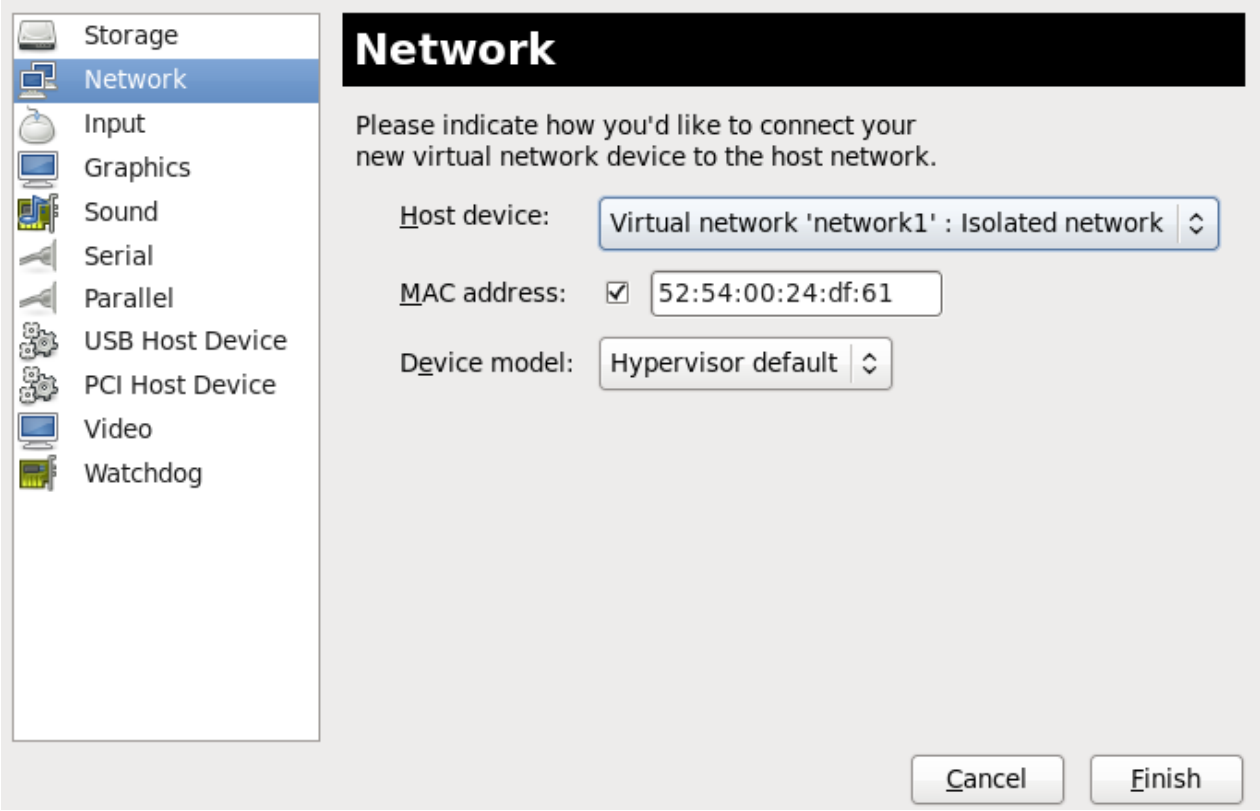


図22.23 新規の仮想ハードウェアの追加ウィンドウからネットワークを選択する

5. 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。

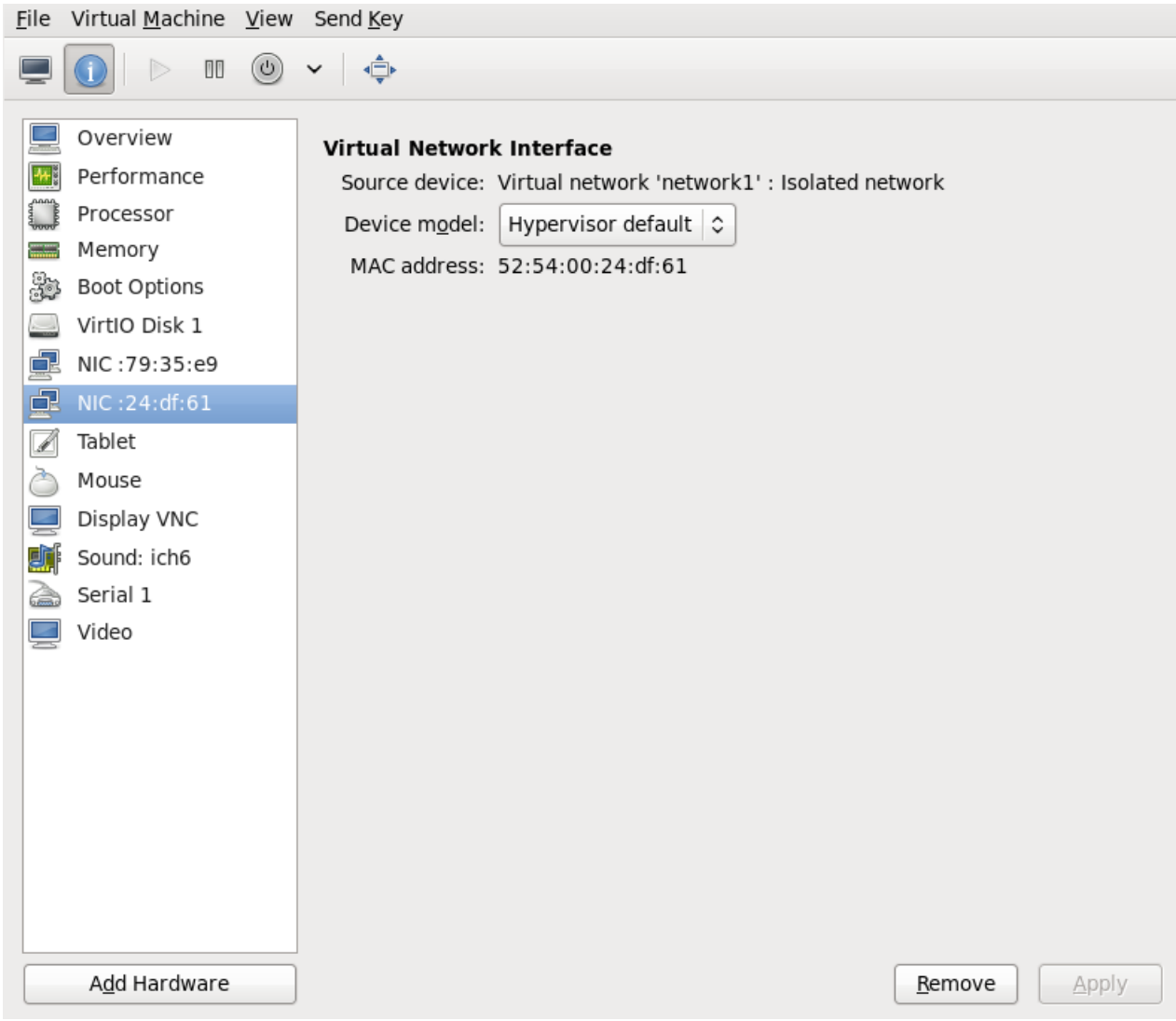


図22.24 ゲストのハードウェア一覧に表示される新しいネットワーク

22.9. 物理インターフェースへの直接割り当て

本章の説明は、仮想マシンの NIC をホスト物理マシンの指定された物理インターフェースに直接割り当てることを支援するために用意されています。割り当て方法 (パススルーとして知られる) を使用する場合は、20章*ゲスト仮想マシンデバイスの設定*および「インターフェースコマンド」を参照してください。このセットアップでは、Linux macvtap ドライバーを利用可能な状態にしておく必要があります。macvtap デバイスの操作モードについては、「vepa」がデフォルトのモードですが、4 つのモードを選択できます。それらの動作を以下に示します。

物理インターフェースの配信モード

vepa

仮想マシンのパケットはすべて外部ブリッジに送信されます。宛先が送信元と同じホスト物理マシン上にある仮想マシンになるパケットは VEPA 対応のブリッジによってホスト物理マシンに送り返されます (最近のブリッジは一般的には VEPA には対応していません)。

bridge

宛先が送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 macvtap デバイスに直接配信されます。送信元のデバイスと目的地のデバイスはいずれも直接配信用の bridge モードにし

ておく必要があります。いずれかを vepa モードにする場合は、VEPA 対応のブリッジが必要です。

private

すべてのパケットは外部ブリッジに送信されます。それらのパケット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 private モードの場合に、この手順が実行されます。

passthrough

移行機能を損なうことなく、SRIOV 対応の NIC の仮想機能を直接 仮想マシンに接続します。すべてのパケットが設定されたネットワークデバイスの VF/IF に送信されます。そのデバイスの機能によっては、追加の要件または制約が適用される場合があります。たとえば、linux では 2.6.38 またはそれ以降のカーネルが必要になります。

4 種類の各モードは、ドメインの XML ファイルを変更して設定します。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' 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。

IEEE 802.1Qbg の場合、VLAN ID にはゼロ以外の値が必要になります。また、スイッチが IEEE 802.1Qbg 標準に設定されている場合、その値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。

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 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、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ないます。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>
  ...
```

22.10. 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
```

22.11. 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。

22.11.1. 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。ルールの変更を行なう場合は、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22.1 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22.2 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ーは 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ーの目的です。

22.11.2. 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ることができます。

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22.3 ARP トラフィックフィルター

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。

22.11.3. 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22.1 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
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22.1「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。

22.11.4. 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の 例22.4「変数タイプの例」 に示します。

例22.4 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22.5 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。

22.11.5. 自動 IP アドレス検出と DHCP スヌーピング

22.11.5.1. はじめに

仮想マシンのインターフェースで使用される 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 ***TRL_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なえなくなります。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使用できる 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なわれず、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。

22.11.5.2. 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、***DHCPSERVER*** 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ないます。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なわないようにして、フィルターが適用されないようにます。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 DHCPSERVER 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22.6 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>
```

22.11.6. 予約済み変数

表22.2「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22.2 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
DHCPSERVER	信頼できる DHCP サーバーの IP アドレス一覧
DHCPSERVERV6	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
CTRL_IP_LEARNING	IP アドレス検出モードの選択

22.11.7. 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。

22.11.8. 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22.7 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。

22.11.9. 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22.8 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22.7「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な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。

22.11.10. 対応しているプロトコル

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ーの***attribute2*** が ***value2*** に一致せず、かつプロトコルプロパティー ***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。

22.11.10.1. 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22.3 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'/>
[...]
```

22.11.10.2. 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22.4 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 (0x0-0xfff, 0 - 4095)	VLAN ID
encap-protocol	UINT16 (0x03c-0xfff), 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

22.11.10.3. 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22.5 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 (0x0-0xfff, 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)
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

属性名	データタイプ	定義
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

22.11.10.4. 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22.6 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
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
gratututous	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 Boolean
comment	STRING	最長 256 文字のテキスト文字列

22.11.10.5. 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22.7 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
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。protocol 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

22.11.10.6. 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22.8 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
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

属性名	データタイプ	定義
protocol	UINT8, STRING	レイヤー 4 プロトコル識別子。protocol に有効な文字列: tcp、udp、udplite、esp、ah、icmpv6、sctp
scrportstart	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	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

22.11.10.7. 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
設定する必要があります。

表22.9 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
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
scrportstart	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	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
state	STRING	NEW、ESTABLISHED、RELATED、INVALID または NONE のコンマで区切った一覧

属性名	データタイプ	定義
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。

22.11.10.8. 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22.10 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	MAD_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
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要。

22.11.10.9. 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22.11 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	MAD_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
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。

22.11.10.10. 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トに
設定します。

表22.12 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

属性名	データタイプ	定義
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
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。

22.11.10.11. 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22.13 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
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。

22.11.10.12. 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22.14 IPv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
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。

22.11.11. 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。

22.11.11.1. 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、追跡を行なうことにより、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22.9 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。

22.11.11.2. 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22.10 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 22.10「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。

22.11.11.3. 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)。

22.11.11.4. 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22.15 ICMPv6 プロトコルタイプ

コマンド名	説明

コマンド名	説明
no-arp-spoofing	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。このフィルターは ARP 要求と返信メッセージだけを許可し、パケットにゲスト仮想マシンの MAC と IP アドレスが必ず含めるよう強制します。
allow-dhcp	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)。
allow-dhcp-server	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 *DHCPSERVER* にしてください。
no-ip-spoofing	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。
no-ip-multicast	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。
clean-traffic	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。

22.11.11.5. 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か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な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。「対応しているプロトコル」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 etable 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22.11 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ないません。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>
```

22.11.11.6. 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22.12 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、ICM 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22.13 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** - このコマンドが使用できる場合はこれを使います。
- **#modprobe ip_conntrack_ftp** - 上記のコマンドが使えない場合にこのコマンドを使用します。

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 and 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>

  <!- - This rule drops all other traffic - ->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>

</filter>
```

22.11.12. 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22.1「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。

22.12. 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。

22.12.1. 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、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22.25 マルチキャストトンネルのドメイン XML サンプル

22.12.2. 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22.26 TCP トンネルのドメイン XML サンプル

22.13. 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 (パススルー) 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22.27 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。

22.14. 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23章 ゲストのリモート管理

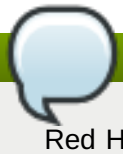
このセクションでは、**ssh** または TLS および SSL を使用してゲストをリモートで管理する方法を説明しま す。SSH の詳細は、『Red Hat Enterprise Linux 導入ガイド』 をご覧ください。

23.1. 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フ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多くなると、SSH のスケーラビリティーは低下します。



注記

Red Hat Enterprise 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。さらに詳しくは、Red Hat Enterprise 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な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人になります。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、所有している必要があります。

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:
   ```

   ここで、**ssh root@host2.example.com** コマンドを使ってマシンにログインしてみます。**.ssh/authorized_keys** ファイルに予期しないキーが追加されていないことを確認します。

   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**

   既存の ssh-agent にパスフレーズを追加する方法を以下に示します。この作業は ssh-agent を実行していないと失敗します。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。さらに詳しくは、『Red Hat Enterprise Linux 導入ガイド』 を参照してください。

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
```

SSH キーがリモートのシステムに追加されます。

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
```

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。パスワード入力をし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 を起動します。
2. **ファイル->接続を追加** の順に開きます。

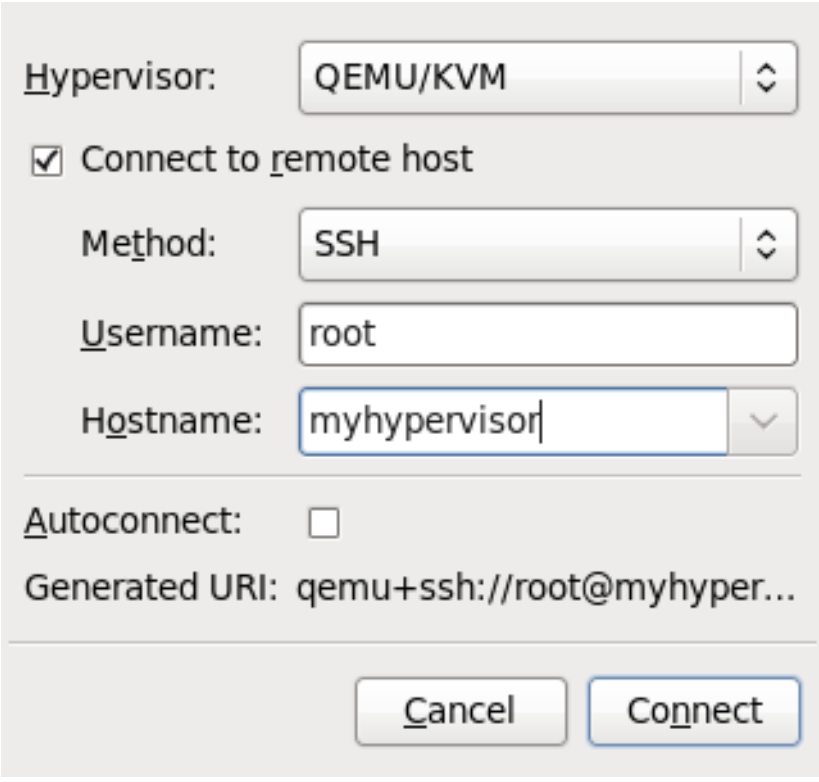


図23.1 接続を追加のメニュー

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。

23.2. TLS と SSL 経由のリモート管理

TLS と SSL 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ーが向上しますが、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 ブラウザーで使用さ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なわれ、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23.1 TLS 管理の認証局 (CA) キーを作成する

1. まず **certtool** 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次のようにしてインストールします。

```
# yum install certtool
```

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。

```
# vim ca.info
```

```
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 ファイルは削除して構いません。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。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** 認証局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、この認証局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23.2 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 にします。

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. 次のコマンドで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
```

4. 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

   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、次のコマンドを実行します。

```
# certtool -i --inifile 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23.3 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) 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
```

この手順は 手順23.2「サーバー証明書の発行」 とよく似ていますが、次の点が異なります。

2. プライベートキーを次のコマンドで作成します。

```
# 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めます (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
```

23.3. 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ッスンします。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。

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となります。ソケットは暗号化されず、認証には SELinux または UNIX のパーミッションを使用します。標準のソケット名は **/var/run/libvirt/libvirt-sock** と **/var/run/libvirt/libvirt-sock-ro** (読み取り専用接続) です。

SSH

Secure Shell protocol (SSH) 接続経由でトランスポートされます。Netcat (*nc* パッケージ) をインストールしておく必要があります。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。SSH アクセス用にポート 22 を開いておく必要があります。いずれかの SSH キー管理 (**ssh-agent** など) を使用しないとパスワードの入力が求められます。

ext

ext パラメーターは、libvirt の対象範囲外となる手段でリモートマシンに接続を行う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、テストを行う場合や信頼できるネットワークで使用する場合などには管理者によって有効にされることがあります。デフォルトのポートは 16509 です。

他に指定がない場合、デフォルトのトランスポートモードは 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** コマンドに **--connect** パラメータを付けて使用すると、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ー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 を渡すのと同等です。

libvirt URI は汎用の形式を取ります (角括弧 [] 内の内容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 (ドライバー) が QEMU の場合、パスは必須になります。XEN の場合、パスはオプションになります。

以下は有効なリモート URI のいくつかの例です。

- qemu://hostname/
- xen://hostname/
- xen+ssh://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。さらに詳しくは、http://libvirt.org/guide/html/Application_Development_Guide-Architecture-Remote_URIs.html 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 [<name>] [--readonly]** です。ここで、**<name>** は説明されているように有効な URI になります。**virsh connect** コマンドの詳細は、「接続」 を参照してください。

```
qemu+ssh://virtuser@host2/
```

- ホスト上にある **host2** という名前のリモート KVM ハイパーバイザーに接続します。TLS を使用します。

```
qemu://host2/
```

テスト事例

- 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で 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るパラメーターを示します (表23.1「追加の URI パラメーター」)。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 エスケープ処理しなければならない点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23.1 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	すべてのモード	name がリモートの **virConnectOpen** 関数に渡されます。name は通常リモートの URI から ***transport***、***hostname***、***port number***、***username***、および追加パラメーターを取り除いたものになります。ただし、非常に複雑な事例の場合、name を明示的に指定する方がよい場合があります。	name=qemu:///system

名前	トランスポートモード	説明	使用法の例
コマンド	ssh と ext	外部コマンドです。ext トランスポートの場合に必要です。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
ソケット	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、これがリモートの **netcat** コマンドに渡されます (netcat を参照)。	socket=/opt/libvirt/run/libvirt/libvirt-sock
netcat	ssh	リモートシステムに接続する場合に **netcat** コマンドを使用することができます。デフォルトの netcat パラメーターは **nc** コマンドを使用します。SSH トランスポートの場合、libvirt により以下の形式で SSH コマンドが構成されます。 ***command*** -p ***port*** [-l ***username***] ***hostname*** ***netcat*** -U socket ***port***、***username*** および ***hostname*** の各パラメーターをリモート URI の一部として指定できます。***command***、***netcat***、および ***socket*** は他の追加パラメーターから取られたものです。	netcat=/opt/netcat/bin/nc
no_verify	tls	ゼロ以外の値に設定すると、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、libvirtd の設定を変更する必要があります。	no_verify=1

名前	トランスポートモード	説明	使用法の例
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 SSH 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

23.4. VNC サーバーの設定

VNC サーバーを設定するには、**システム > 設定** にある **リモートデスクトップ** アプリケーションを使用します。または、**vino-preferences** コマンドを実行することもできます。

次の手順に従って、専用 VNC サーバーセッションのセットアップを行ないます。

必要であれば、**~/.vnc/xstartup** ファイルを作成し、**vncserver** が起動した場合は常に GNOME セッションを起動するよう編集します。**vncserver** スクリプトを初めて実行する際に、VNC セッションに使用するパスワードの入力が求められます。vnc サーバーファイルの詳細は、『Red Hat Enterprise Linux インストールガイド』 を参照してください。

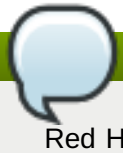
第24章 KSM

最近のオペレーティングシステムでは共有メモリーという概念が一般的になってきました。たとえば、プログラムが始めて起動する時にはその親プログラムと全メモリーを共有します。子プログラムまたは親プログラムのいずれかがメモリーの変更を試行すると、カーネルによって新しいメモリー領域が割り当てられ、オリジナルのコンテンツがコピーされます。これによりプログラムが新しい領域を変更できるようになります。これはコピーオンライト (copy on write) と呼ばれています。

KSM はこの概念を逆に応用した Linux の新しい機能になります。KSM により、カーネルはすでに実行中の複数のプログラムを検査し、それらのメモリーを比較することができます。メモリーの領域またはページがまったく同一である場合は、KSM は複数ある同一メモリーページを 1 つのページに減らし、このページにはコピーオンライトのマークが付けられます。ページのコンテンツがゲスト仮想マシンによって変更された場合は、そのゲスト仮想マシン用に新しいページが作成されます。

KSM は KVM を使った仮想化に便利です。ゲスト仮想マシンの起動時に、ゲスト仮想マシンは親の **qemu-kvm** プロセスからのメモリーしか継承しません。同じオペレーティングシステムやアプリケーションを複数のゲストが実行している場合、ゲスト仮想マシンがゲスト仮想マシンのオペレーティングシステムのコンテンツを実行し始めると、イメージを共有することができるようになります。KSM はまったく同一のページのみを識別し、マージします。このため、ゲスト仮想マシンが干渉されたり、ホスト物理マシンやゲストの安全に影響を与えることはありません。KSM の機能によって KVM は同一のゲスト仮想マシンのメモリー領域が共有されるよう要求することができます。

KSM によってメモリーの速度やその用途が広がります。KSM を使用すると、共通の処理データはキャッシュやメインメモリーに格納されます。これにより KVM ゲストのキャッシュミスが低減されるため、一部のアプリケーションやオペレーティングシステムのパフォーマンスを向上させることができます。また、メモリーを共有することによりゲストの全体的なメモリー使用を抑えるため、より多くのリソースを有効に活用できるようになります。



注記

Red Hat Enterprise Linux 7 より、KSM は NUMA 対応になりました。これにより、ページコアレッシングの実行時に NUMA ローカリティーが考慮されることになり、リモートノードに移行されるページに関連するパフォーマンスの低下を防ぐことができるようになります。Red Hat は、KSM の使用時にはノード間のメモリーマージを控えることを推奨します。KSM が使用中の場合には、**/sys/kernel/mm/ksm/merge_across_nodes** 調整可能パラメーターを ***0*** に変更し、複数の NUMA ノード間でのページのマージを防ぎます。これは、コマンド **virsh node-memory-tune --shm-merge-across-nodes 0** を使って実行できます。多量のノード間マージが実行されると、カーネルメモリーのアカウンティング統計は相反する結果となる可能性があります。そのため、numad も KSM デーモンが多量のメモリーをマージした後に混乱する可能性があります。システムに大量の空きメモリーがある場合、KSM デーモンをオフにし、無効にすることでパフォーマンスを向上させることができます。NUMA の詳細は、『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。

Red Hat Enterprise Linux では、KSM の管理に 2 種類の異なるメソッドを使用しています。

- **ksm** サービス: KSM カーネルスレッドの起動と停止を行います。
- **ksmtuned** サービス: **ksm** の制御と調整を行い、同じページのマージを動的に管理します。このサービスは **ksm** を起動し、 メモリー共有が必要ない場合には**ksm** サービスを停止します。新規のゲストが作成されたり、ゲストが破棄された場合には、***retune*** パラメーターで**ksmtuned** サービスに実行の指示を出さなければなりません。

いずれのサービスも標準のサービス管理ツールで制御されます。

24.1. KSM サービス

24.1. KSM サービス

ksm サービスは *qemu-kvm* パッケージに含まれています。Red Hat Enterprise Linux 7 の KSM はデフォルトではオフになっています。ただし、Red Hat Enterprise Linux 7 を KVM ホスト物理マシンとして使用する場合は **ksm/ksmtuned** サービスによってオンに設定される可能性があります。

ksm サービスを起動していない場合は、KSM で共有されるページは 2000 ページのみになります。このデフォルト値ではページ数が少ないため、メモリー節約で得られる利点も限られます。

ksm サービスを起動すると、KSM はホスト物理マシンシステムのメインメモリーを最大 50% まで共有するようになります。**ksm** サービスを起動して KSM がより多くのメモリーを共有できるようにします。

```
# systemctl start ksm
Starting ksm:                                              [  OK  ]
```

ksm サービスをデフォルトのスタートアップ順序に追加することができます。 systemctl コマンドを使って **ksm** サービスを永続化します。

```
# systemctl enable ksm
```

24.2. KSM チューニングサービス

ksmtuned サービスにはオプションがありません。**ksmtuned** サービスはループして **ksm** の調整を行います。ゲスト仮想マシンが作成されたり、破棄された場合は、libvirt によって **ksmtuned** サービスに通知されます。

```
# systemctl start ksmtuned
Starting ksmtuned:                                         [  OK  ]
```

ksmtuned サービスは ***retune*** パラメーターを使って調整を行います。***retune*** パラメーターの指示によって **ksmtuned** は手動によるチューニング機能を実行します。

ファイル内のパラメーターを変更する前に、明確に理解しておく必要のあるいくつかの用語があります。

- **npages** - **ksmd** がスリープ状態になる前に ksm がスキャンできるページ数です。これは **/sys/kernel/mm/ksm/pages_to_scan** で設定されます。
- **thres`** - キロバイト単位のアクティブ化しきい値です。`thres` 値とすべての `qemu-kvm` プロセスによって消費されるメモリー量である RSZ 値の合計がシステムメモリーの合計を上回ると、KSM サイクルがトリガーされます。このパラメーターは、`KSM_THRES_COEF` パラメーターに定義されるパーセンテージをキロ単位で表したものと同等です。

/etc/ksmtuned.conf ファイルは **ksmtuned** サービスの設定ファイルになります。デフォルトの **ksmtuned.conf** ファイルの出力を以下に示します。

```
# Configuration file for ksmtuned.

# How long ksmtuned should sleep between tuning adjustments
# KSM_MONITOR_INTERVAL=60

# Millisecond sleep between ksm scans for 16Gb server.
# Smaller servers sleep more, bigger sleep less.
# KSM_SLEEP_MSEC=10

# KSM_NPAGES_BOOST is added to the `npages` value, when `free memory` is
```

```
less than `thres`.
# KSM_NPAGES_BOOST=300

# KSM_NPAGES_DECAY Value given is subtracted to the `npages` value, when
`free memory` is greater than `thres`.
# KSM_NPAGES_DECAY=-50

# KSM_NPAGES_MIN is the lower limit for the `npages` value.
# KSM_NPAGES_MIN=64

# KSM_NAGES_MAX is the upper limit for the `npages` value.
# KSM_NPAGES_MAX=1250

# KSM_TRES_COEF - is the RAM percentage to be calculated in parameter
`thres`.
# KSM_THRES_COEF=20

# KSM_THRES_CONST - If this is a low memory system, and the `thres` value
is less than `KSM_THRES_CONST`, then reset `thres` value to
`KSM_THRES_CONST` value.
# KSM_THRES_CONST=2048

# uncomment the following to enable ksmtuned debug information
# LOGFILE=/var/log/ksmtuned
# DEBUG=1
```

24.3. KSM の変数とモニタリング

KSM はモニタリングデータを **/sys/kernel/mm/ksm/** ディレクトリーに格納します。このディレクトリー内のファイルはカーネルによって更新されるため、KSM の使用量と統計値の正確な記録となります。

以下に示す変数も **/etc/ksmtuned.conf** ファイル内にある設定可能な変数となります。

/sys/kernel/mm/ksm/ 以下のファイル

full_scans

実行された完全スキャン数

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。

pages_shared

共有されたページ合計数

pages_sharing

現在共有されているページ数

pages_to_scan

スキャンされなかったページ数

pages_unshared

共有されなくなったページ数

pages_volatile

揮発性のページ数

run

KSM プロセスが実行しているかどうか

sleep_millisecs

スリープのミリ秒数

virsh node-memory-tune コマンドを使用して、これらの変数を手動で調整することができます。以下が例になります。

```
# virsh node-memory-tune --shm-pages-to-scan number
```

共有メモリーサービスがスリープ状態になる前にスキャンするページの数を指定します。

/etc/ksmtuned.conf ファイルに ***DEBUG=1*** の行を追加すると、 KSM チューニングのアクティビティーが **/var/log/ksmtuned** ログファイルに格納されます。***LOGFILE*** パラメーターを使用するとログファイルの場所を変更することができます。ただし、ログファイルの場所の変更は、SELinux 設定に特殊な構成を必要とする場合があるためお勧めできません。

24.4. KSM の停止

KSM にはパフォーマンス上のオーバーヘッドがあり、特定の環境やホスト物理マシンシステムには負荷が大きすぎる場合があります。

ksmtuned と **ksm** のサービスを停止すると KSM の動作を停止させることができます。これらのサービスの停止により KSM は一時的に停止されますが、再起動後は元に戻ります。

```
# systemctl stop ksmtuned
Stopping ksmtuned:                                         [  OK  ]
# systemctl stop ksm
Stopping ksm:                                              [  OK  ]
```

再起動後も KSM の停止を永続化するには **systemctl** コマンドを使用します。サービスを無効にするには次のコマンドを実行します。

```
# systemctl disable ksm
# systemctl disable ksmtuned
```

重要

KSM を使用する場合であっても、コミットする RAM に対して swap サイズの大きさが十分であることを確認してください。KSM は同一または同様のゲストの RAM 使用量を低減します。十分なスワップ領域がなくても KSM を使ってゲストをオーバーコミットすることはおそらく可能ですが、ゲスト仮想マシンのメモリー使用によってページが共有されなくなる可能性があるため推奨されません。

第25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

このセクションでは、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

25.1. 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。

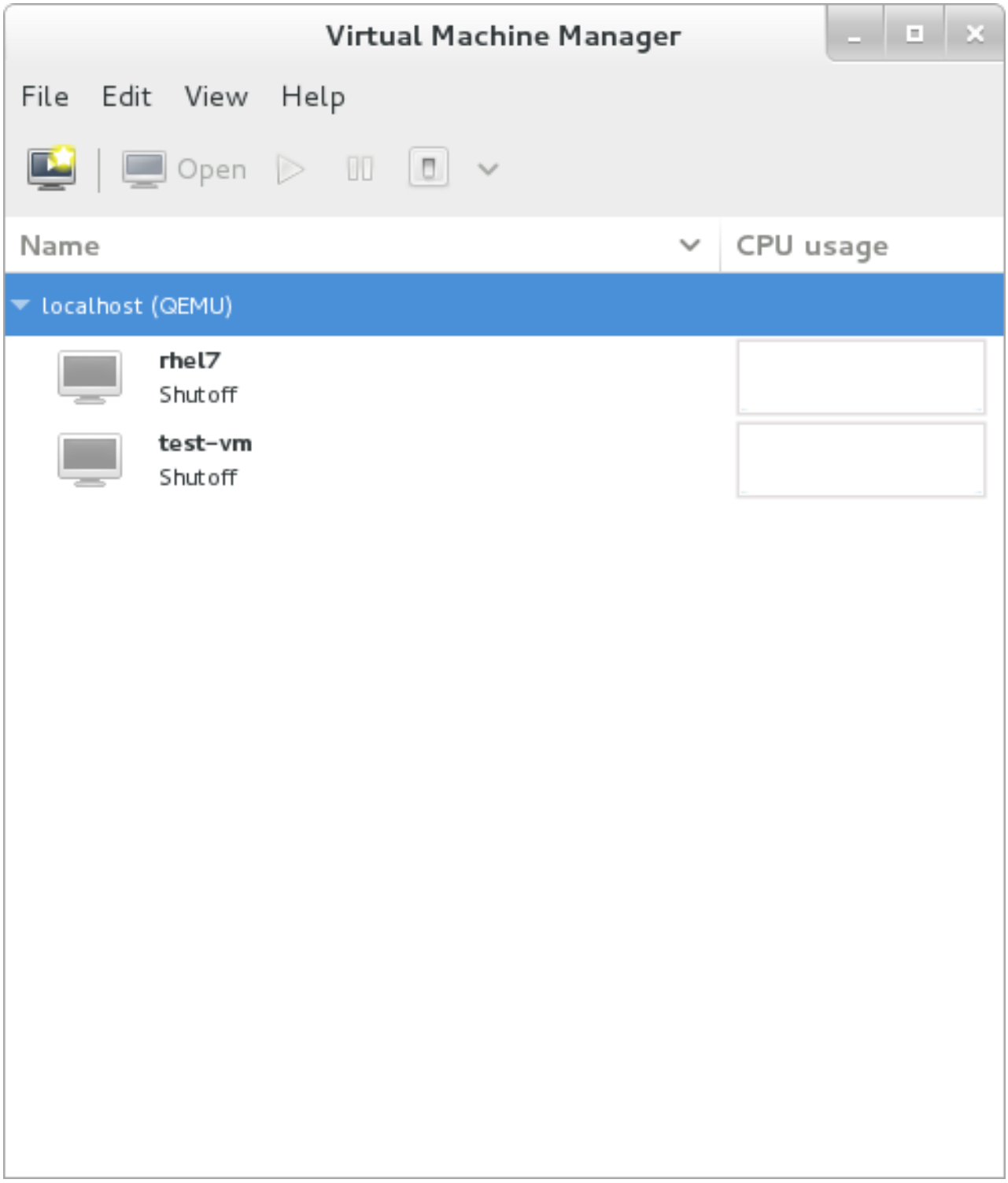


図25.1 `virt-manager` の起動

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。

25.2. 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。

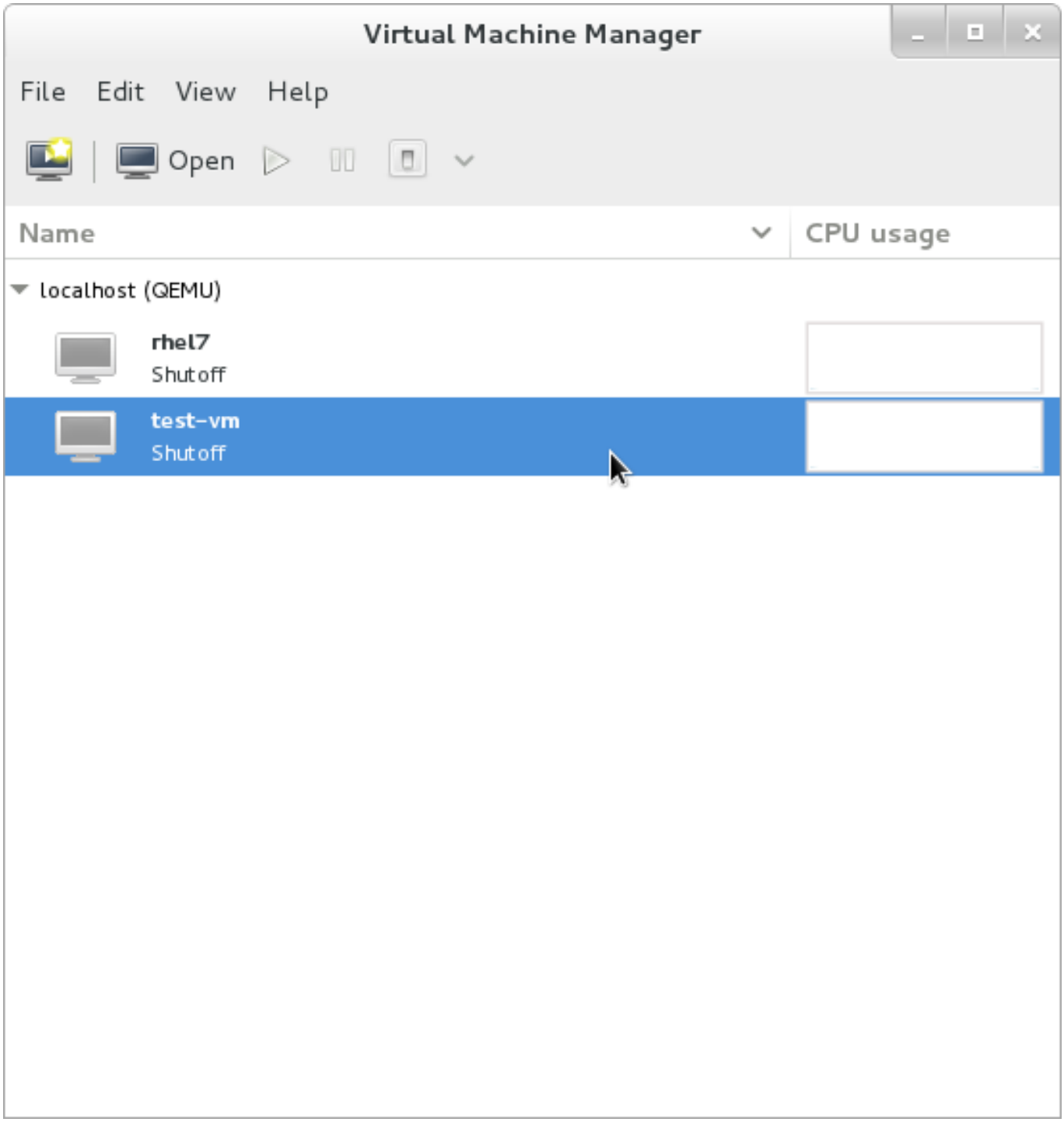


図25.2 仮想マシンマネージャーのメインウィンドウ

25.3. 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。

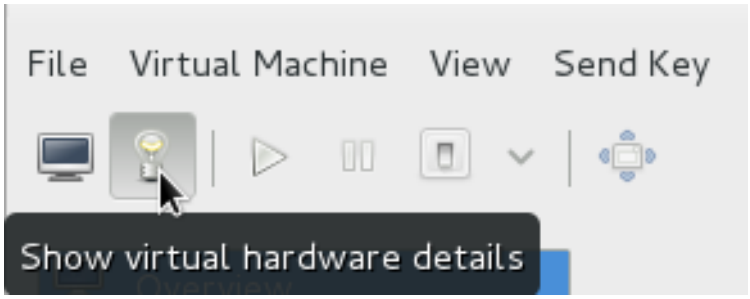


図25.3 仮想ハードウェアの詳細アイコン

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。

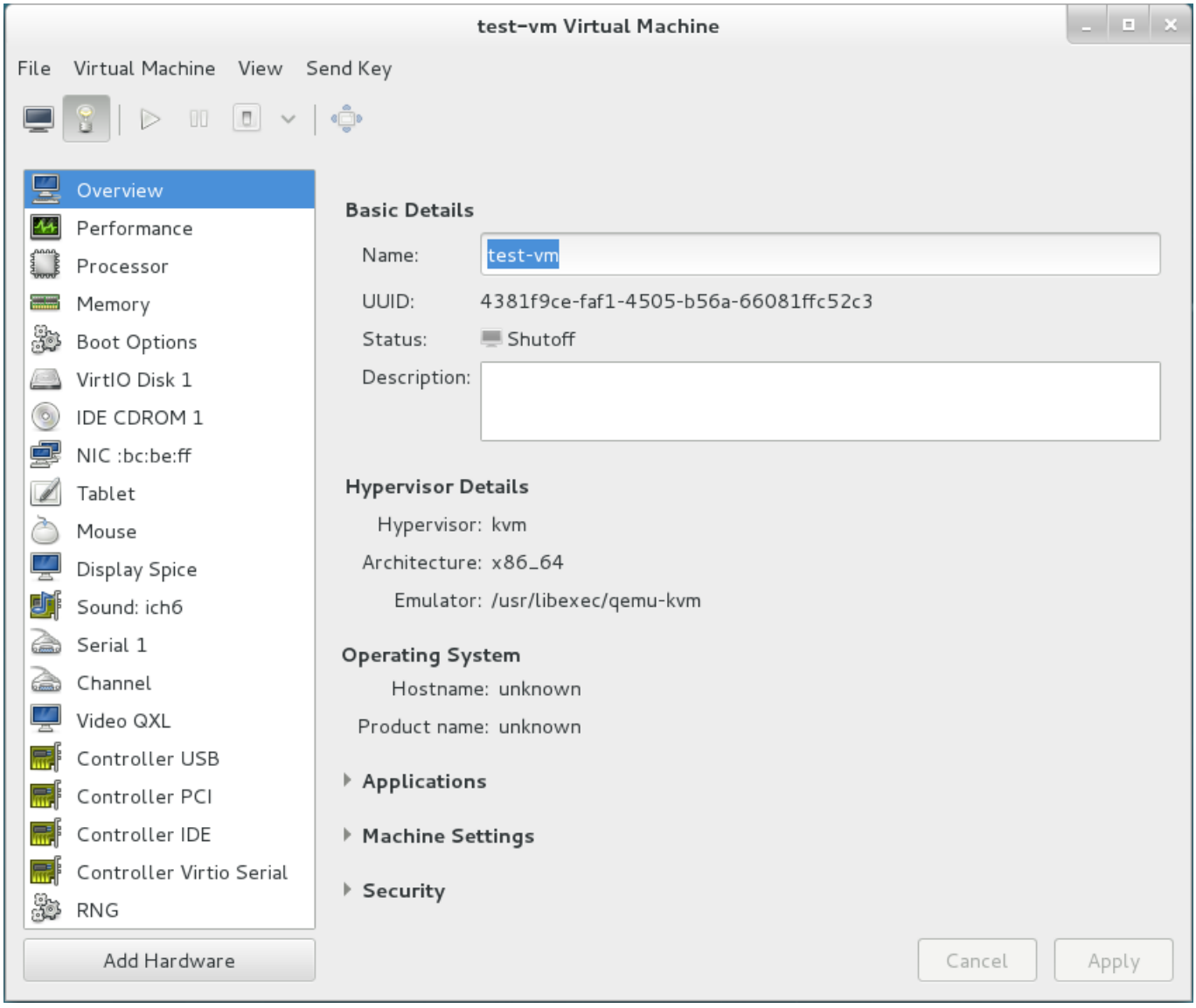


図25.4 仮想ハードウェアの詳細ウィンドウ

25.3.1. 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、次回のゲストの再起動時に変更が発生します。

手順25.1 起動オプションの設定

1. 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。
2. サイドパネルから **Boot Options** を選択してから、以下のオプションのステップのいずれかまたはすべてに入力します。
    a. このゲスト仮想マシンがホスト物理マシンの起動時に毎回機能することを示唆するには、**Autostart** チェックボックスを選択します。
    b. ゲスト仮想マシンが起動される順序を指定するには、**Enable boot menu** チェックボックスをクリックします。このチェックボックスにチェックが付けられていると、起動元になるデバイスを確認でき、矢印キーを使用して、起動時に使用されるゲスト仮想マシンの順序を変更できます。
    c. Linux カーネルから直接起動する場合は、**Direct kernel boot** メニューを拡張します。使用する **Kernel path**、**Initrd path**、および **Kernel arguments** に入力します。
3. **Apply** をクリックします。

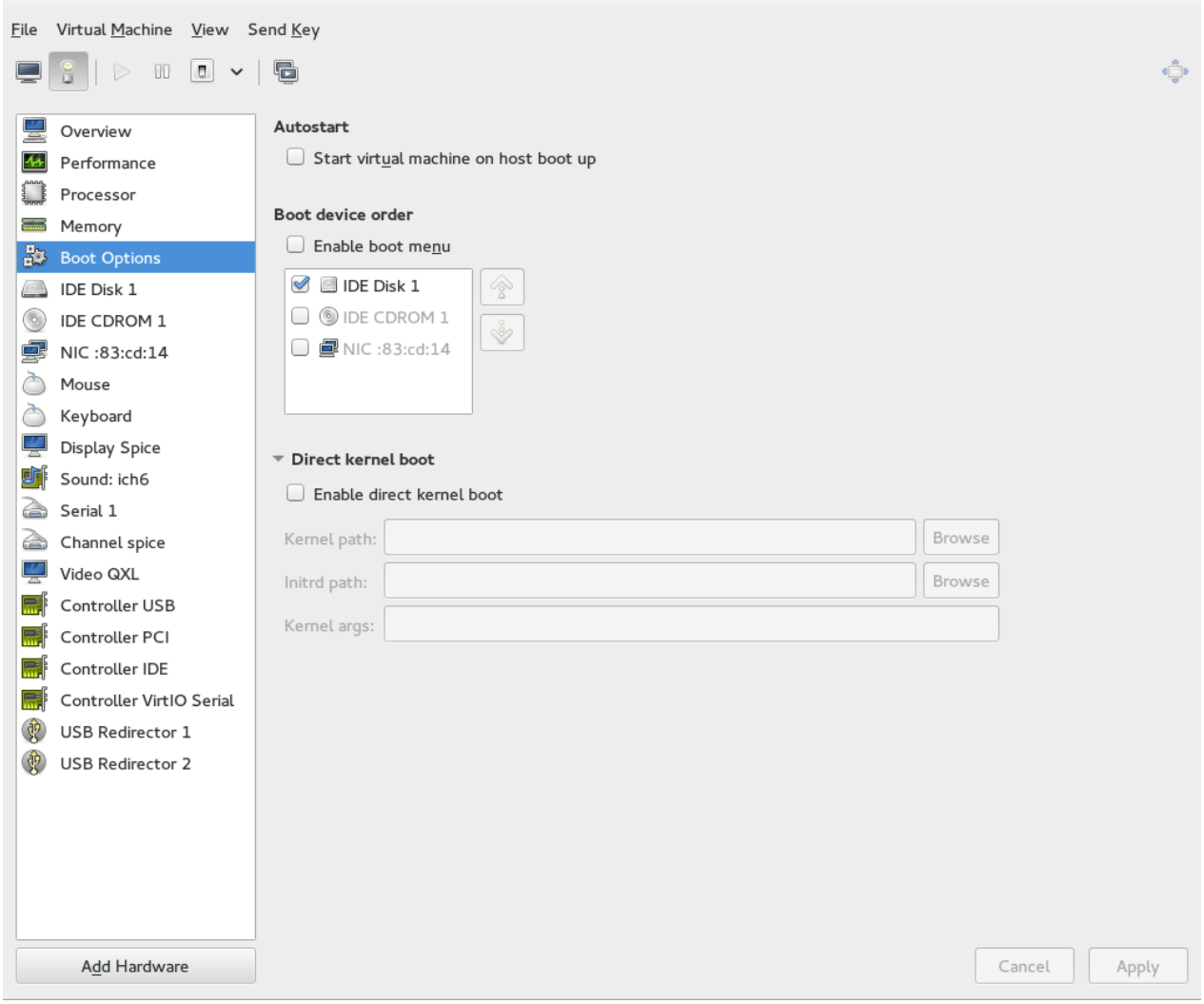


図25.5 起動オプションの設定

25.3.2. USB デバイスのゲスト仮想マシンへの割り当て

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25.2 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。

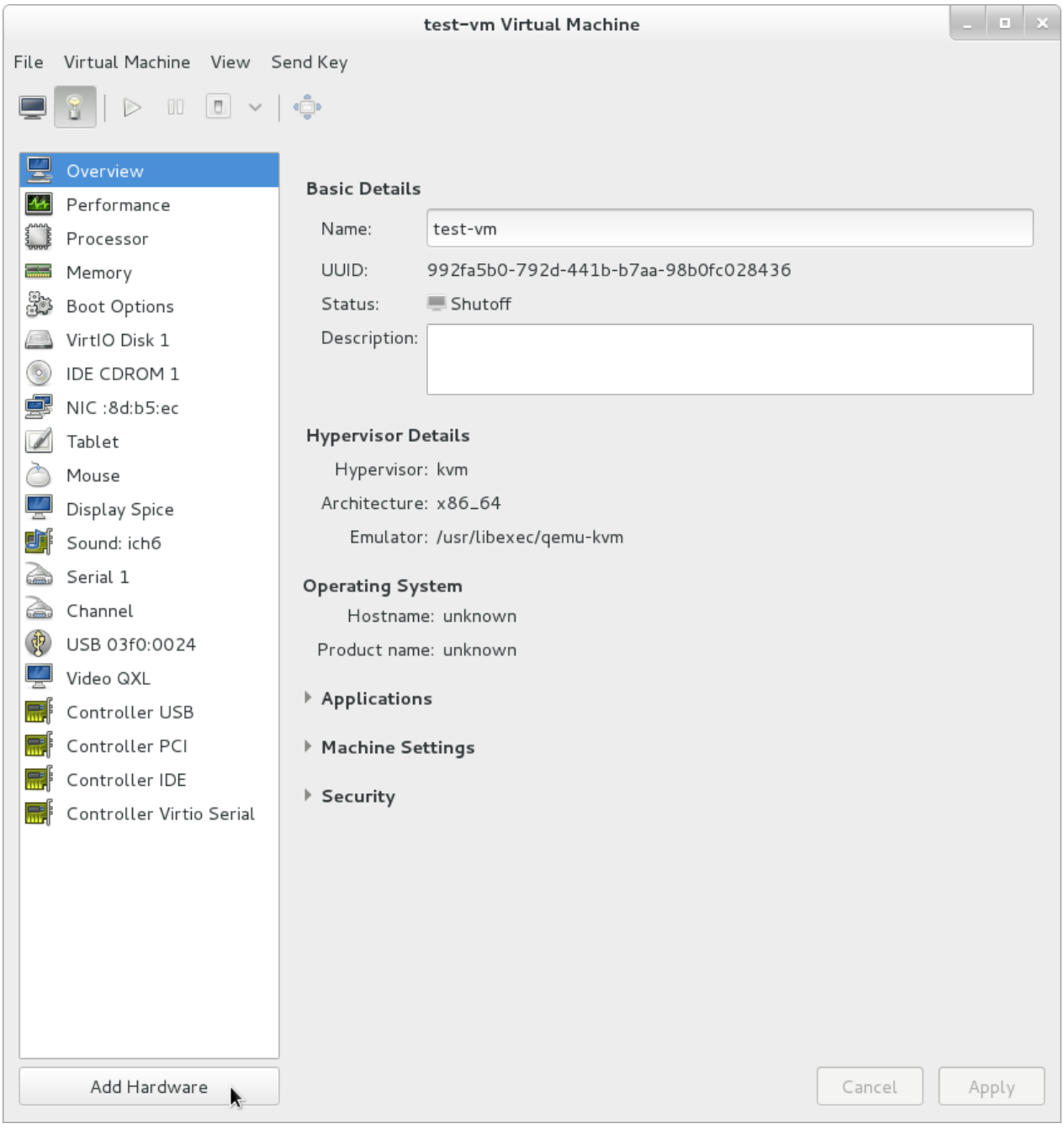


図25.6 ハードウェアを追加ボタン

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** をクリックします。

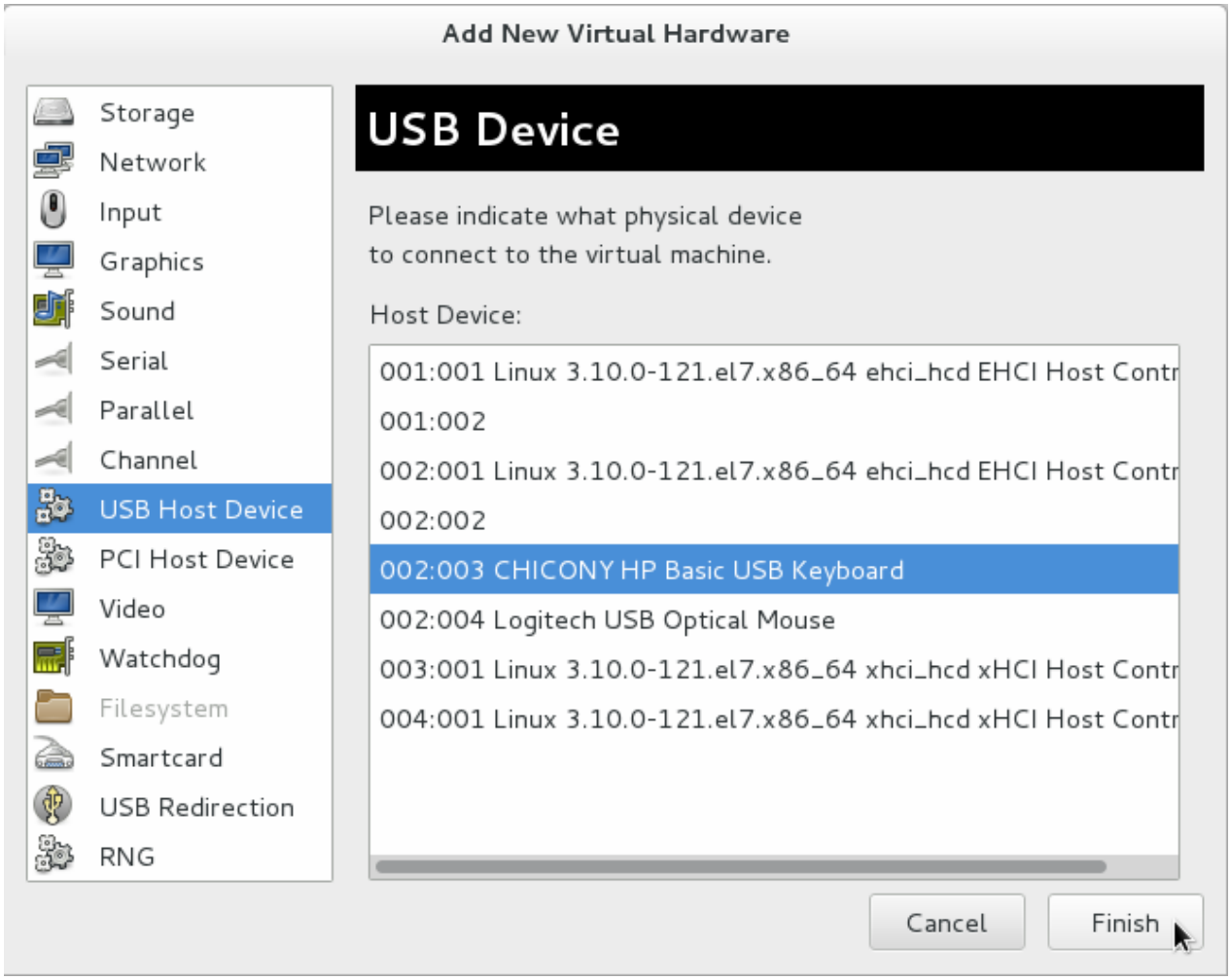


図25.7 USB デバイスの追加

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。

25.3.3. 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、これをシンクライアント上で実行されているゲスト仮想マシンで使用することができます。

手順25.3 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。

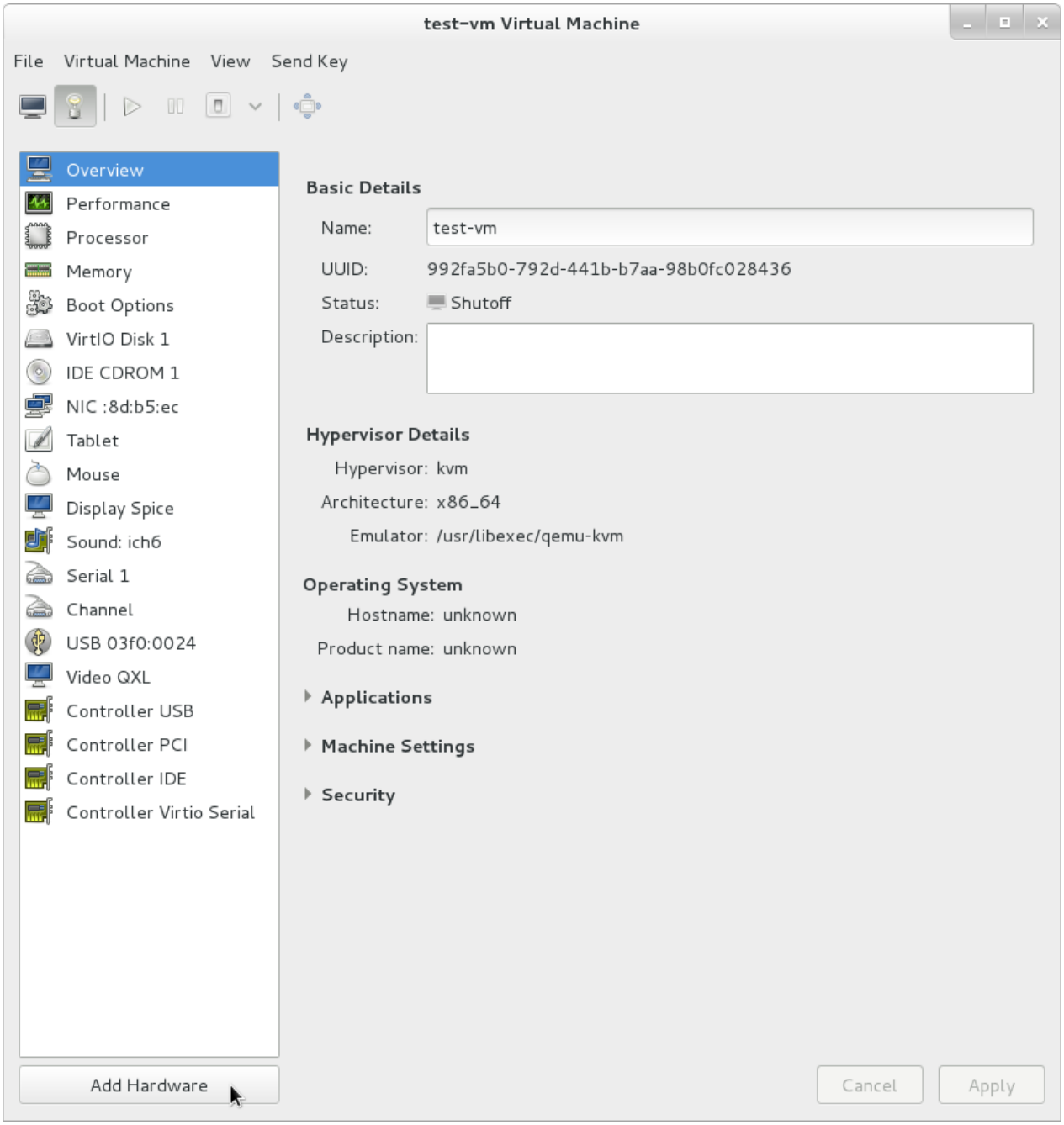


図25.8 ハードウェアを追加ボタン

3. **Add New Virtual Hardware** ポップアップで、**USB Redirection** を選択します。必ず **Spice channel Type** ドロップダウンメニューを選択し、**Finish** をクリックします。

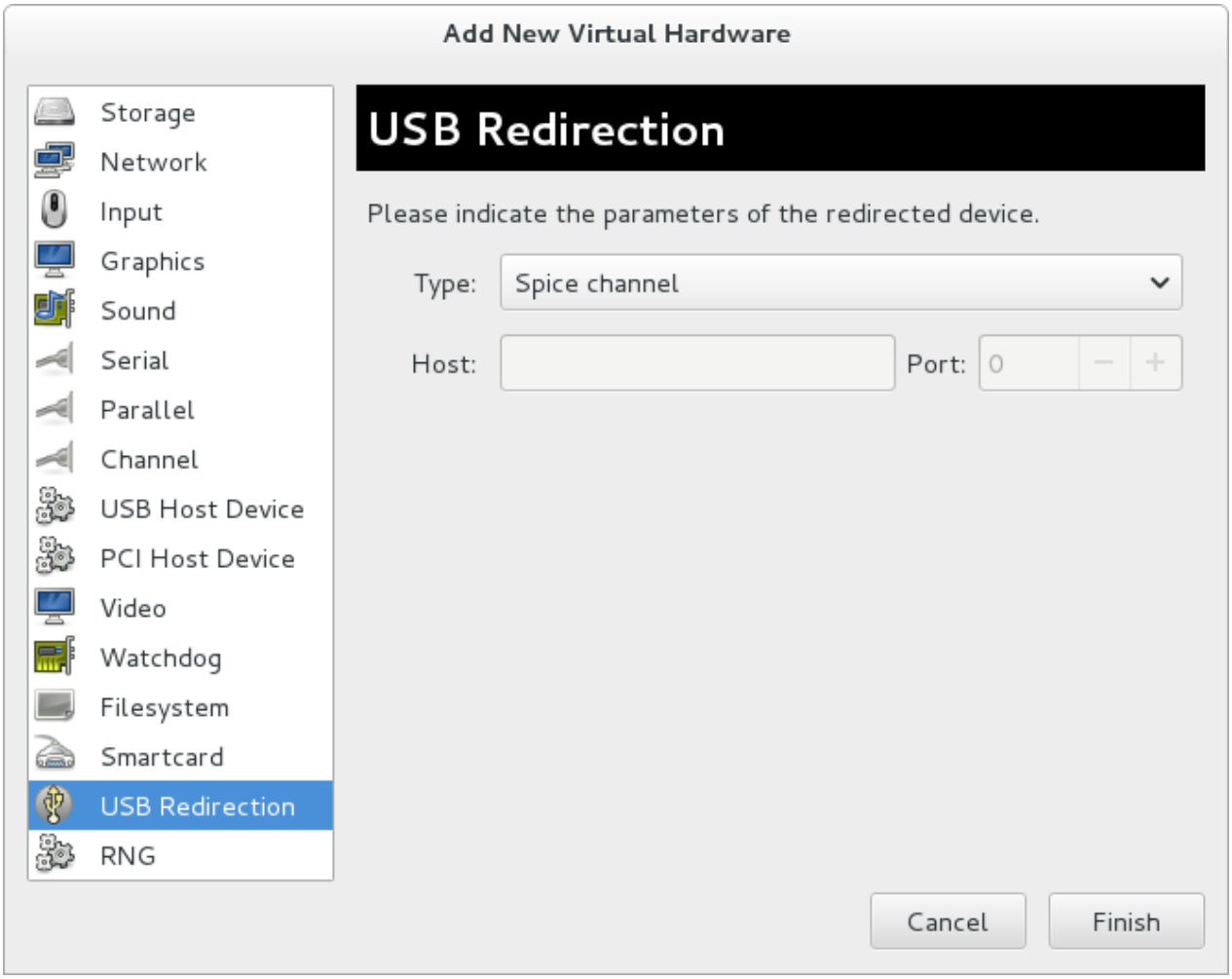


図25.9 新たな仮想ハードウェア追加ウィザード

4. 選択可能なデバイスの一覧と共にポップアップメニューが開きます。チェックボックスをクリックしてデバイスを選択し、**OK** をクリックします。

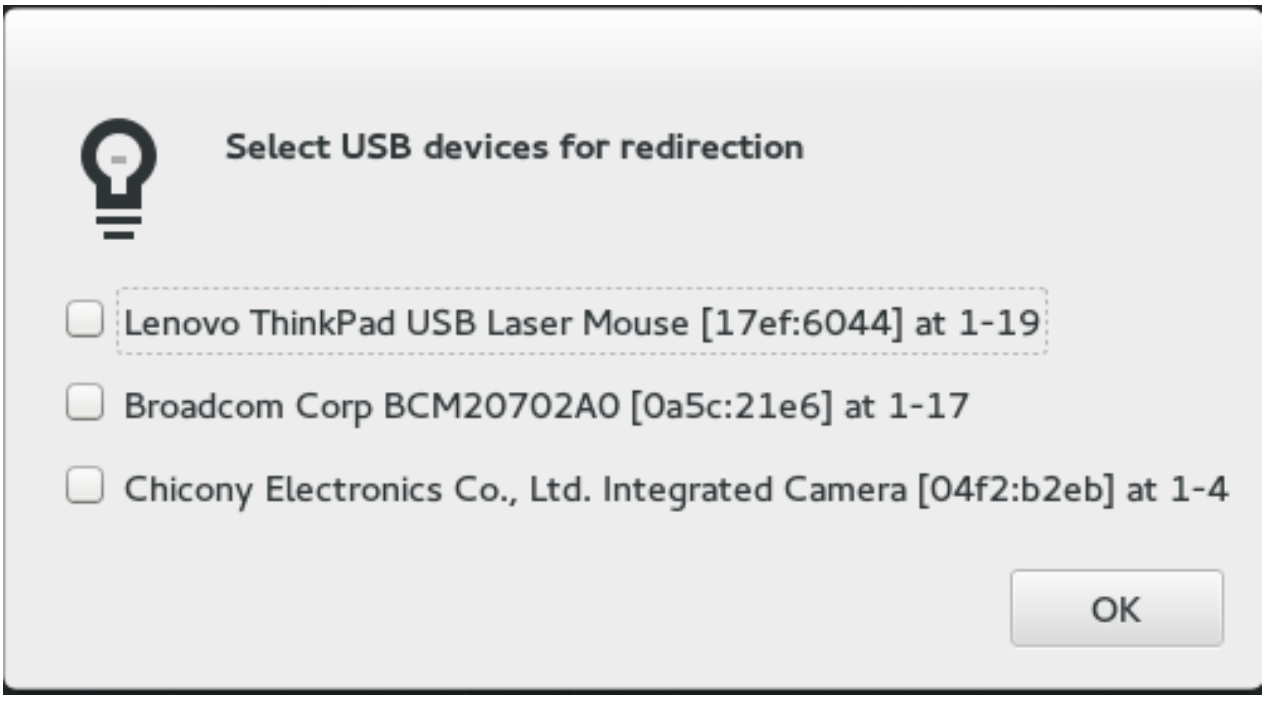


図25.10 USB デバイスの選択

25.4. 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** で対応しているのは **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。

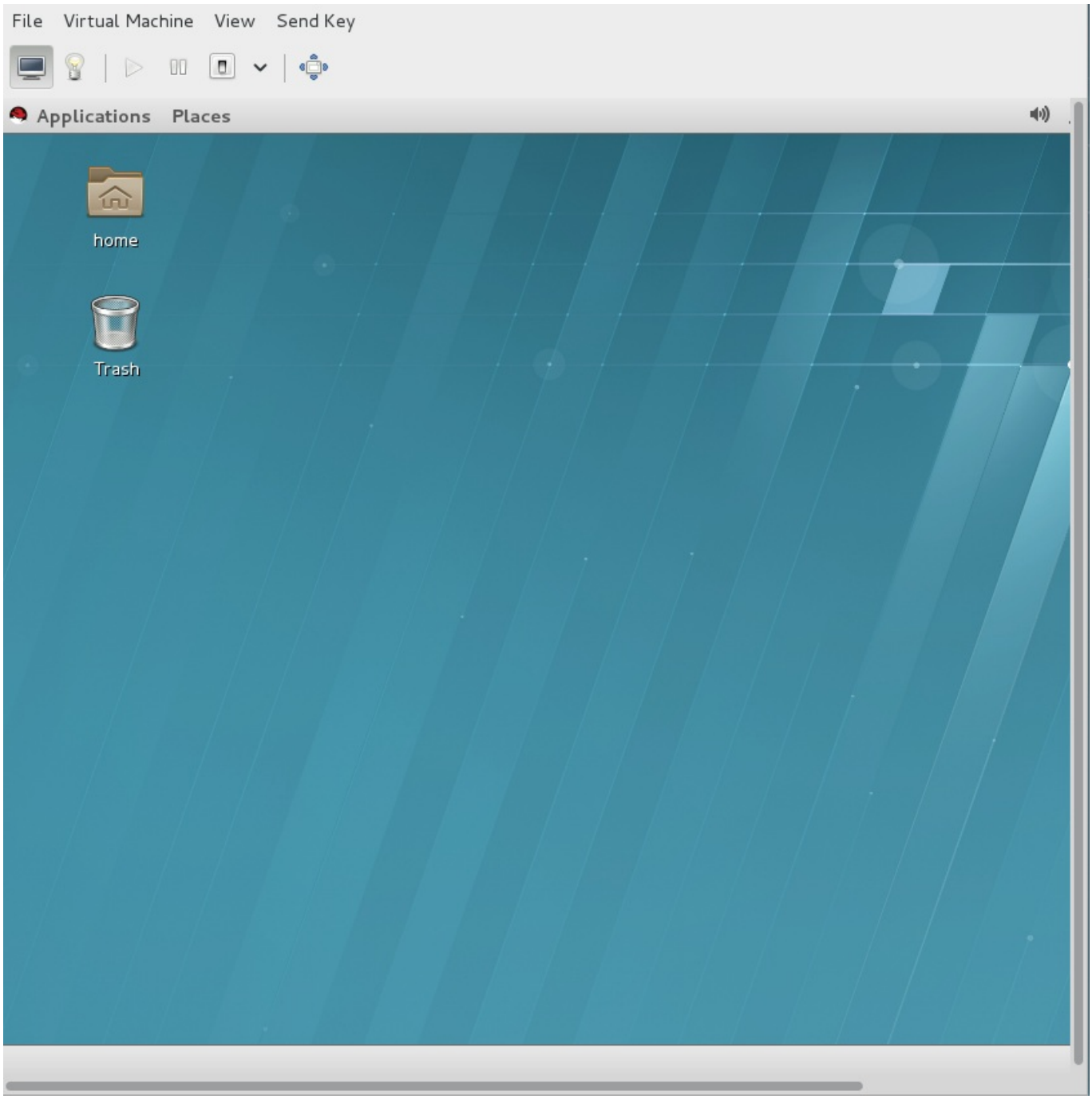


図25.11 グラフィカルのコンソールウィンドウ

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なわずに 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なう場合は、23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 が使用できます。

25.5. 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** の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。デフォルトのインストールで正常に動作する設定です。リモート接続の設定方法については、23章ゲストのリモート管理を参照してください。

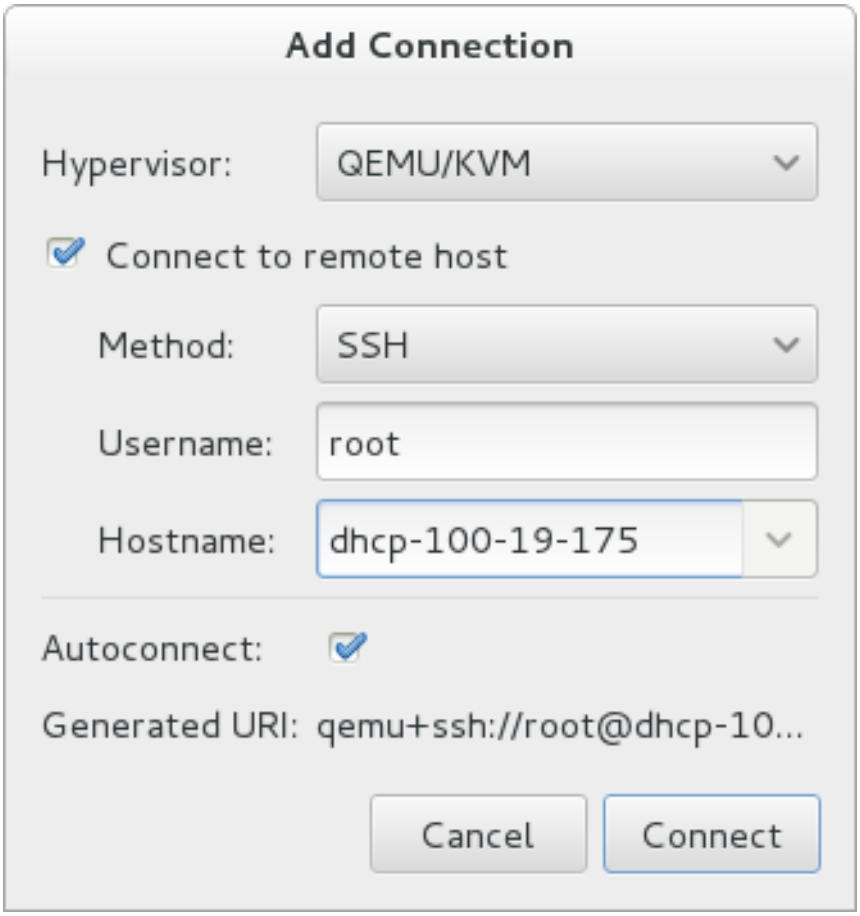


図25.12 接続の追加

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。

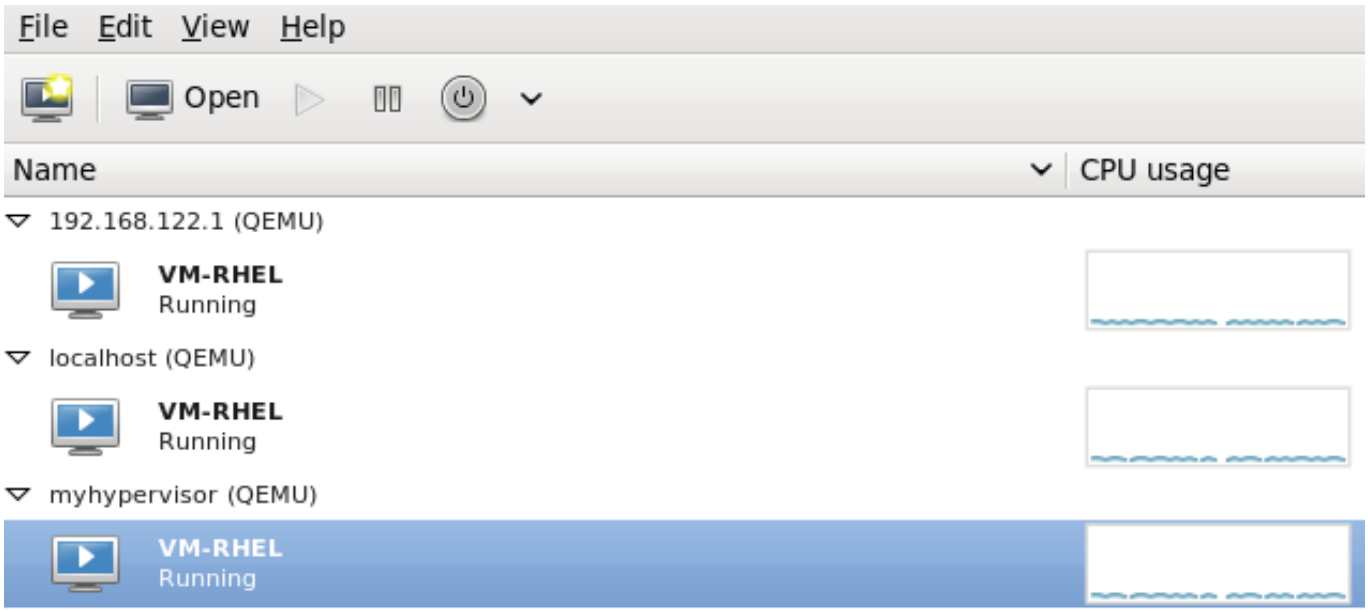


図25.13 virt-manager のメインウィンドウに表示されたリモートホスト

25.6. 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。

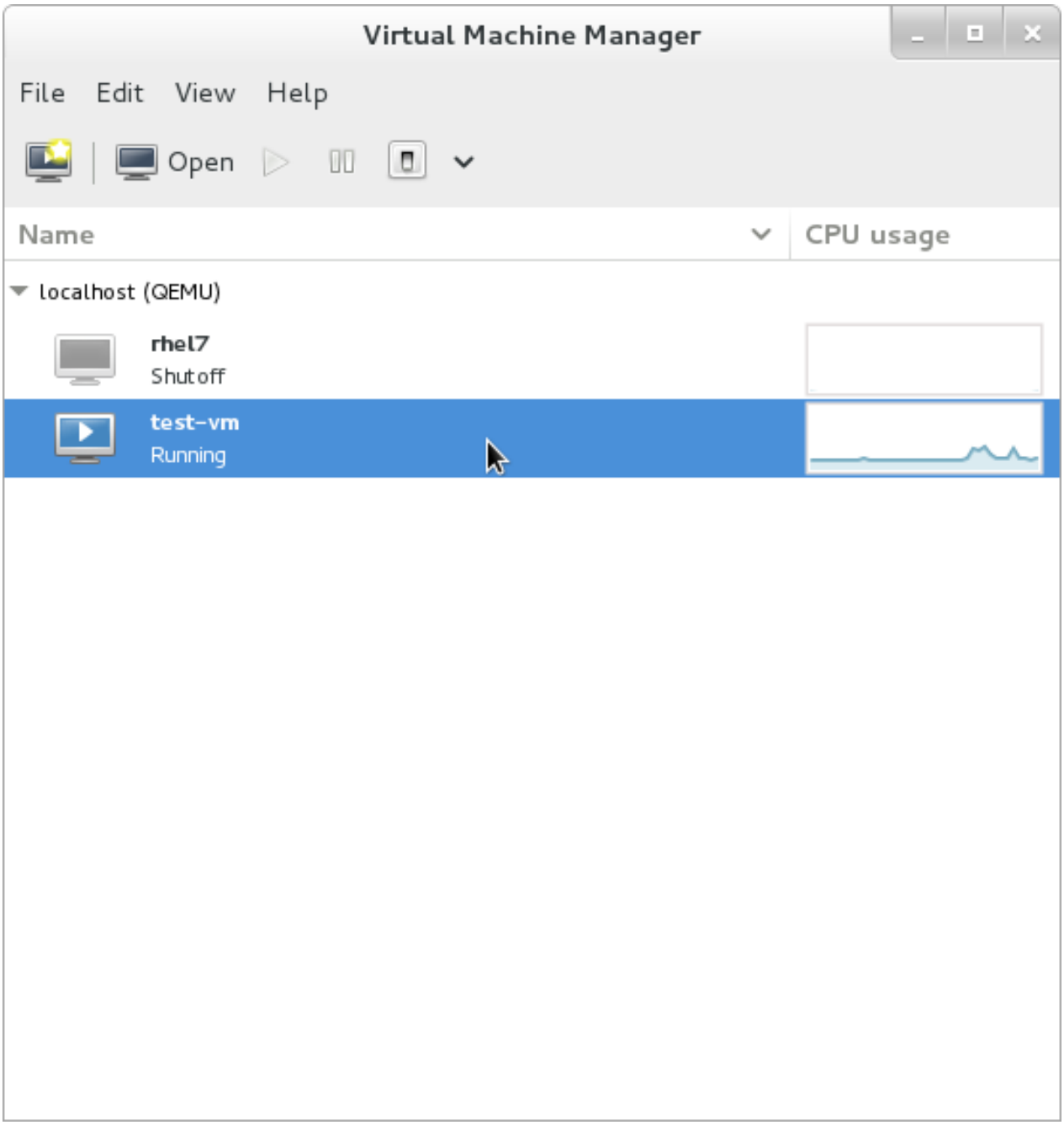


図25.14 表示する仮想マシンの選択

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。

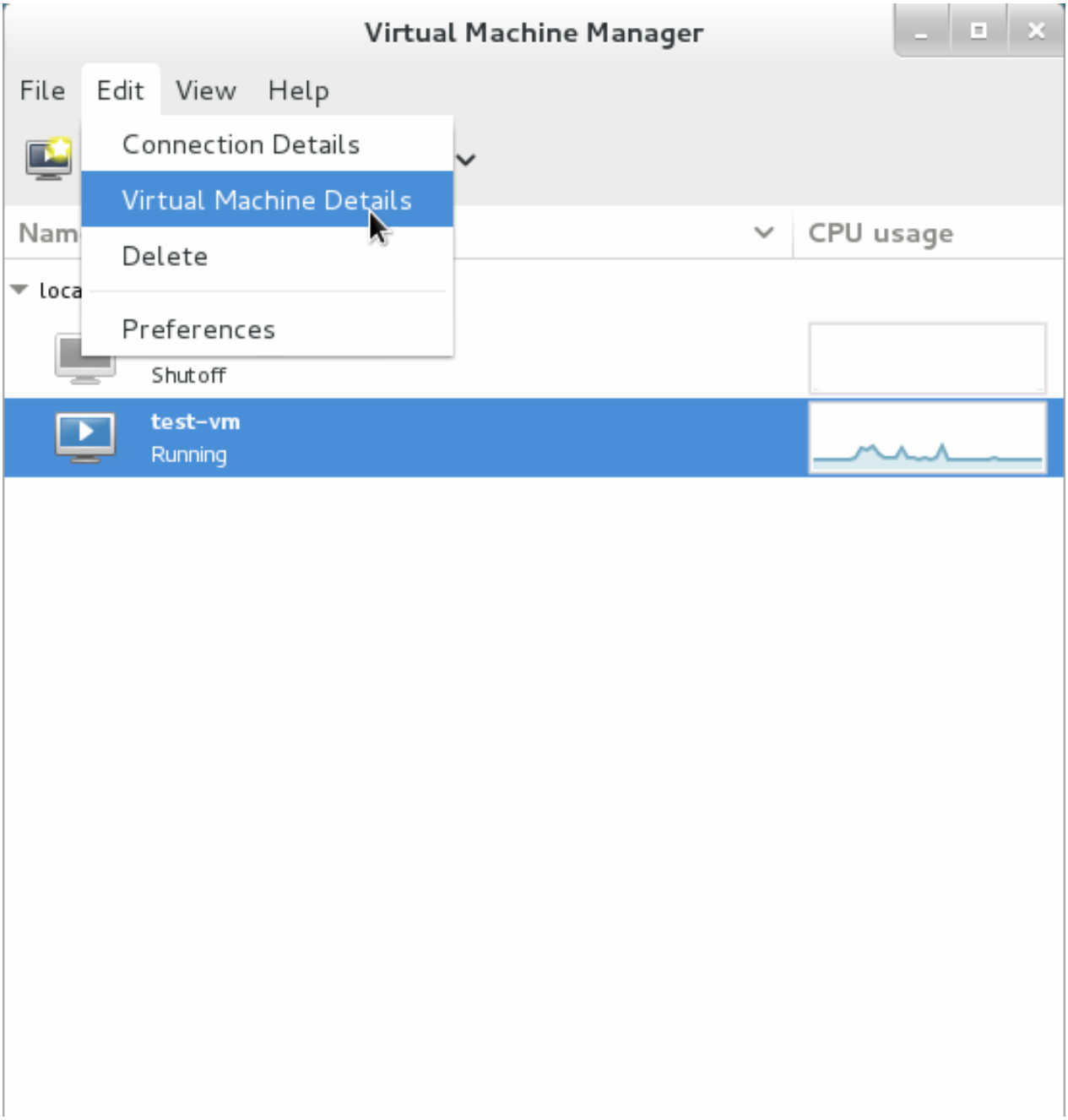


図25.15 仮想マシンの詳細の表示

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、そのゲストの構成情報の要約が表示されます。

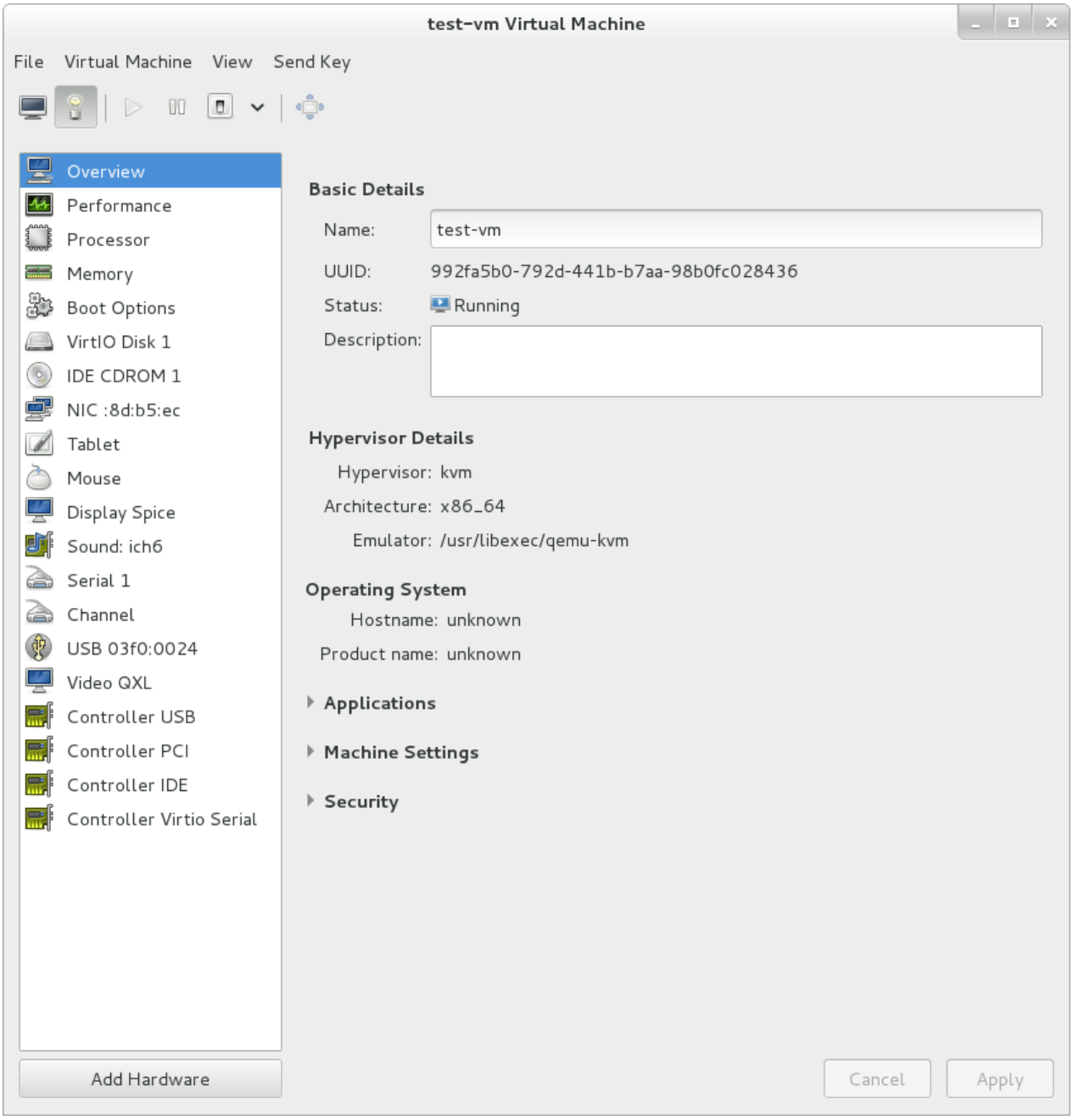


図25.16 ゲスト詳細の概要の表示

3. 左側のナビゲーションペインから **パフォーマンス (Performance)** を選択します。

   パフォーマンス (Performance) のビューでは、CPU やメモリー使用率などゲストのパフォーマンスに関する要約が表示されます。

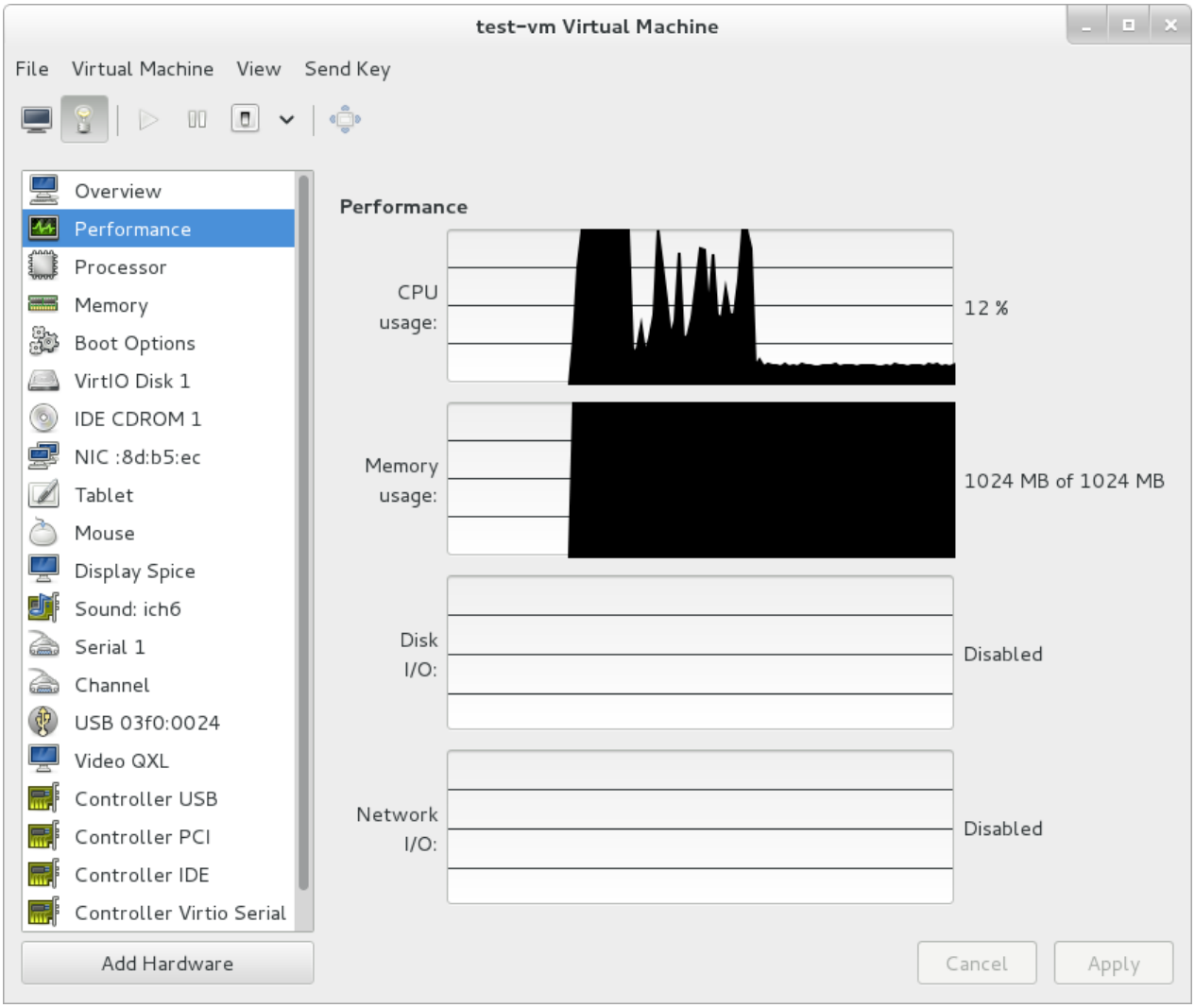


図25.17 ゲストのパフォーマンス詳細の表示

4. 左側のナビゲーションペインから **プロセッサー (Processor)** を選択します。**プロセッサー (Processor)** のビューでは、現在のプロセッサ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。

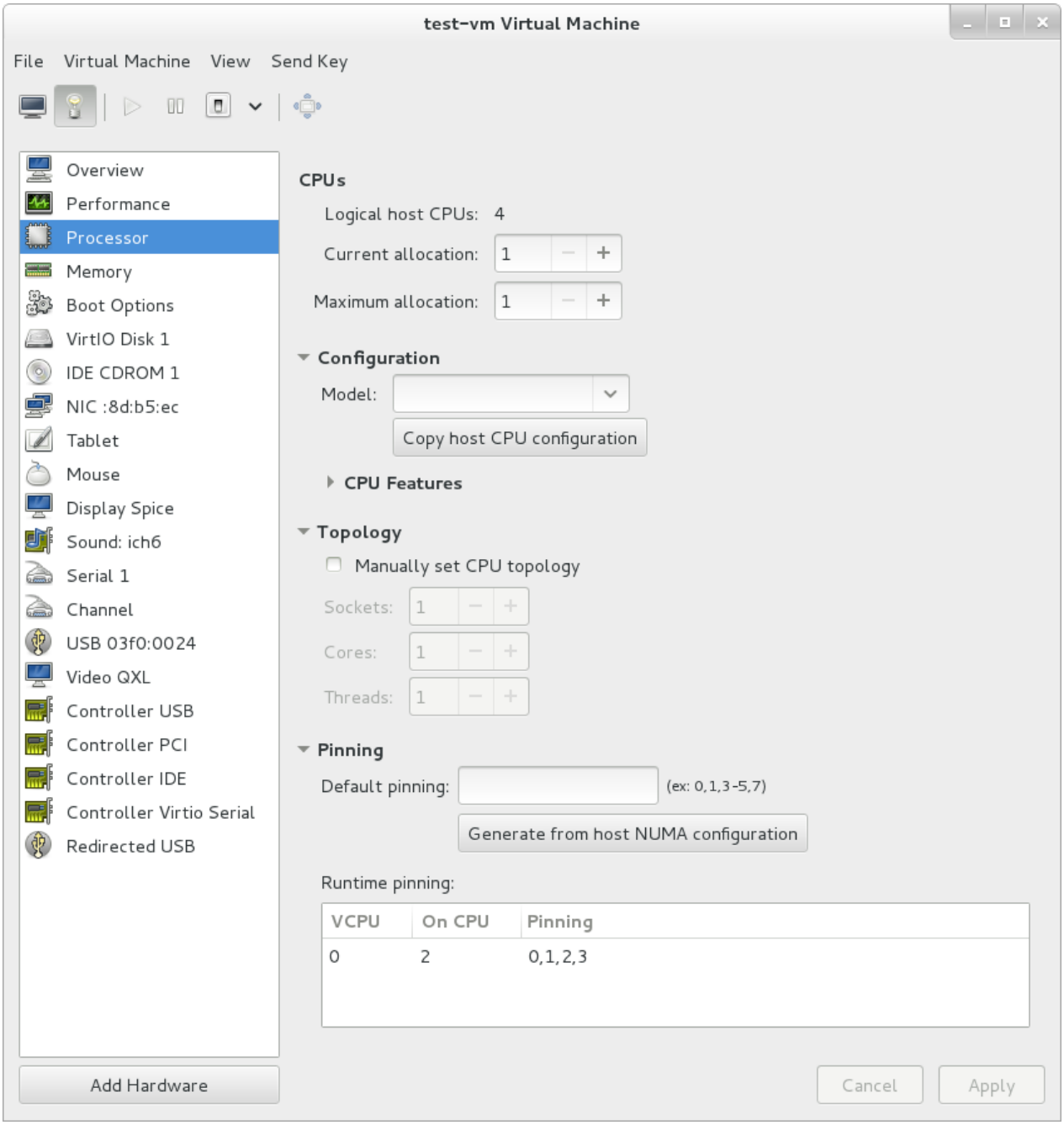


図25.18 プロセッサーの割り当てパネル

5. 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。

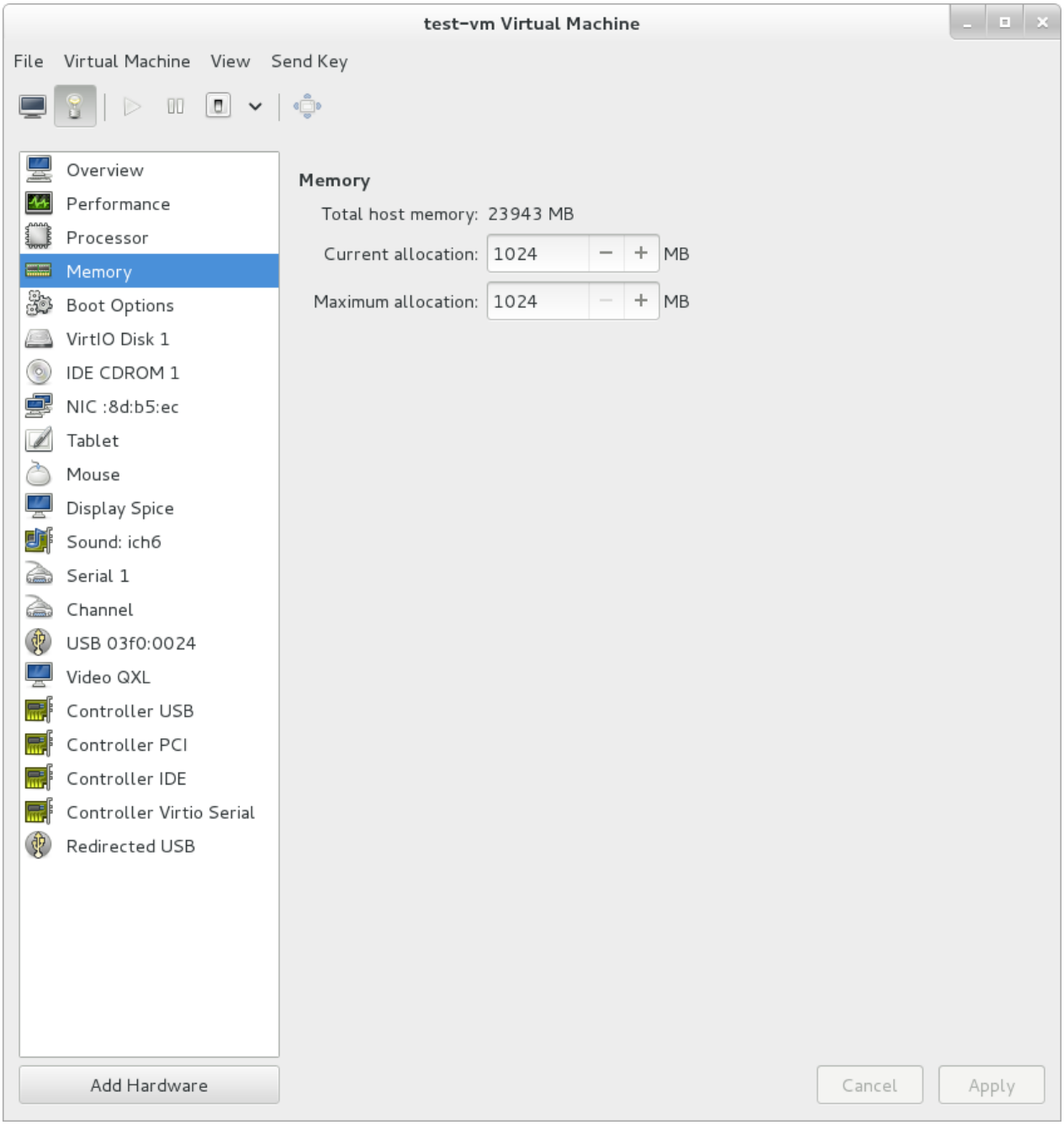


図25.19 メモリー割り当ての表示

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。

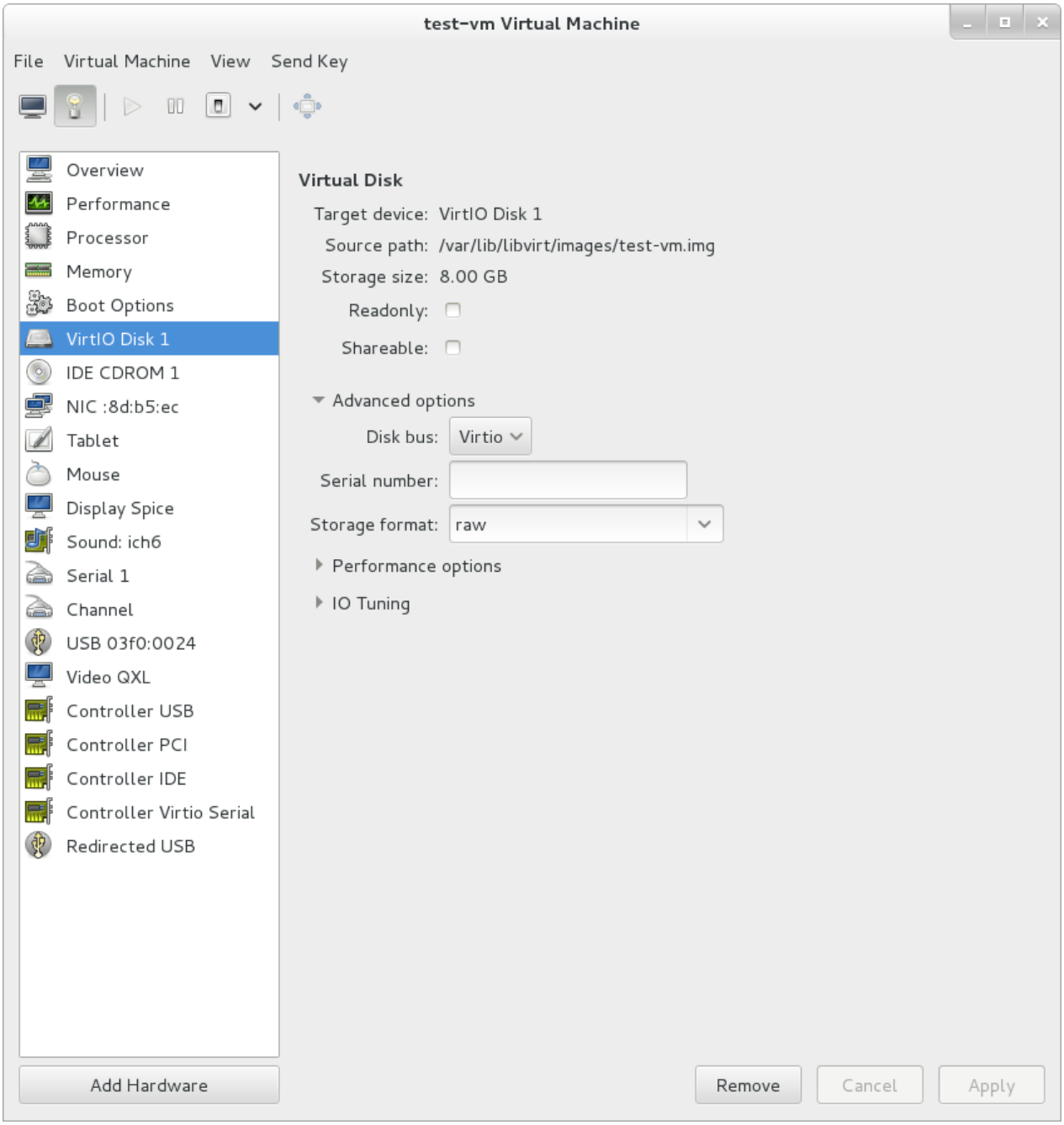


図25.20 ディスク設定の表示

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。

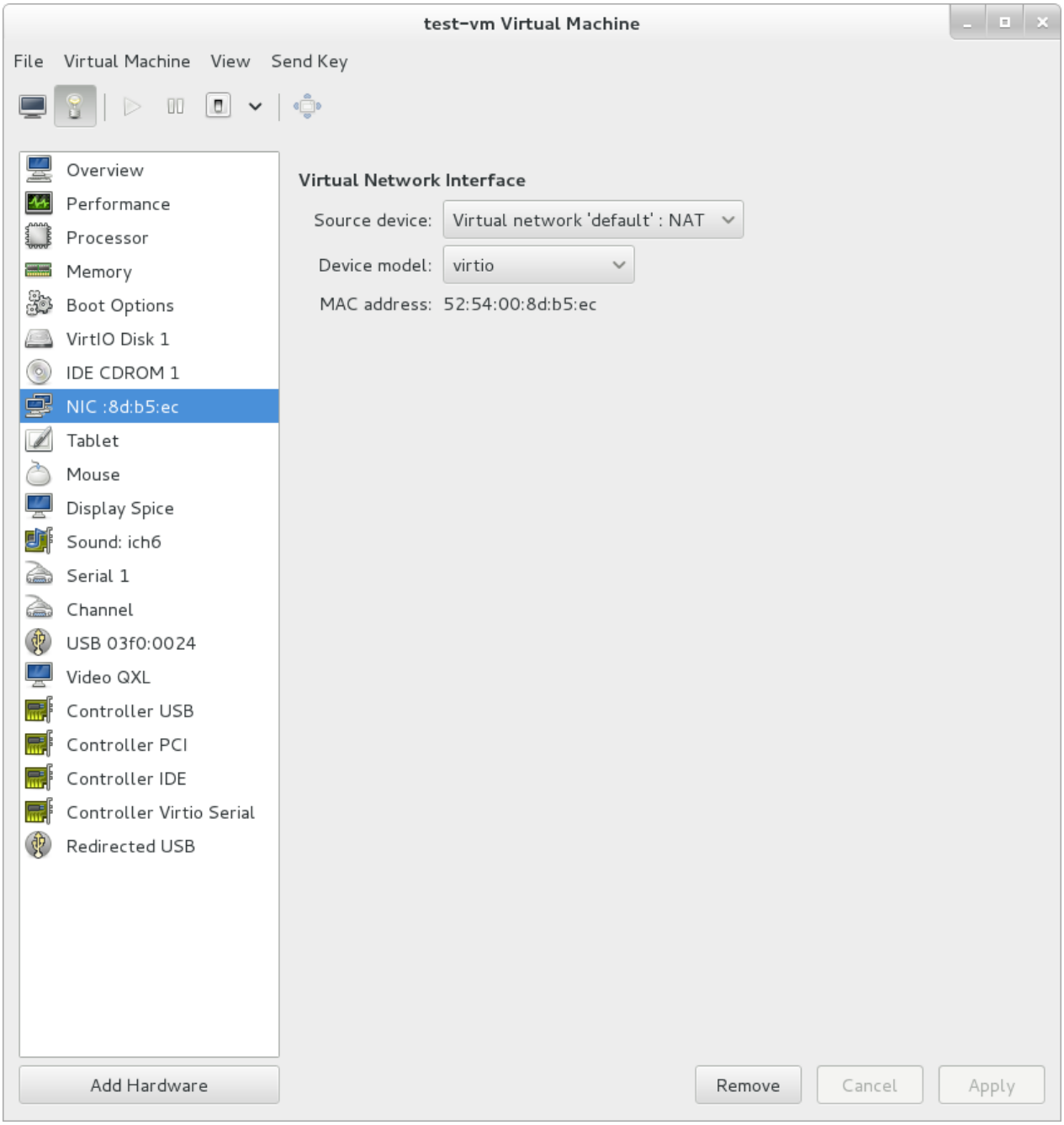


図25.21 ネットワーク設定の表示

25.7. パフォーマンスの監視

パフォーマンスの監視設定は、**`virt-manager`** の設定ウィンドウで修正することができます。

パフォーマンス監視の設定

1. **編集** メニューから **設定** を選択します。

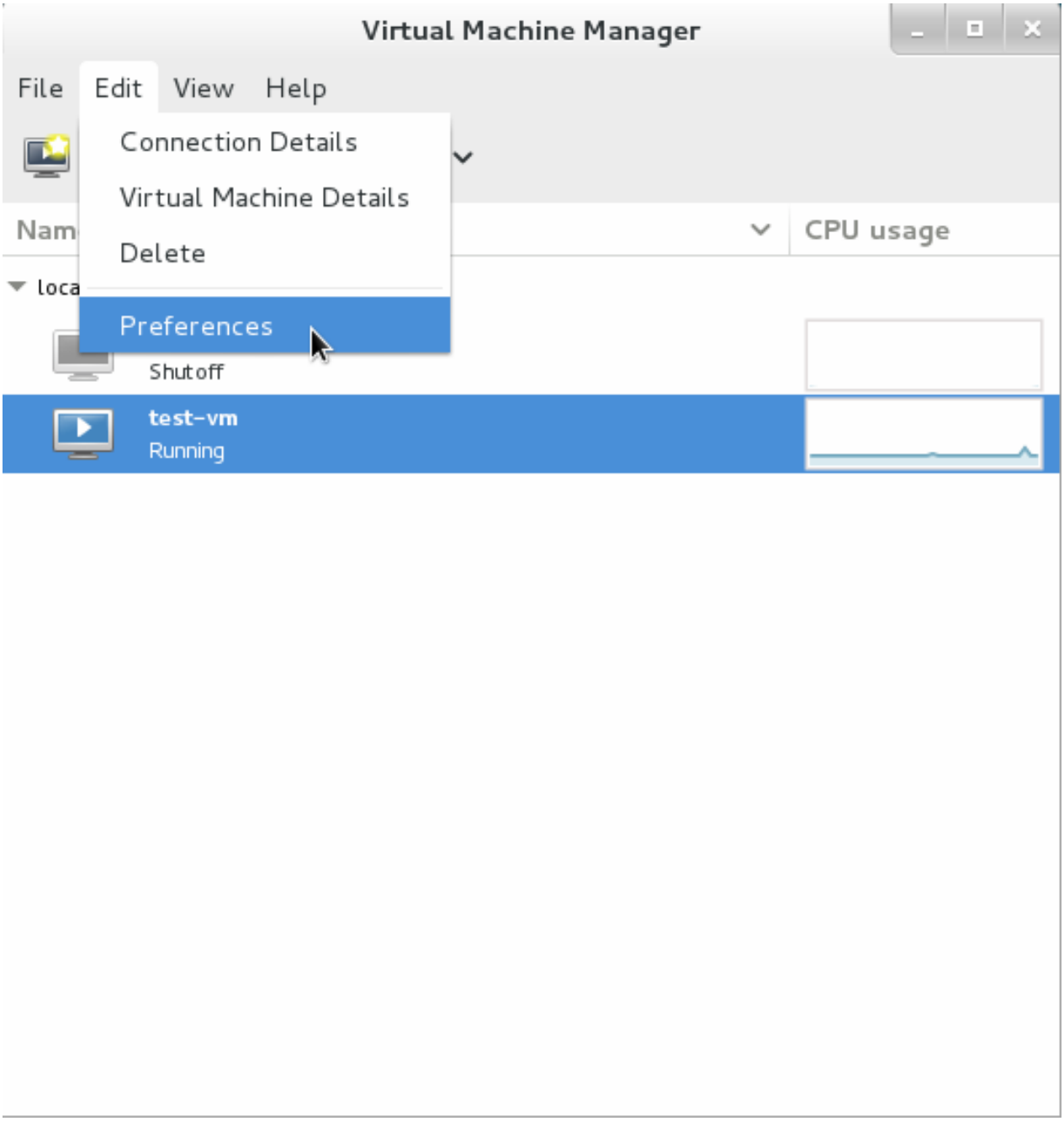


図25.22 ゲスト設定の変更

設定 ウィンドウが表示されます。

2. **統計** のタブで、更新の間隔を秒単位で指定するか、または統計監視オプションを有効にします。

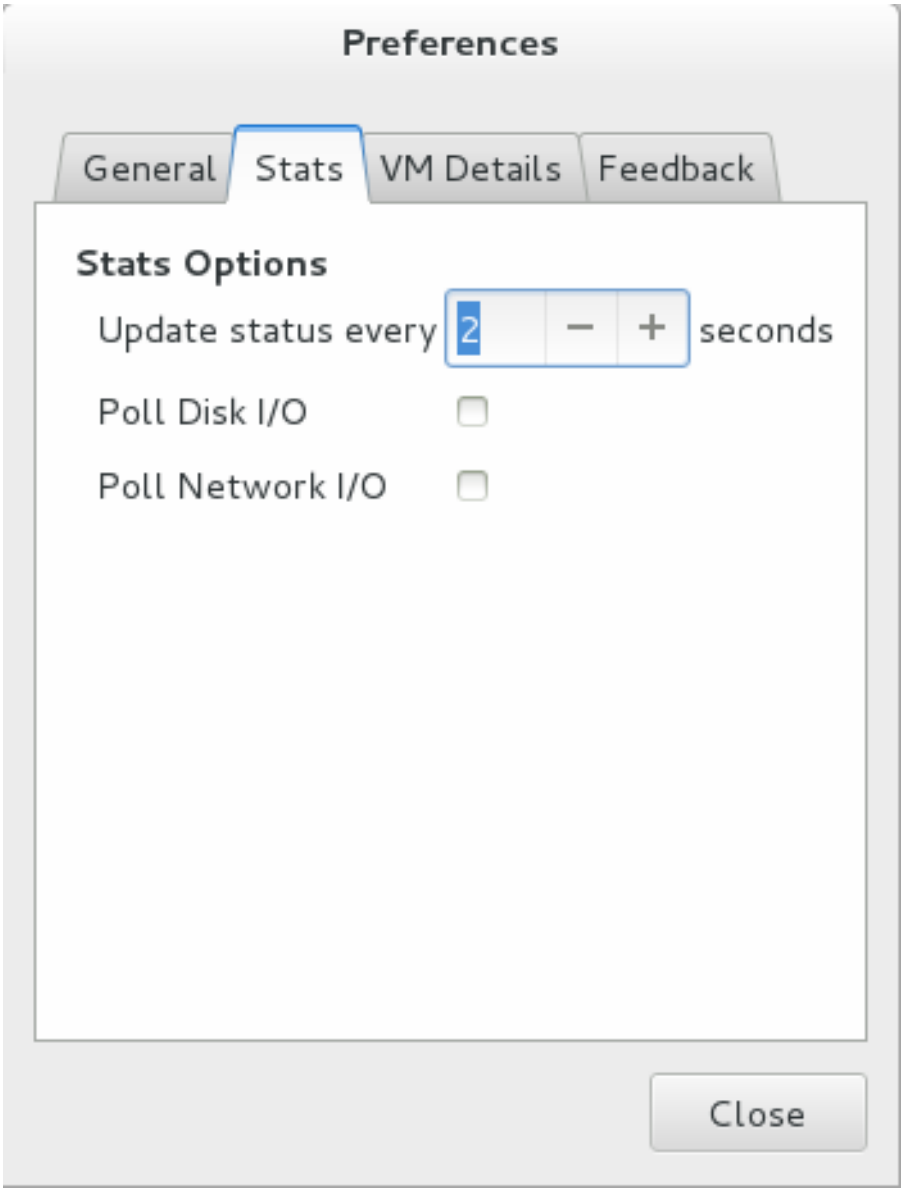


図25.23 パフォーマンス監視の設定

25.8. ゲストの CPU 使用率の表示

システム上の全ゲストの CPU 使用率を表示する

1. **表示** メニューから **グラフ** を選択し、**ゲストの CPU 使用率 (Guest CPU Usage)** のチェックボックスに印を付けます。

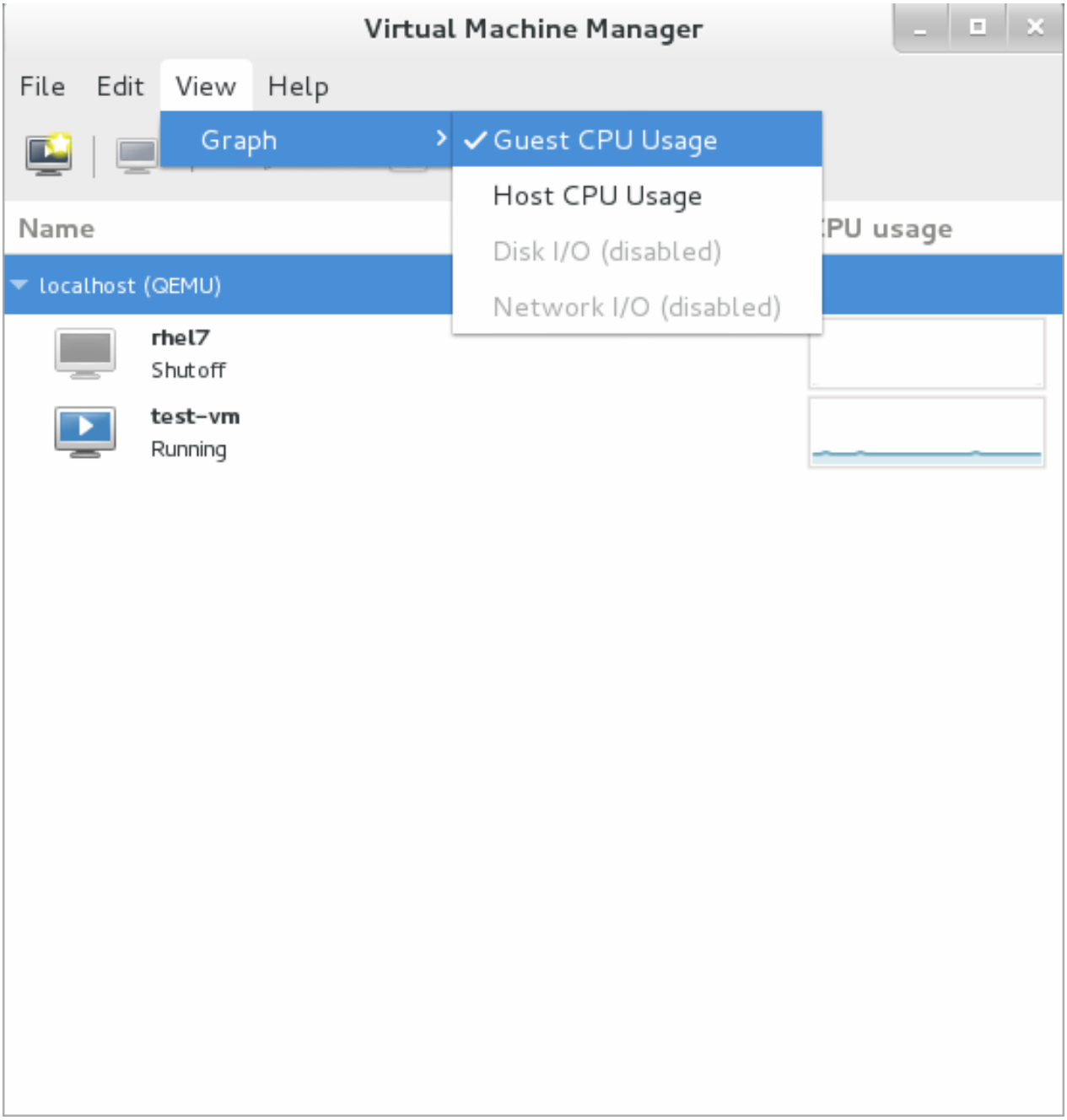


図25.24 ゲストの CPU 使用率の統計値グラフを有効化

2. 仮想マシンマネージャーがシステム上の全仮想マシンの CPU 使用率グラフを表示します。

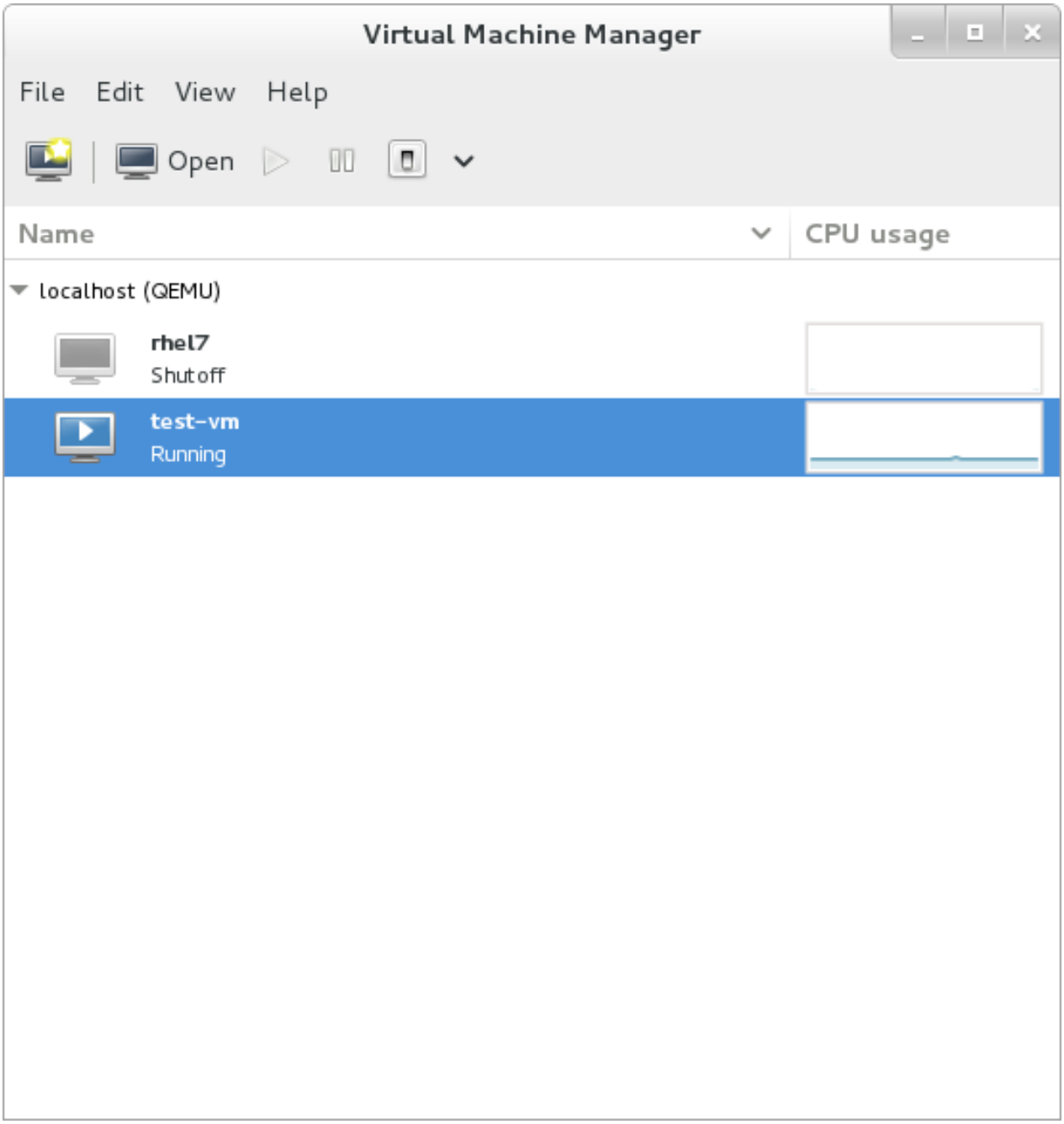


図25.25 ゲストの CPU 使用率グラフ

25.9. ホストの CPU 使用率の表示

システム上の全ホストの CPU 使用率を表示する

1. **表示** メニューから **グラフ** を選択し、**ホストの CPU 使用率 (host CPU Usage)** のチェックボックスに印を付けます。

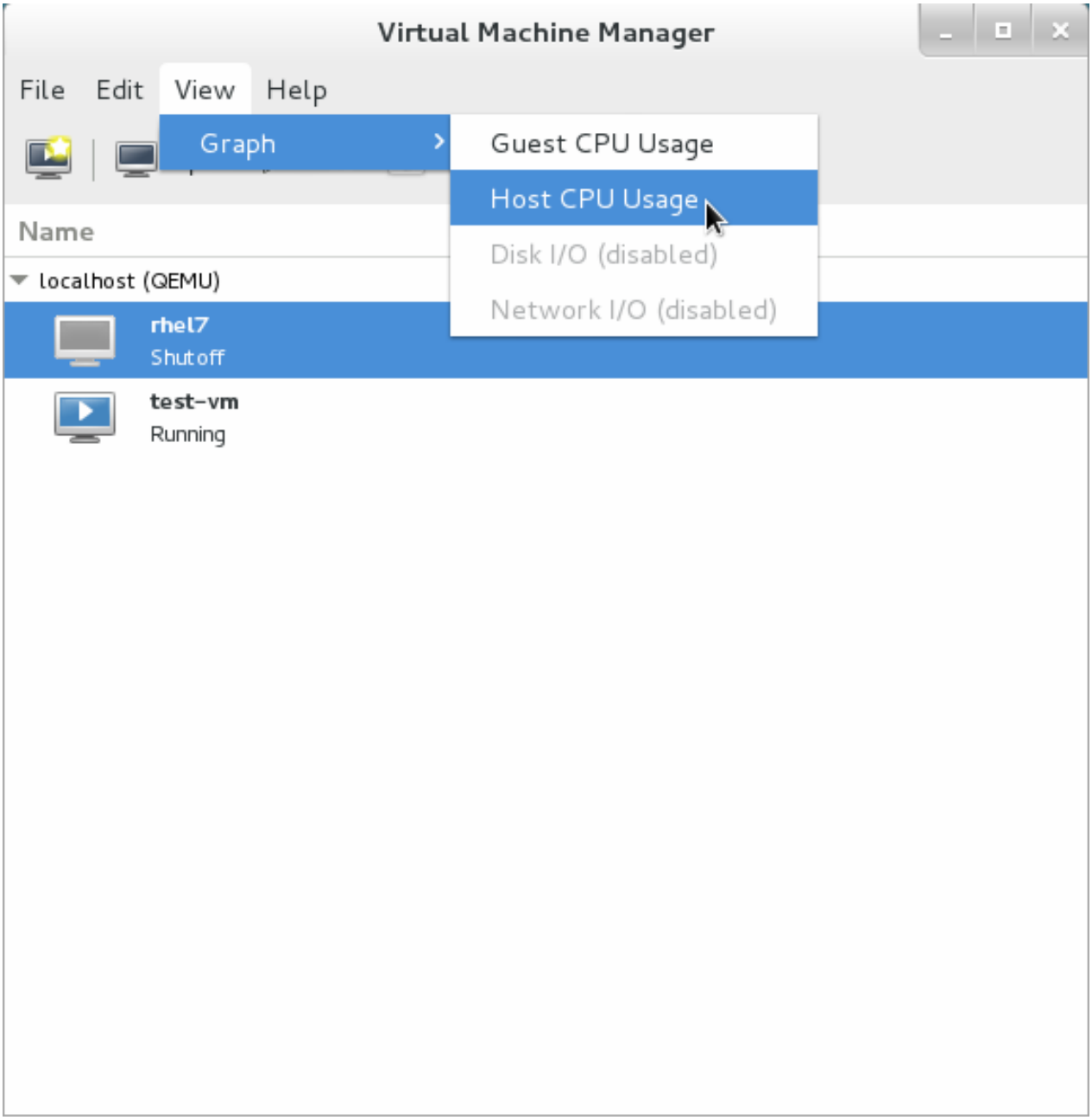


図25.26 ホストの CPU 使用率の統計値グラフの有効化

2. 仮想マシンマネージャーがシステム上のホストの CPU 使用率グラフを表示します。

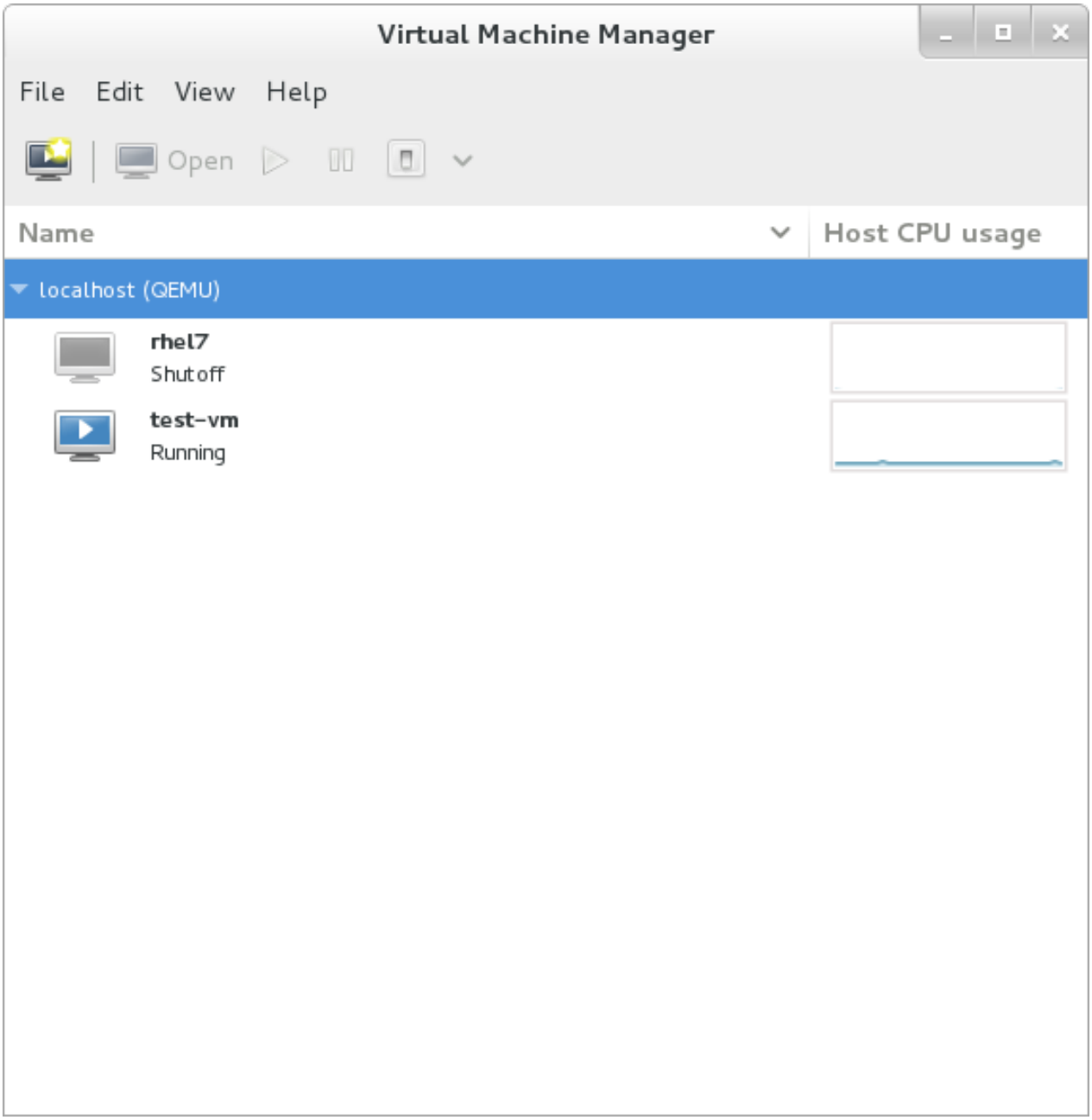


図25.27 ホストの CPU 使用率グラフ

25.10. ディスク I/O の表示

システム上の全仮想マシンのディスク I/O を表示する

1. ディスク I/O の統計値収集の設定が有効になっていることを確認します。**編集** メニューから **設定** を選択します。**統計** タブをクリックします。
2. **ディスク I/O** のチェックボックスに印を付けます。

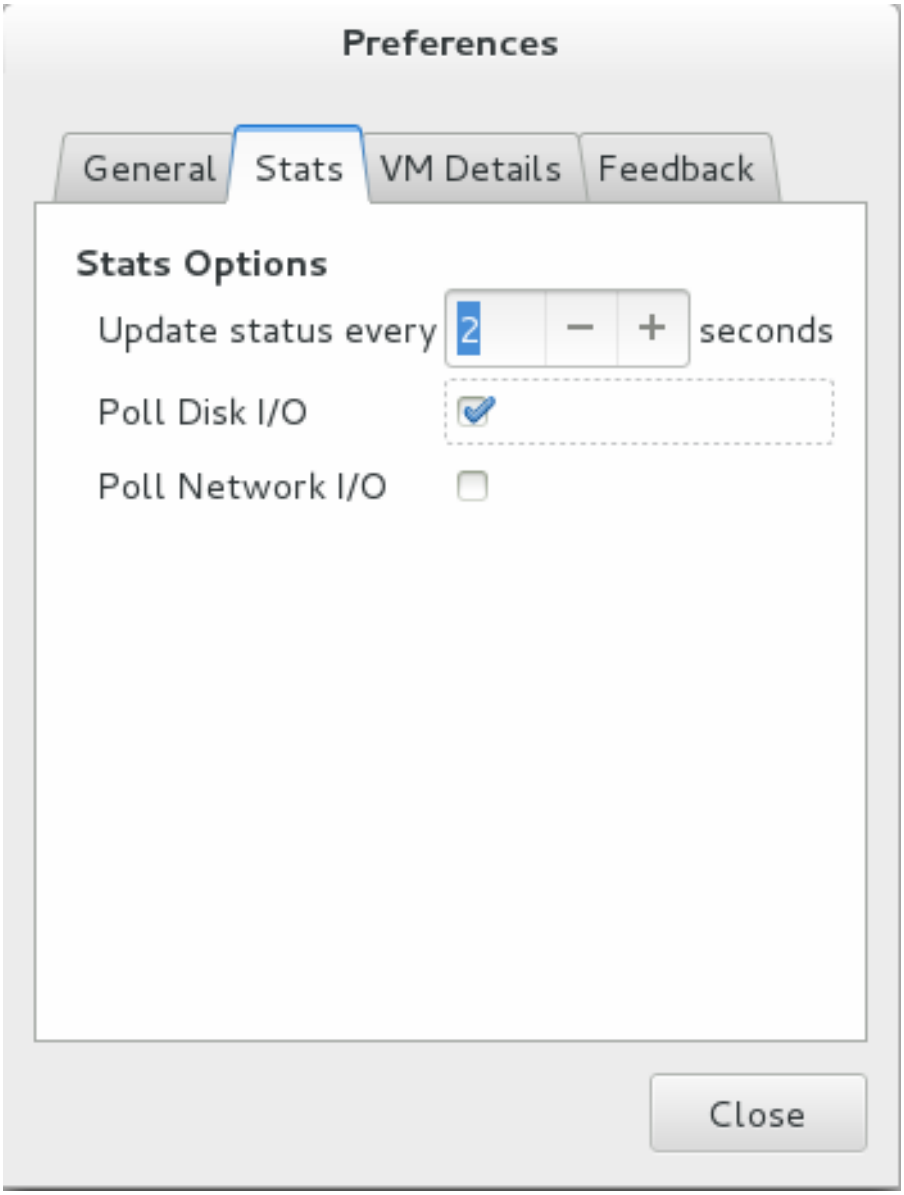


図25.28 ディスク I/O の有効化

3. ディスク I/O の表示を有効にするには、 **表示** メニューから **グラフ**、**ディスク I/O** のチェックボックスの順で選択して印を付けます。

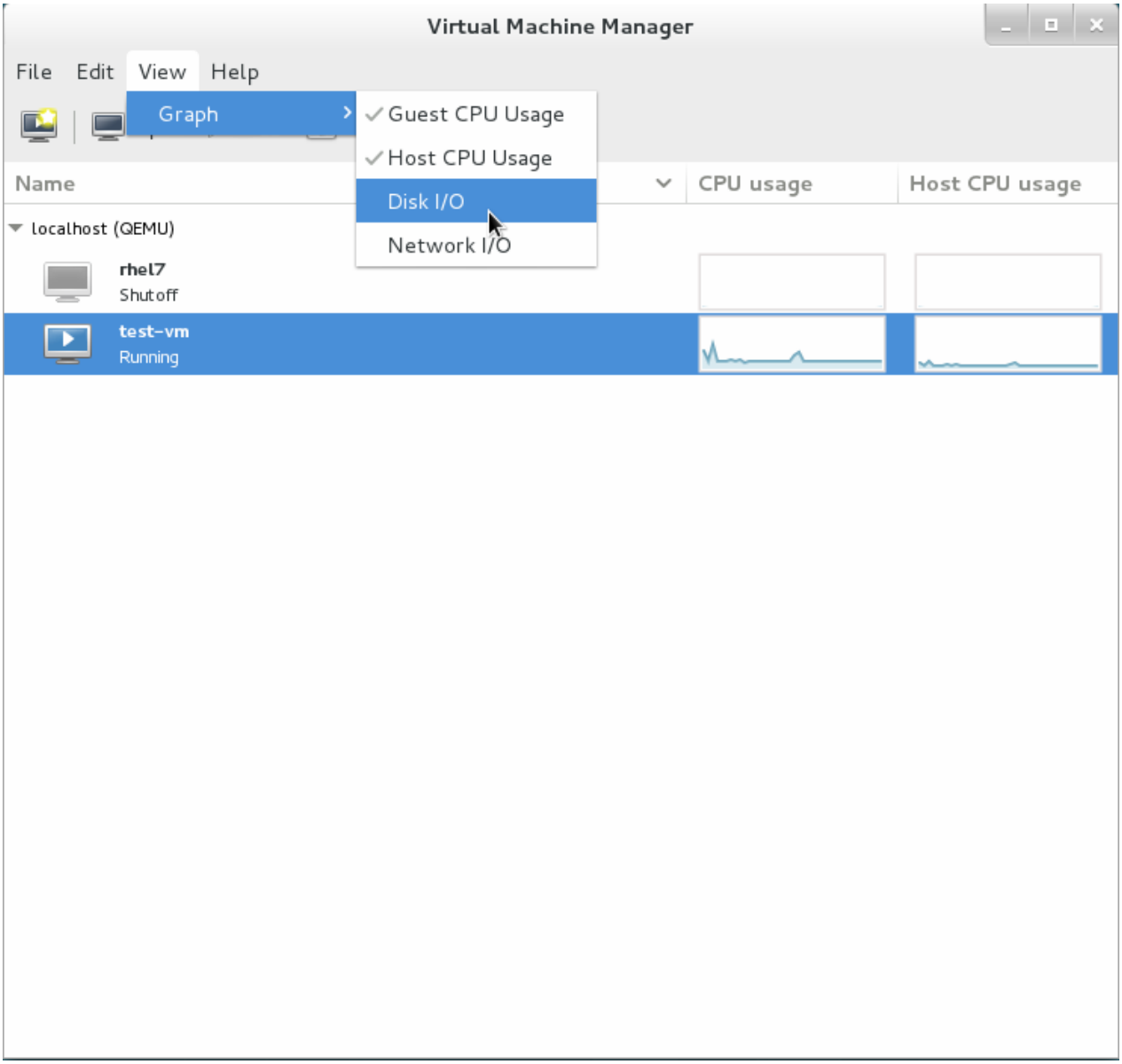


図25.29 ディスク I/O の選択

4. 仮想マシンマネージャーがシステム上の全仮想マシンの ディスク I/O のグラフを表示します。

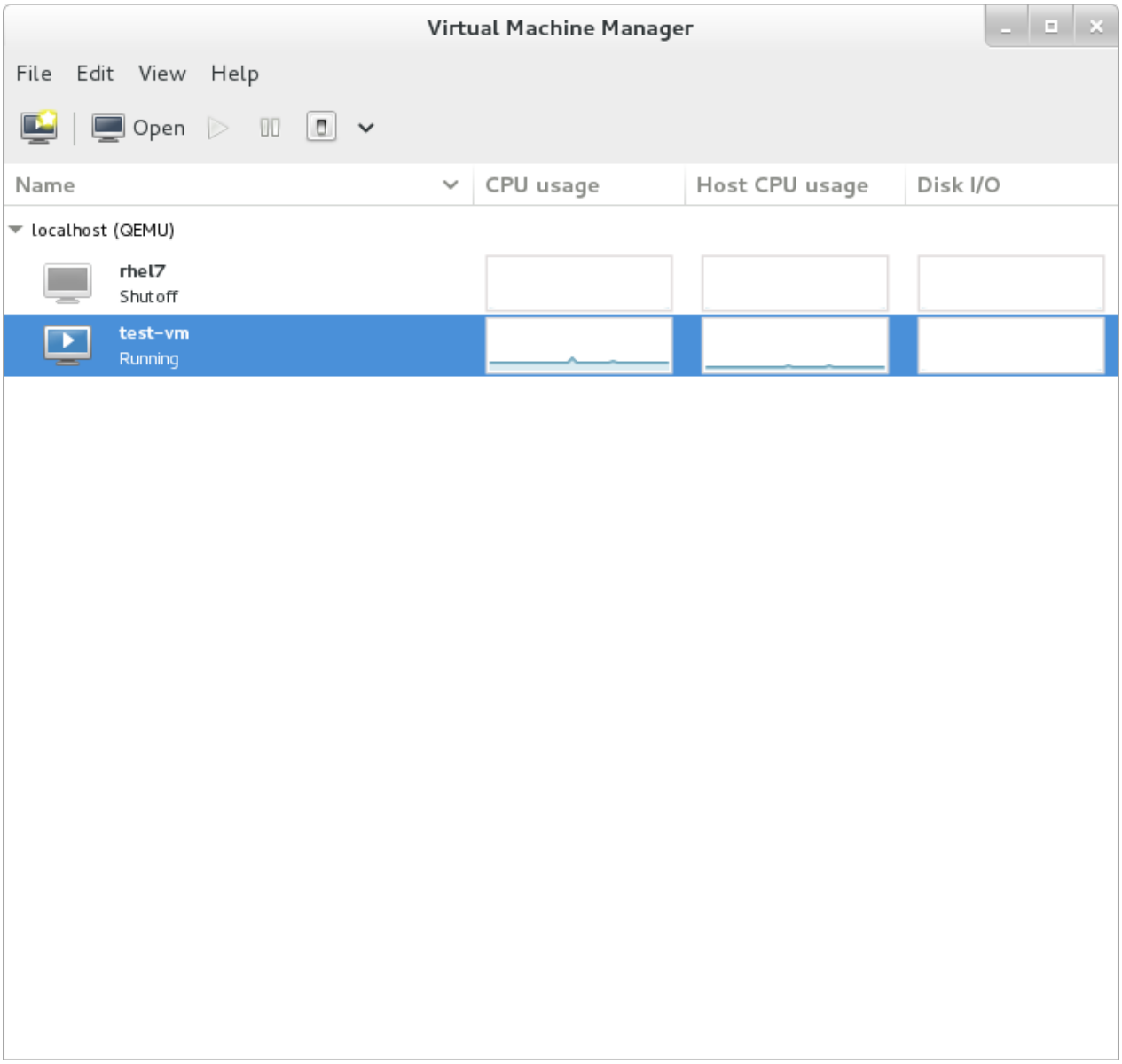


図25.30 ディスク I/O の表示

25.11. ネットワーク I/O の表示

システム上の全仮想マシンのネットワーク I/O を表示する

1. ネットワーク I/O の統計値収集の設定が有効になっていることを確認します。**編集** メニューから **設定** を選択します。**統計** タブをクリックします。
2. **ネットワーク I/O** のチェックボックスに印を付けます。

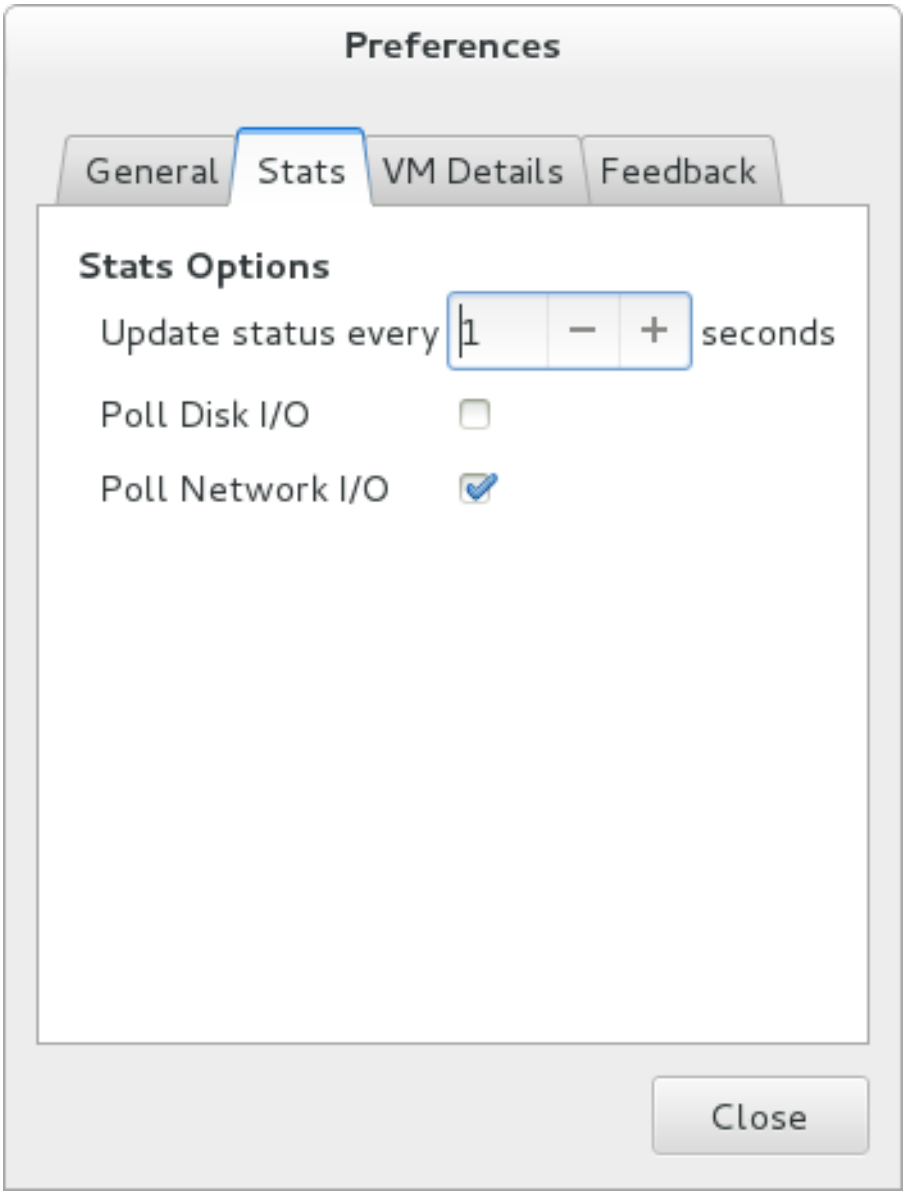


図25.31 ネットワーク I/O の有効化

3. ネットワーク I/O の統計値を表示するには、**表示** メニューから **グラフ**、 **ネットワーク I/O** のチェックボックスの順で選択して印を付けます。

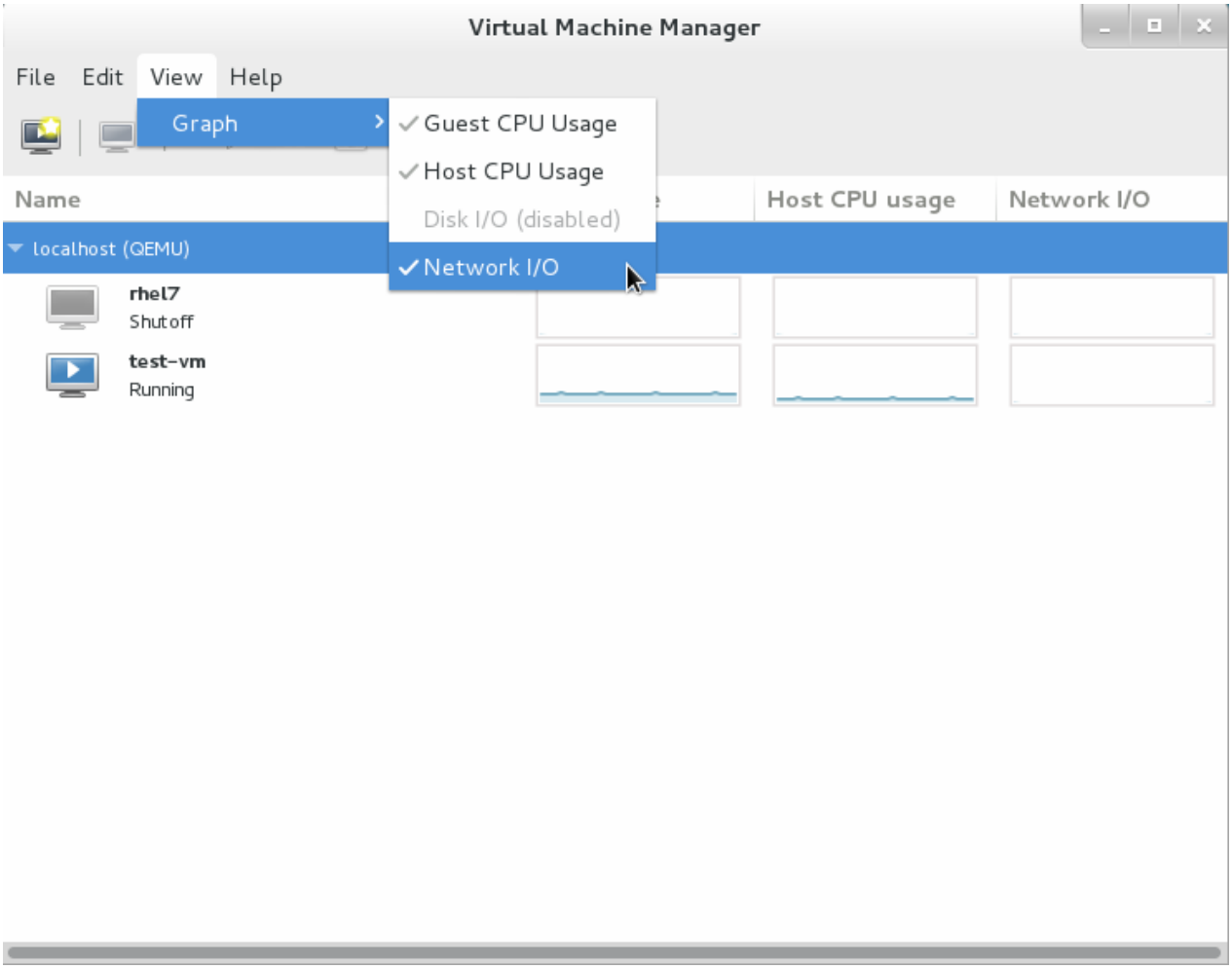


図25.32 ネットワーク I/O の選択

4. 仮想マシンマネージャーがシステム上の全仮想マシンのネットワーク I/O のグラフを表示します。

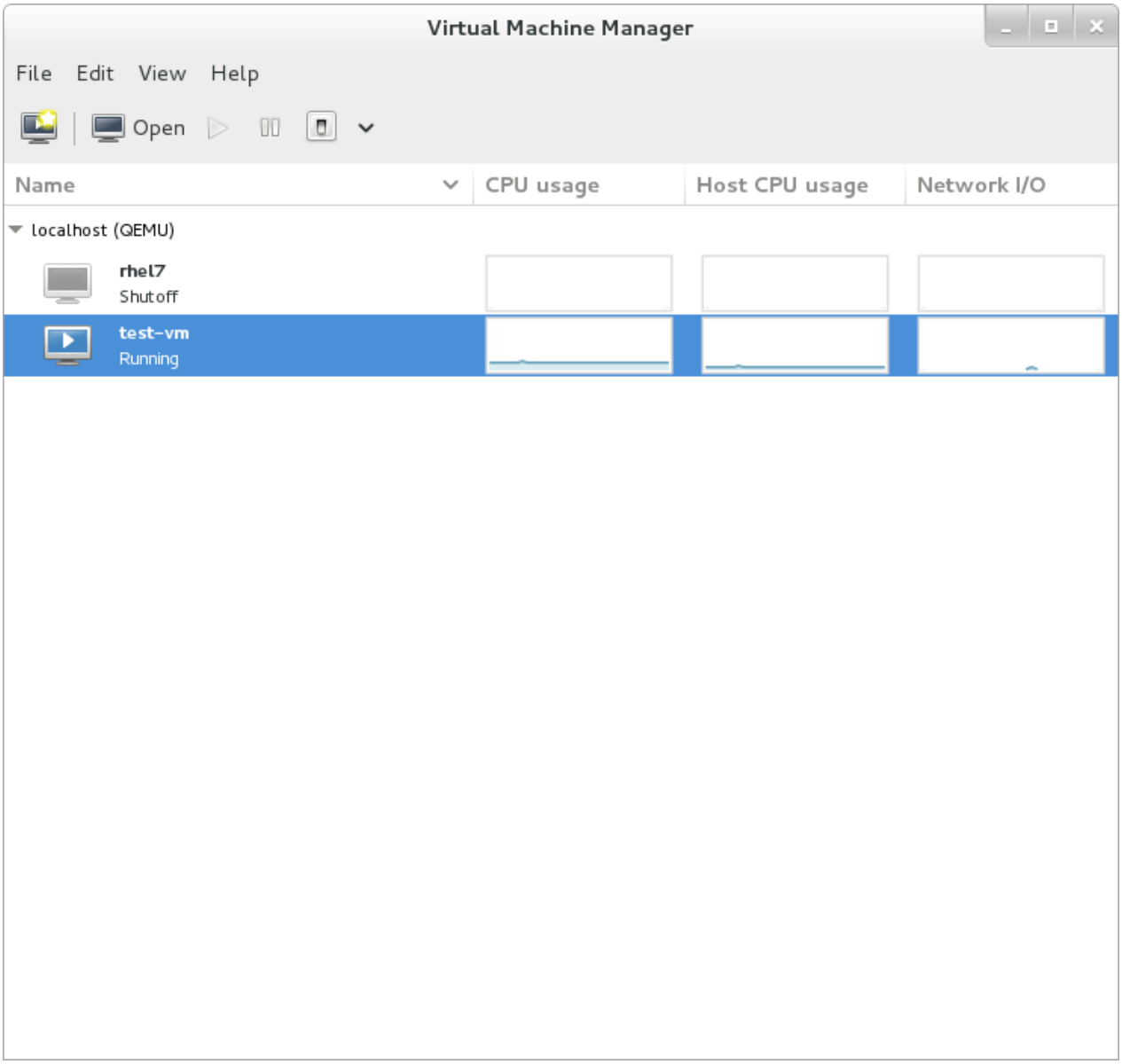


図25.33 ネットワーク I/O の表示

第26章 virsh を使用したゲスト仮想マシンの管理

virsh は、ゲスト仮想マシンとハイパーバイザーの管理に使用するコマンドラインインターフェースです。**virsh** コマンドラインツールは、**libvirt** 管理 API をベースにして構築されており、**qemu-kvm** コマンドやグラフィカルな **virt-manager** アプリケーションの代替として機能します。**virsh** コマンドは、通常のユーザーには読み取り専用モードで使用されますが、root アクセス権を持つユーザーは完全な管理機能を使用できます。**virsh** コマンドは、仮想化管理のスクリプトを作成する際に最適です。

26.1. 一般的なコマンド

このセクションに記載するコマンドは、いずれかのドメインに特有のものではないため、一般的なコマンドになります。

26.1.1. help

$ virsh help [command|group] help コマンドはオプションと共に、またはオプションなしで使用できます。オプションなしで使用すると、すべてのコマンドが各行に 1 つずつ一覧表示されます。オプションと共に使用される場合は、一覧はカテゴリーにグループ化され、各グループのキーワードが表示されます。

特定のオプションに該当するコマンドのみを表示するには、そのグループのキーワードをオプションとして指定する必要があります。以下のようになります。

```
$ virsh help pool
 Storage Pool (help keyword 'pool'):
    find-storage-pool-sources-as   find potential storage pool sources
    find-storage-pool-sources      discover potential storage pool
sources
    pool-autostart                 autostart a pool
    pool-build                     build a pool
    pool-create-as                 create a pool from a set of args
    pool-create                    create a pool from an XML file
    pool-define-as                 define a pool from a set of args
    pool-define                    define (but don't start) a pool from
an XML file
    pool-delete                    delete a pool
    pool-destroy                   destroy (stop) a pool
    pool-dumpxml                   pool information in XML
    pool-edit                      edit XML configuration for a storage
pool
    pool-info                      storage pool information
    pool-list                      list pools
    pool-name                      convert a pool UUID to pool name
    pool-refresh                   refresh a pool
    pool-start                     start a (previously defined) inactive
pool
    pool-undefine                  undefine an inactive pool
    pool-uuid                      convert a pool name to pool UUID
```

この同じコマンドをコマンドオプションと共に使用すると、その特定のコマンドオプションについての help 情報が表示されます。以下のようになります。

```
$virsh help vol-path
  NAME
```

```
    vol-path - returns the volume path for a given volume name or key

  SYNOPSIS
    vol-path <vol> [--pool <string>]

  OPTIONS
    [--vol] <string>  volume name or key
    --pool <string>  pool name or uuid
```

26.1.2. quit と exit

quit コマンドと exit コマンドはターミナルを閉じます。以下のようになります。

```
$virsh exit
```

```
$virsh quit
```

26.1.3. version

version コマンドは現在の libvirt バージョンを表示し、ビルド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 1.1.1
Using library: libvirt 1.1.1
Using API: QEMU 1.1.1
Running hypervisor: QEMU 1.5.3
```

26.1.4. 引数の表示

virsh echo [--shell][--xml][arg] は、指定した引数をエコーまたは表示します。エコーされるそれぞれの引数はスペースで区切られます。**--shell** オプションを使用すると、出力は必要に応じて単一引用符で囲まれるため、シェルコマンドで再利用するのに適しています。**--xml** オプションが使用される場合、出力は XML ファイルでの使用に適したものになります。

26.1.5. 接続

ハイパーバイザーセッションに接続します。シェルが最初に起動する際、このコマンドは URI パラメーターが **-c** コマンドによって要求されると自動的に実行されます。URI はハイパーバイザーに接続する方法を指定します。最もよく使用される URI は以下になります。

- **xen:///**- ローカルの XEN ハイパーバイザーに接続します。
- **qemu:///system** - QEMU および KVM ドメインを監視するデーモンに root としてローカルに接続します。
- **xen:///session** - ユーザーの QEMU および KVM ドメインのセットにユーザーとしてローカルに接続します。
- **lxc:///** - ローカルの Linux コンテナーに接続します。

他の値については libvert の web サイト (http://libvirt.org/uri.html) を参照してください。

コマンドは以下のように実行できます。

```
$virsh connect {name|URI}
```

ここで、**{name}** 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) です。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** を追加します。URI についてさらに詳しくは、リモート URI を参照してください。URI が不明な場合は、**uri** コマンドを実行すると、表示されます。

```
$ virsh uri
qemu:///session
```

26.1.6. 基本的な情報の表示

以下のコマンドは、基本情報を表示するために使用できます。

- $ **hostname** - ハイパーバイザーのホスト名を表示します。
- $ **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)。

26.1.7. NMI の挿入

$ **virsh inject-nmi [domain]** は NMI (マスクが不可能な割り込み) 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。このコマンドを実行するには、以下のようにします。

```
$ virsh inject-nmi guest-1
```

26.2. 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、「ゲストにファイルベースのストレージを追加する」 を参照してください。

手順26.1 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

以下の手順は、USB デバイスをゲスト仮想マシンに割り当てる方法について説明しています。これは、ゲスト仮想マシンの実行中にホットプラグ手順として実行することも、ゲストの停止中に実行することもできます。エミュレートするデバイスは、ホスト物理マシンに割り当てられている必要があります。

1. 以下のコマンドを使って、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 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。

```
   <hostdev mode='subsystem' type='usb' managed='yes'>
      <source>
```

```
        <vendor id='0x17ef'/>
        <product id='0x480f'/>
      </source>
    </hostdev>
  ...
```

図26.1 USB デバイス XML スニペット

3. 以下のコマンドを使って、デバイスを割り当てます。

```
# virsh attach-device rhel7 --file usb_device.xml> --config
```

この例では、[rhel7] はゲスト仮想マシンの名前で、[usb_device.xml] は直前のステップで作成したファイルです。次回の起動時に変更を有効にする場合は、**--config** を使用します。変更を永続化させる場合は、**--persistent** オプションを使用します。変更を現在のドメインで有効にする場合は、**--current** オプションを使用します。他のオプションについては、Virsh の man ページを参照してください。

4. デバイスを切り離す場合 (ホットアンプラグ) は、以下のコマンドを実行します。

```
# virsh detach-device rhel7 --file usb_device.xml>
```

この例では、[rhel7] はゲスト仮想マシンの名前で、[usb_device.xml] は直前のステップで割り当てたファイルです。

26.3. インターフェースデバイスの割り当て

virsh attach-interface*domain type source* コマンドは以下の引数を受け入れます。

- **--live** - 実行中のドメインから値を取得します
- **--config** - 次回起動時に使用される値を取得します
- **--current** - ドメインの現在の状態に基づいて値を取得します
- **--persistent** - オフラインのドメインについては**--config** のように動作し、実行中のドメインについては **--live** のように動作します。
- **--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、これを使用します。
- **--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、これを使用します。
- **--model** - モデルのタイプを指定するには、これを使用します。
- **--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。

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** を使用します。

26.4. CDROM メディアの変更

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。

```
# change-media domain path source --eject --insert --update --current --
live --config --force
```

- **--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** - 実行中のドメイ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

26.5. ドメインコマンド

これらのコマンドのほとんどは指定されたドメインを直接操作するため、ドメイン名が必要になります。ドメインは、短整数 (short integer)(0,1,2...)、名前、または完全な UUID で指定される場合があります。

26.5.1. 起動時に自動的に起動するようにドメインを設定

$ virsh autostart [--disable] domain は、起動時に指定されたドメインを自動的に起動します。**--disable** 引数を使用すると、autostart が無効になります。

```
# virsh autostart rhel7
```

上記の例では、rhel7 ゲスト仮想マシンは、ホスト物理マシンの起動時に自動的に起動します。

```
# virsh autostart rhel7--disable
```

上記の例では、autostart 機能は無効にされているため、ゲスト仮想マシンはホスト物理マシンの起動時にも自動的に起動しません。

26.5.2. 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 <string>] [--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。オプションの --devname <string> 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、プライマリーコンソールが開かれます。**--force** 引数は、コンソールの接続を強制するか、または disconnect と一緒に使用される場合は、接続を切断します。**--safe** 引数を使用すると、安全なコンソール処理がサポートされている場合のみゲストの接続を許可します。

```
$ virsh console virtual_machine --safe
```

26.5.3. XML ファイルによるドメインの定義

define <FILE> コマンドは XML ファイルのドメインを定義します。この場合、ドメイン定義は登録されますが、起動しません。ドメインがすでに実行されている場合は、変更は次回の起動時に有効になります。

26.5.4. ドメインの説明またはタイトルの編集および表示

このコマンドは、ドメインの説明およびタイトルを表示するか、または変更するためにのみ使用されますが、ドメインを設定するためには使用されません。これらの値は、ドメインを簡単に特定できるように任意のテキストデータを格納するためのユーザーフィールドに過ぎません。タイトルは、強制ではないですが、なるべく短くする必要があります。

引数の **--live** または **--config** は、このコマンドをドメインの稼働中の定義または永続的な定義で機能させるかどうかを選択します。**--live** と **--config** の両方が指定される場合、**--config** オプションが、現在の説明を取得する際に優先され、説明の設定中に実行中の設定および永続的な設定の両方が更新されます。**--current** 引数は、現行状態の設定を変更するか、または取得します。これらのいずれも指定されていない場合は、排他的かつ暗示的になります。**--edit** 引数は、現在の説明またはタイトルのコンテンツのあるエディターが開かれた後に、コンテンツが保存されることを指定します。--title は、タイトルフィールドで、説明ではなく操作を選択します。さらに、**--edit** または **--new-desc** のいずれも指定されていない場合、説明が表示されますが、これを変更することはできません。

26.5.5. デバイスブロック統計の表示

このコマンドは、実行中のドメインのブロック統計を表示します。ドメイン名とデバイス名の両方を用意しておく必要があります (**virsh domblklist** を使用してデバイスを一覧表示します)。この場合、ブロックデバイスは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'/>) です。すべてのハイパーバイザーがすべてのフィールドを表示する訳ではないことに注意してください。最も読みやすい形式で出力を表示するようにするには、以下に示すように **--human** 引数を使用します。

```
# virsh domblklist rhel7
Target     Source
------------------------------------------------
vda        /VirtualMachines/rhel7.img
hdc        -

# virsh domblkstat --human rhel7 vda
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
```

26.5.6. ネットワーク統計の取得

domnetstat [domain][interface-device] コマンドは、所定ドメイ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。

```
# domifstat rhel7 eth0
```

26.5.7. ドメインの仮想インターフェースのリンク状態の変更

このコマンドは、指定されたインターフェースをアクティベート/非アクティベートに設定することができます。**domif-setlink [domain][interface-device][state]{--config}** は、指定されたドメインの指定インターフェースの状態を変更します。ドメインの永続的な設定の変更のみが必要な場合は、**--config** 引数を使用する必要があることに注意してください。さらに、互換性維持の目的から**--persistent** は **--config** のエイリアスになります。「interface device」はインターネットのターゲット名または MAC アドレスにすることができます。

```
# domif-setlink rhel7 eth0 up
```

26.5.8. ドメインの仮想インターフェースのリンク状態の一覧表示

このコマンドは、所定ドメインで指定されたインターフェースの状態を照会するために使用できます。ドメインの永続的な設定の変更のみが必要な場合は、**--config** 引数を使用する必要があります。さらに、互換性維持の目的から **--persistent** は **--config** のエイリアスになります。「interface device」はインターネットのターゲット名または MAC アドレスにすることができます。

```
# domif-getlink rhel7 eth0 up
```

26.5.9. 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

domiftune は、ゲスト仮想マシンのネットワークインターフェース帯域幅パラメーターを設定します。以下の形式を使用してください。

```
#virsh domiftune domain interface-device [[--config] [--live] | [--
current]] [--inbound average,peak,burst] [--outbound average,peak,burst]
```

必要なパラメーターはゲスト仮想マシンのドメイン名およびインターフェースデバイスのみであり、**--config**、**--live**、および **--current** 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、現在のネットワークインターフェース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てください。

- <interface-device> これは必須であり、ドメインのネットワークインターフェースの帯域幅パラメーターを設定するか、または照会します。**interface-device** は、インターフェースのターゲット名 (<target dev='name'/>)、または MAC アドレスにすることができます。
- **--inbound** または **--outbound** のいずれも指定されていない場合、このコマンドは帯域幅の設定を照会するか、または表示します。それ以外の場合は、受信または送信帯域幅を設定します。average、peak、burst は **attach-interface** コマンドの場合と同様になります。「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。

26.5.10. 実行中ドメインのメモリー統計の取得

このコマンドは、使用しているハイパーバイザーに応じて異なる結果を返す場合があります。

dommemstat [domain] [--period (sec)][[--config][--live]|[--current]] は、実行中ドメインのメモリー統計を表示します。**--period** 引数を使用する場合、秒単位の時間を指定する必要があります。この引数を 0 より大きな値に設定すると、バルーンドライバーから後続の **domemstat** コマンドで表示される追加の統計を返すことができるようになります。**--period** 引数を 0 に設定すると、バルーンドライバーの収集は停止されますが、バルーンドライバーの統計は消去されません。さらにバルーン

ドライバーの収集期間を設定するために**--live**、**--config**、または **--current** 引数を使用する場合、必ず **--period** オプションを設定する必要があります。**--live** 引数が指定されている場合、実行中のゲストの収集期間のみが影響を受けます。**--config** 引数が使用されている場合、永続的なゲストの次回の起動に影響を与えます。**--current** 引数が使用されている場合、現在のゲストの状態が影響を受けます。

--live および **--config** 引数の両方を使用できますが、**--current** は排他的です。フラグが指定されていない場合、ゲストの状態によって動作は異なります。

```
#virsh domemstat rhel7--current
```

26.5.11. ブロックデバイスのエラーの表示

このコマンドは、I/O エラーのためにドメインが一時停止になっていることを報告する **domstate** の後に使用することが最も適しています。**domblkerror *domain*** コマンドは所定ドメインのエラー状態にあるすべてのブロックデバイスを表示し、デバイスが報告しているエラーメッセージを表示します。

```
# virsh domblkerror rhel7
```

26.5.12. ブロックデバイス容量の表示

以下の場合、ブロックデバイスは固有なターゲット名 (<target dev='name'/>) またはソースファイル (<source file ='name'/>) になります。一覧を取得するには、**domblklist** を実行できます。この **domblkinfo** には *domain* 名が必要です。

```
# virsh domblkinfo rhel7
```

26.5.13. ドメインに関連付けられたブロックデバイスの表示

domblklist *domain* --inactive--details は、指定されたドメイ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、実行中ドメイ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、**domblkinfo** および **snapshot-create** で使用することができます。

```
#domblklist rhel7 --details
```

26.5.14. ドメインに関連付けられた仮想インターフェースの表示

domiflist 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情報を表示する表が作成されます。**domiflist** には、*domain* 名が必要であり、オプションで **--inactive** 引数を取ることができます。

--inactive が指定される場合は、結果には次回起動時に使用されるデバイスが表示され、実行中ドメイ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。

仮想インターフェースの MAC アドレスが必要なコマンド (**detach-interface** または **domif-setlink** など) は、このコマンドで表示される出力を受け入れます。

26.5.15. blockcommit を使用したバッキングチェインの短縮化

このセクションでは、**blockcommit** を使用してバッキングチェインを短縮する方法について説明します。バッキングチェインのさらに詳しい背景情報については、「ライブブロックコピーによるディスクイメージの管理」 を参照してください。

blockcommit は、チェインの一部にあるデータをバッキング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イ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ます。

```
base ← snap1 ← snap2 ← active.
```

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、チェインから 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26.2 virsh blockcommit

- 次のコマンドを実行します。

  ```
  # virsh blockcommit $dom $disk -base snap1 -top snap2 -wait -verbose
  ```

  snap2 のコンテンツが snap1 に移行します。以下のようになります。

  base ← snap1 ← active。Snap2 は無効になり、削除することができません。

警告

blockcommit は、***-base*** 引数に依存するすべてのファイル (ベースをポイントする、***-top*** 引数に依存するファイル以外のもの) を破損させます。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、進捗を画面上に出力することができます。

26.5.16. blockpull を使用したバッキングチェインの短縮化

blockpull は、以下のように応用して使用することができます。

- イメージにそのバッキングイメージチェイ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キングイメージやこれに類するものに依存しなくてすむような自己完結型のファイルになります。
  - 以前: base.img ← Active
  - 今後: base.img がゲストによって使用されなくなり、Active にはデータのすべてが含まれます。
- バッキングイメージチェイ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。
  - 以前: base ← sn1 ←sn2 ← active
  - 今後: base.img ← active active には sn1 および sn2 からのすべてのデータが含まれ、sn1 または sn2 のいずれもゲストによって使用されないことに注意してください。
- 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。
  - 以前 (元のイメージファイル): **/fs1/base.vm.img**

- 今後: **/fs2/active.vm.qcow2** は新規ファイルシステムであり、**/fs1/base.vm.img** は使用されなくなりました。

- 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ソース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。以前:**/source-host/base.vm.img** 今後:**/destination-host/active.vm.qcow2**。**/source-host/base.vm.img** は使用されなくなります。

手順26.3 blockpull を使用したバッキングチェインの短縮化

1. **blockpull** を実行する前に以下のコマンドを実行すると便利です。

   ```
   # virsh snapshot-create-as $dom $name - disk-only
   ```

2. チェインが **base** ← **snap1** ← **snap2** ← **active** のようになる場合、以下を実行します。

   ```
   # virsh blockpull $dom $disk snap1
   ```

   このコマンドは、データを snap2 から active にプルしてから base ← snap1 ← active の状態にすることにより、'snap1' を active のバッキングファイルにします。

3. **blockpull** が完了すると、チェイン内の追加イメージを作成したスナップショットの**libvirt** 追跡は役に立たなくなります。以下のコマンドを使って、古くなったスナップショットでの追跡を削除します。

   ```
   # virsh snapshot-delete $dom $name - metadata
   ```

blockpull のその他の応用は、以下のように実行できます。

- 単一イメージをフラット化し、これにそのバッキングイメージチェインのデータを設定する: **# virsh blockpull example-domain vda - wait**

- バッキングイメージチェインの一部をフラット化する: **# virsh blockpull example-domain vda - base /path/to/base.img - wait**

- ディスクイメージをホスト上の新規ファイルシステムに移動する: **# virsh snapshot-create example-domain - xmlfile /path/to/new.xml - disk-only**、およびその後の **# virsh blockpull example-domain vda - wait**

- 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:

  - 移行先で以下を実行:

    ```
     # qemu-img create -f qcow2 -o backing_file=/source-host/vm.img
    /destination-host/vm.qcow2
    ```

  - ソースで以下を実行:

    ```
    # virsh migrate example-domain
    ```

  - 移行先で以下を実行:

    ```
    # virsh blockpull example-domain vda - wait
    ```

26.5.17. blockresize を使用したドメインパスのサイズ変更

blockresize を使用して、ドメインの実行中にドメイ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、ドメイ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 **domblklist** を使用して、所定ドメイ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 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反映する前にゲストを再起動しておく必要があります。

- 以下のコマンドを実行します: **blockresize [domain] [path size]** ここで、
  - ドメインは、サイズを変更するドメインのファイルの固有のターゲット名またはソースファイルです。
  - サフィックスがない場合、パスサイズはデフォルトで KiB (1024 バイトブロック単位) になる単位付き整数です。バイトについては、「B」のサフィックスを使用する必要があります。

26.5.18. ライブブロックコピーによるディスクイメージの管理

注記

ライブブロックコピーは、Red Hat Enterprise Linux で提供される KVM のバージョンでサポートされない機能です。ライブブロックコピーは、Red Hat Virtualization で提供される KVM のバージョンで利用できます。この機能がサポートされるには、KVM のこのバージョンが物理ホストマシン上で実行されている必要がります。さらに詳しくは、Red Hat の担当者にお問い合わせください。

ライブブロックコピーにより、ゲストの実行中に、使用中のゲストディスクイメージを移行先イメージにコピーでき、ゲストディスクイメージを移行先ゲストイメージに切り替えることができます。ライブマイグレーションではホストのメモリーおよびレジストリー状態を移動する間、ゲストは共有ストレージに保持されます。ライブブロックコピーでは、ゲストの実行中に、ゲストのコンテンツ全体を別のホストにオンザフライで移動できます。また、ライブブロックコピーは、永続的な共有ストレージを必要とすることなく、ライブマイグレーションに使用することもできます。この方法では、ディスクイメージは移行後に移行先ホストにコピーされますが、ゲストの実行中にこれが行なわれます。

ライブブロックコピーは、以下のように適用する場合にとくに便利です。

- ローカルストレージから集中管理できる場所にゲストイメージを移動する
- メンテナンスが必要な場合に、パフォーマンスを低下させることなく、ゲストを別の場所に転送する
- イメージをスピードと効率を維持した状態でゲストイメージを管理する
- ゲストをシャットダウンせずにイメージのフォーマット変換を実行する

例26.1 例 (ライブブロックコピー)

以下の例は、ライブブロックコピーの実行時にどのようになるかを示しています。この例では、ソースと移行先の間で共有されるバッキングファイルがあります。さらに、ソースにのみ 2 つのオーバーレイ (sn1 および sn2) があり、これらはコピーする必要があります。

1. 開始時のバッキングファイルチェインは以下のようになります。

   base ← sn1 ← sn2

   コンポーネントは以下のようになります。

   - base - 元のディスクイメージ
   - sn1 - base ディスクイメージから取られた最初のスナップショット
   - sn2 - 最新のスナップショット
   - active - ディスクのコピー

2. イメージのコピーが sn2 の上に新規イメージとして作成されると、結果は以下のようになります。

   base ← sn1 ← sn2 ← active

3. この時点で、読み取りアクセス権はすべて正しい順序にあり、自動的に設定されます。書き込みアクセス権が適切に設定されていることを確認するために、ミラーメカニズムがすべての書き込みを sn2 と active の両方にリダイレクトし、sn2 と active がいつでも同じ内容を読み込めるようにします (また、このミラーメカニズムが、ライブブロックコピーとイメージストリーミング間の本質的な違いになります)。

4. すべてのディスククラスターにループするバックグラウンドタスクが実行されます。それぞれのクラスターについては、以下のケースおよびアクションが考えられます。

   - クラスターは active ですでに割り当てられており、とくに実行すべきことはない。
   - **bdrv_is_allocated()** を使用し、バッキングファイルチェインをフォローする。クラスターが (共有される) base から読み込まれる場合、とくに実行すべきことはない。
   - **bdrv_is_allocated()** のバリアントが実行可能でない場合、イメージをリベースし、コピーが必要かどうかを決定するために、読み込みデータをベースの書き込みデータと比較する。
   - 上記以外のすべてのケースでは、クラスターを **active** にコピーする

5. コピーが完了したら、active のバッキングファイルが base に切り替わります (リベースと同様)。

一連のスナップショット後にバッキングチェインの長さを短縮するには、以下のコマンドが役立ちます: **blockcommit** および **blockpull** さらに詳しくは、「blockcommit を使用したバッキングチェインの短縮化」 を参照してください。

26.5.19. グラフィカルディスプレイへの接続用の URI を表示

virsh domdisplay コマンドを実行することにより、VNC、SPICE、または RDP 経由でドメイ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。引数 **--include-password** が使用される場合、SPICE チャネルのパスワードが URI に組み込まれます。

26.5.20. 使用されていないブロックの破棄

virsh domfstrim *doman* --minium --mountpoint コマンドは、実行中の指定されたドメイン内のすべてのマウントされたファイルシステム上で fstrim を実行します。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** が使用される場合、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、連続する空き容量の範囲の長さとしてゲストカーネルに送信されます。この容量より小さい値は無視されます。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。デフォルトの最小値はゼロであり、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。ユーザーが 1 つのマウントポイントのみをトリムする必要がある場合は、**--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。

26.5.21. ドメイン検索コマンド

以下のコマンドは、所定ドメインについての異なる情報を表示します。

- **virsh domhostname *domain*** は、指定されるドメインのホスト名を表示します (ハイパーバイザーがこれを公開できる場合)。
- **virsh dominfo *domain*** は、指定されるドメインについての基本情報を表示します。
- **virsh domuid *domain*|*ID*** は、所定のドメイン名または ID を UUID に変換します。
- **virsh domid *domain*|*ID*** は、所定のドメイン名または UUID を ID に変換します。
- **virsh domjobabort *domain*** は、指定されるドメイン上で現在実行されているジョブを中止します。
- **virsh domjobinfo *domain*** は、移行の統計情報を含め、指定されるドメイン上で実行されているジョブについての情報を表示します。
- **virsh domname *domain ID*|*UUID*** は、所定のドメイン ID または UUID をドメイン名に変換します。
- **virsh domstate *domain*** は、所定ドメインの状態を表示します。**--reason** 引数を使用すると、表示された状態についての理由も表示されます。
- **virsh domcontrol *domain*** は、ドメインを制御するために使用された VMM 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26.2 統計的フィードバックの例

ドメインについての情報を取得するために、以下のコマンドを実行します。

```
# virsh domjobinfo rhel7
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
```

26.5.22. QEMU 引数のドメイン XML への変換

virsh domxml-from-native は、libvirt のドメイン XML を使用して QEMU 引数の既存のセットをゲストの記述に変換する方法を提供します。libvirt のドメイン XML 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ことになっている点に注意してください。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使用しないでください。新しいゲストは virsh または virt-manager のいずれかを使って作成する必要があります。追加情報については、こちらをご覧ください。

以下の引数ファイルを持つ QEMU ゲストがあるとします。

```
 $ 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
```

これをドメイン XML に変換してゲストが libvirt で管理できるようにするには、以下を実行します。

```
$ virsh domxml-from-native qemu-argv demo.args
```

このコマンドは、上記の引数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>
```

26.5.23. ドメインのコアのダンプファイルの作成

ドメインのコアを含むダンプファイルを解析するために作成することが必要になる場合があります (とくにトラブルシューティングの場合)。この場合、**virsh dump *domain corefilepath* --bypass-cache --live |--crash |--reset --verbose --memory-only** により、*corefilepath* で指定されるファイルにドメインコアがダンプされます。ハイパーバイザーによっては、このアクションを制限している場合があり、この場合ユーザーは *corefilepath* パラメーターで指定されるファイルとパスに対して適切なアクセス権があることを手動で確認することが必要になる場合があります。このコマンドは、他のパスルーデバイスと共に SR-IOV デバイスでサポートされます。以下の引数がサポートされており、かつ次のような効果があります。

- **--bypass-cache**: 保存されるファイルには、ファイルシステムのキャッシュが含まれません。このオプションを選択すると、ダンプ操作の速度が遅くなる可能性があることに注意してください。
- **--live**: ドメインが実行を継続する際にファイルを保存し、ドメインが一時停止/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、ドメイ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。
- **--reset**: ダンプファイルが正常に保存されると、ドメイ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場合) は、ダンプファイルのコンテンツにドメインのメモリーと 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を作成します。このオプションは、完全なダンプが失敗する場合に使用する必要があります。ダンプの失敗は、ドメインのライブマイグレーションを実行できない場合に生じる可能性があります (パススルー PCI デバイスが原因)。

プロセス全体は **domjobinfo** コマンドを使用して監視でき、**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。

26.5.24. 仮想マシンの XML ダンプの作成 (設定ファイル)

virsh を使用してゲスト仮想マシンの XML 設定ファイルを出力します。

```
# virsh dumpxml {guest-id, guestname or uuid}
```

このコマンドはゲスト仮想マシンの XML 設定ファイルを標準出力 (**stdout**) に対して出力します。この出力をファイルにパイプすることで、データを保存することができます。この出力を *guest.xml* というファイルにパイプする例を以下に示します。

```
# virsh dumpxml GuestID > guest.xml
```

このファイル **guest.xml** はゲスト仮想マシンを再作成することができます (「ゲスト仮想マシンの設定ファイルの編集」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。

virsh dumpxml 出力の一例

```
# virsh dumpxml guest1-rhel7-64
<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>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <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='raw' cache='none' io='threads'/>
      <source file='/home/guest-images/guest1-rhel6-64.img'/>
      <target dev='vda' bus='virtio'/>
      <shareable/<
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x0'/>
    </disk>
    <interface type='bridge'>
      <mac address='52:54:00:b9:35:a9'/>
      <source bridge='br0'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <input type='tablet' bus='usb'/>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes'/>
    <sound model='ich6'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04'
function='0x0'/>
    </sound>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02'
function='0x0'/>
    </video>
    <memballoon model='virtio'>
```

```
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06'
function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

<shareable/> のフラグが設定されている点に注意してください。ドメイン間でのデバイスの共有が予想されることを示しています (ハイパーバイザーおよび OS で対応していると仮定)。つまり、このデバイスに対してはキャッシュ機能は停止されることになります。

26.5.25. 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 **dumpxml** オプションを使用することができます (「仮想マシンの XML ダンプの作成 (設定ファイル)」 を参照)。ゲスト仮想マシンを **virsh** を使って XML ファイルから作成するには、以下を実行します。

```
# virsh create configuration_file.xml
```

26.6. ゲスト仮想マシンの設定ファイルの編集

dumpxml オプション (「仮想マシンの XML ダンプの作成 (設定ファイル)」 を参照) を使用する代わりに、ゲスト仮想マシンは実行中またはオフライン中のいずれかの時点で編集することができます。**virsh edit** コマンドがこの機能を提供します。たとえば ***rhel7*** という名前のゲスト仮想マシンを編集するには、以下を実行します。

```
# virsh edit rhel7
```

このコマンドを実行するとテキストエディターが開きます。 デフォルトのエディターは **$EDITOR** シェルパラメーターになります (デフォルトで **vi** に設定)。

26.6.1. 多機能 PCI デバイスの KVM ゲスト仮想マシンへの追加

このセクションでは多機能 PCI デバイスを KVM ゲスト仮想マシンに追加する方法について説明します。

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 type タグ内の **function='0x0'** に **multifunction='on'** のエントリーを追加します。

   これでゲスト仮想マシンが多機能 PCI デバイスを使用できるようになります。

   ```
   <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。

例:

```
<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** を実行すると、KVM ゲスト仮想マシンの次のような出力が表示されます。

```
$ 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。

26.6.2. 後の再起動に向けた実行中ドメインの停止

virsh managedsave *domain* --bypass-cache --running | --paused | --verbose は、後に同じ状態から再起動できるように実行中のドメインを保存し、破棄 (停止) します。**virsh start** コマンドを併用すると、ドメインはこの保存したポイントから自動的に起動します。**--bypass-cache** 引数を併用すると、保存により、ファイルシステムのキャッシュは回避されます。このオプションは保存プロセスのスピードを低下させることに注意してください。

--verbose: ダンププロセスの進捗が表示されます。

通常の状態では、管理保護は、保存の実行時にドメインが置かれている状態に応じて実行中 (running) の状態または一時停止 (paused) の状態のいずれかを決定します。ただし、実行中の状態のままにすべきであると指定する場合は **--running** 引数を使用するか、または一時停止の状態のままにすべきであると指定する場合は **--paused** 引数を使用することにより、これを上書きすることができます。

管理保存の状態を削除するには、**virsh managedsave-remove** コマンドを使用します。このコマンドは、ドメインの次回の動時に完全な起動を実行するように強制します。

管理保存プロセス全体は **domjobinfo** コマンドを使用して監視することも、**domjobabort** コマンドを使用してキャンセルすることもできることに注意してください。

26.6.3. 指定されたドメインについての CPU 統計の表示

virsh cpu-stats *domain* --total start count コマンドは指定されたドメインについての CPU 統計情報を提供します。デフォルトで、このコマンドは合計と共に、すべての CPU についての統計を表示します。**--total** 引数は、合計の統情報のみを表示します。

26.6.4. スクリーンショットの保存

virsh screenshot コマンドは、現在のドメインコンソールのスクリーンショットを取り、これをファイルに保存します。ただし、ハイパーバイザーが 1 つのドメインについて複数のディスプレイをサポートしている場合、**--screen** を使用し、さらにスクリーン ID を指定してキャプチャーするスクリーンを指定します。複数のグラフィックカードがある場合で、ヘッドがそれらのデバイスの前に列挙される場合、スクリーン ID 5 は 2 番目のカードの 2 番目のヘッドに対応します。

26.6.5. キー入力の組み合わせの指定されたドメインへの送信

virsh send-key *domain* --codeset --holdtime *keycode* コマンドを使用し、指定されたドメインの *キーコード* としてシーケンスを送信できます。

それぞれの *キーコード* には、対応する *コードセット* からの数値またはシンボリック名のいずれを使用することができます。複数の *キーコード* が指定される場合、それらすべてはゲスト仮想マシンに同時に送信されるため、順不同に受信される可能性があります。一意のキーコードを必要とする場合には、**send-key** コマンドを複数回送信する必要があります。

```
# virsh send-key rhel7 --holdtime 1000 0xf
```

--holdtime が指定されると、それぞれのキー入力は指定された時間 (ミリ秒単位) 保持されます。**--codeset** を使用してコードセットを指定できます。デフォルトは Linux ですが、以下のオプションが許可されます。

- **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多くあります。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。

26.6.6. プロセスのシグナル名を仮想プロセスに送信

virsh send-process-signal *domain-ID PID signame* を使用して、ドメイン ID が指定された実行中のドメイン内にある指定された仮想プロセス (プロセス ID または PID によって特定される) にシグナルの signame を送信することができます。整数のシグナル定数に加えて、1 つまたは複数の以下の signame を送信することができます。

- **nop** , **stkflt**
- **hup**, **cont**
- **int**, **chld**
- **quit**, **stop**
- **ill**, **tstp**
- **trap**, **ttin**
- **abrt**, **ttou**
- **bus**, **urg**
- **fpe**, **xcpu**
- **kill**, **xfsz**
- **usr1**, **vtalrm**
- **segv**, **prof**
- **usr2**, **winch**
- **pipe**, **poll**
- **alrm**, **pwr**
- **term**, **sys**
- その他のオプションについては Virsh の man ページに記載されています。これらのシンボルにも **sig** または **sig_** のプレフィックスが付けられており、大文字と小文字が区別されないことに注意してください。

```
# virsh send-process-signal rhel7 187 kill
```

26.6.7. VNC ディスプレイの IP アドレスとポート番号の表示

virsh vncdisplay は、指定されたドメインについての VNC ディスプレイの IP アドレスとポート番号を出力します。情報を使用できない場合、終了コード 1 が表示されます。

```
# virsh vncdisplay rhel7
127.0.0.1:0
```

26.7. NUMA ノードの管理

このセクションでは、NUMA ノードの管理に必要なコマンドについて説明します。

26.7.1. ノード情報の表示

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ノードについての基本情報を表示します。この出力は、**virNodeInfo** の構造に対応します。とくに「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。

```
$ 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
```

26.7.2. NUMA パラメーターの設定

virsh numatune は、指定されたドメインの 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。ドメイン 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。フラグを使用することなく、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ドメインが必要であり、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。ドメインが **strict** モードで起動されたのではない限り、ドメイ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** - ドメイ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回の起動に影響を与えます。
  - **--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。

26.7.3. NUMA セルの空きメモリー容量の表示

virsh freecell 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できます。いずれのオプションも使用されていない場合、マシンの空きメモリーの合計容量が表示されます。**--all** オプションを使用すると、各セル内の空きメモリーとマシンの空きメモリーの合計容量が表示されます。数値の引数またはセル番号と共に **--cellno** を使用すると、指定されるセルの空きメモリーが表示されます。

26.7.4. CPU 一覧の表示

nodecpumap コマンドは、オンラインであるかどうかにかかわらず、ノードで使用できる CPU の数を表示します。さらに、現在オンラインの CPU の数を一覧表示します。

```
$ virsh nodecpumap
   CPUs present: 4
   CPUs online: 1
   CPU map: y
```

26.7.5. CPU 統計の表示

nodecpustats コマンドは、CPU が指定されている場合に、指定される CPU についての統計情報を表示します。CPU が指定されていない場合、ノードの CPU 状態を表示します。パーセントが指定されている場合、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。

以下の例では CPU が指定されていません。

```
$ virsh nodecpustats
user:                1056442260000000
system:               401675280000000
idle:                7549613380000000
iowait:                94593570000000
```

以下の例は、CPU 番号 2 の統計的パーセンテージを示しています。

```
$ virsh nodecpustats 2 --percent
usage:              2.0%
user:               1.0%
system:             1.0%
idle:              98.0%
iowait:             0.0%
```

26.7.6. ホスト物理マシンの一時停止

nodesuspend コマンドは、ホスト物理マシンを 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ノードを起動します。**--target** 引数は、***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ホスト物理マシン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をお勧めします。

```
$ virsh nodesuspend disk 60
```

26.7.7. ノードメモリーパラメーターの設定

node-memory-tune [shm-pages-to-scan] [shm-sleep-milisecs] [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。このコマンドで設定できる 3 つのパラメーターがあります。

- **shm-pages-to-scan** - 共有メモリーサービスが休止する前にスキャンするページの数を設定します。
- **shm-sleep-milisecs** - 共有メモリーサービスが次回のスキャンの前に休止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。許可される値は ***0*** と ***1*** です。***0*** に設定される場合、マージできるページは同じ NUMA ノードのメモリー領域に物理的に存在するページのみです。***1*** に設定されると、すべての NUMA ノードからのページをマージできます。デフォルト設定は ***1*** です。

26.7.8. ホストノード上でのデバイスの作成

virsh nodedev-create *file* コマンドを使用すると、ホストノード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** は通常、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*file* には、ノードデバイスのトップレベルの **<device>** 記述の XML が含まれている必要があります。

このデバイスを停止するには、**nodedev-destroy *device*** コマンドを使用します。

26.7.9. ノードデバイスの切り離し

virsh nodedev-detach は nodedev をホストから切り離し、これが**<hostdev>** パススルー経由でゲストによって安全に使用できるようにします。このアクションは **nodedev-reattach** コマンドを使って無効にできますが、管理サービスについては自動的に実行されます。さらに、このコマンドは **nodedev-dettach** を受け入れます。

異なるドライバーがある場合、デバイスが異なるダミーデバイスにバインドされる可能性があることに注意してください。**--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。

26.7.10. デバイスのダンプ

virsh nodedev-dumpxml *device* は所定のノードデバイスの**<device>** XML 表現をダンプします。この XML 表現には、デバイス名、デバイスを所有する BUS、ベンダー、および製品 ID、および *libvirt* が使用できるデバイスの機能が含まれます。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。

26.7.11. ノード上でのデバイスの一覧表示

virsh nodedev-list *cap* --tree コマンドは、**libvirt** によって認識されているノード上の利用可能なすべてのデバイスを一覧表示します。*cap* は一覧を機能タイプでフィルターするために使用され、それぞれをコンマで区切って表示できますが、**--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置かれます。

```
 #  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

(this is a partial screen)
```

26.7.12. ノードのリセットのトリガー

nodedev-reset *nodedev* コマンドは、指定された *nodedev* の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便利です。*libvirt* は必要に応じてこのアクションを暗黙的に実行しますが、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。

26.8. ゲスト仮想マシンの起動、一時停止、再開、保存および復元

26.8.1. 定義されたドメインの起動

virsh start *domain* --console --paused --autodestroy --bypass-cache --force-boot --pass-fds コマンドは、すでに定義されているものの、最後に管理保存の状態になってから、または新規に起動されてから停止している停止状態のドメインを起動します。コマンドは以下の引数を取ります。

- **--console** - コンソールに割り当てられているドメインを起動します。
- **--paused** - ドライバーによってサポートされている場合、ドメインを起動してから一時停止の状態にします。
- **--autodestroy** - ゲスト仮想マシンは、virsh セッションが閉じるか、または libvirt の接続が閉じる場合に自動的に破棄されます。そうでない場合は、終了します。
- **--bypass-cache** - ドメインが管理保存の状態の場合に使用されます。これが使用される場合、ゲスト仮想マシンが復元され、システムのキャッシュが回避されます。これにより、復元プロセスのスピードが低下することに注意してください。
- **--force-boot** - 管理保存のオプションを破棄し、新規の起動を生じさせます。
- **--pass-fds** - 追加の引数のコンマ区切りの一覧で、ゲスト仮想マシンに渡されます。

26.8.2. ゲスト仮想マシンの一時停止

virsh を使ってゲスト仮想マシンを一時停止します。

```
# virsh suspend {domain-id, domain-name or domain-uuid}
```

ゲスト仮想マシンがサスペンド状態にある場合、システム RAM を消費しますが、プロセッサー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サスペンド中には発生しません。この操作は即時に行なわれるもので、ゲスト仮想マシンは **resume** (「ゲスト仮想マシンの再開」) オプションで再起動できます。

26.8.3. 実行中ドメインのサスペンド

virsh dompmsuspend *domain* --duration --target コマンドは、実行中のドメインを取り、そのドメインが 3 つの状態 (S3、S4、またはこれら 2 つのハイブリッド) のいずれかに置かれるように一時停止します。

```
# virsh dompmsuspend rhel7 --duration 100 --target mem
```

このコマンドは、以下の引数を取ります。

- **--duration** - 状態変更の期間 (秒単位) を設定します。
- **--target** - **mem (suspend to RAM (S3))disk (suspend to disk (S4))** または **hybrid (hybrid suspend)** のいずれかにできます。

26.8.4. pmsuspend 状態のドメインを起動

このコマンドは、設定した期間の有効期限が過ぎるのを待機するのではなく、pmsuspend 状態のゲストに対してウェイクアップアラートを挿入します。この操作は、ドメインが実行されている場合は失敗しません。

```
# dompmwakeup rhel7
```

このコマンドには、たとえば *rhel7* のようにドメインの名前が必要です。

26.8.5. ドメインの定義解除

このコマンドは、ドメインの定義を解除します。これは実行中のドメインで機能しますが、ドメインを停止することなく実行中のドメインを一時的なドメインに変換します。ドメインが停止中の場合、ドメイン設定は削除されます。

virsh undefine*domain*--managed-save--snapshots-metadata --storage --remove-all-storage --wipe-storage コマンドは以下の引数を取ることができます。

- **--managed-save** - この引数は、管理保護状態のドメイ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、管理保護状態のドメインの定義解除を試みると失敗します。
- **--snapshots-metadata** - この引数は、停止中ドメインの定義解除を実行する際に、スナップショット (**snaphot-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ドメイ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。この引数が使用される際に、ドメイ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。これは停止中のドメイ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** - ドメインの定義解除に加え、すべての関連付けられたストレージボリュームが削除されます。
- **--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。

26.8.6. ゲスト仮想マシンの再開

virsh を **resume** オプションと併用して、サスペンド中のゲスト仮想マシンを復元します。

```
# virsh resume {domain-id, domain-name or domain-uuid}
```

この操作は即時に行なわれるもので、ゲスト仮想マシンのパラメーターは **suspend** および **resume** 操作のために保管されます。

26.8.7. ゲスト仮想マシンの保存

virsh コマンドを使用して、ゲスト仮想マシンの現在の状態をファイルに保存します。

```
# virsh save {domain-name|domain-id|domain-uuid} state-file --bypass-
cache --xml --running --paused --verbose
```

これにより、指定したゲスト仮想マシンが停止し、データがファイルに保存されます。これにはゲスト仮想マシンで使用しているメモリー容量に応じて少々の時間がかかる場合があります。ゲスト仮想マシンの状態を復元するには、**restore** オプションを使用します (「ゲスト仮想マシンの復元」)。保存は一時停止に似ていますが、ゲスト仮想マシンを一時停止にする代わりに、ゲスト仮想マシンの現在の状態を保存します。

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、変更にドメイン 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** - ドメイ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オーバーライドします。
- **--paused**- ドメイ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オーバーライドします。
- **--verbose** - 保存の進捗を表示します。

ゲスト仮想マシンを XML ファイルから直接復元する必要がある場合、**virsh restore** コマンドがこれを行います。**domjobinfo** でプロセスを監視したり、**domjobabort** でプロセスをキャンセルしたりすることができます。

26.8.8. ゲストの復元に使用されるドメイン XML ファイルの更新

virsh save-image-define *file xml* **--running|--paused** コマンドは、後で指定ファイルが **virsh restore** コマンドの実行時に使用される際に使用されるドメイン XML ファイルを更新します。*xml* 引数は、代替 XML を含む XML ファイル名でなければなりません。この場合、ドメイン XML のホスト物理マシンに固有の部分にのみ変更が加えられます。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。保存イメージは、ドメインが実行中または一時停止の状態に復元されるかどうかを記録します。引数の **--running** または **--paused** を使用すると、使用される状態が決定されます。

26.8.9. ドメイン XML ファイルの抽出

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ドメイン 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、セキュリティーの機密情報がファイルに組み込まれます。

26.8.10. ドメイン XML 設定ファイルの編集

save-image-edit *file* --running --paused コマンドは、**virsh save** コマンドによって作成された保存済みの *file* に関連付けられた XML 設定ファイルを編集します。

保存イメージはドメインが **--running** または **--paused** 状態に復元されるかどうかを記録します。これらの引数を使用しない場合、状態はファイル自体で決定されます。**--running** または **--paused** を選択することにより、**virsh restore** が使用するはずの状態を上書きすることができます。

26.8.11. ゲスト仮想マシンの復元

virsh を使用して、**virsh save** コマンドを使用して以前に保存したゲスト仮想マシン (「ゲスト仮想マシンの保存」) を復元します。

```
# virsh restore state-file
```

これにより、保存していたゲスト仮想マシンが再起動します。これには少々時間がかかる可能性があります。ゲスト仮想マシンの名前と UUID は保管されていますが、新規 ID で割り当てられます。

virsh restore *state-file* コマンドは以下の引数を取ることができます。

- **--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** - ドメイ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オーバーライドします。
- **--paused**- ドメイ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オーバーライドします。

26.9. ゲスト仮想マシンのシャットダウン、再起動および強制終了

26.9.1. ゲスト仮想マシンのシャットダウン

virsh shutdown コマンドを使用してゲスト仮想マシンをシャットダウンします。

```
# virsh shutdown {domain-id, domain-name or domain-uuid} [--mode method]
```

ゲスト仮想マシンの設定ファイルにある **on_shutdown** パラメーターを変更して、再起動中のゲスト仮想マシンの動作を制御することができます。

26.9.2. Red Hat Enterprise Linux 7 ホスト上の 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* 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 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** コマンドで正しく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26.4 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. **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。

   ```
   # systemctl enable acpid
   # 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、*$guestname* の代わりにゲストの名前を使用します。

   ```
   <channel type='unix'>
      <source mode='bind'
   path='/var/lib/libvirt/qemu/{$guestname}.agent'/>
   ```

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

図26.2 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。**

   QEMU ゲストエージェント (QEMU-GA) をインストールし、14章*QEMU ゲストエージェント*で指示されるようにサービスを起動します。この章では、Windows ゲストを実行している場合の方法についても記載しています。

5. **ゲストをシャットダウンします。**

   a. 以下のコマンドを実行します。

   ```
   # virsh list --all  - this command lists all of the known
   domains
      Id Name                 State
   ----------------------------------
      rhel7                  running
   ```

   b. ゲスト仮想マシンをシャットダウンします。

   ```
   # virsh shutdown rhel7

   Domain rhel7 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。

   ```
   # virsh list --all
    Id Name                  State
   ----------------------------------
     . rhel7                 shut off
   ```

   d. 編集した XML ファイルを使用して、*rhel7* という名前のドメインを起動します。

   ```
   # virsh start rhel7
   ```

   e. *rhel7* 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 rhel7
   ```

   f. すべてのドメインを再度一覧表示します。*rhel6* がまだ一覧表示されており、その電源が切れていることを示しているはずです。

   ```
   # virsh list --all
      Id Name                   State
   ----------------------------------
      rhel7                  shut off
   ```

   g. 編集した XML ファイルを使用して、*rhel7* という名前のドメインを起動します。

```
# virsh start rhel7
```

h. *rhel7* 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 rhel7
```

i. ドメインを一覧表示します。*rhel7* がまだ一覧表示されており、電源が切れていることを示しているはずです。

```
# virsh list --all
   Id Name                 State
----------------------------------
   rhel7                shut off
```

ゲスト仮想マシンは、連続するシャットダウンにおいて、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-guest* 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、「libvirt-guests 設定の設定内容の操作」を参照してください。

26.9.3. libvirt-guests 設定の設定内容の操作

libvirt-guests サービスには、ゲストが適切にシャットダウンされることを保証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。ゲストの電源を切る必要がある場合、*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26.5 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:///'
#URIS=default

# action taken on host boot
# - start   all guests which were running on shutdown are started
on boot
#           regardless on their autostart settings
 (1)
# - ignore  libvirt-guests init script won't start any guest on
boot, however,      (2)
```

```
#           guests marked as autostart will still be automatically
started by       3
#           libvirtd
 4
#ON_BOOT=start
 5
 6
# Number of seconds to wait between each guest start. Set to 0 to
allow             7
# parallel startup.
#START_DELAY=0

# action taken on host shutdown
# - suspend   all running guests are suspended using virsh
managedsave
# - 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.
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
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).
#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.
#BYPASS_CACHE=0
```

1. ***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。
2. ***ON_BOOT*** - ホストの起動時にゲストに対して/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、自動起動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、自動起動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。
3. ***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。
4. ***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。
5. ***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。
6. 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。
7. ***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* サービスを開始します。サービスの再起動により、実行中のすべてのドメインがシャットダウンする可能性があるので実行しないでください。

26.9.4. ゲスト仮想マシンの再起動

virsh reboot コマンドを使用してゲスト仮想マシンを再起動します。再起動が実行されると、このプロントは返されますが、その時点からドメインが実際に再起動するまでには一定の時間差が生じる場合があります。

```
#virsh reboot {domain-id, domain-name or domain-uuid} [--mode method]
```

再起動中のゲスト仮想マシンの動作は、ゲスト仮想マシンの設定ファイルの **on_reboot** 要素を変更することにより、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選択します。代替方法を指定する場合、**--mode** 引数で、**initctl**、**acpi**、**agent**、**signal** が含まれる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。

26.9.5. ゲスト仮想マシンの強制終了

ゲスト仮想マシンを **virsh destroy** コマンドで停止するには、以下のようにします。

```
# virsh destroy {domain-id, domain-name or domain-uuid} [--graceful]
```

このコマンドでは、正常なシャットダウンで行なわれる終了プロセスを経ずに、指定されるゲスト仮想マシンをただちにシャットダウンして停止させます。**virsh destroy** 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。**destroy** オプションは、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。正常なシャットダウンを開始する必要がある場合は、**virsh destroy --graceful** コマンドを使用します。

26.9.6. 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、ただちにドメインをリセットします。リセットは、マシンの電源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場合、データが失われるリスクが発生します。

26.10. ゲスト仮想マシン情報の取得

26.10.1. ゲスト仮想マシンのドメイン ID の取得

ゲスト仮想マシンのドメイン ID を取得するには、以下のようにします。

```
# virsh domid {domain-name or domain-uuid}
```

26.10.2. ゲスト仮想マシンのドメイン名の取得

ゲスト仮想マシンのドメイン名を取得するには、以下のようにします。

```
# virsh domname {domain-id or domain-uuid}
```

26.10.3. ゲスト仮想マシンの UUID の取得

ゲスト仮想マシンの UUID (Universally Unique Identifier) を取得するには、以下のようにします。

```
# virsh domuuid {domain-id or domain-name}
```

virsh domuuid の出力例:

```
# virsh domuuid r5b2-mySQL01
4a4c59a7-ee3f-c781-96e4-288f2862f011
```

26.10.4. ゲスト仮想マシン情報の表示

ゲスト仮想マシンのドメイン ID、ドメイン名、または UUID と **virsh** を併用すると、指定したゲスト仮想マシンの情報を表示することができます。

```
# virsh dominfo {domain-id, domain-name or domain-uuid}
```

以下に **virsh dominfo** 出力例を示します。

```
# virsh dominfo vr-rhel6u1-x86_64-kvm
Id:             9
Name:           vr-rhel6u1-x86_64-kvm
UUID:           a03093a1-5da6-a2a2-3baf-a845db2f10b9
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       21.6s
Max memory:     2097152 kB
Used memory:    1025000 kB
Persistent:     yes
Autostart:      disable
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c612,c921 (permissive)
```

26.11. ストレージプールコマンド

以下のコマンドはストレージプールを操作します。*libvirt* を使用して、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。この機能についてさらに詳しくは、libvirt.org を参照してください。ストレージプールのコマンドの多くは、ドメインに使用されるコマンドに似ています。

26.11.1. ストレージプール XML の検索

find-storage-pool-sources *type srcSpec* コマンドは、確認できる所定の*type* 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。*srcSpec* が指定される場合、これはプールの照会をさらに制限する XML を含むファイルになります。

find-storage-pool-sources-as *type host port initiator* は、確認できる所定の *type* 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。*host*、*port* または *initiator*が指定される場合、それらは照会が実行される場所を制御します。

pool-info *pool-or-uuid* 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。

pool-list --inactive --all --persistent --transient --autostart --no-autostart --details<type> コマンドは、*libvirt* が認識す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、すべてのストレージプールが一覧表示されます。

上記の引数のほかにも、一覧表示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 を必要とするすべてのストレージプールコマンドの場合、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**、**rbd**、および **sheepdog** が含まれます。

--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、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。

pool-refresh *pool-or-uuid* は、プールに含まれるボリュームの一覧表示を最新の状態にします。

26.11.2. ストレージプールの作成、定義、および起動

26.11.2.1. ストレージプールの構築

pool-build *pool-or-uuid* --overwrite --no-overwrite コマンドは、*プール名または UUID* が指定されたプールを構築します。引数の **--overwrite** および **--no-overwrite** は、タイプがファイルシステムの場合にのみ使用できます。いずれの引数も指定されない場合、プールはタイプがファイルシステムのプールになり、結果としてビルドによりディレクトリーのみが作成されます。

--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。エラーが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。

26.11.2.2. XML ファイルからのストレージプールの作成および定義

pool-create *file* は、関連付けられた XML ファイルからストレージプールを作成し、これを起動します。

pool-define *file* は、XML *file* 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。

26.11.2.3. raw パラメーターからのストレージプールの作成および起動

pool-create-as *name* --print-xml type source-host source-path source-dev source-name <target> --source-format <format> コマンドは、指定される raw パラメーターからプールオブジェクトの名前を作成し、これを起動します。

--print-xml が指定される場合、プールを作成せずに、ストレージ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、プールには、プールの構築のためにタイプが必要になります。*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**、**rbd**、および **sheepdog** が含まれます。

pool-define-as *name* --print-xml type source-host source-path source-dev source-name <target> --source-format <format> コマンドは、指定される raw 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、これを起動しません。

--print-xml が指定される場合、プールを定義せずに、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、プールには指定されたタイプが必要になります。*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**、**rbd**、および **sheepdog** が含ま

れます。

pool-start *pool-or-uuid* は、以前に定義されているものの、アクティブでない指定されたストレージプールを開始します。

26.11.2.4. ストレージの自動起動

pool-autostart *pool-or-uuid* --disable コマンドは、起動時のストレージの自動起動を有効または無効にします。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、**--disable** 引数を使用します。

26.11.3. ストレージプールの停止および削除

pool-destroy *pool-or-uuid* はストレージプールを停止します。いったん停止すると、*libvirt* はプールを管理しなくなりますが、プールに含まれる raw データは変更されませんが、後で **pool-create** コマンドを使って復旧できます。

pool-delete *pool-or-uuid* 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。この操作は修復不能で、元に戻せないことを留意するのは重要です。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を許可できます。

pool-undefine *pool-or-uuid* コマンドは、アクティブでないプール設定の定義を解除します。

26.11.4. プール用の XML ダンプファイルの作成

pool-dumpxml --inactive *pool-or-uuid* 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。**--inactive** を使用すると、現在のプール設定とは対照的に、プールの次回の開始時に使用される設定がダンプされます。

26.11.5. ストレージプールの設定ファイルの編集

pool-edit *pool-or-uuid* は、編集用に指定されたストレージプールの XML 設定ファイルを開きます。

この方法は、適用前にエラーのチェックを行うため、XML 設定ファイルの編集用に使用できる唯一の方法になります。

26.11.6. ストレージプールの変換

pool-name *uuid* コマンドは、指定された UUID をプール名に変換します。

pool-uuid *pool* コマンドは、指定されたプールの UUID を返します。

26.12. 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、管理するためのすべてのコマンドについて説明します。いったんストレージプールを作成すると、ストレージプール名または UUID が必要になるため、ストレージプールの作成時にこれらのコマンドを実行するのが最善と言えるでしょう。ストレージプールについてさらに詳しくは、16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さらに詳しくは、17章*ストレージボリューム* を参照してください。

26.12.1. ストレージボリュームの作成

vol-create-from *pool-or-uuid file* --inputpool *pool-or-uuid vol-name-or-key-or-path* は、別のボリュームを入力に使用してボリュームを作成します。このコマンドには、ボリュームの作成に使用するストレージプールの名前または UUID となる *pool-or-uuid* が必要です。

file 引数は、ボリューム定義が含まれる XML ファイルとパスを指定します。**--inputpool *pool-or-uuid*** 引数は、ソースボリュームが入るストレージプールの名前または UUID を指定します。*vol-name-or-key-or-path* 引数は、ソースボリュームの名前またはキー、あるいはパスを指定します。いくつかの例については、「ボリュームの作成」を参照してください。

vol-create-as コマンドは、一連の引数からボリュームを作成します。*pool-or-uuid* 引数には、ボリュームの作成に使用するストレージプールの名前または UUID が含まれます。

```
vol-create-as pool-or-uuid name capacity --allocation <size> --format
<string> --backing-vol <vol-name-or-key-or-path> --backing-vol-format
<string>
```

name は新規ボリュームの名前です。*capacity* は作成されるボリュームのサイズであり、サフィックスがない場合はデフォルトでバイトになる単位付き整数で指定されます。**--allocation <size>** は、ボリュームに割り当てられる初期サイズであり、これもデフォルトでバイトになる単位付き整数で指定されます。**--format <string>** は、コンマ区切りの許容される形式の文字列からなるボリュームファイル形式を指定するためにファイルベースのストレージプールで使用されます。許容される形式には、**raw**、**bochs**、**qcow**、**qcow2**、**vmdk** が含まれ、**--backing-vol *vol-name-or-key-or-path*** は、既存ボリュームのスナップショットを取る場合に使用されるソースのバッキングボリュームです。**--backing-vol-format *string*** は、コンマで区切られる形式の文字列であるスナップショットのバッキングボリュームの形式です。許可される値には、**raw**、**bochs**、**qcow**、**qcow2**、**vmdk**、および **host_device** 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場合のみ使用されることが意図されています。デフォルトでは、使用される 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必要がある場合は、「ターゲット要素の設定」を参照してください。

26.12.1.1. XML ファイルからのストレージボリュームの作成

vol-create *pool-or-uuid file* は、XML *file*からストレージボリュームを作成します。このコマンドにも、ボリュームの作成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-or-uuid* が必要になります。*file* 引数には、ボリューム定義の XML ファイルと共にパスが含まれます。XML ファイルを作成する簡単な方法は、**vol-dumpxml** コマンドを使用して事前に存在するボリュームの定義を取得する方法です。

```
virsh vol-dumpxml --pool storagepool1 appvolume1 > newvolume.xml
virsh edit newvolume.xml
virsh vol-create differentstoragepool newvolume.xml
```

26.12.1.2. ストレージボリュームのクローン作成

vol-clone --pool *pool-or-uuid vol-name-or-key-or-path name* コマンドは、既存のストレージボリュームをクローン作成します。**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る方法ではありません。**--pool *pool-or-uuid*** 引数は、ボリュームの作成に使用されるストレージプールの名前または UUID です。*vol-name-or-key-or-path* 引数は、ソースボリュームの名前またはキー、あるいはパスです。*name* 引数を使用すると、新規ボリュームの名前が参照されます。追加の例については、「ボリュームのクローン作成」を参照してください。

26.12.2. ストレージボリュームの削除

vol-delete --pool *pool-or-uuid vol-name-or-key-or-path* コマンドは、所定ボリュームを削除します。このコマンドには、ボリュームに使用するストレージプールの名前または UUID である特定の **--pool *pool-or-uuid*** が必要です。オプションの *vol-name-or-key-or-path* は、削除するボリュームの名前またはキー、あるいはパスです。

vol-wipe --pool *pool-or-uuid* --algorithm *algorithm vol-name-or-key-or-path* コマンドはボリュームを完全に消去し、ボリューム上にあったデータを今後読み取り用にアクセスすることができないようにします。コマンドには、ボリュームに使用す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。*vol-name-or-key-or-path* には、完全に消去するボリュームの名前またはキー、あるいはパスが含まれます。引数の **--algorithm** を使用し、また以下のサポートされる *algorithm* タイプのいずれかを使用することにより、ゼロからボリューム再作成する代わりに、さまざまな完全消去のアルゴリズムを選択できることに注意してください。

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。

- **zero** - 1-pass all zeroes
- **nnsa** - 4-pass NNSA Policy Letter NAP-14.1-C (XVI-8) for sanitizing removable and non-removable hard disks: random x2, 0x00, verify.
- **dod** - 4-pass DoD 5220.22-M section 8-306 procedure for sanitizing removeable and non-removeable rigid disks: random, 0x00, 0xff, verify.
- **bsi** - 9-pass method recommended by the German Center of Security in Information Technologies (http://www.bsi.bund.de): 0xff, 0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f.
- **gutmann** - The canonical 35-pass sequence described in Gutmann's paper.
- **schneier** - 7-pass method described by Bruce Schneier in "Applied Cryptography" (1996): 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.

26.12.3. ストレージボリューム情報の XML ファイルへのダンプ

vol-dumpxml --pool *pool-or-uuid vol-name-or-key-or-path* コマンドは、ボリューム情報を指定されたファイルへの XML ダンプとして取得します。

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。*vol-name-or-key-or-path* は、結果として作成される XML ファイルを置くボリュームの名前、キーまたはパスです。

26.12.4. ボリューム情報の一覧表示

vol-info --pool *pool-or-uuid vol-name-or-key-or-path* コマンドは、所定のストレージボリューム **--pool** についての基本情報を一覧表示します。ここで、*pool-or-uuid* はボリュームに使用するストレージプールの名前または UUID です。*vol-name-or-key-or-path* は、返す情報の対象となるボリュームの名前またはキー、あるいはパスになります。

vol-list--pool *pool-or-uuid* --details は、指定されたストレージプールに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、ストレージボリュームの名前または UUID である **--pool *pool-or-uuid*** が必要になります。**--details** オプションは、*virsh* に対し、ボリュームタイプおよび容量に関する情報を追加で表示するよう指示します (ある場合)。

26.12.5. ストレージボリューム情報の取得

vol-pool --uuid *vol-key-or-path* コマンドは、所定の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、プール名は返されます。**--uuid** オプションが指定される場合、プールの UUID が代わりに返されます。このコマンドには、要求される情報を返す際に使用されるボリュームのキーまたはパスである *vol-key-or-path* が必要になります。

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* は名前を返すの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のに使用されるボリュームの名前またはパスです。

26.12.6. 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

このセクションでは、ストレージボリュームに対して情報のアップロードおよびダウンロードのやり取りを行う方法について説明します。

26.12.6.1. コンテンツのストレージボリュームへのアップロード

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* のコンテンツ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名前または UUID である **--pool *pool-or-uuid*** が必要です。さらに、完全消去するボリュームの名前またはキー、あるいはパスである *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 *長さ*** 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。

26.12.6.2. ストレージボリュームからのコンテンツのダウンロード

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 local-file のコンテンツをダウンロードします。

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、完全消去するボリュームの名前またはパスである*vol-name-or-key-or-path* が必要です。引数の **--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。

26.12.7. 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またはキー、あるいはパスである *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、「blockresize を使用したドメインパスのサイズ変更」 を参照してください。

26.13. ゲスト仮想マシン別の情報の表示

26.13.1. ゲスト仮想マシンの表示

virsh を使って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
```

以下のようなオプションも使用できます。

- **--inactive** オプション: アクティブではないゲスト仮想マシン (つまり、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。
- **--all** オプション: すべてのゲスト仮想マシンを一覧表示します。たとえば、以下のようになります。

```
# virsh list --all
 Id Name                 State
----------------------------------
  0 Domain-0             running
  1 Domain202            paused
  2 Domain010            inactive
  3 Domain9600           crashed
```

このコマンドを使うと確認できる状態が 7 種類あります。

- Running - **running** 状態は、CPU 上で現在稼働中のゲスト仮想マシンを示します。
- Idle - **idle** 状態は、ドメインが休止中のため、稼働していないか、または稼働できないことを示します。ドメインが IO を待機しているため (従来の待機状態) か、または作業がないためにスリープに入ったことが原因となります。
- Paused - **paused** 状態は、一時停止中のドメインを一覧表示します。これは、管理者が**virt-manager**、または **virsh suspend** で **pause** ボタンを使用した場合に発生します。ゲスト仮想マシンが一時停止になっている場合もメモリーと他のリソースを消費しますが、スケジューリングやハイパーバイザーからの CPU リソースの場合は無効になります。
- Shutdown - **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、ドメインが稼働していないことを示します。 ドメインが完全にシャットダウンしているか、または起動していないことが原因です。
- Crashed - **crashed** 状態は、ドメイ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。
- Dying - **dying** 状態のドメインは停止プロセスに入っています。これは、ドメインが完全にシャットダウンしていないか、またはクラッシュした状態です。

- **--managed-save** このフラグだけではドメインをフィルターしませんが、管理保存の状態が有効になったドメインを一覧表示します。ドメインを別個に一覧表示するには、**--inactive** フラグも使用する必要があります。
- **--name** は、指定されたドメイン名で、一覧に出力されます。**--uuid** が指定される場合、ドメインの UUID が代わりに出力されます。フラグの **--table** を使用すると、表形式の出力が使用されるよう指定されます。これらの 3 つのコマンドは相互に排他的です。
- **--title** このコマンドは **--table** 出力と共に使用する必要があります。**--title** により、追加の列が短いドメインの説明 (タイトル) と共に表に作成されます。
- **--persistent** は、永続的なドメインを一覧に組み込みます。**--transient** 引数を使用します。
- **--with-managed-save** は、管理保存と共に設定されたドメインを一覧表示します。管理保存なしてコマンドを一覧表示するには、コマンド **--without-managed-save** を使用します。
- **--state-running** は、実行中のドメインをフィルター処理します。一時停止のドメインについては**--state-paused**、オフにされているドメインについては**--state-shutoff** が使用されます。**--state-other** は、すべての状態を fallback として一覧表示します。
- **--autostart**: この引数は、自動起動のドメインを一覧表示します。この機能を無効にしてドメインを一覧表示するには、引数の **--no-autostart** を使用します。
- **--with-snapshot** は、スナップショットイメージを一覧表示できるドメインを一覧表示します。スナップショットなしでドメインをフィルター処理するには、引数の **--without-snapshot** を使用します。

```
$ virsh list --title --name

    Id        Name                                           State
Title
    0         Domain-0                                       running
Mailserver1
    2         rhelvm                                         paused
```

26.13.2. 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 rhel7
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
```

26.13.3. vCPU のホスト物理マシンの CPU へのピン設定

virsh vcpupin は、仮想 CPU を物理 CPU に割り当てます。

```
# virsh vcpupin rhel7
VCPU: CPU Affinity
----------------------------------
   0: 0-3
   1: 0-3
```

vcpupin は、以下の引数を取ることができます。

- **--vcpu** には vcpu 番号が必要です。
- **[--cpulist] >string<** は、照会するオプションを設定するか、または省略するために、ホスト物理マシンの CPU 番号を一覧表示します。
- **--config** は次回の起動に影響を与えます。
- **--live** は稼働中のドメインに影響を与えます。
- **--current** は現在のドメインに影響を与えます。

26.13.4. 所定ドメインの仮想 CPU 数についての情報の表示

virsh vcpucount には、*domain*名またはドメイン ID が必要です。

```
# virsh vcpucount rhel7
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、vcpus の上限 (cap) を取得します。
- **--active** は、現在アクティブな vcpus の数を取得します。
- **--live** は、稼働中のドメインから値を取得します。
- **--config** は、次回起動時に使用される値を取得します。
- **--current** は、現在のドメイン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。

26.13.5. 仮想 CPU 親和性の設定

物理 CPU と仮想 CPU の親和性を設定するには、以下のようにします。

```
# 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 パラメーターはゲスト仮想マシンに割り当てられた仮想 CPU の数を示します。**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** パラメーターは VCPU を実行する物理 CPU を指定します。

--config などの追加のパラメーターは次回の起動に影響を与えますが、**--live** は稼働中のドメインに、 **--current** は現在のドメインに影響を与えます。

26.13.6. 仮想 CPU 数の設定

このコマンドを使用して、ゲストドメイ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ドメインで機能します。非アクティブなゲストドメインの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、以下のようにします。

```
# virsh setvcpus {domain-name, domain-id or domain-uuid} count [[--
config] [--live] | [--current]]
```

例を以下に示します。

```
virsh setvcpus guestVM1 2 --live
```

vCPU を guestVM1 に割り当てる数を 2 つ増やします。このアクションは、guestVM1 の実行中に行われます。

同様に、同じ CPU をホットアンプラグするには、以下を実行します。

```
virsh setvcpus guestVM1 1 --live
```

値は、ホスト、ハイパーバイザー、またはゲストドメイ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、ドメインが準仮想化されている場合は稼働中のドメインの仮想 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ドメインの保存された XML 設定に対して行なわれ、ゲストドメイン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、ゲスト仮想マシンのドメイ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 が指定される場合、フラグは現在のゲスト仮想マシンの状態に影響を与えます。いずれのフラグも指定されていない場合、ゲスト仮想マシンドメインがアクティブでないと失敗する **--live** フラグが想定されます。この状況では、**--config** フラグも想定されるかどうかや、そのために XML 設定が変更を永続化するために調整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。

--maximum フラグは、ドメイ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、ゲスト仮想マシンの作成時にゲスト仮想マシンに割り当てられた CPU の数を超えることはできないことに留意してください。

26.13.7. メモリー割り当ての設定

virsh を使ってゲスト仮想マシンのメモリー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
```

```
# 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の作成時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る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 x 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 次回起動時に有効になります。
- --live 実行中のドメインのメモリーを制御します。
- --current 現在のドメイン上のメモリーを制御します。

26.13.8. ドメインのメモリー割り当ての変更

26.13.8. ドメインのメモリー割り当ての変更

virsh setmaxmem *domain size* --config --live --current 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 rhel7 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、実行中のドメインのメモリーを制御します。
- **--current** - 現在のドメイン上のメモリーを制御します。

26.13.9. ゲスト仮想マシンのブロックデバイス情報の表示

virsh domblkstat を使用して、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を表示できます。

```
# virsh domblkstat GuestName block-device
```

26.13.10. 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 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
```

26.14. 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*** — *network-name* で指定されたネットワークを自動的に起動します。
- **virsh net-create *XMLfile*** — 既存の XML ファイルを使って新規ネットワークを生成し、起動します。
- **virsh net-define *XMLfile*** — 既存の XML ファイルから新規のネットワークデバイスを生成しますが、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*** — 停止中のネットワークを起動します。
- **virsh net-undefine *nameOfInactiveNetwork*** — 停止中のネットワークの定義を削除します。

26.15. 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。

26.16. 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、ドメインの **<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、ドメイ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)。

26.16.1. 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。

26.16.2. ホストインターフェース用 XML 設定ファイルの編集

コマンド **iface-edit *interface*** 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、29章 *ドメイン XML の操作* を参照してください)。

26.16.3. アクティブなホストインターフェースの一覧表示

iface-list --inactive --all 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** が指定されている場合、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** が指定される場合、アクティブではないインターフェースのみが一覧表示されます。

26.16.4. MAC アドレスのインターフェース名への変換

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 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。

iface-mac *interface* コマンドは、この場合 *interface* がインターフェース名になりますが、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。

26.16.5. 特定ホスト物理マシンのインターフェースの停止

virsh iface-destroy *interface* コマンドは、ホストで **if-down** を実行する場合と同様に所定のホストインターフェースを破棄 (停止) します。このコマンドは、インターフェースがアクティブに使用されることを停止し、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 **iface-undefine *interface*** コマンドを使用します。

26.16.6. ホスト設定ファイルの表示

virsh iface-dumpxml *interface* --inactive 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、次回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。

26.16.7. ブリッジデバイスの作成

iface-bridge は、ブリッジデバイスが指定するブリッジを作成し、既存のネットワークデバイスインターフェースを新規の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を開始します。

```
# virsh iface-bridge interface bridge --no-stp delay --no-start
```

これらの設定は、--no-stp、--no-start、および遅延の秒数を表す整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すべての 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の詳細は、「ブリッジデバイスの破棄」を参照してください。

26.16.8. ブリッジデバイスの破棄

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に使用状態に戻し、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推奨されてい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。

26.16.9. インターフェーススナップショットの操作

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 (**iface-commit** を使用) されるか、または復元 (**iface-rollback** を使用) されます。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なわれる場合、未定義の動作が生じます。

iface-begin を実行してから行な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、**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。**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。

iface-rollback を使用して、**iface-begin** コマンドが実行された最終時を記録した状態に、すべてのホストインターフェース設定を戻します。**iface-begin** コマンドがそれまでに実行されていなかった場合、**iface-rollback** は失敗します。ホスト物理マシンの再起動は、暗黙的なロールバックポイントとしても機能する点に注意してください。

26.17. スナップショットの管理

以降のセクションでは、ドメイン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、指定された時点のドメインのディスク、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ドメイ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ー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 web サイト を参照してください。

重要

--live が記載されるすべての場合に、ライブスナップショットはRed Hat Enterprise Linux 7 でサポートされないものの、ゲスト仮想マシンの電源がオフにされている場合にスナップショットを作成できることに注意してください。ライブスナップショットの作成は、Red Hat Enterprise Virtualization で可能です。詳細は、サービス担当者にお問い合わせください。

26.17.1. スナップショットの作成

virsh snapshot create コマンドは、ドメインの XML ファイルで指定されたプロパティーでドメイ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。

```
#snapshot-create <domain> <xmlfile> [--redefine [--current] [--no-metadata] [--halt] [--disk-only] [--reuse-external] [--quiesce] [--atomic] [--live]
```

ドメイン名、ID、または UID は、ドメイン要件として使用することができます。XML 要件は、最低でも <name>, <description> および <disks> 要素が含まれる必要のある文字列です。

残りのオプションの引数は以下のようになります。

- **--disk-only** - 残りのフィールドを無視し、libvirt がこれらに自動入力させるようにします。
- 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** を混在させることにより、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ドメインを非アクティブな状態のままにします。**--halt** および **--disk-only** を混在させることにより、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が失われます。
- **--redefine** は、**snapshot-dumpxml** で生成されるすべての XML 要素が有効な場合、以下を実行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、途中でなくなり、後に同じ名前および UUID で再作成される一時的なドメイン用に階層を再作成する、または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ドメイン 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、ドメイ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せん。
- **--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** は、外部スナップショットが使用され、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ドメイ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** を渡すことも求められます。

- **--atomic** は、libvirt に対し、スナップショットが変更なしに成功するか、または失敗することを保証させます。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に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、**dumpxml** を使用する必要があります。

- **--live**。ライブスナップショット を参照してください。このオプションは、ゲストの実行中に libvirt にスナップショットを取得させます。これは、外部チェックポイントのメモリーイメージのサイズを拡大します。現在、これは外部チェックポイントにのみサポートされています。スナップショットのメタデータがあると、永続的なドメインの定義解除の試行が回避されます。ただし、一時的なドメインの場合、ドメイ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)。

26.17.2. 現在のドメインのスナップショットの作成

virsh snapshot-create-as domain コマンドは、ドメインのスナップショットを、ドメイン XML ファイルで指定されたプロパティーを使って作成します ((<name> 要素と <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] [[--live] [--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ドメイ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]** の形式で表わされます。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>, and <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>
```

- **--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ドメイ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、*libirt* 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** は、libvirt に対し、スナップショットが変更なしに成功するか、または失敗することを保証させます。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に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、**dumpxml** を使用する必要があります。

- **--live** ライブスナップショット を参照してください。このオプションは、ゲスト仮想マシンの実行中に libvirt にスナップショットを取得させます。これにより、外部チェックポイントのメモリーイメージのサイズが拡大します。現在これは、外部チェックポイントにのみサポートされています。

26.17.3. 現行ドメインのスナップショットの取得

このコマンドは、現在使用されているスナップショットの照会を行うために使用されます。これを使用するには、以下を実行します。

```
# virsh snapshot-current domain {[--name] | [--security-info] |
[snapshotname]}
```

snapshotname が使用されていない場合、ドメイ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、ドメインに戻さずに、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。

26.17.4. snapshot-edit-domain

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。これを使用するには、以下を実行します。

```
#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 が指定される場合、結果として生じる編集済みのファイルは異なるファイル名で保存されます。**--clone** が指定される場合、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のファイル名からアクセス可能であるため、スナップショット名の変更は注意して行う必要があります。

26.17.5. snapshot-info-domain

snapshot-info-domain は、スナップショットについての情報を表示します。これを使用するには、以下を実行します。

```
# 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。

26.17.6. snapshot-list-domain

所定ドメイ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、作成時およびドメイ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]
```

残りのオプションの引数は以下のようになります。

- **--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。これらの 3 つのオプションは相互に排他的です。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** との互換性がありません。**--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** との互換性がありません。

- **--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、永続的なドメインの定義解除が回避されるか、または一覧が一時的なドメイ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、一覧はドメイ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、一覧はドメイ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、一覧はドメイ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。

26.17.7. snapshot-dumpxml domain snapshot

virsh snapshot-dumpxml domain snapshot は、snapshot という名前のドメイ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、セキュリティー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、**snapshot-current** を使用します。

26.17.8. snapshot-parent domain

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}
```

26.17.9. snapshot-revert domain

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であ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。

```
# 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** フラグ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。

26.17.10. snapshot-delete domain

snapshot-delete domain 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 つのフラグは相互に排他的です。

--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。

26.18. ゲスト仮想マシンの CPU モデルの設定

26.18.1. 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使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。

26.18.2. 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ー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。x86 では、最新の CPU 機能は、CPUID インストラクションを介して公開されます。基本的に、これはそれぞれのビットが特殊な意味を持つ 32ビットの整数のセットに要約されます。AMD と Intel については、これらのビットに関する共通の意味で一致しています。その他のハイパーバイザーは、CPUID マスクの概念をそれらのゲスト仮想マシンの設定形式に直接公開します。しかし、QEMU/KVM は、単なる x86 アーキテクチャーよりも多くをサポートするため、CPUID は規準となる設定形式としては適切でないことは明らかです。QEMU は、最終的に CPU モデルの名前文字列と名前付きのフラグのセットを組み合わせるスキームを使用することになりました。x86 では、CPU モデルはベースライン CPUID マスクにマップして、フラグはそのマスク内でビットをオンとオフに切り替えるために使用できます。libvirt では、これに追従することが決定されており、モデル名とフラグの組み合わせを使用します。

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が、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。

26.18.3. 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26.3 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任意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...
```

図26.4 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26.5 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。

26.19. ゲスト仮想マシンの CPU モデルの設定

単純なデフォルト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 **cpu-baseline** virsh 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** という属性がありますが、以下の値を持つ可能性があります。

- 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。

- 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、その後に失敗します。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っ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。

26.20. ゲスト仮想マシンのリソースの管理

virsh は、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を行うことを許可します。これは *libvirt daemon* によって管理されます。*libvirt daemon* は *cgroups* を作成し、ゲスト仮想マシンの代わりにそれらを管理します。システム管理者は、行なわなければならない唯一のタスクとして、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いずれかを実行する必要があります。以下の調整可能パラメーターを使用できます。

- **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。

cgroup では、グループ階層の作成時にマウントポイントとディレクトリーのセットアップは管理者が完全に決定するようになるため、これはいくつかのマウントポイントを **/etc/fstab** に追加するだけの場合よりも複雑になります。ディレクトリー階層をセットアップし、その中にプロセスを配置する方法を判別することが必要です。これは、以下の virsh コマンドを使用して実行できます。

- **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。

26.21. スケジュールパラメーターの設定

schedinfo は、スケジューラーパラメーターをゲスト仮想マシンに渡すことを許可します。以下のコマンド形式を使用する必要があります。

```
#virsh schedinfo domain --set --weight --cap --current --config --live
```

それぞれのパラメーターを以下に説明します。

- **domain** - これは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。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、次回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。

例26.3 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26.4 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    : 100000
vcpu_quota     : -1
```

26.22. ディスク I/O スロットリング

virsh blkdeviotune 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、**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29.27「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。

26.23. ブロック I/O パラメーターの表示または設定

blkiotune 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] [[--config] [--live] | [--current]]
```

このコマンドについての詳細は、『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。

26.24. メモリーチューニングの設定

virsh memtune virtual_machine --parameter size 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。

26.25. 仮想ネットワークコマンド

以下のコマンドは複数の仮想ネットワークを操作します。*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 web サイト にあるドキュメントを参照してください。仮想ネットワークについて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。

26.25.1. 仮想ネットワークの自動起動

このコマンドは、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。このコマンドを実行するには、以下を実行します。

```
# 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。

26.25.2. XML ファイルによる仮想ネットワークの作成

このコマンドは、XML ファイルから仮想ネットワークを作成します。*libvirt* によって使用される XML ネットワーク形式の説明については、libvirt 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
```

26.25.3. XML ファイルによる仮想ネットワークの定義

このコマンドは、XML ファイルから仮想ネットワークを定義します。ネットワークが定義されますがインスタンス化は行なわれません。仮想ネットワークを定義するには、以下を実行します。

```
# net-define file
```

26.25.4. 仮想ネットワークの停止

このコマンドは、所定の仮想ネットワークをその名前または UUID 別に破棄 (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。

```
# net-destroy network
```

26.25.5. ダンプファイルの作成

このコマンドは、指定された仮想ネットワークの標準出力に対する XML ダンプとして仮想ネットワーク情報を出力します。**--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。ダンプファイルを作成するには、以下を実行します。

```
# virsh net-dumpxml network [--inactive]
```

26.25.6. 仮想ネットワークの XML 設定ファイルの編集

このコマンドは、ネットワークの XML 設定ファイルを編集します。これは以下と同等になります。

```
#virsh net-dumpxml --inactive network > network.xml
vi network.xml (or make changes with your other text editor)
virsh net-define network.xml
```

相違点は一部エラーチェックを実行する点になります。使用されるエディターは $VISUAL または $EDITOR 環境変数を使用して指定でき、デフォルトは "vi" になります。ネットワークを編集するには、以下を実行します。

```
#virsh net-edit network
```

26.25.7. 仮想ネットワークについての情報の取得

このコマンドは、*network* オブジェクトについての基本情報を返します。ネットワーク情報を得るには、以下を実行します。

```
# virsh net-info network
```

26.25.8. 仮想ネットワークについての情報の一覧表示

アクティブなネットワークの一覧を返します。**--all** が指定されている場合、これに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のフィルターは、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。

注意: 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。

```
# net-list [--inactive | --all] [--persistent] [<--transient>] [--
autostart] [<--no-autostart>]
```

26.25.9. ネットワーク UUID からネットワーク名への変換

このコマンドは、ネットワーク UUID をネットワーク名に変換します。これを実行するには、以下を実行します。

```
# virsh net-name network-UUID
```

26.25.10. 停止状態の（定義済み）ネットワークの起動

このコマンドは、(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。これを実行するには、以下を実行します。

```
# virsh net-start network
```

26.25.11. 停止状態のネットワークの設定の定義解除

このコマンドは、非アクティブなネットワーク設定の定義を解除します。これを実行するには、以下を実行します。

```
# net-undefine network
```

26.25.12. ネットワーク名からネットワーク UUID への変換

このコマンドは、ネットワーク名をネットワーク UUID に変換します。これを実行するには、以下を実行します。

```
# virsh net-uuid network-name
```

26.25.13. 既存のネットワーク定義ファイルの更新

このコマンドは、既存のネットワーク定義の所定のセクションを更新します。この更新にはネットワークの破棄および再起動を必要とせず、ただちに有効になります。このコマンドは、"add-first"、"add-last"、"add" (add-last と同意)、"delete"、または "modify" のいずれかになります。セクションは、""bridge"、"domain"、"ip"、"ip-dhcp-host"、"ip-dhcp-range"、"forward"、"forward-interface"、"forward-pf"、"portgroup"、"dns-host"、"dns-txt"、または "dns-srv" のいずれかとなり、それぞれのセクション名は変更対象の要素に関連する xml 要素階層の連結によって付けられます。たとえば、"ip-dhcp-host" はネットワークの **<ip>** 要素内の **<dhcp>** 要素内に含まれる **<host>** 要素を変更します。xml は変更されるタイプの詳細な XML 要素のテキスト (例: "<host mac="00:11:22:33:44:55' ip='1.2.3.4'/>") または詳細な xml 要素を含むファイルの名前のいずれかになります。不明瞭な部分があれば、指定されるテキストの最初の文字を参照して解消できます。最初の文字が "<" の場合は xml テキストであり、最初の文字が ">" ではない場合、それは使用される xml テキストが含まれるファイルの名前になります。**parent-index オプションは、要求された要素が含まれるいくつかの親要素のいずれかを指定するために使用されます (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>** 要素のいずれかに含まれる可能性があります。parent-index が指定されない場合、「最も適した」**<ip>** 要素が選択されますが (通常は **<dhcp>** 要素を含む唯一の要素)、***--parent***-index が指定される場合、**<ip>** の特定の例が修正されます。***--live*** が指定される場合、実行中のネットワークが影響を受けます。***--config*** が指定される場合は、永続的なネットワークの次の起動が影響を受けます。***-- current*** が指定される場合、現在のネットワークの状態が影響を受けます。***--live*** と ***--config*** フラグの両方を指定することができますが、--current は排他的です。フラグを指定しないことは、***--current*** を指定することと同じになります。

設定ファイルを更新するには、以下を実行します。

```
# virsh net-update network command section xml [--parent-index index]
[[--live] [--config] | [--current]]
```

第27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

27.1. はじめに

Red Hat Enterprise Linux 7 には、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複数のツールが同梱されています。これらのたツールは次のように使用することができます。

- 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
- Windows ゲスト仮想マシンの Windows Registry 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ー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、およびフロッピーディスクイメージの読み込み

警告

これらのツールを使用して、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ツールを使用しないか、または少なくともツールを **常に読み取り専用モード** で使用することを強くお勧めします。

注記

libguestfs とそのツールに関する基礎知識については Unix 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。virt ツールについては virt ツールの man ページをご覧ください (virt-df(1))。トラブルシューティング情報については、「libguestfs トラブルシューティング」 を参照してください。

27.1.1. 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 virt コマンドを使用すると、リモート libvirt 接続を指定できます。以下は例になります。

```
# virt-df -c  qemu://remote/system -d Guest
```

Red Hat Enterprise Linux 7 の *libguestfs* 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 NBD 上でリモートディスクソースにアクセスできます。ディスクイメージには、**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

27.2. 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が、かなり低いレベルのケースになります。
- **guestfish (GUEST Filesystem Interactive SHell)** はインテ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使用することができます。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作業を行う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。
- **hivex** および **augeas** は、それぞれ Windows Registry と Linux 設定ファイルを編集するためのライブラリーです。これらは libguestfs とは別のライブラリーですが、 libguestfs の値の多くはこれら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。

27.3. インストール

libguestfs、guestfish、libguestfs ツール、guestmount、および Windows ゲスト仮想マシン向けサポートをインストールするには、RHEL V2WIN チャンネルをサブスクライブします。Red Hat Web サイトに移動し、次のコマンドを実行します。

```
# yum install libguestfs guestfish libguestfs-tools libguestfs-mount
libguestfs-winsupport
```

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実行します。

```
# yum install '*guestf*'
```

27.4. guestfish シェル

guestfish はインテラクティブなシェルで、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実行します。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 libguestfs filesystem interactive shell for
editing virtual machine filesystems.

 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高速化します。

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。

27.4.1. guestfish を使用したファイルシステムの表示

27.4.1.1. 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
```

以下の出力では Windows のディスクイメージを表示しています。

```
><fs> run
><fs> list-filesystems
/dev/vda1: ntfs
/dev/vda2: ntfs
```

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 device

 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場合は、まずそれをマウントする必要があります。この例では、直前の出力にある Windows パーティションの 1 つを使用しています (**/dev/vda2**)。このパーティションは **C:** ドライブに相応します。

```
><fs> mount-ro /dev/vda2 /
><fs> ll /
total 1834753
 drwxrwxrwx  1 root root       4096 Nov  1 11:40 .
 drwxr-xr-x 21 root root       4096 Nov 16 21:45 ..
 lrwxrwxrwx  2 root root         60 Jul 14  2009 Documents and Settings
 drwxrwxrwx  1 root root       4096 Nov 15 18:00 Program Files
 drwxrwxrwx  1 root root       4096 Sep 19 10:34 Users
 drwxrwxrwx  1 root root      16384 Sep 19 10:34 Windows
```

ファイルやディレクトリー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ー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リー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。

27.4.1.2. 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 libguestfs filesystem interactive shell for
 editing virtual machine filesystems.

 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
 [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 や Windows のゲスト仮想マシンの多くで正常に動作します。

27.4.1.3. 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
```

27.4.2. 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。URI がある場合は以下のコマンドを使用します。

```
# guestfish -a  ssh://root@example.com/disk.img
```

URI に使用される形式は、ファイルの名前が disk.img である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:///path/to/disk.img**
- **guestfish -a ftp://[user@]example.com[:port]/disk.img**
- **guestfish -a ftps://[user@]example.com[:port]/disk.img**
- **guestfish -a http://[user@]example.com[:port]/disk.img**
- **guestfish -a https://[user@]example.com[:port]/disk.img**
- **guestfish -a tftp://[user@]example.com[:port]/disk.img**

27.4.3. 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 libguestfs filesystem interactive shell for
 editing virtual machine filesystems.

 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ー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。

27.4.4. 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。

27.4.5. 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 (その日のメッセージ) 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
```

27.4.6. 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 で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ー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。

27.5. 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
  ```

27.6. virt-rescue: The rescue shell

27.6.1. 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。特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。

27.6.2. 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。

27.7. virt-df: ディスク使用量の監視

27.7.1. 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。

27.7.2. 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 を単体で使用して全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 (つまり libvirt が認識できる全ゲスト仮想マシン)。**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 は Windows ゲスト仮想マシンでも動作します。

```
# 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%
 Win7x32:/dev/sda1                 100.0M      24.1M      75.9M   25%
 Win7x32:/dev/sda2      19.9G       7.4G      12.5G   38%
```

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は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、分析や監視を行な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 WindowsGuest
 Virtual Machine,Filesystem,1K-blocks,Used,Available,Use%
 Win7x32,/dev/sda1,102396,24712,77684,24.1%
 Win7x32,/dev/sda2,20866940,7786652,13080288,37.3%
```

傾向についての情報を生成したり、警報を発するための出力の処理方法についての詳細は次の URL をご覧ください (http://virt-tools.org/learning/advanced-virt-df/)。

27.8. virt-resize: オフラインのゲスト仮想マシンのサイズ変更

27.8. virt-resize: オフラインのゲスト仮想マシンのサイズ変更

27.8.1. 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、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。

27.8.2. 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/LogVol00  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
```

ここでは、以下を実行する方法を示します。

- 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/LogVol00** を拡大して 2 番目のパーティションの新領域を埋める

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** のいずれかまたは両方を使用します。

```
# 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。

27.9. virt-inspector: ゲスト仮想マシンの検査

27.9.1. 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。

27.9.2. 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 libguestfs-devel
```

Windows ゲスト仮想マシンを処理する場合は、*libguestfs-winsupport* もインストールする必要があります。詳細は、「インストール」を参照してください。XML 出力例や出力用の Relax-NG スキーマなどに関するドキュメントは **/usr/share/doc/libguestfs-devel-*/** にインストールされます (「*」は libguestfs のバージョン番号に置き換わります)。

27.9.3. 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 --xml -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 --xml -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 <operatingsytems> 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 にはシンプル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は、 XPath ライブラリーに合わせて使い慣れているプログラミング言語の使用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 --xml 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 --xml GuestName | xpath //application/name
 [...long list...]
```

27.10. virt-win-reg: Windows レジストリーの読み込みおよび編集

27.10.1. はじめに

virt-win-reg は、Windows ゲスト仮想マシンのレジストリーを操作するツールです。これは、Windows ゲスト仮想マシンのレジストリーのキーを読み込み、変更するために使用することができます。

警告

virt-win-reg を使用してライブまたは実行中のゲスト仮想マシンに変更をマージしたり、書き込んだり **しないでください**。これを実行すると、ディスクが破損します。**いつでも** ゲストをまず停止するようにしてください。

27.10.2. インストール

virt-win-reg をインストールするには、**root** ユーザーとして以下の **yum** コマンドを実行します。

```
# yum install libguestfs-tools libguestfs-winsupport
```

27.10.3. virt-win-reg の使用

レジストリーキーを読み込むには、変更するゲストの名前を指定する必要があります。このゲストはローカルにもリモートにもすることができ、そのディスクイメージファイルを指定することもできます。また、必要なレジストリーキーへの完全パスを指定します。必要なレギストリーキーの名前の周りには一重引用符を使用する必要があります。ゲスト (ローカルおよびリモート) やゲストイメージファイルについて以下のコマンド例を参照してください。これらは **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall** のサンプルキーに対し、また **より少ない** ページャーに (オプションで) パイプされた出力と共に使用されています。

```
# virt-win-reg WindowsGuest \

'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall'
| less
```

```
# virt-win-reg -c qemu+ssh://host_ip/system WindowsGuest \

'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall'
| less
```

```
# virt-win-reg WindowsGuest.qcow2 \

'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall'
| less
```

出力には、Windows 上で **.reg** ファイルで使用される同じテキストベースの形式が使用されます。

注記

この形式は文字列のポータブルなエンコーディングを適切に定義しないため、 文字列には Hex-quoting が使用されます。**.reg** ファイルを 1 つのマシンから別のマシンに移す場合に再現性を確保できるのはこの方法だけです。

virt-win-reg の出力を次の簡単な Perl スクリプトでパイプすると Hex-quoting が実行された文字列を出力できるようになります。

```
perl -MEncode -pe's?hex\((\d+)\):(\S+)?
$t=$1;$_=$2;s,\,,,g;"str($t):\"".decode(utf16le=>pack("H*",$_))."\""
?eg'
```

警告

Windows レジストリーを変更することには内在的なリスクが存在します。ターゲットゲストがオフラインであり、信頼できるバックアップがあることを確認してください。

オフラインのゲスト仮想マシンの Windows レジストリーへの変更をマージするには、まず変更内容が含まれる適切な **.reg** ファイルを用意する必要があります。このファイル形式についての詳細情報は、こちらの Microsoft KB 記事 を参照してください。**.reg** ファイルの準備後には、ローカル、リモートおよびローカルイメージファイルのオプションを示す以下の例に類似するコマンドを使用します。

```
# virt-win-reg --merge WindowsGuest changes.reg
```

```
# virt-win-reg -c qemu+ssh://host_ip/system WindowsGuest --merge
changes.reg
```

```
# virt-win-reg --merge WindowsGuest.qcow2 changes.reg
```

27.11. 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 ＡＣ 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。

27.11.1. 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の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。

27.12. virt-sysprep の使用

virt-sysprep コマンドラインは、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。このプロセスのステップには、SSH ホストキーの削除、永続的

なネットワーク MAC 設定の削除、およびユーザーアカウントの削除が含まれます。また、Virt-sysprep は、たとえば SSH キー、ユーザーまたはロゴを追加することにより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。sysprep という用語は、Microsoft Windows システムと共に使用される「System Preparation」ツール (sysprep.exe) に由来します。名前自体は間接的に Microsoft に由来しますが、このツールは Windows ゲストでは現在機能しないため、Windoes ゲストで対応されていると想定しないでください。

virt-sysprep は、実行中のゲストまたはディスクイメージを変更します。*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。ゲスト仮想マシンの既存のコンテンツを保持したい場合は、まずディスクのスナップショットを取り、このコピーまたはクローン作成を行なってください。詳細は、libguestfs.org を参照してください。

root として *virt-sysprep* を実行する必要はありません。実際は実行しないことを推奨します。root としてこれを実行する唯一の状況としては、ディスクイメージにアクセスするために root が必要となる場合がありますが、この場合でも、root 以外のユーザーが *virt-sysprep* を実行してディスクイメージを書き込み可能にできるようにアクセス権を変更する方が良いでしょう。

virt-sysprep をインストールするには、以下のコマンドを実行します。

```
$ sudo yum install virt-sysprep
```

virt-sysprep で使用できるコマンドーを以下の表に示します。

表27.1 *virt-sysprep* commands

コマンド	説明	例
--help	特定のコマンドまたはパッケージ全体についての簡潔なヘルプエントリーを表示します。詳細のヘルプについては、MAN ページを参照してください。	**$ virt-sysprep --help**
-a [*file*] or --add [*file*]	ゲスト仮想マシンからディスクイメージとなるはずの指定された *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、**--format** オプションを使用します。	**$ virt-sysprep --add /dev/vms/disk.img**
-a [*URI*] or --add [*URI*]	リモートディスクを追加します。URI 形式は guestfish と互換性があります。詳細は、「guestfish を使用したファイルの追加」を参照してください。	**$ virt-sysprep -a http://[user@]example.com[:port]/disk.img**
-c [*URI*] or --connect [*URI*]	*libvirt* 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 (**virt-sysprep -a**)、*libvirt* は一切使用されません。	**$ virt-sysprep -c qemu:///system**

コマンド	説明	例
-d [*guest*] or --domain [*guest*]	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。	$ **virt-sysprep --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***
-n or --dry-run	ゲスト仮想マシンに読み取り専用の「dry run」を実行します。これは sysprep 操作を実行しますが、最後にはすべての変更をディスクにスローします。	$ **virt-sysprep -n**
--enable [*operations*]	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、--list コマンドを使用します。	>$ **virt-sysprep --enable ssh-hotkeys,udev-persistent-net**
--operation or --operations	実行する sysprep 操作を選択します。操作を無効にするには、操作名の前に - を使用します。	>$ **virt-sysprep --operations ssh-hotkeys,udev-persistent-net** は両方の操作を有効にしますが、$ **virt-sysprep --operations firewall-rules,-tmp-files** は firewall-rules 操作を有効にし、tmp-files 操作を無効にします。有効な操作の一覧については、libvirt.org を参照してください。
--format [**raw**|**qcow2**|**auto**]	-a 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 -a オプションのディスク形式が強制実行されます。--format を使用すると、後続の -a オプションの自動検出に自動的に切り替わります (上記 -a コマンドを参照)。	$ **virt-sysprep --format raw -a *disk.img*** は disk.img の raw 形式 (自動検出なし) を強制しますが、**virt-sysprep --format raw -a *disk.img* --format auto -a another.img** は **disk.img** の raw 形式 (自動検出なし) を強制実行してから **another.img** 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ー上の問題を防ぐことができます。

コマンド	説明	例

--list-operations	virt-sysprep 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、**--enable** 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 * 文字が入り、有効にされていない場合はブランクになります。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。	`$ virt-sysprep --list-operations`
		`bash-history * Remove the bash history in the guest`
		`cron-spool * Remove user at-jobs and cron-jobs`
		`dhcp-client-state * Remove DHCP client leases`
		`dhcp-server-state * Remove DHCP server leases`
		`...`

--mount-options	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。mountpoint:options のペアのセミコロン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。	--mount-options "/:noatime" は、**notime** 操作で root ディレクトリーをマウントします。
-q or --quiet	ログメッセージの出力を防ぎます。	$ **virt-sysprep -q**
-v or --verbose	デバッグ目的で詳細なメッセージを有効にします。	$ **virt-sysprep -v**
-V or --version	virt-sysprep バージョン番号を表示し、終了します。	$ **virt-sysprep -V**

詳細は、libguestfs の web ページ を参照してください。

第28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 のほかにも、ゲスト仮想マシンのコンソールにアクセスできるようにする他のツールがあります。以下のセクションでは、これらのツールについて説明します。

28.1. virt-viewer コマンドラインの使用

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ツールです。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-viewer ツールをインストールするには、以下を実行します。

```
# sudo yum install virt-viewer
```

基本的な virt viewer コマンドは以下のようになります。

```
# virt-viewer [OPTIONS] {domain-name|id|uuid}
```

以下のオプションを virt-viewer と併用できます。

- **-h**、または **--help** - コマンドラインの help の要約を表示します。
- **-V**、または **--version** - virt-viewer のバージョン番号を表示します。
- **-v**、または **--verbose** - ゲスト仮想マシンへの接続についての情報を表示します。
- **-c *URI***、または **--connect=*URI*** - ハイパーバイザーの接続 URI を指定します。
- **-w** または **--wait** - コンソールへの接続を試行する前にドメインを起動させます。
- **-r** または **--reconnect** - ドメインがシャットダウンおよび再起動する場合にドメインに自動的に再接続します。
- **-z *PCT*** または **--zoom=*PCT*** - 指定されるパーセントでディスプレイウィンドウのズームレベルを調整します。許可される範囲は 10-200% です。
- **-a** または **--attach** - *libvirt* を使用して、TCP/UNIX ソケット接続を行う代わりにローカルディスプレイに直接割り当てます。これにより、*libvirt* で認証がすでに許可されている場合はリモートディスプレイでの認証の必要がなくなります。このオプションはリモートディスプレイでは機能しません。
- **-f**、または **--full-screen** - フルスクリーンサイズに最大化したウィンドウで起動します。
- **-h *hotkeys*** または **--hotkeys *hotkeys*** - 新たに指定されたホットキーでデフォルトのホットキー設定をオーバーライドします。例28.5「ホットキーの設定」 を参照してください。
- **--debug** - デバッグ情報を出力します。

例28.1 ゲスト仮想マシンへの接続

XEN ハイパーバイザーを使用する場合:

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

KVM-QEMU ハイパーバイザーを使用する場合:

```
# virt-viewer --connect qemu:///system 7
```

例28.2 接続前に特定ゲストの起動を待機する

以下のコマンドを実行します。

```
# virt-viewer --reconnect --wait 66ab33c0-6919-a3f7-e659-16c82d248521
```

例28.3 TLS を使用してリモートコンソールに接続する

以下のコマンドを実行します。

```
# virt-viewer --connect xen://example.org/ demo
```

例28.4 SSH を使用してリモートホストに接続する

ゲスト設定を検索してから、トンネルを使用せずにコンソールへ直接接続します。

```
# virt-viewer --direct --connect xen+ssh://root@example.org/ demo
```

例28.5 ホットキーの設定

カスタマイズされたホットキーを作成するには、以下のコマンドを実行します。

```
# virt-viewer --hotkeys=action1=key-combination1, action2=key-
combination2
```

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。それぞれのホットキー設定には固有のキーの組み合わせがあります。

たとえば、ホットキーを作成してフルスクリーンモードに変更するには、以下のようにします。

```
# virt-viewer --hotkeys=toggle-fullscreen=shift+f11 qemu:///system 7
```

28.2. remote-viewer

28.2.1. 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。

remote-viewer ツールをインストールするには、以下を実行します。

```
# sudo yum install remote-viewer
```

基本的な remote viewer コマンドは以下のようになります。

```
remote-viewer [OPTIONS] -- URI
```

以下のオプションを remote-viewer と併用できます。

- **-h**、または **--help** - コマンドラインの help の要約を表示します。
- **-V** または **--version** - remote-viewer のバージョン番号を表示します。
- **-v**、または **--verbose** - ゲスト仮想マシンへの接続についての情報を表示します。
- **-z *PCT*** または **--zoom=*PCT*** - 指定されるパーセントでディスプレイウィンドウのズームレベルを調整します。許可される範囲は 10-200% です。
- **-f** または **--full-screen=auto-conf** - フルスクリーンサイズに最大化したウィンドウで起動します。オプションの引数 'auto-conf' が指定される場合、追加モニターを随時有効または無効にすることにより、リモートディスプレイがクライアント物理モニター設定に一致するように再設定されます。現在、これは Spice バックエンドのみで実装されています。
- **-t *title*** または **--title *title*** - 指定される文字列にウィンドウのタイトルを設定します。
- **--spice-controller** - SPICE コントローラーを使用し、SPICE サーバーで接続を初期化します。このオプションは、web ページからクライアントを起動することを許可するために SPICE ブラウザープラグインによって使用されます。
- **--debug** - デバッグ情報を出力します。

詳細は、remote-viewer の man ページを参照してください。

28.3. 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。Boxes は最小の設定で、デスクトップから異なるオペレーティングシステムとアプリケーションをテストする方法を提供します。

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28.1 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ーを表示します。

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28.1 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。

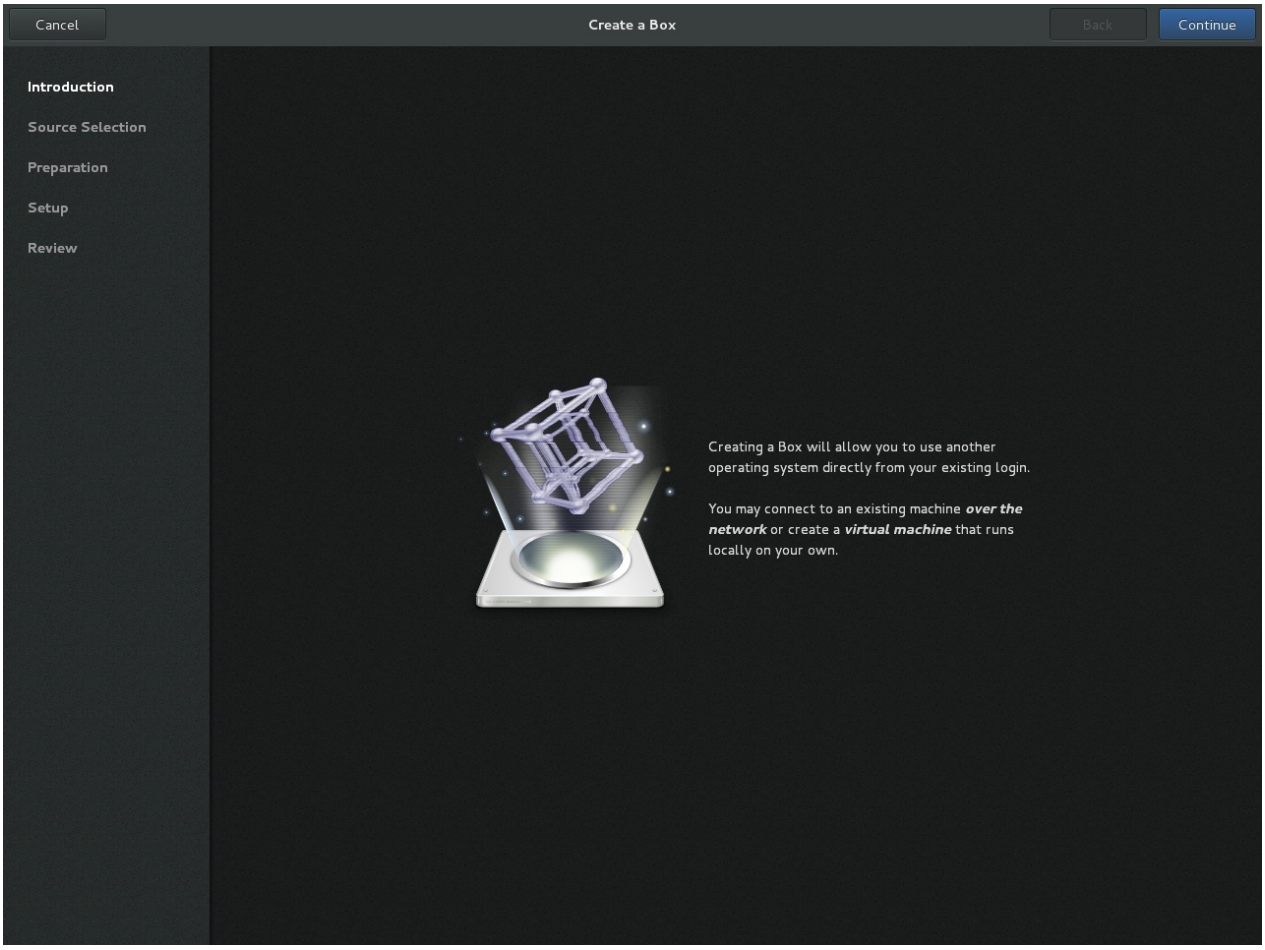


図28.2 Introduction 画面

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。

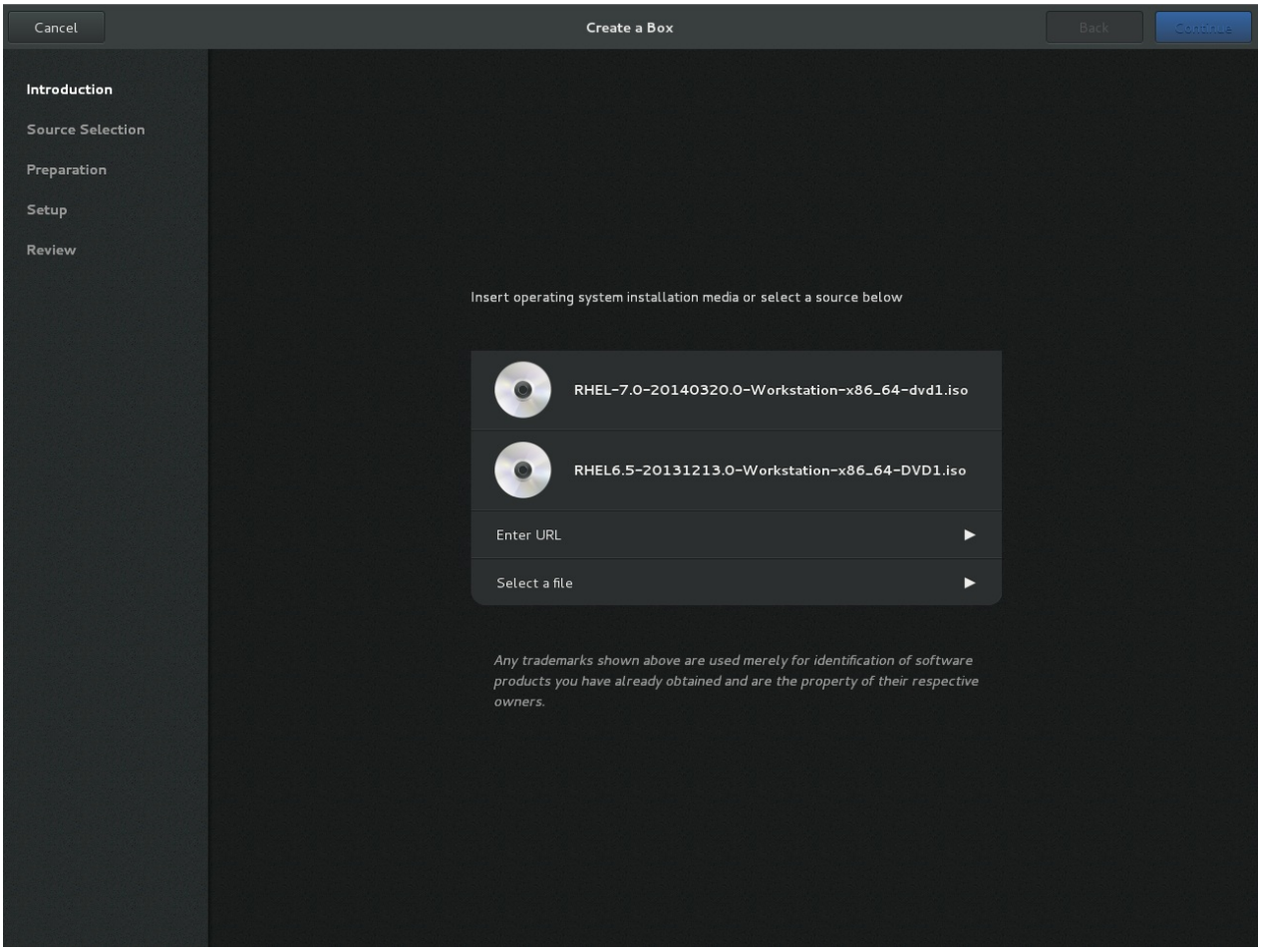


図28.3 ソース選択画面

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。

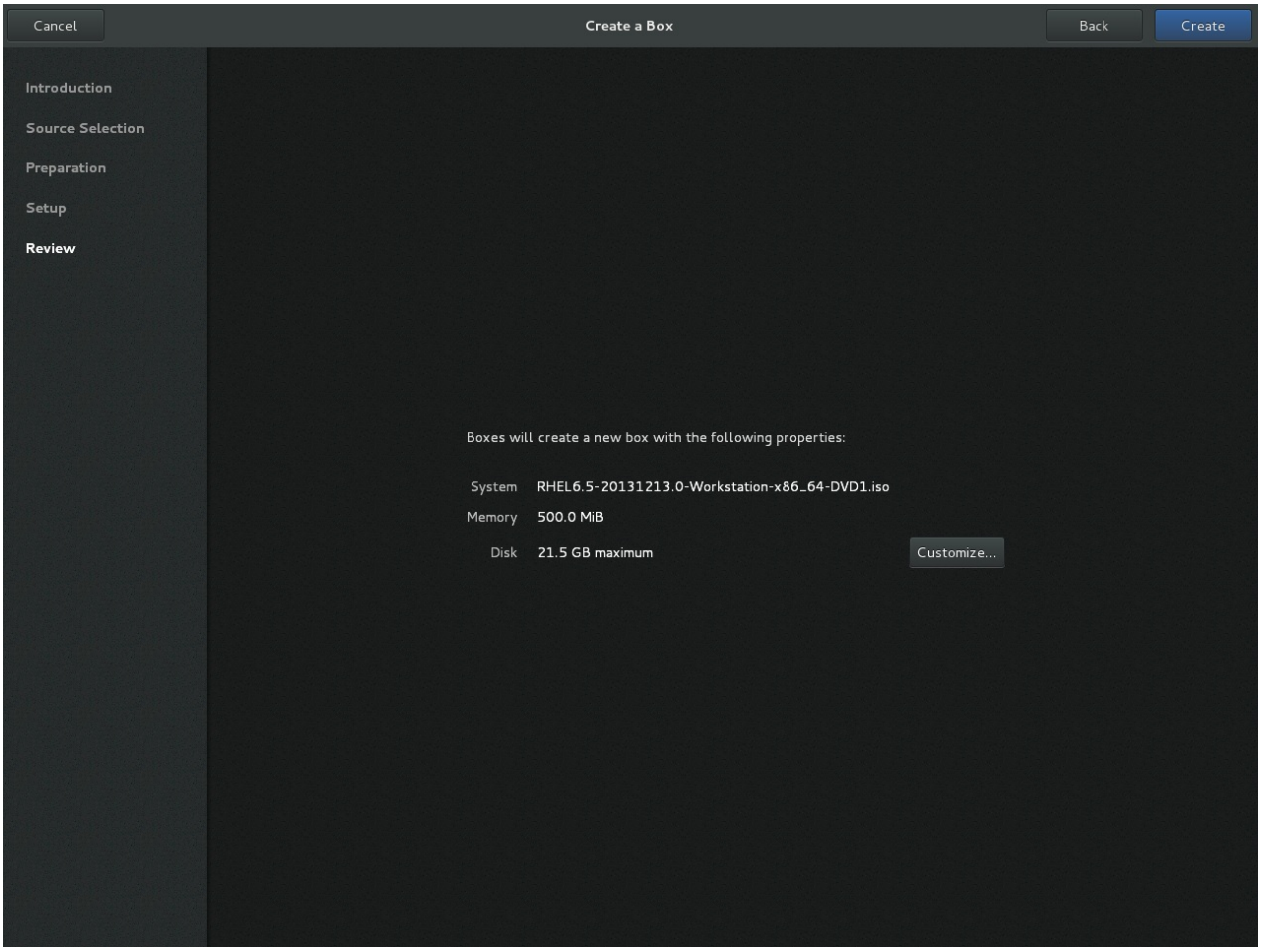


図28.4 確認画面

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。

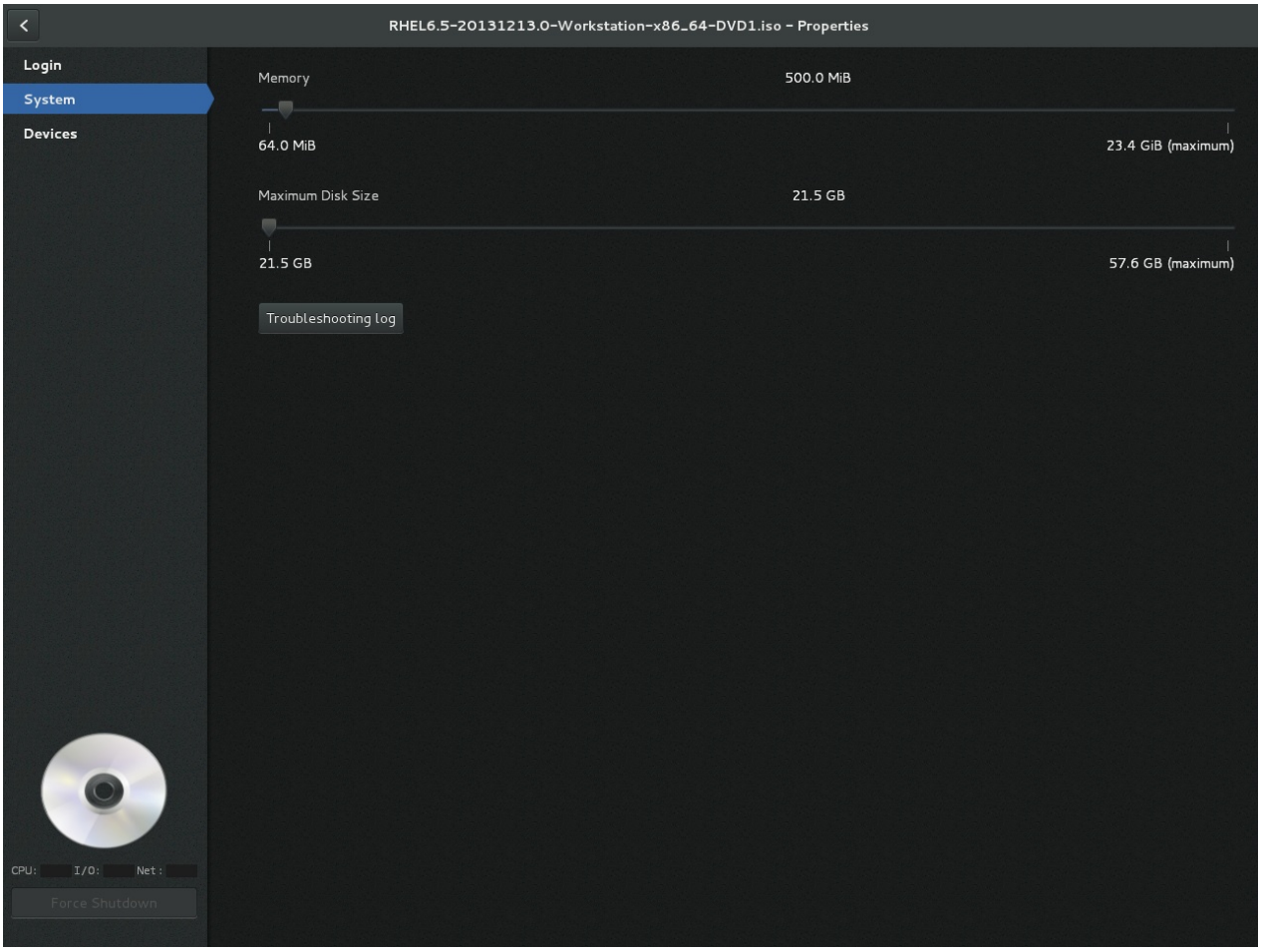


図28.5 カスタマイズ画面

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29章 ドメイン XML の操作

この章では、domain.xml 設定ファイル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**<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。この章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。本マニュアルの追加の章は、ドメイン XML の操作情報が必要な場合に本章に言及する可能性があります。

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、編集したりしないようにしてください。

29.1. 一般的な情報およびメタデータ

この情報はドメイン XML の以下の部分に入ります。

```
<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29.1 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29.1 一般的なメタデータの要素

要素	説明
<name>	仮想マシンの名前を割り当てます。この名前には英数文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。これは、多くの場合、永続的な設定ファイルを格納する目的でファイル名を作成するのに使用されます。

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 **3e3fce45-4f53-4fa7-bb32-11f34168b82b** 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、**sysinfo** 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>	title とは異なり、このデータは libvirt では全く使用されず、ここには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することができます。アプリケーションは、XML ノード/ツリーのカスタムの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)。

29.2. 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、およびコンテナーによる起動が含まれます。

29.2.1. BIOS ブートローダー

BIOS から起動する方法は、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>
  <loader>/usr/lib/kvm/boot/hvmloader</loader>
  <boot dev='hd'/>
  <boot dev='cdrom'/>
  <bootmenu enable='yes'/>
  <smbios mode='sysinfo'/>
  <bios useserial='yes' rebootTimeout='0'/>
</os>
...
```

図29.2 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29.2 BIOS ブートローダー要素

要素	説明
<type>	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。**hvm** 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。**linux** 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、**arch** は仮想化に対する CPU アーキテクチャーを指定し、**machine** はマシンタイプを指します。詳細は、『Driver Capabilities』 を参照してください。
<loader>	ドメインの作成プロセスを支援するために使用されるファームウェアの部分を参照します。これは、KVM 完全仮想化ドメインを使用する場合にのみ必要になります。
<boot>	**fd**、**hd**、**cdrom** または **network** 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。boot 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 **virDomainGetXMLDesc** から) で返される 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 bootable (起動可能) というマークが付けられます。詳細は、『BIOS bootloader』 を参照してください。
<bootmenu>	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。**enable** 属性には **yes** または **no** 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。
<smbios>	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。**mode** 属性は、**emulate** (ハイパーバイザーがすべての値を生成) か、または **host** (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。**virConnectGetSysinfo** 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 **sysinfo** (sysinfo 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、**yes** または **no** の値が使用される可能性のある属性 **useserial** 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 Serial Graphics Adapter 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 **rebootTimeout** という別の属性もあります。値は、最大値が **65535** のミリ秒単位で設定され、特別な値を **-1** に設定すると再起動が無効になります。

29.2.2. ホスト物理マシンのブートローダー

準仮想化を採用するハイパーバイザーは通常 BIOS をエミュレートせず、代わりにホスト物理マシンがオペレーティングシステムの起動を行います。これにより、ゲスト仮想マシンのカーネルを選択するためのインターフェースを提供するために、ホスト物理マシンで擬似ブートローダーが使用される場合があります。以下は、KVM の場合の **PyGrub** の例です。

```
...
 <bootloader>/usr/bin/pygrub</bootloader>
 <bootloader_args>--append single</bootloader_args>
 ...
```

図29.3 ホスト物理マシン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29.3 BIOS ブートローダー要素

要素	説明
<bootloader>	ホスト物理マシン OS のブートローダー実行可能プログラムへの完全修飾パスを提供します。このブートローダーは起動するカーネルを選択します。ブートローダーの必要な出力は、使用されるハイパーバイザーによって異なります。
<bootloader_args>	ブートローダーに渡されるコマンドライン引数を許可します (オプションのコマンド)

29.2.3. 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多くあります。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>
  <loader>/usr/lib/kvm/boot/hvmloader</loader>
  <kernel>/root/f8-i386-vmlinuz</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29.4 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29.4 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。
<loader>	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。
<cmdline>	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。

29.2.4. コンテナーによる起動

カーネルまたはブートイメージの代わりにコンテナー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** バイナリー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>hvm</type>
  <loader>/usr/lib/kvm/boot/hvmloader</loader>
  <kernel>/root/f8-i386-vmlinuz</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29.5 コンテナーによる起動

29.3. 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29.6 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。

- **<smbios>** - サブ要素は特定の SMBIOS 値を呼び出します。これは、**<os>** 要素の**smbios** 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。

29.4. CPU の割り当て

```
<domain>
  ...
  <vcpu placement='static' cpuset="1-4,^3,6" current="1">2</vcpu>
```

```
  ...
</domain>
```

図29.7 CPU の割り当て

<cpu> 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 の固定ポリシーは、**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、**vcpupin** の値を設定している仮想 CPU では、**cpuset** 設定は無視されます。**vcpupin** 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。

29.5. CPU のチューニング

```
<domain>
  ...
  <cputune>
    <vcpupin vcpu="0" cpuset="1-4,^2"/>
    <vcpupin vcpu="1" cpuset="0,1"/>
    <vcpupin vcpu="2" cpuset="2,3"/>
    <vcpupin 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29.8 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29.5 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。
<vcpupin>	ドメイン VCPU 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、vCPU 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** 属性は **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。
<shares>	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 2048 の **<shares>** 値で設定されている場合、CPU 時間は 1024 の **<shares>** 値で設定されるゲスト仮想マシンの CPU 時間の 2 倍になります。
<period>	施行間隔をマイクロ秒単位で指定します。**<period>** を使用することにより、ドメインの vcpu のそれぞれは、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 **1000-1000000** の範囲内にある必要があります。**<period>** の後に値 **0** が来る場合は、値なしを意味します。
<quota>	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 **<quota>** 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 **1000 - 18446744073709551** の範囲内にするか、または **0** 未満にする必要があります。**0** の値を持つ **quota** は値なしを意味します。この機能を使用して、すべての vcpus が同じ速度で実行されるようにすることができます。

要素	説明
<emulator_period>	施行間隔をマイクロ秒単位で指定します。**<emulator_period>** 内で、ドメインのエミュレータースレッド (vCPU を除く) は、実行時間に相当する **<emulator_quota>** を超える量を消費することが許可されません。**<emulator_period>** 値は **1000 - 1000000** の範囲内にある必要があります。**0** の値を持つ **<emulator_period>** は値なしを意味します。
<emulator_quota>	マイクロ秒単位でドメインのエミュレータースレッドに許可される最大帯域幅を指定します (vCPU を除く)。負の値の **<emulator_quota>** を持つドメインは、ドメインがエミュレータースレッド (vCPU 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 **1000 - 18446744073709551** の範囲内にするか、または **0** 未満にする必要があります。**0** の値を持つ **<emulator_quota>** は値なしを意味します。

29.6. 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/>
  </memoryBacking>
  ...
</domain>
```

図29.9 メモリーのバッキング

オプションの **<memoryBacking>** 要素には、その中に **<hugepages>** 要素を設定することができます。これは、ハイパーバイザーに対し、ゲスト仮想マシンのメモリーを、通常のデフォルトのネイティブページのサイズではなく、hugepage を使って割り当てるよう指示します。

29.7. 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29.10 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29.6 メモリーチューニング要素

要素	説明
<memtune>	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 **<memory>** 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。
<hard_limit>	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 **キビバイト** (1024 バイトのブロック) で表わされます。
<soft_limit>	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。
<swap_hard_limit>	これは、ゲスト仮想マシンが使用できる最大メモリーに swap 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、**<hard_limit>** 値よりも大きくなければなりません。
<min_guarantee>	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。

29.8. メモリの割り当て

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'***) の制御を行なうことができます。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ル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29.11 メモリー単位

29.9. 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29.12 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29.7 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 **mode** は、**interleave**、**strict**、または **preferred** 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 **strict** になります。属性 **nodeset** 属性は、要素**<vcpu>** の属性 **cpuset** と同じ構文を使用して NUMA ノードを指定します。属性 **placement** 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 **static** または **auto** 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 **static** になります。**auto** 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 nodeset の値は、それが指定されている場合は無視されます。**vcpu** の属性 **placement** が **auto** であり、属性 **<numatune>** が指定されていない場合、**<placement> auto** およびモード **strict** が指定されたデフォルトの **<numatune>** が暗黙的に追加されます。

29.10. 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29.13 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29.8 ブロック I/O チューニング要素

要素	説明
<blkiotune>	このオプションの要素は、ドメインの**blkio** cgroup 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、デフォルトで OS が指定するデフォルトになります。
<weight>	このオプションの weight 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。
<device>	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 **<device>** 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でき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 **<disk>** に適用される **<iotune>** 要素と対比できます)。それぞれの device 要素には、デバイスの絶対パスを記述する **<path>** 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 **<weight>** の 2 つの必須のサブ要素があります。

29.11. 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29.14 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。

29.12. 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29.15 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29.16 CPU モデルおよびトポロジーサンプル 2

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

図29.17 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29.18 CPU モデルおよびトポロジーサンプル 4

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29.9 CPU モデルおよびトポロジー要素

要素	説明
<cpu>	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ーです。

要素	説明
<match>	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 **<cpu>** 内の唯一の要素である場合、**match** 属性を省略することができます。**match** 属性に使用できる値は、以下になります。 - **minimum** - 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。 - **exact** - 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。 - **strict** - 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。 **match** 属性は省略でき、デフォルトで**exact** になることに注意してください。

要素	説明
<mode>	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。**mode** 属性に使用できる属性は以下になります。 - **custom** - CPU 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、**mode** 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。 - **host-model** - これは基本的に、capabilities XML 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。**match** 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、libvirt domain XML CPU models を参照してください。 - **host-passthrough** 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。**model** または **feature** 要素もいずれもこのモードでは許可されません。 - **host-model** と **host-passthrough** 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 (**host-passthrough** モードの値と近似する) CPU 定義は、**virDomainGetXMLDesc** API を呼び出す際に **VIR_DOMAIN_XML_UPDATE_CPU** 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 custom モードに書き換えることができます。

要素	説明
<model>	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 **cpu_map.xml** 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 **fallback** 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。fallback 属性のサポートされている値は、**allow** (デフォルト) と **forbid** です。オプションの **vendor_id** 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、**AuthenticAMD** と **GenuineIntel** です。
<vendor>	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 **cpu_map.xml** にあります。
<topology>	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>	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、**cpu_map.xml** ファイルにあります。それぞれの feature 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。 - **force** - 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。 - **require** - 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。 - **optional** - 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。 - **disable** - これは仮想 CPU によってサポートされません。 - **forbid** - 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。

29.12.1. 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29.1 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29.19 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>

<!--original 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29.20 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再起動します。

29.12.2. 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29.21 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。

29.13. 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オーバーライドできます。

```
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<on_lockfailure>poweroff</on_lockfailure>
```

図29.22 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29.10 イベント設定の要素

状態	説明
<on_poweroff>	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。 - **destroy** - 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。 - **restart** - 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。 - **preserve** - 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。 - **rename-restart** - 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。

状態	説明
<on_reboot>	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。 - **destroy** - 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。 - **restart** - 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。 - **preserve** - 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。 - **rename-restart** - 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。
<on_crash>	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。 - **coredump-destroy** - 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。 - **coredump-restart** - 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。 以下の 4 つの引数を使用できます。 - **destroy** - 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。 - **restart** - 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。 - **preserve** - 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。 - **rename-restart** - 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。

状態	説明
<on_lockfailure>	ロックマネージャーがリソースロックを失う場合に取る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、libvirt 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。 - **poweroff** - 強制的にドメインの電源をオフにします。 - **restart** - ロックを再取得するようにドメインを再起動します。 - **pause** - 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。 - **ignore** - 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。

29.14. 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29.23 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。

29.15. 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29.24 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29.11 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 **on** と **off** を含むオプションの属性 **eoi** 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。
<hyperv>	Microsoft Windows を実行するゲスト仮想マシンの動作を改善する各種の機能を有効にします。オプションの属性 **relaxed** に値 **on** または **off** を指定して、タイマーの制約の緩和を有効/無効にします。

29.16. 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。Windows ゲスト仮想マシンの場合は、ゲスト仮想マシンを **ローカル時間** に設定する必要があります。

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設定ファイルの編集」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29.25 Timekeeping

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29.12 Timekeeping 要素

状態	説明

状態	説明
<clock>	**<clock>** 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。**offset** 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。 • **utc** - クロックを起動時に UTC に同期します。**utc** モードは **variable** モードに変換でき、これは adjustment 属性を使用して制御できます。値が **reset** の場合、変換は実行されません。数値は、初期 adjustment としてその値を使用することで、**variable** モードへの変換を強制します。デフォルトの adjustment はハイパーバイザーに固有のものになります。 • **localtime** - ゲスト仮想マシンクロックを、起動時ににホスト物理マシンの設定タイムゾーンに同期します。adjustment 属性は **utc** モードと同様に機能します。 • **timezone** - 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。 • **variable** - basis 属性に基づいて、ゲスト仮想マシンクロックに、UTC または **localtime** 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 **localtime**) に対する差分は、**adjustment** 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 (リアルクロックタイム) を自由に調整することができ、行な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 **utc** と **localtime** モード (オプション属性 **adjustment='reset'** を指定) とは対照的です。さらに、**basis** 属性には、**utc** (デフォルト) または **localtime** のいずれかを使用することができます。**clock** 要素には、ゼロまたは 1 以上の **<timer>** 要素を含めることができます。
<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。**yes** または **no** 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29.1 常に UTC に同期する

```
<clock offset="utc" />
```

例29.2 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29.3 任意のタイムゾーンに同期する

```
<clock offset="timezone" timezone="Europe/Paris" />
```

例29.4 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" />
```

29.17. 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29.13 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、**tickpolicy** delay 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 **tickpolicy catchup** 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 pvclock または kvm-clock 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。
hypervclock	Windows ゲスト仮想マシンのみの Hyper-V クロックです。これは、すべての Windows ゲストの推奨されるクロックソースです。

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29.14 track 属性の値

値	説明
boot	旧オプションの **host physical machine** 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29.15 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29.16 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に強制します。

29.18. 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29.26 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。

29.18.1. 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29.27 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29.28 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29.29 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 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29.30 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 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29.31 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 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29.32 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 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-
nopool/2'>
```

```
        <host name='example.com' port='3260'/>
      </source>
      <auth username='myuser'>
        <secret type='chap' usage='libvirtiscsi'/>
      </auth>
      <target dev='vda' bus='virtio'/>
    </disk>
```

図29.33 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 7

```
    <disk type='network' device='lun'>
      <driver name='qemu' type='raw'/>
      <source protocol='iscsi' name='iqn.2013-07.com.example:iscsi-
nopool/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29.34 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 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29.35 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 9

29.18.1.1. ディスク要素

<disk> 要素は、ディスクを記述するためのメインコンテナーです。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、Disk Elements を参照してください。

29.18.1.2. ソース要素

<disk type='file'> の場合、**file** 属性は、ディスクを保持するファイルへの完全修飾パスを指定します。**<disk type='block'>** の場合、**dev** 属性は、ディスクとして機能するホスト物理マシンへのパスを指定します。**file** と **block** の両方の場合に、下記の 1 つ以上のオプションサブ要素**seclabel** を使用して、そのソースファイルのドメインセキュリティーラベルポリシーをオーバーライドすることができます。ディスクタイプが **dir** の場合、**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。ディスクタイプが **network** の場合、プロトコル属性は、要求されるイメージにアクセスするためのプロトコルを指定します。使用できる値は、**nbd**、**rbd**、**sheepdog** または **gluster** です。

プロトコル属性が **rbd**、**sheepdog** または **gluster** の場合、追加属性の **name** は、使用するボリュームおよび/またはイメージを指定するのに必須となります。ディスクタイプが **network** の場合、**source** には、**type='dir'** および **type='network'** などの、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または 1 つ以上の **host** サブ要素が含まれる場合があります。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** 属性を処理することによって実行されます。

- **mandatory** は、何らかの理由で見当たらない場合に、失敗を生じさせます。これはデフォルトの設定です。
- **requisite** は、ブート時に見つからない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。

29.18.1.3. Mirror 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、ディスクはピボットできる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。

29.18.1.4. Target 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。詳細は、Target Elements を参照してください。

29.18.1.5. 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 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ができません。
- **<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 回数です。

29.18.1.6. 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、プライマリー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。使用できるタイプの一覧は、Driver Elements を参照してください。
- 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** をオーバーライドします。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。

- 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。

29.18.1.7. 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。詳細は、Device Elements を参照してください。
- **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。
- **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ーのいずれかでオーバーライド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。

29.18.2. 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にあります。

```
...
 <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29.36 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。ドライバーブロックには、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アクセス権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アクセス権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。コンテナーは、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。その後にハード制限が施行されます。

29.18.3. 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 0xfff 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 0xfff 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。

29.18.4. 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29.37 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 or "vmpvscsi"**。virtio-scsi コントローラーおよびドライバーは KVM および Windows ゲスト仮想マシンの両方で機能することに注意してください。また、**<controller type='scsi'>** には、指定するキューの数に対する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。

"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29.38 デバイス - コントローラー - USB

29.18.5. 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。通常の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29.39 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。

29.18.6. ホスト物理マシンのデバイス割り当て

29.18.6.1. 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29.40 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29.41 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29.42 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29.17 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

パラメーター	説明
hostdev	これは、ホスト物理マシンを説明するためのメインコンテナーです。USB デバイスパススルーの場合、**mode** は常に **subsystem** であり、**type** は、USB デバイスの場合は **usb**、PCI デバイスの場合は **pci** になります。**managed** が PCI デバイスについて **yes** とされる場合、これは、ゲスト仮想マシンに渡される前に、ホスト物理マシンから切り離され、ゲスト仮想マシンが終了した後にホスト物理マシンに再度割り当てられます。**managed** が省略されるか、または PCI および USB デバイスに対して **no** となる場合、ユーザーは、ゲスト仮想マシンの開始またはデバイスのホットプラグの前に引数 **virNodeDeviceDettach** (または **virsh nodedev-dettach**) を使用することができ、ゲスト仮想マシンのホットアンプラグまたは停止の後に **virNodeDeviceReAttach** (または **virsh nodedev-reattach**) を使用することができます。
source	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、**vendor** および **product** 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 **address** 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 **startupPolicy** 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。 - **mandatory** - 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) - **requisite** - 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。 - **optional** - 開始の試行時に見つからない場合にドロップします。
vendor, product	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 **id** 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。
boot	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 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。

パラメーター	説明
rom	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 **bar** 属性は、**on** または **off** 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、**rombar** 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 **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。
address	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 **bus** および **device** 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 **lspci** または **virsh nodedev-list** で検索できるように指定することができます。

29.18.6.2. 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、コンテナー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29.43 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29.44 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29.45 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 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29.18 ブロック / キャラクターデバイス要素

パラメーター	説明
hostdev	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 **mode** は常に **capabilities** で、ブロックデバイスの場合 **type** は **block** で、キャラクターデバイスの場合 **char** になります。
source	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 **block** 要素に指定され、キャラクターデバイスの場合は、**char** 要素が使用されます。

29.18.7. 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29.46 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29.19 リダイレクトされるデバイス要素

パラメーター	説明
redirdev	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。**bus** は、USB デバイスの場合は **usb** 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、**type='tcp'** または **type='spicevmc'** (SPICE グラフィックスデバイスの usbredir チャネルを使用) です。**redirdev** 要素にはオプションのサブ要素 **address** 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、**source** などのサブ要素が、指定される **type** に基づいて必要になる場合があります。ただし、**target** 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)。
boot	デバイスが起動可能であることを指定します。**order** 属性は、ブートシーケンス時にデバイスを試行す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ません。
redirfilter	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 **usbdev** を使用します。**class** 属性は USB クラスコードです。

29.18.8. 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できないことから、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29.47 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。

モード属性は以下のようになります。

表29.20 スマートカードの mode 要素

パラメーター	説明
mode='host'	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 **address** サブ要素の使用については、以下を参照してください。

パラメーター	説明
mode='host-certificates'	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 **certutil -d /etc/pki/nssdb -x -t CT,CT,CT -S -s CN=cert1 -n cert1,** 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 **certificate** 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 **database** 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 **certutil** コマンドの **-d** フラグに一致)。これが表示されない場合、デフォルトは **/etc/pki/nssdb** になります。
mode='passthrough'	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 **type** 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。**type='tcp'** または **type='spicevmc'** (SPICE グラフィックスデバイスのスマートカードチャネルを使用) などが一般的な例になります。さらに、**source** 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、**target** 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)。

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 (「デバイスアドレス」を参照してください)。

29.18.9. 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='bridge'>
    <source bridge='kvmbr0'/>
    <mac address='00:16:3e:5d:c7:9e'/>
    <script path='vif-bridge'/>
    <boot order='1'/>
```

```
      <rom bar='off'/>
    </interface>
  </devices>
  ...
```

図29.48 デバイス - ネットワークインターフェース

ゲスト仮想マシンに表示されるネットワークインターフェースを指定するには、いくつかの方法が考えられます。以下のサブセクションでは、それぞれ共通のセットアップオプションについて詳述しています。さらに、それぞれの **<interface>** 要素には、オプションの **<address>** サブ要素があり、このサブ要素は、属性 **type='pci'** により、インターフェースを特定の pci スロットに関連付けることができます (「デバイスアドレス」 を参照)。

29.18.9.1. 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>** 要素でオーバーライドすることができます (「ターゲット要素のオーバーライド」 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、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、vepa (802.1Qbg) または 802.1Qbh 互換スイッチまたは 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、仮想ポートのタイプ属性を省略したり、複数の異なる仮想ポートから属性を指定 (さらに特定の属性を省略) したりすることができます。ドメインの起動時に、完全な **<virtualport>** 要素が、ネットワークで定義されるタイプと属性、およびインターフェースで参照されるポートグループをマージすることによって構成されます。新たに構成される仮想ポートは、これら 2 つを組み合わせたものです。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ません。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なネットワークを動作させるには、no **type** を指定する選択ができますが、接続を機能させるには、**profileid** (スイッチが 802.1Qbh の場合) と **interfaceid** (スイッチが Open vSwitch の場合) の両方を指定する必要があります。さらに、ネットワークの **virtualport** から入力される **managerid**、**typeid**、または **profileid** などの他の属性は省略することもできます。ゲスト仮想マシンを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ますが、一部のまたはすべてのパラメーターを省略することができます。この場合、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します。

```
...
<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29.49 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

29.18.9.2. LAN のブリッジ

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>** 要素でオーバーライドできます (「ターゲット要素のオーバーライド」 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29.50 デバイス - ネットワークインターフェース- LAN のデバイス

29.18.9.3. 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29.51 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。

29.18.9.4. 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、ゲスト仮想マシンに発信アクセスを持たせる必要のある権限を持たないユーザーにとっての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29.52 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

29.18.9.5. 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** 要素でオーバーライド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期されます。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、これをオーバーライドすることができます (「ターゲット要素のオーバーライド」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29.53 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

29.18.9.6. 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。

このセットアップでは、Linux macvtap ドライバーを利用可能にする必要があります。モード **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29.54 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29.21「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なわれます。

表29.21 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)。
bridge	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 macvtap 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 bridge モードにする必要があります。これらのいずれかが **vepa** モードの場合は、VEPA 対応のブリッジが必要になります。
private	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 private モードの場合に、この手順が実行されます。
passthrough	この機能は、SR-IOV 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 VF/IF 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 2.6.38 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29.22「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29.22 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
managerid	VSI Manager ID は、VSI 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 **0** 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。
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。 グローバルに固有となる識別子です。
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29.55 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29.56 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

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。

29.18.9.7. 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) を行なった後に、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) デバイスのみを、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29.57 デバイス - ネットワークインターフェース- PCI パススルー

29.18.9.8. 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に、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29.58 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

29.18.9.9. TCP トンネル

TCP クライアント/サーバーアーキテクチャーを作成することは、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29.59 デバイス - ネットワークインターフェース- TCP トンネル

29.18.9.10. 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29.60 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29.23 virtio NIC ドライバー要素

パラメーター	説明
name	オプションの **name** 属性は、使用するバックエンドドライバーのタイプを強制します。値は、**kvm** (ユーザースペースバックエンド) または **vhost** (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 vhost ドライバーを要求する試みは拒否されます。vhost ドライバーが存在する場合、デフォルト設定は **vhost** ですが、存在しない場合はサイレントに **kvm** にフォールバックします。

パラメーター	説明
txmode	送信バッファーが一杯の場合に、パケット送信の処理方法を指定します。値は、**iothread** または **timer** のいずれかに設定できます。**iothread** に設定される場合、packet tx がドライバーの下半分の iothread ですべて実行されます (このオプションは、**"tx=bh"** を **kvm** コマンドラインの -device virtio-net-pci オプションに変換します)。**timer** に設定される場合、tx の作業は KVM で実行され、現行時間に送信できる以上の tx データがある場合、タイマーが KVM 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なわれます。通常、この値を変更しないことが推奨されています。
ioeventfd	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、**on** と **off** 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、KVM 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。
event_idx	**event_idx** 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 **on** または **off** のいずれかにすることができます。**on** 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。

29.18.9.11. ターゲット要素のオーバーライド

ターゲット要素をオーバーライドするには、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29.61 デバイス - ネットワークインターフェース- ターゲット要素のオーバーライド

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。

29.18.9.12. 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 に以下の変更を行ないます。

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <boot order='1'/>
  </interface>
</devices>
...
```

図29.62 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。

29.18.9.13. 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29.63 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ーファイルを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。

29.18.9.14. 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 による決定が行なわれる 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 が設定され (少なくとも **average**) る必要があるものの、floor 属性には**average** を指定する必要がない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29.64 QoS (Quality of Service)

29.18.9.15. 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29.65 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 トランキングを実行することを希望していると仮定されます。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。

29.18.9.16. 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29.66 仮想リンク状態の変更

29.18.10. 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29.67 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。

29.18.11. 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29.68 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。

29.18.12. グラフィカルフレームバッファー

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、通常は管理者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ーか、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ー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29.69 グラフィカルフレームバッファー

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** または **desktop** を取ります。

表29.24 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
sdl	これは、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。以下の 3 つのオプションの引数を取ることができます。使用するディスプレイには **display** 属性、認証 ID には **xauth** 属性、および値の **yes** または **no** を受け入れるオプションの **fullscreen** 属性です。

パラメーター	説明
vnc	VNC サーバーを起動します。**port** 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 **-1** を指定)。**autoport** 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。**listen** 属性は、サーバーがリッスンする IP アドレスです。**passwd** 属性は、クリアテキストで VNC パスワードを提供します。**keymap** 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、**timestamp passwdValidTo='2010-04-09T15:51:00'** を UTC に指定するなどして、設定することができます。**connected** 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 **keep** 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。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ソケット属性をサポートします。
spice	SPICE サーバーを起動します。**port** 属性は TCP ポート番号を指定 (自動割り当てであることを示すレガシー構文として -1 を指定) し、**tlsPort** は代替のセキュアポート番号を指定します。**autoport** 属性は、両方のポート番号の自動割り当てを示すために推奨される新たな構文です。**listen** 属性は、サーバーがリッスンする IP アドレスです。**passwd** 属性は、クリアテキストに SPICE パスワードを提供します。**keymap** 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を、**timestamp passwdValidTo='2010-04-09T15:51:00'** を UTC に指定するなどして設定できます。**connected** 属性は、パスワードの変更時に接続されるクライアントの制御を可能にします。SPICE は、クライアントを接続状態にするために **keep** を受け入れ、クライアントの接続を解除するために **disconnect** を、パスワードの変更における失敗には **fail** を受け入れます。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。**defaultMode** 属性は、デフォルトのチャネルセキュリティーポリシーを設定し、有効な値は **secure**、**insecure** でデフォルトは **any** です。(可能な場合は **secure** 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 **insecure** にフォールバックします)。

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29.70 SPICE 設定

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、以下のすべての要素の **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** プロパティーを **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29.25 追加の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
rdp	RDP サーバーを起動します。**port** 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 -1 を指定する)。**autoport** 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。**replaceUser** 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。**multiUser** 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。
desktop	この値は、しばらくの間 VirtualBox ドメイン用に保持されます。これは、"sdl" 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、VirtualBox ビューアーを使用します。"sdl" のように、これはオプションの属性の **display** および **fullscreen** を受け入れます。

パラメーター	説明
listen	**graphics** のグラフィックスタイプ vnc および spice 用にリッ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、**listen** 属性という **graphics** のサブ要素を指定できます (上記の例を参照)。**listen** は以下の属性を受け入れます。 - **type** - **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 listen 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 (それがリッ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。 - **address** - この属性には、リッスンする 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、**type='network'** の場合でも、リッスンするために使用される IP アドレスに設定されます。 - **network** - **type='network'** の場合、network 属性には、設定済みネットワークの libvirt 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、適切なリッ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 IPv4 アドレスを持つ場合 (たとえば、ルートの forward タイプまたは、NAT、または no forward タイプ (isolated) の場合)、ネットワークの設定にリストされる最初の IPv4 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 IPv4 アドレスが使用され、ネットワークが 'direct' (macvtap) モードのいずれかを記述する場合には、最初の forward dev の最初の IPv4 アドレスが使用されます。

29.18.13. 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29.71 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29.26 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
video	**video** 要素は、ビデオデバイスを記述するためのコンテナーです。下位互換性のため、video が設定されていないものの、ドメイン XML に **graphics** 要素がある場合、libvirt は、ゲスト仮想マシンタイプに基づいてデフォルトの **video** を追加します。"ram" または "vram" が指定されていない場合、デフォルト値が使用されます。
model	これには、必須の **type** 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 **vga**、**cirrus**, **vmvga**、**kvm**、**vbox**、または **qxl** 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 vram 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。
acceleration	aceleration がサポートされる場合、**acceleration** 要素で **accel3d** と **accel2d** 属性を使用してこれを有効にする必要があります。
address	オプションの address 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。

29.18.14. 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29.72 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オーバーライド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。

29.18.15. 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29.73 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29.74 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。

29.18.16. チャネル

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29.75 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、メインチャンネル間のメッセージをピギーバッキングできます。**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。

29.18.17. 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29.27 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。	`<devices> <console type='stdio'> <target port='1'/> </console> </devices>`
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。	`<devices> <serial type="file"> <source path="/var/log/vm/vm-serial.log"/> <target port="1"/> </serial> </devices>`
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。	`<devices> <serial type='vc'> <target port="1"/> </serial> </devices>`
Null device	キャラクターデバイスを void 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。	`<devices> <serial type='null'> <target port="1"/> </serial> </devices>`

パラメーター	説明	XML スニペット
Pseudo TTY	Pseudo TTY は、**/dev/ptmx** を使用して割り当てられます。**virsh console** などの適切なプロキシーは、シリアルポートとローカルに対話することができます。	**<devices> <serial type="pty"> <source path="/dev/pts/3"/> <target port="1"/> </serial> </devices>**
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、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。	**<devices> <serial type="dev"> <source path="/dev/ttyS0"/> <target port="1"/> </serial> </devices>**
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、pipe(7) man ページを参照してください。	**<devices> <serial type="pipe"> <source path="/tmp/mypipe"/> <target port="1"/> </serial> </devices>**
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。	**<devices> <serial type="tcp"> <source mode="connect" host="0.0.0.0" service="2445"/> <protocol type="raw"/>**

パラメーター	説明	XML スニペット
		<target port="1"/> </serial> </devices>
		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。
		<devices> <serial type="tcp"> <source mode="bind" host="127.0.0.1" service="2445"/> <protocol type="raw"/> <target port="1"/> </serial> </devices>
		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。
		<devices> <serial type="tcp"> <source mode="connect" host="0.0.0.0" service="2445"/> <protocol type="telnet"/> <target port="1"/> </serial> <serial type="tcp"> <source mode="bind" host="127.0.0.1" service="2445"/> <protocol type="telnet"/> <target port="1"/> </serial> </devices>

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。これは損失の多いサービスです。	`<devices> <serial type="udp"> <source mode="bind" host="0.0.0.0" service="2445"/> <source mode="connect" host="0.0.0.0" service="2445"/> <target port="1"/> </serial> </devices>`
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。	`<devices> <serial type="unix"> <source mode="bind" path="/tmp/foo"/> <target port="1"/> </serial> </devices>`

29.18.18. 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='sb16'/>
</devices>
...
```

図29.76 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29.77 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。

29.18.19. 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29.78 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 シグナルに反応でき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。

29.18.20. パニックデバイスの設定

Red Hat Enterpise Linux には、Linux ゲスト仮想マシンのカーネルパニックを検出する手段が備わっています (現時点で Windows ゲストのサポートはありません)。これを実行するには、pvpanic デバイスを有効にしてから、**<device>** 親要素の子として **<panic>** 要素を追加する必要があります。*libvirt* はデバイスをゲスト仮想マシンに公開し、デバイスを駆動する方法を認識できる新たな Linux カーネルを実行中の場合は、ゲストカーネルのパニックをいつでも libvirt に通知するためにそのデバイスを使用します。このパニックはイベントとして *libvirt* に送られ、ドメイン XML の**<on_crash>** 要素は、クラッシュの結果として *libvirt* が実行する内容を決定します。これには、ゲスト仮想マシンのコアダンプのキャプチャー、ゲスト仮想マシンの再起動、または単に別のアクションを待機するためにゲスト仮想マシンを停止することが含まれる場合があります。

パニックメカニズムを設定するには、**virsh edit** を実行して XML を開いてから編集し、以下のスニペットをドメイン XML の **devices** 要素に置きます。

```
<devices>
   <panic>
      <address type='isa' iobase='0x505'/>
   </panic>
</devices>
```

図29.79 パニック要素のコンテンツ

要素 **<address>** 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。

29.18.21. 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、membaloon デバイスを KVM で自動的に追加しています。

```
...
<devices>
  <memballoon model='virtio'/>
</devices>
...
```

図29.80 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29.81 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。

29.18.22. TPM デバイス

TPM デバイスは、ゲスト仮想マシンが TPM 機能にアクセスできるようにします。TPM パススルーデバイスタイプは、1 つのゲスト仮想マシンのホスト物理マシンの TPM へのアクセスを提供します。それ以外のソフトウェアは、ゲスト仮想マシンが起動される時点で TPM デバイスを使用する可能性がありません (通常は **/dev/tpm0**)。以下のドメイン XML 例は、TPM パススルーデバイスの使用について示しています。

```
...
<devices>
  <tpm model='tpm-tis'>
    <backend type='passthrough'>
      <backend path='/dev/tpm0'/>
    </backend>
  </tpm>
</devices>
...
```

図29.82 TPM デバイス

model 属性は、KVM がゲスト仮想マシンに提供するデバイスモデルを指定します。モデル名が指定されていない場合、**tpm-tis** が自動的に選択されます。**<backend>** 要素は、TPM デバイスのタイプを指定しま

す。以下のタイプがサポートされます。**'passthrough'** — ホスト物理マシンの TPM デバイスおよび **'passthrough'** を使用します。このバックエンドタイプには、ホスト物理マシン上の TPM デバイスへの排他的アクセスが必要になります。このようなデバイスの例は、**/dev/tpm0** です。ファイル名は、source 要素のパス属性として指定されます。ファイル名が指定されない場合、**/dev/tpm0** が自動的に使用されます。

29.19. 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**。

29.19.1. 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29.83 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29.28「*virt-sysprep* commands」 で説明されています。

表29.28 *virt-sysprep* commands

要素	説明
<name>	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。
<uuid>	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。

要素	説明
<allocation>	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。
<capacity>	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。
<available>	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、空き容量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。

29.19.2. ソース要素

<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29.84 ソース要素オプション 1

```
...
<source>
  <adapter type='fc_host' parent='scsi_host5'
wwnn='20000000c9831b4b' wwpn='10000000c9831b4b'/>
</source>
...
```

図29.85 ソース要素オプション 2

<source> で許可される子要素は 表29.29「*Source child elements* コマンド」 で説明されています。

表29.29 *Source child elements* 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 (「ストレージプール」 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 **path** 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。
<adapter>	SCSI アダプターをベースとするストレージのソース (**<pool type='scsi'>**) を提供します。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。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 **type** はアダプタータイプを指定します。有効な値は **'fc_host'| 'scsi_host'**。これが省略され、**name** 属性が指定される場合、これはデフォルトで **type='scsi_host'** に設定されます。後方互換性を維持するには、属性 **type** は **type='scsi_host'** アダプターにはオプションになりますが、**type='fc_host'** アダプターには必須になります。属性 **wwnn** (Word Wide Node Name) および **wwpn** (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 **type='fc_host'** 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)。**wwnn** および **wwpn** の両方を指定する必要があります ((v)HBA の wwnn/wwpn を取得する方法については、「デバイスのダンプ」 を参照してください)。オプションの属性 **parent** は、**type='fc_host'** アダプターの親デバイスを指定します。
<host>	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 (**type='netfs'|'iscsi'|'rbd'|'sheepdog'|'gluster'**)。この要素は、**<directory>** または **<device>** 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 **name** 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 **port** 属性が含まれる場合があります。

要素	説明
<auth>	**<auth>** 要素は、**type** 属性 (pool **type='iscsi'|'rbd'**) 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。**type** は **type='chap'** または **type='ceph'** 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。さらに必須の属性ユーザー名は、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 libvirt 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 secret サブ要素を指定します。secret 要素には秘密オブジェクトの UUID を指定した uuid 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 usage 属性のいずれかが必要になります。詳細は、ceph を参照してください。
<name>	名前付き要素 **<type>** 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: (**type='logical'|'rbd'|'sheepdog','gluster'**)。
<format>	以下の値を取ることのできるストレージプール **<type>** の形式についての情報を提供します: **type='logical'|'disk'|'fs'|'netfs'**)。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。
<vendor>	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 **<name>** が含まれます。
<product>	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 **<name>** が含まれます。

重要

type='ceph' がリストされるすべての場合において、ceph はRed Hat Enterprise Linux 7 ではサポートされなくても、**type='chap'** でストレージプールを作成することができる点に注意してください。**type='ceph'** は Red Hat Enterprise 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。

29.19.3. ターゲット要素の作成

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29.86 ターゲット要素 XML の例

表 (表29.30「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29.30 ターゲット子要素

要素	説明
<path>	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。デバイスベ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、**/dev/** 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、**/dev/disk/by-{path,id,uuid,label}** のいずれかのように安定した場所を使用する方が好ましいと言えます。

要素	説明
<permissions>	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アクセス権についての情報を提供します。**<mode>** 要素には、8 進数のアクセス権セットについての情報が含まれます。**<owner>** 要素には、数値のユーザー ID が含まれます。**<group>** 要素には、数値のグループ ID が含まれます。**<label>** 要素には、MAC (例: SELinux) ラベル文字列が含まれます。
<timestamps>	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、**timestamps='atime'|'btime|'ctime'|'mtime'** は、既知の場合はストレージボリュームの access、birth、change、および modification time を保持します。使用される時間形式は **<seconds>** です。**<nanoseconds>**: 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、nanoseconds 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。
<encryption>	(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、ストレージ暗号化のページ を参照してください。

29.19.4. 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。

29.20. 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。

29.20.1. 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29.87 ストレージボリュームの一般的なメタデータ

表 (表29.31「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29.31 ボリューム子要素

要素	説明
<name>	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。
<key>	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。
<allocation>	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 **dmeventd** を設定します。**unit** についての注 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、**unit** についての注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

29.20.2. ターゲット要素の設定

<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29.88 ターゲット子要素

<target> の特定の子要素は 表29.32「ターゲット子要素」 で説明されています。

表29.32 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ーベースのストレージプールの場合は、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、**type** 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、16章ストレージプール 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。
<permissions>	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。**<mode>** 要素には、8 進数のアクセス権セットが含まれます。**<owner>** 要素には数字のユーザー ID が含まれます。**<group>** 要素には、数字のグループ ID が含まれます。**<label>** 要素には、MAC (例: SELinux) ラベル文字列が含まれます。

要素	説明
<compat>	互換性レベルを指定します。現在のところ、これは **<type='qcow2'>** 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 **<compat>**0.10**</compat>**、および qcow2 (バージョン 3) には **<compat>**1.1**</compat>** です。**<feature>** 要素がある場合、**<compat>**1.1**</compat>** 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。
<features>	形式固有の機能です。現在は、**<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、**<lazy_refcounts/>** 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。

29.20.3. バッキングストア要素の設定

単一の **<backingStore>** 要素は、トップレベルの **<volume>** 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) のバッキ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29.89 バッキングストアの子要素

表29.33 バッキングストアの子要素

要素	説明

要素	説明
<path>	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。
<format>	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は、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、**<type>** 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、qcow2 形式は異なるバッキングストア形式を許可します。
<permissions>	バッキングファイルのアクセス権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。**<owner>** 要素には、数字のユーザー ID が含まれます。**<group>** 要素には、数字のグループ ID が含まれます。**<label>** 要素には、MAC (例: SELinux) ラベル文字列が含まれます。

29.21. 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29.90 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ー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され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ー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。

29.22. 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29.91 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には、書籍から学ぶには難しい実践や経験が必要になります。トラブルシューティングの技能を磨くには 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ーのサイトから解決案が見つかるかも知れません。「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用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_stat」を参照してください。
- **trace-cmd**
- **ftrace** 『Red Hat Enterprise Linux 開発者ガイド』 を参照してください。
- **vmstat**
- **iostat**
- **lsof**
- **systemtap**
- **crash**
- **sysrq**
- **sysrq t**
- **sysrq w**

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ツールが役立ちます。

- **ifconfig**
- **tcpdump**

  tcpdump コマンドは、ネットワークパケット関連の問題を感知します。ネットワーク認証における問題や異常を発見するのに役に立ちます。**wireshark** と言う名前で **tcpdump** のグラフィカルバージョンも存在します。

- **brctl**

  brctl は、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。次の例で使用しているコマンドを実行する場合は、まず root のアクセス権を取得しておく必要があります。

```
# brctl show
bridge-name    bridge-id          STP  enabled  interfaces
```

```
-------------------------------------------------------------------------
-------
virtbr0             8000.fefffff       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.fefffffffff
designated-root        8000.fefffffffff
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を追跡するコマンドです。
- **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、**vncviwer** 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 vnc-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、以下のログファイルを参照すると便利です。

- **/var/log/messages** (Red Hat Enterprise Linux 7 から、デフォルトで libvirtd ログがここに記録されます)
- **/var/log/audit/audit.log**

A.2. 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アクセス権があることを手動で確認することが必要になります。**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。**virsh dump** ファイルを作成するには、以下を実行します。

```
#virsh dump <domain> <corefilepath> [--bypass-cache] { [--live] | [--
crash] | [--reset] } [--verbose] [--memory-only]
```

ドメイン (ゲスト仮想マシンのドメイン名) と corefilepath (新たに作成されたコアダンプファイルの場所) が必須である一方、以下の引数はオプションになります。

- **--live**: 実行中のマシンにダンプファイルを作成し、これを一時停止しません。

- **--crash**: ゲスト仮想マシンを停止し、ダンプファイルを生成します。ゲスト仮想マシンは「Crashed」を理由として「Shut off」として一覧表示されます。**virt-manager** ではステータスが「Shut off」として一覧表示されることにも注意してください。
- **--reset**: ダンプが正常に行なわれた後にゲスト仮想マシンをリセットします。これらの 3 つのスイッチは相互に排他的であることに注意してください。
- **--bypass-cache**: ファイルシステムのキャッシュをバイパスするために O_DIRECT を使用します。
- **--memory-only**: ダンプファイルは、elf ファイルとして保存され、ドメインのメモリーと cpu 共通レジスター値のみが含まれます。このオプションは、ドメインがホストデバイスを直接使用する場合に非常に便利です。
- **--verbose**: ダンプの進捗が表示されます。

ダンププロセス全体は、**virsh domjobinfo** コマンドを使用して監視でき、**virsh domjobabort** を実行してキャンセルできます。

A.3. 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ーに保存するために 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ーは、バッファーが一杯になると新規エントリーによって上書きされます。

手順A.1 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 ディレクトリーにコピーします。

   ```
   # 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ーサイズ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.2 トレースバッファーの検査

1. **トレースバッファー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 というバッファーダ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.4. 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であり、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。「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状況を要求します。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** 又は、**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。例えば:

```
# 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 output

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 -l
kvm_ack_irq     0          0
kvm_age_page     0          0
kvm_apic     44         44
kvm_apic_accept_irq    12         12
kvm_apic_ipi     4          4
kvm_async_pf_completed    0          0
kvm_async_pf_doublefault   0          0
kvm_async_pf_not_present   0          0
kvm_async_pf_ready    0          0
kvm_cpuid     0          0
```

```
kvm_cr      0          0
kvm_emulate_insn    2          2
kvm_entry     56         56
kvm_eoi      12         12
kvm_exit     56         56個別の終了の理由が続きます。詳細は、kvm_exit (NAME) を参照してください。
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_fpu     4          4
kvm_hv_hypercall   0          0
kvm_hypercall    0          0
kvm_inj_exception   0          0
kvm_inj_virq    12         12
kvm_invlpga    0          0
kvm_ioapic_set_irq   0          0
kvm_mmio    0          0
kvm_msi_set_irq    0          0
kvm_msr     40         40
kvm_nested_intercepts   0          0
kvm_nested_intr_vmexit   0          0
kvm_nested_vmexit   0          0
kvm_nested_vmexit_inject         0          0
kvm_nested_vmrun                 0          0
kvm_page_fault                   0          0
kvm_pic_set_irq                  0          0
kvm_pio                          0          0
kvm_pv_eoi    12         12
kvm_set_irq                      0          0
kvm_skinit                       0          0
kvm_track_tsc                    0          0
kvm_try_async_get_page           0          0
kvm_update_master_clock   0          0
kvm_userspace_exit               0          0
kvm_write_tsc_offset   0          0
vcpu_match_mmio                  0          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

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

プロセッサー間の割り込みの回数です。

kvm_async_pf_completed

非同期ページフォールトの完了数です。

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

非同期ページフォールトの完了数です。

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 レジスターアクセス (CR0、CR3、CR4、CR8) の番号です。

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

APIC EOI (割り込み終点) 通知の数です。

kvm_exit

VM-exits の数です。

kvm_exit (NAME)

プロセッサー固有の個々の出口です。詳細は、プロセッサーの資料を参照してください。

kvm_fpu

KVM 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 MMIO 操作の回数です。

kvm_msi_set_irq

MSI 割り込みの回数です。

kvm_msr

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_vmrun

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 PIC コントローラー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 PIO 操作の回数です。

kvm_pv_eoi

準仮想 EOI 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

TSC 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回数です。

kvm_write_tsc_offset

TSC オフセットの書き込み回数です。

vcpu_match_mmio

SPTE でキャッシュされた 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.5. 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いるホスト物理マシンに対して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、完全仮想化ゲストに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** コマンドを使用すると表示させることができます。

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 の場合は **ttyS0** と呼ばれますが、Windows の場合は**COM1** になります。

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/logvol00 \
 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 /boot/grub2/grub.cfg
```

ゲストの再起動

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。

```
# virsh console
```

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** の **Serial 1** を選択します。

A.6. 仮想化のログファイル

- 完全仮想化ゲストの各ログは **/var/log/libvirt/qemu/** ディレクトリーにあります。各ゲストのログには *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。

仮想マシンマネージャでエラーに遭遇した場合は、**$HOME/.virt-manager** ディレクトリー内に存在する **virt-manager.log** ファイルの生成データを確認することができます。

A.7. 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加する必要があるかも知れません。デフォルトの設定では、最大８つのアクティブループが可能です。８つ以上のファイルベースゲストか、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/** ディレクトリ内で調節することができます。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。完全仮想化のシステムにループデバイスバックアップの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.8. ライブマイグレーションに関するエラー

ゲストのメモリー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これを再度転送し、完了に失敗する場合があります (converge)。この問題は現在のところ Red Hat Enterprise Linux 6 では解決される予定はありませんが、Red Hat Enterprise Linux 7.1 で解決される予定です。

現在のライブマイグレーション実装は、デフォルトの移行時間が 30 ミリ秒に設定されています。この値により、 残りを転送するために移行の最後で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集中する確率が高くなります。

A.9. BIOS で Intel VT-x と AMD-V の仮想化ハードウェア拡張を有効にする

このセクションでは、ハードウェア仮想化拡張を識別し、拡張が無効になっている場合は BIOS 内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 内で無効になっている可能性があります。一部のラップトップ販売店では、デフォルトで CPU 内の Intel VT-x 拡張が無効にされています。

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。

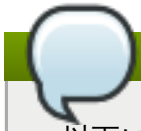
無効になっている仮想化拡張を有効にする方法については、次のセクションを参照してください。

仮想化拡張が BIOS で有効になっているか確認します。Intel VT や AMD-V の BIOS セッティングは、通常 **Chipset** または **Processor** のメニュー内にあります。メニュー名はこのガイドと異なる場合があります。仮想化拡張のセッティングは **Security Settings** のところ、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.3 BIOS 内で仮想化拡張を有効にする

1. コンピュータを再起動して、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 内で仮想化拡張を有効にする**



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ーの種類、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などを参照してください。

   a. **Processor** サブメニューを開きます。プロセッサーセッティングメニューは **Chipset** や **Advanced CPU Configuration**、 **Northbridge** などに隠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 内では無効にできないため、すでに有効になっているはずです。 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 とシステムの BIOS によっては、 これ以外の名前が付けられていることもあります。

   c. Intel VT-d あるいは 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、 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.10. 固有の MAC アドレスを新たに生成する

ゲスト仮想マシンに固有の MAC アドレスを新たに生成しなければならない場合があります。本ガイドの作成時点では、新しい MAC アドレスを生成するための使用できるコマンドラインツールはありません。このため、ここでは以下に示すスクリプトを使用してゲスト仮想マシンの新しい MAC アドレスを生成することができます。このスクリプトには **macgen.py** という名前を付けてゲスト仮想マシンに保存します。そのディレクトリーから **./macgen.py** コマンドを使ってスクリプトを実行します。これで新しい MAC アドレスが生成されます。出力例を以下に示します。

```
$ ./macgen.py
00:16:3e:20:b0:11
```

```
#!/usr/bin/python
# macgen.py script to generate a MAC address for guest virtual machines
#
import random
#
def randomMAC():
 mac = [ 0x00, 0x16, 0x3e,
  random.randint(0x00, 0x7f),
  random.randint(0x00, 0xff),
  random.randint(0x00, 0xff) ]
 return ':'.join(map(lambda x: "%02x" % x, mac))
#
print randomMAC()
```

ゲスト仮想マシンの新しい MAC を作成する別の方法

python-virtinst の組み込みモジュールを使って、ゲスト仮想マシンの設定ファイルで使用する新しい MAC アドレスと **UUID** を生成することもできます。

```
# echo  'import virtinst.util ; print\
 virtinst.util.uuidToString(virtinst.util.randomUUID())' | python
# echo  'import virtinst.util ; print virtinst.util.randomMAC()' |
python
```

上記のスクリプトは以下のようにスクリプトファイルとして実装することもできます。

```
#!/usr/bin/env python
#  -*- mode: python; -*-
print ""
print "New UUID:"
import virtinst.util ; print
virtinst.util.uuidToString(virtinst.util.randomUUID())
print "New MAC:"
import virtinst.util ; print virtinst.util.randomMAC()
print ""
```

A.11. KVM ネットワークのパフォーマンス

デフォルトでは、KVM 仮想マシンには、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) が割り当てられています。Red Hat Enterprise Linux のゲストには、デフォルトで virtio NIC が割り当ててられますが、Windows ゲストにはゲストタイプは指定されません。

rtl8139 仮想化 NIC はほとんどの環境で正常に機能します。しかし、このデバイスは、たとえば 10 Gigabi Ethernet ネットワークなどの一部のネットワークでは、パフォーマンスが低下する問題の原因となる場合があります。

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薦します。

手順A.4 virtio ドライバーに切り替える

1. ゲストの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 コマンドでは、使用するエディターは**$EDITOR** シェル変数で確定されます。
3. ネットワークインターフェースの設定セクションを見つけます。以下にネットワークインターフェースの設定セクションを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、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開します。

他のネットワークドライバーを使用して新しいゲストを作成する

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*と言うゲスト名) 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.12. libvirt による外部スナップショット作成の回避策

QEMU ゲストには 2 つのクラスのスナップショットがあります。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* によって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、および復帰の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ますが、qcow2 ディスクを使用するために *libvirt* で必要になります。このファイルタイプのもう一つの短所は、QEMU の改良点を受けることができない点にあります。

他方、外部スナップショットはオリジナル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、さらに QEMU のアクティブな改良点を受けることができます。*libvirt* では、**--disk-only** オプションまたは **--memspec** オプションを **snapshot-create-as** に対して使用する場合 (または明示的な XML ファイルを同じ機能を実行する **snapshot-create** に指定する場合) に作成されます。現在のところ、外部スナップショットは、*libvirt* はそれらを作成できるものの、それらを使ってそれ以上のタスクを実行できないため、一方的な操作になります。回避策については、こちら に説明されています。

A.13. 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ローカルでマシンに接続すると、下線 (_ 記号) などのタイプした文字がゲストコンソールでは正しく表示されない結果になることがあります。これは、必要なキーマップがデフォルトで正しくセットされていないことが理由です。

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).
atkbd.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** で問題のゲストを開きます。
- **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。
- **キーマップ** プルダウンメニューの **Auto** を **ja** に変更します。
- **適用** ボタンをクリックします。

また、 目的のゲストで **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 <target guest>** 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.14. ゲスト仮想マシンがシャットダウンに失敗します。

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は静かにシャットダウンしました。現在、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.5 ゲスト仮想マシンのゲストエージェントチャネルの設定

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.1 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.15. Windows XP ゲストに関する既知の問題

Windows XP ゲストを使ってデバイスの追加を行なった直後すぐにデバイスの削除を行なうと、 デバイスがイジェクトされず次のようなエラーが表示されます。 「The device (device name) cannot be stopped because of an unknown error. Since the device is still being used, do not remove it. (不明なエラーが発生したため、 デバイス (デバイス名) を停止することができません。 デバイスはまだ使用中のため、 取り出さないでください。)」。 最近の Windows OS バージョンのゲスト、 また既知の Linux ゲストではいずれもこの問題は見られません。 この問題が発生しないようにするため、 追加したデバイスを削除する場合は間をあけるようにしてください。

A.16. ゲスト仮想マシンの SMART ディスクモニタリングを無効にする

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.17. libguestfs 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.18. 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、**`Solution`** (解決法) の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

エラー	問題の概要	解決法
libvirtd failed to start	**libvirt** デーモンがスタートに失敗するが、**/var/log/messages** にはこのエラーに関する情報がない。	「**libvirtd** がスタート失敗」
Cannot read CA certificate	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。	「URI がハイパーバイザー接続に失敗する」
Failed to connect socket ... : Permission denied	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。	「URI がハイパーバイザー接続に失敗する」
他の接続エラー	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。	「URI がハイパーバイザー接続に失敗する」
Internal error guest CPU is not compatible with host CPU	ホストとゲストのプロセッサーが異なるため、ゲスト仮想マシンがスタートできない。	「ゲスト仮想マシンがスタートできずエラーが発生: **internal error guest CPU is not compatible with host CPU**」
Failed to create domain from vm.xml error: monitor socket did not show up.: Connection refused	ゲスト仮想マシン (またはドメイン) がスタートに失敗し、このエラーまたは同様のエラーを返す。	「ゲストがスタートに失敗しエラーが発生: **monitor socket did not show up**」
Internal error cannot find character device (null)	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。ゲスト仮想マシン用に設定されたシリアルコンソールがない、とレポートされます。	「**Internal error cannot find character device (null)**」
No boot device	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、ゲストの起動がストールするが、**QEMU** コマンドを直接使うとゲストは正常にスタートできる。	「ゲスト仮想マシンの起動がストールしエラーが発生: **No boot device**」
The virtual network *"default"* has not been started	*default* ネットワーク (またはローカル作成された別のネットワーク) がスタートできないと、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スタートに失敗する。	「Virtual network *default* has not been started」
ゲスト上の PXE ブート (または DHCP) が失敗	ゲスト仮想マシンは正常にスタートするが、DHCP から IP アドレスを取得できない、または PXE を使用したブートができない、もしくはその両方。この原因は多くの場合、ブリッジでの転送遅延時間が長く設定されているか、*iptables* パッケージとカーネルがチェックサム難号化ルールをサポートしないためです。	「ゲスト上の PXE ブート (または DHCP) が失敗」

エラー	問題の概要	解決法
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	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使用の設定後にはホストに接続できない。 この状況は、実際にはエラーではなく macvtap の定義済み動作です。	「ゲストは外部ネットワークにアクセスできるが、macvtap 使用時にはホストにアクセスできない」
Could not add rule to fixup DHCP response checksums on network *'default'*	この警告メッセージはほとんどの場合、無害ですが、間違って問題の証拠と見なされることが多くあります。	「Could not add rule to fixup DHCP response checksums on network *'default'*」
Unable to add bridge br0 port vnet0: No such device	このエラーメッセージと、同様の **Failed to add tap interface to bridge *'br0'*: No such device** 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 **type='ethernet'** (別名、ジェネリックイーサネット) インターフェースの設定後にゲスト仮想マシンがスタートしない。このエラーまたは同様のエラーが **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。	「ゲストがスタートできずエラーが発生: **warning: could not open /dev/net/tun**」
Unable to resolve address *name_of_host* service '49155': Name or service not known	**QEMU** ゲストマイグレーションが失敗し、このエラーメッセージが知らないホスト名とともに表示される。	「マイグレーションに失敗しエラーが発生 **Error: unable to resolve address**」
Unable to allow access for disk path /var/lib/libvirt/images/qemu.img: No such file or directory	**libvirt** がディスクイメージにアクセスできないため、ゲスト仮想マシンを移行できない。	「マイグレーションに失敗しエラーが発生: **Unable to allow access for disk path: No such file or directory**」
libvirtd のスタート時にゲスト仮想マシンがない	**libvirt** デーモンは正常にスタートしたが、**virsh list --all** を実行してもゲスト仮想マシンが見当たらない	「**libvirtd** のスタート時にゲスト仮想マシンが見当たらない」
Unable to connect to server at 'host:16509': Connection refused ... error: failed to connect to the hypervisor	**libvirtd** が接続のために TCP ポートをリッスンしている間に、ハイパーバイザーへの接続が失敗する。	「Unable to connect to server at 'host:16509': Connection refused ... error: failed to connect to the hypervisor」

エラー	問題の概要	解決法
一般的な XML エラー	**libvirt** 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 **libvirt** 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.18.1. libvirtd がスタート失敗

現象

libvirt デーモンが自動的にスタートしない。手動での**libvirt** デーモンのスタートも失敗

```
# /etc/init.d/libvirtd start
* 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
```

さらには、**/var/log/messages** にこのエラーの**'more info'** もない。

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確認します。

libvirtd がまだ正常にスタートしない場合、**/var/log/messages** ファイルに以下と同様のエラーが表示されます。

```
Feb  6 17:22:09 bart libvirtd: 17576: info : libvirt version:
0.9.9
Feb  6 17:22:09 bart libvirtd: 17576: error :
virNetTLSContextCheckCertFile:92: Cannot read CA certificate
```

```
'/etc/pki/CA/cacert.pem': No such file or directory
Feb  6 17:22:09 bart /etc/init.d/libvirtd[17573]: start-stop-
daemon: failed to start `/usr/sbin/libvirtd'
Feb  6 17:22:09 bart /etc/init.d/libvirtd[17565]: ERROR: libvirtd
failed to start
```

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、見つからない **cacert.pem** ファイルが TLS 認証として使用されたことが示されています。つまり、***--listen*** パラメーターがパスされています。

解決法

libvirt デー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 導入ガイド』 の認証の設定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** と **listen_tcp = 1** に設定する。デフォルト値は、**listen_tls = 1** と **listen_tcp = 0** 。
- ***--listen*** パラメーターをパスしない。**/etc/sysconfig/libvirtd.conf** で ***LIBVIRTD_ARGS*** 変数を変更する。

A.18.2. URI がハイパーバイザー接続に失敗する

サーバー接続時にいくつかの異なるエラーが発生する (例: **virsh** 実行時) 。

A.18.2.1. CA 証明書を読み込めない

現象

コマンド実行中に、以下のエラー (または同様のエラー) が表示される。

```
$ virsh -c name_of_uri list
error: Cannot read CA certificate '/etc/pki/CA/cacert.pem': No
such file or directory
error: failed to connect to the hypervisor
```

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様々な要素によって起こり得ます。

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** はそれぞれのホスト名 ***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。例えば、***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** トランスポートがデフォルトで**TLS** を選択します。これで証明書が作成されます。

接続が未設定

URI は正しいが (例えば、***qemu[+tls]://server/system***) マシン上で証明書が正しく設定されていない。TLS の設定に関する詳細は、**libvirt** ウェブサイトの Setting up **libvirt** for TLS を参照してください。

A.18.2.2. Failed to connect socket ... : Permission denied

現象

virsh コマンド実行中に、以下のエラー (または同様のエラー) が表示される。

```
$ virsh -c qemu:///system list
error: Failed to connect socket to '/var/run/libvirt/libvirt-
sock': Permission denied
error: failed to connect to the hypervisor
```

調査

ホスト名が指定されていないと、**QEMU** への接続にはデフォルトで UNIX ソケットが使用されます。root でこのコマンド実行時にエラーが発生しない場合、**/etc/libvirt/libvirtd.conf** のUNIX ソケットオプションの設定が間違っている可能性があります。

解決法

root 以外のユーザーでUNIX ソケットを使用して接続するには、**/etc/libvirt/libvirtd.conf** で以下のオプションを設定します。

```
unix_sock_group = <group>
unix_sock_ro_perms = <perms>
unix_sock_rw_perms = <perms>
```

注記

virsh を実行するユーザーは、**unix_sock_group** オプションで指定された **group** のメンバーである必要があります。

A.18.2.3. 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いて、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 トランスポート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.18.3. ゲスト仮想マシンがスタートできずエラーが発生:internal error guest CPU is not compatible with host CPU

現象

Intel Core i7 プロセッサー (**virt-manager** が **Nehalem** と呼ぶ。また、以前の Core 2 Duo の場合は **Penryn** と呼ぶ) 上で動作する KVM ゲスト (またはドメイン) は、**virt-manager** を使用して作成されます。インストール後に、ゲストのプロセッサーはホストの CPU に適合するように変更されます。するとゲストはスタートできず、以下のエラーメッセージを表示します。

```
2012-02-06 17:49:15.985+0000: 20757: error :
qemuBuildCpuArgStr:3565 : internal error guest CPU is not
compatible with host CPU
```

また、**virt-manager** で **Copy host CPU configuration** をクリックすると、**Nehalem** や **Penryn** ではなく、Pentium III が表示されます。

調査

/usr/share/libvirt/cpu_map.xml ファイルは、各 CPU モデルを定義するフラグをリスト表示します。**Nehalem** および **Penryn** の定義には、以下が含まれます。

```
<feature name='nx'/>
```

その結果、 CPU を **Nehalem** または **Penryn** と特定するには、**NX** (または **No eXecute**) フラグを提示する必要があります。しかし、**/proc/cpuinfo** にはこのフラグが欠けています。

解決法

ほとんどすべての新規 BIOSes では、**No eXecute** ビットの有効化や無効化が可能です。しかし、無効となっている場合、このフラグをレポートしない CPU もあり、そのため **libvirt** は別の CPU を検出します。この機能を有効にすると、**libvirt** が正しい CPU にレポートするように指示します。この問題の詳細な指示に関しては、ご自分のハードウェアの資料を参照してください。

A.18.4. ゲストがスタート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) がスタートに失敗し、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が示しているのは、以下の点です。

1. **libvirt** は動作している
2. **QEMU** プロセスはスタートに失敗した

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

エラーの詳細を理解するために、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.monito
r,server,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含まれています。

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を喪失するなど) 、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移行することで **QEMU** のスタート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ステップを取り、**virsh start --force-boot *name_of_guest*** を追加して管理保存イメージも迂回します。

A.18.5. Internal error cannot find character device (null)

現象

ゲスト仮想マシンのコンソールに接続を試みる際にこのエラーメッセージが表示されます。

```
# 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設定されたシリアルコンソールがないことを示

しています。

解決法

ゲストの XML ファイル内でシリアルコンソールを設定します。

手順A.6 ゲストの XML ファイル内でのシリアルコンソール設定

1. **virsh edit** を使用して、以下の XML をゲストの仮想マシンの 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にコンソールを設定します。

   これを行うには、ゲスト仮想マシンにログインして **/boot/grub/grub.conf** ファイルを直接編集するか、**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.18.6. ゲスト仮想マシンの起動がストールしエラーが発生: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、ゲストの起動がストールし、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スタートする。

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
```

しかし、**QEMU** を直接使用してゲスト仮想マシンを起動したコマンドラインではバスの種類に **virtio** を使用したことを示している

```
# 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用に **libvirt** が生成した ***bus=*** がゲストの XMLにあることに注意。

```
<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.7 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てゲストの XML を編集し、ディスクのバスの種類を訂正します。

A.18.7. Virtual network *default* has not been started

現象

通常、*libvirt* パッケージの一部として *default* 名で仮想ネットワークの設定がインストールされており、**libvirtd** のスタート時に自動スタートするように設定されています。

default ネットワーク (またはローカルで作成されたネットワーク) がスタートできない場合、接続にそのネットワークを使うように接続されている仮想マシンもスタートに失敗し、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スタート失敗でよくある理由の一つは、そのネットワーク上でクライアントからのDHCP および DNS リクエストに応えるために必要な dnsmasq インスタンスがスタート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 shell から **virsh net-start default** を実行し、*default* 仮想ネットワークをスタートします。

この方法で仮想ネットワークが正常にスタートしない場合、**/var/log/libvirt/libvirtd.log** を開いて完全な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ワイドの dnsmasq インスタンスである可能性が高いです。このインスタンスは、**libvirt** 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最も注目すべき点は、**dnsmasq** と **exit status 2** です。

```
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。

物理ネットワークで DHCP 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 行のうちのどちらかを追加またはコメント解除します。3 行すべての追加またはコメント解除はしないでください。

```
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、*default* ネットワークを **virsh net-start default** コマンドでスタートします。

仮想マシンを起動します。

A.18.8. 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スタートするが、DHCP から IP アドレスを取得できない、または PXE を使用したブートができない、もしくはその両方。このエラーには 2 つの一般的な原因があります。ブリッジで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難号化ルールをサポートしない場合です。

ブリッジの転送遅延時間が長い

調査

このエラーにおける最も一般的な原因です。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、ブリッジは少なくとも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の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、クライアントの作業は失敗し、ゲストは (PXE の場合) 起動に失敗するか (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、またはブリッジの STP を無効にする、もしく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 (**libvirt** が使用する最も一般的なブリッジのユースケース) に使用されている場合にだけ、適用されます。

ゲストが **libvirt** 管理の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。例えば、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、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もしくは編集して、STP を on に、遅延を 0 秒にします。

```
STP=on
DELAY=0
```

設定ファイルの変更後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は最終的には root ブリッジ で設定されている遅延時間をリセットします。この場合の唯一の解決法は、*name_of_bridge* 上で完全に STP を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難号化ルールをサポートしない場合

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'`** が含まれている。

- 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。

- ホスト上で直接実行中の DHCP サーバーからホストが 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 またはそれ以降のもの。

  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と、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

はじめの 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は、ホストからゲストへ送信された UDP パケットに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に見えるようになります。

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* もしくはそれ以降に更新することです。別の方法としては、この特定のゲストの **vhost-net** ドライバーを無効にしてしまうことが最も的確な修正方法です。これを行うには、以下のコマンドでゲスト設定を編集します。

```
virsh edit name_of_guest
```

<interface> セクションに **<driver>** 行を変更もしくは追加します。

```
<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。

これで解決されない場合、問題の原因は **firewalld** とデフォルトの **libvirt** ネットワークの競合である可能性があります。

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.18.9. ゲストは外部ネットワークにアクセスできるが、macvtap 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 (別名 ***type='direct'***) ネットワークインターフェース使用の設定後にはホストに接続できない。

調査

Virtual Ethernet Port Aggregator (VEPA) や VN-Link 対応スイッチに接続していない時でも、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は物理ネットワークに直接接続することができます。従来のホストブリッジデバイスにつきものの設定問題 (または、*NetworkManager* の非互換性) がありません。

しかし、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、ゲストは自分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 スタックに跳ね返ってくることができません。さらには、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 ブリッジに跳ね返って来ず、ゲストに転送することができません。

解決法

libvirt を使って、分離し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用に 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したネットワーク上で直接通信できる一方、*NetworkManager* との互換性も維持できます。

手順A.8 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。自分のネットワーク上で 192.168.254.0/24 が既に使われている場合、別のネットワークを選びます。

```
<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>
```

2. 以下のコマンドでネットワークを作成します: **`virsh net-define /tmp/isolated.xml`**
3. **`virsh net-autostart isolated`** コマンドでネットワークの自動スタートを設定します。
4. **`virsh net-start isolated`** コマンドで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'>
  <source network='isolated'/>
  <model type='virtio'/>
</interface>
```

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 の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に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。

A.18.10. 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.18.11. 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
```

例えばブリッジ名が *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 およびそれ以前では、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
```

例えばブリッジ名が *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 **ifconfig *br0*** 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 を使って既存のブリッジもしくはネットワークの設定を編集するか、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。

例えば、**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 およびそれ以降では、**virsh iface-bridge** コマンドでブリッジデバイスが作成できます。これで作成されるのは、**eth0** のある ブリッジデバイス *br0* で、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、以下のコマンドでオリジナルの **eth0** 設定を回復することもできます。

```
virsh iface-unbridge br0
```

ホストブリッジを手動で作成

libvirt の古いバージョンでは、ホスト上に手動でブリッジデバイスを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キング」 を参照してください。

A.18.12. ゲストがスタート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'** (別名、ジェネリックイーサネット) インターフェースの設定後にゲスト仮想マシンがスタートしない。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、もしく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

ジェネリック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ー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、時には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も必要です。例えば、 **openvswitch** は **libvirt** では *libvirt-0.9.11* までサポートされていなかったので、**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スタートしなくなります。

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からです。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 はこれを妨げるためのチェックをいくつか確立しています。(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** にパスします。)

解決法

ホストシステムがジェネリック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.9 ホストシステムがジェネリック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** に設定して SELinux to permissive とします。

```
# 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 shell から **setenforce permissive** コマンドを実行します。

3. **/etc/libvirt/qemu.conf** で以下の行を追加もしくは編集します。

```
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

これらのステップは **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'>** 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 Security-Enhanced Linux User Guide』を参照してください。

A.18.13. マイグレーションに失敗しエラーが発生 `Error: unable to resolve address`

現象

QEMU ゲストマイグレーションが失敗し、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
```

例えば、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" はどこにも使っておらず、違和感があります。

調査

マイグレーション中に、移行先ホスト上で稼働している**libvirtd** はマイグレーションデータの受信を予想してい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 **libvirtd** 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'* に設定していました。何らかの理由で、そのホスト上で実行中の **libvirtd** 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 **libvirtd** が名前を解決することを期待して、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ったりすると、このようなことが発生します。

マイグレーションデータに使用されるアドレスは、移行先 **libvirtd** への接続に使用されるアドレス (例えば、 **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 **libvirtd** と通信するために、移行元の**libvirtd** 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

最善の解決法は、DNS を正しく設定し、マイグレーション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です。

DNS をこのように設定できない場合は、各ホスト上の **/etc/hosts** ファイルにマイグレーションに使用するすべてのホストのリストを手動で追加することができます。しかし、ダイナミックな環境ではそのようなリストの一貫性の維持は困難です。

どの方法でもホスト名を解決可能にできない場合、**virsh migrate** はマイグレーション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 **libvirtd** 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 (例えば、ファイアウォール設定のため) 、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
```

別のオプションでは、トンネル化したマイグレーションを使用します。トンネル化したマイグレーションでは、マイグレーション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 **libvirtd** との通信で使用され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.18.14. マイグレーション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
```

例えば、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、マイグレーション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) 。マイグレーション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マウントしてセットアップします。一番簡単な方法は、NFS を使用することです。

手順A.10 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 サーバーはマイグレーションに関与しているホストの一つでかまいません。

```
# 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キスポートされたディレクトリーをマウントします。例えば、NFS サーバーの IP アドレスが 192.168.122.1 なら、ディレクトリーを以下のコマンドでマウントします。

```
# 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 を使っている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、両方のホスト上の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マイグレーション中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オーナーやアクセス権、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.18.15. libvirtd のスタート時にゲスト仮想マシンが見当たらない

現象

libvirt デーモンは正常にスタートしたが、ゲスト仮想マシンが見当たらない

```
# virsh list --all
 Id    Name                           State
----------------------------------------------------
#
```

調査

この問題の原因は、いくつも考えられます。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する。

```
# 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 shell 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、**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、有効となっ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
```

ご自分のハードウェアの 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、ご自分のハードウェア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
```

例えば、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にデフォルトで話しかける場合に、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リストを表示させます。

```
# virsh list --all
```

A.18.16. Unable to connect to server at 'host:16509': Connection refused ... error: failed to connect to the hypervisor

現象

libvirtd 接続で 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: unable to connect to server at 'host:16509': Connection
refused
error: failed to connect to the hypervisor
```

/etc/libvirt/libvirtd.conf で設定を変更した後でも、**libvirt** デーモンが TCP ポートをリッスンしない。

```
# grep listen_ /etc/libvirt/libvirtd.conf
listen_tls = 1
listen_tcp = 1
listen_addr = "0.0.0.0"
```

しかし、設定変更も **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** オプションなしにスタート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     27314  0.0  0.0 1000920 18304 ?       Sl   Feb16   1:19
libvirtd --daemon
```

アウトプットには **--listen** は含まれません。

解決法

--listen オプションでデーモンをスタートします。

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。

```
#LIBVIRTD_ARGS="--listen"
```

以下の行で **libvirtd** サービスを再スタートします。

```
# /etc/init.d/libvirtd restart
```

A.18.17. 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.18.17.1. 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も時には必要になります。編集目的でゲストの 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** が解析します。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チェックします。

```
# xmllint --noout config.xml
```

```
# virt-xml-validate config.xml
```

エラーが表示されない場合、XML 記述子は整形式で **libvirt** スキーマに合致しています。スキーマはすべての制約を捕捉しませんが、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 が保存する 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ています。ドキュメントは自動生成され、手動での編集は推奨されません。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で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 を常に読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 要素で混乱する可能性があります。

A.18.17.2. 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 行から成っています。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、上の行で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象徴的な名前で、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数です。

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.18.17.2.1. 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 **<** 記号の後で、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数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 行目のテキストをみつけます。

```
<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、新たな要素を終了させます。

A.18.17.2.2. 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て、閉じる必要があります。

解決法

すべての属性値文字列を正しく開いて、閉じます。

A.18.17.2.3. 開始および終了タグのミスマッチ

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>** にタグのミスマッチ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記号で、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 タグのミスマッチが原因で発生します。すべての XML タグは、開始および終了タグでマッチする必要があります。

XML タグの他のミスマッチ例

以下の例では同様のエラーメッセージが生成され、XML タグのミスマッチの例が示されています。

このスニペットには、終了タグ (**</name>**) がないため、**<features>** の不一致エラーが含まれていません。

```
<domain type='kvm'>
 ...
 <features>
   <acpi/>
   <pae/>
 ...
 </domain>
```

下の例では、**</name>** に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、終了していることを確認します。

A.18.17.2.4. 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.18.17.3. 論理および設定エラー

適切にフォーマットされた XML ドキュメントで構文が正しくても、**libvirt** が解析できないエラーを含んでいる場合があります。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.18.17.3.1. 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、理解できる構築のみを探し他のものは全て無視するので、**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** コマンドにわたす前に確認します。**libvirt** 開発者は、**libvirt** とバンドルされている XML スキーマ一式を維持します。これは、**libvirt** が使用する XML ドキュメントで許される構築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
```

このコマンドが通れば、多くの場合は **libvirt** がXML からの全構築を理解します。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。例えば、**virsh dump** コマンドの結果として **libvirt** が生成した XML は、エラーなしで有効になります。

A.18.17.3.2. 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' ディスクイメージは、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、代わりに **type='file'** を使用します。

付録B 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。

B.1. オンラインリソース

- http://www.libvirt.org/ は **libvirt** 仮想化 API の公式 web サイトになります。
- http://virt-manager.et.redhat.com/ 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 Enterprise Virtualization

  http://www.redhat.com/products/cloud-computing/virtualization/
- Red Hat 提供のドキュメントです。

  https://access.redhat.com/site/documentation/
- 仮想化技術に関する概要です。

  http://virt.kernelnewbies.org
- Red Hat の最新技術に関する記載です。

  http://et.redhat.com

B.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>*** — **virsh** 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他、 **libvirt** 仮想化ライブラリ API に関して総合的に解説されています。
- **/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相互作用できるプログラムを作成できるようになります。
- **/usr/share/doc/python-virtinst-*<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。

付録C NetKVM ドライバーパラメーター

NetKVM ドライバーは、インストール後に自分の環境に適合するように設定できます。このセクションにあるパラメーターは、Windows の **デバイスマネージャー** (**devmgmt.msc**) で設定できます。

重要

ドライバーのパラメーターを修正すると、Windows はそのドライバーを再ロードします。これは既存のネットワークアクティビティーに割り込みます。

手順C.1 NetKVM パラメーターの設定

1. **デバイスマネージャー を開く**

   スタート ボタンをクリックします。右側のペインで**コンピューター** を右クリックし、**管理** をクリックします。要求されたら、**ユーザーアカウント制御** ウィンドウの **続行** をクリックします。これで **コンピューターの管理** ウィンドウが開きます。

   コンピューターの管理 ウィンドウの左側のペインで、**デバイスマネージャー** をクリックします。

2. **正しいデバイスの特定**

   コンピューターの管理 ウィンドウの中央のペインで、**ネットワークアダプター** の横にある**+** 記号をクリックします。

   Red Hat VirtIO Ethernet Adapter デバイスの下にあるリストで、**NetKVM** をダブルクリックします。これでそのデバイスの **設定** ウィンドウが開きます。

3. **デバイスパラメーターの表示**

   設定 ウィンドウで **詳細** タブをクリックします。

4. **デバイスパラメーターの修正**

   修正するパラメーターをクリックし、そのパラメーターのオプションを表示させます。

   オプションを適切に修正し、**OK** をクリックして変更を保存します。

C.1. 設定可能な NetKVM パラメーター

ロギングパラメーター

Logging.Enable

ロギングが有効かどうかを決定するブール値。デフォルト値は **1** (有効) 。

Logging.Level

ロギングレベルを定義する整数。この整数が増すにつれ、ログの冗長も増します。デフォルト値は、**0** です (エラーのみ) 。**1-2** は設定メッセージを追加します。**3-4** はパケットフロー情報を追加します。**5-6** は割り込みおよび DPC レベル追跡情報を追加します。

重要

高いロギングレベルは、ゲスト仮想マシンのパフォーマンスを低下させます。

Logging.Statistics(sec)

ログ統計が印刷されるかどうかと各期間の統計印刷の秒単位の間隔を定義する整数。デフォルト値は、**0** (ロギング統計なし) 。

初期パラメーター

Assign MAC

準仮想化 NIC 用にローカルで管理された MAC アドレスを定義する文字列。デフォルトでは設定されていません。

Init.ConnectionRate(Mb)

接続率をメガバイトで表示する整数。Windows 2008 およびそれ以降のデフォルト値は **10000** 。

Init.Do802.1PQ

Priority/VLAN タグ取り込みおよび削除サポートを有効化するブール値。デフォルト値は **1** (有効) 。

Init.UseMergedBuffers

マージ可能な RX バッファーを有効化するブール値。デフォルト値は **1** (有効) 。

Init.UsePublishEvents

公開されたイベント使用を有効化するブール値。デフォルト値は **1** (有効) 。

Init.MTUSize

maximum transmission unit (MTU) を定義する整数。デフォルト値は **1500** 。500 から 65500 の間であれば使用可能。

Init.IndirectTx

間接的なリング記述子の使用を制御。デフォルト値は **Disable** で、これは間接的なリング記述子の使用を無効にします。他の有効な値は **Enable** で、これは間接的なリング記述子の使用を可能にします。また、**Enable*** は、間接的なリング記述子の条件付き使用を可能にします。

Init.MaxTxBuffers

割り当てられる TX リング記述子の量を表示する整数。デフォルト値は、**1024** 。有効な値は、16、32、64、128、256、512、1024 のいずれか。

Init.MaxRxBuffers

割り当てられる RX リング記述子の量を表示する整数。デフォルト値は、**256** 。有効な値は、16、32、64、128、256、512、1024 のいずれか。

Offload.Tx.Checksum

TX チェックサムオフロードモードを指定

TX チェックサムオフロードモードを指定。

Red Hat Enterprise Linux 7 でのこのパラメーターの有効な値は以下のとおりです。**All** (デフォルト): IPv4 と IPv6 の両方で IP、TCP、UDP のチェックサムオフロードを有効にします。**TCP/UDP(v4,v6)**: IPv4 と IPv6 の両方で TCP と UDP のチェックサムオフロードを有効にします。**TCP/UDP(v4)**: IPv4 のみで TCP と UDP のチェックサムオフロードを有効にします。**TCP(v4)**: IPv4 のみで TCP のチェックサムオフロードだけを有効にします。

Offload.Tx.LSO

TX TCP Large Segment Offload (LSO) を有効にするブール値。デフォルト値は **1** (有効) 。

Offload.Rx.Checksum

RX チェックサムオフロードモードを指定

Red Hat Enterprise Linux 7 でのこのパラメーターの有効な値は以下のとおりです。**All** (デフォルト): IPv4 と IPv6 の両方で IP、TCP、UDP のチェックサムオフロードを有効にします。**TCP/UDP(v4,v6)**: IPv4 と IPv6 の両方で TCP と UDP のチェックサムオフロードを有効にします。**TCP/UDP(v4)**: IPv4 のみで TCP と UDP のチェックサムオフロードを有効にします。**TCP(v4)**: IPv4 のみで TCP のチェックサムオフロードだけを有効にします。

テストおよびデバッグパラメーター

重要

テストおよびデバッグパラメーターは、テストもしくはデバッグのみに使用することをお勧めします。本番環境での使用は推奨されません。

TestOnly.DelayConnect(ms)

スタート時に接続を遅らせる時間 (ミリ秒単位) 。デフォルト値は、**0** 。

TestOnly.DPCChecking

DPC 確認モードを設定。**0** (デフォルト値) は DPC 確認を無効にします。**1** は DPC 確認を有効にします。各ハングテストは DPC アクティビティーを検証し、 DPC が作成されたかのように行動します。**2** は、デバイス割り込みステータスを消去するほかは、**1** と同じです。

TestOnly.Scatter-Gather

スキャッター/ギャザー機能が有効かどうかを判断するブール値。デフォルト値は **1** (有効) 。この値を **0** に設定すると、スキャッター/ギャザー機能とすべての依存機能が無効になります。

TestOnly.InterruptRecovery

割り込み回復が有効かどうかを決定するブール値。デフォルト値は **1** (有効) 。

TestOnly.PacketFilter

パケットフィルタリングが有効かどうかを決定するブール値。デフォルト値は **1** (有効) 。

TestOnly.BatchReceive

パケットがバッチかそれとも一回で受信されたかを決定するブール値。デフォルト値は **1** で、パケットのバッチ受信が有効。

TestOnly.Promiscuous

プロミスキャスモードが有効かどうかを決定するブール値。デフォルト値は **0** (無効) 。

TestOnly.AnalyzeIPPackets

送信 IP パケットのチェックサムフィールドがデバッグ目的でテストされ、検証されたかどうかを決定するブール値。デフォルト値は **0** (確認なし) 。

TestOnly.RXThrottle

単一 DPC で処理される受信パケット数を決定する整数。デフォルト値は、**1000** 。

TestOnly.UseSwTxChecksum

ハードウェアチェックサムが有効かどうかを決定するブール値。デフォルト値は **0** (無効) 。

付録D 仮想ホストメトリックスデーモン (vhostmd)

vhostmd (Virtual Host Metrics Daemon: 仮想ホストメトリックスデーモン) の使用により、仮想マシンがその稼働元であるホストの限定的情報を認識できるようになります。このデーモンは、Red Hat Enterprise Linux for SAP でのみ提供されます。

ホスト内では、デーモン (**vhostmd**) が稼働しており、メトリックスを定期的にディスクイメージに書き込みます。このディスクイメージは、ゲスト仮想マシンに読み込み専用としてエキスポートされます。ゲスト仮想マシンはディスクイメージを読み取ってメトリックスを認識することができます。簡単な同期化により、ゲスト仮想マシンが古いか、または破損したメトリックスを読み込むのを防ぎます。

システム管理者は、ゲスト仮想マシンごとに使用できるメトリックスを選択します。さらに、システム管理者は 1 つ以上のゲスト仮想マシンがメトリック設定にアクセスするのを防ぐことができます。

vhostmd と **vm-dump-metrics** を使用する必要のあるお客様は、パッケージをインストールするのに、SAP を実行する Red Hat Enterprise システムをカスタマーポータルか、または Subscription Manager の「RHEL for SAP」チャンネルにサブスクライブさせるようにするために「Red Hat Enterprise Linux for SAP Business Applications」のサブスクリプションが必要になります。Red Hat カスタマーポータルの以下のナレッジベース記事では、RHEL 上の vhostmd のセットアップについて説明しています。https://access.redhat.com/knowledge/solutions/41566

付録E 改訂履歴

改訂 2-0.2 **Tue Nov 24 2015** **Aiko Sasaki**
ピアレビュー実施およびコメント反映

改訂 2-0.1 **Mon Nov 23 2015** **Aiko Sasaki**
翻訳ファイルを XML ソースバージョン 1-201 と同期

改訂 1-201 **Wed Feb 18 2015** **Laura Novich**
7.1 GA リリース向けのバージョン

改訂 1-200 **Thu Feb 12 2015** **Charlie Boyle**
フィードバックに基づくセクション 17.4.2 の更新

改訂 1-199 **Wed Feb 11 2015** **Laura Novich**
BZ1140477: GPU デバイスの割り当てを含むデバイスの章の更新

改訂 1-198 **Wed Feb 11 2015** **Laura Novich**
BZ1136603: SME フィードバックに従って KVM マイグレーションを更新

改訂 1-197 **Tue Feb 10 2015** **Charlie Boyle**
17.4.2 の警告および情報を更新

改訂 1-196 **Fri Feb 6 2015** **Charlie Boyle**
スイッチが機能しないため PCI ホールの段落を削除

改訂 1-195 **Fri Feb 6 2015** **Charlie Boyle**
SeaBios 使用時の PCI 情報を更新

改訂 1-194 **Tue Feb 3 2015** **Laura Novich**
BZ1136603: SME フィードバックに従って KVM マイグレーションを更新

改訂 1-193 **Tue Feb 3 2015** **Laura Novich**
トレースを実行するための systemtap の指示を作成

改訂 1-192 **Tue Feb 3 2015** **Laura Novich**
kvm-stat を更新

改訂 1-191 **Wed Jan 21 2015** **Dayle Parker**
SME フィードバックに基づき QEMU ゲストエージェントの章を編集 (BZ#1148427)。

改訂 1-190 **Wed Jan 21 2015** **Dayle Parker**
NPIV コンテンツを編集 (BZ#879895)。

改訂 1-189 **Wed Jan 21 2015** **Scott Radvan**
systemctl 構文を修正 (BZ#1183965)。

改訂 1-188 **Wed Jan 21 2015** **Scott Radvan**
max_anonymous_clients コンテンツを追加 (BZ#1086175)。

改訂 1-187.8 **Mon Jan 19 2015** **Charlie Boyle**

セクション 29.18.9.14 の peak 属性を更新 (BZ#1073265)。

改訂 1-187.5 **Mon Jan 19 2015** **Charlie Boyle**
セクション 2.1 の 3 つのドキュメントリンクを更新 (BZ#1073244)。

改訂 1-187 **Wed Jan 14 2015** **Dayle Parker**
NPIV コンテンツを追加 (BZ#879895)。

改訂 1-186 **Tue Jan 13 2015** **Dayle Parker**
ベータ向けバージョンのドキュメントを更新。

改訂 1-185 **Mon Jan 12 2015** **Laura Novich**
双方向の互換性を反映するために KVM マイグレーションバージョン表を更新。

改訂 1-184 **Mon Jan 12 2015** **Scott Radvan**
7.1 で VCPU 制限 (240) を導入することに言及 (BZ#1134198)。

改訂 1-183 **Wed Jan 7 2015** **Dayle Parker**
virsh qemu-agent-command (未対応のコマンド) を詳述した 2 つのセクションを削除および QEMU ゲストエージェントの章を編集 (BZ#1148427)。

改訂 1-182 **Fri Dec 12 2014** **Dayle Parker**
Red Hat Enterprise Linux 7.1 Beta リリースの再ビルド。
virtualization Kickstart パッケージの一覧を更新 (BZ#1151682)。

改訂 1-181 **Mon Dec 8 2014** **Dayle Parker**
RHEL 7 スプラッシュページに sort_order を新たに実装するために更新。

改訂 1-180 **Fri Dec 5 2014** **Dayle Parker**
RHEL 7.1 Beta リリースのパブリッシュ。

改訂 1-179 **Thurs Dec 4 2014** **Dayle Parker**
「iSCSI ストレージプールの保護」手順を編集 (BZ#1051267)。

改訂 1-177 **Wed Dec 3 2014** **Dayle Parker**
SME フィードバックに基づき「iSCSI ストレージプールの保護」手順を編集 (BZ#1051267)。

改訂 1-175 **Wed Nov 26 2014** **Dayle Parker**
iSCSI ストレージプール手順を編集 (BZ#1051267)。
ガイドの改訂履歴を削除。

改訂 1-170 **Sun Nov 16 2014** **Laura Novich**
1159611 を修正

改訂 1-169 **Sun Nov 16 2014** **Laura Novich**
1159611 を修正

改訂 1-168 **Thu Nov 13 2014** **Laura Novich**
フィードバックに基づき virsh dump コマンドを修正

改訂 1-167 **Thu Nov 13 2014** **Laura Novich**

1084371 を修正

改訂 1-166 **Thu Nov 13 2014** **Laura Novich**
Virt Manager の章を修正 (BZ1084371)

改訂 1-165 **Wed Nov 12 2014** **Dayle Parker**
iSCSI ストレージプール手順を編集 (BZ#1051267)。

改訂 1-164 **Thu Nov 6 2014** **Laura Novich**
フィードバックに基づき 1084371 を修正

改訂 1-163 **Thu Nov 6 2014** **Laura Novich**
1083342 を修正 - kvm clock の章を変更

改訂 1-162 **Wed Nov 5 2014** **Laura Novich**
1159613 を修正 - 警告を Libvirt デバイスに追加

改訂 1-161 **Wed Nov 5 2014** **Dayle Parker**
SME フィードバックに基づきネットワークブリッジコンテンツを編集 (BZ#1113329)。

改訂 1-160 **Thurs Oct 30 2014** **Dayle Parker**
iSCSI ストレージプール手順を編集 (BZ#1051267)。

改訂 1-159 **Thurs Oct 30 2014** **Dayle Parker**
SME フィードバックに基づきネットワークブリッジコンテンツを編集 (BZ#1113329)。

改訂 1-158 **Wed Oct 29 2014** **Scott Radvan**
virt-win-reg セクションをレビュー、リモート URL を使用する新しい機能を指定 (BZ#1127233)。

改訂 1-157 **Wed Oct 29 2014** **Dayle Parker**
Red Hat Enterprise Linux Networking Guide に言及するためにブリッジを使用するため NetworkManager を無効にする方法についての注記を入れ替える (BZ#1113329)。

改訂 1-156 **Mon Oct 27 2014** **Laura Novich**
フィードバックに基づき https://bugzilla.redhat.com/show_bug.cgi?id=1084371 を修正

改訂 1-155 **Fri Oct 24 2014** **Scott Radvan**
7.1 におけるライブ移行 auto-convergence のサポートを指定 (BZ#1059555)。

改訂 1-154 **Wed Oct 22 2014** **Laura Novich**
お客様のリクエストに基づき 1090694 を修正

改訂 1-153 **Mon October 22 2014** **Laura Bailey**
Inovich のビルドのために更新。

改訂 1-152 **Mon October 17 2014** **Scott Radvan**
7.0 async 更新のためのパブリッシュ。

改訂 1-151 **Mon October 13 2014** **Scott Radvan**
ネットワークブリッジの作成方法を示すセクションに virt-install の例のリンクを追加 (BZ#1118252)。

改訂 1-150 **Thu October 9 2014** **Scott Radvan**

virsh シャットダウンエラーを回避するために guest-agent を使用する方法についての詳細を追加 (BZ#1149534)。

改訂 1-149 **Wed October 8 2014** **Scott Radvan**

USB3/XHCI サポートおよびドメイン XML 例についての詳細を追加 (BZ#1135870)。

改訂 1-148 **Fri September 5 2014** **Jodi Biddle**

RHN についての言及を削除し、Subscription Manager の言及を随時追加 (BZ#1133784)。

改訂 1-147 **Tue August 5 2014** **Jodi Biddle**

QEMU の互換性の問題を避けるために virtio ドライバーを最新の状態に保つようにとユーザーに指示するモを追加 (BZ#983321)。

改訂 1-146 **Fri July 18 2014** **Tahlia Richardson**

「新規デバイス用に KVM virtio ドライバーを使用する」手順を訂正および再構成 (BZ#1101857)。

改訂 1-145 **Wed Jun 11 2014** **Tahlia Richardson**

リンク切れを修正

改訂 1-144 **Tue Jun 3 2014** **Laura Novich**

7.0 GA リリース向け最終バージョン