Red Hat Enterprise Linux 7

仮想化の導入および管理ガイド

RHEL 物理マシンでの仮想マシンのインストール、設定、および管理

Last Updated: 2023-10-19

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

RHEL 物理マシンでの仮想マシンのインストール、設定、および管理

Jiri Herrmann
Red Hat Customer Content Services
jherrman@redhat.com

Parth Shah
Red Hat Customer Content Services
pashah@redhat.com

Yehuda Zimmerman
Red Hat Customer Content Services

Laura Novich
Red Hat Customer Content Services

Dayle Parker
Red Hat Customer Content Services

Scott Radvan
Red Hat Customer Content Services

Tahlia Richardson
Red Hat Customer Content Services

法律上の通知

概要

本ガイドでは、Red Hat Enterprise Linux 7 マシンを仮想化ホストシステムとして機能するように設定する方法と、KVM ハイパーバイザーを使用してゲスト仮想マシンをインストールおよび設定する方法を説明します。他のトピックには、PCI デバイスの設定、SR-IOV、ネットワーク、ストレージ、デバイス、およびゲストの仮想マシン管理やトラブルシューティング、互換性、制限が含まれます。ゲスト仮想マシンで実行する必要のある手順は、明示的に記載されています。 本ガイドのすべての手順は、特に記載がない限り、AMD64 または Intel 64 のホストマシンで実行することが意図されています。AMD64 および Intel 64 以外のアーキテクチャーで Red Hat Enterprise Linux 7 仮想化を使用する場合は、 を参照してください。 Red Hat が提供する仮想化ソリューションの概要は、Red Hat Enterprise Linux 7 Virtualization Getting Started Guide を参照してください。

目次

パート I. DEPLOYMENT

この部分では、仮想ホストシステムとして機能する Red Hat Enterprise Linux 7 マシンをインストールして設定する方法と、KVM ハイパーバイザーを使用してゲスト仮想マシンをインストールして設定する方法を説明します。

第1章 システム要件

仮想化は、Intel 64 および AMD64 アーキテクチャー上の Red Hat Enterprise Linux 7 の KVM ハイパーバイザーで利用できます。本章では、仮想マシン (VM とも呼ばれる) のシステム要件を紹介します。

仮想化パッケージのインストール方法は、*2章 仮想化パッケージのインストール*を参照してください。

1.1. ホストシステム要件

ホストシステムの最小要件

- 6 GB の空きディスク容量
- 2 GB RAM

推奨されるシステム要件

- 仮想化 CPU とホストごとに 1 つのコアまたはスレッド
- 2 GB のメモリーと仮想マシン用の追加の RAM。
- ホスト用に 6 GB のディスク領域と、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には、最低 6GB のディスク領域が必要です。各ゲストに追加するストレージ領域は、ゲストのワークロードによって異なります。

  スワップ領域

  Linux のスワップ領域は、物理メモリー (RAM) の容量がいっぱいになったときに使用されます。システムに多くのメモリーリソースが必要で、RAM が不足すると、メモリーの非アクティブなページがスワップ領域に移動します。スワップ領域は、RAM が少ないマシンで役に立ちますが、RAM の代わりに使用しないようにしてください。スワップ領域はハードドライブにあり、そのアクセス速度は物理メモリーに比べると遅くなります。スワップパーティションのサイズは、ホストの物理 RAM から計算できます。Red Hat カスタマーポータルには、スワップパーティションのサイズを安全かつ効率的に判断するための記事 https://access.redhat.com/site/solutions/15244 が記載されています。

  - RAW イメージファイルを使用する場合、必要なディスクスペースは、イメージファイルに必要なスペース、ホストオペレーティングシステムに必要な 6 GB のスペース、およびゲスト用のスワップスペースの合計以上になります。

    式1.1 raw イメージを使用したゲスト仮想マシンに必要な領域の計算

    total for raw format = images + hostspace + swap

    qcow イメージの場合は、qcow イメージおよび qcow2 イメージが必要に応じて拡張できるため、ゲスト **(total for qcow format)** の予想される最大ストレージ要件も計算する必要があります。この拡張を可能にするには、最初にゲスト **(expected maximum guest storage)** のストレージ要件の上限に 1.01 を乗じ、これにホスト **(host)** に必要な領域と、必要なスワップ領域 **(swap)** を加算します。

    式1.2 qcow イメージを使用したゲスト仮想マシンに必要な領域の計算

    total for qcow format = (expected maximum guest storage * 1.01) + host + swap

ゲストの仮想マシン要件の概要は、*7章KVM でのオーバーコミット* を参照してください。

1.2. KVM ハイパーバイザーの要件

KVM ハイパーバイザーには、以下が必要です。

- x86 ベースのシステム用の Intel VT-x および Intel 64 の仮想化拡張機能を搭載した Intel プロセッサー。
- AMD-V および AMD64 仮想化拡張機能を搭載した AMD プロセッサー。

完全仮想化には、仮想化拡張機能 (Intel VT-x または AMD-V) が必要です。次のコマンドを入力して、システムにハードウェア仮想化拡張機能があるかどうか、およびそれらが有効になっているかどうかを確認します。

手順1.1 仮想化拡張機能の検証

1. **CPU 仮想化拡張機能が利用可能であることを確認します。**
   次のコマンドを実行して、CPU 仮想化拡張機能が利用可能であることを確認します。

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

2. **出力を分析します。**

   - 以下の例では、Intel VT-x 拡張機能を持つ Intel プロセッサーを示す**vmx** エントリーが含まれています。

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

   - 以下の例では、AMD-V 拡張機能を持つ AMD プロセッサーを示す**svm** エントリーが含まれています。

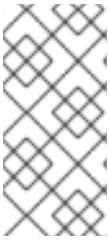
     ```
     flags   :  fpu tsc msr pae mce cx8 apic mtrr mca cmov pat pse36 clflush
     mmx fxsr sse sse2 ht syscall nx mmxext svm fxsr_opt lm 3dnowext 3dnow pni cx16
     lahf_lm cmp_legacy svm cr8legacy ts fid vid ttp tm stc
     ```

   grep -E 'svm|vmx' /proc/cpuinfo コマンドが出力を返す場合、プロセッサーにはハードウェア仮想化の拡張機能が含まれます。状況によっては、製造元が BIOS で仮想化拡張機能を無効にする場合があります。拡張機能が表示されない、または完全仮想化が機能しない場合は、BIOS 設定ユーティリティーで拡張機能を有効にする手順の 手順A.3「BIOS での仮想化拡張機能の有効化」 を参照してください。

3. **KVM カーネルモジュールが読み込まれていることを確認します。**
   追加の確認として、以下のコマンドを使用して、**kvm** モジュールがカーネルに読み込まれていることを確認します。

   ```
   # lsmod | grep kvm
   ```

   出力に **kvm_intel** または **kvm_amd** が含まれる場合は、**kvm** ハードウェア仮想化モジュールが読み込まれます。



注記

virsh ユーティリティー (libvirt-client パッケージにより提供) では、以下のコマンドを使用して、システムの仮想化機能の完全なリストを出力できます。

```
# virsh capabilities
```

1.3. KVM ゲスト仮想マシンの互換性

Red Hat Enterprise Linux 7 サーバーには、特定のサポート制限があります。

以下の URL では、Red Hat Enterprise Linux のプロセッサーとメモリーの制限を説明します。

- ホストシステムの場合: https://access.redhat.com/articles/rhel-limits
- KVM ハイパーバイザーの場合: https://access.redhat.com/articles/rhel-kvm-limits

以下の URL は、Red Hat Enterprise Linux KVM ホストで実行することが認定されているゲストオペレーティングシステムのリストです。

- https://access.redhat.com/articles/973163

注記

KVM ハイパーバイザーの制限およびサポート制限の詳細は、付録C *仮想化の制限*を参照してください。

1.4. サポートされているゲスト CPU モデル

すべてのハイパーバイザーには、ゲストにデフォルトで表示される CPU 機能に関する独自のポリシーがあります。 ハイパーバイザーによりゲストに提示される CPU 機能のセットは、ゲスト仮想マシンの設定で選択された CPU モデルによって異なります。

1.4.1. ゲスト CPU モデルのリスト表示

アーキテクチャータイプで対応している CPU モデルのリストを表示するには、**virsh cpu-models *architecture*** を実行します。以下に例を示します。

```
$ 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
```

対応している CPU モデルと機能のリストは、**/usr/share/libvirt/** にある **cpu_map.xml** ファイルに記載されています。

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

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

ホストモデルは、必要に応じて、指定した機能セットを使用するように設定できます。詳細は、「指定した CPU の機能セットの変更」 を参照してください。

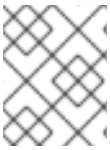
第2章 仮想化パッケージのインストール

仮想化を使用するには、コンピューターに Red Hat 仮想化パッケージがインストールされている必要があります。仮想化パッケージは、Red Hat Enterprise Linux のインストール時、またはインストール後に、**yum** コマンドと Subscription Manager アプリケーションを使用してインストールできます。

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

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

本セクションでは、Red Hat Enterprise Linux のインストール時に仮想化パッケージをインストールする方法を説明します。



注記

Red Hat Enterprise Linux のインストールの詳細は、Red Hat Enterprise Linux 7 インストールガイド を参照してください。

重要

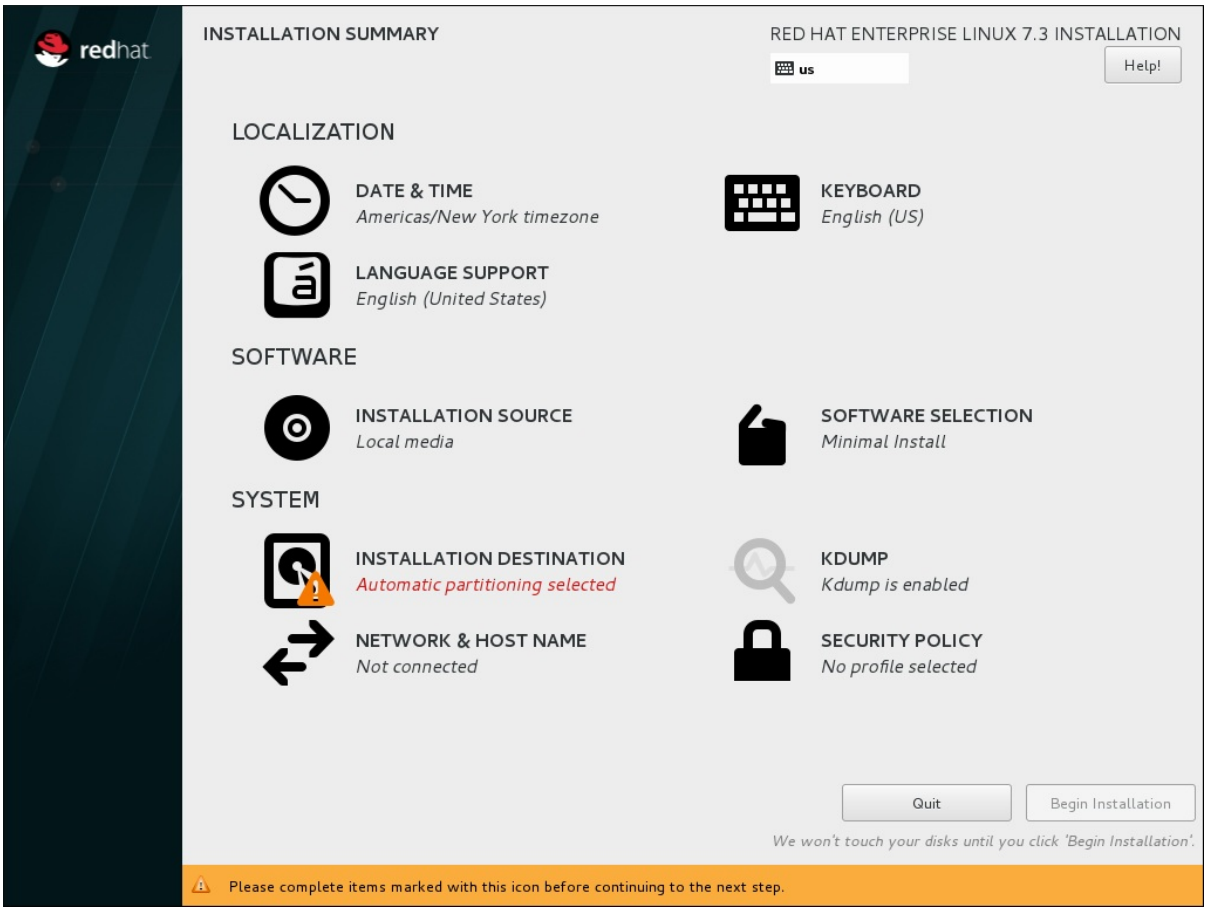
Anaconda インターフェイスでは、Red Hat Enterprise Linux Server のインストール時に Red Hat 仮想化パッケージをインストールするオプションのみが提供されます。

Red Hat Enterprise Linux Workstation をインストールする場合、Red Hat 仮想化パッケージはワークステーションのインストールが完了しないとインストールできません。「既存の Red Hat Enterprise Linux システムへの仮想化パッケージのインストール」を参照してください。

手順2.1 仮想化パッケージのインストール

1. **ソフトウェアの選択**
   Installation Summary 画面までのインストール手順に従います。

図2.1 Installation Summary 画面



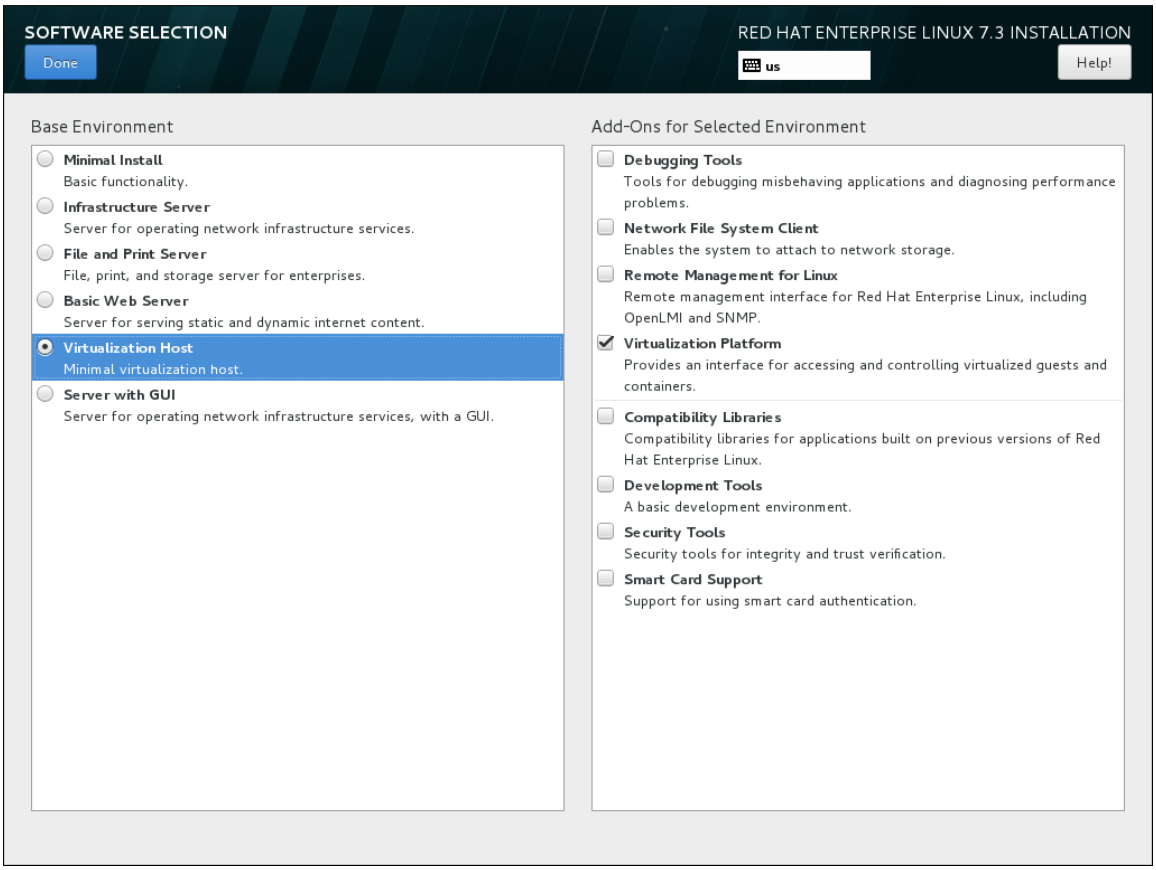
インストールの概要 画面で、**ソフトウェアの選択** を選択します。**ソフトウェアの選択** 画面が開きます。

2. **サーバータイプおよびパッケージグループの選択**
Red Hat Enterprise Linux 7 は、基本的な仮想化パッケージのみ、またはグラフィカルユーザーインターフェイスを介してゲストの管理を可能にするパッケージでインストールできます。次のいずれかを行います。

   - **最小限の仮想化ホストをインストールします**

     ベース環境ペインの **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** ペインの **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** を使用して実行、またはネットワーク経由でリモートで実行できる基本的な仮想化環境がインストールされます。

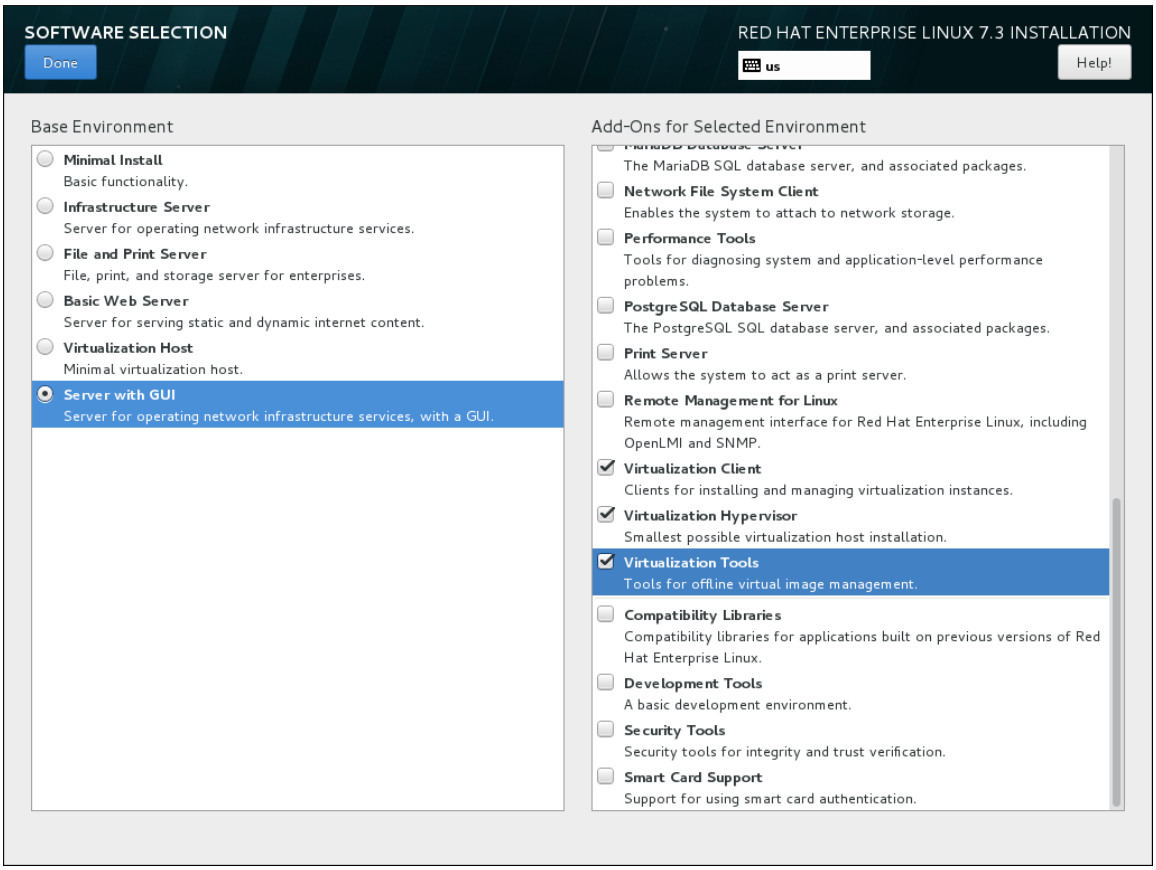
図2.2 ソフトウェアの選択の画面で選択した仮想化ホスト



- **グラフィカルユーザーインターフェイスを使用した仮想化ホストのインストール**

  ベース環境ペインの **サーバー (GUI 使用)** ラジオボタンを選択し、**選択した環境のアドオン**ペインの **仮想化クライアント**、**仮想化ハイパーバイザー**、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、仮想化環境に加えて、ゲスト仮想マシンのインストールおよび管理用のグラフィカルツールがインストールされます。

図2.3 ソフトウェア選択画面で GUI が選択されているサーバー



3. **インストールの終了**
   完了 をクリックして、インストールを続行します。

重要

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

2.1.1. キックスタートファイルを使用した 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して、仮想化パッケージで Red Hat Enterprise Linux をインストールする場合は、キックスタートファイルの **%packages** セクションに以下のパッケージグループを追加します。

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

キックスタートファイルを使用したインストールの詳細は、Red Hat Enterprise Linux 7 Installation Guide を参照してください。

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

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

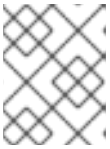
パッケージをインストールするには、マシンを登録し、Red Hat カスタマーポータルにサブスクライブする必要があります。Red Hat Subscription Manager を使用して登録する場合は、**subscription-manager register** コマンドを実行し、プロンプトに従います。または、**アプリケーション** → デスクトップの **システムツール** で Red Hat Subscription Manager アプリケーションを実行して登録します。

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 にアクセスして取得してください。Red Hat カスタマーポータルへのシステムの登録およびサブスクライブの詳細は、https://access.redhat.com/solutions/253273 を参照してください。

2.2.1. 仮想化パッケージの手動インストール

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

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



注記

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

- libvirt: このパッケージは、ハイパーバイザーおよびホストシステムと対話するためにサーバーおよびホスト側のライブラリーを提供し、ライブラリー呼び出しの処理、仮想マシンの管理およびハイパーバイザーの制御を行う **libvirtd** デーモンも提供します。

これらのパッケージをインストールするには、以下のコマンドを入力します。

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

いくつかの追加の仮想化管理パッケージも利用可能であり、仮想化を使用する場合に推奨されます。

- virt-install: このパッケージは、コマンドラインから仮想マシンを作成するための **virt-install** コマンドを提供します。
- libvirt-python: このパッケージには、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 で提供されるインターフェイスを使用できるようにするモジュールが含まれています。
- virt-manager: このパッケージは、**仮想マシンマネージャー** と呼ばれる **virt-manager** ツールを提供します。これは、仮想マシンを管理するためのグラフィカルツールです。管理 API として libvirt-client ライブラリーを使用します。
- libvirt-client: このパッケージは、libvirt サーバーにアクセスするためのクライアント側の API とライブラリーを提供します。libvirt-client パッケージには、コマンドラインまたは特別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および制御する **virsh** コマンドラインツールが含まれます。

以下のコマンドを使用すると、推奨される仮想化パッケージをすべてインストールできます。

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

2.2.2. 仮想化パッケージグループのインストール

仮想化パッケージは、パッケージグループからインストールすることもできます。**yum grouplist hidden** コマンドを実行すると、使用可能なグループのリストを表示できます。

利用可能なパッケージグループの完全なリストのうち、次の表では、仮想化パッケージグループとそれらが提供するものについて説明します。

表2.1 仮想化パッケージグループ

パッケージグループ	説明	必須パッケージ	オプションパッケージ
Virtualization Hypervisor	可能な限り最小の仮想化ホストのインストール	libvirt、qemu-kvm、qemu-img	qemu-kvm-tools
Virtualization Client	仮想化インスタンスのインストールおよび管理を行うクライアント	gnome-boxes、virt-install、virt-manager、virt-viewer、qemu-img	virt-top、libguestfs-tools、libguestfs-tools-c
Virtualization Platform	仮想マシンおよびコンテナーへのアクセスと制御を行うインターフェイスを提供します。	libvirt、libvirt-client、virt-who、qemu-img	fence-virtd-libvirt、fence-virtd-multicast、fence-virtd-serial、libvirt-cim、libvirt-java、libvirt-snmp、perl-Sys-Virt
Virtualization Tools	オフラインの仮想イメージ管理用ツール	libguestfs、qemu-img	libguestfs-java、libguestfs-tools、libguestfs-tools-c

パッケージグループをインストールするには、**yum group install *package_group*** コマンドを実行します。たとえば、**Virtualization Tools** パッケージグループのすべてのパッケージタイプをインストールするには、次のコマンドを実行します。

```
# yum group install "Virtualization Tools" --setopt=group_package_types=mandatory,default,optional
```

パッケージグループのインストールの詳細については、『Red Hat Enterprise Linux で yum を使用してパッケージのグループをインストールする方法』 を参照してください。ナレッジベース記事。

第3章 仮想マシンの作成

Red Hat Enterprise Linux 7 ホストシステムに virtualization packages をインストールしたら、**virt-manager** インターフェイスを使用して、仮想マシンを作成し、ゲストオペレーティングシステムをインストールできます。あるいは、パラメーターのリストまたはスクリプトを使用して、**virt-install** コマンドラインユーティリティーを使用できます。この章では、両方の方法について説明します。

3.1. ゲスト仮想マシンのデプロイメントに関する考慮事項

ゲスト仮想マシンを作成する場合は、さまざまな要因を考慮してください。仮想マシンのロールはデプロイメントの前に評価する必要がありますが、変数ファクター (負荷、クライアント量) に基づく定期的な監視と評価も行う必要があります。要因は以下のとおりです。

Performance

ゲスト仮想マシンは、目的のタスクに基づいてデプロイおよび設定する必要があります。一部のゲストシステム (データベースサーバーを実行しているゲストなど) では、パフォーマンスに関する特別な考慮事項が必要になる場合があります。ゲストは、そのロールと、予測されるシステム負荷に基づいて、より多くの割り当てられた CPU またはメモリーを必要とする場合があります。

入出力の要件と入出力の種類

ゲスト仮想マシンによっては、I/O 要件が特に高い場合や、I/O のタイプ (通常のディスクブロックサイズのアクセスやクライアント数など) に基づいて、さらに検討または予測が必要になる場合があります。

ストレージ

ゲスト仮想マシンによっては、ストレージまたは高速なディスクタイプへのより高い優先度のアクセスが必要な場合や、ストレージの領域への排他的アクセスが必要な場合があります。ゲストが使用するストレージの量も定期的に監視され、ストレージのデプロイメントおよびメンテナンスの際に考慮される必要があります。Red Hat Enterprise Linux 7 Virtualization Security Guide で説明されているすべての考慮事項を必ずお読みください。また、物理ストレージが仮想ストレージのオプションを制限する可能性があることを理解しておくことが重要です。

ネットワークとネットワークインフラストラクチャー

使用環境によっては、ゲスト仮想マシンの中には、他のゲストよりも速いネットワークリンクが必要なものもあります。帯域幅または遅延は、特に要件や負荷の変更として、ゲストのデプロイメントおよびメンテナンスを行う際の要因となることがよくあります。

要件の要求

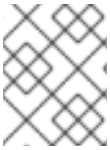
virtio ドライブがディスク全体でバックアップされており、ディスクデバイスパラメーターが ドメイン XML ファイル で **lun** に設定されている場合は、virtio ドライブのゲスト仮想マシンにのみ SCSI 要求を発行できます。以下に例を示します。

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

3.2. VIRT-INSTALL を使用したゲストの作成

virt-install コマンドを使用して仮想マシンを作成し、それらの仮想マシンにオペレーティングシステム

をコマンドラインからインストールすることができます。**virt-install** は、インタラクティブに、またはスクリプトの一部として使用して、仮想マシンの作成を自動化できます。対話式のグラフィカルインストールを使用している場合は、virt-viewer をインストールしてから **virt-install** を実行してください。また、キックスタートファイルを使用した **virt-install** を使用して、仮想マシンのオペレーティングシステムの自動インストールを起動できます。



注記

virt-install コマンドの中には、正常に完了するために root 特権が必要なものがあります。

virt-install ユーティリティーは、多くのコマンドラインオプションを使用します。ただし、ほとんどの**virt-install**オプションは必要ありません。

仮想ゲストマシンのインストールに必要な主なオプションは以下のとおりです。

--name

仮想マシンの名前。

--memory

ゲストに割り当てるメモリーの量 (MiB)。

ゲストストレージ

次のいずれかのゲストストレージオプションを使用します。

- **--disk**

  仮想マシンのストレージ設定の詳細**--disk none** オプションを使用すると、仮想マシンが作成されますが、ディスク領域はありません。

- **--filesystem**

  仮想マシンゲストのファイルシステムのパス。

インストール方法

次のいずれかのインストール方法を使用します。

- **--location**

  インストールメディアの場所。

- **--cdrom**

  仮想 CD-ROM デバイスとして使用されるファイルまたはデバイス。これは、ISO イメージへのパス、または最小ブート ISO イメージをフェッチまたはアクセスするための URL にすることができます。ただし、物理ホストの CD-ROM または DVD-ROM デバイスは使用できません。

- **--pxe**

  PXE 起動プロトコルを使用して、ゲストのインストールプロセスを開始する初期 ramdisk とカーネルを読み込みます。

- **--import**

OS のインストールプロセスをスキップし、既存のディスクイメージの周囲にゲストを構築します。起動に使用されるデバイスは、**disk** または **filesystem** オプションで指定された最初のデバイスです。

- **--boot**

  インストール後の仮想マシンの起動設定このオプションにより、起動デバイスの順序を指定し、オプションのカーネル引数を指定して BIOS 起動メニューを有効にし、カーネルと initrd を永続的に起動できます。

オプションのリストを表示するには、次のコマンドを実行します。

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

オプションの属性のリストを表示するには、次のコマンドを実行します。

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

また、**virt-install** の man ページでは、各コマンドオプション、重要な変数、および例が説明されています。

virt-install を実行する前に、**qemu-img** を使用してストレージオプションを設定する必要がある場合もあります。**qemu-img** の使用方法は、14章*qemu-img の使用* を参照してください。

3.2.1. ISO イメージからの仮想マシンのインストール

以下の例では、ISO イメージから仮想マシンをインストールします。

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

--cdrom /path/to/rhel7.iso オプションは、指定した場所に CD または DVD イメージから仮想マシンをインストールすることを指定します。

3.2.2. 仮想マシンイメージのインポート

以下の例では、仮想ディスクイメージから仮想マシンをインポートします。

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

--import オプションは、**--disk /path/to/imported/disk.qcow** オプションで指定した仮想ディスクイメージから仮想マシンをインポートすることを指定します。

3.2.3. ネットワークからの仮想マシンのインストール

以下の例では、ネットワークの場所から仮想マシンをインストールします。

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

--location http://example.com/path/to/os オプションは、インストールツリーが指定したネットワークロケーションにあることを指定します。

3.2.4. PXE を使用した仮想マシンのインストール

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

以下の例では、PXE を使用して仮想マシンをインストールします。

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

3.2.5. キックスタートを使用した仮想マシンのインストール

以下の例では、キックスタートファイルを使用して仮想マシンをインストールします。

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

initrd-inject オプションおよび **extra-args** オプションでは、キックスターターファイルを使用して仮想マシンをインストールすることを指定します。

3.2.6. ゲストの作成時のゲスト仮想マシンネットワークの設定

ゲスト仮想マシンを作成する場合は、仮想マシンのネットワークを指定および設定できます。このセクションでは、ゲスト仮想マシンの各メインネットワークタイプのオプションを説明します。

NAT を使用したデフォルトネットワーク

デフォルトネットワークは、**libvirtd** のネットワークアドレス変換 (NAT) 仮想ネットワークスイッチを使用します。NAT の詳細は「libvirt を使用した NAT (ネットワークアドレス変換)」 を参照してください。

NAT を使用したデフォルトネットワークでゲスト仮想マシンを作成する前に、libvirt-daemon-config-network パッケージがインストールされていることを確認してください。

ゲスト仮想マシンに NAT ネットワークを設定するには、**virt-install** に次のオプションを使用します。

```
--network default
```

注記

network オプションを指定しない場合は、ゲスト仮想マシンが NAT を使用したデフォルトネットワークで設定されます。

DHCP によるブリッジネットワーク
ブリッジネットワーク用に設定すると、ゲストは外部の DHCP サーバーを使用します。ホストに静的ネットワーク設定があり、ゲストがローカルエリアネットワーク (LAN) との完全な着信接続および発信接続を必要とする場合は、このオプションを使用してください。ゲスト仮想マシンでライブマイグレーションを実行する場合に使用します。ゲスト仮想マシンに DHCP を使用したブリッジネットワークを設定するには、次のオプションを使用します。

```
--network br0
```

注記

virt-install を実行する前に、ブリッジを個別に作成する必要があります。ネットワークブリッジの作成方法は、「Red Hat Enterprise Linux 7 ホストでのブリッジネットワークの設定」 を参照してください。

静的 IP アドレスを持つブリッジネットワーク
ブリッジネットワークは、静的 IP アドレスを使用するようにゲストを設定するためにも使用できます。ゲスト仮想マシンに静的 IP アドレスを使用してブリッジネットワークを設定するには、次のオプションを使用します。

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

ネットワークブートオプションの詳細は、Red Hat Enterprise Linux 7 Installation Guide を参照してください。

ネットワークなし
ネットワークインターフェイスを使用せずにゲスト仮想マシンを設定する場合は、次のオプションを使用します。

```
--network=none
```

3.3. VIRT-MANAGER を使用したゲストの作成

Virtual Machine Manager (**virt-manager** とも呼ばれる) は、ゲスト仮想マシンを作成および管理するグラフィカルツールです。

本セクションでは、**virt-manager** を使用して、Red Hat Enterprise Linux 7 ホストに 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。

この手順では、KVM ハイパーバイザーおよびその他の必要なパッケージがインストールされ、ホストが仮想化用に設定されていることを前提としています。仮想化パッケージのインストール方法は、2章 *仮想化パッケージのインストール* を参照してください。

3.3.1. virt-manager インストールの概要

New VM ウィザードでは、仮想マシンの作成プロセスが 5 つの手順に分けられます。

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

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

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

この手順では、ローカルに保存されたインストール DVD または DVD イメージを使用して、Red Hat Enterprise Linux 7 ゲスト仮想マシンを作成する方法を説明します。Red Hat Enterprise Linux 7 の DVD イメージは、Red Hat カスタマーポータル から入手できます。

注記

SecureBoot を有効にして仮想マシンをインストールする場合は、virt-manager を使用した SecureBoot Red Hat Enterprise Linux 7 ゲストの作成 を参照してください。

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

1. **オプション: 準備**
   仮想マシンのストレージ環境を準備します。ストレージの準備の詳細は、13章 *仮想マシン用のストレージの管理* を参照してください。

   重要

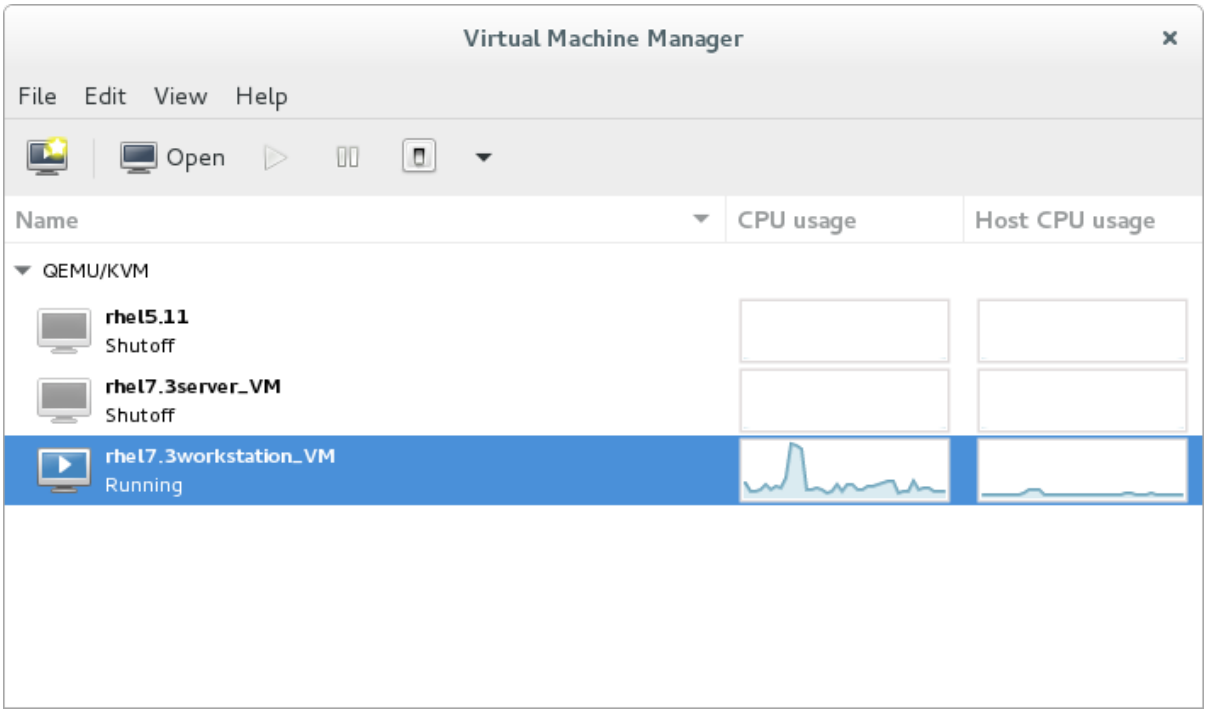
   ゲスト仮想マシンの保存には、さまざまなストレージタイプを使用できます。ただし、仮想マシンで移行機能を使用できるようにするには、ネットワークストレージに仮想マシンを作成する必要があります。

   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。ただし、Red Hat Enterprise Linux 7 のインストールおよびこのガイドの手順には、Red Hat は、少なくとも 5GB のストレージ領域を使用することを推奨します。

2. **virt-manager を開き、ウィザードを起動します。**

root で **virt-manager** コマンドを実行するか、**Applications → System Tools → Virtual Machine Manager** を開き、virt-manager を開きます。

図3.1 Virtual Machine Manager ウィンドウ



必要に応じて、ハイパーバイザーを選択し、**Connect** ボタンをクリックして、リモートハイパーバイザーを開きます。

 をクリックし、新しい仮想化ゲストウィザードを開始します。

New VM ウィンドウが開きます。

3. **インストールタイプを指定します。**
インストールタイプを選択します。

ローカルインストールメディア (ISO イメージまたは CDROM)

この方法では、インストールディスクのイメージ (**.iso** など) を使用します。ただし、ホスト CD-ROM または DVD-ROM デバイスを使用することは、できません。

ネットワークインストール (HTTP、FTP、または NFS)

この方法では、ミラーリングした Red Hat Enterprise Linux または Fedora インストールツリーを使用し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、FTP、または NFS のいずれかを介してアクセスできる必要があります。

Network Install を選択した場合は、インストール URL と、カーネルオプション (必要な場合) を指定します。

ネットワーク起動 (PXE)

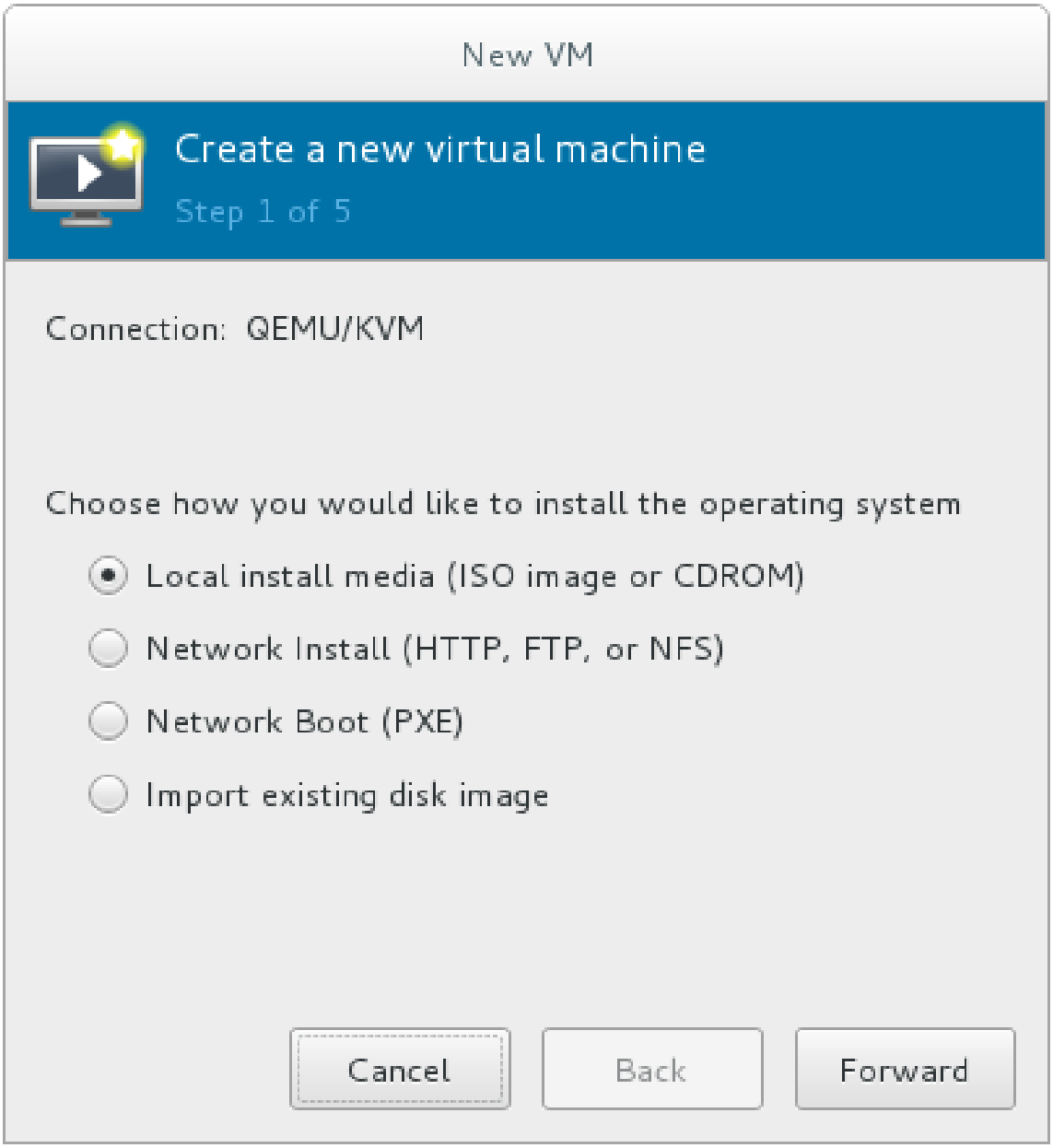
この方法では、PXE (Preboot eXecution Environment) サーバーを使用して、ゲスト仮想マシンをインストールします。PXE サーバーの設定については、『Red Hat Enterprise Linux 7 Installation Guide』 で説明されています。ネットワークブートを使用してインストールするには、ゲストに、ルーティング可能な IP アドレスまたは共有ネットワークデバイスが必要です。

Network Boot を選択した場合は、手順 5 に進みます。すべての手順が完了すると、DHCP 要求が送信され、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します。

既存のディスクイメージのインポート

この方法を使用すると、新しいゲスト仮想マシンを作成し、(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) ディスクイメージを読み込むことができます。

図3.2 仮想マシンのインストール方法

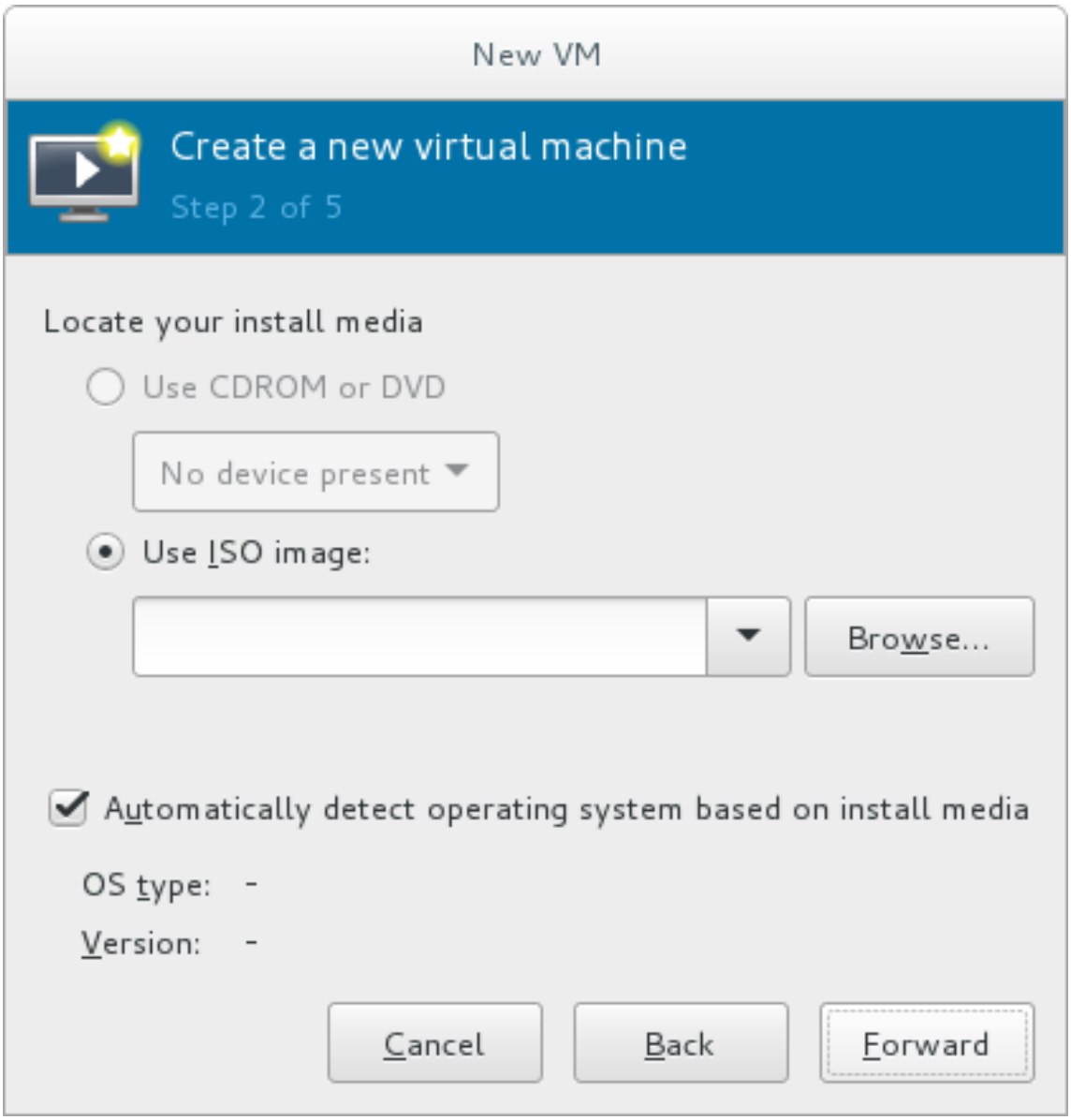


Forward をクリックして続けます。

4. **インストールソースを選択します。**

   a. **Local install media (ISO image or CDROM)** を選択した場合は、目的のローカルインストールメディアを指定します。

図3.3 ローカル ISO イメージのインストール



警告

このオプションは、現在 GUI にありますが、ホストの物理 CD-ROM または DVD デバイスからインストールすることはできません。そのため、**Use CDROM or DVD** オプションを選択すると、仮想マシンのインストールに失敗します。詳しくは、Red Hat ナレッジベース を参照してください。

ISO イメージからインストールする場合は、**Use ISO image** を選択し、**Browse...** ボタンをクリックして、**Locate media volume** 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、**Choose Volume** をクリックします。

Locate media volume ウィンドウにイメージが表示されない場合は、**Browse Local** ボタンをクリックして、インストールイメージがあるホストマシン、またはインストールディ

スクが含まれる DVD ドライブを参照します。インストールディスクが含まれるインストールイメージまたは DVD ドライブを選択し、**Open** をクリックします。使用するボリュームが選択され、**Create a new virtual machine** ウィザードに戻ります。

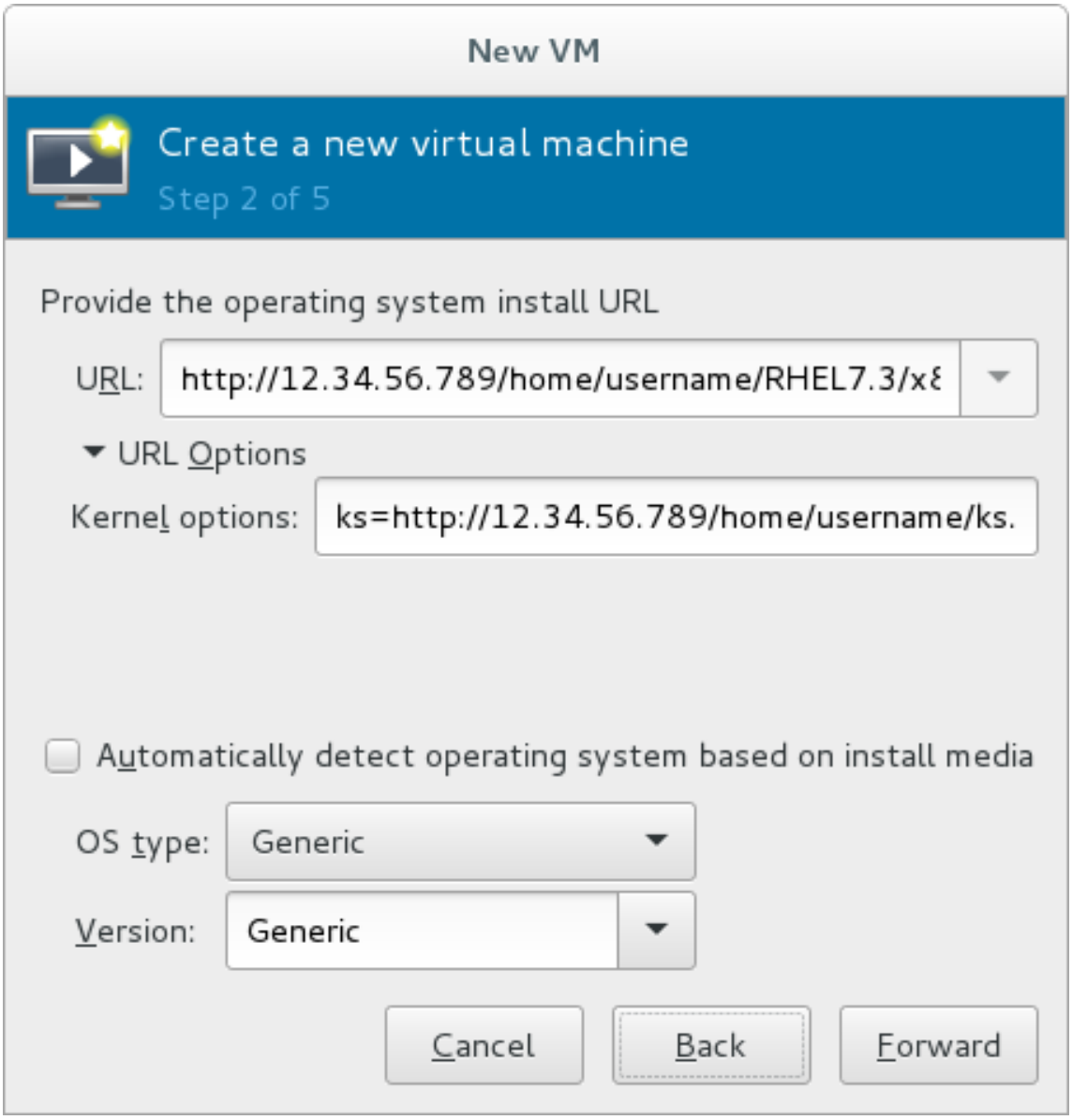
重要

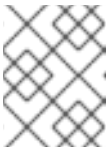
ISO イメージファイルおよびゲストストレージイメージの場合、推奨される使用場所は **/var/lib/libvirt/images/** です。それ以外の場所では、SELinux による追加の設定が必要になる場合があります。SELinux の設定方法の詳細は、Red Hat Enterprise Linux Virtualization Security Guide または Red Hat Enterprise Linux SELinux User's and Administrator's Guide を参照してください。

b. **Network Install** を選択した場合は、インストールソースの URL と、必要なカーネルオプション (存在する場合) を入力します。URL は、インストールツリーの root ディレクトリーを指している必要があります。このディレクトリーは、HTTP、FTP、または NFS のいずれかを介してアクセスできなければなりません。

キックスタートインストールを実行するには、カーネルオプションで **ks=** から始まるキックスタートファイルの URL を指定します。

図3.4 ネットワークキックスタートインストール





注記

カーネルオプションのリストは、 Red Hat Enterprise Linux 7 Installation Guide を参照してください。

次に、インストールの **OS type** と **Version** を設定します。仮想マシンに適したオペレーティングシステムの種類を選択していることを確認します。これは、手動で指定するか、**Automatically detect operating system based on install media** チェックボックスを選択して指定することができます。

Forward をクリックして続けます。

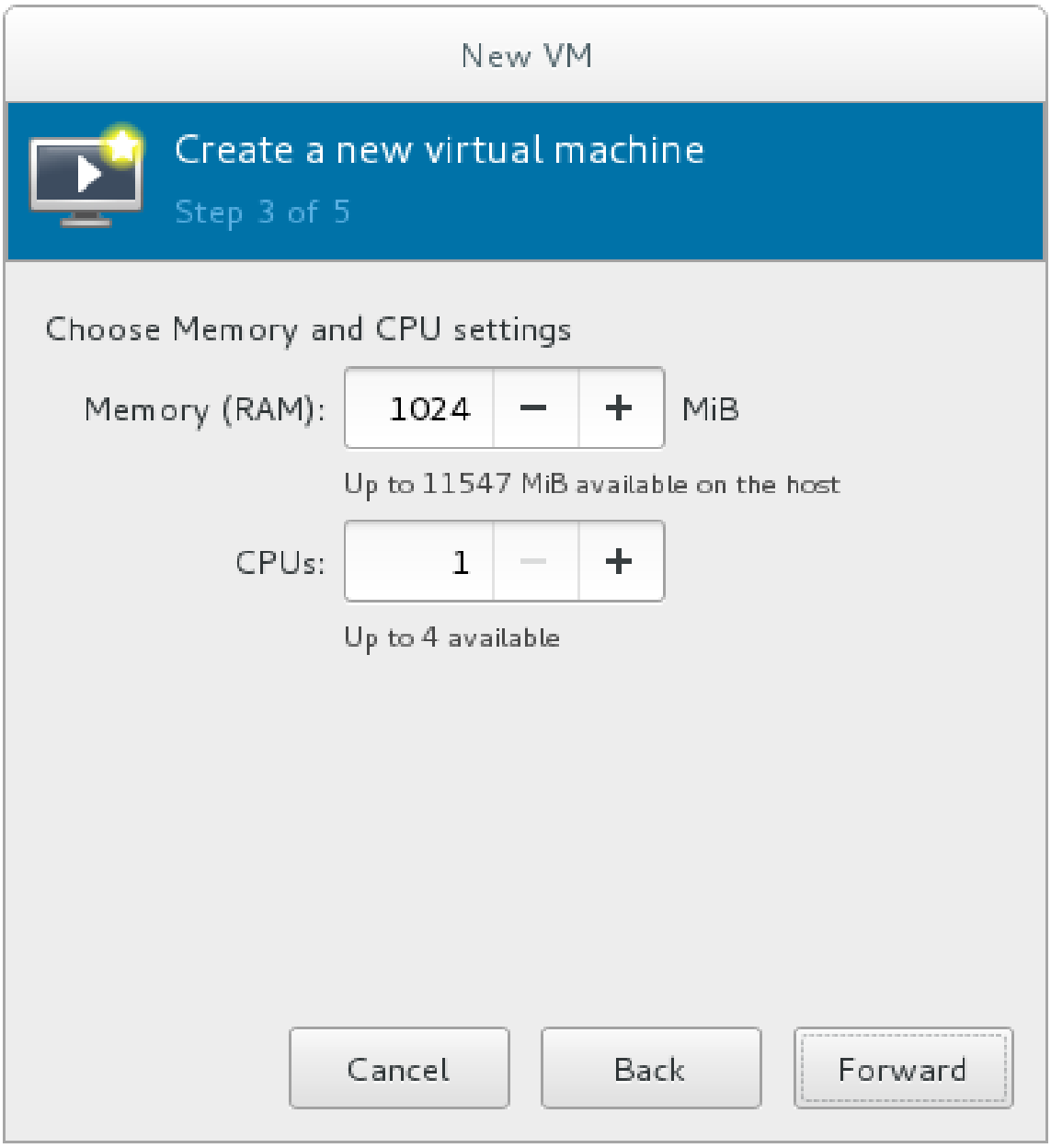
5. **メモリー (RAM) および仮想 CPU の設定**
仮想マシンに割り当てる CPU の数とメモリー量 (RAM) を指定します。ウィザードには、割り当てることができる CPU の数とメモリーの量が表示されます。この値は、ホストのパフォーマンスやゲストのパフォーマンスに影響を及ぼします。

   仮想マシンを効率的かつ効果的に実行するには、十分な物理メモリー (RAM) が必要です。Red Hat は、仮想マシンに 512MB 以上の RAM をサポートします。Red Hat では、各論理コアに 1024MB 以上の RAM を使用することを推奨しています。

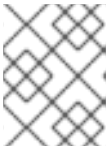
   仮想マシンに十分な仮想 CPU を割り当てます。仮想マシンでマルチスレッドアプリケーションを実行している場合は、ゲスト仮想マシンが効率的に実行するために必要な仮想 CPU の数を割り当てます。

   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 の数は、**Up to *X* available** フィールドに記載されています。

図3.5 メモリーおよび CPU の設定



メモリーと CPU の設定が完了したら、**Forward** をクリックして続行します。

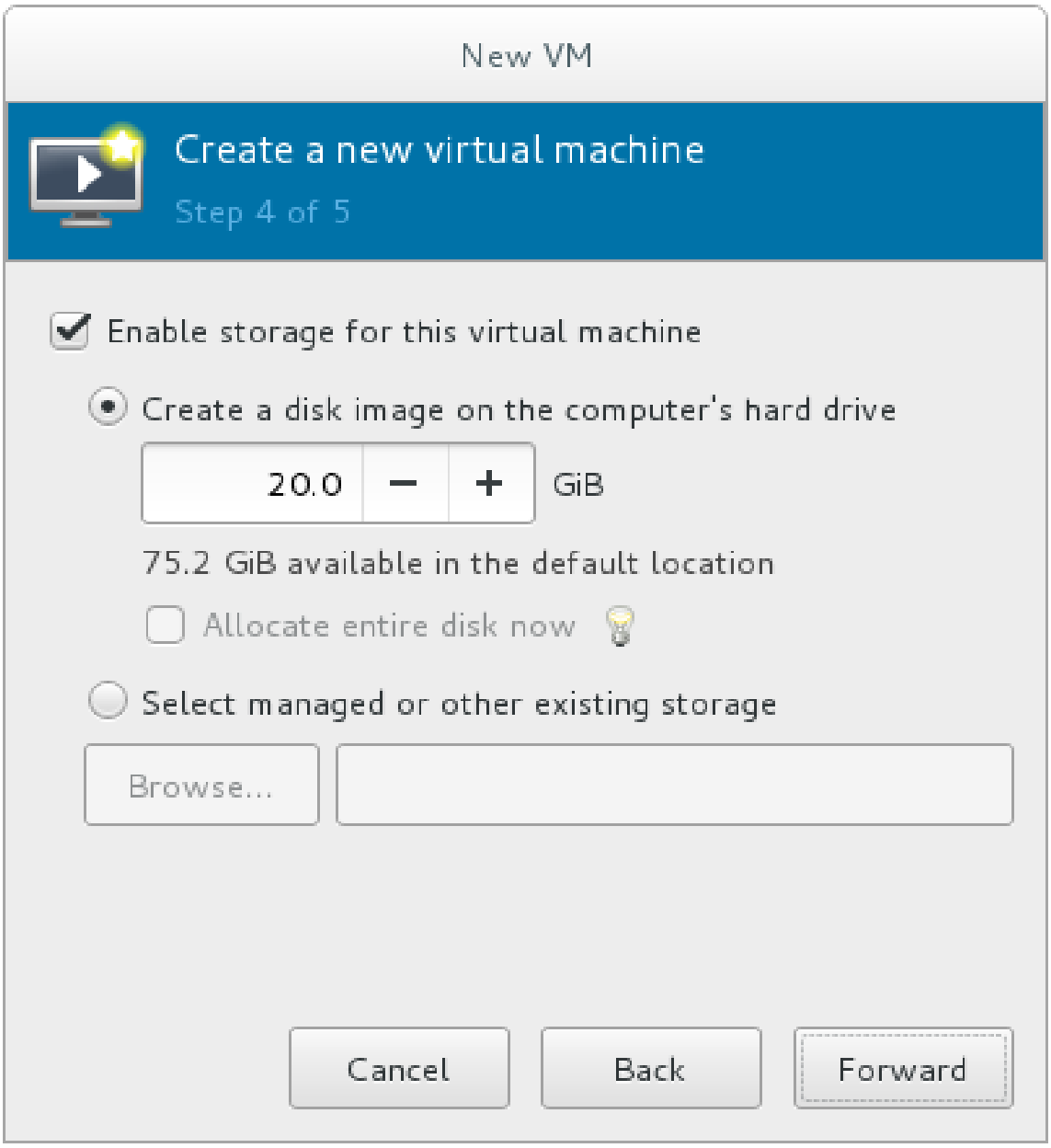


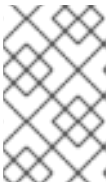
注記

メモリーおよび仮想 CPU はオーバーコミットできます。オーバーコミットの詳細は、*7章KVM でのオーバーコミット* を参照してください。

6. **ストレージを設定します。**
仮想マシンおよび必要なアプリケーションに十分な領域を有効にして割り当てます。デスクトップインストールの場合は 5GB 以上、最小インストールの場合は 1GB 以上を割り当てます。

図3.6 仮想ストレージの設定

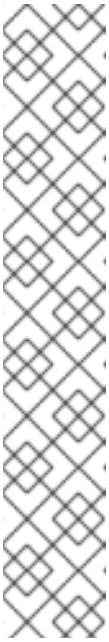




注記

ライブおよびオフラインの移行では、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。仮想マシンの共有ストレージの設定方法は、「共有ストレージの例: 単純な移行のための NFS」 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の場合**
Create a disk image on the computer's hard drive のラジオボタンを選択して、デフォルト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。**Allocate entire disk now** チェックボックスを選択すると、指定したサイズのディスクイメージが即座に作成されます。そうでない場合、ディスクイメージはいっぱいになると大きくなります。



注記

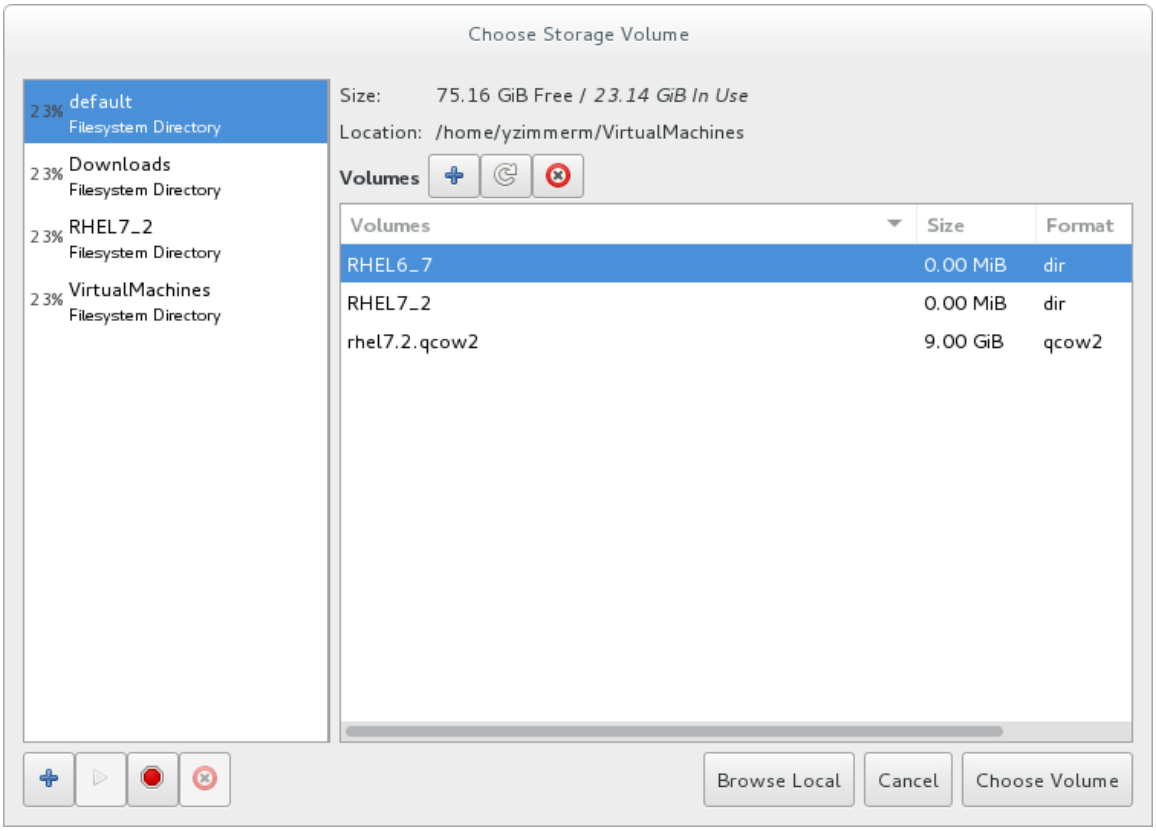
ストレージプールは仮想コンテナーですが、次の 2 つの要因で制限されます。つまり、qemu-kvm が許可する最大サイズと、ホストの物理マシンのディスクのサイズです。ストレージプールは、ホストの物理マシン上のディスクのサイズを超えてはなりません。最大サイズは、以下のとおりです。

- 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 = ~ 16TB (4KB のブロックサイズを使用)
- XFS = ~8 エクサバイト
- qcow2 およびホストのファイルシステムでは、非常に大きなイメージサイズを試行する際に、独自のメタデータとスケーラビリティーを評価/調整する必要があります。raw ディスクを使用すると、スケーラビリティーまたは最大サイズに影響を与えるレイヤーが減ります。

Forward を選択して、ローカル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、**Select managed or other existing storage** を選択し、**Browse** を選択して管理ストレージを設定します。

b. **ストレージプールを使用する場合**
Select managed or other existing storage を選択してストレージプールを使用する場合は、**Browse** をクリックして **Locate or create storage volume** ウィンドウを開きます。

図3.7 Choose Storage Volume ウィンドウ

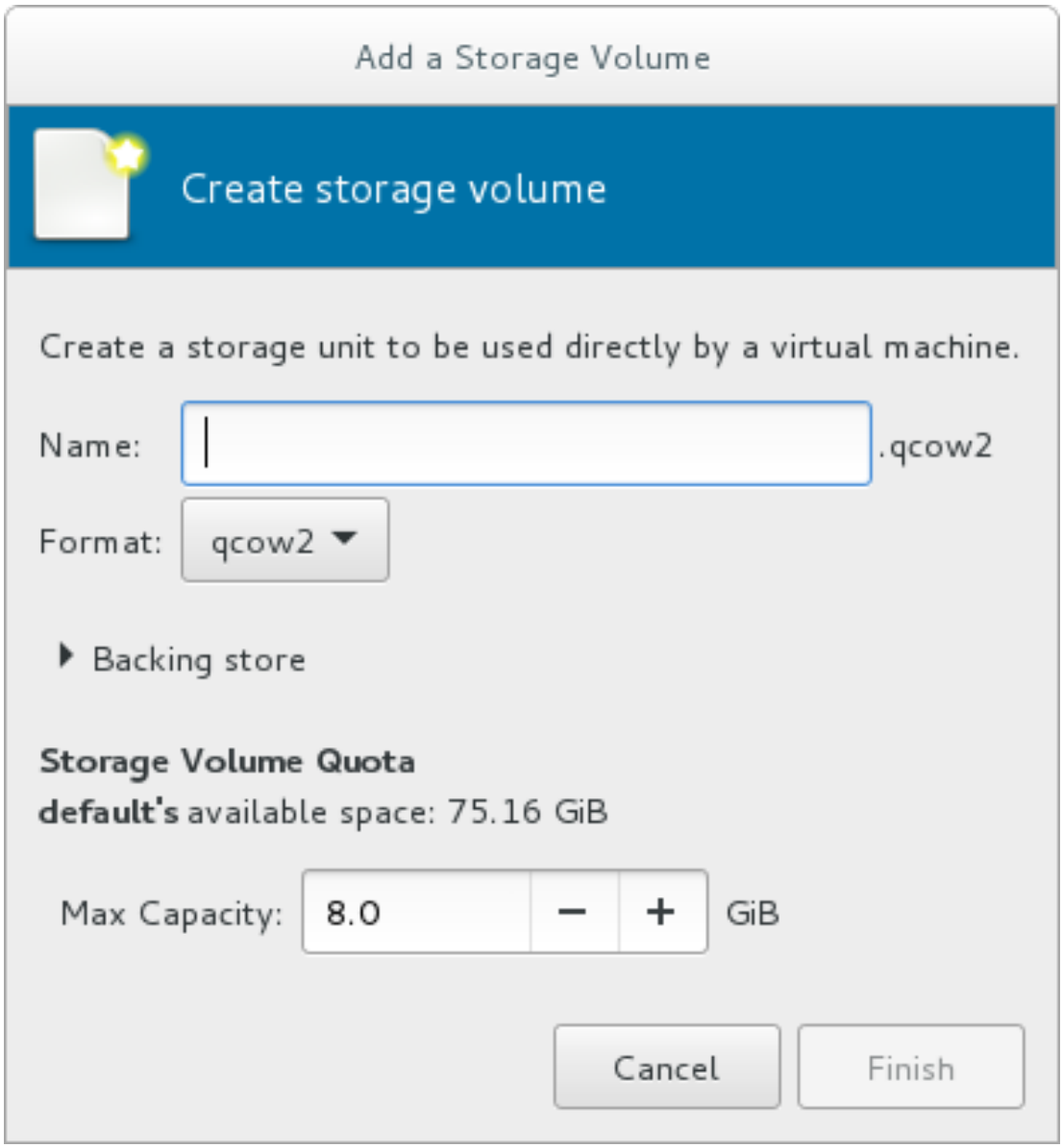


i. **Storage Pools** リストからストレージプールを選択します。

ii. オプション:  をクリックして、新しいストレージボリュームを作成します。**Add a Storage Volume** のスクリーンが表示されます。新しいストレージボリュームの名前を入力します。

Format ドロップダウンメニューから形式オプションを選択します。形式のオプションには、raw、qcow2、 および qedなどがあります。必要に応じてその他のフィールドを調整します。ここで使用される qcow2 バージョンはバージョン 3 であることに注意してください。qcow バージョンを変更するには、「target 要素の設定」 を参照してください。

図3.8 Add a Storage Volume ウィンドウ



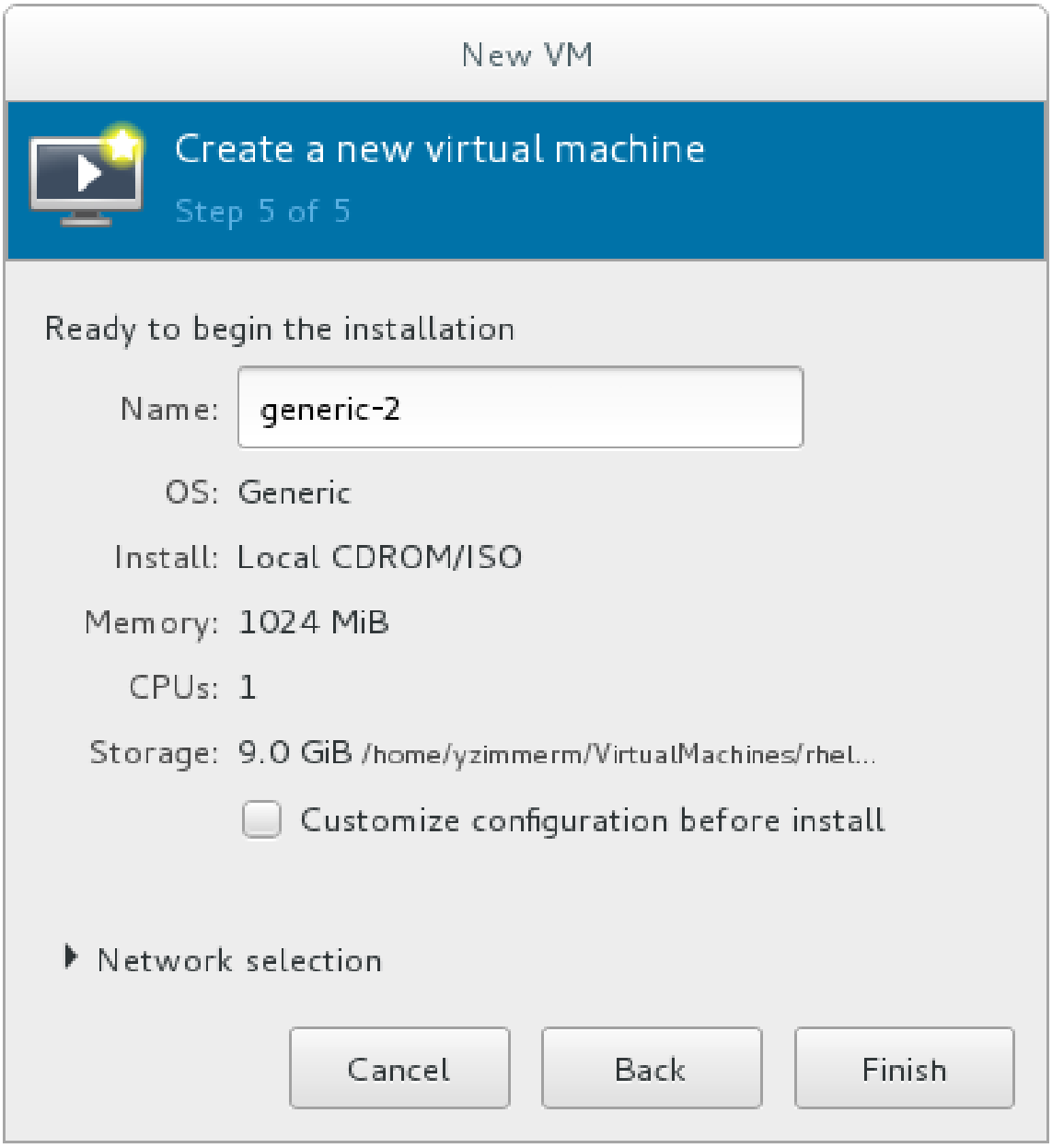
新しいボリュームを選択し、**Choose volume** をクリックします。次に、**Finish** をクリックして、**New VM** ウィザードに戻ります。**Forward** をクリックして続けます。

7. **名前を付けて最終設定を行います。**
仮想マシンに名前を付けます。仮想マシン名には、文字、数字、およびアンダースコア (_)、ピリオド (.)、およびハイフン (-) を含めることができます。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。

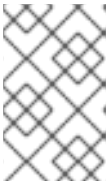
デフォルトで、仮想マシンは 'default' というネットワークの Network Address Translation (NAT) を使用して作成されます。ネットワークの選択を変更するには、**Network selection** をクリックしてホストデバイスとソースモードを選択します。

仮想マシンの設定を確認し、必要に応じて **Finish** をクリックします。指定したネットワーク設定、仮想化の種類、およびアーキテクチャーを持つ仮想マシンが作成されます。

図3.9 設定の確認



あるいは、仮想マシンのハードウェアをさらに設定する場合は、**Customize configuration before install** のチェックボックスにチェックを入れ、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、デバイスを追加します。これにより、別のウィザードが開き、仮想マシンのハードウェア設定の追加、削除、および設定ができるようになります。



注記

Red Hat Enterprise Linux 4 または Red Hat Enterprise Linux 5 のゲスト仮想マシンは、グラフィカルモードではインストールできません。このため、ビデオカードとして、QXL ではなく Cirrus を選択する必要があります。

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

警告

リモートメディアから、TCP/IP 接続を設定せずに Red Hat Enterprise Linux 7 ゲスト仮想マシンをインストールすると、インストールに失敗します。ただし、このような状況で Red Hat Enterprise Linux 5 または 6 のゲスト仮想マシンをインストールすると、インストーラーは "Configure TCP/IP" のインターフェイスを開きます。

この相違点の詳細は、the related knowledgebase article を参照してください。

Finish をクリックして、Red Hat Enterprise Linux インストールシーケンスに進みます。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す。

3.4. VIRT-INSTALL インストールオプションと VIRT-MANAGER インストールオプションの比較

この表では、仮想マシンのインストール時に、同等である **virt-install** と **virt-manager** のインストールオプションを比較するためのクイックリファレンスを提供します。

ほとんどの**virt-install**オプションは必要ありません。最低要件は、**--name**、**--memory**、guest storage (**--disk**、**--filesystem** または **--disk none**)、およびインストールメソッド (**--location**、**--cdrom**、**--pxe**、**--import**、または **boot**) です。これらのオプションは、引数でさらに指定されます。コマンドオプションと関連する引数の完全なリストを表示するには、次のコマンドを入力します。

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

virt-manager では、少なくとも名前、インストール方法、メモリー (RAM)、vCPU、ストレージが必要です。

表3.1 ゲストインストール用の virt-install と virt-manager の設定の比較

仮想マシンでの設定	virt-install オプション	virt-manager インストールウィザードのラベルと手順番号
Virtual machine name	--name、 -n	名前 (手順 5)
割り当てる RAM (MiB)	--ram, -r	メモリー (RAM) (手順 3)

仮想マシンでの設定	virt-install オプション	virt-manager インストールウィザードのラベルと手順番号
ストレージ - ストレージメディアを指定	--disk	この仮想マシンのストレージを有効にする → コンピューターのハードドライブにディスクイメージを作成するか、マネージドまたはその他の既存のストレージを選択します (手順 4)
ストレージ - ホストディレクトリーをゲストにエクスポートします。	--filesystem	この仮想マシンでストレージを有効にする → マネージドストレージまたはその他の既存ストレージを選択します (手順 4)
ストレージ - ゲストにローカルディスクストレージを設定しません。	--nodisks	この仮想マシンのストレージを有効にするチェックボックスの選択を解除します (手順 4)。
インストールメディアの場所 (ローカルインストール)	--file	ローカルインストールメディア → インストールメディアの場所 (手順 1-2)
配布ツリーを使用したインストール (ネットワークインストール)	--location	ネットワークインストール → URL (手順 1-2)
PXE を使用したゲストのインストール	--pxe	ネットワークブート (手順 1)
vCPU の数	--vcpus	CPU (手順 3)
ホストネットワーク	--network	高度なオプションドロップダウンメニュー (手順 5)
オペレーティングシステムのバリアント/バージョン	--os-variant	バージョン (手順 2)
グラフィカル表示の方法	--graphics、--nographics	*** virt-manager は、GUI インストールのみを提供します。**

第4章 仮想マシンのクローン作成

ゲストコピーの作成に使用されるゲスト仮想マシンインスタンスには、以下の 2 つのタイプがあります。

- **クローン** は、1 台の仮想マシンのインスタンスです。クローンを使用すると、同じ仮想マシンのネットワークを設定したり、別の宛先に配布したりできます。
- **テンプレート** は、クローン作成のソースとして使用するように設計された仮想マシンのインスタンスです。テンプレートから複数のクローンを作成し、各クローンにマイナーな変更を加えることができます。これは、この変更がシステムに与える影響を確認する際に役立ちます。

クローンとテンプレートはどちらも仮想マシンインスタンスです。これらの違いは、使用方法にあります。

作成されたクローンが正しく機能するには、通常、クローンを作成する前に、クローンを作成する仮想マシンに固有の情報と設定を削除する必要があります。削除する情報は、クローンの使用方法によって異なります。

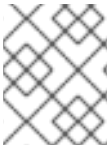
削除する情報および設定は、次のいずれかのレベルになります。

- **プラットフォームレベル**の情報および設定には、仮想化ソリューションが仮想マシンに割り当てたものが含まれます。例には、ネットワークインターフェイスカード (NIC) の数と、その MAC アドレスが含まれます。
- **ゲストオペレーティングシステムレベル**情報および設定には、仮想マシン内で設定されたものが含まれます。例には SSH 鍵が含まれます。
- **アプリケーションレベル** の情報および設定には、仮想マシンにインストールされているアプリケーションで設定したものが含まれます。例には、アクティベーションコードおよび登録情報が含まれます。

注記

情報およびアプローチは各アプリケーションに固有のものであるため、本章には、アプリケーションレベルの削除に関する情報は記載されていません。

その結果、仮想マシン内の情報および設定の一部を削除する必要がありますが、その他の情報および設定は、仮想化環境 (Virtual Machine Manager や VMware など) を使用して仮想マシンから削除する必要があります。



注記

ストレージボリュームのクローン作成の詳細は、「virsh を使用したストレージボリュームの作成」 を参照してください。

4.1. クローンを作成する仮想マシンの準備

仮想マシンのクローンを作成する前に、ディスクイメージで virt-sysprep ユーティリティーを実行するか、次の手順に従って仮想マシンを準備する必要があります。

手順4.1 クローンを作成する仮想マシンの準備

1. **仮想マシンのセットアップ**

a. クローンまたはテンプレートに使用する仮想マシンを構築します。

- クローンに必要なソフトウェアをインストールします。
- オペレーティングシステムに一意でない設定を設定します。
- 固有でないアプリケーション設定を設定します。

2. **ネットワーク設定を削除します。**

a. 以下のコマンドを使用して、永続的な udev ルールを削除します。

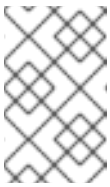
```
# rm -f /etc/udev/rules.d/70-persistent-net.rules
```

注記

udev ルールを削除しない場合は、最初の NIC の名前が eth0 ではなく eth1 になります。

b. **/etc/sysconfig/network-scripts/ifcfg-eth[x]** で以下の編集を行い、ifcfg スクリプトから一意のネットワークの詳細を削除します。

i. HWADDR 行および Static 行を削除します。



注記

HWADDR が新しいゲストの MAC アドレスと一致しない場合、ifcfg は無視されます。したがって、ファイルから HWADDR を削除することが重要です。

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

ii. HWADDR または一意の情報が含まれていない DHCP 設定が残っていることを確認します。

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

iii. ファイルに以下の行が含まれていることを確認します。

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

c. 以下のファイルが存在する場合は、そのファイルに同じ内容が含まれていることを確認してください。

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

注記

NetworkManager または特殊な設定を仮想マシンで使用した場合は、追加の固有情報が ifcfg スクリプトから削除されていることを確認してください。

3. **登録の詳細を削除します。**

a. 以下のいずれかを使用して、登録の詳細を削除します。

- Red Hat Network (RHN) の登録済みゲスト仮想マシンの場合は、次のコマンドを使用します。

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

- Red Hat Subscription Manager (RHSM) の登録済みゲスト仮想マシンの場合は、次のコマンドを使用します。
  - 元の仮想マシンを使用しない場合は、次のコマンドを使用します。

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

  - 元の仮想マシンを使用する場合は、次のコマンドのみを実行します。

```
# subscription-manager clean
```

元の RHSM プロファイルはポータルに残ります。クローン作成後に仮想マシンで RHSM 登録を再アクティベートするには、次の手順を行います。

1. カスタマー ID コードを取得します。

```
# subscription-manager identity
subscription-manager identity: 71rd64fx-6216-4409-bf3a-e4b7c7bd8ac9
```

2. 取得した ID コードを使用して仮想マシンを登録します。

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

4. **その他の固有の詳細の削除**

a. 次のコマンドを使用して、sshd の公開鍵と秘密鍵のペアを削除します。

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

注記

ssh キーを削除すると、このホストを信頼しない ssh クライアントの問題が回避されます。

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

5. **次回のシステムの起動時に設定ウィザードを実行するように仮想マシンを設定します。**

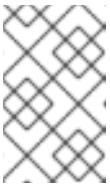
a. 以下のいずれかの方法で、仮想マシンを次回起動したときに、関連する設定ウィザードが実行されるように設定します。

- Red Hat Enterprise Linux 6 以前の場合は、以下のコマンドを使用して、root ファイルシステムに .unconfigured という名前の空のファイルを作成します。

```
# touch /.unconfigured
```

- Red Hat Enterprise Linux 7 の場合は、次のコマンドを実行して、最初の起動ウィザードおよび初期設定ウィザードを有効にします。

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



注記

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

4.2. 仮想マシンのクローン作成

クローンの作成を続行する前に、仮想マシンをシャットダウンします。**virt-clone** または **virt-manager** を使用して、仮想マシンのクローンを作成できます。

4.2.1. virt-clone を使用したゲストのクローン作成

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

virt-clone を正常に完了するには、root 権限が必要であることに注意してください。

virt-clone コマンドには、コマンドラインで渡すことができるオプションが多数用意されています。これには、一般的なオプション、ストレージ設定オプション、ネットワーク設定オプション、およびその他のオプションが含まれます。**--orginal** のみが必要です。オプションのリストを表示するには、次のコマンドを実行します。

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

また、**virt-clone** の man ページでは、各コマンドオプション、重要な変数、および例が記載されています。

以下の例は、デフォルト接続で、demo と呼ばれるゲスト仮想マシンのクローンを作成する方法を示しています。これにより、新しい名前とディスククローンパスが自動的に生成されます。

例4.1 virt-clone を使用したゲストのクローン作成

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

以下の例は、demo と呼ばれる QEMU ゲスト仮想マシンのクローンを、複数のディスクで作成する方法を示しています。

例4.2 virt-clone を使用したゲストのクローン作成

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

4.2.2. virt-manager を使用したゲストのクローン作成

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

手順4.2 virt-manager を使用した仮想マシンのクローンの作成

1. **virt-manager を開く**
   virt-manager を起動します。**Applications** メニューおよび **System Tools** サブメニューから **Virtual Machine Manager** アプリケーションを起動します。または、root で **virt-manager** を実行します。

   Virtual Machine Manager にあるゲスト仮想マシンのリストから、クローンを作成するゲスト仮想マシンを選択します。

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

図4.1 Clone Virtual Machine ウィンドウ

2. **クローンの設定**

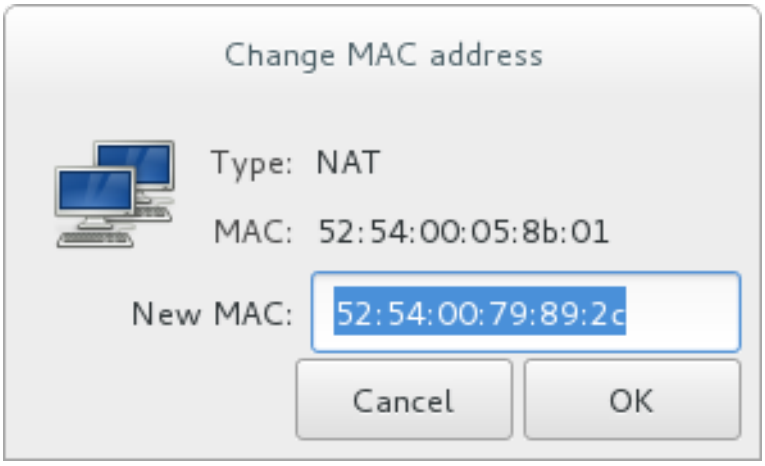
   - クローンの名前を変更する場合は、クローンの新しい名前を入力します。

- ネットワーク設定を変更する場合は、**Details** を選択します。

  クローンの新しい MAC アドレスを入力します。

  OK をクリックします。

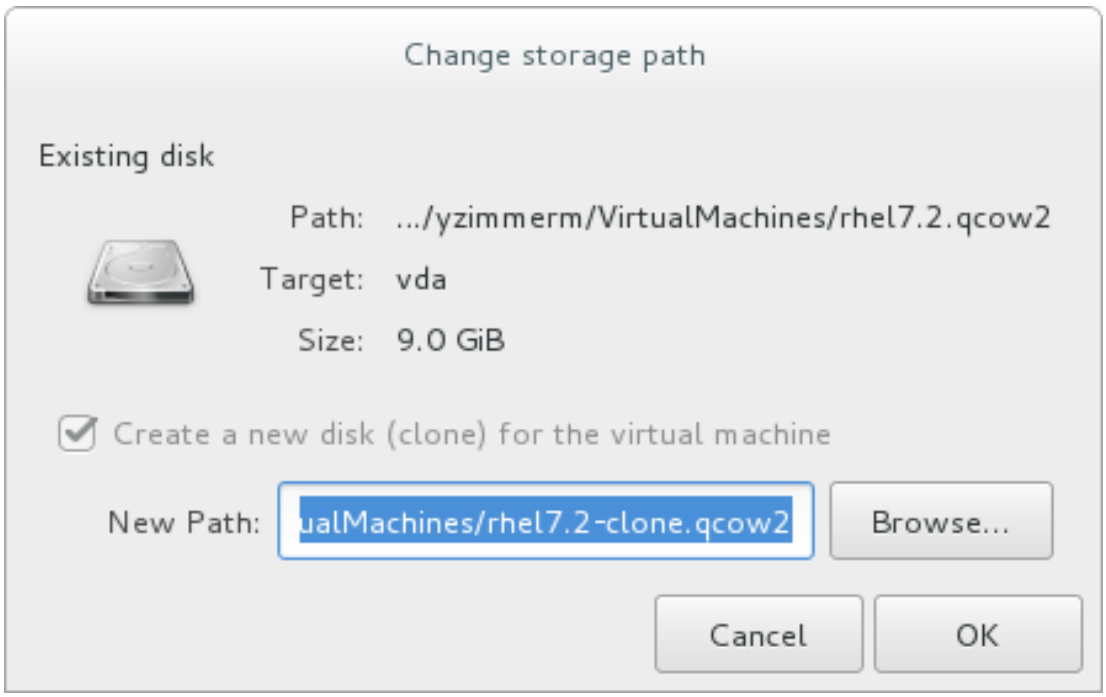
  図4.2 Change MAC Address ウィンドウ



- クローンを作成したゲスト仮想マシンのディスクごとに、次のいずれかのオプションを選択します。

  - **Clone this disk** - ディスクは、クローンとして作成されたゲスト仮想マシンのクローンとして作成されます。
  - **Share disk with *guest virtual machine name*** - ディスクは、クローンを作成されるゲスト仮想マシンとそのクローンで共有されます。
  - **Details** - ストレージパスの変更 画面を開きます。これにより、ディスクへの新しいパスの選択が可能になります。

    図4.3 Change storage path ウィンドウ



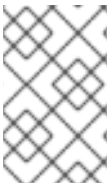
3. **ゲスト仮想マシンのクローンを作成する**

Clone をクリックします。

第5章 KVM 準仮想化 (VIRTIO) ドライバー

準仮想化ドライバーはゲストのパフォーマンスを向上し、ゲスト I/O レイテンシーを下げ、ベアメタルレベルまでスループットを増加させます。I/O 負荷の高いタスクおよびアプリケーションを実行する完全に仮想化されたゲストには、準仮想化ドライバーを使用することが推奨されます。

Virtio ドライバーは、KVM ホストで実行しているゲスト仮想マシンで利用可能な、KVM の準仮想化デバイスドライバーです。これらのドライバーは、**virtio** パッケージに同梱されています。virtio パッケージは、ブロック (ストレージ) デバイスおよびネットワークインターフェイスコントローラーに対応しています。



注記

PCI デバイスは、仮想システムのアーキテクチャーにより制限されます。割り当てられたデバイスを使用する場合の追加の制限は、16章 *ゲスト仮想マシンのデバイス設定* を参照してください。

5.1. 既存のストレージデバイスへの KVM VIRTIO ドライバーの使用

ゲストに接続されている既存のハードディスクデバイスを変更して、仮想 IDE ドライバーの代わりに **virtio** ドライバーを使用できます。このセクションの例では、libvirt 設定ファイルを編集します。これらの手順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、ゲストが完全にシャットダウンされて再起動されるまで、変更は適用されないことに注意してください。

手順5.1 既存のデバイスへの KVM virtio ドライバーの使用

1. この手順を続行する前に、適切なドライバー (**viostor**) がインストールされていることを確認してください。

2. root で **virsh edit *guestname*** コマンドを実行し、使用しているデバイスの XML 設定ファイルを編集します。たとえば、**virsh edit guest1** です。設定ファイルは、**/etc/libvirt/qemu/** ディレクトリーにあります。

3. 以下は、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一般的なエントリーです。

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

4. **bus=** エントリーを **virtio** に変更して、virtio デバイスを使用するエントリーを変更します。ディスクが以前 IDE だった場合は、**hda**、**hdb**、または **hdc** のようなターゲットがあることに注意してください。**bus=virtio** に変更する場合は、それに応じてターゲットを **vda**、**vdb**、または **vdc** に変更する必要があります。

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

5. **ディスク** タグ内の **アドレス** タグを削除します。これは、この手順が機能するために必ず行う必要があります。libvirt は、仮想マシンが次に起動したときに、**アドレス** タグを適切に再生成します。

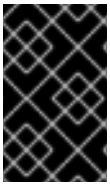
あるいは、**virt-manager**、**virsh attach-disk**、または **virsh attach-interface** は、virtio ドライバーを使用して新しいデバイスを追加できます。

Virtio の使用方法は、libvirt の Web サイトを参照してください。 http://www.linux-kvm.org/page/Virtio

5.2. 新しいストレージデバイス用に KVM VIRTIO ドライバーを使用する

この手順では、**virt-manager** で KVM virtio ドライバーを使用して新しいストレージデバイスを作成する方法を説明します。

あるいは、**virsh attach-disk** コマンドまたは **virsh attach-interface** コマンドを使用して、virtio ドライバーを使用してデバイスを接続できます。



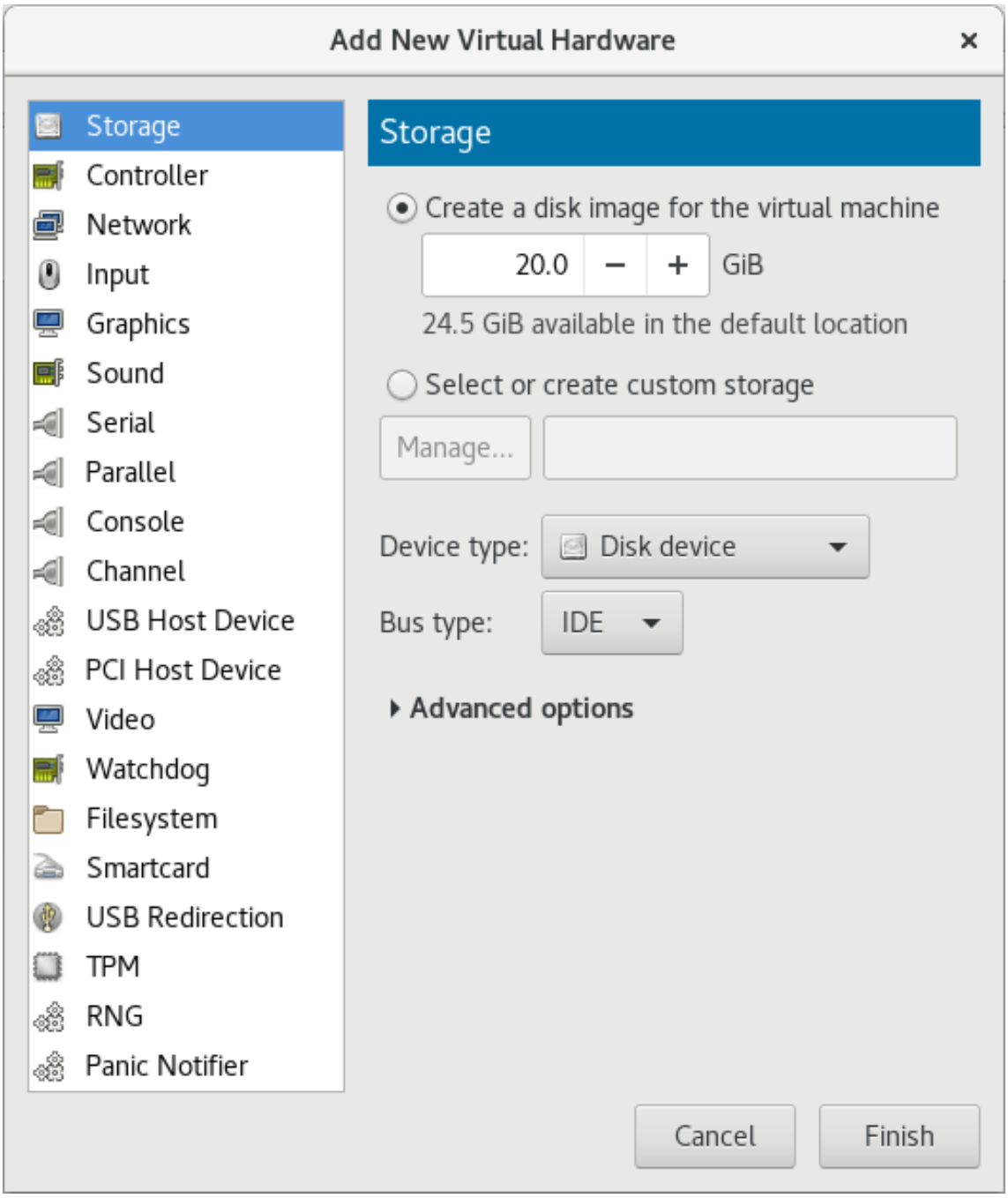
重要

新しいデバイスのインストールを続行する前に、ドライバーがゲストにインストールされていることを確認します。ドライバーが利用できない場合は、デバイスが認識されず、動作しません。

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

1. **virt-manager** でゲストの名前をダブルクリックして、ゲスト仮想マシンを開きます。
2.  をクリックして、**Show virtual hardware details** タブを開きます。
3. **Show virtual hardware details** タブで、**Add Hardware** ボタンをクリックします。
4. **ハードウェアタイプの選択**
   Hardware typeとして **Storage** を選択します。

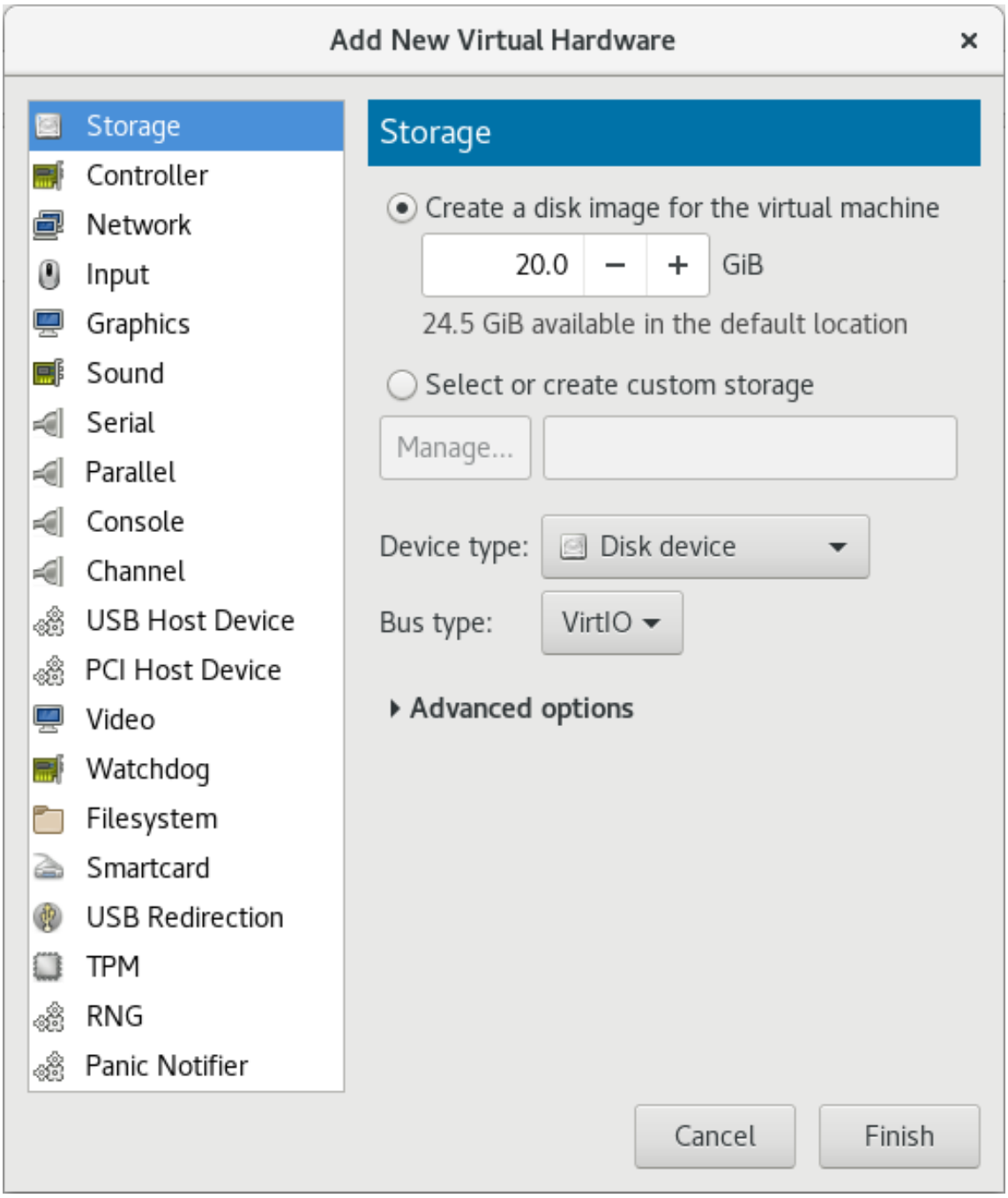
図5.1 Add new virtual hardware ウィザード



5. **ストレージデバイスとドライバーを選択します。**
新しい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。

Device type を **Disk device** に設定し、 **Bus type** を **VirtIO** に設定して、virtio ドライバーを使用します。

図5.2 Add new virtual hardware ウィザード

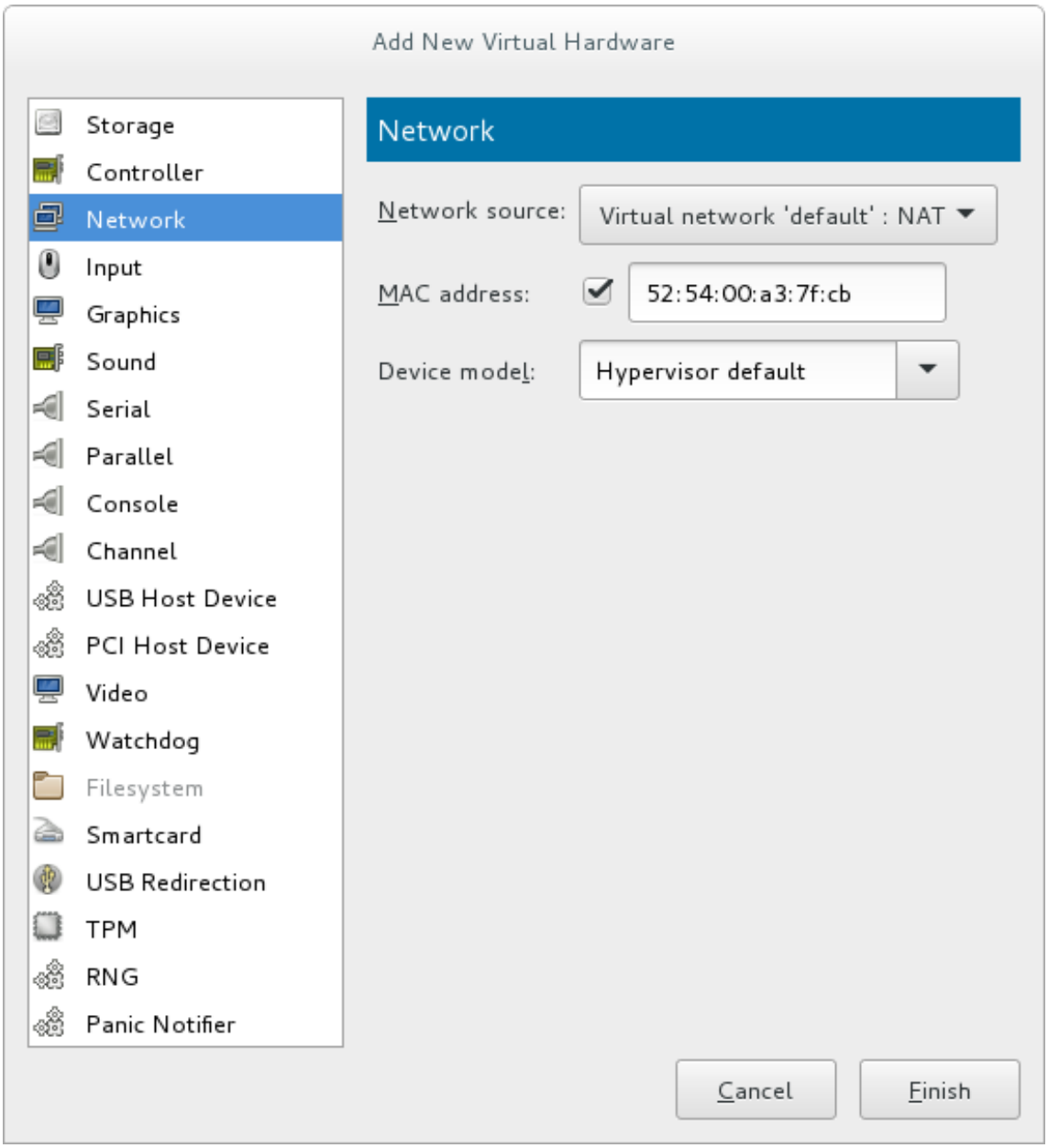


Finish をクリックして手順を完了します。

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

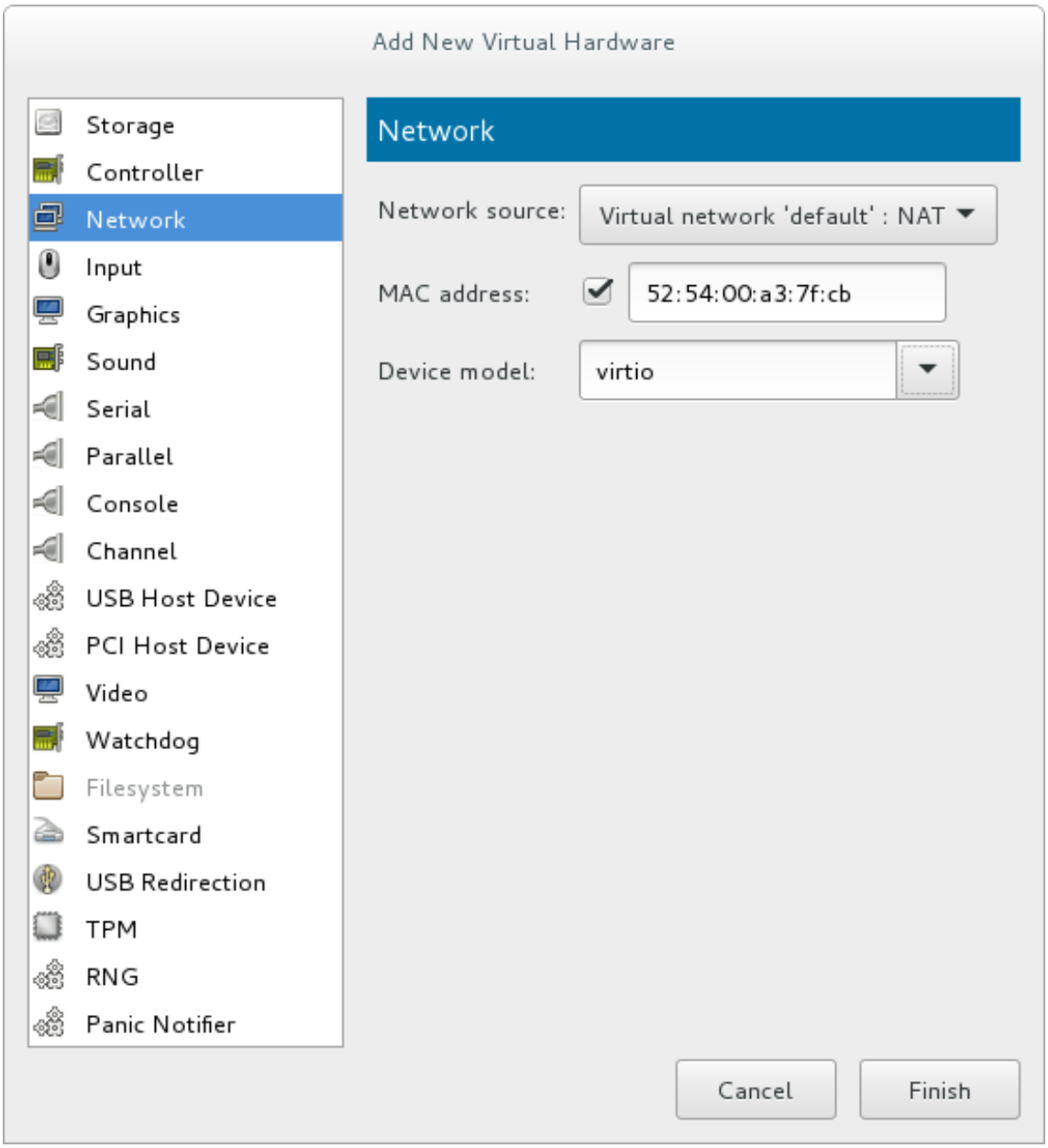
1. **virt-manager** でゲストの名前をダブルクリックして、ゲスト仮想マシンを開きます。
2.  をクリックして、**Show virtual hardware details** タブを開きます。
3. **Show virtual hardware details** タブで、**Add Hardware** ボタンをクリックします。
4. **ハードウェアタイプの選択**
   Hardware type として **Network** を選択します。

図5.3 Add new virtual hardware ウィザード



5. **ネットワークデバイスとドライバーを選択します。**
   Device model を **virtio** に設定して、virtio ドライバーを使用します。必要な **Host device** を選択します。

図5.4 Add new virtual hardware ウィザード



Finish をクリックして手順を完了します。

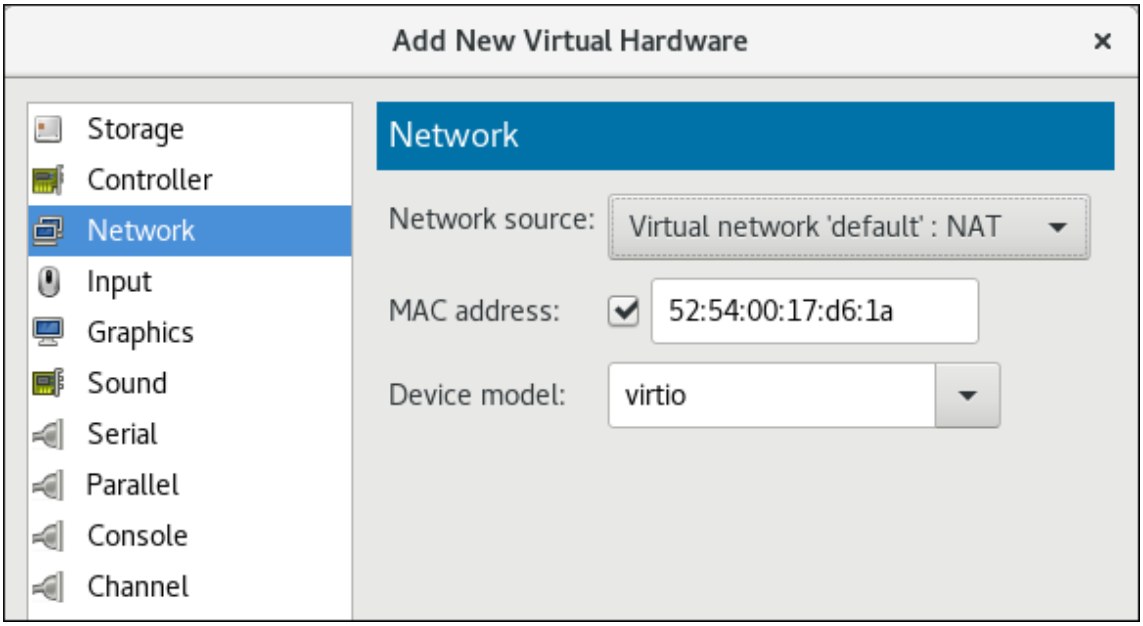
すべての新規デバイスが追加されたら、仮想マシンを再起動します。ゲストを再起動するまで、仮想マシンがデバイスを認識しない場合があります。

5.3. ネットワークインターフェイスデバイス用の KVM VIRTIO ドライバーの使用

ネットワークインターフェイスが KVM virtio ドライバーを使用する場合、KVM はネットワークハードウェアをエミュレートしないため、処理のオーバーヘッドがなくなり、ゲストパフォーマンスが向上します。Red Hat Enterprise Linux 7 では、virtio がデフォルトのネットワークインターフェイスタイプとして使用されます。ただし、システムで設定が異なる場合は、以下の手順を使用できます。

- ゲストに **virtio ネットワークデバイスを接続** するには、**model --virtio** オプションを指定して、**virsh attach-interface** コマンドを実行します。

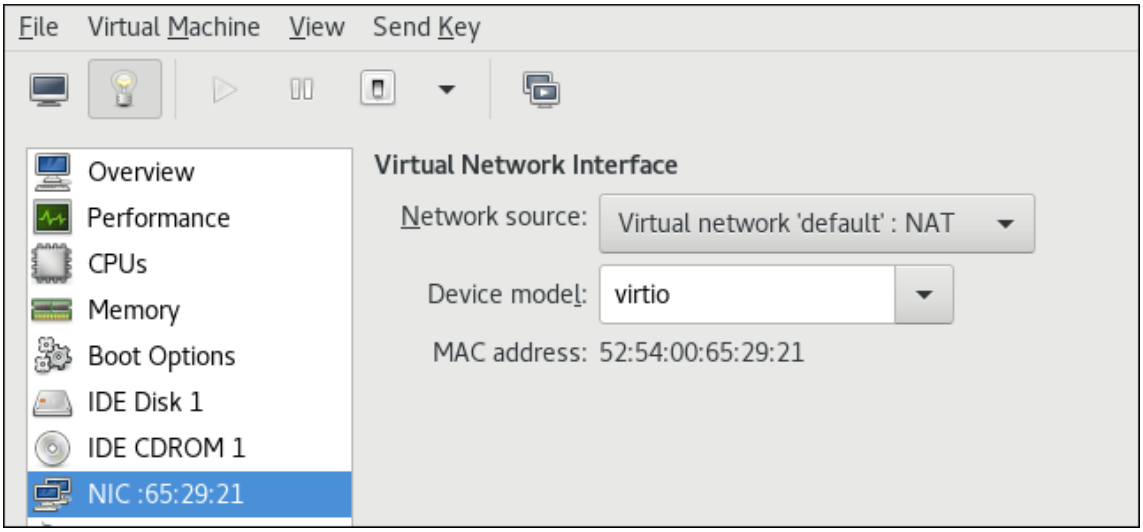
または、virt-manager インターフェイスで、ゲストの **Virtual hardware details** 画面に移動し、**Add Hardware** をクリックします。**Add New Virtual Hardware** 画面で **Network** を選択し、**Device model** を **virtio** に変更します。

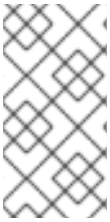


- **既存インターフェイスのタイプを virtio に変更する**するには、**virsh edit** コマンドを使用して、目的のゲストの XML 設定を変更し、**model type** 属性を **virtio** に変更します。以下に例を示します。

```
<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>
...
```

または、virt-manager インターフェイスで、ゲストの **Virtual hardware details** 画面に移動し、**NIC** アイテムを選択して、**Device model** を **virtio** に変更します。





注記

ゲスト内のネットワークインターフェイスの名前が再起動後も維持されない場合は、ゲストに提示されるすべてのインターフェイスが同じデバイスモデル (できれば**virtio-net**) であることを確認してください。詳細は、Red Hat ナレッジベース を参照してください。

第6章 NETWORK CONFIGURATION

本章では、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を紹介します。

Red Hat Enterprise Linux 7 では、仮想化に次のネットワークセットアップをサポートします。

- ネットワークアドレス変換 (*NAT*) を使用した仮想ネットワーク
- PCI デバイスの割り当てを使用して直接割り当てた物理デバイス
- 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

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

6.1. LIBVIRT を使用した NAT (ネットワークアドレス変換)

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、ネットワークアドレス変換 (NAT) 転送 (仮想ネットワークとも呼ばれます) を使用することです。

ホストの設定

すべての標準的な**libvirt** インストールでは、デフォルトの仮想ネットワークとして、仮想マシンに NAT ベースの接続が提供されます。**virsh net-list --all** コマンドで使用できることを確認します。

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

確認できない場合は、ゲストの XML 設定ファイル (/etc/libvirtd/qemu/myguest.xml など) で以下を使用できます。

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

デフォルトネットワークは、**/etc/libvirt/qemu/networks/default.xml** から定義されます。

デフォルトのネットワークを自動的に起動するように設定します。

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

デフォルトのネットワークを起動します。

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

libvirt のデフォルトネットワークが実行すると、分離されたブリッジデバイスが表示されます。このデバイスには、物理インターフェイスが**追加されていません**。新しいデバイスは、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。新規インターフェイ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
```

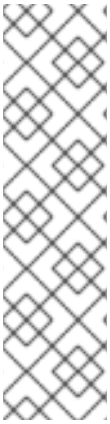
libvirt は、**INPUT**、**FORWARD**、**OUTPUT**、および **POSTROUTING** チェーンで **virbr0** デバイスにアタッチされたゲスト仮想マシンとの間のトラフィックを許可する **iptables** ルールを追加します。次に、**libvirt** は **ip_forward** パラメーターを有効にしようとします。他のアプリケーションでは**ip_forward** が無効になる場合があるため、最善の選択肢は、以下を **/etc/sysctl.conf** に追加することです。

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

ゲスト仮想マシンの設定

ホストの設定が完了したら、ゲスト仮想マシンは、その名前を基にして仮想ネットワークに接続できます。ゲストを 'default' の仮想ネットワークに接続するには、ゲストの XML 設定ファイル (**/etc/libvirtd/qemu/myguest.xml** など) で以下を使用できます。

```
<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。定義しない場合は、MAC アドレスが自動的に生成され、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 アドレスを手動で設定すると、環境全体で一貫性や簡単な参照を維持したり、競合の可能性が非常に低い場合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>
```

6.2. VHOST-NET の無効化

vhost-net モジュールは、virtio ネットワーク用のカーネルレベルのバックエンドです。これは、ユーザー空間 (QEMU プロセス) からカーネル (**vhost-net** ドライバー) に virtio パケット処理タスクを移動する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イスでのみ利用できます。vhost-net カーネルモジュールが読み込まれていると、すべての virtio インターフェイスでデフォルトで有効になりますが、vhost-net の使用時に特定のワークロードでパフォーマンスが低下する場合は、インターフェイ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そのホスト上のゲスト仮想マシンに送信される場合、ゲスト仮想マシンが受信データをホストマシンが送信する速度よりも遅い速度で処理すると、パフォーマンスが低下する可能性があります。この場合は、**vhost-net** を有効にすると、UDP ソケットの受信バッファーがより速くオーバーフローし、パケットロスが大きくなります。そのため、このような状況では **vhost-net** を無効にしてトラフィックを遅くし、全体的なパフォーマンスを向上させることが推奨されます。

vhost-net を無効にするには、ゲスト仮想マシンの XML 設定ファイルの **<interface>** サブ要素を変更し、次のようにネットワークを定義します。

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

ドライバー名を **qemu** に設定すると、パケット処理が QEMU ユーザー空間に強制され、そのインターフェイスの vhost-net を事実上無効にします。

6.3. VHOST-NET のゼロコピーの有効化

Red Hat Enterprise Linux 7 では、vhost-net zero-copy はデフォルトで無効にされています。永続的にこの動作を有効にするには、以下の内容で、**/etc/modprobe.d** に新しいファイル**vhost-net.conf**を追加します。

```
options vhost_net  experimental_zcopytx=1
```

これを再び無効にする場合は、以下を実行してください。

```
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 番目のコマンドは新しいファイルを作成して (上記のように)、ゼロコピーを無効にします。これを使用しても有効にできますが、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** の出力を確認します。以下が表示されます。

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

6.4. ブリッジネットワーク

ブリッジネットワーク (ネットワークブリッジングまたは仮想ネットワークスイッチングとも呼ばれます) は、仮想マシンのネットワークインターフェイスを、物理インターフェイスと同じネットワークに配置するために使用されます。ブリッジは最小限の設定でネットワークに仮想マシンを表示できるため、管理オーバーヘッドが低減し、ネットワークの複雑さが低減します。ブリッジにはコンポーネントと設定変数がほとんど含まれていないため、必要に応じて、理解とトラブルシューティングが簡単な透過的なセットアップを提供します。

ブリッジは、標準の Red Hat Enterprise Linux ツール、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説明します。

ただし、仮想環境であっても、ホストオペレーティングシステムのネットワークツールを使用すると、ブリッジをより簡単に作成できます。このブリッジ作成方法の詳細については、Red Hat Enterprise Linux 7 Networking Guide を参照してください。

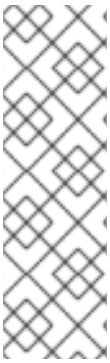
6.4.1. Red Hat Enterprise Linux 7 ホストでのブリッジネットワークの設定

ブリッジネットワークは、仮想化管理ツールとは無関係に、Red Hat Enterprise Linux ホストの仮想マシン用に設定できます。この設定は、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イスである場合、またはホストの管理ネットワークインターフェイスである場合に、主に推奨されます。

仮想化ツールを使用せずにネットワークブリッジを設定する方法は、Red Hat Enterprise Linux 7 Networking Guide を参照してください。

6.4.2. Virtual Machine Manager を使用したブリッジネットワーク

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



注記

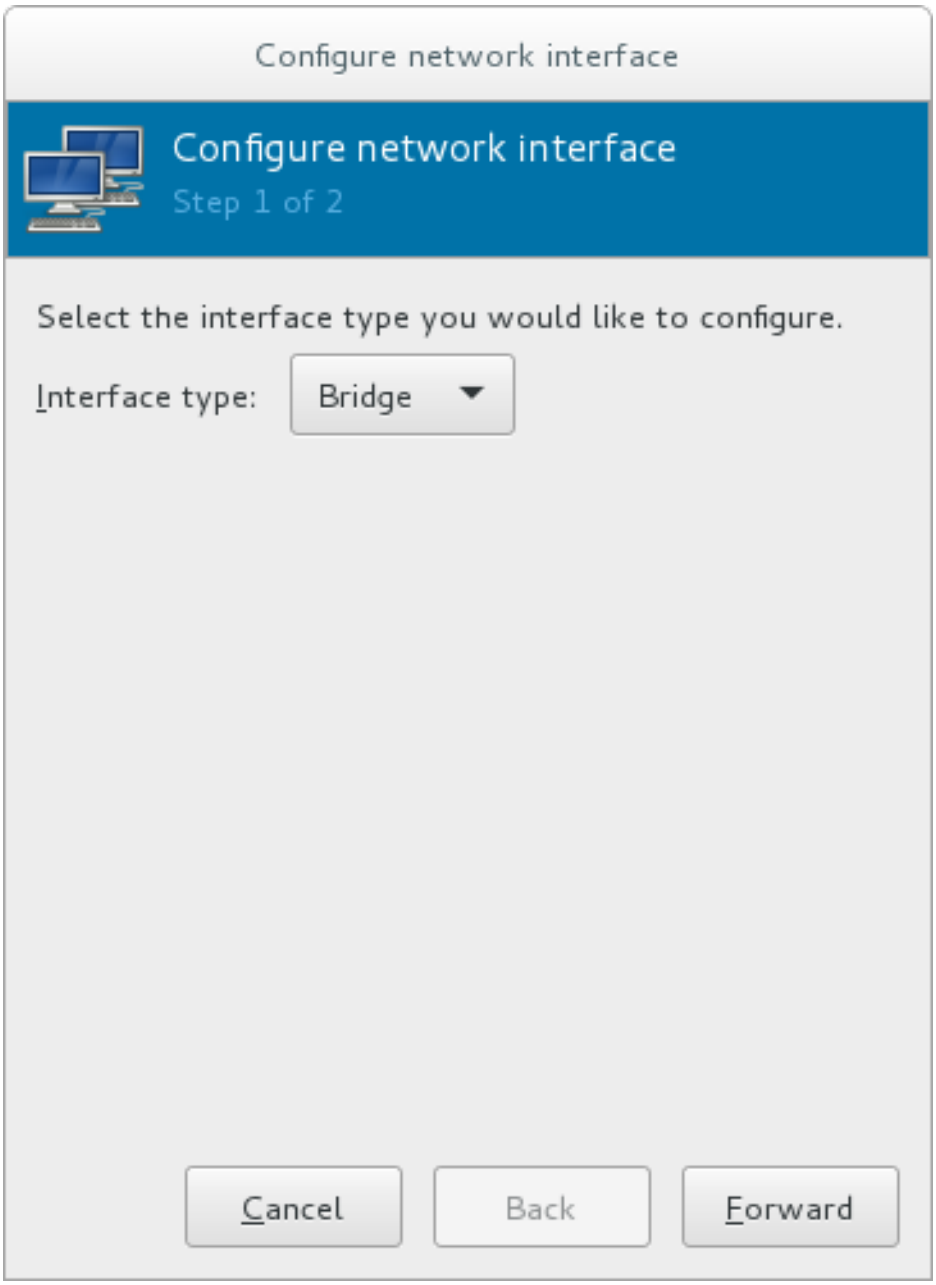
ご使用の環境によっては、Red Hat Enterprise Linux 7 で libvirt ツールでブリッジを設定する際に、ネットワークマネージャーを無効にする必要があります。これは Red Hat では推奨されていません。libvirt で作成したブリッジでは、ネットワーク接続を維持するために、ブリッジに対して libvirtd を実行する必要があります。

ブリッジの作成後に libvirt を使用して仮想マシンのインターフェイスをブリッジに追加しながら、Red Hat Enterprise Linux 7 Networking Guide の説明に従って物理 Red Hat Enterprise Linux ホストでブリッジネットワークを設定することが推奨されます。

手順6.1 virt-manager を使用したブリッジの作成

1. virt-manager のメインメニューで、**Edit ⇒ Connection Details** をクリックして、**Connection Details** ウィンドウを開きます。
2. **Network Interfaces** タブをクリックします。
3. 画面下部の+をクリックして、新しいネットワークインターフェイスを設定します。
4. **Interface type** ドロップダウンメニューで **Bridge** を選択し、**Forward** をクリックして続行します。

図6.1 ブリッジの追加



5. a. **Name** フィールドに、ブリッジの名前 (*br0* など) を入力します。

   b. ドロップダウンメニューから、**Start mode** を選択します。以下のいずれかを選択します。

   - none - ブリッジを非アクティブにします。
   - onboot - 次回のゲスト仮想マシンの再起動時にブリッジをアクティブにします。
   - hotplug - ゲストの仮想マシンが実行中であっても、ブリッジをアクティブにします。

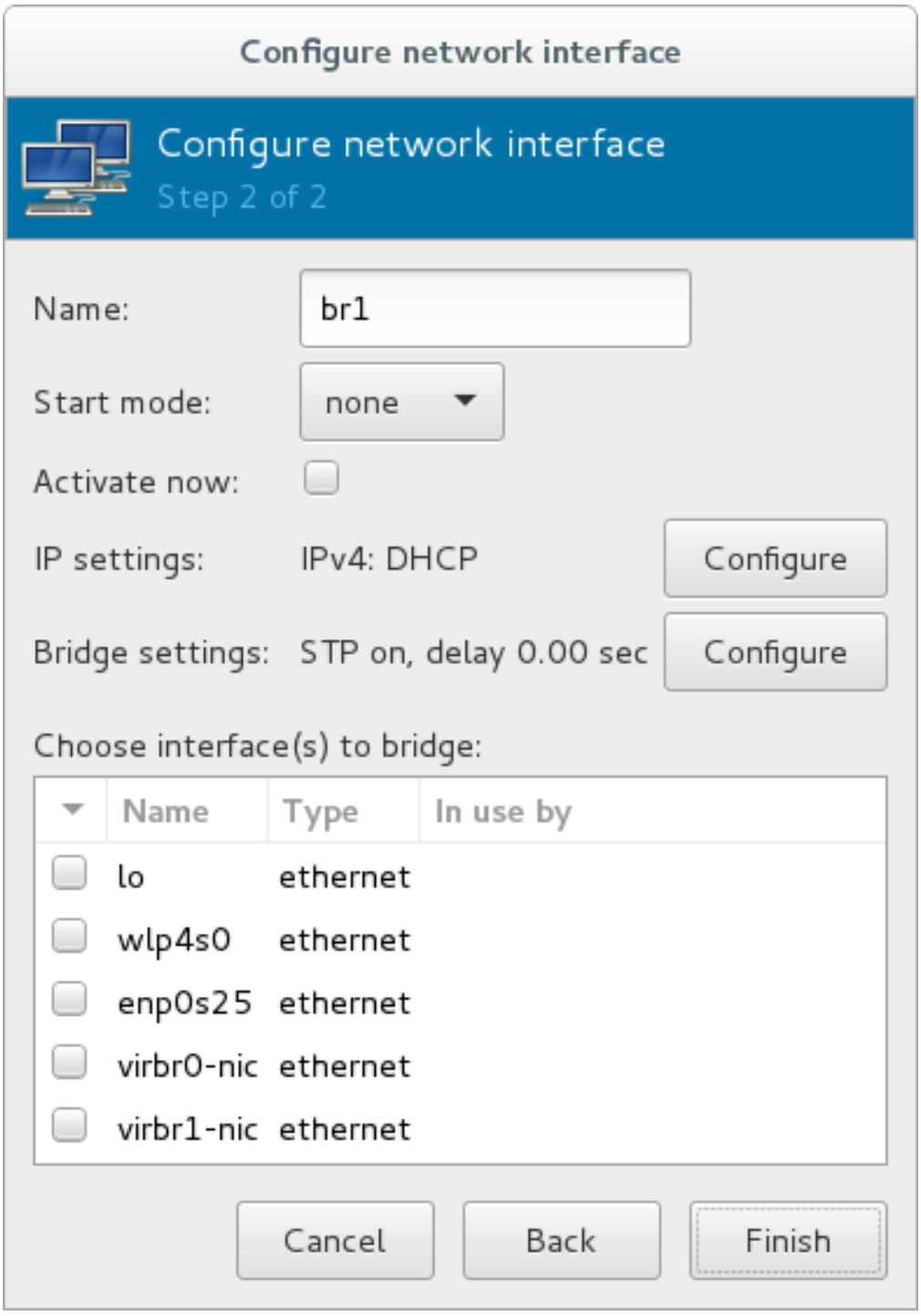
   c. すぐにブリッジを有効にするには、**Activate now** のチェックボックスを選択します。

   d. **IP settings** または **Bridge settings** のいずれかを設定する場合は、適切な **Configure** ボタンをクリックします。別のウィンドウが開いて、必要な設定を指定します。必要な変更を加え、完了したら **OK** をクリックします。

   e. 仮想マシンに接続する物理インターフェイスを選択します。 インターフェイスが現在別のゲスト仮想マシンで使用されている場合は、警告メッセージが表示されます。

6. **Finish** をクリックすると、ウィザードが閉じ、**Connections** メニューに戻ります。

図6.2 ブリッジの追加



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

インターフェイスを停止するには、**Stop Interface** キーを選択します。ブリッジが停止したら、インターフェイスを削除する **Delete Interface** キーを選択します。

6.4.3. libvirt を使用したブリッジネットワーク

ご使用の環境によっては、Red Hat Enterprise Linux 7 で libvirt を使用してブリッジを設定する際に、ネットワークマネージャーを無効にする必要があります。これは Red Hat では推奨されていません。これには、ブリッジを動作させるために libvirtd が実行している必要があります。

Red Hat Enterprise Linux 7 Networking Guide の説明に従って、物理 Red Hat Enterprise Linux ホストでブリッジネットワークを設定することが推奨されます。

重要

libvirt は、新しいカーネル調整可能パラメーターを利用して、ホストブリッジ転送データベース (FDB) エントリーを管理できるようになりました。これにより、複数の仮想マシンをブリッジする際にシステムのネットワークパフォーマンスが改善される可能性があります。ホストコンピューターの XML 設定ファイルで、ネットワークの ***<bridge>*** 要素の ***macTableManager*** 属性を **'libvirt'** に設定します。

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

これにより、すべてのブリッジポートでラーニング (flood) モードが無効になり、libvirt は必要に応じて FDB にエントリーを追加または削除します。MAC アドレス用の適切なフォワーディングポートを学習するオーバーヘッドを取り除くとともに、これにより、カーネルはブリッジをネットワークに接続する物理デバイスで無差別モードを無効にし、オーバーヘッドをさらに削減できます。

第7章 KVM でのオーバーコミット

7.1. 導入部分

KVM ハイパーバイザーは、CPU とメモリーを自動的にオーバーコミットします。つまり、仮想化した CPU とメモリーは、システムにある物理リソースよりも仮想マシンに割り当てることができます。これが可能なのは、ほとんどのプロセスが割り当てられたリソースの 100% に常にアクセスするわけではないためです。

その結果、十分に活用されていない仮想化サーバーまたはデスクトップをより少ないホストで実行できるため、多くのシステムリソースが節約され、サーバーハードウェアへの電力、冷却、および投資が削減されます。

7.2. メモリーのオーバーコミット

KVM ハイパーバイザーで実行しているゲスト仮想マシンには、物理 RAM の専用ブロックが割り当てられていません。代わりに、各ゲスト仮想マシンは、ホスト物理マシンの Linux カーネルが、要求された場合にのみメモリーを割り当てる Linux プロセスとして機能します。また、ホストのメモリーマネージャーは、ゲスト仮想マシンのメモリーを、自身の物理メモリーとスワップ領域の間で移動できます。

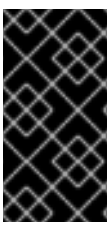
オーバーコミットを行うには、ホストの物理マシンに、すべてのゲスト仮想マシンに対応する十分なスワップ領域と、ホストの物理マシンのプロセスに十分なメモリーを割り当てる必要があります。基本的には、ホスト物理マシンのオペレーティングシステムには、最大 4GB のメモリーと、最小 4GB のスワップ領域が必要です。スワップパーティションに適したサイズを判断する方法は、Red Hat ナレッジベース を参照してください。

重要

オーバーコミットは、一般的なメモリー問題には理想的な解決策ではありません。メモリー不足に対処するための推奨される方法は、ゲストごとに割り当てるメモリーを減らすか、ホストに物理メモリーを追加するか、スワップスペースを利用することです。

仮想マシンを頻繁にスワップすると、仮想マシンの実行が遅くなります。また、オーバーコミットによりシステムのメモリーが不足 (OOM) する可能性があります。これにより、Linux カーネルが重要なシステムプロセスをシャットダウンする可能性があります。メモリーをオーバーコミットする場合は、十分なテストが行われていることを確認してください。オーバーコミットのサポートについては、Red Hat サポートまでご連絡ください。

オーバーコミットはすべての仮想マシンで機能するわけではありませんが、最小限の集中的な使用または KSM で複数の同一ゲストを実行するデスクトップ仮想化セットアップで機能することがわかっています。KSM とオーバーコミットの詳細は、Red Hat Enterprise Linux 7 Virtualization Tuning and Optimization Guide を参照してください。



重要

メモリーのオーバーコミットは、デバイスの割り当てでは対応していません。これは、デバイスの割り当て が使用中の場合に、割り当てられたデバイスでダイレクトメモリーアクセス (DMA) を有効にするには、仮想マシンのすべてのメモリーを静的に事前に割り当てる必要があるためです。

7.3. 仮想 CPU のオーバーコミット

KVM ハイパーバイザーは、仮想化 CPU (vCPU) のオーバーコミットに対応します。仮想化 CPU は、ゲスト仮想マシンの負荷制限が許す限り、オーバーコミットできます。負荷が 100% に近いと、要求が破棄されたり、使用できない応答時間が発生する可能性があるため、vCPU をオーバーコミットする場合は注意してください。

Red Hat Enterprise Linux 7 では、SMP (対称型マルチプロセッシング) 仮想マシンと呼ばれる複数の vCPU でゲストをオーバーコミットすることができます。ただし、仮想マシンで、物理 CPU にあるコアよりも多くのコアを実行すると、パフォーマンスが低下する場合があります。

たとえば、vCPU が 4 つある仮想マシンは、デュアルコアプロセッサーを搭載したホストマシンではなく、クアッドコアホストで実行してください。処理コアの物理数を超えて SMP 仮想マシンをオーバーコミットすると、プログラムの CPU 時間が必要な時間よりも短くなるため、パフォーマンスが大幅に低下します。また、物理プロセッサーコアごとに、割り当てられた vCPU の合計が 10 を超えることは推奨されません。

SMP ゲストでは、処理オーバーヘッドの一部が発生します。CPU オーバーコミットは、SMP のオーバーヘッドを増加させます。これは、タイムスライシングを使用してリソースをゲストに割り当てると、ゲスト内の CPU 間の通信が遅くなる可能性があるためです。このオーバーヘッドは、vCPU の数が多いゲストや、オーバーコミット率が高いゲストで増加します。

仮想化された CPU は、単一のホストに複数のゲストがあり、各ゲストにホスト CPU の数と比較して少数の vCPU がある場合に、最適にオーバーコミットされます。KVM は、1 つのホスト上の 1 つの物理 CPU に対して 5 つの vCPU(5 つの仮想マシン上) の比率で 100% 未満の負荷を持つゲストを安全にサポートする必要があります。KVM ハイパーバイザーは、すべての仮想マシンを切り替えて、負荷のバランスがとれていることを確認します。

パフォーマンスを最大化するために、Red Hat では、各ゲスト内にあるプログラムを実行するのに必要な数の vCPU のみをゲストに割り当てることを推奨しています。

重要

オーバーコミットされた環境では、メモリーを 100% 使用するアプリケーションや処理リソースが不安定になる可能性があります。CPU のオーバーコミット率および SMP の量はワークロードに依存するため、詳細なテストを行わずに実稼働環境でメモリーまたは CPU をオーバーコミットしないでください。

第8章 KVM ゲストのタイミング管理

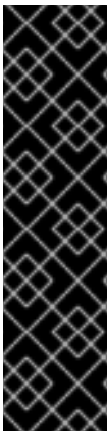
仮想化には、ゲスト仮想マシンでの時間管理に関するいくつかの課題が含まれます。

- 割り込みは、すべてのゲスト仮想マシンに常に同時に瞬時に配信されるとは限りません。これは、仮想マシンの割り込みは真の割り込みではないためです。代わりに、ホストマシンによりゲスト仮想マシンに挿入されます。

- ホストは、別のゲスト仮想マシンを実行している場合や、別のプロセスの場合があります。したがって、割り込みにより通常必要とされる正確なタイミングは、常に可能となるとは限りません。

正確な時刻管理が行われていないゲスト仮想マシンでは、セッションの有効性、移行、およびその他のネットワークアクティビティーがタイムスタンプに依存して正しい状態を維持するため、ネットワークアプリケーションとプロセスで問題が発生する可能性があります。

KVM 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。ただし、ゲストの移行など、時間管理の不正確さによって影響を受ける可能性のあるアクティビティーを試行する前に、タイミングをテストすることは依然として重要です。



重要

上記の問題を回避するには、ホストとゲスト仮想マシンで、ネットワークタイムプロトコル (NTP) を設定する必要があります。Red Hat Enterprise Linux 6 以前を使用するゲストでは、NTP が **ntpd** サービスにより実装されます。詳細は、Red Hat Enterprise 6 Deployment Guide を参照してください。

Red Hat Enterprise Linux 7 を使用するシステムの場合、NTP の時刻同期サービスは、**ntpd** または **chronyd** サービスによって提供できます。Chrony には、仮想マシンに利点があることに注意してください。詳細は、Red Hat Enterprise Linux 7 System Administrator's Guide の Configuring NTP Using the chrony Suite および Configuring NTP Using ntpd セクションを参照してください。

ゲスト仮想マシンの時間同期のメカニズム

デフォルトでは、ゲストは次のようにハイパーバイザーと時刻を同期します。

- ゲストシステムの起動時に、ゲストはエミュレートリアルタイムクロック (RTC) から時間を読み取ります。

- NTP プロトコルが開始すると、ゲストクロックが自動的に同期します。その後、通常のゲスト操作時に、NTP はゲストでクロック調整を実行します。

- 一時停止または復元プロセス後にゲストを再開する場合は、管理ソフトウェアー (virt-manager など) がゲストのクロックを指定した値に同期させるコマンドを実行する必要があります。この同期は、QEMU ゲストエージェントがゲストにインストールされており、この機能に対応している場合にのみ機能します。ゲストのクロックが同期する値は、通常、ホストのクロック値です。

Constant タイムスタンプカウンター (TSC)

最新の Intel および AMD の CPU では、Constant TSC (Time Stamp Counter) が提供されます。消費電力ポリシーに従うなど、CPU コア自体の周波数が変更しても、Constant TSC のカウント周波数は変化しません。TSC を KVM ゲストのクロックソースとして使用するには、Constant TSC 周波数の CPU が必要です。

constant_tsc フラグが存在する場合、CPU には一定のタイムスタンプカウンターがあります。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
```

いずれかの出力が得られると、CPU には **constant_tsc** ビットがあります。出力が表示されない場合は、以下の手順に従ってください。

Constant タイムスタンプカウンターを使用しないホストの設定

TSC の周波数が一定でないシステムは、仮想マシンのクロックソースとして TSC を使用できないため、追加の設定が必要になります。電源管理機能は正確な時間管理を妨げるため、ゲスト仮想マシンが KVM で時間を正確に保持するには、無効にする必要があります。

重要

この手順は、AMD リビジョン F の CPU のみを対象としています。

CPU に **constant_tsc** ビットがない場合は、 で省電力機能をすべて無効にしてください。各システムには、時間を維持するために使用するいくつかのタイマーがあります。TSC はホストで安定していません。これは、**cpufreq**の変更、ディープ C ステート、またはより高速な TSC を使用したホストへの移行が原因である場合があります。ディープ C のスリープ状態は、TSC を停止する可能性があります。ディープ C 状態を使用するカーネルを防ぐには、**processor.max_cstate=1** をカーネルブートに追加します。この変更を永続化するには、**/etc/default/grub**file で **GRUB_CMDLINE_LINUX** キーの値を変更します。たとえば、システムの起動ごとに緊急モードを有効にする場合は、以下のようにエントリーを編集します。

```
GRUB_CMDLINE_LINUX="emergency"
```

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

cpufreq を無効にする (**constant_tsc** を使用しないホストでのみ必要) には、kernel-tools をインストールして、**cpupower.service** (**systemctl enable cpupower.service**) を有効にします。ゲスト仮想マシンが起動するたびにこのサービスを無効にする場合は、**/etc/sysconfig/cpupower** の設定ファイルを変更して CPUPOWER_START_OPTS および CPUPOWER_STOP_OPTS を変更します。有効な制限は、**/sys/devices/system/cpu/*cpuid*/cpufreq/scaling_available_governors** ファイルに記載されています。このパッケージの詳細、または電源管理およびガバナーの詳細は、『Red Hat Enterprise Linux 7 Power Management Guide』 を参照してください。

8.1. ホスト全体の時間同期

KVM ゲストの仮想ネットワークデバイスはハードウェアタイムスタンプをサポートしていません。つまり、NTP や PTP などのネットワークプロトコルを使用するゲストのクロックを数十マイクロ秒よりも高い精度で同期することは困難です。

ゲストの同期をより正確に行う必要がある場合は、NTP または PTP を使用してホストのクロックをハードウェアのタイムスタンプと同期させ、ゲストをホストに直接同期させることが推奨されます。Red Hat Enterprise Linux 7.5 以降では、マイクロ秒未満の精度でゲストをホストに同期できるようにする仮想 PTP ハードウェアクロック (PHC) が提供されます。

重要

PHC が適切に機能するようにするには、ホストとゲストの両方がオペレーティングシステム (OS) として RHEL 7.5 以降を使用している必要があります。

PHC デバイスを有効にするには、ゲスト OS で次の手順を実行します。

1. 再起動後に読み込む**ptp_kvm** モジュールを設定します。

```
# echo ptp_kvm > /etc/modules-load.d/ptp_kvm.conf
```

2. **/dev/ptp0** クロックを chrony 設定のリファレンスとして追加します。

```
# echo "refclock PHC /dev/ptp0 poll 2" >> /etc/chrony.conf
```

3. chrony デーモンを再起動します。

```
# systemctl restart chronyd
```

4. host-guest の時刻同期が正しく設定されていることを確認するには、ゲストで **chronyc sources** コマンドを使用します。この出力は、以下のようになります。

```
# chronyc sources
210 Number of sources = 1
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
====
#* PHC0                          0   2   377     4     -6ns[   -6ns] +/-  726ns
```

8.2. RED HAT ENTERPRISE LINUX ゲストに必要な時間管理パラメーター

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

注記

Red Hat Enterprise Linux 5.5 以降、Red Hat Enterprise Linux 6.0 以降、および Red Hat Enterprise Linux 7 では、**kvm-clock** がデフォルトのクロックソースとして使用されます。**kvm-clock** を実行するとカーネルパラメーターを追加する必要がなくなるため、Red Hat では推奨しています。

以下の表は、Red Hat Enterprise Linux のバージョンと、指定したシステムで必要なパラメーターを一覧表示しています。

表8.1 カーネルパラメーターの要件

Red Hat Enterprise Linux のバージョン	ゲストカーネルの追加パラメーター
7.0 以降 (AMD64 および Intel 64 システム、kvm-clock 使用)	追加のパラメーターは必要ありません。

Red Hat Enterprise Linux のバージョン	ゲストカーネルの追加パラメーター
6.1 以降 (AMD64 および Intel 64 システム、kvm-clock 使用)	追加のパラメーターは必要ありません。
6.0 (AMD64 および Intel 64 システム、kvm-clock 使用)	追加のパラメーターは必要ありません。
6.0 (AMD64 および Intel 64 システム、kvm-clock 使用せず)	notsc lpj=*n*

注記

lpj パラメーターでは、ゲスト仮想マシンを実行している特定の CPU の **jiffy あたりのループ数** 値と同じ数値が必要です。この値がわからない場合は、***lpj*** パラメーターを設定しないでください。

8.3. スチールタイムアカウンティング

スチールタイムは、ホストが提供していないゲスト仮想マシンが必要とする CPU 時間の量です。スチールタイムは、ホストがこれらのリソースを別の場所 (たとえば、別のゲスト) に割り当てるときに発生します。

スチールタイムは、**/proc/stat** の CPU 時間フィールドで報告されます。これは、**top** および **vmstat** などのユーティリティーにより自動的に報告されます。これは、"%st" または "st" 列で表示されます。これは、オフにできないことに注意してください。

スチールタイムが長くなると CPU 競合が発生するため、ゲストのパフォーマンスが低下す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 クォータを上げるか、ホストで実行するゲストを減らします。

第9章 LIBVIRT でのネットワークブート

ゲストの仮想マシンは、PXE を有効にして起動できます。PXE を使用すると、ゲスト仮想マシンを起動して、ネットワーク自体から設定を読み込むことができます。本セクションでは、libvirt で PXE ゲストを設定する基本的な設定手順を説明します。

本セクションでは、ブートイメージまたは PXE サーバーの作成は説明しません。これは、プライベートネットワーク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するために使用されます。

警告

以下の手順は、例としてのみ提供されます。続行する前に、十分なバックアップがあることを確認してください。

9.1. ブートサーバーの準備

本章の手順を実行するには、以下が必要になります。

- PXE サーバー (DHCP および TFTP) - libvirt の内部サーバー、手動で設定した dhcpd および tftpd、dnsmasq、Cobbler が設定したサーバー、またはその他のサーバーを指します。
- ブートイメージ - たとえば、手動または Cobbler で設定した PXELINUX。

9.1.1. プライベートの libvirt ネットワークに PXE ブートサーバーを設定する

この例では、**デフォルト** のネットワークを使用します。以下の手順を実行します。

手順9.1 PXE ブートサーバーの設定

1. PXE ブートイメージおよび設定を **/var/lib/tftpboot** に置きます。
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の **<ip>** 要素を編集して、適切なアドレス、ネットワークマスク、DHCP アドレス範囲、および起動ファイルを追加します。*BOOT_FILENAME* は、ゲスト仮想マシンの起動に使用するファイル名を表します。

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

4. 以下を実行します。

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

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

9.2. PXE を使用したゲストの起動

本セクションでは、PXE でゲスト仮想マシンを起動する方法を説明します。

9.2.1. ブリッジネットワークの使用

手順9.2 PXE およびブリッジネットワークを使用したゲストの起動

1. ネットワークで PXE ブートサーバーが使用できるように、ブリッジが有効になっていることを確認します。

2. PXE 起動が有効になっているゲスト仮想マシンを起動します。以下のコマンド例に示すように、**virt-install** コマンドを使用して、PXE ブートが有効になっている新しい仮想マシンを作成できます。

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

または、以下の例のように、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、XML ゲスト設定ファイルの **<os>** 要素内に **<boot dev='network'/>** 要素があることを確認します。

```
<os>
  <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
  <boot dev='network'/>
  <boot dev='hd'/>
</os>
<interface type='bridge'>
  <mac address='52:54:00:5a:ad:cb'/>
  <source bridge='breth0'/>
  <target dev='vnet0'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

9.2.2. プライベートの libvirt ネットワークの使用

手順9.3 プライベートの libvirt ネットワークの使用

1. 「プライベートの libvirt ネットワークに PXE ブートサーバーを設定する」 に示すように、libvirt で PXE ブートを設定します。

2. PXE 起動を有効にして libvirt を使用してゲスト仮想マシンを起動します。**virt-install** コマンドを使用して、PXE を使用して新しい仮想マシンを作成/インストールできます。

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

または、以下の例のように、ゲストネットワークがプライベート libvirt ネットワークを使用するように設定されており、XML ゲスト設定ファイルの **<os>** 要素内に **<boot dev='network'/>** 要素があることを確認します。

```
<os>
  <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
  <boot dev='network'/>
  <boot dev='hd'/>
</os>
```

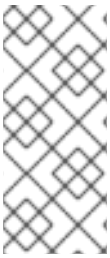
また、ゲスト仮想マシンがプライベートネットワークに接続されていることを確認します。

```
<interface type='network'>
  <mac address='52:54:00:66:79:14'/>
  <source network='default'/>
  <target dev='vnet0'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

第10章 ハイパーバイザーおよび仮想マシンの登録

Red Hat Enterprise Linux 6 および 7 では、すべてのゲスト仮想マシンが特定のハイパーバイザーにマッピングされ、すべてのゲストに同じレベルのサブスクリプションサービスが割り当てられるようにする必要があります。これを行うには、インストールされて登録されている各 KVM ハイパーバイザーで、すべてのゲスト仮想マシンを自動的に検出するサブスクリプションエージェントをインストールする必要があります。これにより、ホストにあるマッピングファイルが作成されます。このマッピングファイルにより、すべてのゲスト仮想マシンに次の利点があります。

- 仮想システムに固有のサブスクリプションは簡単に利用でき、関連するすべてのゲスト仮想マシンに適用できます。
- ハイパーバイザーから継承できるすべてのサブスクリプションのメリットは、すぐに利用でき、関連するすべてのゲスト仮想マシンに適用できます。



注記

本章で説明する情報は、Red Hat Enterprise Linux のサブスクリプションのみを対象としています。Red Hat Virtualization サブスクリプションまたは Red Hat Satellite サブスクリプションもある場合は、それらのサブスクリプションで提供される virt-who 情報も参照してください。Red Hat Subscription Management の詳細は、カスタマーポータルの Red Hat Subscription Management ガイドを参照してください。

10.1. ホストの物理マシンに VIRT-WHO をインストールする

1. **KVM ハイパーバイザーの登録**
   ホスト物理マシンの root ユーザーで、ターミナルで **subscription-manager register [options]** コマンドを実行して、KVM Hypervisor を登録します。# **subscription-manager register --help** メニューを使用すると、より多くのオプションを使用できます。ユーザーネームおよびパスワードを使用している場合は、**Subscription Manager** アプリケーションが認識している認証情報を使用します。初めてサブスクライブする際に、ユーザーアカウントがない場合は、カスタマーサポートにご連絡ください。たとえば、仮想マシンを 'admin' として登録 (パスワードは 'secret') するには、次のコマンドを送信します。

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

2. **virt-who パッケージをインストールします。**
   ホストの物理マシンで次のコマンドを実行して、virt-who パッケージをインストールします。

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

3. **virt-who 設定ファイルの作成**
   ハイパーバイザーごとに、**/etc/virt-who.d/** ディレクトリーに設定ファイルを追加します。ファイルには、少なくとも以下のスニペットが含まれている必要があります。

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

   virt-who の設定の詳細は、「**virt-who** の設定」 を参照してください。

4. **virt-who サービスを再起動します。**
   ホストの物理マシンで次のコマンドを実行して、virt-who サービスを起動します。

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

5. **virt-who サービスがゲスト情報を受信していることを確認する**
この時点で、virt-who サービスは、ホストからドメインのリストの収集を開始します。ホストの物理マシンの **/var/log/rhsm/rhsm.log** ファイルを確認して、ゲスト仮想マシンのリストがファイルに含まれていることを確認します。例:

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

手順10.1 カスタマーポータルでのサブスクリプションの管理

1. **ハイパーバイザーのサブスクライブ**
仮想マシンはハイパーバイザーと同じサブスクリプションのメリットを享受するため、ハイパーバイザーに有効なサブスクリプションがあり、仮想マシンがサブスクリプションを使用できることが重要です。

   a. **カスタマーポータルへのログイン**
   Red Hat カスタマーポータル で Red Hat アカウントの認証情報を指定してログインします。

   b. **Systems リンクをクリックします。**
   My Subscriptions インターフェイスの Systems セクションに移動します。

   c. **ハイパーバイザーを選択します。**
   Systems ページには、サブスクライブしているすべてのシステムの表があります。ハイパーバイザーの名前 (**localhost.localdomain** など) を選択します。表示された詳細ページで **Attach a subscription** をクリックし、リスト表示されたすべてのサブスクリプションを選択します。**Attach Selected** をクリックします。これにより、ホストの物理サブスクリプションがハイパーバイザーに割り当てられ、ゲストがサブスクリプションを利用できるようになります。

2. **ゲスト仮想マシンのサブスクライブ - 初めての使用**
この手順は、新しいサブスクリプションを持っていて、これまでゲスト仮想マシンをサブスクライブしたことがない人を対象としています。仮想マシンを追加する場合は、この手順をスキップします。virt-who サービスを実行しているマシンで、ハイパーバイザープロファイルに割り当てられたサブスクリプションを使用するには、ゲスト仮想マシンのターミナルで次のコマンドを実行して自動サブスクライブします。

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

3. **追加のゲスト仮想マシンのサブスクライブ**
初めて仮想マシンをサブスクライブする場合は、この手順を省略してください。仮想マシンを追加する場合、このコマンドを実行しても、必ずしも同じサブスクリプションがゲスト仮想マシンに再接続されるとは限らないことに注意してください。これは、すべてのサブスクリプションを削除してから自動登録を許可して、指定したゲスト仮想マシンに必要なものを解決できるようにすると、以前とは異なるサブスクリプションが使用される可能性があるためです。これはシステムには影響を及ぼさない可能性がありますが、注意してください。以下で説明されていない手動の接続手順を使用して仮想マシンを接続した場合は、自動接続が機能しないため、これらの仮想マシンを手動で再接続する必要があります。次のコマンドを使用して、最初に古いゲストのサブスクリプションを削除してから、自動アタッチを使用してすべてのゲストにサブスクリプションを割り当てます。ゲスト仮想マシンで次のコマンドを実行します。

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

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

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

以下のような出力が表示されます。サブスクリプションの詳細に注意してください。
'Subscription is current' と表示されるはずです。

```
[root@virt-who ~]# subscription-manager list --consumed
+-------------------------------------------+
   Consumed Subscriptions
+-------------------------------------------+
Subscription Name: Awesome OS with unlimited virtual guests
Provides:   Awesome OS Server Bits
SKU:    awesomeos-virt-unlimited
Contract:   0
Account:   ######### Your account number #####
Serial:   ######### Your serial number ######
Pool ID:   XYZ123
Provides Management:  No
Active:   True
Quantity Used:   1
Service Level:
Service Type:
Status Details:  Subscription is current
Subscription Type:
Starts:   01/01/2015
Ends:    12/31/2015
System Type:   Virtual
```

???
The ID for the subscription to attach to the system is displayed here. You will need this ID if you need to attach the subscription manually.

???
Indicates if your subscription is current. If your subscription is not current, an error message appears. One example is Guest has not been reported on any host and is using a temporary unmapped guest subscription. In this case the guest needs to be subscribed. In other cases, use the information as indicated in 「サブスクリプションステータスエラーが表示された場合の対応」.

5. **追加のゲストを登録**
ハイパーバイザーに新しいゲスト仮想マシンをインストールする場合は、ゲスト仮想マシンで次のコマンドを実行して、新しい仮想マシンを登録し、ハイパーバイザーに割り当てたサブスクリプションを使用する必要があります。

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

10.1.1. virt-who の設定

virt-who サービスは、以下のファイルを使用して設定します。

- **/etc/virt-who.conf** - 接続されているハイパーバイザーで変更を確認する間隔など、一般的な設定情報が含まれます。
- **/etc/virt-who.d/*hypervisor_name*.conf** - 特定のハイパーバイザーの設定情報が含まれます。

ハイパーバイザー設定ファイルと、**virt-who.conf**に必要なスニペットを生成する Web ベースのウィザードが提供されます。ウィザードを実行する場合は、カスタマーポータルの Red Hat Virtualization Agent (virt-who) Configuration Helper を参照します。

ウィザードの 2 ページ目で、次のオプションを選択します。

- Where does your virt-who report to?: **Subscription Asset Manager**
- Hypervisor Type: **libvirt**

ウィザードに従って設定を完了します。設定が正しく行われると、**virt-who** は選択したサブスクリプションを、指定されたハイパーバイザーにある既存および今後のゲストに自動的に提供します。

ハイパーバイザー設定ファイルの詳細は、**virt-who-config** の man ページを参照してください。

10.2. 新しいゲスト仮想マシンの登録

登録済みで実行中のホストで、新しいゲスト仮想マシンを作成する場合は、virt-who サービスも実行している必要があります。これにより、virt-who サービスがゲストをハイパーバイザーにマップできるため、システムが仮想システムとして適切に登録されます。仮想マシンを登録するには、次のコマンドを実行します。

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

10.3. ゲスト仮想マシンエントリーの削除

ゲスト仮想マシンが実行されている場合は、ターミナルウィンドウでゲストの root として次のコマンドを実行して、システムの登録を解除します。

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

ただし、システムが削除された場合、仮想サービスは、サービスが削除されたか、一時停止したかを認識できません。この場合は、以下の手順に従って、サーバー側からシステムを手動で削除する必要があります。

1. **サブスクリプションマネージャーへのログイン**
   サブスクリプションマネージャーは、Red Hat カスタマーポータル にあります。画面最上部にあるログインアイコンをクリックし、ユーザー名とパスワードを使用してカスタマーポータルにログインします。
2. **Subscriptions タブをクリックします。**
   Subscriptions タブをクリックします。
3. **Systems リンクをクリックします。**

ページを下にスクロールして、**Systems** をクリックします。

4. **システムを削除します。**
システムプロファイルを削除するには、テーブルで指定したシステムのプロファイルを検索し、その名前の横にあるチェックボックスを選択して、**Delete** をクリックします。

10.4. 手動での VIRT-WHO のインストール

本セクションでは、ハイパーバイザーが提供するサブスクリプションを手動で割り当てる方法を説明します。

手順10.2 サブスクリプションを手動で割り当てる方法

1. **サブスクリプション情報のリストを表示し、プール ID を見つけます。**
最初に、仮想タイプで利用可能なサブスクリプションをリスト表示する必要があります。以下のコマンドを入力します。

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

表示されたプール ID をメモします。この ID は、次の手順で必要に応じてコピーします。

2. **プール ID を使用したサブスクリプションの割り当て**
前の手順でコピーしたプール ID を使用して、attach コマンドを実行します。プール ID *XYZ123* を、取得したプール ID に置き換えます。以下のコマンドを入力します。

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

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

10.5. VIRT-WHO のトラブルシューティング

10.5.1. ハイパーバイザーのステータスが赤である理由

シナリオ: サーバー側で、サブスクリプションを持たないハイパーバイザーにゲストをデプロイします。24 時間後、ハイパーバイザーはステータスを赤で表示します。この状況を改善するには、そのハイパーバイザーのサブスクリプションを取得する必要があります。または、サブスクリプションを使用してゲストをハイパーバイザーに永続的に移行します。

10.5.2. サブスクリプションステータスエラーが表示された場合の対応

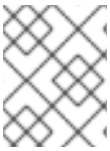
シナリオ: 以下のいずれかのエラーメッセージが表示されます。

- System not properly subscribed
- Status unknown
- Late binding of a guest to a hypervisor through virt-who (host/guest mapping)

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

第11章 QEMU ゲストエージェントおよび SPICE エージェントでの仮想化の強化

QEMU ゲストエージェント、SPICE エージェントなどの Red Hat Enterprise Linux のエージェントをデプロイすると、仮想化ツールをシステムでより最適に実行できます。この章では、これらのエージェントについて説明します。



注記

ホストおよびゲストのパフォーマンスをさらに最適化および調整する方法は、Red Hat Enterprise Linux 7 Virtualization Tuning and Optimization Guide を参照してください。

11.1. QEMU ゲストエージェント

QEMU ゲストエージェントはゲスト内で実行し、libvirt を使用して、ホストマシンがゲストオペレーティングシステムにコマンドを実行できるようになります。これにより、ファイルシステムのフリーズや解凍などの機能をサポートします。ゲストオペレーティングシステムは、これらのコマンドに非同期に応答します。QEMU ゲストエージェントパッケージ qemu-guest-agent は、Red Hat Enterprise Linux 7 にデフォルトでインストールされます。

本セクションでは、ゲストエージェントが使用できる libvirt コマンドとオプションを説明します。

重要

信頼できるゲストによって実行される場合にのみ、QEMU ゲストエージェントに依存することが安全であることに注意してください。信頼できないゲストは、ゲストエージェントプロトコルを悪意を持って無視または悪用する可能性があります。ホストに対するサービス拒否攻撃を防ぐための保護機能が組み込まれていますが、ホストは、操作を期待どおりに実行するためにゲストの協力を必要とします。

QEMU ゲストエージェントを使用して、ゲストの実行中に仮想 CPU(vCPU) を有効または無効にできるため、ホットプラグおよびホットアンプラグ機能を使用せずに vCPU の数を調整できることに注意してください。詳細は、「仮想 CPU 数の設定」 を参照してください。

11.1.1. QEMU ゲストエージェントとホスト間の通信の設定

ホストマシンは、ホストとゲストマシン間の VirtIO シリアル接続を介して QEMU 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 (通常は Unix ソケット) を介してホストに接続し、ゲストはこのシリアルチャネルをリッスンします。

注記

qemu-guest-agentは、ホストが VirtIO シリアルチャンネルをリッスンしているかどうかを検出しません。ただし、このチャネルの現在の使用法はホストからゲストへのイベントをリッスンすることであるため、リスナーなしでチャネルに書き込むことによってゲスト仮想マシンが問題に遭遇する可能性は非常に低くなります。また、qemu-guest-agent プロトコルには、コマンド実行時にホスト物理マシンがゲスト仮想マシンを強制的に再同期させる同期マーカー (synchronization marker) が含まれ、libvirt がこのマーカーをすでに使用しているため、ゲスト仮想マシンは、保留中の未配信の応答を安全に破棄することができます。

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

QEMU ゲストエージェントは、仮想マシンの実行またはシャットダウンで設定できます。実行中のゲストで設定されている場合は、ゲストはすぐにゲストエージェントの使用を開始します。ゲストがシャットダウンすると、次回の起動時に QEMU ゲストエージェントが有効になります。

virsh または **virt-manager** のいずれかを使用して、ゲストと QEMU ゲストエージェント間の通信を設定できます。ここでは、Linux ゲストで QEMU ゲストエージェントを設定する方法を説明します。

手順11.1 シャットダウンした Linux ゲストで、ゲストエージェントとホストとの間の通信を virsh を使用して設定する

1. **仮想マシンのシャットダウン**
   QEMU ゲストエージェントを設定する前に、仮想マシン (この例では *rhel7* という名前) がシャットダウンしていることを確認します。

   ```
   # virsh shutdown rhel7
   ```

2. **QEMU ゲストエージェントチャネルをゲスト XML 設定に追加します。**
   ゲストの XML ファイルを編集して、QEMU ゲストエージェントの詳細を追加します。

   ```
   # virsh edit rhel7
   ```

   ゲストの XML ファイルに以下を追加し、変更を保存します。

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

3. **仮想マシンの起動**

   ```
   # virsh start rhel7
   ```

4. **ゲストに QEMU ゲストエージェントをインストールします。**
   ゲスト仮想マシンに QEMU ゲストエージェントがインストールされていない場合はインストールします。

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

5. **ゲストでの QEMU ゲストエージェントの起動**
   ゲストで QEMU ゲストエージェントサービスを起動します。

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

または、次の手順で、実行中のゲストに QEMU ゲストエージェントを設定できます。

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

1. **QEMU ゲストエージェントの XML ファイルの作成**

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

2. **QEMU ゲストエージェントの仮想マシンへの割り当て**
このコマンドを使用して、QEMU ゲストエージェントを、実行中の仮想マシン (この例では *rhel7*) に割り当てます。

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

3. **ゲストに QEMU ゲストエージェントをインストールします。**
ゲスト仮想マシンに QEMU ゲストエージェントがインストールされていない場合はインストールします。

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

4. **ゲストでの QEMU ゲストエージェントの起動**
ゲストで QEMU ゲストエージェントサービスを起動します。

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

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

1. **仮想マシンのシャットダウン**
QEMU ゲストエージェントを設定する前に、仮想マシンがシャットダウンしていることを確認します。

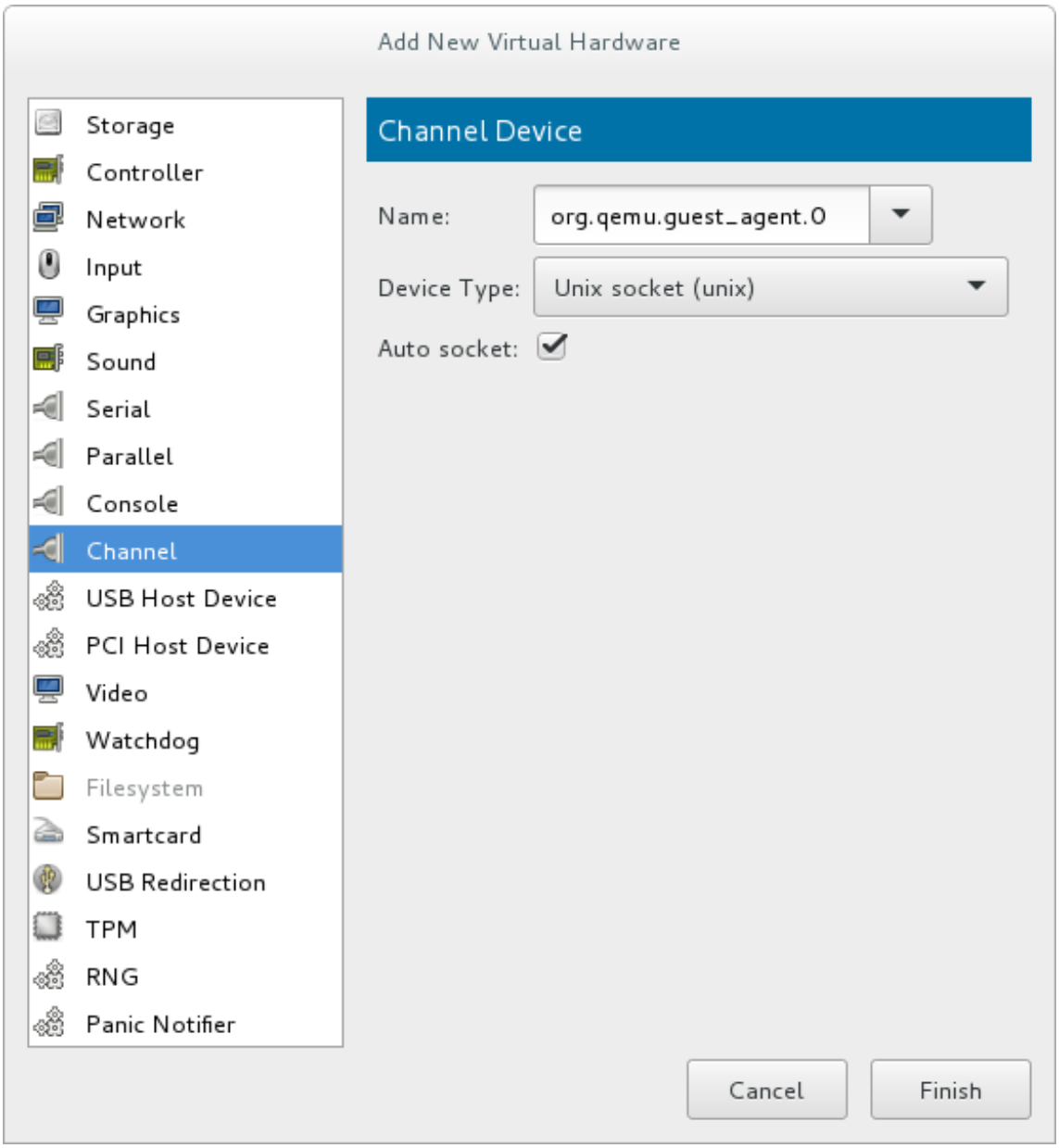
   仮想マシンをシャットダウンするには、**Virtual Machine Manager**の仮想マシン一覧から仮想マシンを選択し、メニューバーのライトスイッチアイコンをクリックします。

2. **QEMU ゲストエージェントチャネルをゲストに追加します。**
ゲストウィンドウ上部の電球アイコンをクリックして、仮想マシンのハードウェアの詳細を開きます。

   Add Hardware ボタンをクリックして **Add New Virtual Hardware** ウィンドウを開き、**Channel** を選択します。

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

図11.1 QEMU ゲストエージェントチャネルデバイスの選択



3. **仮想マシンの起動**
   仮想マシンを起動するには、**Virtual Machine Manager** の仮想マシンリストから仮想マシンを選択し、メニューバーの  をクリックします。

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

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

5. **ゲストでの QEMU ゲストエージェントの起動**
   ゲストで QEMU ゲストエージェントサービスを起動します。

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

これで、*rhel7* 仮想マシンに QEMU ゲストエージェントが設定されます。

11.2. LIBVIRT での QEMU ゲストエージェントの使用

QEMU ゲストエージェントをインストールすると、さまざまな libvirt コマンドのパフォーマンスが向上します。ゲストエージェントは、次の**virsh** コマンドを拡張します。

- **virsh shutdown --mode=agent** - このシャットダウン方法は、 **virsh shutdown --mode=acpi** よりも信頼性が高くなります。QEMU ゲストエージェントで使用する **virsh shutdown** は、クリーンな状態で協調ゲストをシャットダウンすることが保証されます。エージェントが存在しない場合、libvirt は代わりに ACPI シャットダウンイベントの挿入に依存する必要がありますが、一部のゲストはそのイベントを無視するため、シャットダウンしません。

  virsh reboot 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 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、データベーストランザクションの一部を失うことなくスナップショットを使用できます。ゲストエージェントは、ゲストの相互作用を提供することで、高レベルのディスクコンテンツの安定性を実現します。

- **virsh domfsfreeze** および **virsh domfsthaw** - ゲストファイルシステムを分離して静止します。

- **virsh domfstrim** - ファイルシステムをトリミングするようにゲストに指示します。

- **virsh domtime** - ゲストの時計をクエリーまたは設定します。

- **virsh setvcpus --guest** - CPU をオフラインにするようにゲストに指示します。

- **virsh domifaddr --source agent** - ゲストエージェントを介してゲストオペレーティングシステムの IP アドレスを問い合わせます。

- **virsh domfsinfo** - 実行中のゲストでマウントされているファイルシステムのリストを表示します。

- **virsh set-user-password** - ゲストのユーザーアカウントのパスワードを設定します。

11.2.1. ゲストディスクのバックアップの作成

libvirt は、qemu-guest-agent と通信して、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的に整合性があり、必要に応じて使用できるようにすることができます。ゲストシステム管理者は、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てインストールできます。ファイルシステムをフリーズする前に、qemu-guest-agent はメインフックスクリプト (qemu-guest-agent パッケージに含まれる) を呼び出します。フリーズプロセスでは、ゲスト仮想マシンのアプリケーションがすべて一時的に非アクティブになります。

スナップショットプロセスは、以下の手順で設定されます。

- ファイルシステムアプリケーション/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。

- アプリケーションがデータファイルを一貫した状態にします。

- メインフックスクリプトが返されます。

- qemu-guest-agent がファイルシステムをフリーズし、、管理スタックが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した。
- ファイルシステムの機能が再開します。

フリーズ解除は逆の順序で行われます。

ゲストのファイルシステムのスナップショットを作成するには、**virsh snapshot-create --quiesce --disk-only** コマンドを実行します (または、**virsh snapshot-create-as *guest_name* --quiesce --disk-only** を実行します。詳細は 「現在のゲスト仮想マシンのスナップショットの作成」 を参照)。

注記

アプリケーション固有のフックスクリプトでは、データベースと通信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に、正しく実行するために様々な SELinux のパーミッションが必要になることがあります。一般に、ローカル SELinux ポリシーは、そのような目的のために開発およびインストールする必要があります。**/etc/qemu-ga/fsfreeze-hook.d/** というラベルが付いた表の行の 表11.1「QEMU ゲストエージェントパッケージの内容」 にリスト表示されている **restorecon -FvvR** コマンドを実行した後、ファイルシステムノードへのアクセスがすぐに機能するようになります。

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

表11.1 QEMU ゲストエージェントパッケージの内容

File name	説明
/usr/lib/systemd/system/qemu-guest-agent.service	QEMU ゲストエージェントのサービス制御スクリプト (start/stop)
/etc/sysconfig/qemu-ga	**/usr/lib/systemd/system/qemu-guest-agent.service** 制御スクリプトにより読み込まれる QEMU ゲストエージェントの設定ファイル。設定は、シェルスクリプトのコメントが記載されたファイルで説明されています。
/usr/bin/qemu-ga	QEMU ゲストエージェントのバイナリーファイル。
/etc/qemu-ga	フックスクリプトのルートディレクトリー。
/etc/qemu-ga/fsfreeze-hook	メインフックスクリプト。ここでは変更は必要ありません。
/etc/qemu-ga/fsfreeze-hook.d	個々のアプリケーション固有のフックスクリプトのディレクトリー。ゲストシステム管理者は、フックスクリプトを手動でこのディレクトリーにコピーし、適切なファイルモードビットを確認してから、このディレクトリーで **restorecon -FvvR** を実行する必要があります。

File name	説明
/usr/share/qemu-kvm/qemu-ga/	サンプルスクリプトを含むディレクトリー (サンプル目的のみ)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、独自のメッセージと、アプリケーション固有のスクリプトの標準出力およびエラーメッセージを、ログファイル **/var/log/qemu-ga/fsfreeze-hook.log** に記録します。詳細は、libvirt アップストリームの Web サイト を参照してください。

11.3. SPICE エージェント

SPICE エージェントは、ゲストオペレーティングシステムと SPICE クライアントの統合を支援することで、**virt-manager** などのグラフィカルアプリケーションの実行をより円滑に支援します。

たとえば、**virt-manager** でウィンドウのサイズを変更する場合、SPICE エージェントはクライアントの解像度に対する X セッションの解像度の自動調整を可能にします。SPICE エージェントは、ホストとゲスト間のコピーアンドペーストにも対応しており、マウスカーソルの遅延を防ぎます。

SPICE エージェントの機能に関するシステム固有の情報は、spice-vdagent パッケージの README ファイルを参照してください。

11.3.1. SPICE エージェントとホスト間の通信の設定

SPICE エージェントは、仮想マシンの実行またはシャットダウンで設定できます。実行中のゲストで設定されている場合は、ゲストはすぐにゲストエージェントの使用を開始します。ゲストがシャットダウンすると、システムの次回起動時に SPICE エージェントが有効になります。

virsh または **virt-manager** のいずれかを使用して、ゲストと SPICE エージェント間の通信を設定できます。ここでは、Linux ゲストで SPICE エージェントを設定する方法を説明します。

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

1. **仮想マシンのシャットダウン**
   SPICE エージェントを設定する前に、仮想マシン (この例では *rhel7* という名前) がシャットダウンしていることを確認します。

   ```
   # virsh shutdown rhel7
   ```

2. **SPICE エージェントチャネルをゲスト XML 設定に追加します。**
   ゲストの XML ファイルを編集して、SPICE エージェントの詳細を追加します。

   ```
   # virsh edit rhel7
   ```

   ゲストの XML ファイルに以下を追加し、変更を保存します。

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

3. **仮想マシンの起動**

```
# virsh start rhel7
```

4. **ゲストに SPICE エージェントをインストールします。**
ゲスト仮想マシンに SPICE エージェントがインストールされていない場合はインストールします。

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

5. **ゲストで SPICE エージェントを起動します。**
ゲストで SPICE エージェントサービスを起動します。

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

または、次の手順で、実行中のゲストに SPICE エージェントを設定できます。

手順11.5 SPICE エージェントと、実行中の Linux ゲストのホストとの間の通信の設定

1. **SPICE エージェントの XML ファイルの作成**

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

2. **SPICE エージェントの仮想マシンへの割り当て**
このコマンドを使用して、SPICE エージェントを、実行中の仮想マシン (この例では *rhel7*) に割り当てます。

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

3. **ゲストに SPICE エージェントをインストールします。**
ゲスト仮想マシンに SPICE エージェントがインストールされていない場合はインストールします。

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

4. **ゲストで SPICE エージェントを起動します。**
ゲストで SPICE エージェントサービスを起動します。

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

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

1. **仮想マシンのシャットダウン**
SPICE エージェントを設定する前に、仮想マシンがシャットダウンしていることを確認します。

   仮想マシンをシャットダウンするには、**Virtual Machine Manager**の仮想マシン一覧から仮想マシンを選択し、メニューバーのライトスイッチアイコンをクリックします。

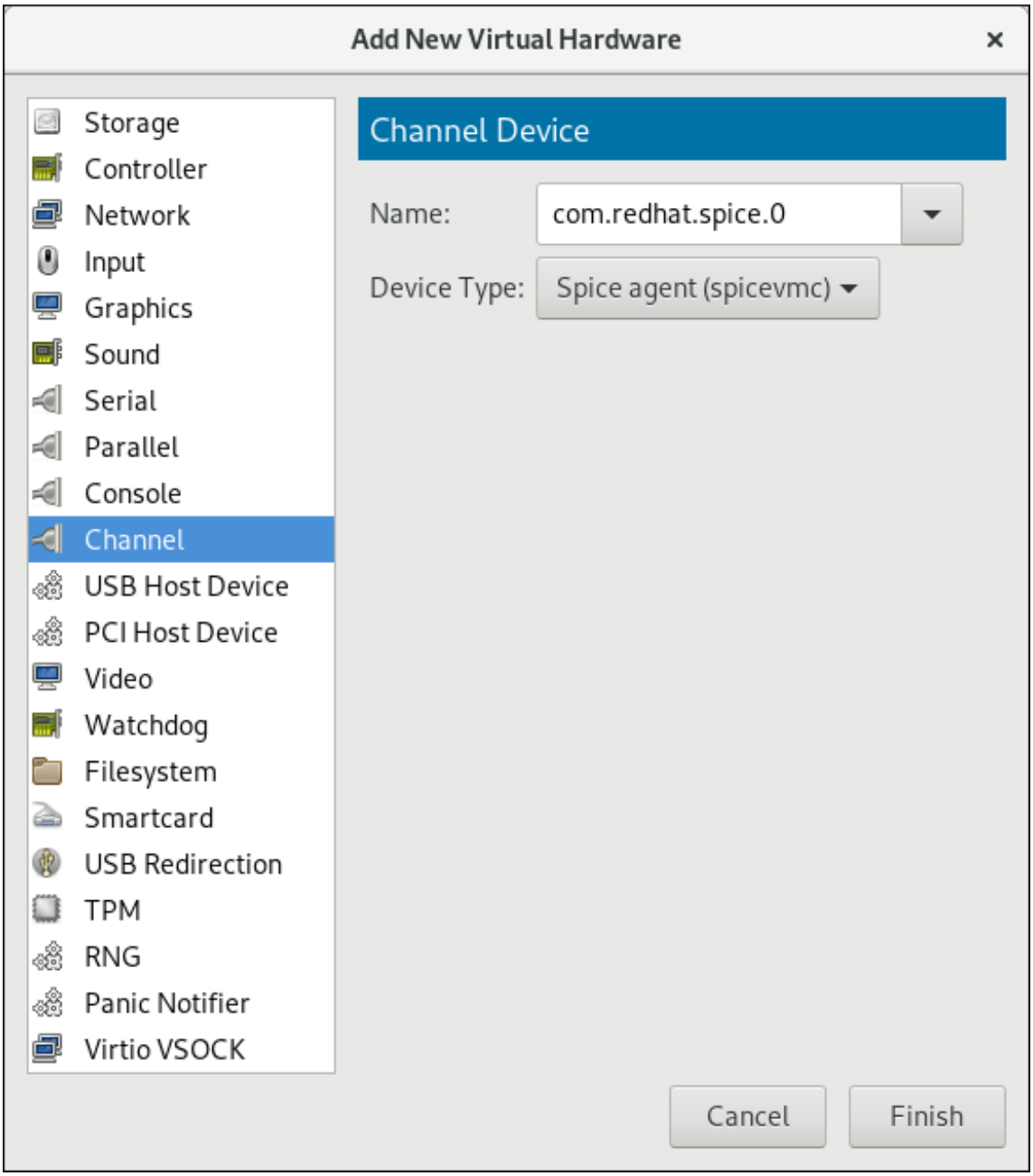
2. **SPICE エージェントチャネルをゲストに追加します。**

ゲストウィンドウ上部の電球アイコンをクリックして、仮想マシンのハードウェアの詳細を開きます。

Add Hardware ボタンをクリックして **Add New Virtual Hardware** ウィンドウを開き、**Channel** を選択します。

Name ドロップダウンリストから SPICE エージェントを選択し、チャネルアドレスを変更して、**Finish** をクリックします。

図11.2 SPICE エージェントのチャネルデバイスの選択



3. **仮想マシンの起動**
   仮想マシンを起動するには、Virtual Machine Manager の仮想マシンリストから仮想マシンを選択し、メニューバーの  をクリックします。

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

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

5. **ゲストで SPICE エージェントを起動します。**
   ゲストで SPICE エージェントサービスを起動します。

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

これで、*rhel7* 仮想マシンに SPICE エージェントが設定されます。

第12章 NESTED VIRTUALIZATION

12.1. 概要

Red Hat Enterprise Linux 7.5 以降、*nested virtualization* は KVM ゲスト仮想マシンのテクノロジープレビューとして利用できます。この機能では、物理ホスト (**レベル 0** または **L0**) で実行しているゲスト仮想マシン (**レベル 1** または **L1**) がハイパーバイザーとして機能し、独自のゲスト仮想マシン (**L2**) を作成できます。

ネストされた仮想化は、制約のある環境でのハイパーバイザーのデバッグや、限られた量の物理リソースでの大規模な仮想デプロイメントのテストなど、さまざまなシナリオで役立ちます。ただし、ネストされた仮想化は、実稼働ユーザー環境では対応しておらず、推奨されていないため、主に開発とテストを目的としています。

ネストされた仮想化は、ホストの仮想化拡張機能に依存して機能します。また、Red Hat Enterprise Linux ではサポートされていない QEMU Tiny Code Generator (TCG) エミュレーションを使用して仮想環境でゲストを実行することと混同しないでください。

12.2. 設定

次の手順に従って、ネストされた仮想化を有効にし、設定し、起動します。

1. **有効**: この機能はデフォルトで無効になっています。これを有効にするには、L0 ホストの物理マシンで次の手順を使用します。

   Intel の場合:

   a. ネストされた仮想化がホストシステムで利用可能であるかどうかを確認します。

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

   このコマンドが **Y** または **1** を返すと、機能が有効になります。

   コマンドが **0** または **N** を返す場合は、手順 **ii** および **iii** を使用します。

   b. **kvm_intel** モジュールをアンロードします。

   ```
   # modprobe -r kvm_intel
   ```

   c. ネスト機能をアクティブにします。

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

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

   ```
   options kvm_intel nested=1
   ```

   AMD の場合:

   a. ネストされた仮想化がシステムで利用可能であるかどうかを確認します。

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

このコマンドが **Y** または **1** を返すと、機能が有効になります。

コマンドが **0** または **N** を返す場合は、手順 **ii** および **iii** を使用します。

b. **kvm_amd** モジュールをアンロードします。

```
# modprobe -r kvm_amd
```

c. ネスト機能をアクティブにします。

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

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

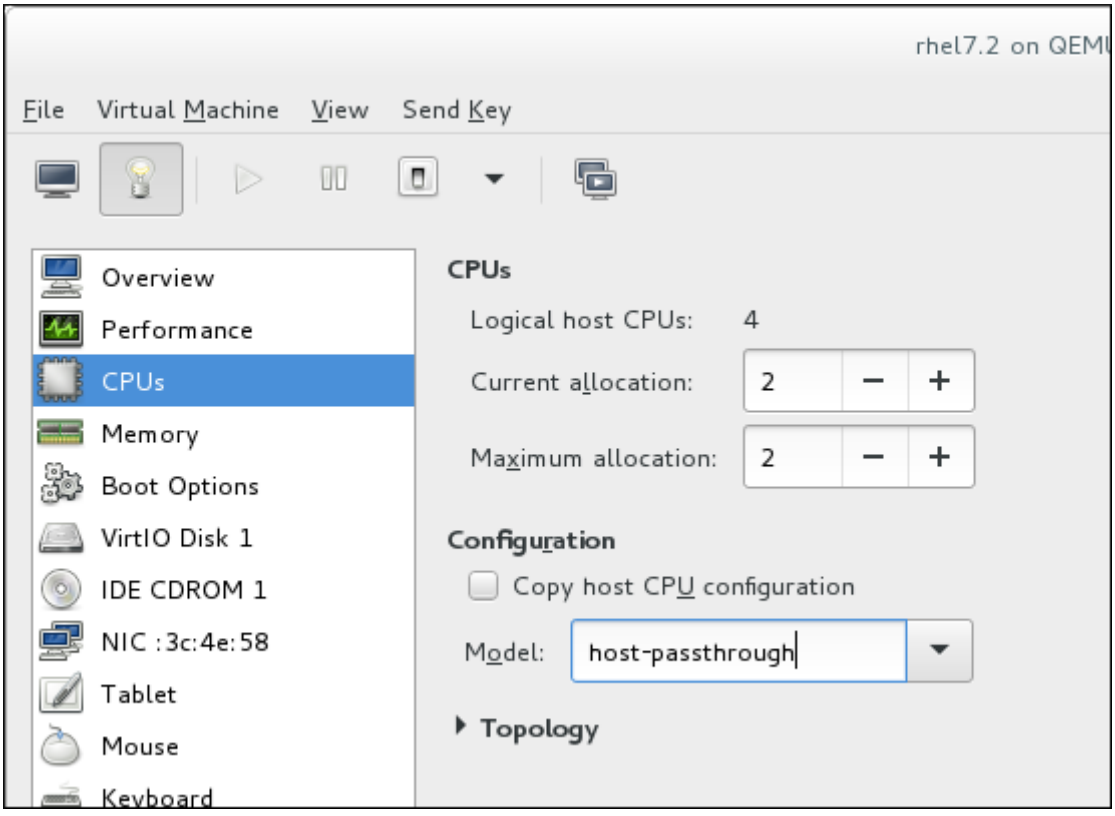
```
options kvm_amd nested=1
```

2. 以下のいずれかの方法で、ネストした仮想化用に L1 仮想マシンを**設定**します。

virt-manager

a. 目的のゲストの GUI を開き、**Show Virtual Hardware Details** アイコンをクリックします。

b. **Processor** メニューを選択し、**Configuration** セクションで、**Model** フィールドに **host-passthrough** と入力し (ドロップダウン選択を使用しません)、**Apply** をクリックします。



[D]

ドメイン XML

ゲストのドメイン XML ファイルに、次の行を追加します。

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

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

3. ネストされた仮想化の**使用を開始する**するには、L1 ゲスト内に L2 ゲストをインストールします。これには、L1 ゲストのインストールと同じ手順に従います。詳細は 3章*仮想マシンの作成* を参照してください。

12.3. 制限事項

- L0 ホストおよび L1 ゲストで Red Hat Enterprise Linux 7.2 以降を実行することが強く推奨されます。L2 ゲストには、Red Hat がサポートするゲストシステムを含めることができます。
- L1 または L2 ゲストの移行はサポートされていません。
- ハイパーバイザーとしての L2 ゲストの使用および L3 ゲストの作成はサポートされていません。
- ホストで利用可能な機能がすべて L1 ハイパーバイザーで使用できる訳ではありません。たとえば、IOMMU/VT-d や APICv は、L1 ハイパーバイザーでは使用できません。
- ネストされた仮想化を使用するには、ホストの CPU に必要な機能フラグが必要です。L0 ハイパーバイザーおよび L1 ハイパーバイザーが正しく設定されているかを確認するには、L0 および L1 の両方で **cat /proc/cpuinfo** コマンドを使用し、両方のハイパーバイザーの各 CPU に次のフラグが記載されていることを確認します。
  - Intel の場合 - **-vmx** (ハードウェア仮想化) および**ept** (拡張ページテーブル)
  - AMD の場合 - **svm** (vmx と同等) および **npt** (ept と同等)

パート II. 管理

このパートでは、仮想マシンの管理に関連するトピックを取り上げ、仮想ネットワーク、ストレージ、PCI 割り当てなどの仮想化機能がどのように機能するかを説明します。また、このパートでは、**qemu-img** ツール、**virt-manager** ツール、および **virsh** ツールを使用したデバイスおよびゲストの仮想マシンの管理方法も説明します。

第13章 仮想マシン用のストレージの管理

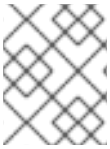
本章では、仮想マシンのストレージについて説明します。仮想ストレージは、仮想マシンの接続に割り当てられた物理ストレージから抽象化されます。ストレージは、準仮想化またはエミュレートのブロックデバイスドライバーを使用して仮想マシンに接続されています。

13.1. ストレージの概念

storage pool は、ゲスト仮想マシンが使用するために確保するストレージの量です。ストレージプールは、*ストレージボリューム* に分類されます。各ストレージボリュームは、ゲストバスの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。

ストレージプールとボリュームは、libvirt を使用して管理されます。libvirt のリモートプロトコルを使用すると、ゲスト仮想マシンのライフサイクルのあらゆる側面、およびゲスト仮想マシンに必要なリソースの設定を管理できます。この操作は、リモートホストで実行できます。そして、libvirt を使用した管理アプリケーション (**Virtual Machine Manager** など) は、ゲスト仮想マシンにホスト物理マシンを設定するために必要なすべてのタスクを実行できるようにします。これには、シェルアクセスやその他の制御チャネルを必要とせずに、リソースの割り当て、ゲスト仮想マシンの実行、シャットダウン、リソースの割り当て解除などが含まれます。

libvirt の API を使用すると、ストレージプールのボリューム一覧を照会したり、ストレージプールの容量、割り当て、利用可能なストレージに関する情報を取得したりできます。ストレージプール内のストレージボリュームは、スパースボリュームで異なる可能性がある割り当てや容量などの情報を取得するためにクエリーされることがあります。



注記

スパースボリュームの詳細は、Virtualization Getting Started Guide を参照してください。

ストレージプールがこれに対応する場合は、libvirt API を使用してストレージボリュームを作成、クローン、サイズ変更、および削除できます。API は、ストレージボリュームへのデータのアップロード、ストレージボリュームからのデータのダウンロード、またはストレージボリュームからのデータの消去にも使用できます。

ストレージプールを起動すると、ドメイン XML 内のボリュームへのホストパスの代わりに、ストレージプール名とストレージボリューム名を使用して、ストレージボリュームをゲストに割り当てることができます。

注記

ドメイン XML の詳細は、*23章 ドメインXML の操作* を参照してください。

ストレージプールは停止 (破棄) できます。これによりデータの抽象化は削除されますが、データは保持されます。

たとえば、**mount -t nfs nfs.example.com:/path/to/share /path/to/data** を使用する NFS サーバーです。ストレージ管理者は、仮想化ホストで NFS ストレージプールを定義して、エクスポートしたサーバーパスおよびクライアントターゲットパスを記述できます。これにより、libvirt が起動した場合に自動的に、または必要に応じて、libvirt の実行中に libvirt がマウントを実行できるようになります。NFS サーバーがエクスポートしたディレクトリーが含まれるファイルは、NFS ストレージプールのストレージボリュームとしてリスト表示されます。

ストレージボリュームをゲストに追加する際、管理者がターゲットパスをボリュームに追加する必要はありません。名前でストレージプールとストレージボリュームを追加する必要があります。したがって、ターゲットクライアントのパスが変更しても、仮想マシンには影響を及ぼしません。

ストレージプールが開始すると、libvirt は、システム管理者がログインして **mount nfs.example.com:/path/to/share /vmdata** を実行した場合と同じように、指定したディレクトリーに共有をマウントします。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は、libvirt が起動したときに、指定したディレクトリーに NFS 共有ディスクがマウントされていることが libvirt により確認されます。

ストレージプールが起動すると、NFS 共有ディスク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 を使用してクエリーすることができます。ストレージボリュームのパスは、ゲスト仮想マシンの XML 定義のセクションにコピーできます。このセクションは、ゲスト仮想マシンのブロックデバイスのソースストレージを記述します。NFS の場合、libvirt API を使用するアプリケーションは、ストレージプール (NFS 共有内のファイル) にあるストレージボリュームを、プールのサイズ (共有のストレージ容量) の上限まで作成および削除できます。

ボリュームの作成および削除は、すべてのストレージプールタイプで対応しているわけではありません。ストレージプールを停止する (pool-destroy) と、開始操作を取り消します。この場合は、NFS 共有のマウントを解除します。コマンドの名前が記載されているにも関わらず、共有上のデータは destroy 操作で修正されません。詳細は、**man virsh** を参照してください。

手順13.1 ストレージの作成と割り当て

この手順により、仮想マシンゲストのストレージを作成および割り当てるために必要な手順の概要がわかります。

1. **ストレージプールの作成**
   利用可能なストレージメディアから、1 つ以上のストレージプールを作成します。詳細は、「ストレージプールの使用」 を参照してください。

2. **ストレージボリュームの作成**
   利用可能なストレージプールから、ストレージボリュームを 1 つ以上作成します。詳細は、「ストレージボリュームの使用」 を参照してください。

3. **仮想マシンへのストレージデバイスの割り当て**
   ストレージボリュームから抽象化されたストレージデバイスを、ゲスト仮想マシンに割り当てます。詳細は、「ゲストへのストレージデバイスの追加」 を参照してください。

13.2. ストレージプールの使用

本セクションでは、仮想マシンでストレージプールを使用する方法を説明します。概念的な情報 に加えて、**virsh** コマンドと **Virtual Machine Manager** を使用してストレージプールを 作成、設定、および削除 するための詳細な手順を提供します。

13.2.1. ストレージプールの概念

ストレージプールは、仮想マシンにストレージを提供するために、libvirt が管理するファイル、ディレクトリー、またはストレージデバイスです。ストレージプールは、仮想マシンイメージを保存するストレージボリュームに分割されるか、追加のストレージとして仮想マシンに割り当てられます。複数のゲストで同じストレージプールを共有できるため、ストレージリソースの割り当てを改善できます。

ストレージプールは、ローカルまたはネットワークベース (共有) にできます。

ローカルストレージのプール

ローカルストレージプールは、ホストサーバーに直接割り当てることができます。これには、ローカルデバイスのローカルディレクトリー、直接接続したディスク、物理パーティション、および論理ボリューム管理 (LVM) ボリュームグループが含まれます。ローカルストレージプールは、移行や大量の仮想マシンを必要としない開発、テスト、および小規模なデプロイメントに役立ちます。ローカルストレージプールは、ライブマイグレーションに使用できないため、多くの実稼働環境には適していない場合があります。

ネットワーク (共有) ストレージプール

ネットワークストレージプールには、標準プロトコルを使用してネットワーク経由で共有されるストレージデバイスが含まれます。**virt-manager** を使用してホスト間で仮想マシンを移行する場合はネットワークストレージが必要ですが、**virsh** を使用して移行する場合は任意です。

仮想マシンの移行の詳細は、15章*KVM の移行* を参照してください。

以下は、Red Hat Enterprise Linux が対応しているストレージプールのタイプのリストです。

- ディレクトリーベースのストレージプール
- ディスクベースのストレージプール
- パーティションベースのストレージプール
- GlusterFS ストレージプール
- iSCSI ベースのストレージプール
- LVM ベースのストレージプール
- NFS ベースのストレージプール
- SCSI デバイスを使用した vHBA ベースのストレージプール

以下は、Red Hat Enterprise Linux で対応していない**libvirt**ストレージプールタイプのリストです。

- マルチパスベースのストレージプール
- RBD ベースのストレージプール
- sheepdog ベースのストレージプール
- vstorage ベースのストレージプール
- ZFS ベースのストレージプール

注記

対応していないストレージプールタイプの一部は、**Virtual Machine Manager**インターフェイスに表示されます。ただし、これを使用することはできません。

13.2.2. ストレージプールの作成

本セクションでは、**virsh** および **Virtual Machine Manager** を使用してストレージプールを作成する方法を説明します。**virsh** を使用すると、すべてのパラメーターを指定できますが、**Virtual Machine Manager** を使用すると、より単純なストレージプールを作成するためのグラフィックメソッドが提供されます。

13.2.2.1. virsh を使用したストレージプールの作成

注記

本セクションでは、例としてパーティションベースのストレージプールの作成を示します。

手順13.2 virsh を使用したストレージプールの作成

1. **推奨事項を読み、前提条件がすべて満たされていることを確認します。**
   ストレージプールによっては、このガイドでは特定の方法に従うことが推奨されます。また、一部のタイプのストレージプールには前提条件があります。推奨事項および前提条件 (存在する場合) を表示する場合は、「ストレージプール固有のもの」 を参照してください。

2. **ストレージプールの定義**
   ストレージプールは永続的または一時的なものにできます。永続ストレージプールは、ホストマシンのシステムを再起動しても維持します。一時的なストレージプールは、ホストが再起動すると削除されます。

   次のいずれかを行います。

   - XML ファイルを使用してストレージプールを定義します。

     a.新規デバイスに必要なストレージプール情報を含む一時的な XML ファイルを作成します。

     XML ファイルには、ストレージプールのタイプに基づいた特定のフィールドが含まれている必要があります。詳細は、「ストレージプール固有のもの」 を参照してください。

     以下は、ストレージプール定義の XML ファイルの例を示しています。この例では、ファイルは ~/**guest_images.xml** に保存されています。

     ```
     <pool type='fs'>
       <name>guest_images_fs</name>
       <source>
         <device path='/dev/sdc1'/>
       </source>
       <target>
         <path>/guest_images</path>
       </target>
     </pool>
     ```

     b.**virsh pool-define** コマンドを使用して、永続ストレージプールを作成するか、**virsh pool-create** コマンドを使用して一時ストレージプールを作成して起動します。

     ```
     # virsh pool-define ~/guest_images.xml
     Pool defined from guest_images_fs
     ```

     または

     ```
     # virsh pool-create ~/guest_images.xml
     Pool created from guest_images_fs
     ```

     c.手順 a で作成した XML ファイルを削除します。

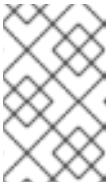
- **virsh pool-define-as** コマンドを使用して、永続ストレージプールを作成するか、**virsh pool-create-as** コマンドを使用して一時的なストレージプールを作成します。

  以下の例では、永続化した後、/**guest_images** ディレクトリーから /**dev**/**sdc1** にマッピングされた一時的なファイルシステムベースのストレージプールを作成します。

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

  または

  ```
  # virsh pool-create-as guest_images_fs fs - - /dev/sdc1 - "/guest_images"
  Pool guest_images_fs created
  ```

  注記

  virsh インターフェイスを使用する場合、コマンド内のオプション名は任意です。オプション名を使用しない場合は、指定する必要のないフィールドにハイフンを使用します。

3. **プールが作成されたことを確認します。**
   virsh pool-list --allを使用して、存在するすべてのストレージプールのリストを表示します。

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

4. **ストレージプールのターゲットパスの定義**
   virsh pool-build コマンドを使用して、フォーマット済みファイルシステムのストレージプール用のストレージプールターゲットパスを作成し、ストレージソースデバイスを初期化して、データのフォーマットを定義します。次に、**virsh pool-list** を使用して、ストレージプールがリスト表示されていることを確認します。

   ```
   # 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
   ```

   注記

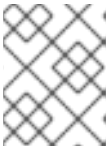
   ターゲットパスの構築は、ディスクベース、ファイルシステムベース、論理ストレージプールにのみ必要です。libvirt は、**overwrite** オプションが指定されている場合を除き、ソースストレージデバイスのデータフォーマットが、選択したストレージプールタイプと異なることを検出すると、ビルドに失敗します。

5. **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、ソースデバイスを使用できるように準備します。

実行されるアクションは、ストレージプールのタイプによって異なります。たとえば、ファイルシステムベースのストレージプールの場合、**virsh pool-start** コマンドはファイルシステムをマウントします。LVM ベースのストレージプールの場合、**virsh pool-start** コマンドは、**vgchange** コマンドを使用してボリュームグループをアクティブにします。

次に、**virsh pool-list command** を使用して、ストレージプールがアクティブであることを確認します。

```
# 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
```



注記

virsh pool-start コマンドは、永続ストレージプールにのみ必要です。一時的なストレージプールは、作成時に自動的に起動します。

6. **自動起動を有効にする (オプション)**
デフォルトでは、**virsh** で定義されたストレージプールは、**libvirtd** が起動するたびに自動的に起動するように設定されていません。**virsh pool-autostart** を使用すると、ストレージプールを自動的に起動するように設定できます。

```
# 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
```

ストレージプールは、**libvirtd** が起動するたびに自動的に起動するようになりました。

7. **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、報告されたサイズが期待どおりで、状態が **running** と報告されたことを確認します。ファイルシステムのターゲットパスに lost+found ディレクトリーがあることを確認します。これは、デバイスがマウントされていることを示しています。

```
# virsh pool-info guest_images_fs
Name:           guest_images_fs
UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       458.39 GB
Allocation:     197.91 MB
Available:      458.20 GB
```

```
# mount | grep /guest_images
/dev/sdc1 on /guest_images type ext4 (rw)
# ls -la /guest_images
total 24
drwxr-xr-x.  3 root root  4096 May 31 19:47 .
dr-xr-xr-x. 25 root root  4096 May 31 19:38 ..
drwx------.  2 root root 16384 May 31 14:18 lost+found
```

13.2.2.2. Virtual Machine Manager を使用したストレージプールの作成

注記

このセクションでは、例としてディスクベースのストレージプールの作成について説明します。

手順13.3 Virtual Machine Manager を使用したストレージプールの作成

1. **ストレージプールを作成するメディアを準備します。**
   これは、ストレージプールのタイプにより異なります。詳細は 「ストレージプール固有のもの」 を参照してください。

   この例では、*GUID Partition Table* でディスクに再ラベル付けする必要がある場合があります。

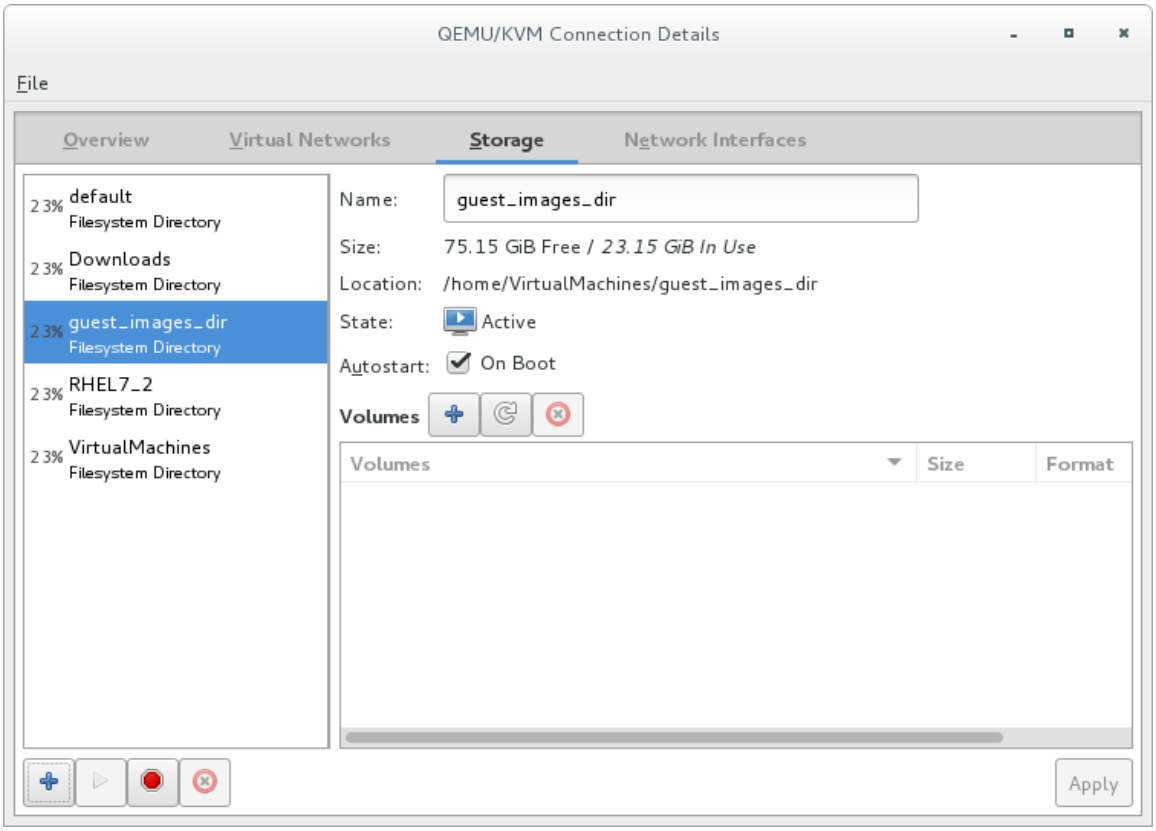
2. **ストレージ設定を開く**

   a. **Virtual Machine Manager** で、設定するホスト接続を選択します。

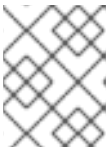
   Edit メニューを開き、**Connection Details** を選択します。

   b. **Connection Details** ウィンドウの **Storage** タブをクリックします。

   図13.1 ストレージタブ



3. **新しいストレージプールを作成します。**



注記

Virtual Machine Manager を使用して、永続ストレージプールのみを作成できます。一時ストレージプールは、**virsh** を使用してのみ作成できます。

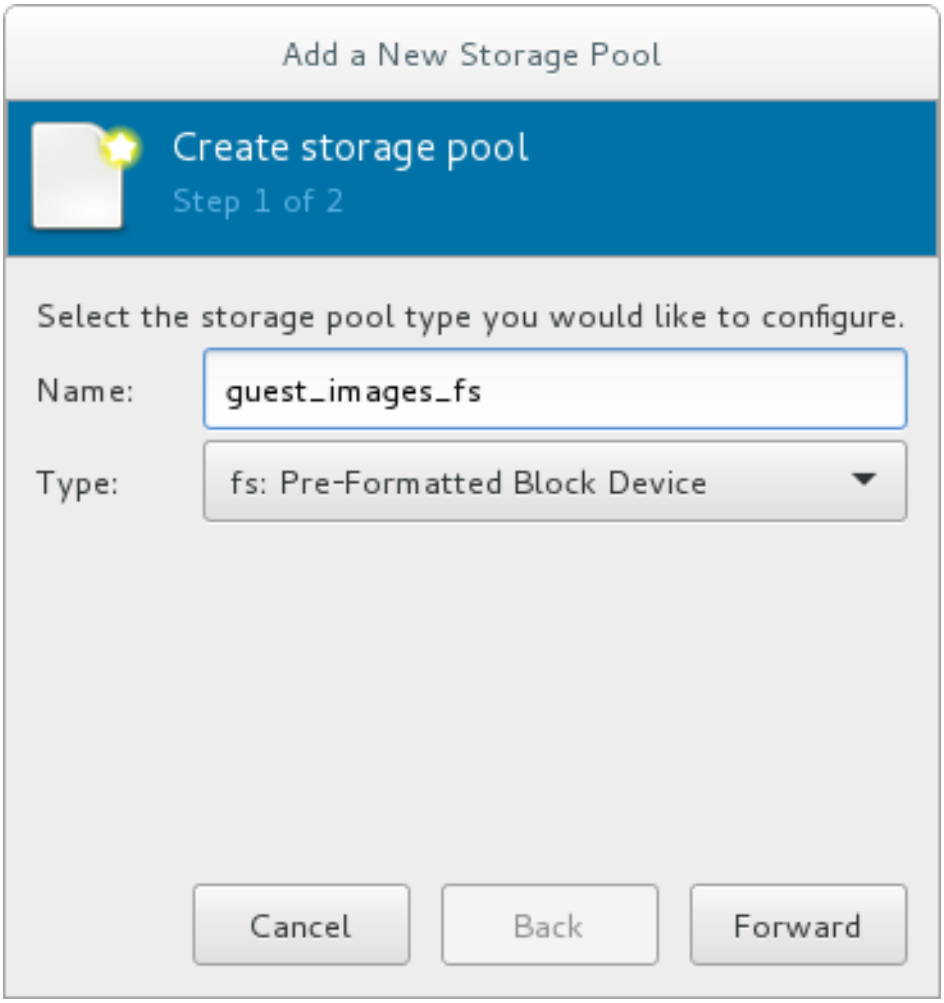
a. **新しいストレージプールの追加 (パート 1)**

ウィンドウんの下部の  ボタンをクリックします。**Add a New Storage Pool** ウィザードが表示されます。

ストレージプールの **Name** を入力します。この例では、*guest_images_fs* という名前を使用します。

Type ドロップダウンリストから、作成するストレージプールの種類を選択します。この例では、**fs: Pre-Formatted Block Device** を使用しています。

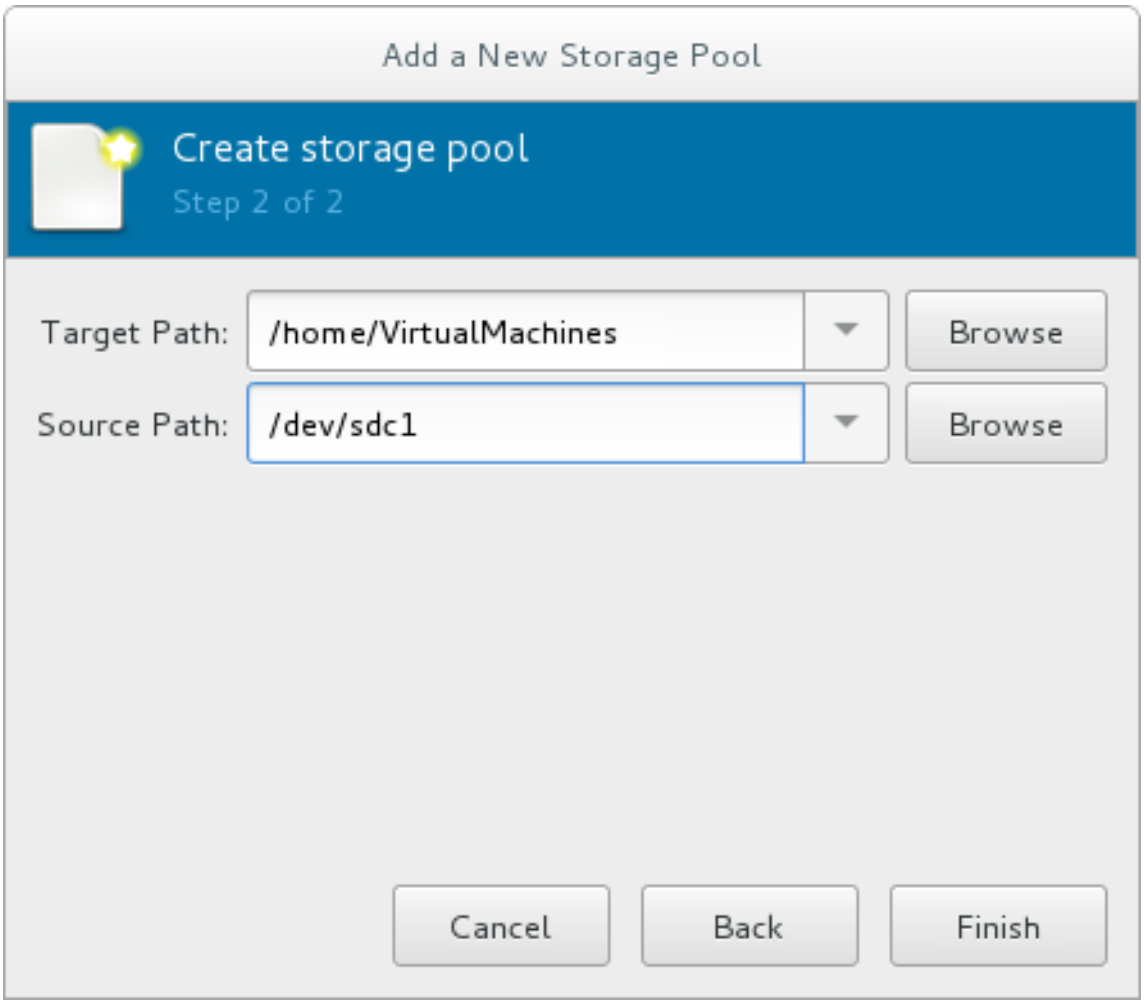
図13.2 ストレージプールの名前とタイプ



Forward ボタンを選択して続行します。

b. **新しいプールの追加 (パート 2)**

図13.3 ストレージプールのパス



関連するパラメーターを使用してストレージプールを設定します。各タイプのストレージプールのパラメーターの詳細は、「ストレージプール固有のもの」 を参照してください。

ストレージプールの種類によっては、**Build Pool** チェックボックスが表示されます。ストレージからストレージプールを構築する場合は、**Build Pool** チェックボックスをチェックします。

詳細を確認し、**Finish** をクリックしてストレージプールを作成します。

13.2.3. ストレージプール固有のもの

このセクションでは、前提条件、パラメーター、および追加情報を含む、各タイプのストレージプールに固有の情報を説明します。これには、以下のトピックが含まれます。

- 「ディレクトリーベースのストレージプール」
- 「ディスクベースのストレージプール」
- 「ファイルシステムベースのストレージプール」
- 「GlusterFS ベースのストレージプール」
- 「iSCSI ベースのストレージプール」
- 「LVM ベースのストレージプール」

- 「NFS ベースのストレージプール」
- 「SCSI デバイスを使用する vHBA ベースのストレージプール」

13.2.3.1. ディレクトリーベースのストレージプール

パラメーター
以下の表は、ディレクトリーベースのストレージプールを作成する場合に必要な、XML ファイル、**virsh pool-define-as** コマンド、および **Virtual Machine Manager** アプリケーションのパラメーターのリストを示しています。

表13.1 ディレクトリーベースのストレージプールのパラメーター

説明	XML	pool-define-as	Virtual Machine Manager
ストレージプールのタイプ	<pool type='dir'>	[type] directory	dir: ファイルシステムディレクトリー
ストレージプールの名前	<name>*name*</name>	[name] *name*	Name
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<target> <path>*target_path*</path> </target>	target *path_to_pool*	ターゲットパス

virsh を使用してストレージプールを作成する場合は、 プールが作成されたことの確認して 続行します。

例
以下は、/**guest_images** ディレクトリーに基づいたストレージプールの XML ファイルの例です。

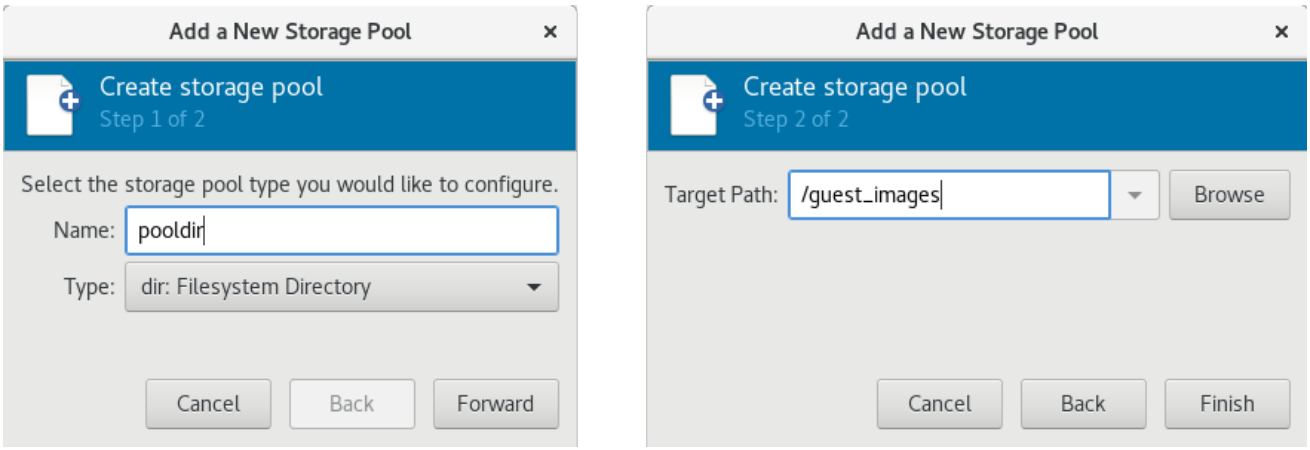
```
<pool type='dir'>
  <name>dirpool</name>
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

以下は、/**guest_images** ディレクトリーに基づいてストレージプールを作成するコマンドの例になります。

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

以下の図は、/**guest_images** ディレクトリーに基づいてストレージプールを作成するための、**Virtual Machine Manager** の Add a New Storage Pool ダイアログボックスの例を示しています。

図13.4 新しいディレクトリーベースのストレージプールの例を追加します。



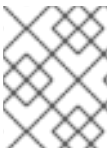
13.2.3.2. ディスクベースのストレージプール

推奨事項
ディスクベースのストレージプールを作成する前に、以下の点に注意してください。

- 使用されている libvirt のバージョンに応じて、ディスクをストレージプール専用にすると、現在ディスクデバイスに格納されているすべてのデータが再フォーマットされて消去される可能性があります。ストレージプールを作成する前に、ストレージデバイスのデータのバックアップを作成することを強く推奨します。
- ゲストには、ディスク全体またはブロックデバイス (**/dev/sdb** など) への書き込みアクセス権を付与しないでください。パーティション (**/dev/sdb1** など) または LVM ボリュームを使用します。

  ブロックデバイス全体をゲストに渡すと、ゲストはブロックデバイスをパーティションに分割するか、ブロックデバイスに独自の LVM グループを作成します。これにより、ホストの物理マシンがこのようなパーティションや LVM グループを検出し、エラーが発生する場合があります。

前提条件



注記

本セクションの手順は、**virsh pool-build** コマンドを実行しない場合に限り必要になります。

ホストディスクにディスクベースのストレージプールを作成する前に、ディスクに、*GUID Partition Table* (GPT) ディスクラベルを付けて再ラベル付けする必要があります。GPT ディスクラベルは、各デバイスに最大 128 個のパーティションを作成できます。

```
# 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.
#
```

ディスクに再ラベル付けしたら、ストレージプールの定義 でストレージプールの作成を続行します。

パラメーター
以下の表は、ディスクベースのストレージプールを作成する場合に必要な、XML ファイル、**virsh pool-define-as** コマンド、および **Virtual Machine Manager** アプリケーションのパラメーターのリストを示しています。

表13.2 ディスクベースのストレージプールのパラメーター

説明	XML	pool-define-as	Virtual Machine Manager
ストレージプールのタイプ	<pool type='disk'>	[type] disk	disk: 物理ディスクデバイス
ストレージプールの名前	<name>*name*</name>	[name] *name*	Name
ストレージデバイスを指定するパス。たとえば、/dev/sdb	<source> <device path=/dev/*sdb*/> <source>	source-dev *path_to_disk*	ソースパス
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<target> <path>/*path_to_pool*</path> </target>	target *path_to_pool*	ターゲットパス

virsh を使用してストレージプールを作成する場合は、ストレージプールの定義 に進みます。

例
以下は、ディスクに基づいたストレージプールに対する XML ファイルの例です。

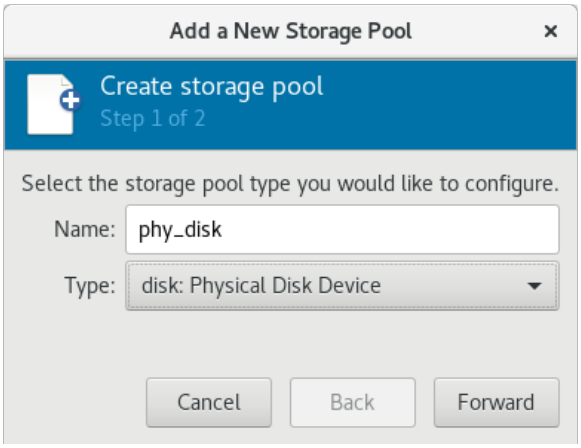
```
<pool type='disk'>
  <name>phy_disk</name>
  <source>
    <device path='/dev/sdb'/>
    <format type='gpt'/>
  </source>
  <target>
    <path>/dev</path>
  </target>
</pool>
```

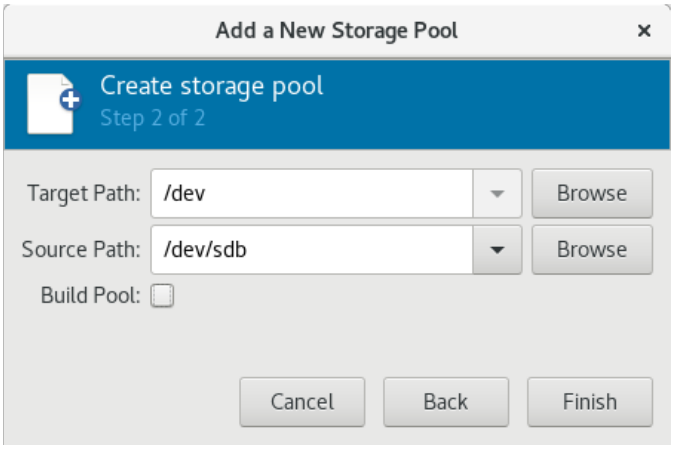
以下は、ディスクベースのストレージプールを作成するコマンドの例です。

```
# virsh pool-define-as phy_disk disk --source-format=gpt --source-dev=/dev/sdb --target /dev
Pool phy_disk defined
```

以下の図は、ディスクベースのストレージプールを作成するための、仮想マシン XML 設定 **Virtual Machine Manager** の Add a New Storage Pool ダイアログボックスの例を示しています。

図13.5 新しいディスクベースのストレージプールの例を追加します。

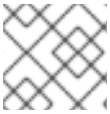




13.2.3.3. ファイルシステムベースのストレージプール

推奨事項
このセクションの手順に従って、ディスク全体 (**/dev/sdb** など) をストレージプールとして割り当てないでください。ゲストには、ディスク全体またはブロックデバイスへの書き込みアクセス権を付与しないでください。この方法は、パーティション (**/dev/sdb1** など) をストレージプールに割り当てる場合に限り使用してください。

前提条件



注記

これは、**virsh pool-build** コマンドを実行しない場合にのみ必要になります。

パーティションからストレージプールを作成するには、ファイルシステムを ext4 にフォーマットします。

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

ファイルシステムをフォーマットしたら、ストレージプールの定義 でストレージプールの作成を続行します。

パラメーター
以下の表は、パーティションからファイルシステムベースのストレージプールを作成する場合に必要な、XML ファイル、**virsh pool-define-as** コマンド、および **Virtual Machine Manager** アプリケーションのパラメーターのリストを示しています。

表13.3 ファイルシステムベースのストレージプールパラメーター

説明	XML	pool-define-as	Virtual Machine Manager
ストレージプールのタイプ	<pool type='fs'>	[type] fs	fs: フォーマット済みブロックデバイス

説明	XML	pool-define-as	Virtual Machine Manager
ストレージプールの名前	<name>*name*</name>	[name] *name*	Name
パーミッションを指定するパス。たとえば、/dev/sdc1	<source> <device path='*source_path*' />	[source] *path_to_partition*	ソースパス
ファイルシステムのタイプ (ext4 など)。	<format type='*fs_type*' /> </source>	[ソースフォーマット]*FS-format*	該当なし
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<target> <path>*/path_to_pool*</path> </target>	[target] *path_to_pool*	ターゲットパス

virsh を使用してストレージプールを作成する場合は、ストレージプールが作成されたことの確認 に進みます。

例
次に示すのは、ファイルシステムベースのストレージプールの XML ファイルの例です。

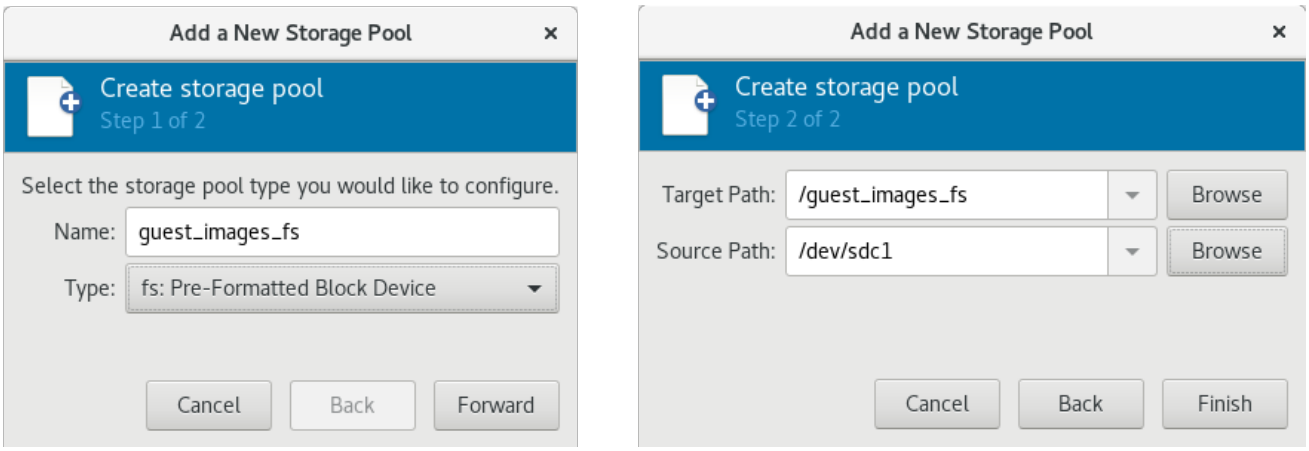
```
<pool type='fs'>
  <name>guest_images_fs</name>
  <source>
    <device path='/dev/sdc1'/>
    <format type='auto'/>
  </source>
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

以下は、パーティションベースのストレージプールを作成するコマンドの例です。

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

以下の図は、ファイルシステムベースのストレージプールを作成するための、仮想マシン XML 設定**Virtual Machine Manager**の Add a New Storage Pool ダイアログボックスの例を示しています。

図13.6 新しいファイルシステムベースのストレージプールの例を追加します。



13.2.3.4. GlusterFS ベースのストレージプール

推奨事項
GlusterFS は、ユーザー空間のファイルシステム (FUSE) を使用するユーザー空間のファイルシステムです。

前提条件
GlusterFS ベースのストレージプールをホストに作成する前に、Gluster サーバーを準備する必要があります。

手順13.4 Gluster サーバーの準備

1. 次のコマンドを使用してそのステータスをリスト表示して、Gluster サーバーの IP アドレスを取得します。

```
# gluster volume status
Status of volume: gluster-vol1
Gluster process        Port Online Pid
------------------------------------------------------------------------------
Brick 222.111.222.111:/gluster-vol1    49155 Y 18634

Task Status of Volume gluster-vol1
------------------------------------------------------------------------------
There are no active volume tasks
```

2. glusterfs-fuse パッケージがインストールされていない場合はインストールします。

3. virt_use_fusefs ブール値が有効になっていない場合は有効にします。有効になっていることを確認します。

```
# setsebool virt_use_fusefs on
# getsebool virt_use_fusefs
virt_use_fusefs --> on
```

必要なパッケージがインストールされ、有効になったことを確認したら、ストレージプールの作成を続行して、ストレージプールの定義 でストレージプールを作成します。

パラメーター

以下の表は、GlusterFS ベースのストレージプールを作成する場合に必要な、XML ファイル、**virsh pool-define-as** コマンド、および **Virtual Machine Manager** アプリケーションのパラメーターのリストを示しています。

表13.4 GlusterFS ベースのストレージプールパラメーター

説明	XML	pool-define-as	Virtual Machine Manager
ストレージプールのタイプ	<pool type='gluster'>	[type] gluster	Gluster: Gluster ファイルシステム
ストレージプールの名前	<name>*name*</name>	[name] *name*	Name
Gluster サーバーのホスト名または IP アドレス	<source> <hostname='*hostname*' />	source-host *hostname*	ホスト名
Gluster サーバーの名前	<name='*Gluster-name*' />	source-name *Gluster-name*	ソース名
ストレージプールに使用される Gluster サーバーのパス。	<dir path='*Gluster-path*' /> </source>	source-path *Gluster-path*	ソースパス

virsh を使用してストレージプールを作成する場合は、ストレージプールが作成されたことの確認 に進みます。

例
以下は、GlusterFS ベースのストレージプールの XML ファイルの例になります。

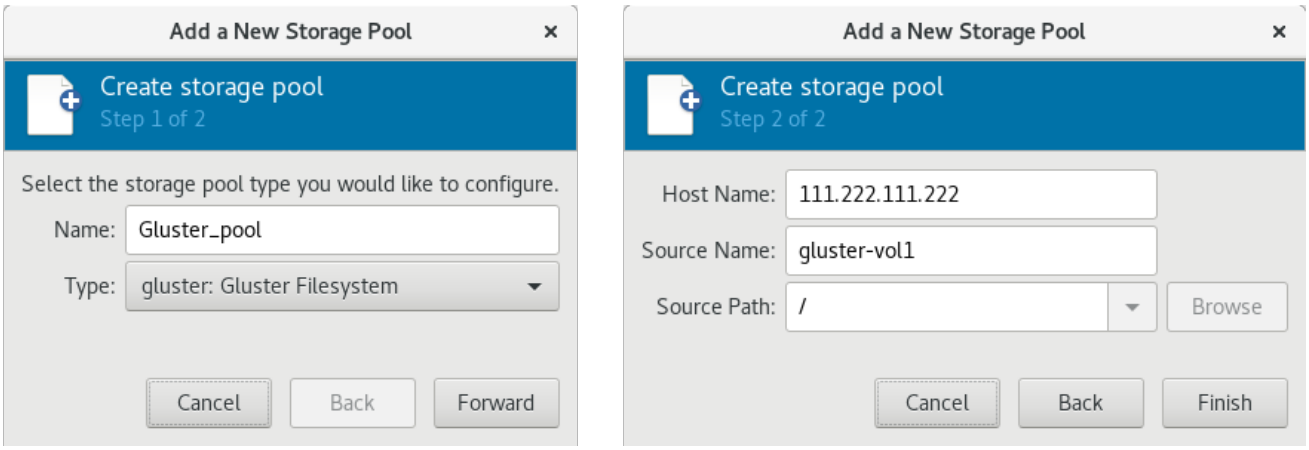
```
<pool type='gluster'>
  <name>Gluster_pool</name>
  <source>
    <host name='111.222.111.222'/>
    <dir path='/'/>
    <name>gluster-vol1</name>
  </source>
</pool>
```

以下は、GlusterFS ベースのストレージプールを作成するコマンドの例になります。

```
# pool-define-as --name Gluster_pool --type gluster --source-host 111.222.111.222 --source-name
gluster-vol1 --source-path /
Pool Gluster_pool defined
```

以下の図は、GlusterFS ベースのストレージプールを作成するための、仮想マシン XML 設定 **Virtual Machine Manager**Add a New Storage Pool ダイアログボックスの例を示しています。

図13.7 新しい GlusterFS ベースのストレージプールの例を追加します。



13.2.3.5. iSCSI ベースのストレージプール

推奨事項
iSCSI (Internet Small Computer System Interface) は、ストレージデバイスを共有するネットワークプロトコルです。iSCSI は、IP 層で SCSI 命令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。

iSCSI ベースのデバイスを使用してゲスト仮想マシンを保存することで、iSCSI をブロックストレージデバイスとして使用するなど、より柔軟なストレージオプションが可能になります。iSCSI デバイスは、Linux-IO (LIO) ターゲットを使用します。これは、Linux 用のマルチプロトコル SCSI ターゲットです。LIO は、iSCSI に加えて、FCoE (Fibre Channel and Fibre Channel over Ethernet) にも対応します。

前提条件
iSCSI ベースのストレージプールを作成する前に、iSCSI ターゲットを作成する必要があります。iSCSI ターゲットは、targetcli パッケージを使用して作成されます。これは、ソフトウェアで対応している iSCSI ターゲットを作成するためのコマンドセットを提供します。

手順13.5 iSCSI ターゲットの作成

1. **targetcli パッケージのインストール**

   ```
   # yum install targetcli
   ```

2. **targetcli コマンドセットの起動**

   ```
   # targetcli
   ```

3. **ストレージオブジェクトの作成**
   ストレージプールを使用して、3 つのストレージオブジェクトを作成します。

   a. ブロックストレージオブジェクトの作成

      i. **/backstores/block** ディレクトリーに移動します。

      ii. **create** コマンドを実行します。

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

以下に例を示します。

```
# create block1 dev=/dev/sdb1
```

b. fileio オブジェクトの作成

i. **/fileio** ディレクトリーに移動します。

ii. **create** コマンドを実行します。

```
# create [fileio-name][image-name] [image-size]
```

以下に例を示します。

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

c. ramdisk オブジェクトの作成

i. **/ramdisk** ディレクトリーに移動します。

ii. **create** コマンドを実行します。

```
# create [ramdisk-name] [ramdisk-size]
```

以下に例を示します。

```
# create ramdisk1 1M
```

d. この手順で作成したディスクの名前を書き留めておきます。これらは後で使用されます。

4. **iSCSI ターゲットの作成**

a. **/iscsi** ディレクトリーに移動します。

b. ターゲットの作成には、以下の 2 つの方法があります。

- パラメーターを指定せずに **create** コマンドを実行します。

  iSCSI qualified name (IQN) は自動的に生成されます。

- IQN とサーバーを指定して **create** コマンドを実行します。以下に例を示します。

```
# create iqn.2010-05.com.example.server1:iscsirhel7guest
```

5. **ポータル IP アドレスの定義**
iSCSI 経由でブロックストレージをエクスポートするには、最初にポータル、LUN、および *access control lists* ACL を設定する必要があります。

ポータルには、ターゲットが監視する IP アドレスと TCP、および接続するイニシエーターが含まれます。iSCSI はポート 3260 を使用します。このポートはデフォルトで設定されています。

ポート 3260 への接続:

a. **/tpg** ディレクトリーに移動します。

b. 以下のコマンドを実行します。

```
# portals/ create
```

このコマンドを実行すると、ポート 3260 をリッスンしている利用可能な IP アドレスがすべて使用できるようになります。

ポート 3260 をリッスンする IP アドレスを 1 つだけにする場合は、コマンドの末尾に IP アドレスを追加します。以下に例を示します。

```
# portals/ create 143.22.16.33
```

6. **LUN の設定と、ストレージオブジェクトのファブリックへの割り当て**
この手順では、ストレージオブジェクトの作成 で作成したストレージオブジェクトを使用します。

a. ポータルの IP アドレスの定義 で作成した TPG の **luns** ディレクトリーに移動します。以下に例を示します。

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

b. 最初の LUN を ramdisk に割り当てます。以下に例を示します。

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

c. 2 番目の LUN をブロックディスクに割り当てます。以下に例を示します。

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

d. 3 番目の LUN を fileio ディスクに割り当てます。以下に例を示します。

```
# create /backstores/fileio/fileio1
```

e. 作成される LUN のリストを表示します。

```
/iscsi/iqn.20...csirhel7guest 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]
```

7. **各イニシエーターの ACL の作成**

イニシエーターの接続時に認証を有効にします。指定した LUN を、指定したイニシエーターに制限することもできます。ターゲットとイニシエーターには一意の名前があります。iSCSI イニシエーターは IQN を使用します。

a. イニシエーター名を使用して、iSCSI イニシエーターの IQN を検索します。以下に例を示します。

```
# cat /etc/iscsi/initiator2.iscsi
InitiatorName=create iqn.2010-05.com.example.server1:iscsirhel7guest
```

この IQN は、ACL を作成するために使用されます。

b. **acls** ディレクトリーへ移動します。

c. 次のいずれかを実行して ACL を作成します。

- パラメーターを指定せずに **create** コマンドを実行して、すべての LUN およびイニシエーターに ACL を作成します。

```
# create
```

- 特定の LUN およびイニシエーターに ACL を作成し、iSCSI インティエーターの IQN を指定して **create** コマンドを実行します。以下に例を示します。

```
# create iqn.2010-05.com.example.server1:888
```

- すべてのイニシエーターに 1 つのユーザー ID と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ます。

```
# set auth userid=user_ID
# set auth password=password
# set attribute authentication=1
# set attribute generate_node_acls=1
```

この手順を完了したら、ストレージプールをセキュリティー保護 して続行します。

8. **設定を保存します。**
以前の起動設定を上書きして、設定を永続化します。

```
# saveconfig
```

9. **サービスを有効にします。**
次回の起動時に保存した設定を適用するには、サービスを有効にします。

```
# systemctl enable target.service
```

オプションの手順
iSCSI ベースのストレージプールを作成する前に、iSCSI ターゲットで実行できるオプションの手順が多数あります。

手順13.6 RAID アレイでの論理ボリュームの設定

1. **RAID5 アレイの作成**

RAID5 アレイの作成方法は、『Red Hat Enterprise Linux 7 Storage Administration Guide 』 を参照してください。

2. **RAID5 アレイに LVM 論理ボリュームを作成する**
RAID5 アレイで LVM 論理ボリュームを作成する方法は、『Red Hat Enterprise Linux 7 Logical Volume Manager Administration Guide』 を参照してください。

手順13.7 検出可能性のテスト

- **新しい iSCSI デバイスが検出可能であることを確認します。**

```
# iscsiadm --mode discovery --type sendtargets --portal server1.example.com
143.22.16.33:3260,1 iqn.2010-05.com.example.server1:iscsirhel7guest
```

手順13.8 デバイスの接続のテスト

1. **新しい iSCSI デバイスの接続**
新しいデバイス (*iqn.2010-05.com.example.server1:iscsirhel7guest*) を接続して、デバイスを接続できるかどうかを判断します。

```
# 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:
143.22.16.33,3260]
Login to [iface: default, target: iqn.2010-05.com.example.server1:iscsirhel7guest, portal:
143.22.16.33,3260] successful.
```

2. **デバイスの取り外し**

```
# 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: 143.22.16.33,3260
Logout of [sid: 2, target: iqn.2010-05.com.example.server1:iscsirhel7guest, portal:
143.22.16.33,3260] successful.
```

手順13.9 iSCSI ストレージプールで libvirt のシークレットを使用する

注記

この手順は、iSCSI ターゲットの作成 時に、*user_ID* および *password* が定義される場合に必要です。

ユーザー名とパスワードのパラメーターは、iSCSI ストレージプールをセキュリティー保護するため、**virsh** で設定できます。これは、プールを定義する前または後に設定できますが、認証設定を有効にするにはプールを起動する必要があります。

1. **libvirt の秘密ファイルを作成する**
チャレンジハンドシェイク認証プロトコル (CHAP) のユーザー名を使用して、libvirt シークレットファイルを作成します。以下に例を示します。

```
<secret ephemeral='no' private='yes'>
   <description>Passphrase for the iSCSI example.com server</description>
   <usage type='iscsi'>
      <target>iscsirhel7secret</target>
   </usage>
</secret>
```

2. **シークレットの定義**

```
# virsh secret-define secret.xml
```

3. **UUID の確認**

```
# virsh secret-list
UUID                                  Usage
--------------------------------------------------------------------------------
2d7891af-20be-4e5e-af83-190e8a922360  iscsi iscsirhel7secret
```

4. **UID へのシークレットの割り当て**
以下のコマンドを使用して、前の手順の出力の UUID に、シークレットを割り当てます。これにより、CHAP ユーザー名とパスワードが、libvirt が制御するシークレットリストにあることが保証されます。

```
# MYSECRET=`printf %s "password123" | base64`
# virsh secret-set-value 2d7891af-20be-4e5e-af83-190e8a922360 $MYSECRET
```

5. **ストレージプールに認証エントリーを追加します。**
virsh edit を使用して、ストレージプールの XML ファイル内の **<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。

以下に例を示します。

```
<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

サブ要素 ***<auth>*** は、ゲスト XML 内の ***<pool>*** 要素および ***<disk>*** 要素の異なる場所に存在します。***<pool>*** の場合は、***<auth>*** が ***<source>*** 要素内に指定されます。認証は一部のプールソース (iSCSI および RBD) のプロパティーであるため、これはプールソースの検索場所を説明する要素となります。ドメインのサブ要素である ***<disk>*** の場合、iSCSI ディスクまたは RBD ディスクに対する認証は、ディスクのプロパティーになります。

また、ディスクのサブ要素 ***<auth>*** は、ストレージプールのサブ要素とは異なります。

```
<auth username='redhat'>
  <secret type='iscsi' usage='iscsirhel7secret'/>
</auth>
```

6. **変更を有効にします。**
こ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する必要があります。

- ストレージプールが起動していない場合は、virsh を使用したストレージプールの作成 の手順に従ってストレージプールを定義し、起動します。
- プールがすでに起動している場合は、次のコマンドを入力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
```

パラメーター
以下の表は、iSCSI ベースのストレージプールを作成する場合に必要な、XML ファイル、**virsh pool-define-as** コマンド、および **Virtual Machine Manager** アプリケーションのパラメーターのリストを示しています。

表13.5 iSCSI ベースのストレージプールパラメーター

説明	XML	pool-define-as	Virtual Machine Manager
ストレージプールのタイプ	<pool type='iscsi'>	[type] iscsi	iscsi: iSCSI ターゲット
ストレージプールの名前	<name>*name*</name>	[name] *name*	Name
ホストの名前。	<source> <host name='*hostname*' />	source-host *hostname*	ホスト名
iSCSI IQN	device path="*iSCSI_IQN*" /> </source>	source-dev *iSCSI_IQN*	ソース IQN

説明	XML	pool-define-as	Virtual Machine Manager
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<target> <path>/dev/*disk/by-path*</path> </target>	target *path_to_pool*	ターゲットパス
(必要に応じて) iSCSI イニシエーターの IQN。これは、ACL が LUN を特定のイニシエーターに制限する場合に限り必要です。	<initiator> <iqn name='*initiator0*' /> </initiator>	**以下の注記を参照してください。**	イニシエーター IQN

注記

iSCSI イニシエーターの IQN は、**virsh find-storage-pool-sources-as iscsi** コマンドを使用して指定できます。

virsh を使用してストレージプールを作成する場合は、ストレージプールが作成されたことの確認 に進みます。

例
以下は、iSCSI ベースのストレージプールの XML ファイルの例になります。

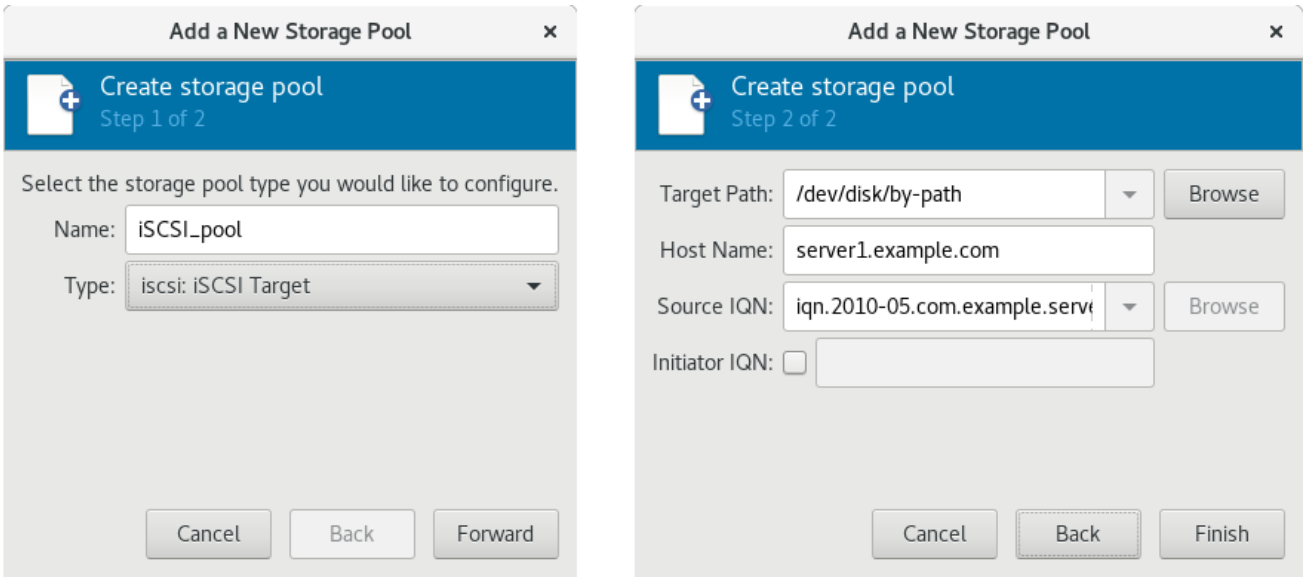
```
<pool type='iscsi'>
  <name>iSCSI_pool</name>
  <source>
    <host name='server1.example.com'/>
    <device path='iqn.2010-05.com.example.server1:iscsirhel7guest'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

以下は、iSCSI ベースのストレージプールを作成するコマンドの例になります。

```
# virsh pool-define-as --name iSCSI_pool --type iscsi --source-host server1.example.com --source-
dev iqn.2010-05.com.example.server1:iscsirhel7guest --target /dev/disk/by-path
Pool iSCSI_pool defined
```

以下の図は、iSCSI ベースのストレージプールを作成するための、仮想マシン XML 設定 **Virtual Machine Manager** の Add a New Storage Pool ダイアログボックスの例を示しています。

図13.8 新しい iSCSI ベースのストレージプールの例を追加します。



13.2.3.6. LVM ベースのストレージプール

推奨事項
LVM ベースのストレージプールを作成する前に、以下の点に注意してください。

- LVM ベースのストレージプールは、LVM の柔軟性を完全には提供しません。
- libvirt は、シン論理ボリュームに対応しますが、シンストレージプールの機能は提供しません。
- LVM ベースのストレージプールは、ボリュームグループです。Logical Volume Manager コマンド、または **virsh** コマンドを使用して、ボリュームグループを作成できます。**virsh** インターフェイスを使用してボリュームグループを管理する場合は、**virsh** コマンドを使用してボリュームグループを作成します。

  ボリュームグループの詳細は、『Red Hat Enterprise Linux Logical Volume Manager Administration Guide』 を参照してください。
- LVM ベースのストレージプールには、完全なディスクパーティションが必要です。この手順で新しいパーティションまたはデバイスをアクティベートすると、パーティションはフォーマットされ、すべてのデータが削除されます。ホストの既存のボリュームグループ (VG) を使用すると、何も消去されません。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作成することが推奨されます。

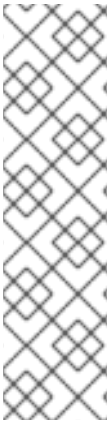
  LVM ボリュームグループの作成方法は、『Red Hat Enterprise Linux Logical Volume Manager Administration Guide』 を参照してください。
- 既存の VG に LVM ベースのストレージプールを作成する場合は、**pool-build** コマンドを実行しないでください。

仮想マシンが準備されたことを確認したら、ストレージプールの定義 でストレージプールの作成を続行します。

パラメーター
以下の表は、LVM ベースのストレージプールを作成する場合に必要な、XML ファイル、**virsh pool-define-as** コマンド、および **Virtual Machine Manager** アプリケーションのパラメーターのリストを示しています。

表13.6 LVM ベースのストレージプールパラメーター

説明	XML	pool-define-as	Virtual Machine Manager
ストレージプールのタイプ	<pool type='logical'>	[type] logical	論理: LVM ボリュームグループ
ストレージプールの名前	<name>*name*</name>	[name] *name*	Name
ストレージプールのデバイスのパス	<source> <device path='*device_path*' />	source-dev *device_path*	ソースパス
ボリュームグループの名前	<name='*VG-name*' />	source-name *VG-name*	ソースパス
仮想グループの形式	<format type='lvm2' /> </source>	source-format lvm2	該当なし
ターゲットパス	<target> <path='*target-path*' /> </target>	target *target-path*	ターゲットパス



注記

論理ボリュームグループが複数のディスクパーティションで作成されている場合は、複数のソースデバイスがリスト表示されている可能性があります。以下に例を示します。

```
<source>
  <device path='/dev/sda1'/>
  <device path='/dev/sdb3'/>
  <device path='/dev/sdc2'/>
  ...
  </source>
```

virsh を使用してストレージプールを作成する場合は、ストレージプールが作成されたことの確認 に進みます。

例
以下は、LVM ベースのストレージプールの XML ファイルの例です。

```
<pool type='logical'>
  <name>guest_images_lvm</name>
  <source>
    <device path='/dev/sdc'/>
```

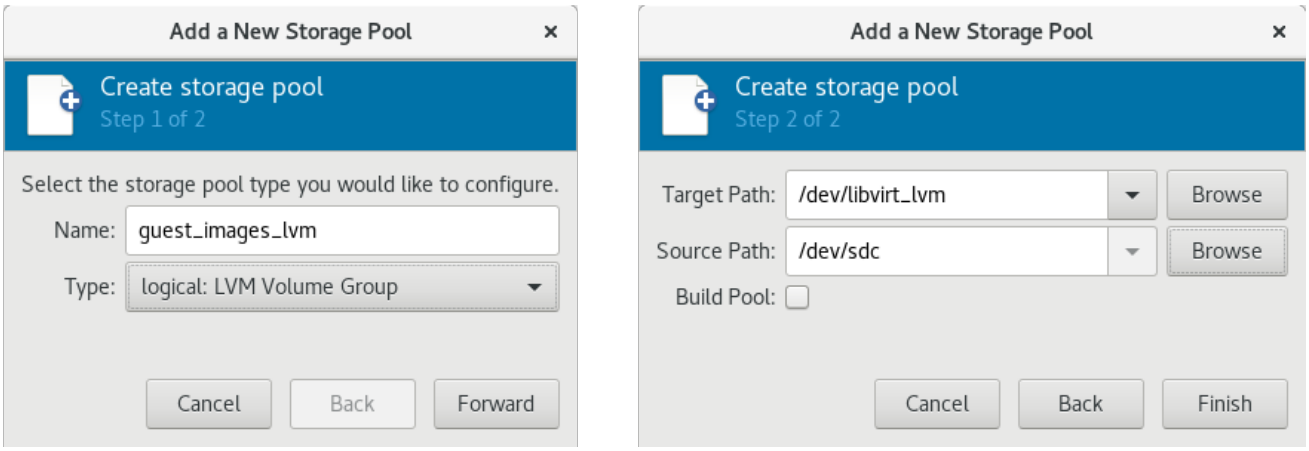
```
    <name>libvirt_lvm</name>
    <format type='lvm2'/>
  </source>
  <target>
    <path>/dev/libvirt_lvm</path>
  </target>
</pool>
```

以下は、LVM ベースのストレージプールを作成するコマンドの例です。

```
# virsh pool-define-as guest_images_lvm logical --source-dev=/dev/sdc --source-name libvirt_lvm --
target /dev/libvirt_lvm
Pool guest_images_lvm defined
```

以下の図は、LVM ベースのストレージプールを作成するための、仮想マシン XML 設定 **Virtual Machine Manager** の Add a New Storage Pool ダイアログボックスの例を示しています。

図13.9 新しい LVM ベースのストレージプールの例を追加します。



13.2.3.7. NFS ベースのストレージプール

前提条件
NFS (Network File System) ベースのストレージプールを作成するには、ホストマシンで NFS サーバーを使用するように設定しておく必要があります。NFS の詳細は、『Red Hat Enterprise Linux Storage Administration Guide』 を参照してください。

NFS サーバーが適切に設定されていることを確認したら、ストレージプールの定義 でストレージプールの作成を続行します。

パラメーター
以下の表は、NFS ベースのストレージプールを作成する場合に必要な、XML ファイル、**virsh pool-define-as** コマンド、および **Virtual Machine Manager** アプリケーションのパラメーターのリストを示しています。

表13.7 NFS ベースのストレージプールパラメーター

説明	XML	pool-define-as	Virtual Machine Manager

説明	XML	pool-define-as	Virtual Machine Manager
ストレージプールのタイプ	<pool type='netfs'>	[type] netfs	netfs: Network Exported Directory
ストレージプールの名前	<name>*name*</name>	[name] *name*	Name
マウントポイントが置かれている NFS サーバーのホスト名。これは、ホスト名または IP アドレスになります。	<source> <host name='*host_name*' />	source-host *host_name*	ホスト名
NFS サーバーで使用されるディレクトリー	<dir path='*source_path*' /> </source>	source-path *source_path*	ソースパス
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<target> <path>/*target_path*</path> </target>	target *target_path*	ターゲットパス

virsh を使用してストレージプールを作成する場合は、ストレージプールが作成されたことの確認 に進みます。

例
以下は、NFS ベースのストレージプールの XML ファイルの例です。

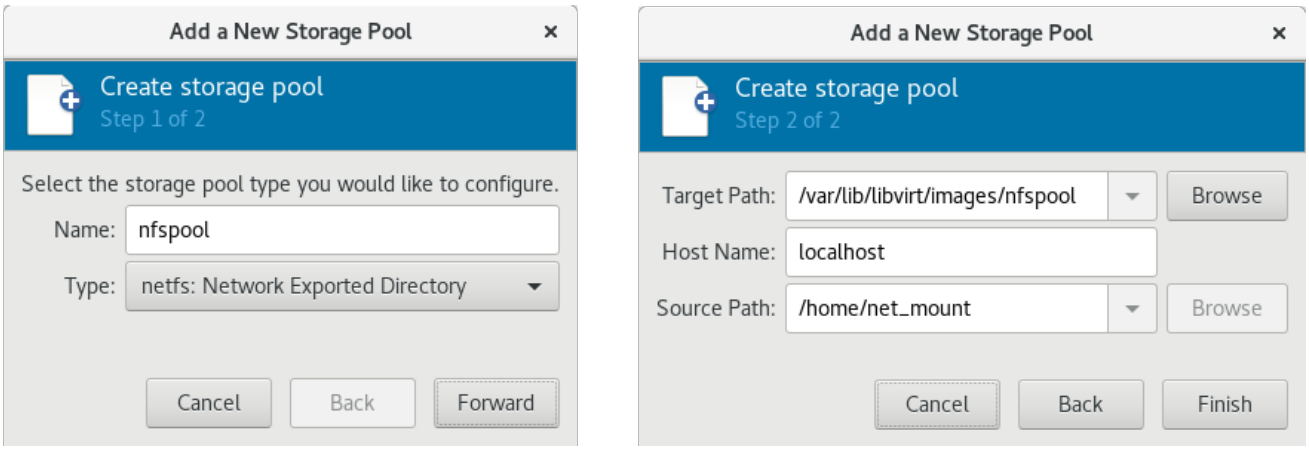
```
<pool type='netfs'>
  <name>nfspool</name>
  <source>
    <host name='localhost'/>
    <dir path='/home/net_mount'/>
  </source>
  <target>
    <path>/var/lib/libvirt/images/nfspool</path>
  </target>
</pool>
```

以下は、NFS ベースのストレージプールを作成するコマンドの例です。

```
# virsh pool-define-as nfspool netfs --source-host localhost --source-path /home/net_mount --target
/var/lib/libvirt/images/nfspool
Pool nfspool defined
```

以下の図は、NFS ベースのストレージプールを作成するための、仮想マシン XML 設定 **Virtual Machine Manager**の Add a New Storage Pool ダイアログボックスの例を示しています。

図13.10 新しい NFS ベースのストレージプールの追加の例



13.2.3.8. SCSI デバイスを使用する vHBA ベースのストレージプール

注記

Virtual Machine Manager を使用し、SCSI デバイスを使用して vHBA ベースのストレージプールを作成することはできません。

推奨事項
N_Port ID Virtualization (NPIV) は、1 つの物理ファイバーチャンネルのホストバスアダプター (HBA) の共有を可能にするソフトウェアテクノロジーです。これにより、複数のゲストが複数の物理ホストから同じストレージを認識できるため、ストレージの移行パスが容易になります。そのため、正しいストレージパスが指定されていれば、移行を使用してストレージを作成またはコピーする必要はありません。

仮想化では、*仮想ホストバスアダプター*、または *vHBA* が、仮想マシンの論理ユニット番号 (LUN) を制御します。複数の KVM ゲスト間でファイバーチャネルデバイスパスを共有するホストには、仮想マシンごとに vHBA を作成する必要があります。1 つの vHBA を、複数の KVM ゲストで使用することはできません。

NPIV の各 vHBA は、その親 HBA と、独自の World Wide Node Name (WWNN) および World Wide Port Name (WWPN) で識別されます。ストレージのパスは、WWNN および WWPN の値で決定します。親 HBA は、**scsi_host#** または WWNN/WWPN ペアとして定義できます。

注記

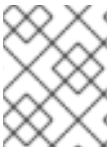
親 HBA が **scsi_host#** として定義され、ハードウェアがホストマシンに追加されている場合、**scsi_host#** の割り当てが変更される可能性があります。したがって、WWNN/WWPN のペアを使用して親 HBA を定義することが推奨されます。

これは、vHBA 設定を保持するため、vHBA に基づいて libvirt ストレージプールを定義することが推奨されます。

libvirt ストレージプールを使用すると、主に以下の利点があります。

- libvirt コードは、virsh コマンドの出力を使用すると、LUN のパスを簡単に見つけることができます。
- 仮想マシンの移行には、ターゲットマシンで同じ vHBA 名を持つストレージプールの定義と起動のみが必要です。これを行うには、仮想マシンの XML 設定で、vHBA LUN、libvirt ストレー

ジプール、およびボリューム名を指定する必要があります。例は、「SCSI デバイスを使用する vHBA ベースのストレージプール」 を参照してください。



注記

vHBA を作成する前に、ホストの LUN でストレージアレイ (SAN) 側のゾーンを設定して、ゲスト間の分離を提供し、データの破損を防ぐことが推奨されます。

永続的な vHBA 設定を作成するには、最初に以下の形式を使用して、libvirt **'scsi'** ストレージプール XML ファイルを作成します。同じ物理 HBA 上のストレージプールを使用する 1 つの vHBA を作成する場合は、システムの **/dev/disk/by-{path|id|uuid|label}** のような場所など、**<path>** 値に安定した場所を使用することが推奨されます。

同じ物理 HBA 上でストレージプールを使用する複数の vHBA を作成する場合は、**<path>** フィールドの値を **/dev/** のみにする必要があります。それ以外の場合は、ストレージプールボリュームが 1 つの vHBA からしか確認できず、NPIV 設定で、ホストのデバイスを複数のゲストに公開することができません。

<path> および **<target>** の要素の詳細は、アップストリームの libvirt ドキュメント を参照してください。

前提条件
vHBA を作成し、SCSI デバイスを使用して vHBA ベースのストレージプールを作成できるようにしてある。

手順13.10 vHBA の作成

1. **ホストシステムで HBA の場所を特定します。**
   ホストシステムの HBA を特定するには、**virsh nodedev-list --cap vports** コマンドを使用します。

   以下の例は、vHBA に対応する HBA が 2 つ搭載されているホストを示しています。

   ```
   # virsh nodedev-list --cap vports
   scsi_host3
   scsi_host4
   ```

2. **HBA の詳細を確認します。**
   virsh nodedev-dumpxml *HBA_device* コマンドを実行して、HBA の詳細を表示します。

   ```
   # virsh nodedev-dumpxml scsi_host3
   ```

   コマンドの出力には、vHBA の作成に使用する **<name>**、**<wwnn>**、および **<wwpn>** フィールドの一覧が表示されます。サポートされる vHBA の最大数が **<max_vports>** に表示されます。以下に例を示します。

   ```
   <device>
     <name>scsi_host3</name>
     <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3</path>
     <parent>pci_0000_10_00_0</parent>
     <capability type='scsi_host'>
       <host>3</host>
       <unique_id>0</unique_id>
       <capability type='fc_host'>
   ```

```
      <wwnn>20000000c9848140</wwnn>
      <wwpn>10000000c9848140</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
    <capability type='vport_ops'>
      <max_vports>127</max_vports>
      <vports>0</vports>
    </capability>
  </capability>
</device>
```

この例では、**<max_vports>** には、HBA 設定で使用できる仮想ポートが 127 個あることを示しています。**<vports>** 値は、現在使用されている仮想ポートの数を示します。この値は、vHBA の作成後に更新されます。

3. **vHBA ホストデバイスの作成**
vHBA ホスト用に、以下のいずれかの XML ファイルを作成します。この例では、ファイル名は *vhba_host3.xml* です。

次の例では、**scsi_host*3*** を使用して親 vHBA を説明します。

```
# cat vhba_host3.xml
<device>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <capability type='fc_host'>
    </capability>
  </capability>
</device>
```

次の例では、WWNN/WWPN のペアを使用して親 vHBA を説明します。

```
# cat vhba_host3.xml
<device>
  <name>vhba</name>
  <parent wwnn='20000000c9848140' wwpn='10000000c9848140'/>
  <capability type='scsi_host'>
    <capability type='fc_host'>
    </capability>
  </capability>
</device>
```

注記

WWNN および WWPN の値は、手順13.10「vHBA の作成」 に表示される HBA の詳細にある値と一致する必要があります。

<parent> フィールドは、この vHBA デバイスに関連付ける HBA デバイスを指定します。**<device>** タグの詳細は、ホスト用の新しい vHBA デバイスを作成するために、次の手順で使用されます。**nodedev** XML 形式の詳細は、libvirt アップストリームページ を参照してください。

4. **vHBA ホストデバイスに新しい vHBA を作成します。**

vhba_host3 に基づいて vHBA を作成するには、**virsh nodedev-create** コマンドを使用します。

```
# virsh nodedev-create vhba_host3.xml
Node device scsi_host5 created from vhba_host3.xml
```

5. **vHBA の確認**
virsh nodedev-dumpxml コマンドを使用して、新しい vHBA の詳細 (**scsi_host5**) を確認します。

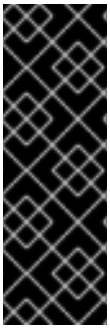
```
# virsh nodedev-dumpxml scsi_host5
<device>
  <name>scsi_host5</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3/vport-3:0-0/host5</path>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <host>5</host>
    <unique_id>2</unique_id>
    <capability type='fc_host'>
      <wwnn>5001a4a93526d0a1</wwnn>
      <wwpn>5001a4ace3ee047d</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
  </capability>
</device>
```

vHBA を確認したら、ストレージプールの定義 でストレージプールの作成を続行します。

パラメーター
以下の表は、vHBA ベースのストレージプールを作成する場合に必要な、XML ファイル、**virsh pool-define-as** コマンド、および **Virtual Machine Manager** アプリケーションのパラメーターのリストを示しています。

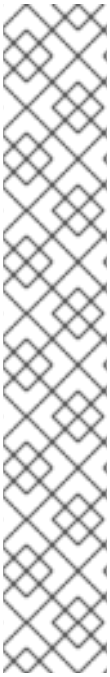
表13.8 vHBA ベースのストレージプールパラメーター

説明	XML	pool-define-as
ストレージプールのタイプ	<pool type='scsi'>	scsi
ストレージプールの名前	<name>*name*</name>	--adapter-name *name*
vHBA の識別子。**parent** 属性はオプションです。	<source> <adapter type='fc_host' [parent=*parent_scsi_device*] wwnn='*WWNN*' wwpn='*WWPN*' /> </source>	[--adapter-parent *parent*] --adapter-wwnn *wwnn* --adapter-wpnn *wwpn*
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<target> <path>*target_path*</path> </target>	target *path_to_pool*



重要

<path> フィールドが **/dev/** の場合、libvirt は、ボリュームデバイスパスで一意の短いデバイスパスを生成します。たとえば、**/dev/sdc** です。それ以外の場合は、物理ホストパスが使用されます。たとえば、**/dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0** などです。一意の短いデバイスパスを使用すると、複数のストレージプールで、同じボリュームを複数のゲストにリスト表示できます。物理ホストのパスを複数のゲストで使用すると、デバイスタイプが重複していることを示す警告が発生することがあります。



注記

parent 属性は、**<adapter>** フィールドで、パスを変更して NPIV LUN を使用できる物理 HBA の親を識別するために使用できます。このフィールドの **scsi_host*N*** は、**vports** 属性および **max_vports** 属性と組み合わせて、親 ID を完了します。**parent** 属性、**parent_wwnn** 属性、**parent_wwpn** 属性、または **parent_fabric_wwn** 属性では、ホストの再起動後に同じ HBA が使用されるというさまざまな程度の保証を提供します。

- **parent**を指定しないと、libvirt は、NPIV に対応する最初の **scsi_host*N*** アダプターを使用します。
- **parent** のみが指定されている場合、設定に SCSI ホストアダプターを追加すると、問題が発生する可能性があります。
- **parent_wwnn** または **parent_wwpn** を指定した場合は、ホストの再起動後に同じ HBA が使用されます。
- **parent_fabric_wwn** を使用する場合は、ホストの再起動後、使用されている**scsi_host*N***に関係なく、同じファブリックの HBA が選択されます。

virsh を使用してストレージプールを作成する場合は、ストレージプールが作成されたことの確認 に進みます。

例

以下は、vHBA ベースのストレージプールの XML ファイルの例です。最初の例は、HBA にある唯一のストレージプールの例です。2 つ目は、単一の vHBA を使用し、**parent** 属性を使用して、SCSI ホストデバイスを識別するいくつかのストレージプールの 1 つであるストレージプールの例になります。

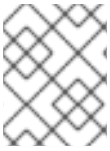
```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' wwnn='5001a4a93526d0a1' wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' parent='scsi_host3' wwnn='5001a4a93526d0a1'
wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
```

```
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

以下は、vHBA ベースのストレージプールを作成するコマンドの例です。

```
# virsh pool-define-as vhbapool_host3 scsi --adapter-parent scsi_host3 --adapter-wwnn
5001a4a93526d0a1 --adapter-wwpn 5001a4ace3ee047d --target /dev/disk/by-path
Pool vhbapool_host3 defined
```



注記

virsh では、**parent_wwnn** 属性、**parent_wwpn** 属性、または **parent_fabric_wwn** 属性を定義する方法は提供されていません。

vHBA LUN を使用するように仮想マシンを設定する
vHBA 用にストレージプールを作成したら、vHBA LUN を仮想マシンの設定に追加する必要があります。

1. 仮想マシンの XML に、仮想マシンにディスクボリュームを作成します。
2. **<source>** パラメーターで、**storage pool** と **storage volume** を指定します。

以下は例になります。

```
<disk type='volume' device='disk'>
  <driver name='qemu' type='raw'/>
  <source pool='vhbapool_host3' volume='unit:0:4:0'/>
  <target dev='hda' bus='ide'/>
</disk>
```

disk の代わりに**lun** デバイスーを指定するには、以下の例を参照してください。

```
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source pool='vhbapool_host3' volume='unit:0:4:0' mode='host'/>
  <target dev='sda' bus='scsi'/>
  <shareable />
</disk>
```

SCSI LUN ベースのストレージをゲストに追加する XML 設定例は、「ゲストへの SCSI LUN ベースのストレージの追加」 を参照してください。

ハードウェア障害が発生した場合に、LUN への再接続が成功するように、**fast_io_fail_tmo** オプションおよび **dev_loss_tmo** オプションを変更することが推奨されます。詳細は、Reconnecting to an exposed LUN after a hardware failure を参照してください。

13.2.4. ストレージプールの削除

virsh または 同Virtual Machine Manager を使用して、ストレージプールを削除できます。

13.2.4.1. ストレージプールを削除するための前提条件

削除するストレージプールを使用するその他のゲスト仮想マシンに悪影響を及ぼさないようにするには、ストレージプールを停止して、ストレージプールによって使用されているリソースを解放することを推奨します。

13.2.4.2. virsh を使用したストレージプールの削除

1. 定義したストレージプールのリストを表示します。

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

2. 削除するストレージプールを停止します。

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

3. **(任意)** ストレージプールの種類によっては、ストレージプールが含まれるディレクトリーを削除できる場合があります。

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

4. ストレージプールの定義を削除します。

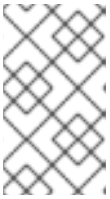
```
# virsh pool-undefine guest_images_disk
```

5. プールが定義されていないことを確認します。

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

13.2.4.3. Virtual Machine Manager を使用したストレージプールの削除

1. Connection Details ウィンドウ の Storage タブにあるストレージプールリストで、削除するストレージプールを選択します。
2. Storage ウィンドウの下部にある  をクリックします。これによりストレージプールが停止し、使用中のリソースがすべて解放されます。
3.  をクリックします。



注記

 アイコンは、ストレージプールが停止している場合にのみ有効になります。

ストレージプールが削除されます。

13.3. ストレージボリュームの使用

このセクションでは、ストレージボリュームの使用方法を説明します。ここでは、概念情報と、**virsh** コマンドおよび **Virtual Machine Manager** を使用してストレージボリュームを作成、設定、および削除する方法を詳細に説明します。

13.3.1. ストレージボリュームの概念

ストレージプールは、*ストレージボリューム* に分類されます。ストレージボリュームは、libvirt が処理する物理パーティション、LVM 論理ボリューム、ファイルベースのディスクイメージ、その他のストレージタイプの抽象化です。ストレージボリュームは、基本となるハードウェアに関係なく、ゲスト仮想マシンにローカルストレージデバイスとして提示されます。

注記

以下のセクションでは、virsh がストレージボリュームの管理で使用するコマンドと引数をすべて含んでいるわけではありません。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。

ホストマシンでは、ストレージボリュームは、その名前と、そこから派生するストレージプールの識別子で参照されます。**virsh** コマンドラインでは、***--pool*** *storage_pool volume_name* の形式を取ります。

たとえば、*guest_images* プールの *firstimage* という名前のボリュームの場合は以下のようになります。

```
# virsh vol-info --pool guest_images firstimage
  Name:           firstimage
  Type:           block
  Capacity:       20.00 GB
  Allocation:     20.00 GB
```

追加のパラメーターおよび引数は、「ボリューム情報のリスト表示」 を参照してください。

13.3.2. ストレージボリュームの作成

本セクションでは、**virsh** および **Virtual Machine Manager** を使用して、ストレージプールからストレージボリュームを作成する方法を説明します。ストレージボリュームを作成したら、ストレージデバイスをゲストに追加することができます。

13.3.2.1. virsh を使用したストレージボリュームの作成

次のいずれかを行います。

- XML ファイルを使用してストレージボリュームを定義します。

  a.新規デバイスに必要なストレージボリューム情報を含む一時 XML ファイルを作成します。

  XML ファイルには、以下を含む特定のフィールドが含まれている必要があります。

  - **名前** - ストレージボリュームの名前。
  - **割り当て** - ストレージボリュームのストレージ割り当ての合計数。

- **容量** - ストレージボリュームの論理容量。ボリュームがスパースの場合、この値は **allocation** の値とは異なります。
- **ターゲット** - ホストシステムのストレージボリュームのパス、さらに任意でそのパーミッションとラベル。

以下は、ストレージボリューム定義の XML ファイルの例です。この例では、ファイルは ~/**guest_volume.xml** に保存されます。

```
<volume>
  <name>volume1</name>
  <allocation>0</allocation>
  <capacity>20G</capacity>
  <target>
    <path>/var/lib/virt/images/sparse.img</path>
  </target>
</volume>
```

b.**virsh vol-create** コマンドを使用して、XML ファイルに基づいてストレージボリュームを作成します。

```
# virsh vol-create guest_images_dir ~/guest_volume.xml
  Vol volume1 created
```

c.手順 a で作成した XML ファイルを削除します。

- **virsh vol-create-as** を実行して、ストレージボリュームを作成します。

```
# virsh vol-create-as guest_images_dir volume1 20GB --allocation 0
```

- **virsh vol-clone** コマンドを使用して、既存のストレージボリュームのクローンを作成します。**virsh vol-clone** コマンドは、クローンを作成するストレージボリュームを含むストレージプールと、新しく作成したストレージボリュームの名前を指定する必要があります。

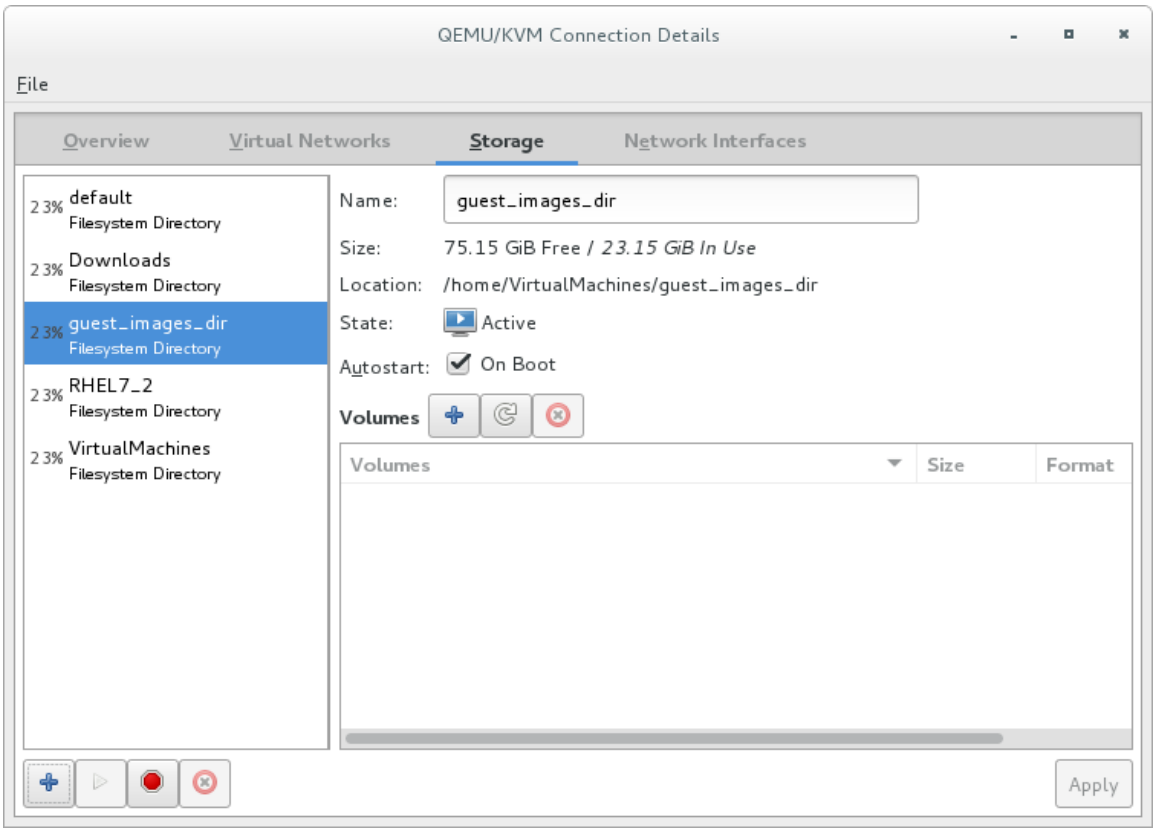
```
# virsh vol-clone --pool guest_images_dir volume1 clone1
```

13.3.2.2. Virtual Machine Manager を使用したストレージボリュームの作成

手順13.11 Virtual Machine Manager を使用したストレージボリュームの作成

1. **ストレージ設定を開く**
   a. Virtual Machine Manager で **Edit** メニューを開き、**Connection Details** を選択します。
   b. **Connection Details** ウィンドウの **Storage** タブをクリックします。

図13.11 ストレージタブ



Connection Details ウィンドウの左側のペインには、ストレージプールのリストが表示されます。

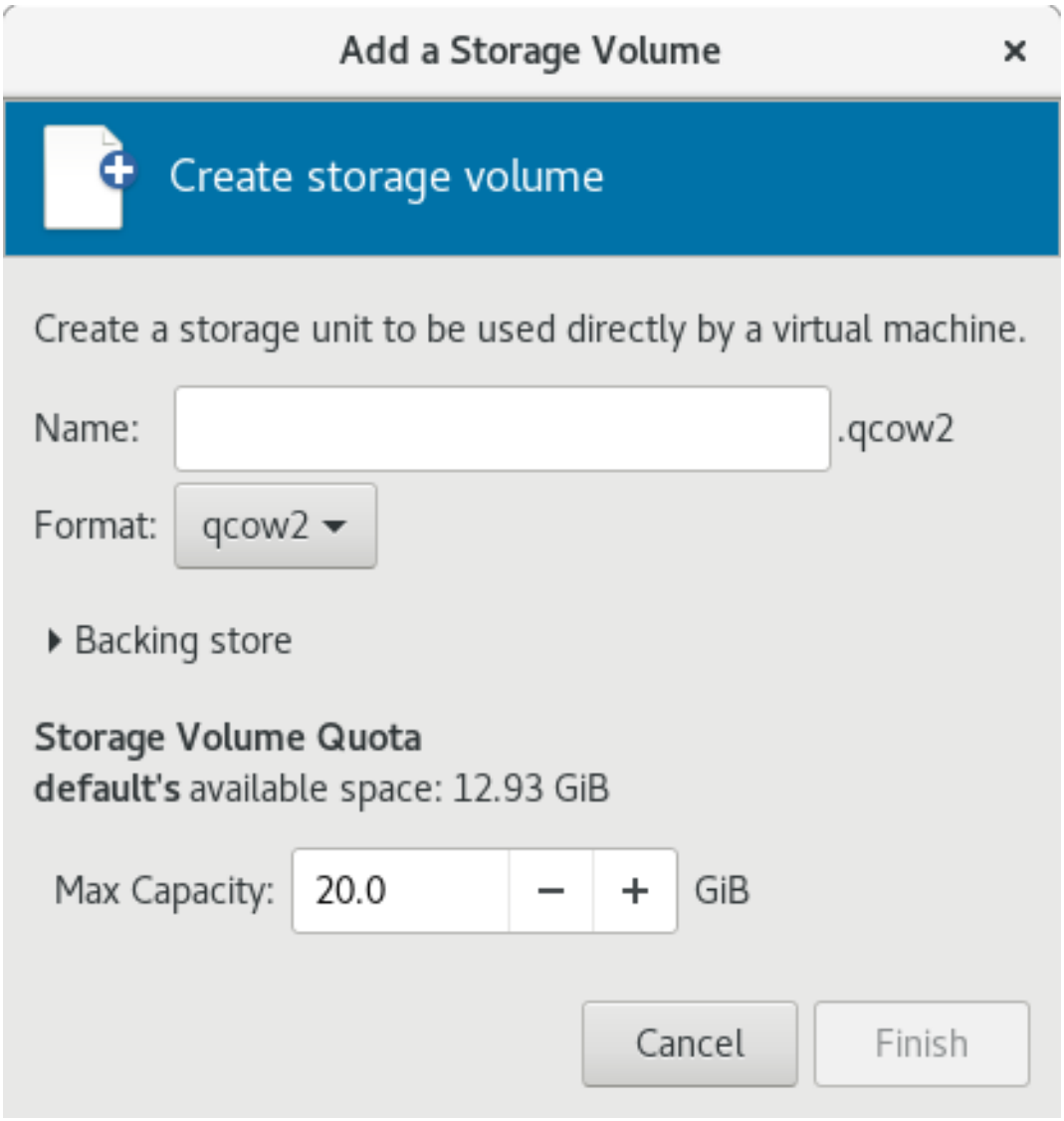
2. **ストレージボリュームを作成するストレージプールを選択します。**
ストレージプールのリストで、ストレージボリュームを作成するストレージプールをクリックします。

   選択したストレージプールに設定したストレージボリュームは、画面下部の **Volumes** ペインに表示されます。

3. **新しいストレージボリュームの追加**

   Volumes リストの上にある  ボタンをクリックします。**Add a Storage Volume** のダイアログが表示されます。

図13.12 ストレージボリュームの作成



4. **ストレージボリュームの設定**
以下のパラメーターを使用して、ストレージボリュームを設定します。

   - **Name** フィールドにストレージプールの名前を入力します。
   - **Format** リストからストレージボリュームの形式を選択します。
   - **Max Capacity** フィールドに、ストレージボリュームの最大サイズを入力します。

5. **作成を終了します。**
Finish をクリックします。**Add a Storage Volume** のダイアログが閉じ、ストレージボリュームが **Volumes** リストに表示されます。

13.3.3. ストレージボリュームの表示

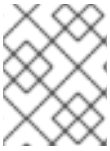
ストレージプールから複数のストレージボリュームを作成できます。**virsh vol-list** を使用して、ストレージプールのストレージボリュームをリスト表示することもできます。以下の例では、*guest_images_disk* に 3 つのボリュームが含まれています。

```
# 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              /home/VirtualMachines/guest_images_dir/volume1
volume2              /home/VirtualMachines/guest_images_dir/volume2
volume3              /home/VirtualMachines/guest_images_dir/volume3
```

13.3.4. データの管理

このセクションでは、ストレージボリュームのデータの管理について説明します。



注記

ストレージボリュームの種類によっては、データ管理コマンドに対応していないものもあります。詳細は、以下のセクションを参照してください。

13.3.4.1. ストレージボリュームのワイプ

ストレージボリュームのデータにアクセスできないようにするには、**virsh vol-wipe** コマンドを使用して、ストレージボリュームをワイプできます。

virsh vol-wipe コマンドを実行して、ストレージボリュームをワイプします。

```
# virsh vol-wipe new-vol vdisk
```

デフォルトでは、データはゼロで上書きされます。ただし、ストレージボリュームをワイプする方法は複数指定できます。**virsh vol-wipe** コマンドのすべてのオプションの詳細は、「ストレージボリュームのコンテンツの削除」 を参照してください。

13.3.4.2. ストレージボリュームへのデータのアップロード

virsh vol-upload コマンドを使用すると、指定したローカルファイルからストレージボリュームにデータをアップロードできます。

```
# virsh vol-upload --pool pool-or-uuid --offset bytes --length bytes vol-name-or-key-or-path local-file
```

以下は、主な**virsh vol-upload**オプションです。

- **--pool *pool-or-uuid*** - ボリュームが存在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*** - アップロードするボリュームの名前または鍵またはパスです。
- **--offset *bytes*** データの書き込みを開始するストレージボリュームの位置。
- **--length *length*** - アップロードするデータ量の上限です。

注記

local-file が指定した**--length**よりも大きい場合は、エラーが発生します。

例13.1 ストレージボリュームへのデータのアップロード

```
# virsh vol-upload sde1 /tmp/data500m.empty disk-pool
```

この例の**sde1** は、**disk-pool** ストレージプールのボリュームです。**/tmp/data500m.empty** のデータが **sde1** にコピーされます。

13.3.4.3. ストレージボリュームへのデータのダウンロード

virsh vol-download を使用すると、ストレージボリュームから、指定したローカルファイルにデータをダウンロードできます。

```
# vol-download --pool pool-or-uuid --offset bytes --length bytes vol-name-or-key-or-path local-file
```

以下は、主な**virsh vol-download**オプションです。

- **--pool *pool-or-uuid*** - ボリュームが存在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*** - ダウンロードするボリュームの名前、鍵、またはパスを指定します。
- **--offset** - データの読み込みを開始するストレージボリュームの位置です。
- **--length *length*** - ダウンロードするデータ量の上限です。

例13.2 ストレージボリュームからのデータのダウンロード

```
# virsh vol-download sde1 /tmp/data-sde1.tmp disk-pool
```

この例の**sde1** は、**disk-pool** ストレージプールのボリュームです。**sde1** のデータは、**/tmp/data-sde1.tmp** にダウンロードされます。

13.3.4.4. ストレージボリュームのサイズ変更

vol-resize コマンドを使用すると、指定したストレージボリュームの容量のサイズを変更できます。

```
# virsh vol-resize --pool pool-or-uuid vol-name-or-path pool-or-uuid capacity --allocate --delta --shrink
```

*capacity*はバイト単位で表されます。このコマンドには、ボリュームが存在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。また、このコマンドでサイズを変更するには、ボリュームの*vol-name-or-key-or-path*、名前、鍵、またはパスが必要です。

--allocate が指定されていない限り、新しい容量はスパースである可能性があります。通常、容量は新しいサイズになりますが、**--delta** が存在する場合は、既存のサイズに追加されます。**--shrink** が存在しない限り、ボリュームを縮小しようとすると失敗します。

--shrink が指定されていない限り、*容量* を負にすることはできず、負の符号は必要ないことに注意して

ください。*capacity* はスケーリングされた整数であり、接尾辞がない場合はデフォルトでバイトになります。さらに、このコマンドは、アクティブなゲストが使用していないストレージボリュームに対してのみ安全であることに注意してください。ライブサイズの変更については、「ゲスト仮想マシンのブロックデバイスのサイズの変更」 を参照してください。

例13.3 ストレージボリュームのサイズを変更

たとえば、50M のストレージボリュームを作成した場合は、次のコマンドを使用して 100M にサイズを変更できます。

```
# virsh vol-resize --pool disk-pool sde1 100M
```

13.3.5. ストレージボリュームの削除

virsh または Virtual Machine Manager を使用して、ストレージボリュームを削除できます。

注記

削除するストレージボリュームを使用するゲスト仮想マシンに悪影響を及ぼさないようにするには、これを使用するリソースを解放することを推奨します。

13.3.5.1. virsh を使用したストレージボリュームの削除

virsh vol-delete コマンドを使用して、ストレージボリュームを削除します。このコマンドは、ストレージボリュームの名前またはパスと、ストレージボリュームの抽象化の元となるストレージプールを指定する必要があります。

以下の例では、*guest_images_dir* ストレージプールから *volume_name* ストレージボリュームを削除します。

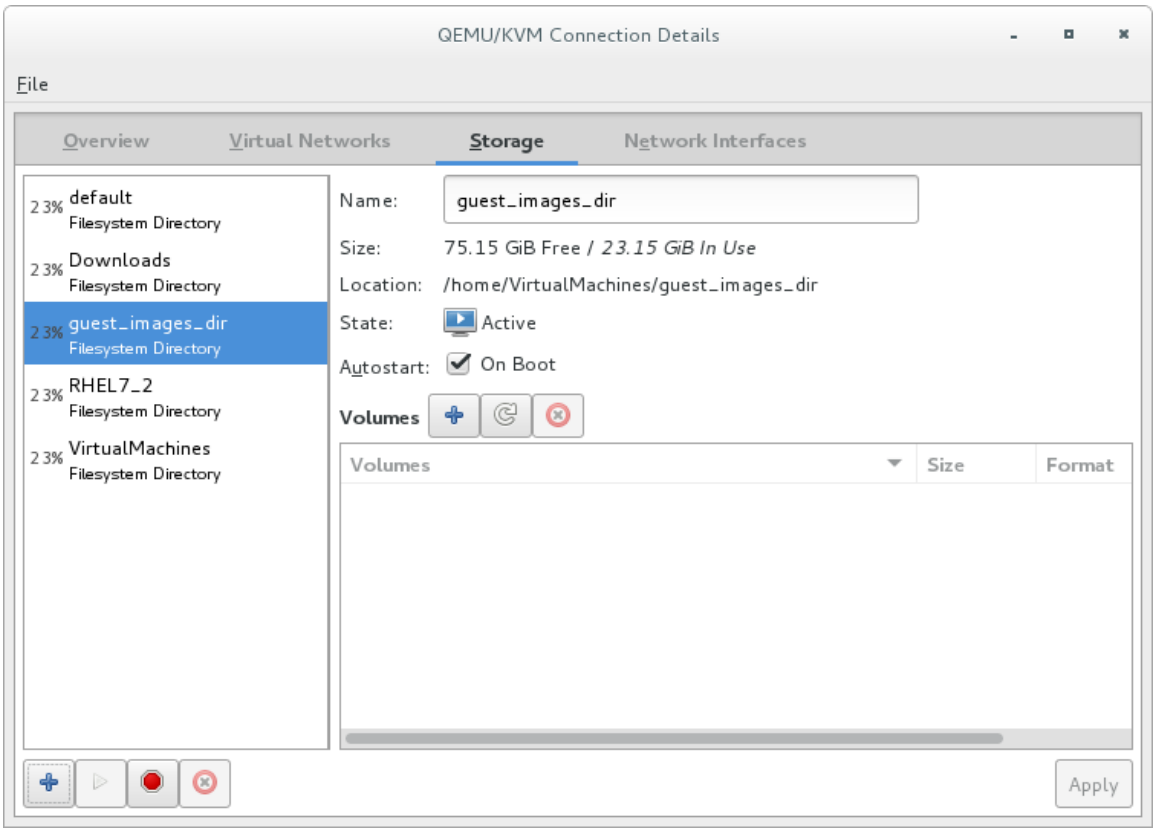
```
# virsh vol-delete volume_name --pool guest_images_dir
  vol volume_name deleted
```

13.3.5.2. Virtual Machine Manager を使用したストレージボリュームの削除

手順13.12 Virtual Machine Manager を使用したストレージボリュームの削除

1. **ストレージ設定を開く**
   a. Virtual Machine Manager で **Edit** メニューを開き、**Connection Details** を選択します。
   b. **Connection Details** ウィンドウの **Storage** タブをクリックします。

図13.13 ストレージタブ



Connection Details ウィンドウの左側のペインには、ストレージプールのリストが表示されます。

2. **削除するストレージボリュームを選択します。**

   a. ストレージプールのリストで、ストレージボリュームが抽象化されたストレージプールをクリックします。

   選択したストレージプールに設定されているストレージボリュームのリストが、画面下部の **Volumes** ペインに表示されます。

   b. 削除するストレージボリュームを選択します。

3. **ストレージボリュームを削除します。**

   a.  ボタン (**Volumes** リストの上) をクリックします。確認ダイアログが表示されます。

   b. **Yes** をクリックします。選択したストレージボリュームが削除されます。

13.3.6. ゲストへのストレージデバイスの追加

virsh または Virtual Machine Manager を使用して、ストレージデバイスをゲスト仮想マシンに追加できます。

13.3.6.1. virsh を使用したゲストへのストレージデバイスの追加

ストレージデバイスをゲスト仮想マシンに追加するには、**attach-disk** コマンドを使用します。追加するディスクに関する情報を含む引数は、XML ファイルまたはコマンドラインで指定できます。

以下は、ストレージの定義があるサンプルの XML ファイルです。

```
<disk type='file' device='disk>'>
  <driver name='qemu' type='raw' cache='none'/>
  <source file='/var/lib/libvirt/images/FileName.img'/>
  <target dev='vdb' bus='virtio'/>
</disk>
```

次のコマンドは、**NewStorage.xml** と呼ばれる XML ファイルを使用して、ディスクを *Guest1* に割り当てます。

```
# virsh attach-disk --config Guest1 ~/NewStorage.xml
```

次のコマンドは、xml ファイルを使用せずに、ディスクを *Guest1* に割り当てます。

```
# virsh attach-disk --config Guest1 --source /var/lib/libvirt/images/FileName.img --target vdb
```

13.3.6.2. Virtual Machine Manager を使用したゲストへのストレージデバイスの追加

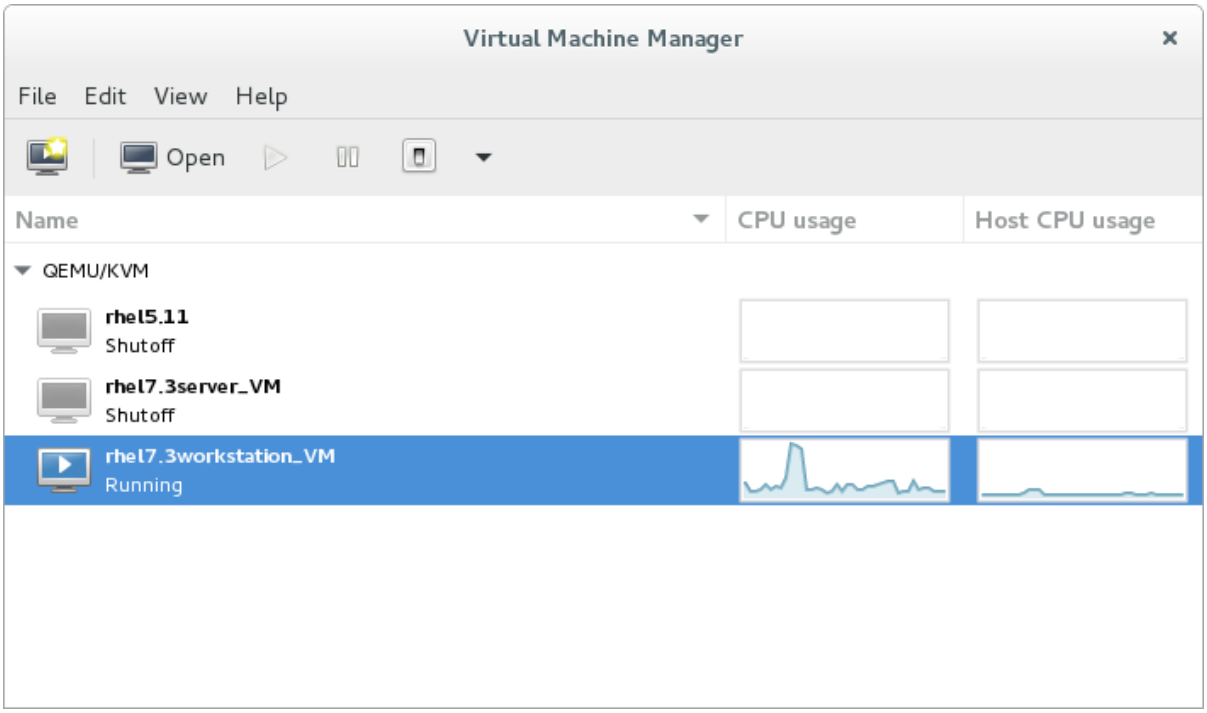
ストレージボリュームをゲスト仮想マシンに追加するか、デフォルトのストレージデバイスを作成してゲスト仮想マシンに追加できます。

13.3.6.2.1. ゲストへのストレージボリュームの追加

ゲスト仮想マシンにストレージボリュームを追加するには、次のコマンドを実行します。

1. **仮想マシンのハードウェアの詳細ウィンドウで Virtual Machine Manager を開きます。**
   root で **virt-manager** コマンドを実行するか、**Applications → System Tools → Virtual Machine Manager** を開き、virt-manager を開きます。

   図13.14 Virtual Machine Manager ウィンドウ

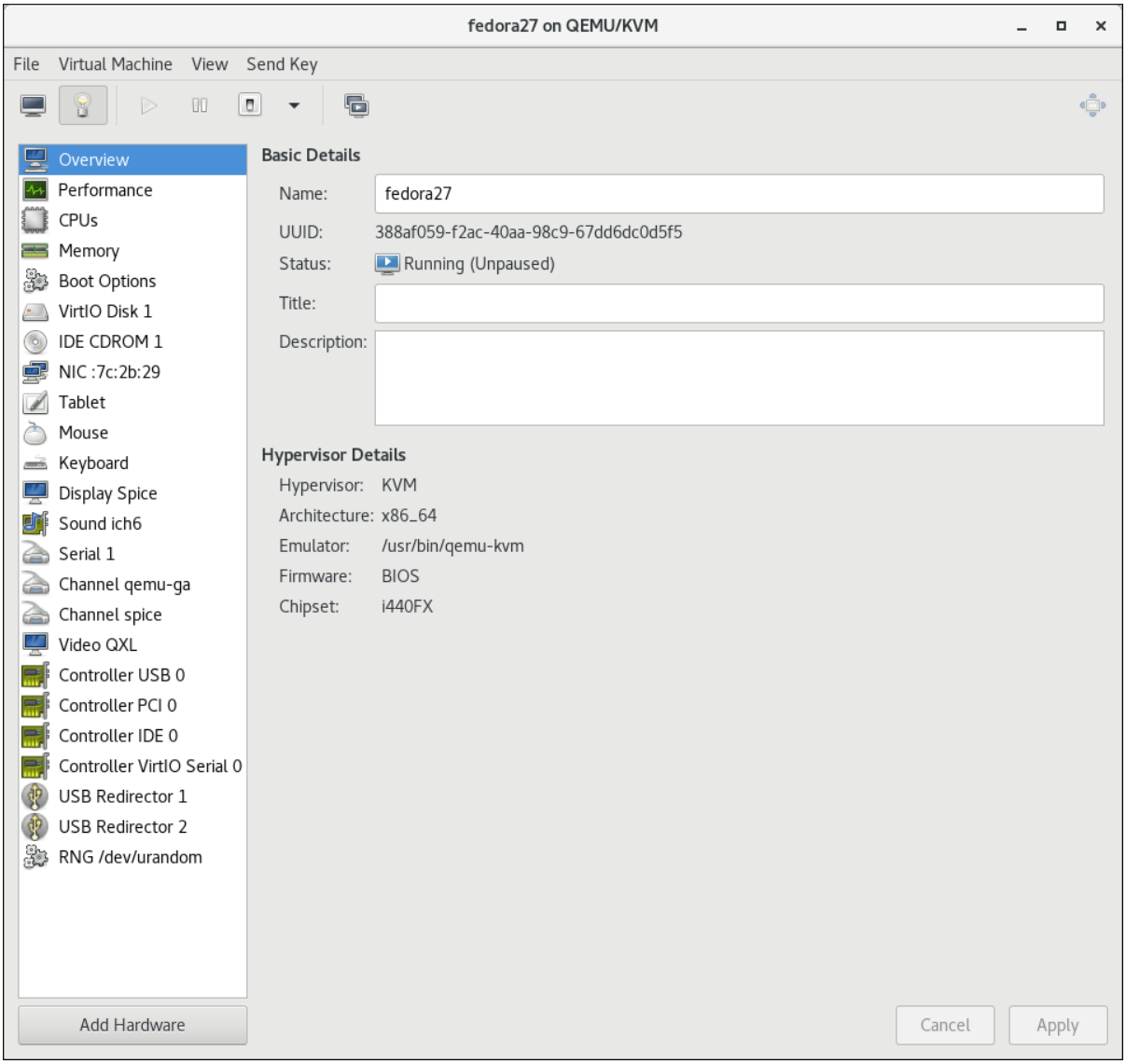
   

   ストレージボリュームを追加するゲスト仮想マシンを選択します。

Open をクリックします。仮想マシンのウィンドウが開きます。

 をクリックします。ハードウェアの詳細ウィンドウが表示されます。

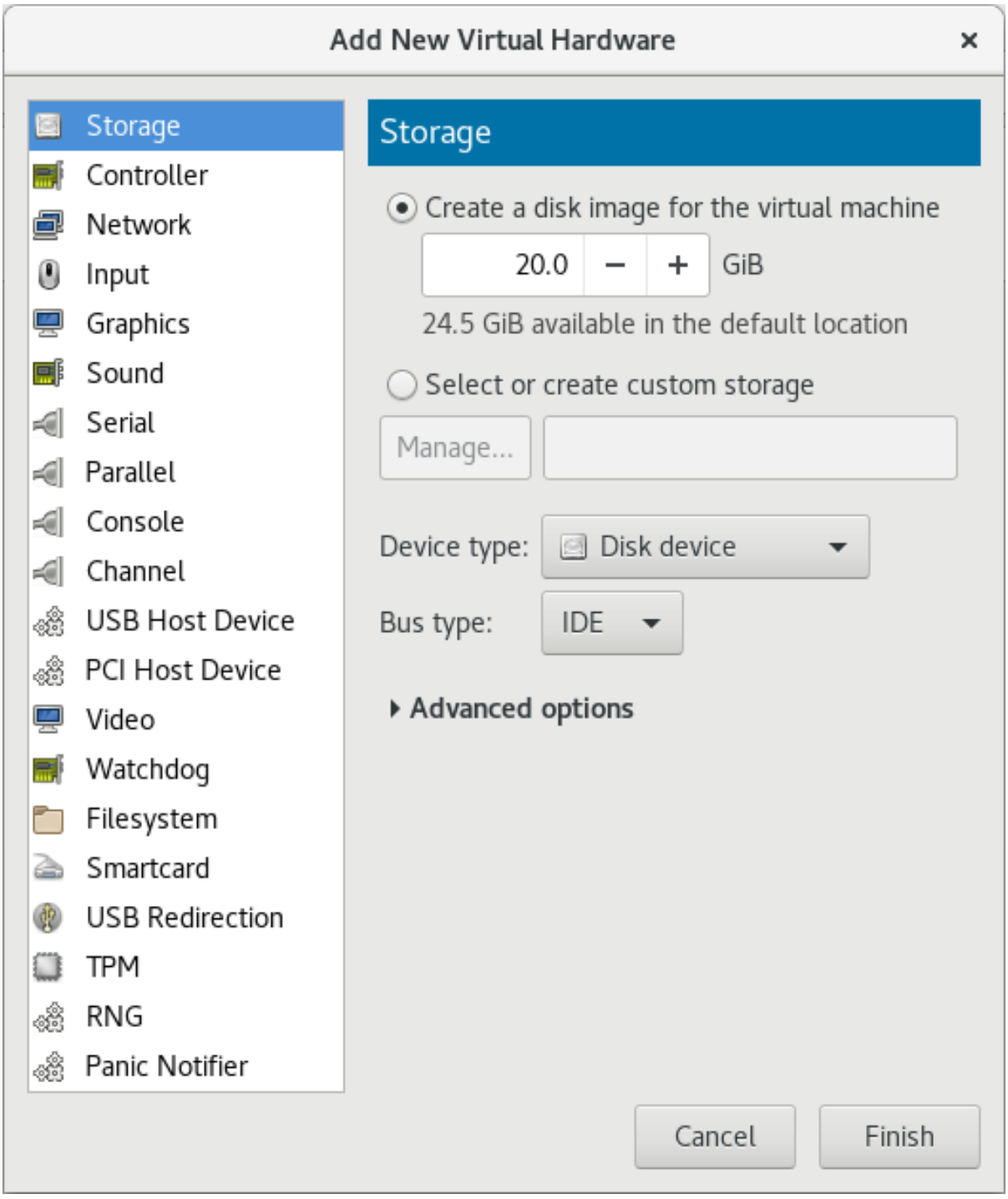
図13.15 Hardware Details ウィンドウ



2. **Add New Virtual Hardware ウィンドウが開きます。**
   Add Hardware をクリックします。Add New Virtual Hardware ウィンドウが表示されます。

   ハードウェアタイプペインで **Storage** が選択されていることを確認します。

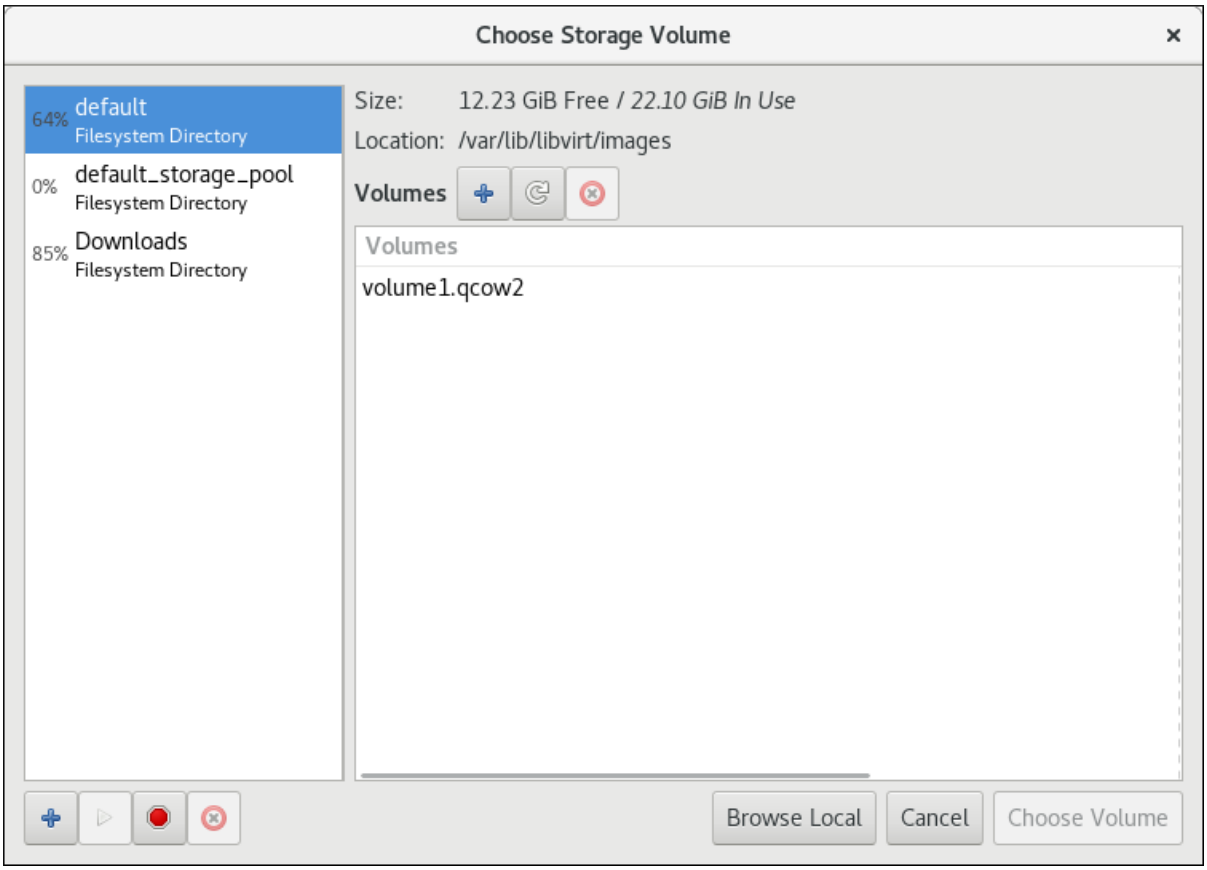
図13.16 Add New Virtual Hardware ウィンドウ



3. **ストレージボリュームのリストを表示します。**
Select or create custom storage オプションのボタンを選択します。

Manage をクリックします。Choose Storage Volume ダイアログが表示されます。

図13.17 Select Storage Volume ウィンドウ



4. **ストレージボリュームを選択します。**
Select Storage Volume ウィンドウの左側にあるリストからストレージプールを選択します。選択したストレージプールのストレージボリュームのリストが、**Volumes** リストに表示されます。

注記

ストレージプールは、Select Storage Volume ウィンドウから作成できます。詳細は、「Virtual Machine Manager を使用したストレージプールの作成」 を参照してください。

Volumes リストからストレージボリュームを選択します。

注記

Select Storage Volume ウィンドウから、ストレージボリュームを作成できます。詳細は、「Virtual Machine Manager を使用したストレージボリュームの作成」 を参照してください。

Choose Volume をクリックします。Select Storage Volume ウィンドウが閉じます。

5. **ストレージボリュームの設定**
Device type リストからデバイスタイプを選択します。利用可能なタイプは、ディスクデバイス、フロッピーデバイス、および LUN パススルーです。

Bus type リストからバスの種類を選択します。利用可能なバスの種類は、選択したデバイスタイプによって異なります。

Cache modeリストからキャッシュモードを選択します。利用可能なキャッシュモードは次のとおりです: Hypervisor default、none、writethrough、writeback、directsync、unsafe

Finish をクリックします。Add New Virtual Hardware ウィンドウが閉じます。

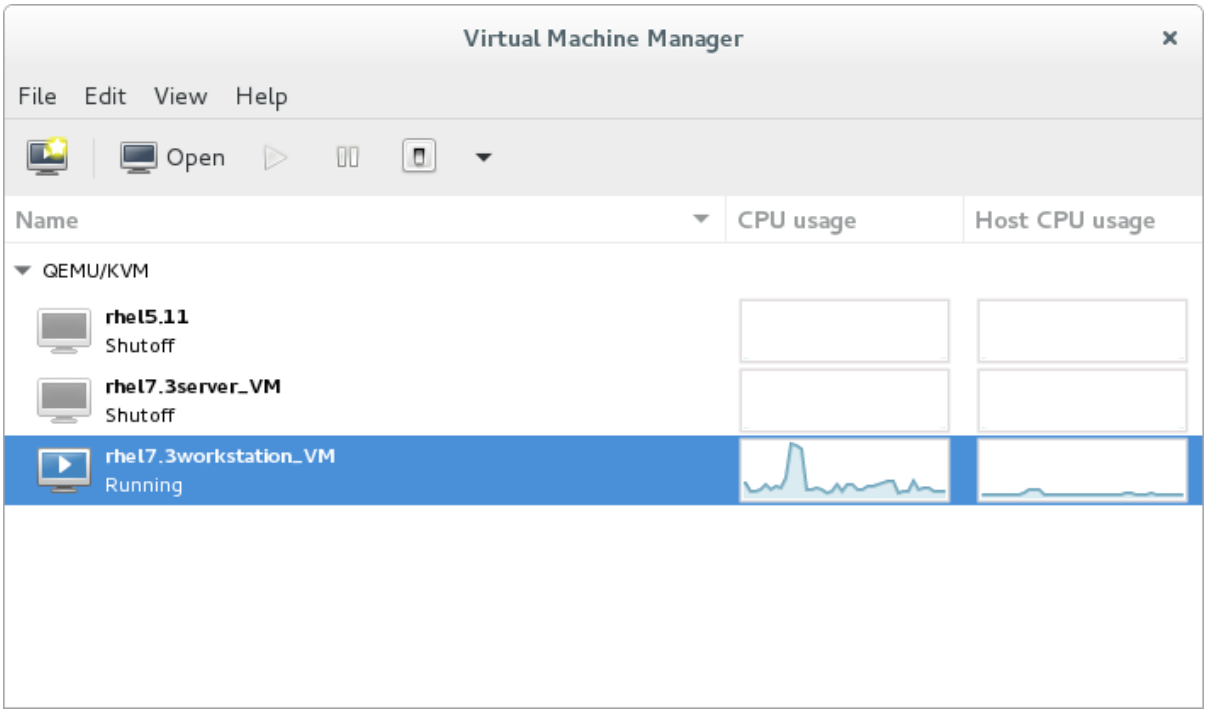
13.3.6.2.2. ゲストへのデフォルトストレージの追加

デフォルトのストレージプールは、**/var/lib/libvirt/images/** ディレクトリーのファイルベースのイメージです。

ゲスト仮想マシンにデフォルトストレージを追加するには、次のコマンドを実行します。

1. **仮想マシンのハードウェアの詳細ウィンドウで Virtual Machine Manager を開きます。**
   root で **virt-manager** コマンドを実行するか、**Applications → System Tools → Virtual Machine Manager** を開き、virt-manager を開きます。

   図13.18 Virtual Machine Manager ウィンドウ

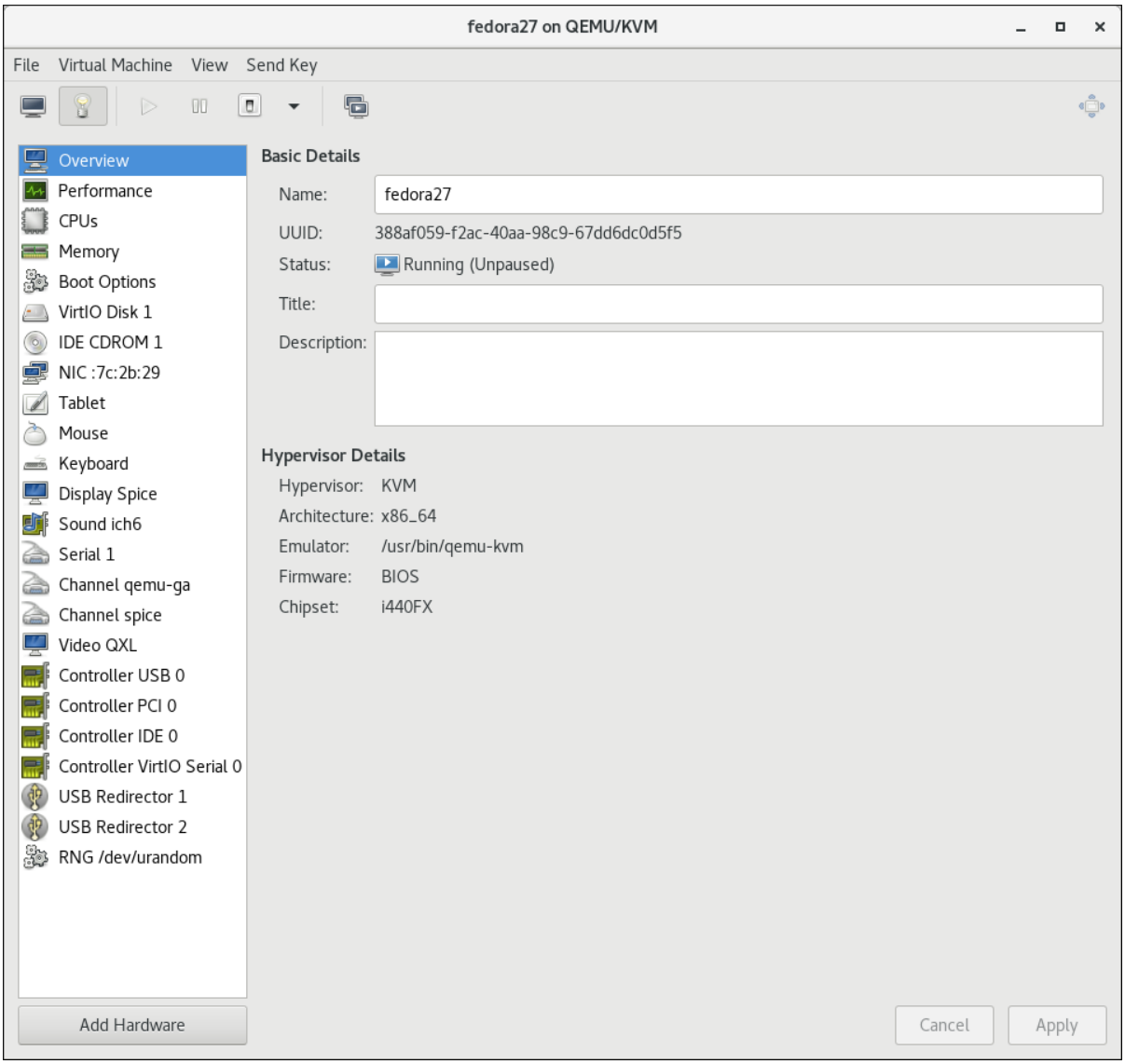


   ストレージボリュームを追加するゲスト仮想マシンを選択します。

   Open をクリックします。仮想マシンのウィンドウが開きます。

    をクリックします。ハードウェアの詳細ウィンドウが表示されます。

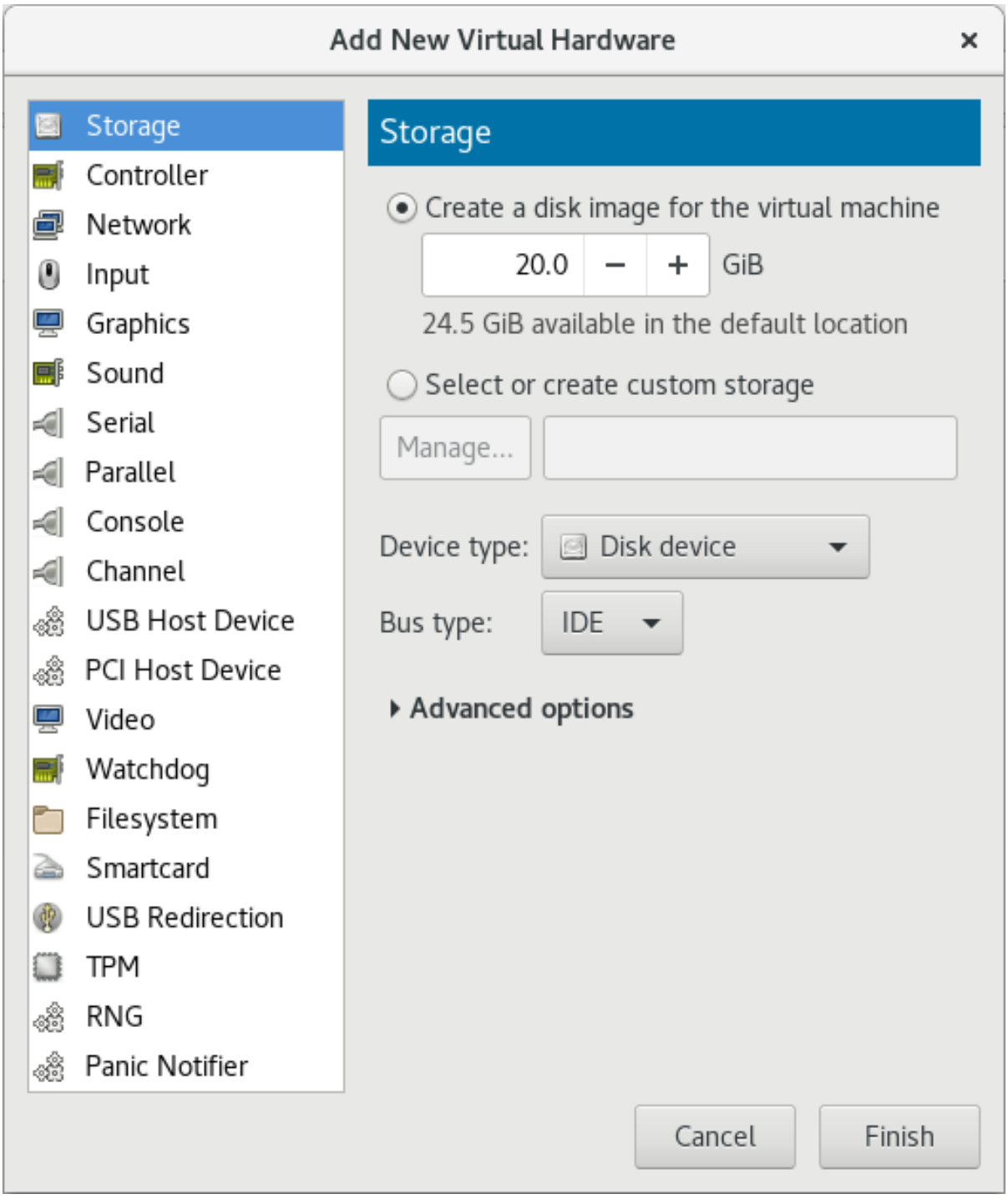
図13.19 Hardware Details ウィンドウ



2. **Add New Virtual Hardware ウィンドウが開きます。**
   Add Hardware をクリックします。Add New Virtual Hardware ウィンドウが表示されます。

   ハードウェアタイプペインで **Storage** が選択されていることを確認します。

図13.20 Add New Virtual Hardware ウィンドウ



3. **ゲスト用のディスクの作成**
Create a disk image for the virtual machine が選択できることを確認します。

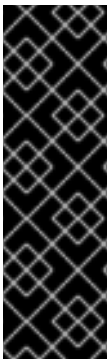
Create a disk image for the virtual machine オプションボタンの下にあるテキストボックスに、作成するディスクのサイズを入力します。

Finish をクリックします。Add New Virtual Hardware ウィンドウが閉じます。

13.3.6.3. ゲストへの SCSI LUN ベースのストレージの追加

ホスト SCSI LUN を完全にゲストに公開する方法は複数あります。SCSI LUN をゲストに公開すると、ゲストの LUN に対して SCSI コマンドを直接実行できるようになります。これは、ゲスト間で LUN を共有したり、ホスト間でファイバーチャネルストレージを共有したりする手段として役立ちます。

SCSI LUN ベースのストレージの詳細は、vHBA-based storage pools using SCSI devices を参照してください。



重要

オプションの ***sgio*** 属性は、非特権 SCSI ジェネリック I/O (SG_IO) コマンドで**device='lun'**ディスクをフィルターにかけるかどうかを制御します。***sgio*** 属性は **'filtered'** または **'unfiltered'** に指定できますが、SG_IO **ioctl** コマンドが永続予約でゲストを通過できるようにするには、**'unfiltered'** に設定する必要があります。

sgio='unfiltered' の設定に加えて、***<shareable>*** 要素を設定してゲスト間で LUN を共有する必要があります。指定されていない場合、***sgio*** 属性のデフォルトは **'filtered'** になります。

<disk> XML 属性の **device='lun'** は、次のゲストディスク設定で有効です。

- **<source dev='/dev/disk/by-{path|id|uuid|label}'/>** の **type='block'**

```
<disk type='block' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/disk/by-path/pci-0000\:04\:00.1-fc-0x203400a0b85ad1d7-lun-0'/>
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```

注記

<source>デバイス名のコロンの前にあるバックスラッシュは必須です。

- **<source protocol='iscsi'... />** の **type='network'**

```
<disk type='network' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source protocol='iscsi' name='iqn.2013-07.com.example:iscsi-net-pool/1'>
    <host name='example.com' port='3260'/>
    <auth username='myuser'>
      <secret type='iscsi' usage='libvirtiscsi'/>
    </auth>
  </source>
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```

- **type='volume'** (iSCSI または NPIV/vHBA のソースプールを SCSI ソースプールとして使用する場合)。

  以下の XML 例は、iSCSI ソースプール (名前は *iscsi-net-pool*) を SCSI ソースプールとして使用するゲストを示しています。

```
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source pool='iscsi-net-pool' volume='unit:0:0:1' mode='host'/>
```

```
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```

注記

<source> タグ内の **mode=** オプションはオプションになりますが、使用する場合は **'direct'** ではなく **'host'** に設定する必要があります。**'host'** に設定すると、libvirt はローカルホストデバイスへのパスを見つけます。**'direct'** に設定すると、libvirt はソースプールのソースホストデータを使用してデバイスへのパスを生成します。

上記の例の iSCSI プール (*iscsi-net-pool*) の設定は、以下のようになります。

```
# virsh pool-dumpxml iscsi-net-pool
<pool type='iscsi'>
  <name>iscsi-net-pool</name>
  <capacity unit='bytes'>11274289152</capacity>
  <allocation unit='bytes'>11274289152</allocation>
  <available unit='bytes'>0</available>
  <source>
    <host name='192.168.122.1' port='3260'/>
    <device path='iqn.2013-12.com.example:iscsi-chap-netpool'/>
    <auth type='chap' username='redhat'>
      <secret usage='libvirtiscsi'/>
    </auth>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <mode>0755</mode>
    </permissions>
  </target>
</pool>
```

iSCSI ソースプールで利用可能な LUN の詳細を確認するには、次のコマンドを実行します。

```
# virsh vol-list iscsi-net-pool
Name                 Path
------------------------------------------------------------------------------
unit:0:0:1           /dev/disk/by-path/ip-192.168.122.1:3260-iscsi-iqn.2013-
12.com.example:iscsi-chap-netpool-lun-1
unit:0:0:2           /dev/disk/by-path/ip-192.168.122.1:3260-iscsi-iqn.2013-
12.com.example:iscsi-chap-netpool-lun-2
```

- **type='volume'** (NPIV/vHBA ソースプールを SCSI ソースプールとして使用する場合)。

  以下の XML の例は、NPIV/vHBA ソースプール (名前は *vhbapool_host3*) を SCSI ソースプールとして使用するゲストを示しています。

```
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source pool='vhbapool_host3' volume='unit:0:1:0'/>
```

```
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```

上記の例の NPIV/vHBA プール (*vhbapool_host3*) の設定は、以下のようになります。

```
# virsh pool-dumpxml vhbapool_host3
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <capacity unit='bytes'>0</capacity>
  <allocation unit='bytes'>0</allocation>
  <available unit='bytes'>0</available>
  <source>
    <adapter type='fc_host' parent='scsi_host3' managed='yes' wwnn='5001a4a93526d0a1'
wwpn='5001a4ace3ee045d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <mode>0700</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</pool>
```

vHBA で利用可能な LUN の詳細を確認するには、次のコマンドを入力します。

```
# virsh vol-list vhbapool_host3
Name                 Path
------------------------------------------------------------------------------
unit:0:0:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0
unit:0:1:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016844602198-lun-0
```

SCSI デバイスで NPIV vHBA を使用する方法は、「SCSI デバイスを使用する vHBA ベースのストレージプール」 を参照してください。

以下の手順は、SCSI LUN ベースのストレージデバイスをゲストに追加する例を示しています。上記のいずれかの***<disk device='lun'>*** ゲストディスク設定は、この方法で割り当てることができます。使用環境に応じて、設定を置き換えます。

手順13.13 SCSI LUN ベースのストレージのゲストへの割り当て

1. <disk> 要素を新しいファイルに書き込んでデバイスファイルを作成し、このファイルに XML 拡張子 (この例では *sda.xml*) を付けて保存します。

```
# cat sda.xml
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source pool='vhbapool_host3' volume='unit:0:1:0'/>
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```

2. *sda.xml* で作成したデバイスをゲスト仮想マシン (*Guest1* など) に関連付けます。

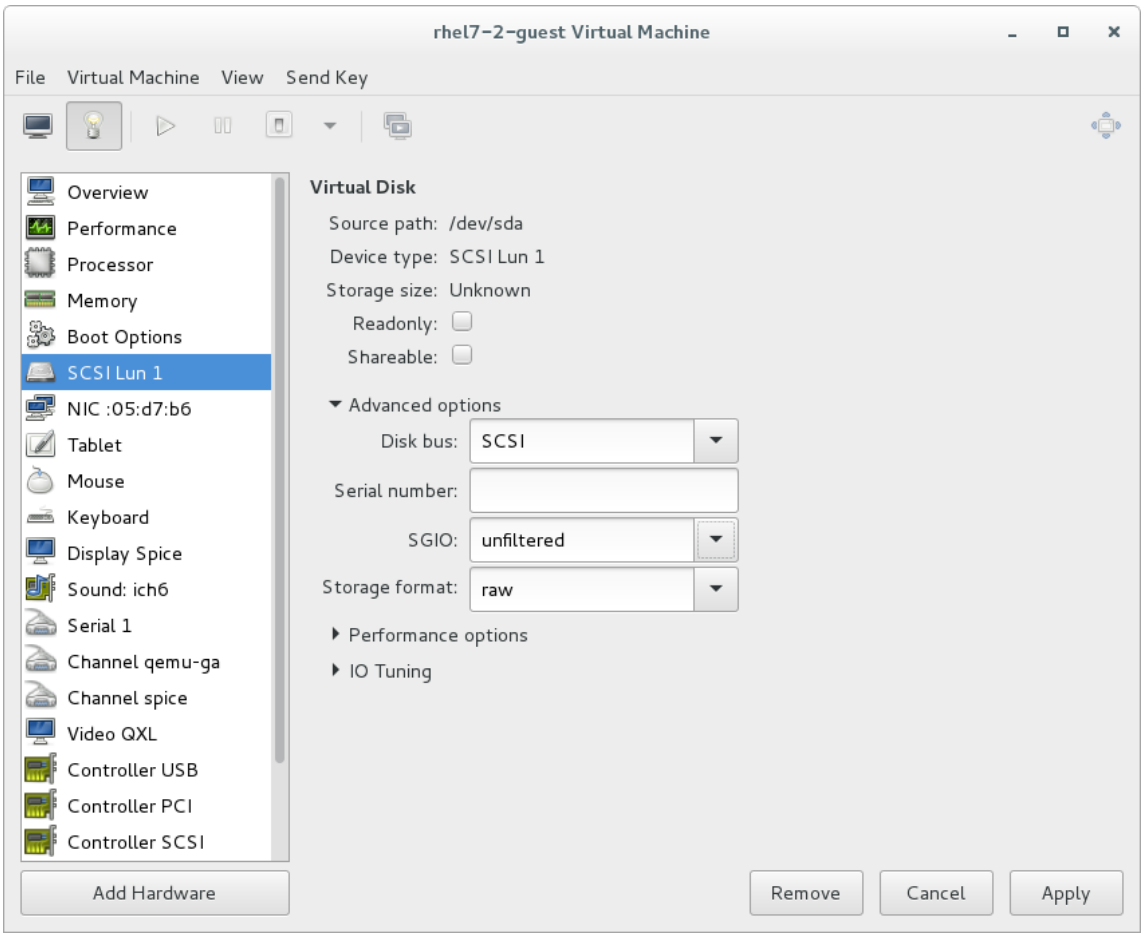
```
# virsh attach-device --config Guest1 ~/sda.xml
```

注記

--config を指定して **virsh attach-device** コマンドを実行すると、ゲストを再起動してデバイスをゲストに永続的に追加する必要があります。また、**--config** の代わりに **--persistent** オプションを使用することもできます。これを使用すると、デバイスをゲストにホットプラグできます。

または、**virt-manager** を使用して、 SCSI LUN ベースのストレージをゲストに接続または設定できます。**virt-manager** を使用してこれを設定する場合は、**Add Hardware** ボタンをクリックし、必要なパラメーターを持つ仮想ディスクを追加するか、このウィンドウで既存の SCSI LUN デバイスの設定を変更します。Red Hat Enterprise Linux 7.2 以降では、**virt-manager** で SGIO 値を設定することもできます。

図13.21 virt-manager を使用した SCSI LUN ストレージの設定



ハードウェア障害後に公開された LUN に再接続する

公開されているファイバーチャネル (FC) LUN への接続が、ハードウェア (ホストバスアダプターなど) の障害により失われた場合は、ハードウェアの障害が修正されても、ゲストで公開されている LUN が、引き続き障害として表示されることがあります。これを防ぐには、**dev_loss_tmo** オプションおよび **fast_io_fail_tmo** カーネルオプションを変更します。

- **dev_loss_tmo** は、SCSI デバイスに障害が発生してから、SCSI レイヤーが障害としてマークされるまでの待ち時間を制御します。タイムアウトを防ぐため、最大値 (**2147483647**) に設定することが推奨されます。

- **fast_io_fail_tmo**は、SCSI デバイスに障害が発生してから I / O にフェイルバックするまでに SCSI 層が待機する時間を制御します。**dev_loss_tmo** がカーネルに無視されないようにするには、このオプションの値を **dev_loss_tmo** より小さい値に設定します。

dev_loss_tmo および **fast_io_fail** を変更するには、以下のいずれかの操作を行います。

- **/etc/multipath.conf** ファイルを編集し、**defaults** セクションに値を設定します。

```
defaults {
...
fast_io_fail_tmo     20
dev_loss_tmo    infinity
}
```

- **dev_loss_tmo** および **fast_io_fail** は、以下のように、FC ホストまたはリモートポートのレベルで設定します。

```
# echo 20 > /sys/devices/pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-
0/fc_remote_ports/rport-1:0-0/fast_io_fail_tmo
# echo 2147483647 > /sys/devices/pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-
0/fc_remote_ports/rport-1:0-0/dev_loss_tmo
```

dev_loss_tmo および **fast_io_fail** の新しい値が有効であることを確認するには、以下のコマンドを使用します。

```
# find /sys -name dev_loss_tmo -print -exec cat {} \;
```

パラメーターが正しく設定されていれば、出力は以下のようになります (**pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-0/fc_remote_ports/rport-1:0-0** の代わりに、適切なデバイスが使用されます)。

```
# find /sys -name dev_loss_tmo -print -exec cat {} \;
...
/sys/devices/pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-0/fc_remote_ports/rport-1:0-
0/dev_loss_tmo
2147483647
...
```

13.3.6.4. ゲスト仮想マシンで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 デバイスではゲスト仮想マシンにコントローラーが存在する必要があります。本セクションでは、仮想 SCSI コントローラー ("Host Bus Adapter" または HBA とも呼ばれます) 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に必要な手順を詳細に説明します。

手順13.14 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、以前から存在している SCSI コントローラーを検索します。

   ```
   # 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、このコマンドにより、次のような行が 1 つ以上出力されます。

   ```
   -
   ```

```
<controller type='scsi' model='virtio-scsi' index='0'/>
```

2. 前の手順でデバイスコントローラーが表示されなかった場合は、以下の手順に従って、新しいファイルにデバイスコントローラーの説明を作成し、仮想マシンに追加します。

   a. 新しいファイルに **<controller>** 要素を書き込んでデバイスコントローラーを作成し、このファイルを XML 拡張子で保存します。たとえば、**virtio-scsi-controller.xml** となります。

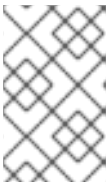
   ```
   <controller type='scsi' model='virtio-scsi'/>
   ```

   b. **virtio-scsi-controller.xml** で作成したデバイスコントローラーをゲスト仮想マシン (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と同じように動作します。詳細は、「ゲストへのストレージデバイスの追加」 を参照してください。

3. 新しい SCSI ディスクまたは CD-ROM を追加します。新規ディスクは、「ゲストへのストレージデバイスの追加」 の方法を使用して追加できます。SCSI ディスクを作成する場合は、*sd* で始まるターゲットデバイス名を指定します。



注記

各コントローラーでサポートされる制限は 1024 virtio-scsi ディスクですが、ホストで利用可能なその他のリソース (ファイル記述子など) が使い切られ、ディスクが少なくなる可能性があります。

詳細は、次の Red Hat Enterprise Linux 6 ホワイトペーパーを参照してください: The next-generation storage interface for the Red Hat Enterprise Linux Kernel Virtual Machine: virtio-scsi

```
# virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.img sdb --cache none
```

ゲスト仮想マシン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で新しいディスクがすぐに検出されない場合があります。『Red Hat Enterprise Linux Storage Administration Guide』 の手順に従います。

13.3.7. ゲストからのストレージデバイスの削除

virsh または **Virtual Machine Manager** を使用して、仮想ゲストマシンからストレージデバイスを削除できます。

13.3.7.1. virsh を使用した仮想マシンからのストレージの削除

以下の例では、Guest1 仮想マシンから vdb ストレージボリュームを削除します。

```
# virsh detach-disk Guest1 vdb
```

13.3.7.2. Virtual Machine Manager を使用した仮想マシンからのストレージの削除

手順13.15 Virtual Machine Manager を使用した仮想マシンからのストレージの削除

Virtual Machine Managerを使用してゲスト仮想マシンからストレージを削除するには、次のコマンドを実行します。

1. **仮想マシンのハードウェアの詳細ウィンドウで Virtual Machine Manager を開きます。**
   root で **virt-manager** コマンドを実行するか、**Applications → System Tools → Virtual Machine Manager** を開き、virt-manager を開きます。

   ストレージデバイスを削除するゲスト仮想マシンを選択します。

   Open をクリックします。仮想マシンのウィンドウが開きます。

    をクリックします。ハードウェアの詳細ウィンドウが表示されます。

2. **ゲスト仮想マシンからストレージを削除します。**
   ハードウェアの詳細ペインの左側にあるハードウェアのリストから、ストレージデバイスを選択します。

   削除 をクリックします。確認ダイアログが表示されます。

   Yes をクリックします。ストレージがゲスト仮想マシンから削除されます。

第14章 QEMU-IMG の使用

qemu-img コマンドラインツールは、KVM が使用するさまざまなファイルシステムのフォーマット、修正、および検証に使用されます。qemu-img のオプションと使用法は、以降のセクションで強調表示されています。

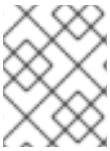
警告

qemu-img を使用して、実行中の仮想マシンなど、使用中のイメージを修正しないでください。イメージが破壊される可能性があります。また、別のプロセスにより変更されているイメージをクエリーすると、状態が不整合になる可能性があることに注意してください。

14.1. ディスクイメージの確認

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実行する場合。

```
# qemu-img check [-f format] imgname
```



注記

整合性チェックには、選択した形式のグループのみが対応します。対応する形式には、*qcow2*、*vdi*、*vhdx*、*vmdk*、および *qed* が含まれます。

14.2. イメージへの変更の確定

指定したイメージファイル (*imgname*) に記録された変更を、**qemu-img commit** コマンドでファイルのベースイメージにコミットします。必要に応じて、ファイル形式タイプ (*fmt*) を指定します。

```
# qemu-img commit [-f fmt] [-t cache] imgname
```

14.3. イメージの比較

指定した 2 つのイメージファイル (*imgname1* および *imgname2*) の内容を、**qemu-img compare** コマンドで比較します。必要に応じて、ファイルの形式タイプ (*fmt*) を指定します。イメージの形式と設定は異なります。

一方のイメージの末尾以後の領域に、大きい方のイメージに割り当てられていないセクターまたはゼロになっているセクターのみが含まれる場合、デフォルトでは、異なるサイズのイメージは同一と見なされます。また、一方のイメージにセクターが割り当てられておらず、他方のセクターに含まれるバイト数がゼロの場合は、等しいと評価されます。**-s** オプションを指定しても、イメージのサイズが異なる場合、またはセクターが 1 つのイメージに割り当てられており、2 番目のイメージには割り当てられていない場合に、イメージは同一とは見なされません。

```
# qemu-img compare [-f fmt] [-F fmt] [-p] [-s] [-q] imgname1 imgname2
```

qemu-img compare コマンドは、以下の終了コードのいずれかで終了します。

- **0** - イメージは同一です。
- **1** - イメージが異なります。
- **2** - イメージの 1 つを開いている際にエラーが発生しました。
- **3** - セクター割り当ての確認中にエラーが発生しました。
- **4** - データの読み取り中にエラーが発生しました。

14.4. イメージのマッピング

qemu-img map コマンドを使用すると、指定したイメージファイル (*imgname*) のメタデータと、そのバッキングファイルチェーンをダンプできます。ダンプは、(*imgname*) 内のすべてのセクターの割り当て状態を示し、最上位のファイルをバッキングファイルチェーンで割り当てています。必要に応じて、ファイル形式タイプ (*fmt*) を指定します。

```
# qemu-img map [-f fmt] [--output=fmt] imgname
```

出力形式には、**human** 形式と **json** 形式の 2 つがあります。

14.4.1. human 形式

デフォルト形式 (**human**) は、ゼロ以外で割り当てられたファイル部分のみをダンプします。この出力により、データの読み取り元となるファイルと、ファイル内のオフセットが識別されます。各行には、4 つのフィールドが含まれます。以下は、出力の例になります。

```
Offset          Length          Mapped to       File
0               0x20000         0x50000         /tmp/overlay.qcow2
0x100000        0x10000         0x95380000      /tmp/backing.qcow2
```

最初の行は、イメージのオフセット 0 から始まる **0x20000** (131072) バイトが、オフセット **0x50000** (327680) から始まる **tmp/overlay.qcow2** (raw 形式で開かれた) で利用可能であることを示しています。圧縮、暗号化、または raw 形式で利用できないデータは、**human** 形式が指定されているとエラーを発生させます。

注記

ファイル名には改行文字を使用できます。そのため、スクリプトで**human** 形式の出力を解析しても安全ではありません。

14.4.2. json 形式

json を指定すると、JSON 形式の辞書の配列を返します。**human** オプションに記載されている情報のほかに、以下の情報が含まれます。

- **data** - セクターにデータが含まれるかどうかを示すブール値フィールドです。
- **zero** - データがゼロとして読み取られることが知られているかどうかを示すブール値フィールドです。
- **depth** - **filename** のバッキングファイルの深さです。

注記

json オプションが指定されている場合、**offset** フィールドはオプションになります。

qemu-img map コマンドとその他のオプションの詳細は、関連する man ページを参照してください。

14.5. イメージの修正

イメージファイルのイメージ形式固有のオプションを修正します。必要に応じて、ファイル形式タイプ (*fmt*) を指定します。

```
# qemu-img amend [-p] [-f fmt] [-t cache] -o options filename
```

注記

この操作は、qcow2 ファイル形式でのみ対応しています。

14.6. 既存のイメージを別の形式に変換

convert オプションは、認識されたイメージ形式を、別のイメージ形式に変換するために使用されます。使用可能な形式のリストは、「対応する qemu-img 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示し (すべてのコマンドではなく任意)、***-S*** フラグは、ディスクイメージに含まれる *sparse file* の作成を許可します。ゼロのみを含む (つまり、何も含まない) 物理ブロックを除いて、あらゆる目的のスパースファイルは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認識すると、たとえ実際にはディスクを使用していなくても、存在しているものとして扱われ、実際のディスク領域を消費します。ゲスト仮想マシン用のディスクを作成する場合に特に役立ちます。これにより、ディスクに必要なディスク領域よりもはるかに多くのディスク領域が使用されるようになります。たとえば、10Gb のディスクイメージで -S を 50Gb に設定すると、実際には 10Gb しか使用されていなくても、そのディスク領域の 10Gb は 60Gb に見えます。

形式 ***output_format*** を使用して、ディスクイメージの***filename***を、ディスクイメージの***output_filename***に変換します。ディスクイメージは、***-c*** オプションで圧縮するか、**-o encryption** を設定して ***-o*** オプションで暗号化できます。***-o*** パラメーターで使用できるオプションは、選択した形式により異なることに注意してください。

qcow2 と *qcow2* 形式のみが暗号化または圧縮をサポートします。**qcow2** 暗号化は、セキュアな 128 ビット鍵で AES 形式を使用します。**qcow2** 圧縮は読み取り専用であるため、圧縮したセクターが **qcow2** 形式から変換されると、非圧縮データとして新しい形式に書き込まれます。

イメージの変換は、**qcow** または **cow** など、サイズを大きくできる形式を使用する場合に、小さなイメージを取得する場合にも役立ちます。空のセクターは、宛先イメージから検出され、抑制されます。

14.7. 新しいイメージまたはデバイスの作成とフォーマット

サイズが ***size*** で形式が ***format*** の新しいディスクイメージ*filename*を作成します。

```
# qemu-img create [-f format] [-o options] filename [size]
```

ベースイメージが **-o backing_file=*filename*** で指定されている場合は、イメージ自体とベースイメージの違いのみが記録されます。バッキングファイルは、**commit** コマンドを使用しない限り変更されません。この場合、サイズの指定は必要ありません。

14.8. イメージ情報の表示

info パラメーターは、ディスクイメージの*filename*に関する情報を表示します。**info** オプションの形式は、以下のとおりです。

```
# qemu-img info [-f format] filename
```

このコマンドは、多くの場合、表示されているサイズとは異なるディスクに予約されているサイズを検出するために使用されます。スナップショットがディスクイメージに保存されている場合は、そのスナップショットも表示されます。このコマンドは、たとえば、ブロックデバイスの qcow2 イメージが使用している領域を表示します。これは、**qemu-img** を実行して行います。使用中のイメージが、**qemu-img check** コマンドで **qemu-img info** コマンドの出力と一致するイメージ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
```

14.9. イメージのバックアップファイルの再生

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の形式が *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が *backing_format* に変更されます。

注記

バッキングファイル (リベース) の変更に対応するのは、*qcow2* 形式のみです。

rebase が動作できるモードには、**safe** と **unsafe** の 2 つがあります。

safe モードは、デフォルトで使用され、実際のリベース操作を実行します。新しいバッキングファイルは古いファイルとは異なる場合があり、**qemu-img rebase** は、ゲストの仮想マシンで認識される *filename* の内容を変更せずに保持します。これを実現するため、*filename* の *backing_file* と古いバッキングファイルとで異なるクラスターは、すべて *filename* にマージされてから、バッキングファイルを変更します。

safe モードはイメージの変換に相当する費用のかかる動作であることに注意してください。古いバッキングファイルは、正常に完了するために必要です。

unsafe モードは、*-u* オプションが **qemu-img rebase** に渡される場合に使用されます。このモードでは、ファイルの内容を確認せずに、バッキングファイルの名前と *filename* の形式のみが変更されます。新しいバッキングファイルが正しく指定されていることを確認してください。指定されていない場合、イメージのゲストに表示されるコンテンツが破損します。

このモードは、バッキングファイルの名前変更や移動に役立ちます。アクセス可能な古いバッキングファイルがなくても使用できます。たとえば、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を修正するために使用できます。

14.10. ディスクイメージのサイズ変更

サイズ *size*で作成されたかのように、ディスクイメージの*filename*を変更します。raw 形式のイメージのみが両方向にサイズ変更できますが、qcow2 イメージは拡大できても縮小はできません。

次のコマンドを使用して、ディスクイメージの*filename*のサイズを *size* バイトに設定します。

```
# qemu-img resize filename size
```

また、ディスクイメージの現在のサイズを基準にしてサイズを変更することもできます。現在のサイズに相対的なサイズを指定するには、バイト数の前に **+** を付けて拡大するか、**-** を付けてディスクイメージのサイズをそのバイト数だけ縮小します。ユニットの接尾辞を追加すると、イメージのサイズをキロバイト (K)、メガバイト (M)、ギガバイト (G)、またはテラバイト (T) で設定できます。

```
# qemu-img resize filename [+|-]size[K|M|G|T]
```

警告

このコマンドを使用してディスクイメージを縮小する前に、仮想マシン自体の内部でファイルシステムとパーティションツールを使用して、割り当てられたファイルシステムとパーティションサイズを適宜減らす**必要**があります。そうしないと、データが失われることになります。

このコマンドを使用してディスクイメージを拡張したら、ファイルシステムと仮想マシン内のパーティションツールを使用して、デバイス上の新しい領域の使用を実際に開始する必要があります。

14.11. スナップショットのリスト表示、作成、適用、および削除

qemu-img snapshot コマンドとは異なるパラメーターを使用して、指定したイメージ (*filename*) の既存のスナップショット (*snapshot*) のリスト表示、適用、作成、または削除を行うことができます。

```
# qemu-img snapshot [ -l | -a snapshot | -c snapshot | -d snapshot ] filename
```

指定できる引数は以下のとおりです。

- **-l** は、指定したディスクイメージに関連付けられているすべてのスナップショットをリスト表示します。
- apply オプション **-a** は、ディスクイメージ (*filename*) を、以前保存した*スナップショット*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、指定したスナップショットを削除します。

14.12. 対応する QEMU-IMG 形式

qemu-img コマンドのいずれかで形式が指定されている場合、次の形式タイプを使用できます。

- **raw** - raw ディスクイメージ形式 (デフォルト)これは、ファイルベースで最も高速な形式になります。ファイルシステムがホール (ext2 や ext3 など) に対応している場合は、書き込まれたセクターのみが領域を確保します。**qemu-img info** を使用して、イメージが使用する実際のサイズを取得するか、Unix/Linux の **ls -ls** を取得します。raw イメージは最適なパフォーマンスを提供しますが、raw イメージでは非常に基本的な機能しか利用できません。たとえば、スナップショットは利用できません。

- **qcow2** - QEMU イメージ形式。最も汎用性が高く、機能セットが最適な形式です。これを使用して、オプションの AES 暗号化、zlib ベースの圧縮、複数の仮想マシンスナップショットへの対応、および小規模イメージを取得します。これは、ホールに対応していないファイルシステムで役に立ちます 。この拡張機能セットはパフォーマンスに影響を与えることに注意してください。

  ゲスト仮想マシンまたはホスト物理マシンでの実行には上記の形式のみを使用できますが、**qemu-img** は、**raw**、または **qcow2** 形式に変換するために、以下の形式も認識してサポートします。イメージの形式は、通常、自動的に検出されます。この形式を **raw** または **qcow2** に変換することに加えて、**raw** または **qcow2** から元の形式に戻すことができます。Red Hat Enterprise Linux 7 に同梱される 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 に同梱されている形式は 0.10 です。イメージファイルは、以前のバージョンの qcow2 に戻すことができます。使用しているバージョンを確認するには、**qemu-img info qcow2** ***[imagefilename.img]*** コマンドを実行します。qcow バージョンを変更するには、「target 要素の設定」 を参照してください。

- **bochs** - Bochs ディスクイメージ形式。

- **cloop** - Linux Compressed Loop イメージ。Knoppix CD-ROM などにある直接圧縮 CD-ROM イメージを再利用する場合にのみ役立ちます。

- **cow** - User Mode Linux Copy On Write イメージ形式。 **cow** 形式は、以前のバージョンとの互換性のためにのみ同梱されています。

- **dmg** - Mac ディスクイメージ形式。

- **nbd** - ネットワークブロックデバイス。

- **parallels** - パラレル仮想化ディスクイメージ形式。

- **qcow** - 古い QEMU イメージ形式。古いバージョンとの互換性にのみ含まれます。

- **qed** - 古い QEMU イメージ形式。古いバージョンとの互換性にのみ含まれます。

- **vdi** - Oracle VM VirtualBox ハードディスクイメージ形式。

- **vhdx** - Microsoft Hyper-V virtual hard disk-X ディスクイメージ形式。

- **vmdk** - VMware 3 および 4 互換のイメージ形式。

- **vfat** - Virtual VFAT ディスクイメージ形式。

第15章 KVM の移行

本章では、KVM ハイパーバイザーを実行しているホストの物理マシンから、別のホストにゲスト仮想マシンを移行する方法を説明します。仮想マシンはハードウェアで直接実行するのではなく、仮想化環境で実行するため、ゲストの移行が可能になります。

15.1. 移行の定義と利点

移行は、ゲスト仮想マシンのメモリーと仮想デバイスの状態を移行先ホストの物理マシンに送信することで機能します。ネットワーク上の共有ストレージを使用して、移行するゲストのイメージを保存することが推奨されます。仮想マシンを移行する場合は、共有ストレージに libvirt が管理する ストレージプール を使用することも推奨されます。

移行は、**ライブ** (実行中) ゲストおよび **非ライブ** (シャットダウン) ゲストの両方で実行できます。

*live migration*では、ゲスト仮想マシンが移行元ホストマシンで実行し続け、ゲストのメモリーページが移行先ホストマシンに転送されます。移行中、KVM は、すでに転送されたページの変更についてソースを監視し、すべての初期ページが転送されたときにこれらの変更の転送を開始します。KVM は、移行中の転送速度も推定するため、転送するデータの残りの量が一定の設定可能な期間 (デフォルトでは 10 ミリ秒) に達すると、KVM が元のゲスト仮想マシンを一時停止し、残りのデータを転送して、移行先ホストの物理マシンで同じゲスト仮想マシンを再開します。

一方、*non-live migration* (オフライン移行) はゲスト仮想マシンを一時停止し、ゲストのメモリーを移行先ホストマシンにコピーします。その後、移行先ホストマシンでゲストが再開し、移行元ホストマシンで使用していたゲストのメモリーが解放されます。このような移行を完了するためにかかる時間は、ネットワークの帯域幅と遅延にのみ依存します。ネットワークが頻繁に使用されているか、帯域幅が狭い場合、移行にははるかに長い時間がかかります。元のゲスト仮想マシンが、KVM が宛先ホスト物理マシンに転送できる速度よりも速くページを変更する場合は、ライブマイグレーションが完了しないため、オフラインマイグレーションを使用する必要があります。

移行は、以下の場合に役立ちます。

負荷分散

ゲスト仮想マシンは、ホストマシンが過負荷になった場合、または別のホストマシンが十分に活用されていない場合に、使用率の低いホスト物理マシンに移動できます。

ハードウェアの非依存性

ホストの物理マシンでハードウェアデバイスのアップグレード、追加、または削除が必要になった場合は、ゲスト仮想マシンを別のホストの物理マシンに安全に再配置できます。つまり、ゲスト仮想マシンでは、ハードウェアの改善のためのダウンタイムが発生しません。

エネルギー節約

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

地理的な移行

仮想マシンは、待ち時間を短縮するため、または他の理由で必要な場合に、別の場所に移動できます。

15.2. 移行の要件および制限

KVM の移行を使用する前に、システムが移行の要件を満たしていることを確認し、その制限事項について理解しておく必要があります。

移行の要件

- 次のいずれかのプロトコルを使用して、共有ストレージにインストールされたゲスト仮想マシン。
  - 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 アダプターおよび 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
- **libvirtd** サービスが有効で、実行していることを確認します。

```
# systemctl enable libvirtd.service
# systemctl restart libvirtd.service
```

- 効果的に移行する機能は、**/etc/libvirt/libvirtd.conf** ファイル内のパラメーター設定により異なります。このファイルを編集するには、以下の手順を使用します。

手順15.1 libvirtd.conf の設定

1. **libvirtd.conf** を開くには、root で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
```

- 移行プラットフォームとバージョンは、表15.1「ライブマイグレーションの互換性」 に対してチェックする必要があります。
- 共有ストレージメディアをエクスポートする別のシステムを使用してください。ストレージは、移行に使用される 2 台のホスト物理マシンのいずれにも存在しないようにする必要があります。
- 共有ストレージは、移行元システムと移行先システムの同じ場所にマウントする必要があります。マウントするディレクトリー名は同じである必要があります。別のパスを使用してイメージを保持することもできますが、推奨されません。virt-manager を使用して移行を実行する場合は、パス名が同じである必要があります。virsh を使用して移行を実行する場合は、**--xml** オプションまたは pre-hooks で別のネットワーク設定とマウントディレクトリーを使用できます。pre-hooks の詳細は libvirt upstream documentation を参照し、XML オプションの詳細は23章 *ドメイン XML の操作* を参照してください。

- パブリックブリッジ + タップネットワーク内の既存のゲスト仮想マシンで移行を試行する場合、移行元ホストマシンと移行先ホストマシンが同じネットワークに存在する必要があります。この手順を行わないと、ゲストの仮想マシンネットワークが移行後に動作しません。

移行の制限

- KVM に基づく仮想化技術を使用する Red Hat Enterprise Linux でゲスト仮想マシンの移行を使用する場合には、以下の制限があります。
  - ポイントツーポイント移行 – 送信元ハイパーバイザーから宛先ハイパーバイザーを指定するには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
  - ターゲットの決定は手動でのみ行うことができます。
  - ストレージの移行は Red Hat Enterprise Linux 7™ ではライブで実行できませんが、ゲスト仮想マシンの電源がオフのときにストレージを移行できます。ライブストレージの移行は、Red Hat Virtualization™ で利用できます。詳細は、サービス担当者にお問い合わせください。

注記

virtio デバイスを含むゲストマシンを移行する場合は、いずれかのプラットフォームの virtio デバイスのベクトル数を 32 以下に設定してください。詳細は、「Devices」 を参照してください。

15.3. ライブマイグレーションと RED HAT ENTERPRISE LINUX バージョンの互換性

ライブマイグレーションは、表15.1「ライブマイグレーションの互換性」 に示すようにサポートされています。

表15.1 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	ライブ移行のサポート	注記
前方	メジャーリリース	6.5+ → 7.x	フルサポート	問題がある場合は報告する必要があります
後方	メジャーリリース	7.x → 6.y	サポート対象外	
前方	マイナーリリース	7.x → 7.y (7.0 → 7.1)	フルサポート	問題がある場合は報告する必要があります
後方	マイナーリリース	7.y → 7.x (7.1 → 7.0)	フルサポート	問題がある場合は報告する必要があります

移行に関する問題のトラブルシューティング

- **移行プロトコルの問題** - 後方移行が "unknown section error" で終了する場合は、一時的なエラーである可能性があるため、移行プロセスを繰り返すことで問題を修復できます。そうでない場合は、問題を報告してください。

- **オーディオデバイスに関する問題** - Red Hat Enterprise Linux 6.x から Red Hat Enterprise Linux 7.y への移行時に、es1370 オーディオカードに対応しなくなった点に注意してください。代わりに ac97 オーディオカードを使用してください。

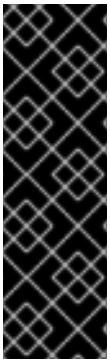
- **ネットワークカードに関する問題** - Red Hat Enterprise Linux 6.x から Red Hat Enterprise Linux 7.y に移行する際に、pcnet ネットワークカードおよび ne2k_pci ネットワークカードに対応しなくなった点に注意してください。代わりに virtio-net ネットワークデバイスを使用してください。

ネットワークストレージの設定

共有ストレージを設定し、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。

あるいは、「共有ストレージの例: 単純な移行のための NFS」 の NFS の例を使用します。

15.4. 共有ストレージの例: 単純な移行のための NFS



重要

この例では、NFS を使用して、ゲスト仮想マシンのイメージを別の KVM ホストの物理マシンと共有します。大規模なインストールでは実用的ではありませんが、移行技術のみを実証することが推奨されます。この例を、複数のゲスト仮想マシンの移行または実行に使用しないでください。また、***synch*** パラメーターが有効になっ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ために必要です。

大規模なデプロイメントでは、iSCSI ストレージの方が適しています。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。

NFS の設定、IP テーブルの開放、およびファイアウォールの設定の詳細は、『Red Hat Linux Storage Administration Guide』 を参照してください。

KVM では対応していないため、NFS ファイルのロック機能を使用しない。

1. **libvirt イメージディレクトリーのエクスポート**
   移行には、移行ターゲットシステムとは別のシステムにストレージを置く必要があります。別のシステムでは、デフォルトのイメージディレクトリーを **/etc/exports** ファイルに追加して、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)
   ```

   必要に応じて ***hostname*** パラメーターを変更します。

2. **NFS の起動**

   a. NFS パッケージがインストールされていない場合は、インストールします。

   ```
   # yum install nfs-utils
   ```

   b. **iptable** (2049 など) の NFS のポートが開いていることを確認し、 **/etc/hosts.allow** ファイルに NFS を追加します。

c. NFS サービスを開始します。

```
# systemctl start nfs-server
```

3. **移行元と移行先の共有ストレージをマウントします。**
移行元システムおよび移行先システムで、**/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

移行元ホストの物理マシンのディレクトリーは、移行先ホストの物理マシンのディレクトリーとまったく同じである必要があります。これは、すべてのタイプの共有ストレージに適用されます。ディレクトリーが同じである必要があります。そうでないと、virt-manager による移行が失敗します。

15.5. VIRSH を使用した KVM のライブ移行

virsh コマンドを使用すると、ゲスト仮想マシンを別のホスト物理マシンに移行できます。**migrate** コマンドは、以下の形式のパラメーターを受け付けます。

```
# virsh migrate --live GuestName DestinationURL
```

ライブマイグレーションが必要ない場合に、*--live* オプションが削除される可能性がある点に注意してください。追加オプションは、「virsh migrate コマンドの追加オプション」 にリスト表示されています。

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。

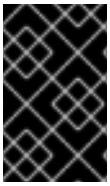
DestinationURL パラメーターは、宛先ホスト物理マシンの接続 URL です。移行先システムで同じバージョンの Red Hat Enterprise Linux を実行し、同じハイパーバイザーを使用し、**libvirt** を実行している必要があります。

注記

通常の移行と peer2peer の移行の ***DestinationURL*** パラメーターのセマンティックは異なります。

- 通常の移行 - ***DestinationURL*** は、移行元ゲスト仮想マシンから見た、移行先ホストの物理マシンの URL です。
- peer2peer 移行: ***DestinationURL*** は、ソースホスト物理マシンから見たターゲットホスト物理マシンの URL です。

コマンドを入力すると、インストール先システムの root パスワードを求められます。



重要

移行を成功させるには、名前解決が両方 (移行元と移行先) で機能している必要があります。両側が互いを見つけられるようにする必要があります。名前解決が機能していることを確認するために、一方を他方に ping できることを確認してください。

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** に移行します。使用環境に合わせて、ホストの物理マシン名を変更します。この例では、**guest1-rhel6-64** という名前の仮想マシンを移行します。

この例では、共有ストレージを完全に設定し、すべての前提条件 (ここでは 移行の要件) を満たしていることを前提としています。

1. **ゲスト仮想マシンが実行していることを確認します。**
   移行元システムで、**host1.example.com** を実行し、**guest1-rhel6-64** が実行していることを確認します。

   ```
   [root@host1 ~]# virsh list
   Id Name                 State
   ----------------------------------
    10 guest1-rhel6-64     running
   ```

2. **ゲスト仮想マシンの移行**
   次のコマンドを実行して、ゲスト仮想マシンを移行先 **host2.example.com** にライブ移行します。リンク先の URL の末尾に **/system** を追加し、フルアクセスが必要であることを libvirt に指示します。

   ```
   # virsh migrate --live guest1-rhel7-64 qemu+ssh://host2.example.com/system
   ```

   コマンドを入力すると、インストール先システムの root パスワードを求められます。

3. **Wait**
   負荷やゲスト仮想マシンのサイズによっては、移行に時間がかかる場合があります。**virsh** はエラーのみを報告します。ゲスト仮想マシンは、完全に移行するまで、移行元ホストの物理マシンで実行し続けます。

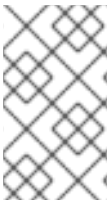
4. **ゲスト仮想マシンが移行先ホストに到達していることを確認する**
   移行先システムから、**host2.example.com**、**guest1-rhel7-64** が実行していることを確認します。

   ```
   [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 ソケット、UNIX ソケット、SSH、暗号化されていない TCP など、さまざまなネットワーク方式をサポートしています。他の方法の使用方法は、18章 *ゲストのリモート管理* を参照してください。



注記

実行中でないゲスト仮想マシンは、次のコマンドを使用して移行できます。

```
# virsh migrate --offline --persistent
```

15.5.1. virsh を使用した移行に関する追加のヒント

複数の同時ライブマイグレーションを実行できます。各移行は、個別のコマンドシェルで実行されます。ただし、これは慎重に行ってください。各移行インスタンスでは、双方 (ソースおよびターゲット) から 1 つの MAX_CLIENT を使用するため、注意して計算を行う必要があります。デフォルト設定は 20 であるため、設定を変更せずに 10 インスタンスを実行できます。設定を変更する必要がある場合は、手順15.1「libvirtd.conf の設定」 の手順を参照してください。

1. 手順15.1「libvirtd.conf の設定」 の説明に従って、libvirtd.conf ファイルを開きます。
2. Processing controls のセクションを探します。

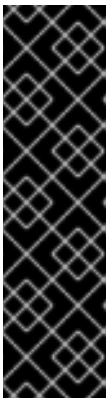
```
#################################################################
#
# Processing controls
#

# The maximum number of concurrent client connections to allow
# over all sockets combined.
#max_clients = 5000

# The maximum length of queue of connections waiting to be
# accepted by the daemon. Note, that some protocols supporting
# retransmission may obey this so that a later reattempt at
# connection succeeds.
#max_queued_clients = 1000

# The minimum limit sets the number of workers to start up
# initially. If the number of active clients exceeds this,
# then more threads are spawned, upto max_workers limit.
# Typically you'd want max_workers to equal maximum number
# of clients allowed
#min_workers = 5
#max_workers = 20


# The number of priority workers. If all workers from above
# pool will stuck, some calls marked as high priority
# (notably domainDestroy) can be executed in this pool.
#prio_workers = 5

# Total global limit on concurrent RPC calls. Should be
# at least as large as max_workers. Beyond this, RPC requests
# will be read into memory and queued. This directly impact
# memory usage, currently each request requires 256 KB of
# memory. So by default upto 5 MB of memory is used
#
# XXX this isn't actually enforced yet, only the per-client
# limit is used so far
```

```
#max_requests = 20

# Limit on concurrent requests from a single client
# connection. To avoid one client monopolizing the server
# this should be a small fraction of the global max_requests
# and max_workers parameter
#max_client_requests = 5

#################################################################
```

3. ***max_clients*** パラメーターおよび ***max_workers*** パラメーターの設定を変更します。両方のパラメーターの番号が同じであることが推奨されます。***max_clients*** は、移行ごとに 2 つのクライアント (各側に 1 つ) を使用します。***max_workers*** は、実行フェーズ中に移行元で 1 つのワーカーと、移行先で 0 のワーカーを使用し、終了フェーズ中に移行先でワーカーを 1 つ使用します。

重要

max_clients パラメーターおよび ***max_workers*** パラメーターの設定は、libvirtd サービスへのゲスト仮想マシンのすべての接続の影響を受けます。つまり、同じゲスト仮想マシンを使用し、同時に移行を実行しているユーザーは、***max_clients*** パラメーターおよび ***max_workers*** パラメーターの設定で設定された制限に従います。このため、同時ライブマイグレーションを実行する前に、最大値を慎重に検討する必要があります。



重要

max_clients パラメーターは、libvirt への接続を許可するクライアントの数を制御します。一度に多数のコンテナーを起動すると、この制限に簡単に到達して超過する可能性があります。***max_clients*** パラメーターの値は、これを回避するために増やすことができますが、増やすと、インスタンスに対する DoS (DoS) 攻撃に対してシステムが脆弱になる可能性があります。この問題を軽減するために、Red Hat Enterprise Linux 7.0 では、許可されているものの、まだ認証されていない接続の制限を指定する新しい ***max_anonymous_clients*** 設定が導入されました。ワークロードに合わせて、***max_clients*** と ***max_anonymous_clients*** を組み合わせて実装できます。

4. ファイルを保存し、サービスを再起動します。

注記

起動したにもかかわらず認証されていない ssh セッションが多すぎるために、移行接続が切断する場合があります。デフォルトでは、**sshd** で許可されるセッションは 10 セッションのみで、常に "pre-authenticated state" となります。この設定は、sshd 設定ファイル (ここでは **/etc/ssh/sshd_config**) の **MaxStartups** パラメーターで制御されます。これには調整が必要な場合があります。この制限は DoS 攻撃 (および一般的なリソースの過剰使用) を防ぐために設定されているため、このパラメーターの調整は慎重に行ってください。この値を高く設定しすぎると、目的が無効になります。このパラメーターを変更するには、ファイル/**etc/ssh/sshd_config**を変更し、**MaxStartups** 行の先頭から**#**を削除して、**10** (デフォルト値) をより大きな数値に変更します。必ず保存して、**sshd** サービスを再起動してください。詳細は、man ページの **sshd_config** を参照してください。

15.5.2. virsh migrate コマンドの追加オプション

--live のほかに、virsh migrate では以下のオプションを利用できます。

- **--direct** - 直接移行に使用されます。
- **--p2p** - ピアツーピア移行に使用されます。
- **--tunneled** - トンネルマイグレーションに使用されます。
- **--offline** - 移行先のドメインを起動せず、また、移行元ホストで停止することなく、ドメイン定義を移行します。オフラインマイグレーションは、非アクティブなドメインで使用できるため、**--persistent** オプションで使用する必要があります。
- **--persistent** - 移行先ホストの物理マシンでドメインを永続的な状態にします。
- **--undefinesource** - 移行元ホストの物理マシンでドメインの定義を解除する
- **--suspend** - ドメインを移行先ホスト物理マシンで一時停止したままにします。
- **--change-protection** - 移行の進行中に、互換性のない設定変更がドメインに行われないように強制します。ハイパーバイザーによるサポートがある場合にこのフラグが暗黙的に有効になりますが、ハイパーバイザーに変更保護の対応がない場合に明示的に使用して移行を拒否できます。
- **--unsafe** - すべての安全手順を無視して、強制的に移行を実行します。
- **--verbose** - 移行の進行状況を表示します。
- **--compressed** - ライブマイグレーション中に繰り返し転送する必要があるメモリーページの圧縮をアクティブにします。
- **--abort-on-error** - 移行中にソフトエラー (I/O エラーなど) が発生すると、移行をキャンセルします。
- **--domain *[name]*** - ドメインネーム、ID、または UUID を設定します。
- **--desturi *[URI]*** - クライアント (通常の移行) またはソース (p2p 移行) から見た宛先ホストの接続 URI。
- **--migrateuri *[URI]*** - 移行 URI です。通常は省略できます。
- **--graphicsuri *[URI]*** - シームレスなグラフィックス移行に使用されるグラフィックス URI。
- **--listen-address *[address]*** - 移行先のハイパーバイザーが着信時にバインドす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、ゲスト仮想マシンが強制的にサスペンドします。ライブマイグレーションでのみ使用できます。タイムアウトが開始されると、一時停止されたゲスト仮想マシンで移行が続行されます。
- **--dname *[newname]*** - 移行中にドメインの名前を変更するために使用されますが、通常は省略できます。
- **--xml *[filename]*** - 指定されたファイル名を使用して、宛先で使用する別の XML ファイルを提供できます。このファイル名は、基となるストレージへのアクセスで、ソースと宛先の名前の相違を考慮するなど、ホスト固有のドメイン XML の部分にさらに多くの変更を加えます。このオプションは通常、省略されます。

- **--migrate-disks *[disk_identifiers]*** - このオプションを使用して、移行中にコピーするディスクを選択できます。これにより、特定のディスクをコピーすることが望ましくない場合 (コピー先にディスクがすでに存在する場合や、不要になった場合など) に、より効率的なライブマイグレーションが可能になります。*[disk_identifiers]* は、ドメイン XML ファイルの **<target dev= />** 行にある引数で識別される、移行するディスクのコンマ区切りのリストに置き換える必要があります。

また、次のコマンドも役に立ちます。

- **virsh migrate-setmaxdowntime *[domain] [downtime]*** - 別のホストにライブマイグレーションされているドメインで許容されるダウンタイムの上限を設定します。指定されたダウンタイムはミリ秒単位で表されます。指定するドメインは、移行す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サイズ (バイト) を設定または取得します。**--size** を使用しないと、圧縮キャッシュの現在のサイズが表示されます。**--size** を使用し、バイト単位で指定すると、ハイパーバイザーは、指定されたサイズに合わせて圧縮を変更するように求められます。これに続いて、現在のサイズが表示されます。**--size** 引数は、ドメインがライブマイグレーション中に、マイグレーションの進捗状況と、**domjobinfo**から取得される圧縮キャッシュミスの数の増加への反応として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されたドメインの移行帯域幅を Mib /秒で設定します。

- **virsh migrate-getspeed *[domain]*** - 指定されたドメインで使用可能な最大移行帯域幅を Mib / 秒で取得します。

詳細は、 移行の制限 または virsh の man ページを参照してください。

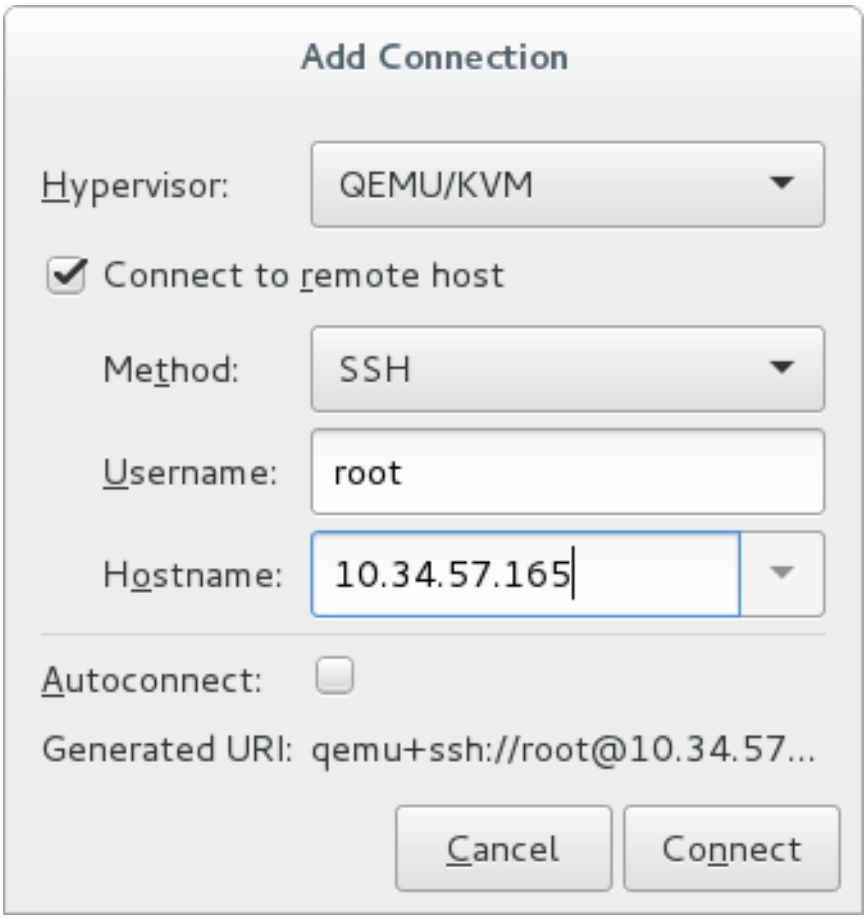
15.6. VIRT-MANAGER を使用した移行

本セクションでは、**virt-manager** を使用した KVM ゲスト仮想マシンの、ホスト物理マシンから別のホスト物理マシンへの移行を説明します。

1. **ターゲットホストの物理マシンへの接続**
   virt-manager インターフェイス で、**File** メニューを選択して、ターゲットホストの物理マシンに接続し、**Add Connection** をクリックします。

2. **接続の追加**
   Add Connection ウィンドウが表示されます。

図15.1 ターゲットホストの物理マシンへの接続の追加



以下の詳細を入力します。

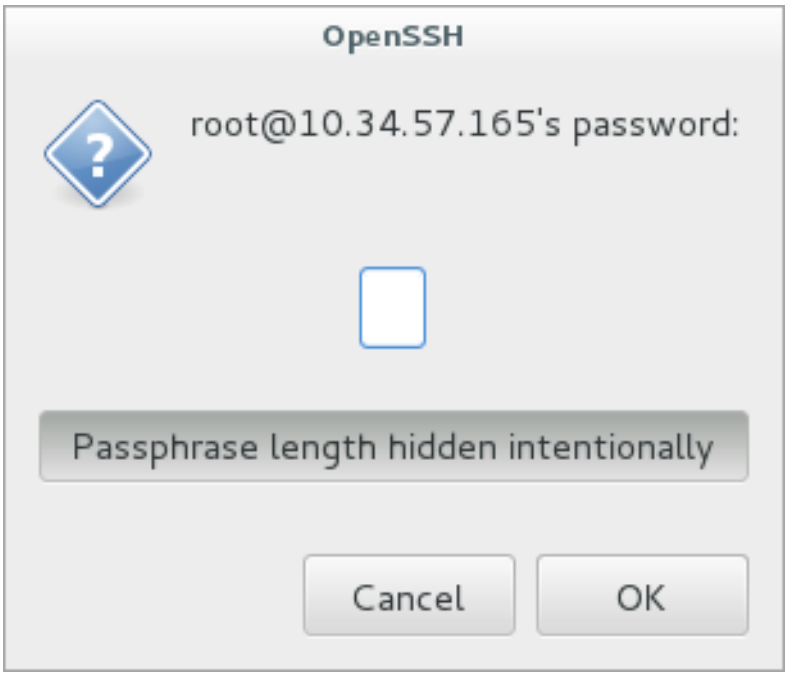
- **Hypervisor**: QEMU/KVM を選択します。
- **Method**: 接続方法を選択します。
- **Username**: リモートホストの物理マシンのユーザー名を入力します。
- **Hostname**: リモートホストの物理マシンのホスト名を入力します。

注記

接続オプションの詳細は、「リモート接続の追加」 を参照してください。

Connect 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、次の手順で指定するユーザーのパスワードを入力する必要があります。

図15.2 パスワードを入力

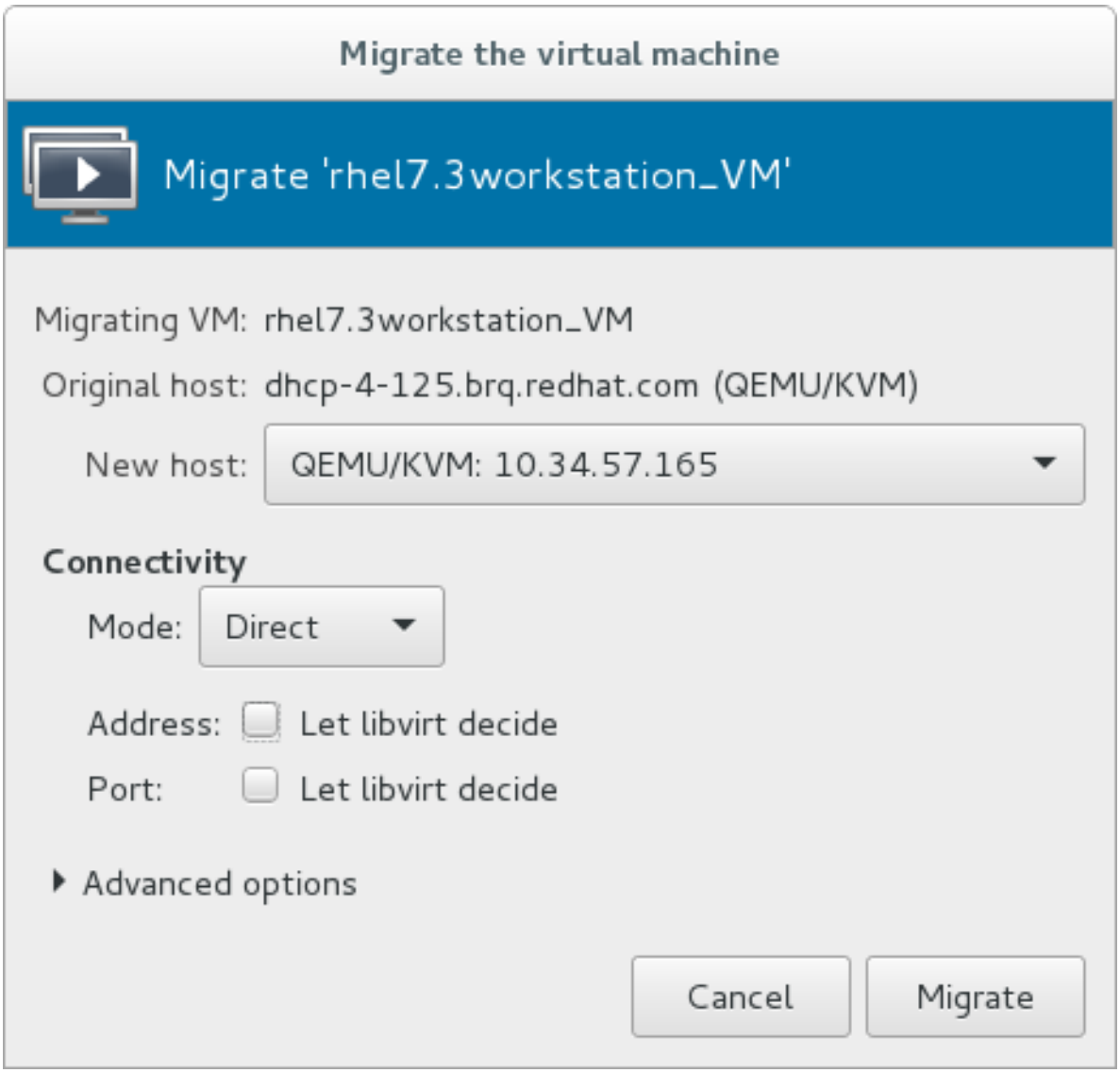


3. **共有ストレージの設定**
   移行元ホストと移行先ホストの両方が、ストレージ (NFS の使用 など) を共有していることを確認します。

4. **ゲスト仮想マシンの移行**
   移行するゲストを右クリックし、**Migrate** をクリックします。

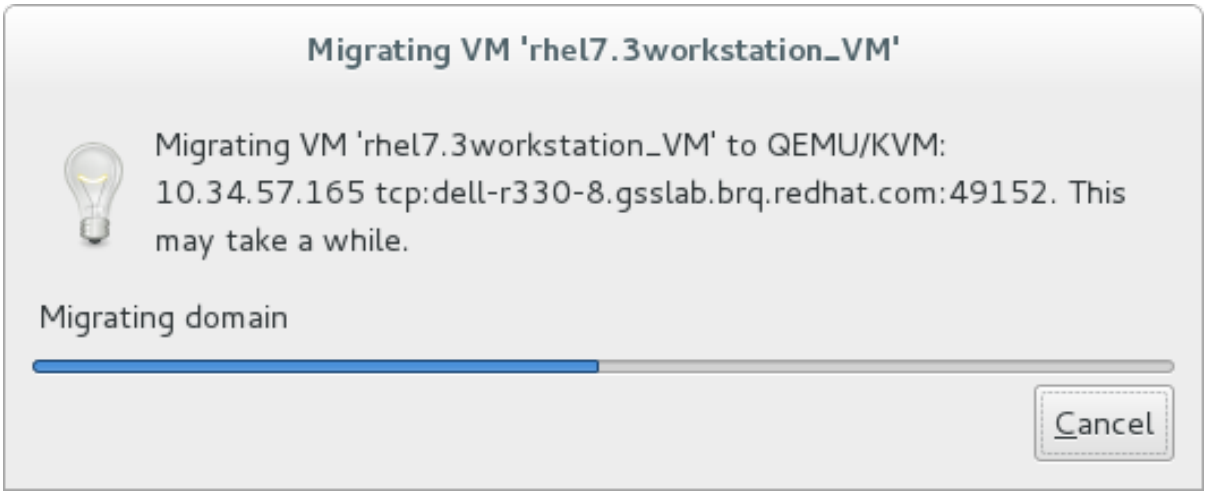
   New Host フィールドで、ドロップダウンリストを使用して、ゲスト仮想マシンの移行先となるホスト物理マシンを選択し、**Migrate** をクリックします。

図15.3 移行先ホストの物理マシンの選択と、移行プロセスの開始



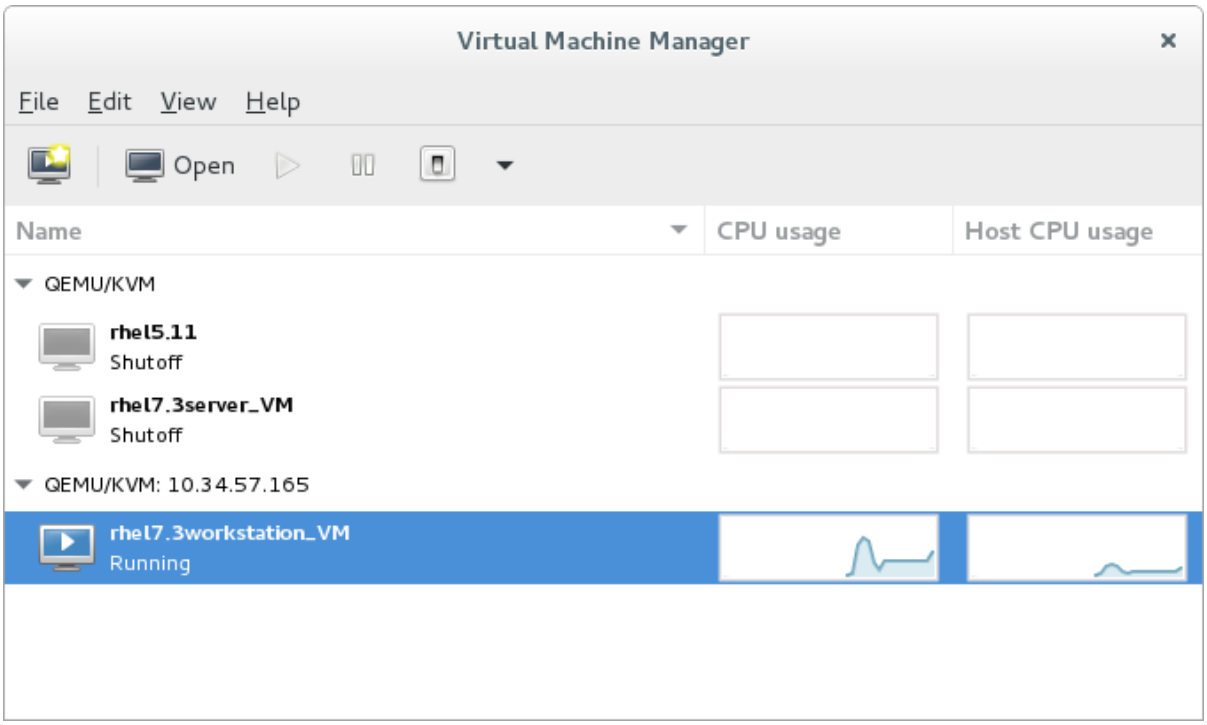
進捗ウィンドウが表示されます。

図15.4 進捗ウィンドウ



移行が問題なく終了すると、**virt-manager** は、移行先ホストで実行している、新しく移行したゲスト仮想マシンを表示します。

図15.5 移行先ホストの物理マシンで実行している移行ゲスト仮想マシン



第16章 ゲスト仮想マシンのデバイス設定

Red Hat Enterprise Linux 7 は、ゲスト仮想マシン用に 3 つのクラスのデバイスに対応します。

- **エミュレートされたデバイス** は、実際のハードウェアを模倣する純粋な仮想デバイスであり、未変更のゲストオペレーティングシステムが標準のインボックスドライバーを使用して動作できるようにします。

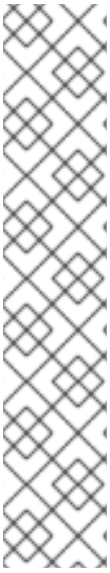
- **Virtio デバイス** (*paravirtualized* とも呼ばれます) は、純粋に仮想マシンで最適に機能するように設計された仮想デバイスです。Virtio デバイスはエミュレートされたデバイスに似ていますが、Linux 以外の仮想マシンには、デフォルトで必要なドライバーが含まれません。Virtual Machine Manager (**virt-manager**) や Red Hat Virtualization Hypervisor などの仮想化管理ソフトウェアは、サポートされている Linux 以外のゲストオペレーティングシステム用にこれらのドライバーを自動的にインストールします。Red Hat Enterprise Linux 7 は、最大 216 の virtio デバイスに対応します。詳細は、*5章KVM 準仮想化 (virtio) ドライバー*. を参照してください。

- **割り当てられたデバイス** は、仮想マシンに公開される物理デバイスです。この方法は *passthrough* としても知られています。デバイスの割り当てにより、仮想マシンはさまざまなタスクで PCI デバイスに排他的にアクセスできるようになり、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見え、動作します。Red Hat Enterprise Linux 7 は、仮想マシンごとに最大 32 の割り当てられたデバイスをサポートします。

  デバイスの割り当ては、一部のグラフィックデバイス を含む PCIe デバイスでサポートされています。パラレル PCI デバイスは、割り当てられたデバイスとしてサポートされる場合がありますが、セキュリティーとシステム設定の競合のために厳しい制限があります。

Red Hat Enterprise Linux 7 は、単一機能スロットとして仮想マシンに公開されるデバイスの PCI ホットプラグに対応します。これを可能にするために、単一機能のホストデバイスや、多機能のホストデバイスの個別の機能を設定できます。デバイスを多機能 PCI スロットとして仮想マシンに公開する設定は、ホットプラグ以外のアプリケーションにのみ推奨されます。

特定のデバイスおよび関連の制限の詳細は、「Devices」 を参照してください。



注記

デバイスが割り当てられたゲストをホストから完全に分離するには、プラットフォームが割り込みの再マッピングをサポートしている必要があります。このようなサポートがないと、ホストは悪意のあるゲストからの割り込み注入攻撃に対して脆弱と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、***vfio_iommu_type1*** モジュールへの **allow_unsafe_interrupts** を使用した PCI デバイスの割り当てを引き続き許可するように、管理者がオプトインする場合があります。これは、以下を含む **/etc/modprobe.d** に .conf ファイル (**local.conf** など) を追加することで、永続的に実行できます。

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

または sysfs エントリーを使用して動的に同じことを行います。

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

16.1. PCI デバイス

PCI デバイスの割り当ては、Intel VT-d または AMD IOMMU のいずれかに対応するハードウェアプラットフォームでのみ利用できます。PCI デバイスの割り当てを機能させるには、この Intel VT-d または AMD IOMMU の仕様が、ホスト BIOS で有効になっている必要があります。

手順16.1 PCI デバイス割り当てのための Intel システムの準備

1. **Intel VT-d 仕様を有効にする**
   Intel VT-d 仕様では、物理デバイスを仮想マシンに直接割り当てるハードウェアサポートが提供されます。この仕様は、Red Hat Enterprise Linux で PCI デバイスの割り当てを使用するために必要です。

   Intel VT-d 仕様は、BIOS で有効にする必要があります。システムの製造元によっては、この仕様をデフォルトで無効にしている場合があります。これらの仕様を表示するのに使用される用語はメーカーにより異なります。適切な用語は、システムの製造元のドキュメント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。**
   /etc/sysconfig/grub ファイル内の GRUB_CMDLINX_LINUX 行の末尾に、引用符で囲んで ***intel_iommu=on*** パラメーターおよび ***iommu=pt*** パラメーターを追加して、カーネル内で Intel VT-d をアクティブにします。

   以下は、Intel VT-d を有効にした修正**grub**です。

   ```
   GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
   vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
   vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ] && /usr/sbin/
   rhcrashkernel-param || :) rhgb quiet intel_iommu=on iommu=pt"
   ```

3. **設定ファイルの再生成**
   以下を実行して /etc/grub2.cfg を再生成します。

   ```
   grub2-mkconfig -o /etc/grub2.cfg
   ```

   UEFI ベースのホストを使用している場合は、ターゲットファイルが**/etc/grub2-efi.cfg** であることに注意してください。

4. **使用準備完了**
   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 デバイスの割り当てが可能になります。

手順16.2 PCI デバイス割り当て用の AMD システムの準備

1. **AMD IOMMU 仕様を有効にする**
   Red Hat Enterprise Linux で PCI デバイスの割り当てを使用するには、AMD IOMMU の仕様が必要です。この仕様は、BIOS で有効にする必要があります。システムの製造元によっては、この仕様をデフォルトで無効にしている場合があります。

2. **IOMMU カーネルサポートの有効化**
   システムの起動時に AMD IOMMU 仕様が有効になるように、**/etc/sysconfig/grub** の GRUB_CMDLINX_LINUX 行の末尾に引用符で囲って ***iommu=pt*** を追加します。

3. **設定ファイルの再生成**
   以下を実行して /etc/grub2.cfg を再生成します。

   ```
   grub2-mkconfig -o /etc/grub2.cfg
   ```

   UEFI ベースのホストを使用している場合は、ターゲットファイルが**/etc/grub2-efi.cfg** であることに注意してください。

4. **使用準備完了**
   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 デバイスの割り当てが可能になります。

注記

IOMMU の詳細は、付録E *IOMMU グループの使用* を参照してください。

16.1.1. virsh を使用した PCI デバイスの割り当て

この手順では、KVM ハイパーバイザーの仮想マシンに PCI デバイスを割り当てる方法を説明します。

この例では、PCI 識別子コード、**pci_0000_01_00_0**、および完全に仮想化されたゲストマシン *guest1-rhel7-64* を持つ PCIe ネットワークコントローラーを使用します。

手順16.3 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. **デバイスの識別**
   まず、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。使用可能な PCI デバイスのリストを表示する場合は、**lspci** コマンドを実行します。**grep** を使用して、**lspci** の出力を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、短い識別子 **00:19.0** で表示されます。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 ID を確認する必要があります。

   これを行うには、**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タイプ (**pci**) のデバイスをすべてリスト表示します。次に、使用するデバイスの短い識別子にマップする文字列の出力を調べます。

   この例では、短い識別子 **00:19.0** を使用して、イーサネットコントローラーにマップする文字列を示しています。**:** 文字および **.** 文字は、完全識別子の下線に置き換えられ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する PCI デバイス番号を記録します。これは別の手順で必要になります。

2. **デバイス情報の確認**
ドメイン、バス、および機能の情報は、**virsh nodedev-dumpxml** コマンドの出力から取得できます。

図16.1 ダンプの内容

```
# 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 からの視認性とデバイスの分離に基づいて決定されます。各 IOMMU グループには、1 つ以上のデバイスを含めることができます。複数のデバイスが存在する場合は、グループ内のすべてのデバイスがゲストに割り当てられるため、IOMMU グループ内のすべてのエンドポイントが要求されます。これは、追加のエンドポイントをゲストに割り当てるか、**virsh nodedev-detach** を使用してホストドライバーから外すことで実行できます。1 つのグループに含まれるデバイスが複数のゲストに分割されたり、ホストとゲストが分割されたりすることはありません。PCIe root ポート、スイッチポート、ブリッジなどの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しないでください。また、エンドポイントの割り当てに影響を及ぼしません。

IOMMU グループ内のデバイスは、**virsh nodedev-dumpxml** 出力の IOMMU グループセクションを使用して決定できます。グループの各メンバーは、別のアドレスフィールドで提供されます。この情報は、sysfs で以下を使用して取得することもできます。

```
$ ls /sys/bus/pci/devices/0000:01:00.0/iommu_group/devices/
```

この出力の例を以下に示します。

```
0000:01:00.0  0000:01:00.1
```

ゲストに 0000.01.00.0 のみを割り当てるには、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分離する必要があります。

```
$ virsh nodedev-detach pci_0000_01_00_1
```

3. **必要な設定の詳細を決定する**
   設定ファイルに必要な値は、**virsh nodedev-dumpxml pci_0000_00_19_0** コマンドの出力を参照してください。

   この例のデバイスは、bus = 0、slot = 25、および function = 0 の値を持ちます。10 進数の設定では、この 3 つの値が使用されます。

   ```
   bus='0'
   slot='25'
   function='0'
   ```

4. **設定の詳細の追加**
   virsh edit を実行し、仮想マシン名を指定し、**<devices>** 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。以下に例を示します。

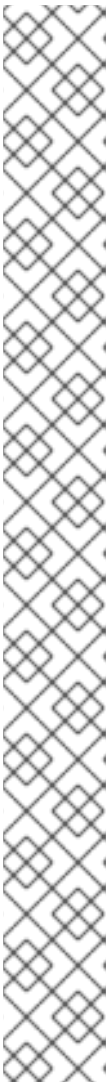
   ```
   # virsh edit guest1-rhel7-64
   ```

図16.2 PCI デバイスの追加

```
<devices>
 [...]
 <hostdev mode='subsystem' type='pci' managed='yes'>
   <source>
     <address domain='0' bus='0' slot='25' function='0'/>
   </source>
 </hostdev>
 [...]
</devices>
```

または、**virsh attach-device** を実行して、仮想マシンの名前とゲストの XML ファイルを指定します。

```
virsh attach-device guest1-rhel7-64 file.xml
```



注記

PCI デバイスには、デバイスファームウェアまたはデバイス用の起動前ドライバー (PXE など) を提供するための *オプションの読み取り専用メモリー (ROM) モジュール* (*オプション ROM* または *拡張 ROM* としても知られている) が含まれます。通常、このオプション ROM は、PCI デバイスの割り当てを使用して物理 PCI デバイスを仮想マシンに接続する際に、仮想環境でも機能します。

ただし、オプション ROM が必要ない場合があります。これにより、仮想マシンの起動に時間がかかったり、デバイスが提供する起動前ドライバーが仮想化と互換性がなくなり、ゲスト OS の起動に失敗する可能性があります。この場合、Red Hat は、仮想マシンからオプション ROM をマスクすることを推奨します。これを行うには、以下を行います。

1. ホストで、割り当てるデバイスに拡張 ROM ベースアドレスレジスタ (BAR) があることを確認します。これを行うには、デバイスに対して **lspci -v** コマンドを使用し、以下を含む行の出力を確認します。

   ```
   Expansion ROM at
   ```

2. **<rom bar='off'/>** 要素を、ゲストの XML 設定の **<hostdev>** 要素の子として追加します。

   ```
   <hostdev mode='subsystem' type='pci' managed='yes'>
     <source>
       <address domain='0' bus='0' slot='25' function='0'/>
     </source>
     <rom bar='off'/>
   </hostdev>
   ```

5. **仮想マシンの起動**

   ```
   # virsh start guest1-rhel7-64
   ```

これにより、PCI デバイスが仮想マシンに正常に割り当てられ、ゲストオペレーティングシステムにアクセスできるようになります。

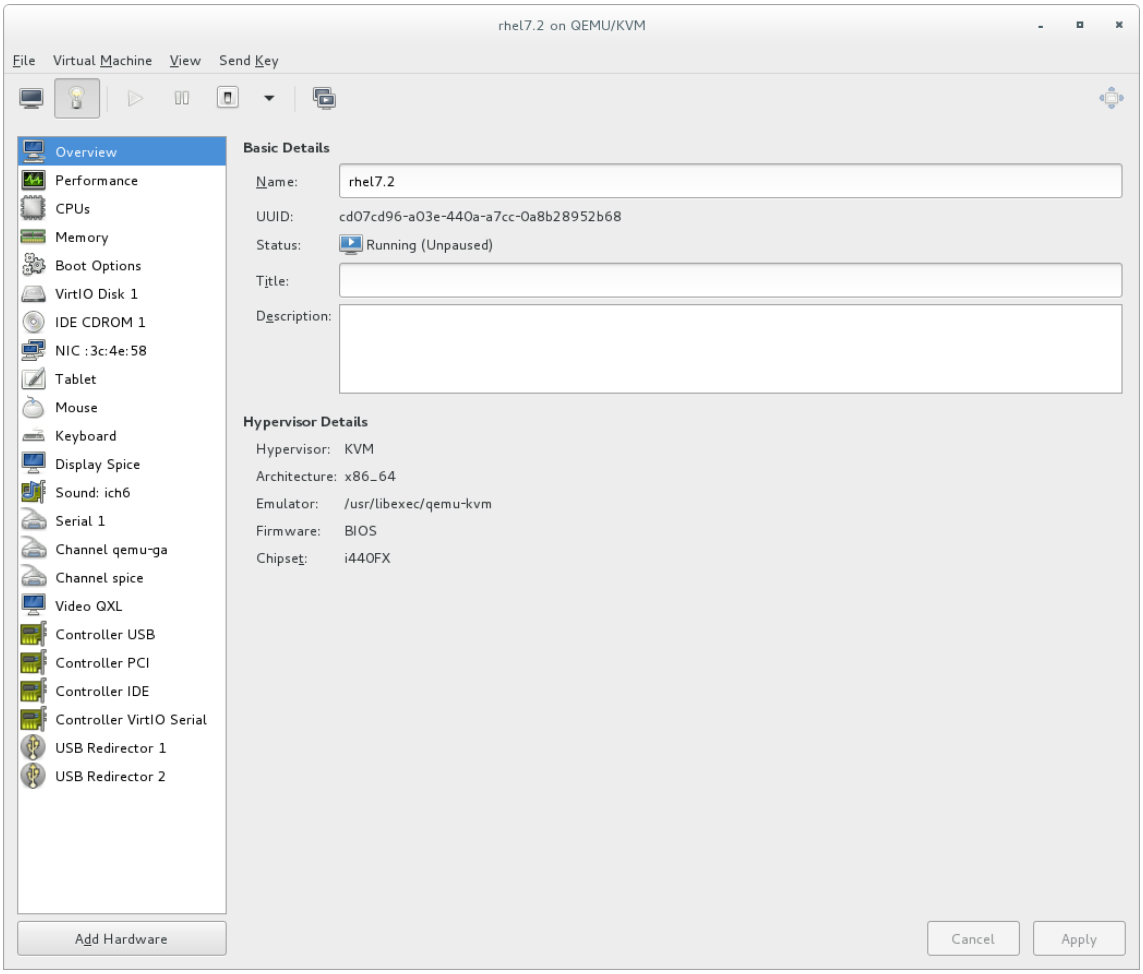
16.1.2. virt-manager を使用した PCI デバイスの割り当て

PCI デバイスは、グラフィカル**virt-manager**ツールを使用してゲスト仮想マシンに追加できます。次の手順では、ギガビット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16.4 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く**
   ゲスト仮想マシンを開き、**Add Hardware** をクリックして、仮想マシンに新しいデバイスを追加します。

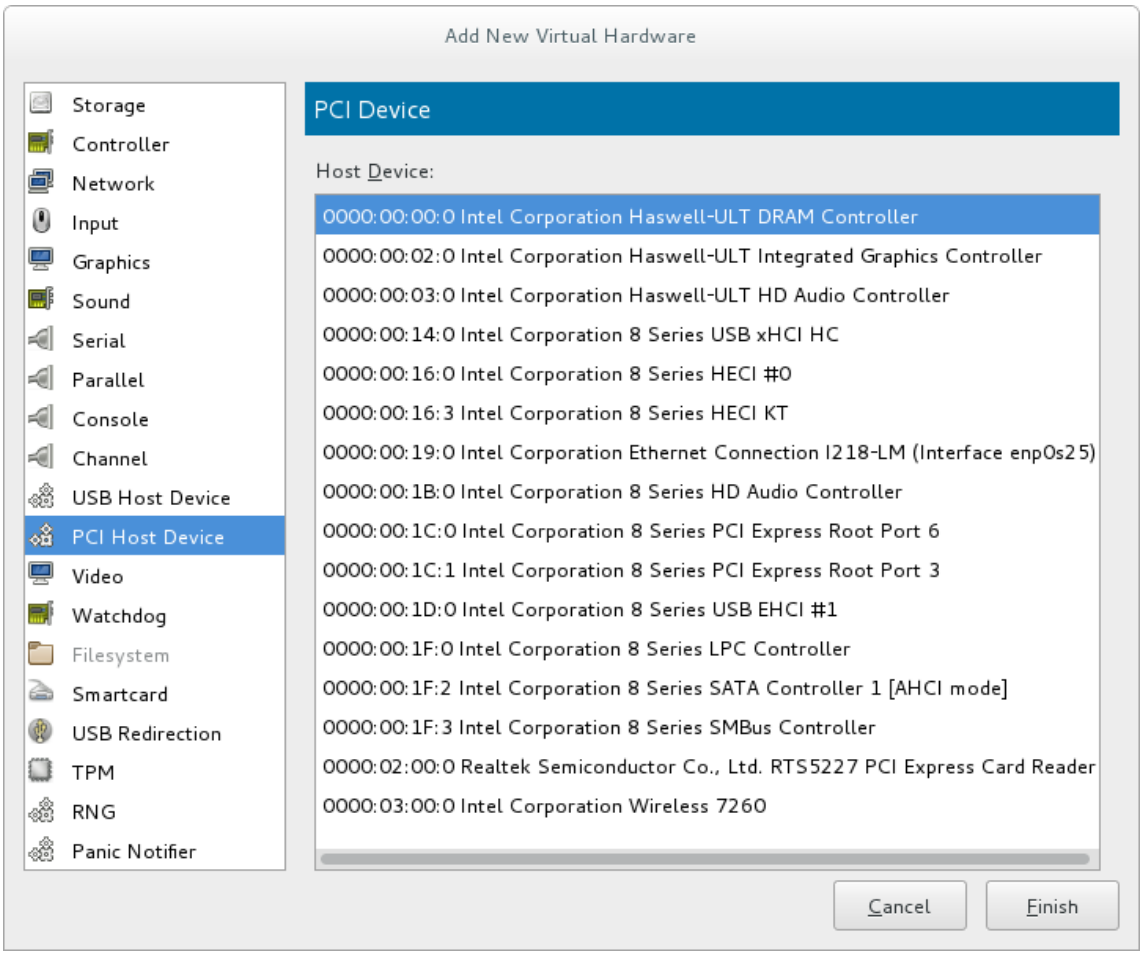
   図16.3 仮想マシンのハードウェア情報ウィンドウ



2. **PCI デバイスの選択**
   左側の **Hardware** リストから **PCI Host Device** を選択します。

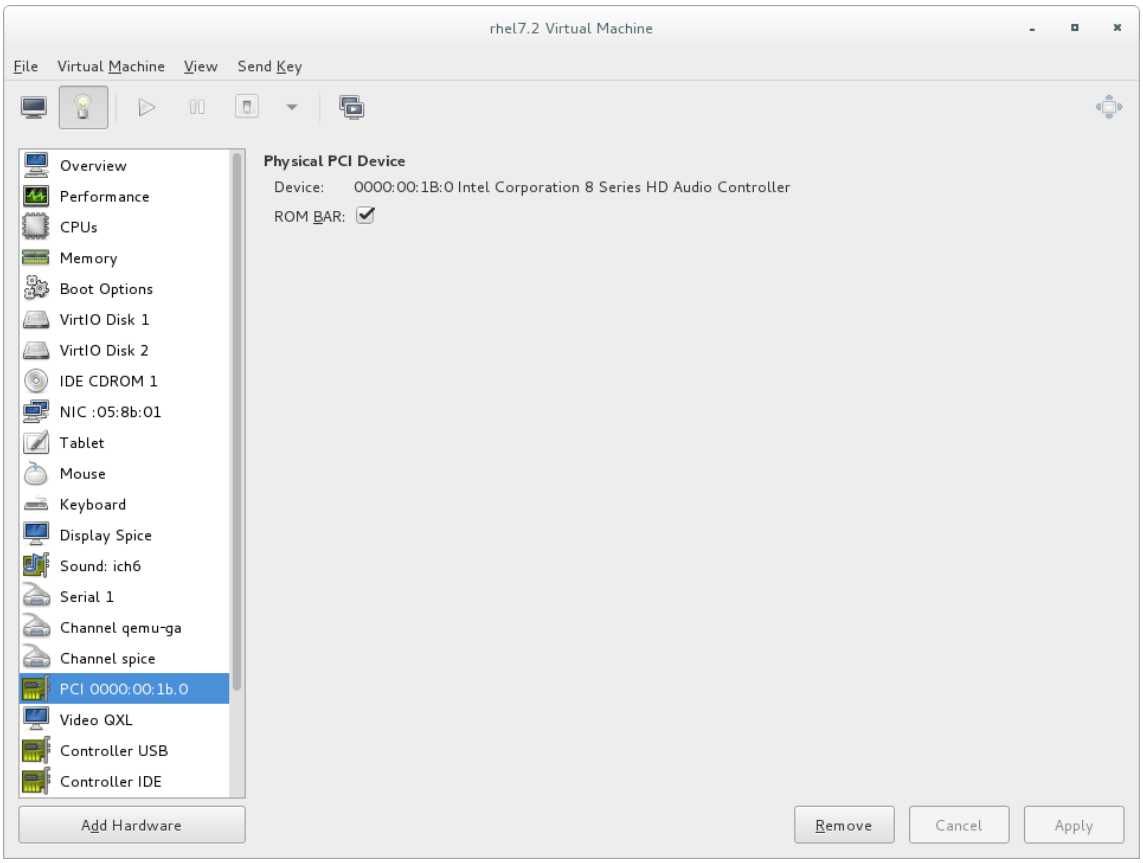
   未使用の PCI デバイスを選択します。現在、別のゲストが使用している PCI デバイスを選択すると、エラーが発生することに注意してください。この例では、予備のオーディオコントローラーが使用されています。**Finish** を選択して設定を完了します。

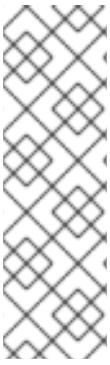
図16.4 Add new virtual hardware ウィザード



3. **新しいデバイスの追加**
   セットアップが完了し、ゲスト仮想マシンが PCI デバイスに直接アクセスできるようになりました。

図16.5 仮想マシンのハードウェア情報ウィンドウ





注記

デバイスの割り当てに失敗すると、同じ IOMMU グループに、ホストに依然として接続されているその他のエンドポイントが存在す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 コマンドを使用すると、IOMMU グループの境界を分析したり、必要に応じてデバイスを隔離したりできます。

IOMMU グループの詳細と、virsh を使用してエンドポイントデバイスの割り当てを解除する方法は、「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。

16.1.3. virt-install を使用した PCI デバイスの割り当て

virt-install でゲストをインストールする際に、PCI デバイスを割り当てることができます。これには、***--host-device*** パラメーターを使用します。

手順16.5 virt-install を使用した、仮想マシンへの PCI デバイスの割り当て

1. **デバイスの識別**
   ゲスト仮想マシンに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すべてのデバイスのリストを表示し、各 PCI デバイスを文字列で識別します。出力を PCI デバイスに限定するには、次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 デバイス番号を記録します。この番号は他の手順で確認する必要があります。

ドメイン、バス、および機能の情報は、**virsh nodedev-dumpxml** コマンドの出力から取得できます。

```
# virsh nodedev-dumpxml pci_0000_01_00_0
```

図16.6 PCI デバイスファイルの内容

```
<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、そのすべてがゲストに割り当てられていない場合は、ゲストを起動する前に次のコマンドを実行して、ホストからその他のエンドポイントの割り当てを手動で解除する必要があります。

```
$ 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. **デバイスの追加**
   virsh nodedev コマンドから出力された PCI 識別子を、 ***--host-device*** パラメーターの値として使用します。

```
virt-install \
--name=guest1-rhel7-64 \
--disk path=/var/lib/libvirt/images/guest1-rhel7-64.img,size=8 \
--vcpus=2 --ram=2048 \
--location=http://example1.com/installation_tree/RHEL7.0-Server-x86_64/os \
--nonetworks \
--os-type=linux \
--os-variant=rhel7
--host-device=pci_0000_01_00_0
```

3. **インストールを完了する**
   ゲストのインストールを完了します。PCI デバイスはゲストに接続する必要があります。

16.1.4. 割り当てられた PCI デバイスの取り外し

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、ホストがそのデバイスを使用できなくなり

ます。PCI デバイスが **managed** モード (ドメインの XML ファイル の **managed='yes'** パラメーターを使用して設定) の場合は、ゲストマシンに接続し、ゲストマシンの接続を解除し、必要に応じてホストマシンに再度接続します。PCI デバイスが **managed** モードではない場合は、ゲストマシンから PCI デバイスの接続を解除し、**virsh** または **virt-manager** を使用して再度接続できます。

手順16.6 virsh を使用したゲストからの PCI デバイスの切り離し

1. **デバイスの取り外し**
   次のコマンドを使用して、ゲストの XML ファイルから PCI デバイスを削除し、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、この手順を省略します。デバイスは自動的にホストに戻ります。

   デバイスが ***managed*** モードを使用していない場合は、次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** デバイスーをホストコンピューターに再接続するには、次のコマンドを実行します。

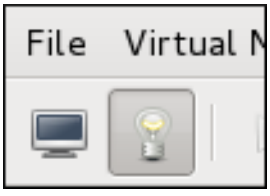
   ```
   # virsh nodedev-reattach pci_0000_01_00_0
   ```

   これで、デバイスがホストで使用できるようになります。

手順16.7 virt-manager を使用したゲストからの PCI デバイスの切り離し

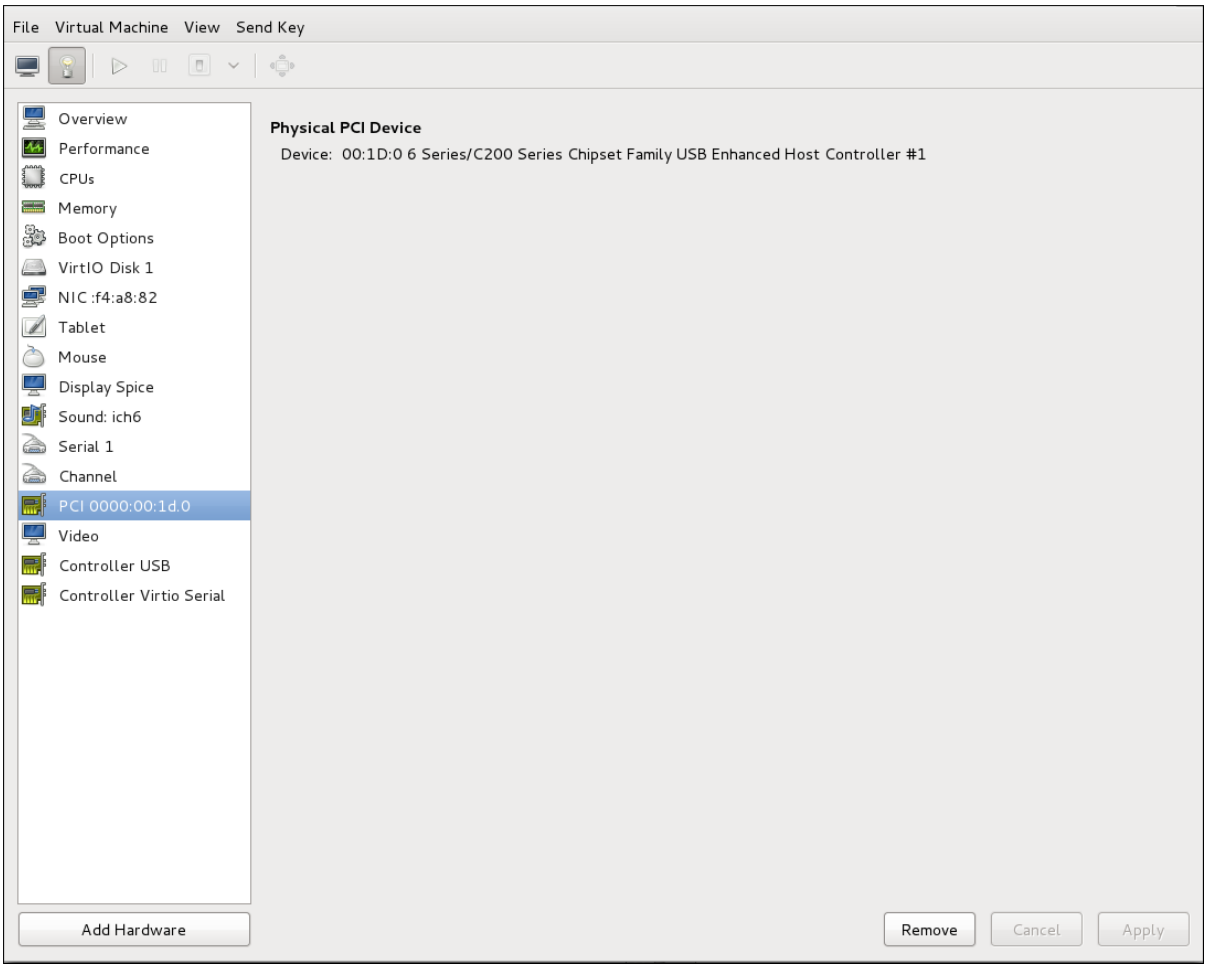
1. **仮想ハードウェアの詳細画面を開きます。**
   virt-manager で、デバイスを含む仮想マシンをダブルクリックします。**Show virtual hardware details** ボタンを選択すると、仮想ハードウェアのリストが表示されます。

   図16.7 仮想ハードウェアの詳細ボタン

2. **デバイスを選択して削除する**
   左側のパネルにある仮想デバイスのリストから、取り外す PCI デバイスを選択します。

図16.8 取り外す PCI デバイスの選択



Remove ボタンをクリックして確定します。これで、デバイスがホストで使用できるようになります。

16.1.5. PCI ブリッジ

PCI (Peripheral Component Interconnect) ブリッジは、ネットワークカード、モデム、サウンド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 ブリッジに接続できます。以前は、31 個の PCI デバイスしかゲスト仮想マシンに追加できませんでした。現在、31 番目の PCI デバイスを追加すると、PCI ブリッジが 31 番目のスロットに自動的に配置され、追加した PCI デバイスを PCI ブリッジに移動します。各 PCI ブリッジには、31 の追加デバイス用に 31 個のスロットがあり、すべてがブリッジになることができます。この方法では、ゲスト仮想マシンで 900 を超えるデバイスを使用できます。

PCI ブリッジの XML 設定の例は、Domain XML example for PCI Bridge を参照してください。この設定は自動的に設定されるため、手動で調整することは推奨しません。

16.1.6. PCI デバイスの割り当ての制限

PCI デバイスの割り当て (仮想マシンへの PCI デバイスの割り当て) では、ホストシステムが AMD IOMMU または Intel VT-d をサポートして、PCIe デバイスのデバイス割り当てを有効にする必要があります。

Red Hat Enterprise Linux 7 では、ゲストデバイスドライバーによる PCI 設定領域のアクセスが制限されています。この制限により、拡張 PCI 設定領域にあるデバイスの機能や、機能に依存するドライバーが設定に失敗する場合があります。

Red Hat Enterprise Linux 7 仮想マシンには、割り当てられているデバイスの合計が 32 個までに制限されています。これは、仮想マシンに存在する PCI ブリッジの数や、これらの機能を組み合わせて多機能スロットを作成する方法に関係なく、合計 32 の PCI 機能に変換されます。

デバイスが割り当てられたゲストをホストから完全に分離するには、プラットフォームが割り込みの再マッピングをサポートしている必要があります。このようなサポートがないと、ホストは悪意のあるゲストからの割り込み注入攻撃に対して脆弱と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、管理者が ***vfio_iommu_type1*** モジュールへの **allow_unsafe_interrupts** オプションを使用して、PCI デバイスの割り当てを引き続き許可するようにオプトインすることができます。これは、以下を含む **/etc/modprobe.d** に .conf ファイル (**local.conf** など) を追加することで、永続的に実行できます。

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

または sysfs エントリーを使用して動的に同じことを行います。

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

16.2. SR-IOV デバイスを使用した PCI デバイスの割り当て

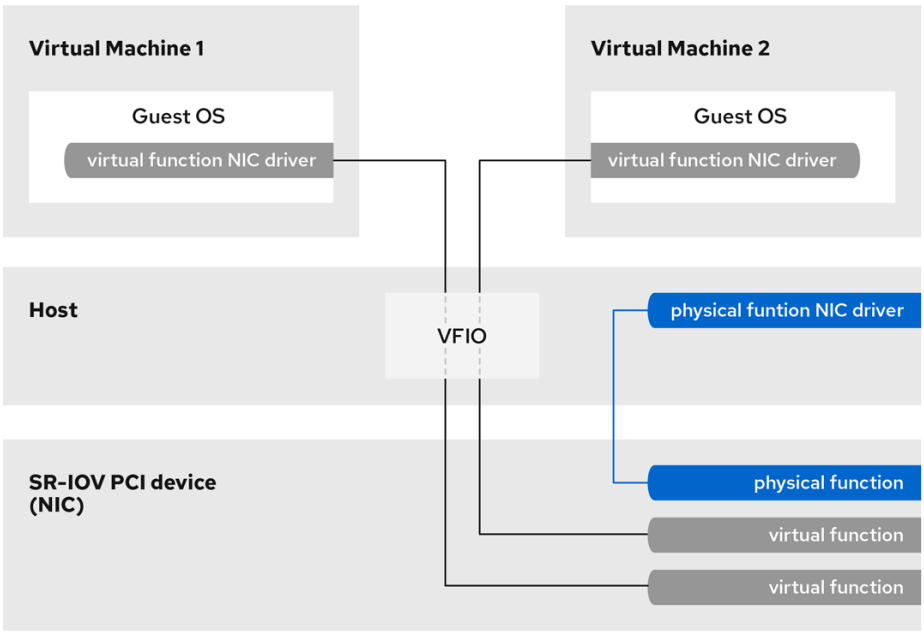
PCI ネットワークデバイス (**<source>** 要素によりドメイン XML で指定) は、直接デバイス割り当て (*passthrough* と呼ばれることもあります) を使用して、ゲストに直接接続できます。 標準的なシングルポートの PCI イーサネットカードドライバー設計の制限により、この方法で割り当てることができるのは *Single Root I/O Virtualization* (SR-IOV) *virtual function* (VF) デバイスのみとなります。標準的なシングルポートの PCI または PCIe イーサネットカードをゲストに割り当てる場合は、従来の **<hostdev>** デバイス定義を使用します。

図16.9 PCI デバイスの割り当ての XML 例

```
 <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>
```

PCI-SIG (PCI Special Interest Group) が開発した SR-IOV (Single Root I/O Virtualization) 仕様は、単一デバイスを複数の仮想マシンに共有できる PCI デバイス割り当てタイプの標準です。SR-IOV は、仮想マシンのデバイスパフォーマンスを改善します。

図16.10 SR-IOV の仕組み



SR-IOV を使用すると、単一の root 機能 (たとえば、単一のイーサネットポート) を複数の個別の物理デバイスとして表示できます。SR-IOV 機能を持つ物理デバイスは、PCI 設定領域に複数の機能として表示されるように設定できます。各デバイスには、ベースアドレスレジスタ (BAR) を含む独自の設定領域があります。

SR-IOV は、次の 2 つの PCI 機能を使用します。

- Physical Function (PF) は、SR-IOV 機能を含む完全な PCIe デバイスです。物理機能は、検出、管理、および通常の PCI デバイスとして設定されます。物理機能は、仮想機能を割り当てることにより、SR-IOV 機能を設定および管理します。

- 仮想機能 (VF) は、I/O のみを処理する単純な PCIe 機能です。各仮想機能は、物理機能から派生しています。デバイスに含まれる仮想機能の数は、デバイスのハードウェアにより制限されます。単一のイーサネットポートである物理デバイスは、仮想マシンで共有できる多くの仮想機能にマップできます。

ハイパーバイザーは、1 つ以上の仮想機能を仮想マシンに割り当てることができます。その後、仮想機能の設定領域が、ゲストに提示される設定領域に割り当てられます。

仮想機能にはリアルハードウェアリソースが必要であるため、各仮想機能は一度に 1 つのゲストにのみ割り当てることができます。仮想マシンには、複数の仮想機能を設定できます。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表示されるのと同じ方法で、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で実装されます。コア実装は PCI サブシステムに含まれていますが、物理機能 (PF) デバイスと仮想機能 (VF) デバイスの両方のドライバーサポートも必要です。SR-IOV 対応デバイスは、PF から VF を割り当てることができます。VF は、キューやレジスタセットなどのリソースにより、物理 PCI デバイスにバッキングされている PCI デバイスとして表示されます。

16.2.1. SR-IOV の利点

SR-IOV デバイスは、単一の物理ポートを複数の仮想マシンと共有できます。

SR-IOV VF が仮想マシンに割り当てられている場合は、(仮想マシンに透過的に) VF を残しているすべてのネットワークトラフィックを特定の VLAN に配置するように設定できます。仮想マシンは、トラ

フィックが VLAN にタグ付けされていることを検出できないため、このタグを変更したり、削除したりできません。

仮想機能は、準仮想化ドライバーおよびエミュレートされたアクセスよりもネイティブに近いパフォーマンスを発揮し、パフォーマンスを向上させます。仮想機能は、データがハードウェアによって管理および制御されるのと同じ物理サーバー上の仮想マシン間のデータ保護を提供します。

この機能により、データセンター内のホストで仮想マシンの密度を上げることができます。

SR-IOV は、複数のゲストがあるデバイスの帯域幅をより有効に利用できます。

16.2.2. SR-IOV の使用

本セクションでは、PCI パススルーを使用して、SR-IOV 対応マルチポー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方法を説明します。

SR-IOV 仮想機能 (VF) は、**virsh edit** コマンドまたは **virsh attach-device** コマンドで ***<hostdev>*** にデバイスエントリーを追加することで、仮想マシンに割り当てることができます。ただし、これは通常のネットワークデバイスとは異なり、SR-IOV VF ネットワークデバイスには永続的な一意の MAC アドレスがなく、ホストを再起動するたびに新しい MAC アドレスが割り当てられるため、問題になることがあります。このため、システムの再起動後にゲストに同じ VF が割り当てられた場合でも、ホストの再起動時に、ゲストのネットワークアダプターが新しい MAC アドレスを持つと判断します。その結果、ゲストは毎回新しいハードウェアが接続されていると考え、通常はゲストのネットワーク設定を再設定する必要があります。

libvirt には、***<interface type='hostdev'>*** インターフェイスデバイスが含まれます。このインターフェイスデバイスを使用すると、**libvirt** は、最初に、示されたネットワーク固有のハードウェアおよびスイッチの初期化 (MAC アドレス、VLAN タグ、802.1Qbh 仮想ポートパラメーターの設定など) を実行し、次にゲストに PCI デバイスを割り当てます。

<interface type='hostdev'> インターフェイスを使用するには、以下が必要です。

- SR-IOV 対応のネットワークカード、
- Intel VT-d 拡張機能または AMD IOMMU 拡張機能のいずれかに対応するホストハードウェア
- 割り当てる VF の PCI アドレス。

重要

仮想マシンへの SR-IOV デバイスの割り当てには、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。

Intel または AMD システムに SR-IOV ネットワークデバイスを接続する場合は、以下の手順を行います。

手順16.8 Intel または AMD システムでの SR-IOV ネットワークデバイスの接続

1. **BIOS およびカーネルで、Intel VT-d または AMD IOMMU の仕様を有効にします。**
   Intel システムでは、BIOS で Intel VT-d が有効になっていない場合は有効にします。BIOS およびカーネルで Intel VT-d を有効にする場合は、手順16.1「PCI デバイス割り当てのための Intel システムの準備」 を参照してください。

   Intel VT-d がすでに有効で機能している場合は、この手順をスキップしてください。

AMD システムでは、BIOS で AMD IOMMU 仕様が有効になっていない場合は有効にします。BIOS で IOMMU を有効にする方法は、手順16.2「PCI デバイス割り当て用の AMD システムの準備」 を参照してください。

2. **サポートの確認**
SR-IOV 機能のある PCI デバイスが検出されたかどうかを確認します。この例では、SR-IOV に対応する Intel 82576 ネットワークインターフェイスカードをリスト表示します。**lspci** コマンドを使用して、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)
```

この出力は、その他のデバイスをすべて削除するように変更されていることに注意してください。

3. **仮想機能のアクティブ化**
以下のコマンドを実行します。

```
# echo ${num_vfs} > /sys/class/net/enp14s0f0/device/sriov_numvfs
```

4. **仮想機能を永続化します。**
再起動後も仮想機能を永続化するには、選択したエディターを使用して、以下のような udev ルールを作成します。ここでは、ネットワークインターフェイスカードでサポートされている制限まで、予定している数の VF (この例では ***2***) を指定します。以下の例では、*enp14s0f0* を PF ネットワークデバイス名に置き換え、**ENV{ID_NET_DRIVER}** の値を、使用中のドライバーに合わせて調整します。

```
# vim /etc/udev/rules.d/enp14s0f0.rules
```

```
ACTION=="add", SUBSYSTEM=="net", ENV{ID_NET_DRIVER}=="ixgbe",
ATTR{device/sriov_numvfs}="2"
```

これにより、システムの起動時に機能が有効になります。

5. **新しい仮想機能の検査**
lspci コマンドを使用して、Intel 82576 ネットワークデバイスに追加した仮想機能のリストを表示します。(もしくは、**grep** を使用して **Virtual Function** を検索し、仮想機能に対応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 デバイスの識別子は、**lspci** コマンドの ***-n*** パラメーターで確認できます。物理機能は、**0b:00.0** および **0b:00.1** に対応します。説明には、すべての仮想機能の**Virtual Function**が記載されています。

6. **デバイスが virsh で存在することを確認します。**
libvirt サービスは、仮想マシンにデバイスを追加する前にデバイスを認識する必要があります。**libvirt** は、**lspci** 出力と同様の表記を使用します。**lspci**出力内の句読点文字 (**:** および **.**) はすべて、アンダースコア (_) に変更されます。

virsh nodedev-list コマンドと **grep** コマンドを使用して、利用可能なホストデバイスのリストから Intel 82576 ネットワークデバイスをフィルターにかけます。***0b*** は、この例題の Intel 82576 ネットワークデバイス用のフィルターです。これはシステムにより異なるため、デバイスが追加される可能性があります。

```
# 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
```

仮想機能と物理機能の PCI アドレスがリストに含まれている必要があります。

7. **virsh でデバイスの詳細を取得する**
pci_0000_0b_00_0 は、物理機能のいずれかで、**pci_0000_0b_10_0** は、その物理機能に対応する最初の仮想機能です。**virsh nodedev-dumpxml** を実行すると、両方のデバイスの情報が表示されます。

```
# virsh nodedev-dumpxml pci_0000_03_00_0
<device>
  <name>pci_0000_03_00_0</name>
  <path>/sys/devices/pci0000:00/0000:00:01.0/0000:03:00.0</path>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>3</bus>
    <slot>0</slot>
    <function>0</function>
```

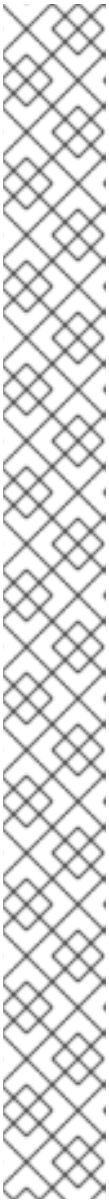
```
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='virt_functions'>
      <address domain='0x0000' bus='0x03' slot='0x10' function='0x0'/>
      <address domain='0x0000' bus='0x03' slot='0x10' function='0x2'/>
      <address domain='0x0000' bus='0x03' slot='0x10' function='0x4'/>
      <address domain='0x0000' bus='0x03' slot='0x10' function='0x6'/>
      <address domain='0x0000' bus='0x03' slot='0x11' function='0x0'/>
      <address domain='0x0000' bus='0x03' slot='0x11' function='0x2'/>
      <address domain='0x0000' bus='0x03' slot='0x11' function='0x4'/>
    </capability>
    <iommuGroup number='14'>
      <address domain='0x0000' bus='0x03' slot='0x00' function='0x0'/>
      <address domain='0x0000' bus='0x03' slot='0x00' function='0x1'/>
    </iommuGroup>
  </capability>
</device>
```

```
# virsh nodedev-dumpxml pci_0000_03_11_5
<device>
  <name>pci_0000_03_11_5</name>
  <path>/sys/devices/pci0000:00/0000:00:01.0/0000:03:11.5</path>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igbvf</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>3</bus>
    <slot>17</slot>
    <function>5</function>
    <product id='0x10ca'>82576 Virtual Function</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='phys_function'>
      <address domain='0x0000' bus='0x03' slot='0x00' function='0x1'/>
    </capability>
    <iommuGroup number='35'>
      <address domain='0x0000' bus='0x03' slot='0x11' function='0x5'/>
    </iommuGroup>
  </capability>
</device>
```

この例では、手順 8 で作成した仮想マシンに、仮想機能 **pci_0000_03_10_2** を追加します。仮想機能の **bus**、**slot**、および **function** パラメーターは、デバイスを追加するために必要です。

このパラメーターは、**/tmp/new-interface.xml** などの一時 XML ファイルにコピーします。

```
<interface type='hostdev' managed='yes'>
  <source>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x10' function='0x2'/>
  </source>
</interface>
```



注記

仮想マシンが起動すると、物理アダプターが提供するタイプのネットワークデバイスと、設定された MAC アドレスが表示されます。この MAC アドレスは、ホストおよびゲストの再起動後も変更されません。

以下の ***<interface>*** の例は、オプションの ***<mac address>***、***<virtualport>***、および ***<vlan>*** 要素のシンタックスを示しています。実際には、例にあるように、両方の要素を同時に使用するのではなく、***<vlan>*** 要素または ***<virtualport>*** 要素のいずれかを使用します。

```
...
 <devices>
   ...
   <interface type='hostdev' managed='yes'>
     <source>
       <address type='pci' domain='0' bus='11' slot='16' function='0'/>
     </source>
     <mac address='52:54:00:6d:90:02'>
     <vlan>
        <tag id='42'/>
     </vlan>
     <virtualport type='802.1Qbh'>
       <parameters profileid='finance'/>
     </virtualport>
   </interface>
   ...
 </devices>
```

MAC アドレスを指定しないと、MAC アドレスが自動的に生成されます。***<virtualport>*** 要素は、802.11Qbh ハードウェアスイッチに接続する場合にのみ使用されます。***<vlan>*** 要素は、VLAN タグ付けされた ***42*** に、ゲストのデバイスを透過的に配置します。

8. **仮想マシンへの仮想機能の追加**
上の手順で作成した一時ファイルを使用し、次のコマンドを実行して仮想マシンに仮想機能を追加します。これにより、新しいデバイスがすぐに接続され、以降のゲストの再起動のために保存されます。

```
virsh attach-device MyGuest /tmp/new-interface.xml --live --config
```

virsh attach-device で **--live** を指定すると、新しいデバイスが実行中のゲストに接続されます。**--config** オプションを使用すると、今後ゲストを再起動しても新しいデバイスを使用できるようになります。

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。実行されていないゲストで **--live** オプションが使用されると、**virsh** はエラーを返します。

仮想マシンが新しいネットワークインターフェイスカードを検出します。この新しいカードは、SR-IOV デバイスの仮想機能です。

16.2.3. SR-IOV デバイスを使用した PCI 割り当ての設定

SR-IOV ネットワークカードは、複数の VF を提供します。各 VF は、PCI デバイスの割り当てを使用してゲスト仮想マシンに個別に割り当てることができます。割り当てると、各デバイスは完全な物理ネットワークデバイスとして動作します。これにより、多くのゲスト仮想マシンは、ホスト物理マシン上の単一のスロットのみを使用しながら、直接 PCI デバイス割り当てのパフォーマンス上の利点を得ることができます。

これらの VF は、従来の方法で **<hostdev>** 要素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。ただし、SR-IOV VF ネットワークデバイスには永続的な一意の MAC アドレスがないため、ホストの物理マシンを再起動す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発生します。これを修正するには、ゲスト仮想マシンを起動するたびに、ホストの物理マシンに VF を割り当てる前に MAC アドレスを設定する必要があります。この MAC アドレスやその他のオプションを割り当てるには、以下の手順を参照してください。

手順16.9 SR-IOV で PCI デバイスを割り当てる MAC アドレス、vLAN、および仮想ポートの設定

<hostdev> 要素は、MAC アドレスの割り当て、vLAN タグ ID の割り当て、仮想ポートの割り当てなどの機能固有の項目には使用できません。これは、**<mac>** 要素、**<vlan>** 要素、および **<virtualport>** 要素が **<hostdev>** の有効な子ではないためです。代わりに、この要素は hostdev インターフェイスタイプ **<interface type='hostdev'>** と一緒に使用できます。このデバイスタイプは、**<インタフェース>** と **<hostdev>** を組み合わせたものとして動作します。そのため、libvirt は、ゲスト仮想マシンに PCI デバイスを割り当てる前に、ゲスト仮想マシンの XML 設定ファイルに示されているネットワーク固有のハードウェア/スイッチを初期化します (MAC アドレスの設定、vLAN タグの設定、802.1Qbh スイッチへの関連付けなど)。vLAN タグの設定方法は、「vLAN タグの設定」 を参照してください。

1. **情報の収集**
   <interface type='hostdev'> を使用するには、SR-IOV 対応のネットワークカード、Intel VT-d 拡張機能または AMD IOMMU 拡張機能に対応するホストの物理マシンハードウェアが必要で、割り当てる VF の PCI アドレスを把握している必要があります。

2. **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 (ここでは *guestVM* という名前)。

   ```
   # virsh shutdown guestVM
   ```

3. **編集する XML ファイルを開く**

   ```
   # virsh edit guestVM.xml
   ```

   オプション: **virsh save** コマンドで作成された XML 設定ファイルの場合、次を実行します。

   ```
   # virsh save-image-edit guestVM.xml --running
   ```

   設定ファイル (この例では *guestVM.xml*) がデフォルトのエディターで開きます。詳細は、「ゲスト仮想マシンの設定の編集」 を参照してください。

4. **XML ファイルの編集**
   設定ファイル (*guestVM.xml*) を更新して、以下のような**<devices>** エントリーが表示されるようにします。

図16.11 hostdev インターフェイスタイプのサンプルドメイン XML

```
<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>
```

注記

MAC アドレスを指定しないと、その他のタイプのインターフェイスデバイスと同様に、MAC アドレスが自動的に生成されます。また、**<virtualport>** 要素は、802.11Qgh ハードウェアスイッチに接続する場合にのみ使用されます。802.11 Qbg (VEPA としても知られている) スイッチには現在対応していません。

5. **ゲスト仮想マシンを再起動します。**
virsh start コマンドを実行して、手順 2 でシャットダウンしたゲスト仮想マシンを再起動します。詳細は、「仮想マシンの起動、再開、および復元」 を参照してください。

```
# virsh start guestVM
```

ゲスト仮想マシンは起動時に、設定された MAC アドレスを持つ、物理ホストマシンのアダプターにより提供されたネットワークデバイスを認識します。この MAC アドレスは、ゲスト仮想マシンやホストの物理マシンを再起動しても変更されません。

16.2.4. SR-IOV 仮想機能のプールからの PCI デバイス割り当ての設定

特定の仮想機能 (VF) の PCI アドレスをゲストの設定にハードコーディングすることには、2 つの重大な制限があります。

- ゲスト仮想マシンを起動する場合は、常に指定した VF を使用できる状態にしておく必要があります。そのため、管理者は、各 VF を 1 つのゲスト仮想マシンに永続的に割り当てる必要があります (または、各ゲスト仮想マシンの設定ファイルを変更して、ゲスト仮想マシンが起動するたびに現在使用されていない VF の PCI アドレスを指定します)。
- 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は、そのホスト物理マシンのハードウェアが PCI バス上の同じ場所にある必要があります (または、ゲスト仮想マシンの設定を起動前に変更する必要があります)。

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で、この両方の問題を回避できます。これが完了したら、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します。ゲストを起動するたびに、プールから 1 つの VF が割り当てられ、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するためにプールに戻されます。

手順16.10 デバイスプールの作成

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 (ここでは *guestVM* という名前)。

   ```
   # virsh shutdown guestVM
   ```

2. **設定ファイルの作成**
   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 (名前は *passthrough.xml* など) を作成します。**pf dev='eth3'** は、独自の SR-IOV デバイスーの *Physical Function* (PF) の netdev 名称に置き換えてください。

   以下は、ホスト物理マシンの "eth3' にある PF を持つ SR-IOV アダプターのすべての VF のプールを使用できるようにするネットワーク定義の例です。

   図16.12 サンプルのネットワーク定義ドメイン XML

   ```
   <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>
   ```

3. **新しい XML ファイルの読み込み**
   /tmp/passthrough.xml を、前の手順で作成した XML ファイルの名前と場所に置き換え、次のコマンドを入力します。

   ```
   # virsh net-define /tmp/passthrough.xml
   ```

4. **ゲストの再起動**
   passthrough.xml を、前の手順で作成した XML ファイルの名前に置き換えて、次のコマンドを実行します。

   ```
   # virsh net-autostart passthrough # virsh net-start passthrough
   ```

5. **ゲスト仮想マシンの再起動**
   virsh start コマンドを実行して、最初の手順でシャットダウンしたゲスト仮想マシンを再起動します (例では、ゲスト仮想マシンのドメイン名として guestVM を使用します)。詳細は、「仮想マシンの起動、再開、および復元」 を参照してください。

   ```
   # virsh start guestVM
   ```

6. **デバイスのパススルーの開始**
   表示されているデバイスは 1 つだけですが、libvirt は、ゲスト仮想マシンが次のようなドメイン XML のインターフェイス定義で初めて起動されたときに、その PF に関連付けられているすべての VF のリストを自動的に取得します。

   図16.13 インターフェイスネットワーク定義のサンプルドメイン XML

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

7. **検証**
   ネットワークを使用する最初のゲストを起動したら、**virsh net-dumpxml *passthrough*** コマンドを実行し、これを確認できます。以下のような出力が得られます。

   図16.14 XML ダンプファイルの*passthrough*コンテンツ

```
<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>
```

16.2.5. SR-IOV の制限

SR-IOV は、以下のデバイスでの徹底したテストのみが行われています。

- Intel® 82576NS ギガビットイーサネットコントローラー (**igb** ドライバー)
- Intel® 82576EB ギガビットイーサネットコントローラー (**igb** ドライバー)
- Intel® 82599ES 10 ギガビットイーサネットコントローラー (**ixgbe** ドライバー)
- Intel® 82599EB 10 ギガビットイーサネットコントローラー (**ixgbe** ドライバー)

その他の SR-IOV デバイスは動作するかもしれませんが、リリース時にはテストされていません。

16.3. USB デバイス

本セクションでは、USB デバイスを扱うために必要なコマンドについて説明します。

16.3.1. ゲスト仮想マシンへの USB デバイスの割り当て

Web カメラ、カードリーダー、ディスクドライブ、キーボード、マウスなどのほとんどのデバイスは、USB ポートとケーブルを使用してコンピューターに接続されています。このようなデバイスをゲスト仮想マシンに渡すには、以下の 2 つの方法があります。

- USB パススルーを使用する - これにより、デバイスが、ゲスト仮想マシンをホストしているホストの物理マシンに物理的に接続されます。この場合は SPICE は必要ありません。ホストの USB デバイスは、コマンドラインまたは**virt-manager** でゲストに渡すことができます。**virt manager** の指示については、「ゲスト仮想マシンへの USB デバイスの接続」 を参照してください。**virt-manager**の指示は、ホットプラグやホットアンプラグのデバイスには適していないことに注意してください。ホットプラグ/ホットアンプラグを行う場合は、手順20.4「ゲスト仮想マシンが使用するホットプラグ USB デバイス」 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、データセンターで実行されているホスト物理マシンがある場合に最適です。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 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接続できます。SPICE クライアントは、デバイスをデータセンターのホスト物理マシンにリダイレクトするため、シンクライアントで実行しているゲスト仮想マシンで使用できます。virt-manager を使用した手順は、「USB リダイレクト」 を参照してください。

16.3.2. USB デバイスリダイレクトの制限の設定

リダイレクトから特定のデバイスを除外するには、filter プロパティーを ***-device usb-redir*** に渡します。filter プロパティーは、フィルタールールで設定される文字列を取ります。ルールの形式は以下のとおりです。

```
<class>:<vendor>:<product>:<version>:<allow>
```

値 ***-1*** を使用して、特定のフィールドに任意の値を使用できるように指定します。同じコマンドラインで、| を区切り文字として使用し、複数のルールを使用できます。デバイスが渡されたルールのいずれにも一致しない場合は、それをリダイレクトできないことに注意してください。

例16.1 ゲスト仮想マシンを使用してリダイレクトを制限する例

1. ゲスト仮想マシンを準備します。
2. ゲスト仮想マシンのドメイン XML ファイルに、以下のコードの抜粋を追加します。

```
<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 デバイスをホストの物理マシンに接続し、**virt-manager** を使用してゲスト仮想マシンに接続します。

5. メニューで **USB device selection** をクリックすると、Some USB devices are blocked by host policy というメッセージが表示されます。**OK** をクリックして確定し、続行します。

   フィルターが有効になります。

6. フィルターが正しくキャプチャーされていることを確認するには、USB デバイスのベンダーと製品を確認してから、USB リダイレクトを可能にするために、ホストの物理マシンのドメイン XML で次の変更を行います。

```
<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てから、**virt-viewer** を使用してゲスト仮想マシンに接続します。これで、USB デバイスは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。

16.4. デバイスコントローラーの設定

ゲスト仮想マシンのアーキテクチャーによっては、一部のデバイスバスは複数回表示され、仮想デバイスのグループは仮想コントローラーに関連付けられています。通常、libvirt は、明示的な XML マークアップを必要とせずに、自動的にこのようなコントローラーを推測できますが、場合によっては仮想コントローラー要素を明示的に設定する方が良い場合があります。

図16.15 仮想コントローラーのドメイン XML の例

```
...
<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>
...
```

各コントローラーには必須の属性 **<controller type>** があり、これは次のいずれかになります。

- ide
- fdc
- scsi

- sata
- usb
- ccid
- virtio-serial
- pci

<controller> 要素には必須の属性 **<controller index>** があります。これは、バスコントローラーが発生した順序を表す 10 進数の整数です (**<address>** 要素のコントローラー属性で使用されます)。**<controller type ='virtio-serial'>** には、追加のオプション属性 (**ports** および **vectors**) が 2 つあります。これは、コントローラーを介して接続できるデバイスの数を制御します。

<controller type ='scsi'> の場合、以下の値を持つことができる任意の属性**model** モデルがあります。

- auto
- buslogic
- ibmvscsi
- lsilogic
- lsisas1068
- lsisas1078
- virtio-scsi
- vmpvscsi

<controller type ='usb'> の場合、以下の値を持つことができる任意の属性**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

ゲスト仮想マシンに対して USB バ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。 .

コントローラー自体が PCI バスまたは USB バスにある場合は、オプションのサブ要素 **<address>** は、「デバイスのアドレスの設定」 に示したセマンティクスを使用して、コントローラーとマスターバスの正確な関係を指定できます。

オプションの sub-element **<driver>** は、ドライバー固有のオプションを指定できます。現在、コントローラーのキュー数を指定する属性キューのみをサポートします。パフォーマンスを最適化するには、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の完全なリレーション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マスターと同じバスにあるため、コンパニオン **index** は等しい値になります。

使用できる XML の例を以下に示します。

図16.16 USB コントローラーのドメイン XML の例

```
  ...
<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>
...
```

PCI コントローラーには、以下の値を持つオプションの **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'** を使用する pci-root コントローラーは自動追加され、PCI デバイスを使用する必要があります。pci-root にはアドレスがありません。PCI ブリッジは、**model='pci-root'** が提供する 1 つのバスに収まらないデバイスが多すぎる場合、または 0 を超える PCI バス番号が指定されている場合に自動追加されます。PCI ブリッジは手動で指定することもできますが、そのアドレスには、指定されている PCI コントローラーが提供する PCI バスのみが表示されるようにしてください。PCI コントローラーインデックスにギャップがあると、設定が無効になる場合があります。以下の XML サンプルは、**<devices>** セクションに追加できます。

図16.17 PCI ブリッジのドメイン XML の例

```
...
<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>
...
```

暗黙的な PCI Express (PCIe) バスを提供するマシンタイプ (Q35 チップセットに基づくマシンタイプなど) の場合、**index='0'** を使用する pcie-root コントローラーはドメインの設定に自動的に追加されます。pcie-root にはアドレスはありませんが、31 スロット (1-31 の番号) を提供し、PCIe デバイスの接続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で標準的な PCI デバイスを接続するには、**model='dmi-to-pci-bridge'** を持つ 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-root が提供するように) PCIe スロットに接続し、それ自体で 31 個の標準的な PCI スロットを提供します (これはホットプラグできません)。ゲストシステムにホットプラグ可能な PCI スロットを確保するために、pci-bridge コントローラーも自動的に作成され、自動作成された dmi-to-pci-bridge コントローラーのスロットの 1 つに接続されます。PCI アドレスが libvirt により自動決定されるすべてのゲストデバイスは、この pci-bridge デバイスに配置されます。

図16.18 PCIe (PCI express) のようなドメイン XML

```
  ...
<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>
...
```

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。

図16.19 USB3/xHCI デバイスのドメイン XML の例

```
  ...
<devices>
  <controller type='usb' index='3' model='nec-xhci'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0f' function='0x0'/>
  </controller>
</devices>
  ...
```

16.5. デバイスのアドレスの設定

多くのデバイスには、ゲスト仮想マシンに提示される仮想バス上のデバイスの場所を説明するのに使用されるオプションの **<address>** サブ要素があります。アドレス (またはアドレス内の任意の属性) が入力で省略されると、libvirt は適切なアドレスを生成します。レイアウトをさらに制御する必要がある場合は明示的なアドレスが必要になります。**<address>** 要素を含むドメイン XML デバイスの例は、図 16.9「PCI デバイスの割り当ての XML 例」 を参照してください。

各アドレスには、デバイスが稼働しているバスを説明する必須の属性 **type** があります。特定のデバイスに使用するアドレスの選択は、デバイスやゲスト仮想マシンのアーキテクチャーによって一部が制約されます。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 ゲスト仮想マシンアーキテクチャー で **type='pci'** を使用します。各アドレスタイプには、表に記載されているように、バス上のどこにデバイスを配置するかを制御するオプションの属性がさらにあります。

表16.1 サポートされているデバイスのアドレスタイプ

アドレスの種類	説明
type='pci'	PCI アドレスには、以下の追加属性があります。 ● ドメイン (2 バイトの 16 進数の整数で、現在 qemu で使用されていません) ● bus (0 から 0xff までの 16 進数の値) ● slot (0x0 から 0x1f までの 16 進数の値) ● 関数 (0 から 7 までの値) ● PCI 制御レジスターの特定のスロット/機能の多機能ビットをオンにする多機能コントロールは、デフォルトではオフに設定されていますが、複数の機能が使用されるスロットの機能 0 ではオンに設定する必要があります。
type='drive'	ドライブアドレスには、以下の追加属性があります。 ● コントローラー (2 桁のコントローラー番号) ● バス (2 桁のバス番号) ● ターゲット (2 桁のバス番号) ● ユニット (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。 ● コントローラー (2 桁のコントローラー番号) ● バス (2 桁のバス番号) ● スロット (バス内の 2 桁のスロット)

アドレスの種類	説明
type='ccid'	スマートカード用の CCID アドレスには、以下の追加属性があります。 ● バス (2 桁のバス番号) ● スロット属性 (バス内の 2 桁のスロット)
type='usb'	USB アドレスには、以下の追加属性があります。 ● bus (0 から 0xfff までの 16 進数の値) ● ポート (1.2 または 2.1.3.1 などの最大 4 オクテットのドット表記)
type='isa'	ISA アドレスには、以下の追加属性があります。 ● iobase ● irq

16.6. 乱数ジェネレーターデバイス

乱数ジェネレーターは、オペレーティングシステムのセキュリティーにとって非常に重要です。仮想オペレーティングシステムのセキュリティーを保護するために、Red Hat Enterprise Linux 7 には、要求に応じて新しいエントロピーをゲストに提供できる仮想ハードウェア乱数ジェネレーターデバイスである **virtio-rng** が同梱されています。

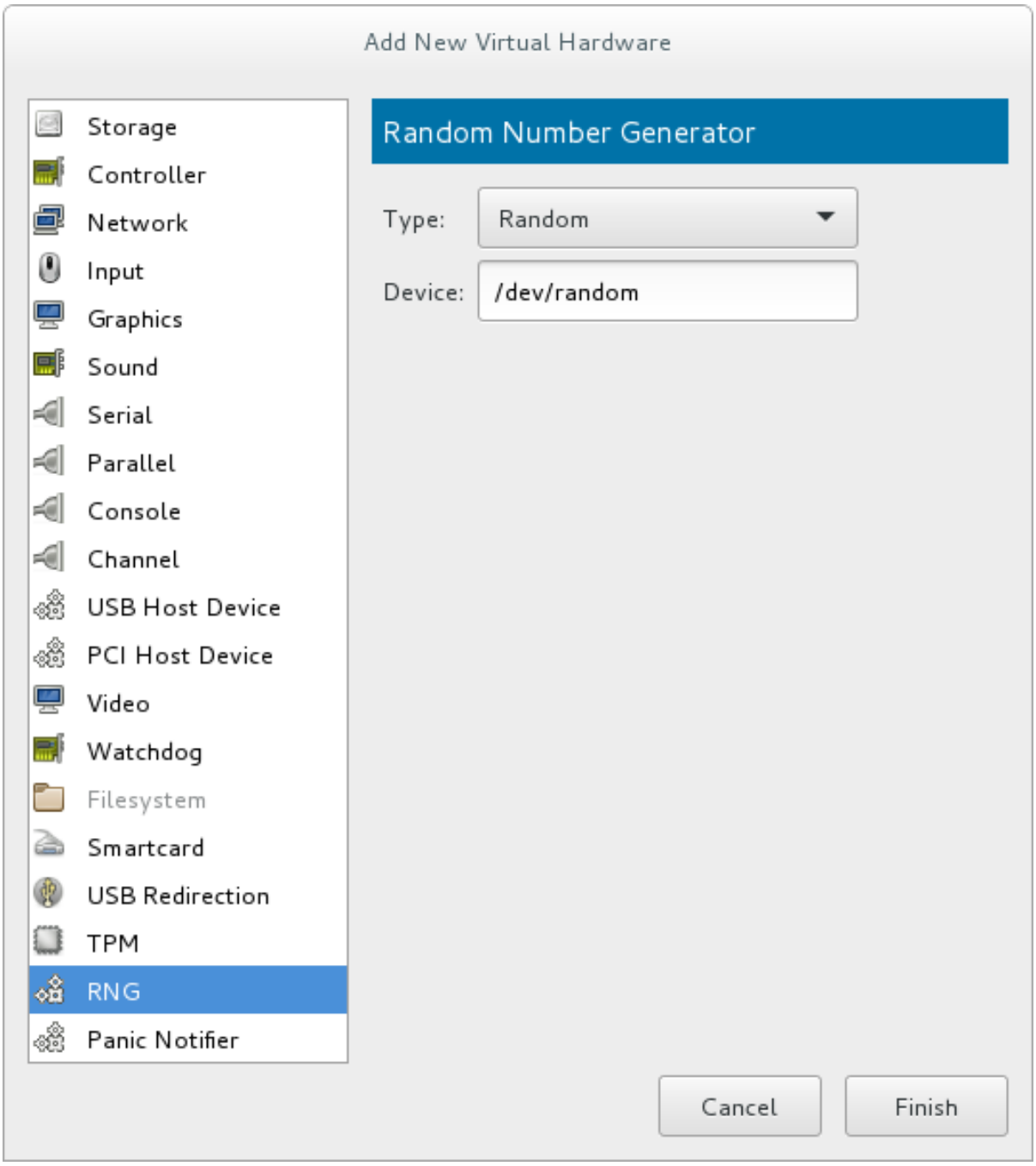
ホスト物理マシンでは、ハードウェア RNG インターフェイスにより、**/dev/hwrng** に chardev が作成されます。これは開いてから読み込み、ホスト物理マシンからエントロピーを取得できます。**rngd** デーモンと連携すると、ホストの物理マシンのエントロピーは、ランダム性の主要なソースであるゲスト仮想マシンの**/dev/random** にルーティングできます。

乱数ジェネレーターの使用は、キーボード、マウス、その他の入力などのデバイスがゲスト仮想マシンで十分なエントロピーを生成するのに十分でない場合に特に役立ちます。仮想乱数ジェネレーターデバイスを使用すると、ホストの物理マシンがエントロピーを通過してゲストの仮想マシンのオペレーティングシステムに到達できます。 この手順は、コマンドラインまたは virt-manager インターフェイスのいずれかを使用して実行できます。手順は、以下を参照してください。**virtio-rng** の詳細は、Red Hat Enterprise Linux Virtual Machines: Access to Random Numbers Made Easy を参照してください。

手順16.11 Virtual Machine 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、**Edit** メニューから**Virtual Machine Details**を選択して、指定したゲスト仮想マシンの Details ウィンドウを開きます。
3. **Add Hardware** ボタンをクリックします。
4. **Add New Virtual Hardware**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16.20 Random Number Generator ウィンドウ



目的のパラメーターを入力し、完了したら **Finish** を押します。パラメーターは、virtio-rng 要素 で説明されています。

手順16.12 コマンドラインツールを使用した virtio-rng の実装

1. ゲスト仮想マシンをシャットダウンします。
2. **virsh edit *domain-name*** コマンドを使用して、目的のゲスト仮想マシンの XML ファイルを開きます。
3. **<devices>** 要素を編集して、以下の内容を追加します。

図16.21 乱数ジェネレーターデバイス

```
...
<devices>
  <rng model='virtio'>
    <rate period='2000' bytes='1234'/>
    <backend model='random'>/dev/random</backend>
    <!-- OR -->
    <backend model='egd' type='udp'>
      <source mode='bind' service='1234'/>
      <source mode='connect' host='1.2.3.4' service='1234'/>
    </backend>
  </rng>
</devices>
...
```

乱数ジェネレーターデバイスでは、以下の XML 属性および要素が使用できます。

virtio-rng 要素

- **<model>** - 必要な **model** 属性は、提供される RNG デバイスタイプを指定します。
- **<backend model>** - **<backend>** 要素は、ゲストで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。対応しているソースモデルには、**'random'** および **'egd'** があります。
  - **<backend model='random'>** - この **<backend>** タイプでは、ノンブロッキングキャラクターデバイスが入力として想定されます。このようなデバイスの例は、**/dev/random** および **/dev/urandom** です。ファイルネームは、**<backend>** 要素のコンテンツとして指定されます。ファイル名を指定しないと、ハイパーバイザーのデフォルトが使用されます。
  - **<backend model='egd'>** - このバックエンドは、EGD プロトコルを使用して送信元に接続します。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ホストの物理マシンインターフェイスを参照してください。

16.7. GPU デバイスの割り当て

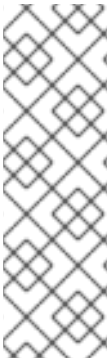
GPU をゲストに割り当てるには、次のいずれかの方法を使用します。

- GPU PCI デバイスの割り当て - この方法を使用すると、ホストから GPU デバイスを削除し、1 つのゲストに割り当てることができます。
- NVIDIA vGPU の割り当て - この方法を使用すると、物理 GPU から複数の *仲介デバイス*を作成し、これらのデバイスを仮想 GPU として複数のゲストに割り当てることができます。これは、選択した NVIDIA GPU でのみ対応しており、1 つのゲストに割り当てることができる仲介デバイスは 1 つだけです。

16.7.1. GPU PCI デバイスの割り当て

Red Hat Enterprise Linux 7 では、以下の PCIe ベースの GPU デバイスから、非 VGA グラフィックデバイスへの PCI デバイスの割り当てに対応しています。

- NVIDIA Quadro K-Series、M-Series、P-Series、およびそれ以降のアーキテクチャー (モデル 2000 シリーズ以降)
- NVIDIA Tesla K-Series、M-Series、およびそれ以降のアーキテクチャー



注記

VM に接続できる GPU の数は、割り当てられた PCI デバイスの最大数 (RHEL 7 では現在 32) によって制限されます。ただし、VM に複数の GPU を接続すると、ゲストのメモリーマップド I/O (MMIO) で問題が発生する可能性があり、その結果、GPU が VM で使用できなくなる可能性があります。

これらの問題を回避するには、より大きな 64 ビット MMIO 空間を設定し、vCPU 物理アドレスビットを設定して、拡張された 64 ビット MMIO 空間をアドレス指定可能にします。

ゲスト仮想マシンに GPU を割り当てるには、ホストマシンで I/O メモリー管理ユニット (IOMMU) を有効にし、**lspci** コマンドを使用して GPU デバイスを識別し、デバイスをホストから切り離して、ゲストに接続します。以下の手順に従って、ゲストで Xorg を設定します。

手順16.13 ホストマシンカーネルで IOMMU サポートを有効にします。

1. **カーネルコマンドラインの編集**
   Intel VT-d システムの場合は、***intel_iommu=on*** パラメーターおよび ***iommu=pt*** パラメーターをカーネルコマンドラインに追加することで IOMMU がアクティブになります。AMD-Vi システムの場合、必要な選択肢は***iommu=pt*** のみです。この項目を有効にするには、以下のように**GRUB_CMDLINX_LINUX** 行を**/etc/sysconfig/grub** 設定ファイルに追加または修正します。

   ```
   GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
   vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
   vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ]  &&
   /usr/sbin/rhcrashkernel-param || :) rhgb quiet intel_iommu=on iommu=pt"
   ```

   注記

   IOMMU の詳細は、付録E *IOMMU グループの使用* を参照してください。

2. **ブートローダー設定の再生成**
   カーネルコマンドラインの変更を適用するには、**grub2-mkconfig** コマンド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 ベースのホストを使用している場合は、ターゲットファイルが**/etc/grub2-efi.cfg** であることに注意してください。

3. **ホストを再起動します。**
   変更を有効にするには、ホストマシンを再起動します。

   ```
   # reboot
   ```

手順16.14 GPU デバイスのホスト物理マシンドライバーへのバインディングからの除外

GPU の割り当てでは、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が推奨されます。このようなドライバーは、デバイスの動的なバインド解除に対応していない場合が多いためです。

1. **PCI バスアドレスの特定**
   デバイスの PCI バスアドレスおよび 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、NVIDIA Quadro カードや GRID カードなどの VGA コントローラーを使用します。

   ```
   # lspci -Dnn | grep VGA
   0000:02:00.0 VGA compatible controller [0300]: NVIDIA Corporation GK106GL [Quadro K4000] [10de:11fa] (rev a1)
   ```

   検索の結果、このデバイスの PCI バスアドレスが 0000:02:00.0 であり、デバイスの PCI ID が 10de:11fa であることが分かります。

2. **ネイティブホストマシンのドライバーが GPU デバイスを使用できないようにします。**
   ネイティブホストマシンのドライバーが GPU デバイスを使用しないようにするには、pci-stub ドライバーで PCI ID を使用できます。これには、以下の例のように、PCI ID を値として、**/etc/sysconfig/grub** 設定ファイルにある **GRUB_CMDLINX_LINUX** 行に **pci-stub.ids** オプションを追加します。

   ```
   GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
   vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
   vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ]  &&
   /usr/sbin/rhcrashkernel-param || :) rhgb quiet intel_iommu=on iommu=pt pci-
   stub.ids=10de:11fa"
   ```

   pci-stub に追加の PCI ID を追加する場合は、コンマで区切ります。

3. **ブートローダー設定の再生成**
   この**grub2-mkconfig**を使用して、ブートローダー設定を再生成し、この項目を追加します。

   ```
   # grub2-mkconfig -o /etc/grub2.cfg
   ```

   UEFI ベースのホストを使用している場合は、ターゲットファイルが**/etc/grub2-efi.cfg** であることに注意してください。

4. **ホストマシンを再起動します。**
   変更を有効にするには、ホストマシンを再起動します。

   ```
   # reboot
   ```

手順16.15 オプション: GPU IOMMU 設定の編集

GPU デバイスを接続する前に、その IOMMU 設定を編集して、GPU がゲストで適切に機能するようにする必要があります。

1. **GPU の XML 情報を表示します。**
   GPU の設定を XML 形式で表示するには、最初に **pci_** を追加し、区切り文字をアンダースコアに変換して、PCI バスアドレスを libvirt 互換形式に変換する必要があります。この例では、0000:02:00.0 バスアドレスで識別される GPU PCI デバイス (以前の手順 で取得したもの) が **pci_0000_02_00_0** になります。**virsh nodedev-dumpxml** でデバイスの libvirt アドレスを使用し、XML 設定を表示します。

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

```
<device>
 <name>pci_0000_02_00_0</name>
 <path>/sys/devices/pci0000:00/0000:00:03.0/0000:02:00.0</path>
 <parent>pci_0000_00_03_0</parent>
 <driver>
  <name>pci-stub</name>
 </driver>
 <capability type='pci'>
  <domain>0</domain>
  <bus>2</bus>
  <slot>0</slot>
  <function>0</function>
  <product id='0x11fa'>GK106GL [Quadro K4000]</product>
  <vendor id='0x10de'>NVIDIA Corporation</vendor>
    <!-- pay attention to the following lines -->
  <iommuGroup number='13'>
   <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
   <address domain='0x0000' bus='0x02' slot='0x00' function='0x1'/>
  </iommuGroup>
  <pci-express>
   <link validity='cap' port='0' speed='8' width='16'/>
   <link validity='sta' speed='2.5' width='16'/>
  </pci-express>
 </capability>
</device>
```

XML の **<iommuGroup>** 要素に注意してください。iommuGroup は、IOMMU の機能および PCI バストポロジーにより、その他のデバイスから分離されていると見なされるデバイスのセットを示します。iommuGroup 内のすべてのエンドポイントデバイス (PCIe ルートポート、ブリッジ、またはスイッチポートではないデバイス) をゲストに割り当てるには、ネイティブホストドライバーからバインドを解除する必要があります。この例では、グループは GPU デバイス (0000:02:00.0) とコンパニオンオーディオデバイス (0000:02:00.1) で設定されています。詳細は、付録E *IOMMU グループの使用* を参照してください。

2. **IOMMU 設定の調整**
   この例では、レガシー割り込みサポートのハードウェアの問題により、NVIDIA オーディオ機能の割り当てはサポートされていません。また、GPU オーディオ機能は、通常、GPU 自体がないと役に立ちません。そのため、GPU をゲストに割り当てるには、最初にオーディオ機能をネイティブホストドライバーから切り離す必要があります。これは、以下のいずれかを使用して行います。

   - 手順16.14「GPU デバイスのホスト物理マシンドライバーへのバインディングからの除外」で詳述しているように、デバイスの PCI ID を検出し、**/etc/sysconfig/grub** ファイルの **pci-stub.ids** オプションに追加します。
   - 以下の例のように、**virsh nodedev-detach** を使用します。

     ```
     # virsh nodedev-detach pci_0000_02_00_1
     Device pci_0000_02_00_1 detached
     ```

手順16.16 GPU のアタッチ

GPU は、次のいずれかの方法を使用してゲストにアタッチできます。

1. **Virtual Machine Manager** インターフェイスの使用詳細は、「virt-manager を使用した PCI デバイスの割り当て」 を参照してください。

2. GPU 用の XML 設定フラグメントを作成し、**virsh attach-device** でアタッチする場合は、次のコマンドを実行します。

    1. 以下のような XML をデバイスに作成します。

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

    2. これをファイルに保存し、**virsh attach-device [domain] [file] --persistent** を実行して、ゲスト設定に XML を含めます。割り当てられた GPU は、ゲストマシンにある既存のエミュレートグラフィックデバイスに追加されることに注意してください。割り当てられた GPU は、仮想マシンではセカンダリーグラフィックデバイスとして扱われます。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ていないため、ゲストの XML でエミュレートされたグラフィックスデバイスは削除しないでください。

3. **virsh edit** を使用したゲスト XML 設定の編集および適切な XML セグメントの手動による追加

手順16.17 ゲストの Xorg 設定の変更

ゲストの GPU の PCI バスアドレスは、ホストの PCI バスアドレスとは異なります。ホストが GPU を適切に使用できるようにするには、割り当てられた GPU アドレスを使用するようにゲストの Xorg ディスプレイサーバーを設定します。

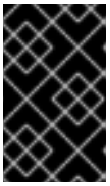
1. ゲストで **lspci** コマンドを使用して GPU の PCI バスアドレスを決定します。

    ```
    # lspci | grep VGA
    00:02.0 VGA compatible controller: Device 1234:111
    00:09.0 VGA compatible controller: NVIDIA Corporation GK106GL [Quadro K4000] (rev a1)
    ```

    この例では、バスアドレスは 00:09.0 です。

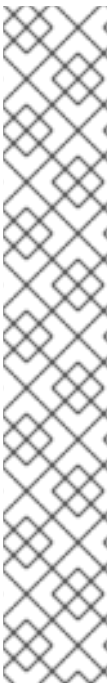
2. ゲストの **/etc/X11/xorg.conf** ファイルに、検出されたアドレスを以下のように調整した **BusID** オプションを追加します。

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



重要

手順 1 で検出したバスアドレスが 16 進数の場合は、区切り文字の値を 10 進数に変換する必要があります。たとえば、00:0a.0 は PCI:0:10:0 に変換する必要があります。



注記

ゲストで、割り当てられた NVIDIA GPU を使用する場合は、NVIDIA ドライバーのみがサポートされます。その他のドライバーは動作せず、エラーが生成される場合があります。Red Hat Enterprise Linux 7 ゲストの場合は、インストール時にカーネルコマンドラインの オプション **modprobe.blacklist=nouveau** を使用して、nouveau ドライバーをブラックリストに追加できます。 その他のゲスト仮想マシンの詳細は、オペレーティングシステム固有のドキュメントを参照してください。

ゲストのオペレーティングシステムによっては、読み込んだ NVIDIA ドライバーで、エミュレートグラフィックと割り当てたグラフィックの両方の使用に同時に対応したり、エミュレートグラフィックを無効にしたりできます。割り当てられたグラフィックフレームバッファーへのアクセスは、**virt-manager** などのアプリケーションでは利用できないことに注意してください。割り当てられた GPU が物理ディスプレイに接続されていない場合、GPU デスクトップにアクセスするために、ゲストベースのリモートソリューションが必要になる場合があります。すべての PCI デバイスの割り当てと同様、割り当てられた GPU を持つゲストの移行はサポートされず、各 GPU は 1 つのゲストのみが所有します。ゲストオペレーティングシステムによっては、GPU のホットプラグ対応が利用できる場合があります。

16.7.2. NVIDIA vGPU の割り当て

NVIDIA vGPU 機能を使用すると、物理 GPU デバイスを、*仲介デバイス* と呼ばれる複数の仮想デバイスに分割できます。仲介されたデバイスは、仮想 GPU として複数のゲストに割り当てることができます。その結果、このゲストは 1 つの物理 GPU のパフォーマンスを共有します。

重要

この機能は、限られた NVIDIA GPU セットでのみ利用できます。このデバイスの最新のリストは、NVIDIA GPU ソフトウェアのドキュメント を参照してください。

16.7.2.1. NVIDIA vGPU の設定

vGPU 機能を設定するには、最初に GPU デバイス用の NVIDIA vGPU ドライバーを取得し、仲介デバイスを作成して、目的のゲストマシンに割り当てる必要があります。

1. NVIDIA vGPU ドライバーを取得して、システムにインストールします。手順は NVIDIA ドキュメント を参照してください。

2. NVIDIA ソフトウェアのインストーラーが **/etc/modprobe.d/nvidia-installer-disable-nouveau.conf** ファイルを作成しなかった場合は、**/etc/modprobe.d/** ディレクトリーに、任意の名前で **conf** ファイルを作成します。ファイルに次の行を追加します。

```
blacklist nouveau
options nouveau modeset=0
```

3. 現在のカーネル用に初期 ramdisk を再生成してから再起動します。

```
# dracut --force
# reboot
```

仲介されたデバイスで、以前対応していたカーネルバージョンを使用する必要がある場合は、インストールしたすべてのカーネルバージョンに対して初期 ramdisk を再生成します。

```
# dracut --regenerate-all --force
# reboot
```

4. **nvidia_vgpu_vfio** モジュールがカーネルによりロードされていて、**nvidia-vgpu-mgr.service** サービスが実行していることを確認してください。

```
# lsmod | grep nvidia_vgpu_vfio
nvidia_vgpu_vfio 45011 0
nvidia 14333621 10 nvidia_vgpu_vfio
mdev 20414 2 vfio_mdev,nvidia_vgpu_vfio
vfio 32695 3 vfio_mdev,nvidia_vgpu_vfio,vfio_iommu_type1
# systemctl status nvidia-vgpu-mgr.service
nvidia-vgpu-mgr.service - NVIDIA vGPU Manager Daemon
   Loaded: loaded (/usr/lib/systemd/system/nvidia-vgpu-mgr.service; enabled; vendor preset:
disabled)
   Active: active (running) since Fri 2018-03-16 10:17:36 CET; 5h 8min ago
 Main PID: 1553 (nvidia-vgpu-mgr)
 [...]
```

5. デバイスの UUID を **/sys/class/mdev_bus/*pci_dev*/mdev_supported_types/*type-id*/create** に書き込みます。*pci_dev* は、ホスト GPU の PCI アドレスで、 *type-id* は、ホスト GPU タイプの ID です。

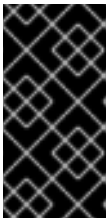
   次の例は、NVIDIA Tesla P4 カードの **nvidia-63** vGPU タイプの仲介デバイスを作成する方法を示しています。

```
# uuidgen
30820a6f-b1a5-4503-91ca-0c10ba58692a
# echo "30820a6f-b1a5-4503-91ca-0c10ba58692a" >
/sys/class/mdev_bus/0000:01:00.0/mdev_supported_types/nvidia-63/create
```

   特定のデバイスの*type-id*値は、**section 1.3.1 を参照してください。仮想 GPU ソフトウェアのドキュメント の仮想 GPU タイプ**。Linux ゲストでは、**GRID P4-2Q** などの Q シリーズ NVIDIA vGPU のみが、仲介デバイス GPU タイプとしてサポートされます。

6. 次の行を、vGPU リソースを共有するゲストの XML 設定の **<devices/>** セクションに追加します。前の手順で **uuidgen** コマンドで生成した UUID 値を使用します。各 UUID は、一度に 1 つのゲストにしか割り当てることができません。

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>
  </source>
</hostdev>
```



重要

割り当てられたゲストで vGPU 仲介デバイスが正しく動作するようにするには、ゲストに対して NVIDIA vGPU ゲストソフトウェアライセンスを設定する必要があります。詳細および手順は、NVIDIA の仮想 GPU ソフトウェアのドキュメント を参照してください。

16.7.2.2. NVIDIAvGPU を使用したビデオストリーミング用の VNC コンソールのセットアップと使用

テクノロジープレビューとして、Red Hat Enterprise Linux 7.7 以降では、仮想ネットワークコンピューティング (VNC) コンソールを、NVIDIA vGPU を含む GPU ベースの仲介デバイスとともに使用できます。その結果、VNC を使用して、NVIDIA vGPU デバイスが提供する高速グラフィカル出力を表示できます。

警告

この機能は現在、テクノロジープレビューとしてのみ提供されており、Red Hat ではサポートされていません。したがって、実稼働環境で以下の手順を使用することは強く推奨しません。

仮想マシンの VNC コンソールで vGPU 出力レンダリングを設定するには、次の手順を実行します。

1. 「NVIDIA vGPU の設定」 の説明に従って、NVIDIA vGPU ドライバーをインストールし、システムに NVIDIA vGPU を設定します。仲介されたデバイスの XML 設定に **display='on'** パラメーターが含まれていることを確認します。以下に例を示します。

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci' display='on'>
  <source>
    <address uuid='ba26a3e2-8e1e-4f39-9de7-b26bd210268a'/>
  </source>
</hostdev>
```

2. 必要に応じて、仮想マシンのビデオモデルタイプを **none** に設定します。以下に例を示します。

```
<video>
  <model type='none'/>
</video>
```

指定しない場合は、2 つの異なるディスプレイ出力 (エミュレートの Cirrus カードまたは QXL カードからの 1 つと、NVIDIA vGPU からの 1 つ) が出力されます。また、現在 **model type='none'** を使用すると、ドライバーが初期化されるまでブートグラフィカルアウトプットを表示できなくなります。その結果、最初に表示されるグラフィック出力はログイン画面になります。

3. 仮想マシンのグラフィックタイプの XML 設定が **vnc** であることを確認します。

   以下に例を示します。

```
<graphics type='vnc' port='-1' autoport='yes'>
  <listen type='address'/>
</graphics>
```

4. 仮想マシンを起動します。

5. VNC ビューアー リモートデスクトップクライアントを使用して仮想マシンに接続します。

注記

仮想マシンが、エミュレートされた VGA をプライマリービデオデバイスとして、vGPU をセカンダリーデバイスとして設定されている場合は、**ctrl+alt+2** キーボードショートカットを使用して、vGPU ディスプレイに切り替えます。

16.7.2.3. NVIDIA vGPU デバイスの削除

仲介 vGPU デバイスを削除するには、デバイスが非アクティブのときに次のコマンドを使用します。*uuid* は、デバイスの UUID (**30820a6f-b1a5-4503-91ca-0c10ba58692a** など) に置き換えます。

```
# echo 1 > /sys/bus/mdev/devices/uuid/remove
```

ゲストで現在使用されている vGPU デバイスを削除しようとすると、以下のエラーが発生することに注意してください。

```
echo: write error: Device or resource busy
```

16.7.2.4. NVIDIA vGPU 機能のクエリー

特定タイプの仲介デバイスをどのぐらい作成できるかなど、システムの仲介デバイスの詳細を取得するには、**virsh nodedev-list --cap mdev_types** コマンドおよび **virsh nodedev-dumpxml** コマンドを使用します。たとえば、以下は、Tesla P4 カードで利用可能な vGPU タイプを表示します。

```
$ virsh nodedev-list --cap mdev_types
pci_0000_01_00_0
$ virsh nodedev-dumpxml pci_0000_01_00_0
<...>
  <capability type='mdev_types'>
    <type id='nvidia-70'>
      <name>GRID P4-8A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>1</availableInstances>
    </type>
    <type id='nvidia-69'>
      <name>GRID P4-4A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>2</availableInstances>
    </type>
```

```
    <type id='nvidia-67'>
      <name>GRID P4-1A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>8</availableInstances>
    </type>
    <type id='nvidia-65'>
      <name>GRID P4-4Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>2</availableInstances>
    </type>
    <type id='nvidia-63'>
      <name>GRID P4-1Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>8</availableInstances>
    </type>
    <type id='nvidia-71'>
      <name>GRID P4-1B</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>8</availableInstances>
    </type>
    <type id='nvidia-68'>
      <name>GRID P4-2A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>4</availableInstances>
    </type>
    <type id='nvidia-66'>
      <name>GRID P4-8Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>1</availableInstances>
    </type>
    <type id='nvidia-64'>
      <name>GRID P4-2Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>4</availableInstances>
    </type>
  </capability>
</...>
```

16.7.2.5. NVIDIA vGPU のリモートデスクトップのストリーミングサービス

以下のリモートデスクトップストリーミングサービスは、Red Hat Enterprise Linux 7 の NVIDIA vGPU 機能での使用が正常にテストされています。

- HP-RGS
- Mechdyne TGX - 現在、Windows Server 2016 ゲストで Mechdyne TGX を使用することはできません。
- NICE DCV - このストリーミングサービスを使用する場合は、解像度を動的にすると、場合によっては画面が黒くなるため、Red Hat は、解像度を固定する設定を使用することを推奨します。

16.7.2.6. NVIDIA vGPU を使用したビデオストリーミング用の VNC コンソールの設定

導入部分

テクノロジープレビュー として、Red Hat Enterprise Linux 8 では、仮想ネットワークコンピューティング (VNC) コンソールを、NVIDIA vGPU を含む GPU ベースの仲介デバイスとともに使用できます。その結果、VNC を使用して、NVIDIA vGPU デバイスが提供する高速グラフィカル出力を表示できます。

重要

テクノロジープレビューのため、この機能は Red Hat では対応していません。したがって、実稼働環境で以下の手順を使用することは強く推奨しません。

設定

仮想マシンの VNC コンソールで vGPU 出力レンダリングを設定するには、次の手順を実行します。

1. 「NVIDIA vGPU の割り当て」 の説明に従って、NVIDIA vGPU ドライバーをインストールし、ホストに NVIDIA vGPU を設定します。仲介されたデバイスの XML 設定に **display='on'** パラメーターが含まれていることを確認します。以下に例を示します。

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci' display='on'>
 <source>
   <address uuid='ba26a3e2-8e1e-4f39-9de7-b26bd210268a'/>
 </source>
</hostdev>
```

2. 必要に応じて、仮想マシンのビデオモデルタイプを **none** に設定します。以下に例を示します。

```
<video>
 <model type='none'/>
</video>
```

3. 仮想マシンのグラフィックタイプの XML 設定が **spice** または **vnc** であることを確認します。

   スパイスの例:

```
<graphics type='spice' autoport='yes'>
 <listen type='address'/>
 <image compression='off'/>
</graphics>
```

   vncの例:

```
<graphics type='vnc' port='-1' autoport='yes'>
 <listen type='address'/>
</graphics>
```

4. 仮想マシンを起動します。

5. 上の手順で設定したグラフィックプロトコルに適したクライアントを使用して、仮想マシンに接続します。

   - VNC の場合は、VNC ビューアー リモートデスクトップクライアントを使用します。仮想マシンが、エミュレートされた VGA をプライマリービデオデバイスとして、vGPU をセカンダリーとして設定されている場合は、**ctrl+alt+2** キーボードショートカットを使用して vGPU ディスプレイに切り替えます。

- SPICE には、virt-viewer アプリケーションを使用します。

第17章 仮想ネットワーク

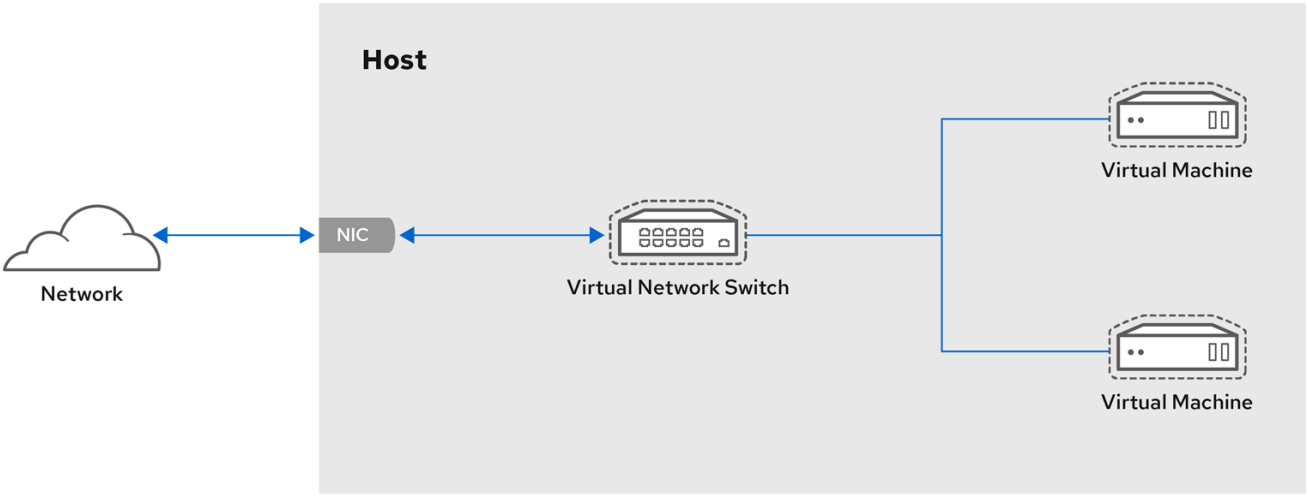
本章では、libvirt で仮想ネットワークを作成、起動、停止、削除、および変更するために必要な概念を説明します。

詳細は、libvirt のリファレンスの章を参照してください。

17.1. 仮想ネットワークスイッチ

Libvirt 仮想ネットワークは、**仮想ネットワークスイッチ** という概念を使用します。仮想ネットワークスイッチは、ホストの物理マシンサーバーで動作し、仮想マシン (ゲスト) が接続するソフトウェア構造です。ゲストのネットワークトラフィックは、このスイッチを介して転送されます。

図17.1 2 つのゲストを持つ仮想ネットワークスイッチ



Linux ホストの物理マシンサーバーは、ネットワークインターフェイスとして仮想ネットワークスイッチを表します。libvirtd デーモン (**libvirtd**) を最初にインストールして起動すると、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イスが **virbr0** になります。

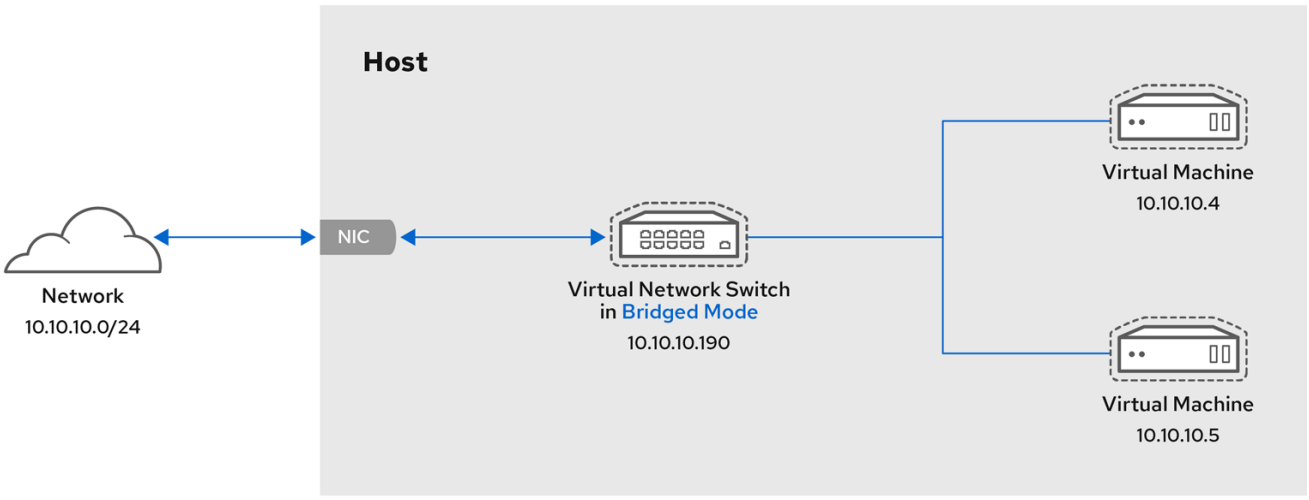
この**virbr0**インターフェイスは、他のインターフェイスと同様に、**ip** コマンドで表示できます。

```
$ ip addr show virbr0
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 1b:c4:94:cf:fd:17 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
```

17.2. ブリッジモード

ブリッジモード を使用する場合は、ゲスト仮想マシンがすべて、ホストの物理マシンと同じサブネット内に表示されます。同じ物理ネットワーク上のその他のすべての物理マシンは、仮想マシンを認識しており、仮想マシンにアクセスできます。ブリッジングは、OSI ネットワークモデルのレイヤー 2 で動作します。

図17.2 ブリッジモードの仮想ネットワークスイッチ



ハイパーバイザーで複数の物理インターフェイスを使用する場合は、*bond* で複数のインターフェイスを結合します。ボンディングがブリッジに追加され、ゲスト仮想マシンもブリッジに追加されます。ただし、ボンディングドライバーには動作モードが複数あり、このモードのごく一部は、仮想ゲストマシンが使用されているブリッジで機能します。

警告

ブリッジモードを使用する場合、仮想マシンで使用するボンディングモードは、モード 1、モード 2、およびモード 4 のみになります。モード 0、3、5、または 6 を使用すると、接続が失敗する可能性が高くなります。また、アドレス解決プロトコル (ARP) の監視が機能しないため、MII (Media-Independent Interface) 監視を使用してボンディングモードを監視する必要があります。

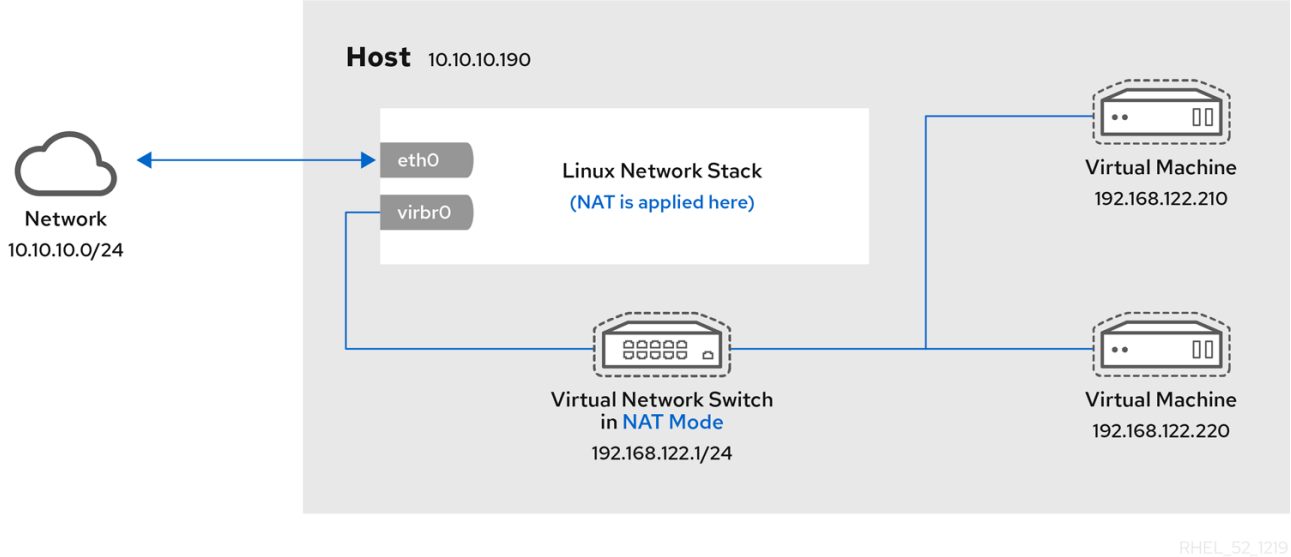
ボンディングモードの詳細は、関連するナレッジベースの記事、または Red Hat Enterprise Linux 7 Networking Guide を参照してください。

ブリッジネットワークモードの設定に使用される bridge_opts パラメーターの詳細な説明は、Red Hat Virtualization Administration Guide を参照してください。

17.3. ネットワークアドレス変換

デフォルトでは、仮想ネットワークスイッチは NAT モードで動作します。Source-NAT (SNAT) または Destination-NAT (DNAT) の代わりに IP マスカレードを使用します。IP マスカレードを使用すると、接続したゲストが、ホストの物理マシンの IP アドレスを使用して外部ネットワークと通信できます。次の図に示すように、デフォルトでは、仮想ネットワークスイッチが NAT モードで動作している場合、ホスト物理マシンの外部に配置されたコンピューターは内部のゲストと通信できません。

図17.3 2 つのゲストで NAT を使用する仮想ネットワークスイッチ



警告

仮想ネットワークスイッチは、iptables ルールで設定された NAT を使用します。スイッチの実行中にこのルールを編集することは推奨されていません。誤ったルールがあると、スイッチが通信できなくなる可能性があるためです。

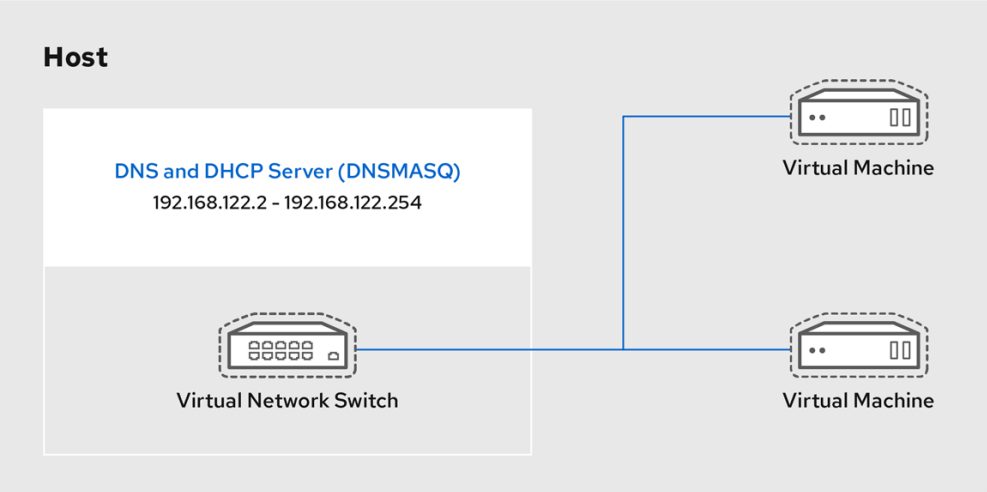
スイッチが実行していない場合は、正引きモードの NAT のパブリック IP 範囲を設定して、ポートマスカレードの範囲を作成するには、以下のコマンドを実行します。

```
# iptables -j SNAT --to-source [start]-[end]
```

17.4. DNS および DHCP

IP 情報は DHCP を介してゲストに割り当てることができます。このため、仮想ネットワークスイッチには、アドレスのプールを割り当てることができます。Libvirt は、これに **dnsmasq** プログラムを使用します。dnsmasq のインスタンスは、それを必要とする各仮想ネットワークの libvirt により自動的に設定され、起動します。

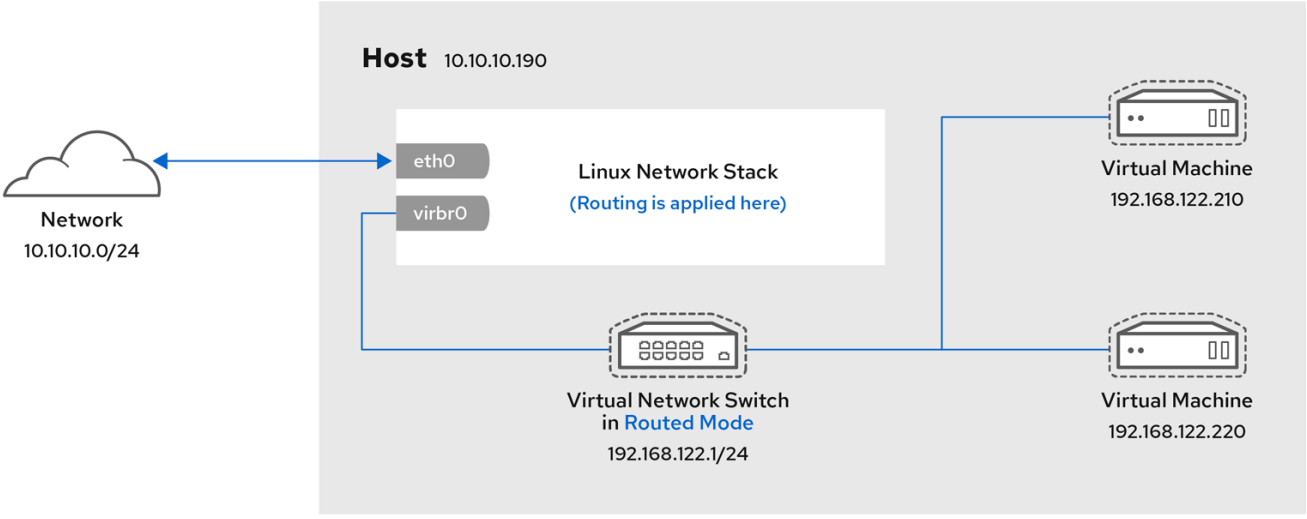
図17.4 dnsmasq を実行している仮想ネットワークスイッチ



17.5. ルーティングモード

ルーティングモード を使用する場合は、仮想スイッチを、ホストマシンに接続された物理 LAN に接続し、NAT を使用せずにトラフィックをやり取りします。仮想スイッチは、すべてのトラフィックを調べ、ネットワークパケットに含まれる情報を使用して、ルーティングの決定を行うことができます。このモードを使用すると、仮想マシンはすべて自身のサブネットにあり、仮想スイッチを介してルーティングされます。物理ネットワーク上の他のホスト物理マシンが手動の物理ルーター設定なしで仮想マシンを認識しておらず、仮想マシンにアクセスできないため、この状況は必ずしも理想的ではありません。ルーティングモードは、OSI ネットワークモデルのレイヤー 3 で動作します。

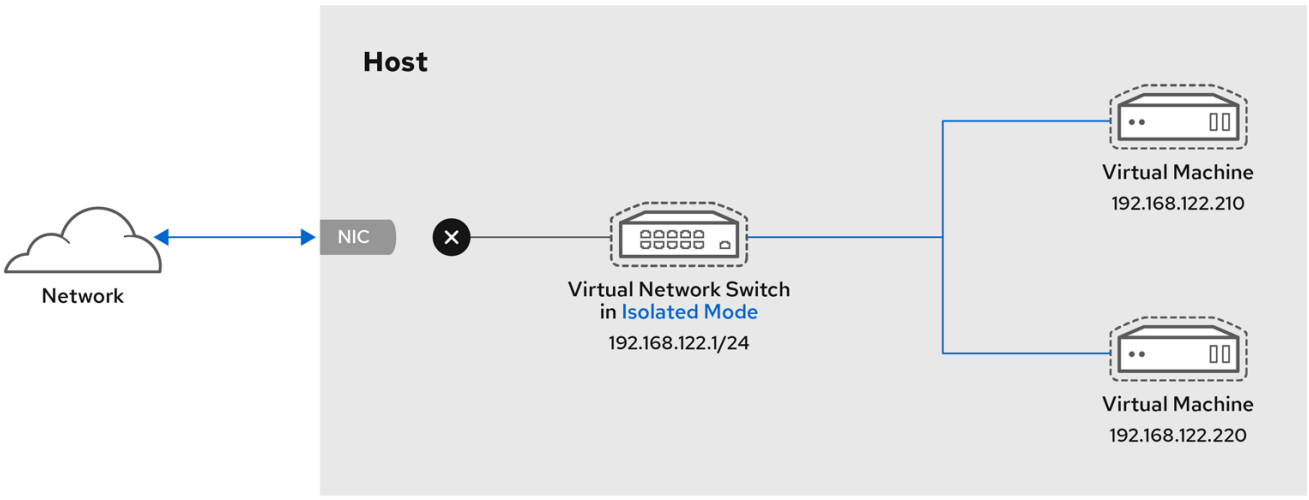
図17.5 ルーティングモードの仮想ネットワークスイッチ



17.6. 分離モード

分離モード を使用する場合は、仮想スイッチに接続したゲストが、ホストの物理マシンと相互通信できます。ただし、ゲストのトラフィックはホストの物理マシンの外部を通過しないため、ホストの物理マシンの外部からのトラフィックを受信できません。DHCP などの基本的な機能には、このモードの dnsmasq を使用する必要があります。ただし、このネットワークが物理ネットワークから分離されていても、DNS 名は解決されません。そのため、DNS 名が解決しても、ICMP echo 要求 (ping) コマンドが失敗すると、問題が発生する可能性があります。

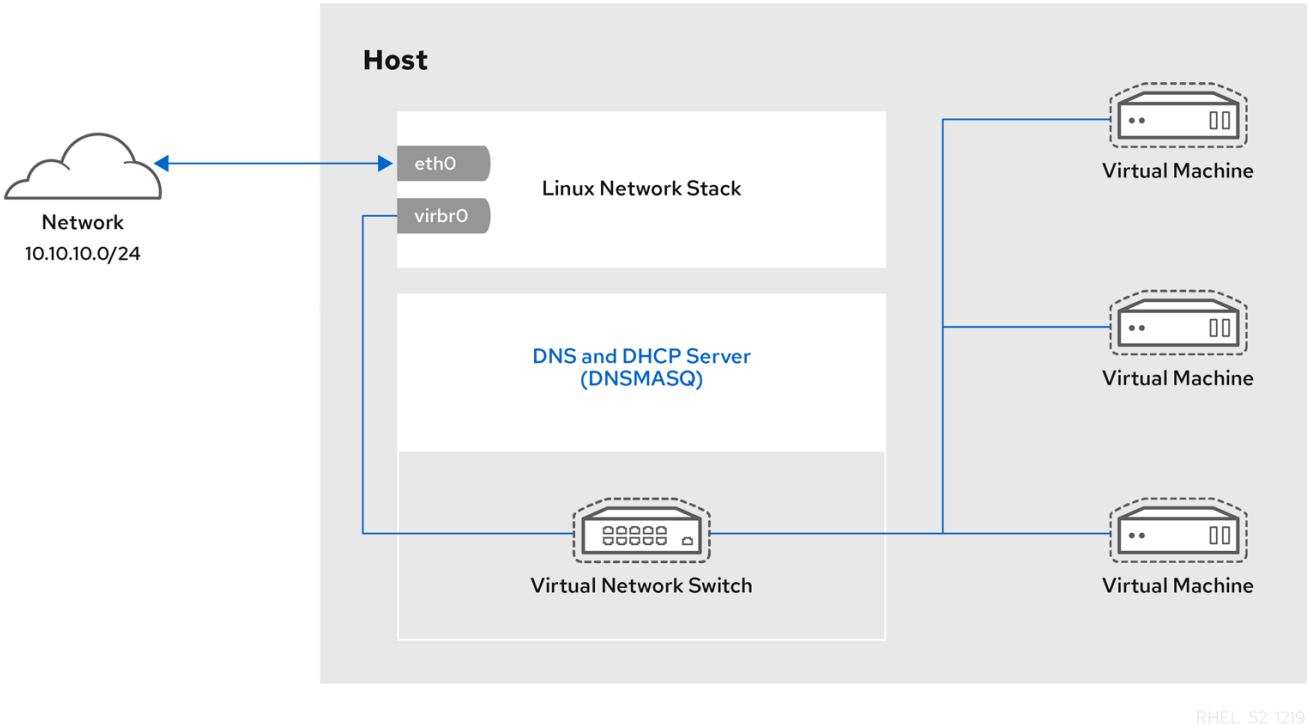
図17.6 分離モードの仮想ネットワークスイッチ



17.7. デフォルト設定

libvirtd デーモン (**libvirtd**) を最初にインストールすると、NAT モードの最初の仮想ネットワークスイッチ設定が含まれます。この設定は、インストールしたゲストが、ホストの物理マシンを介して外部ネットワークと通信できるように使用されます。以下のイメージは、**libvirtd** におけるこのデフォルトの設定を示しています。

図17.7 デフォルトの libvirt ネットワーク設定



注記

仮想ネットワークは、特定の物理インターフェイスに制限できます。これは、複数のインタフェースを持つ物理システムで役立つ場合があります (たとえば、**eth0**、**eth1**、および **eth2**)。これは、ルーティングモードおよび NAT モードでのみ役に立ち、新しい仮想ネットワークを作成する場合は、**dev=<interface>** オプション、または **virt-manager** で定義できます。

17.8. 一般的なシナリオの例

本セクションでは、さまざまな仮想ネットワークモードと、いくつかのシナリオ例を示します。

17.8.1. ブリッジモード

ブリッジモードは、OSI モデルのレイヤー 2 で動作します。これを使用すると、ゲスト仮想マシンはすべて、ホストの物理マシンと同じサブネットに表示されます。ブリッジモードにおける最も一般的なユースケースには、たとえば以下のようなものがあります。

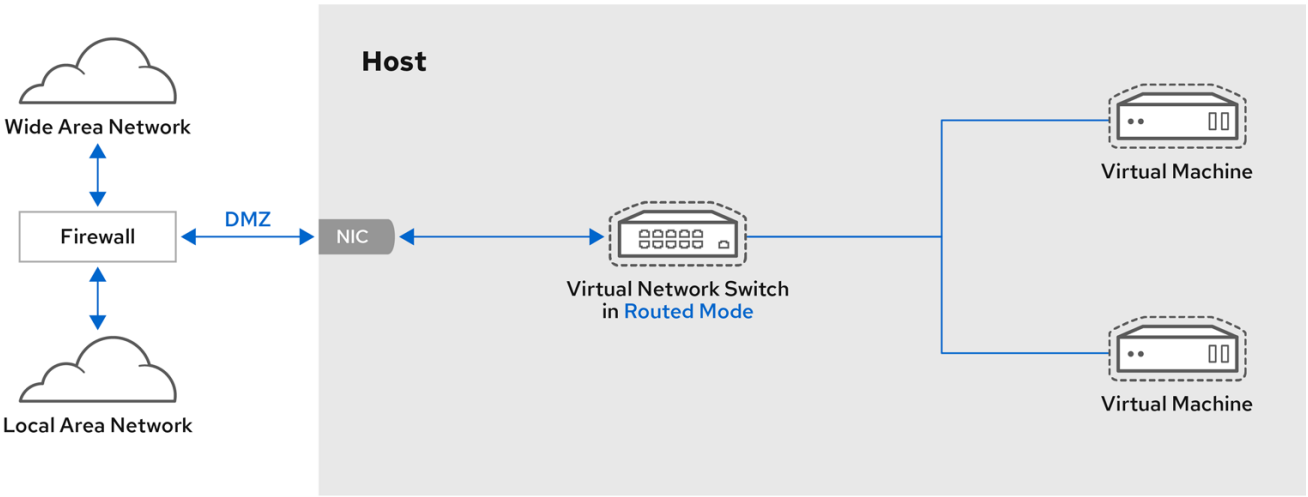
- 既存ネットワークにゲスト仮想マシンをホスト物理マシンとともにデプロイすると、仮想マシンと物理マシンの違いがエンドユーザーに透過的になります。
- 既存の物理ネットワーク設定を変更せずに仮想マシンをデプロイします。
- 既存の物理ネットワークに簡単にアクセスできる必要があるゲスト仮想マシンをデプロイします。既存のブロードキャストドメイン (DHCP など) のサービスにアクセスする必要がある物理ネットワークにゲスト仮想マシンを配置します。
- VLAN が使用されている既存のネットワークにゲスト仮想マシンを接続する。

17.8.2. ルーティングモード

DMZ

セキュリティー上の理由から、1 つ以上のノードが制御されたサブネットワークに配置されているネットワークを検討してください。このような特別なサブネットワークのデプロイメントは一般的な慣例であり、サブネットワークは DMZ と呼ばれています。このレイアウトの詳細は、以下の図を参照してください。

図17.8 サンプルの DMZ 設定

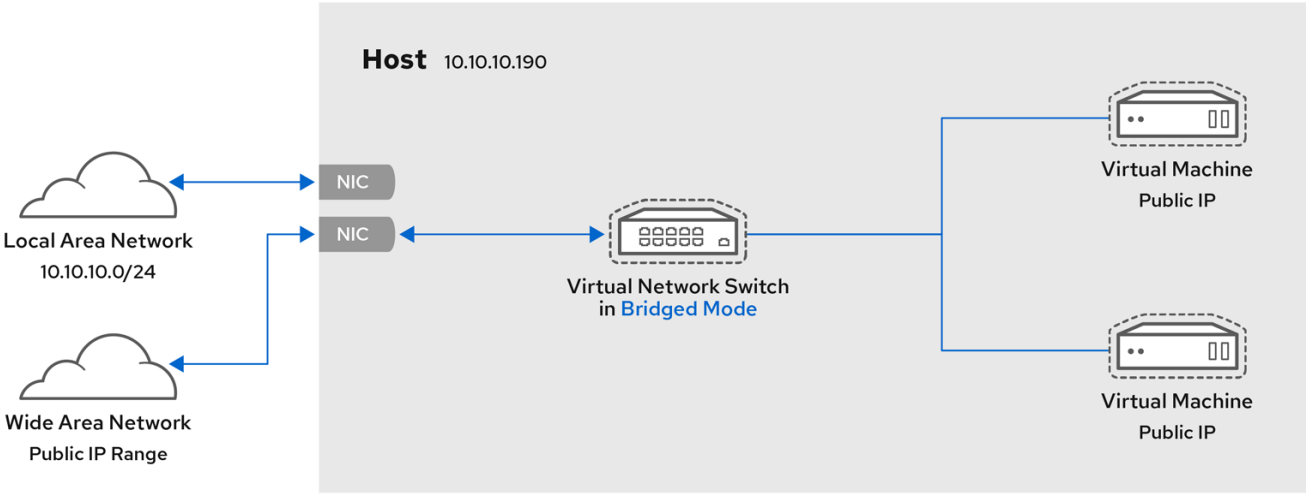


通常、DMZ のホスト物理マシンは、WAN (外部) ホスト物理マシンおよび LAN (内部) ホスト物理マシンにサービスを提供します。これは複数の場所からアクセスできる必要があり、この場所はセキュリティーおよび信頼レベルに基づいて異なる方法で制御および操作されるため、ルーティングモードはこの環境に最適な設定になります。

仮想サーバーのホスト

それぞれが 2 つの物理ネットワーク接続のある複数のホスト物理マシンを持つ仮想サーバーホスティング会社について考えてみます。管理とアカウンティングにはいずれかのインターフェイスが使用されており、もう 1 つは仮想マシンによる接続用です。各ゲストには独自のパブリック IP アドレスがありますが、ホストの物理マシンはプライベート IP アドレスを使用するため、ゲストの管理は内部管理者しか行えません。このシナリオを理解するには、以下の図を参照してください。

図17.9 仮想サーバーホスティングのサンプル設定



17.8.3. NAT モード

NAT (Network Address Translation) モードがデフォルトモードです。直接ネットワークの可視性が必要ない場合にテストに使用できます。

17.8.4. 分離モード

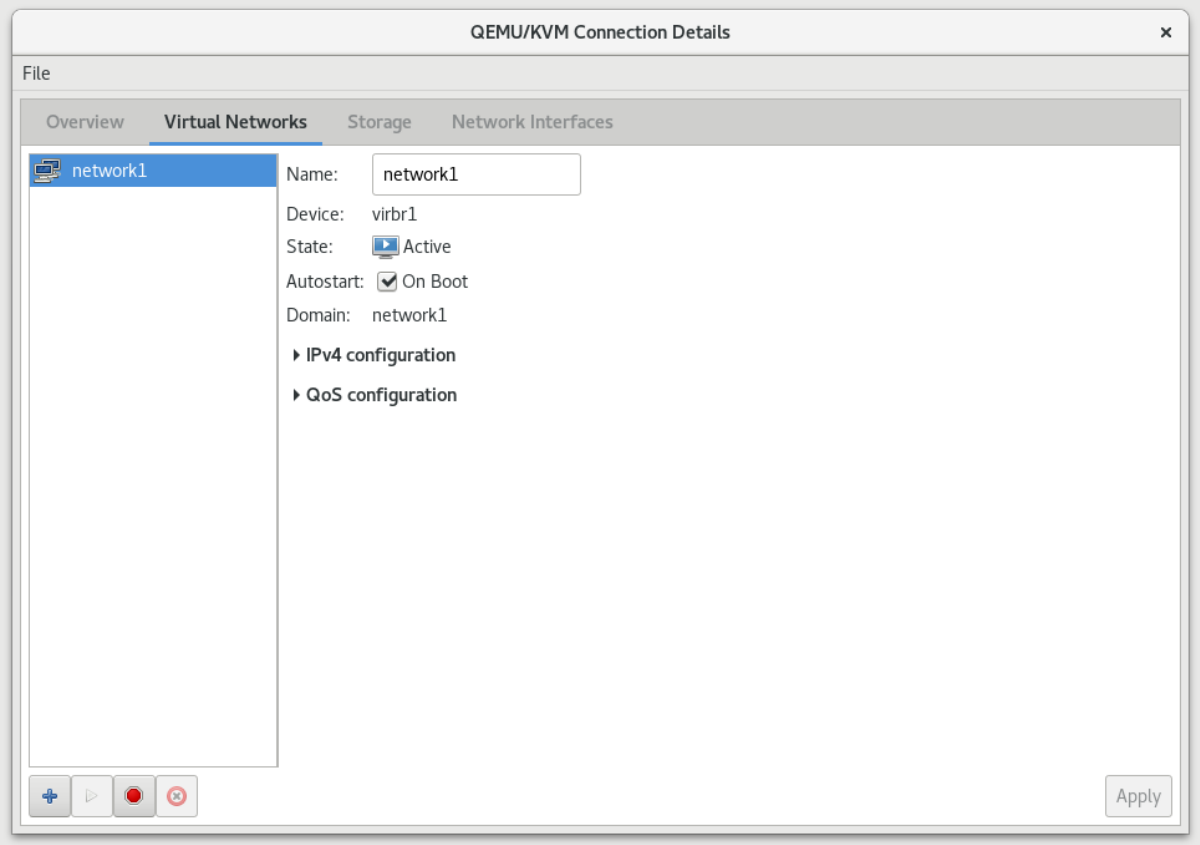
分離モードでは、仮想マシンが相互にのみ通信できます。物理ネットワークと対話することはできません。

17.9. 仮想ネットワークの管理

システムに仮想ネットワークを設定するには、次のコマンドを実行します。

1. **Edit** メニューから、**Connection Details** を選択します。
2. これにより、Connection Details メニューーが開きます。**Virtual Networks** タブをクリックします。

図17.10 仮想ネットワークの設定



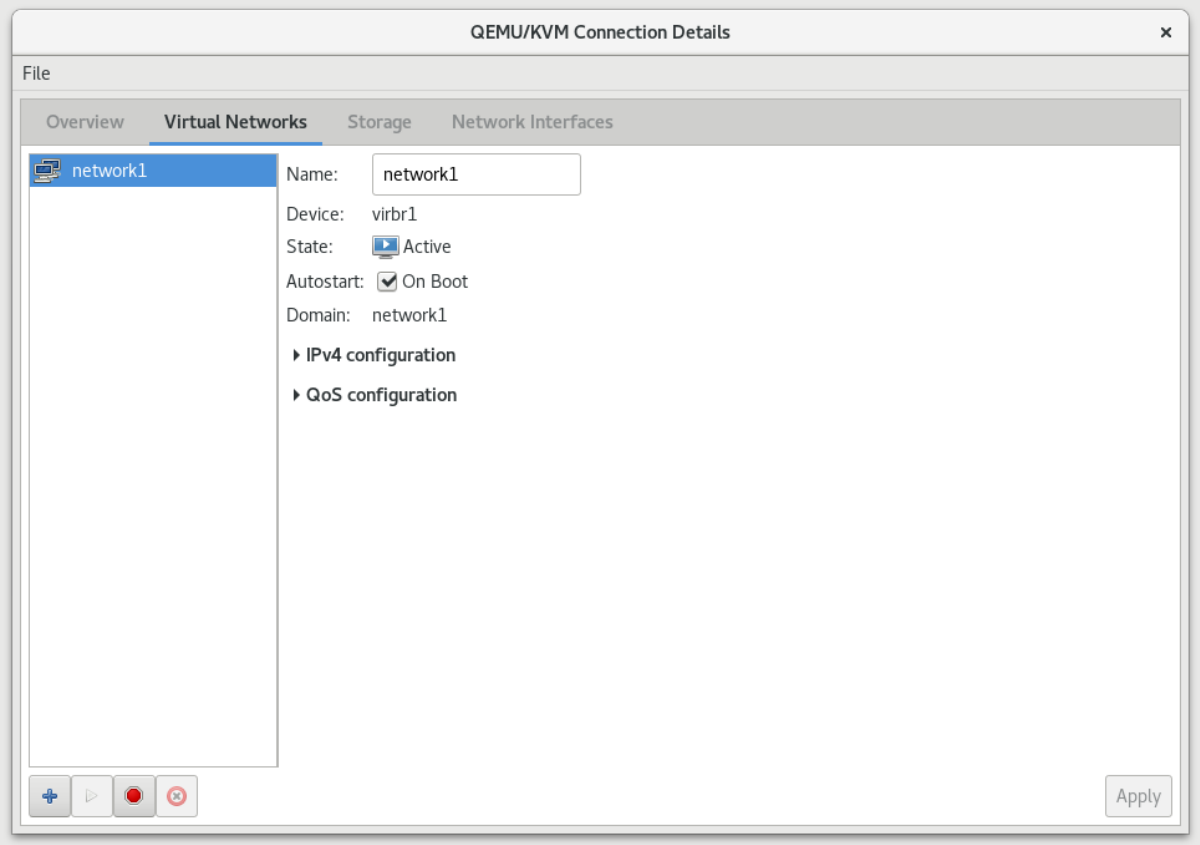
3. 利用可能な仮想ネットワークがすべて、メニューの左側に表示されます。このボックスから仮想ネットワークを選択し、必要に応じて編集することで、仮想ネットワークの設定を編集できます。

17.10. 仮想ネットワークの作成

Virtual Machine Manager (virt-manager) を使用してシステムに仮想ネットワークを作成するには、次のコマンドを実行します。

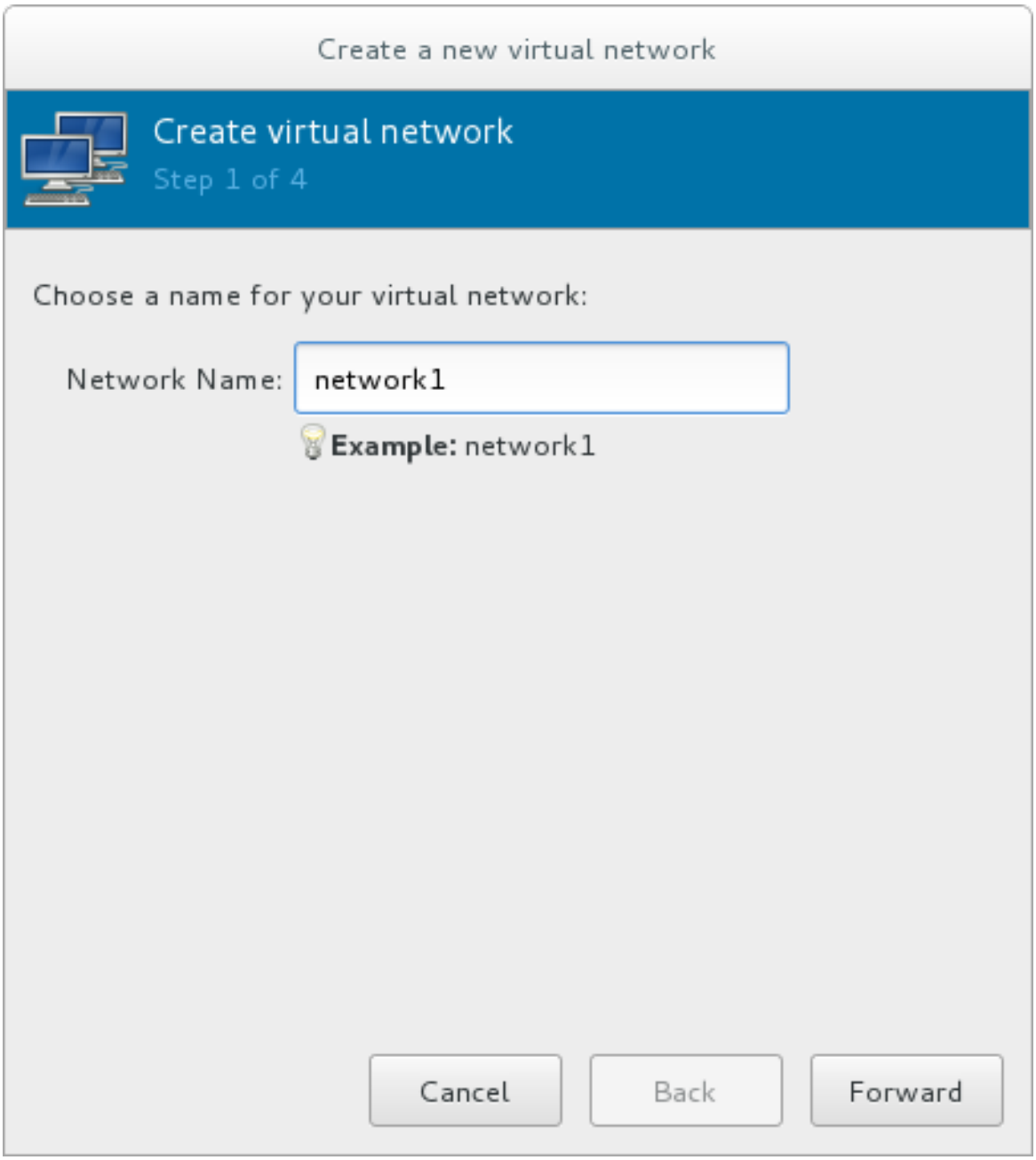
1. **Connection Details** メニューから **Virtual Networks** タブを開きます。プラス記号 (+) のアイコンで識別される **Add Network** ボタンをクリックします。詳細は、「仮想ネットワークの管理」を参照してください。

図17.11 仮想ネットワークの設定



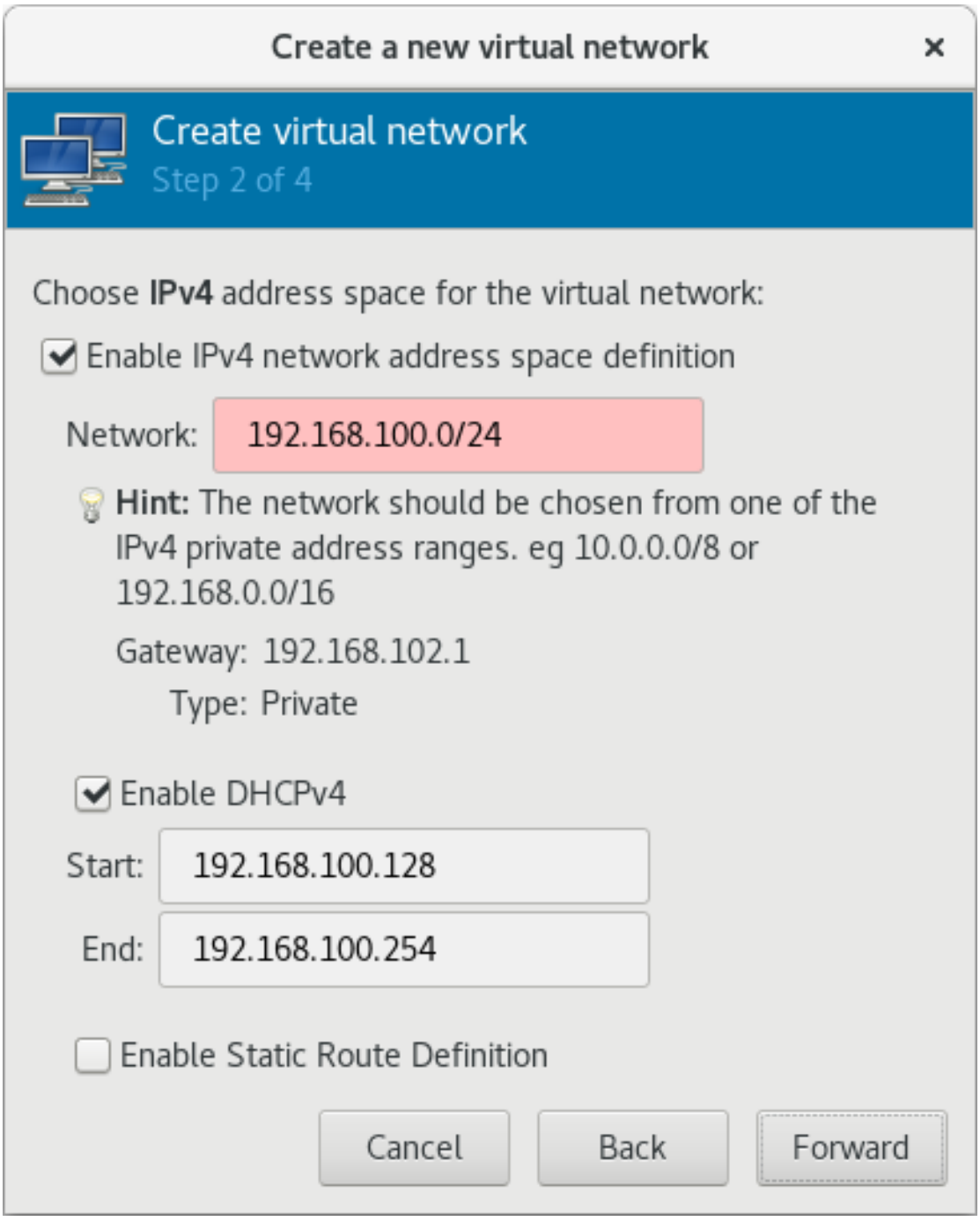
これにより、**Create a new virtual network**ウィンドウが開きます。**Forward** をクリックして続けます。

図17.12 新しい仮想ネットワークの命名



2. 仮想ネットワークに適切な名前を入力し、**Forward** をクリックします。

図17.13 IPv4 アドレス空間の選択



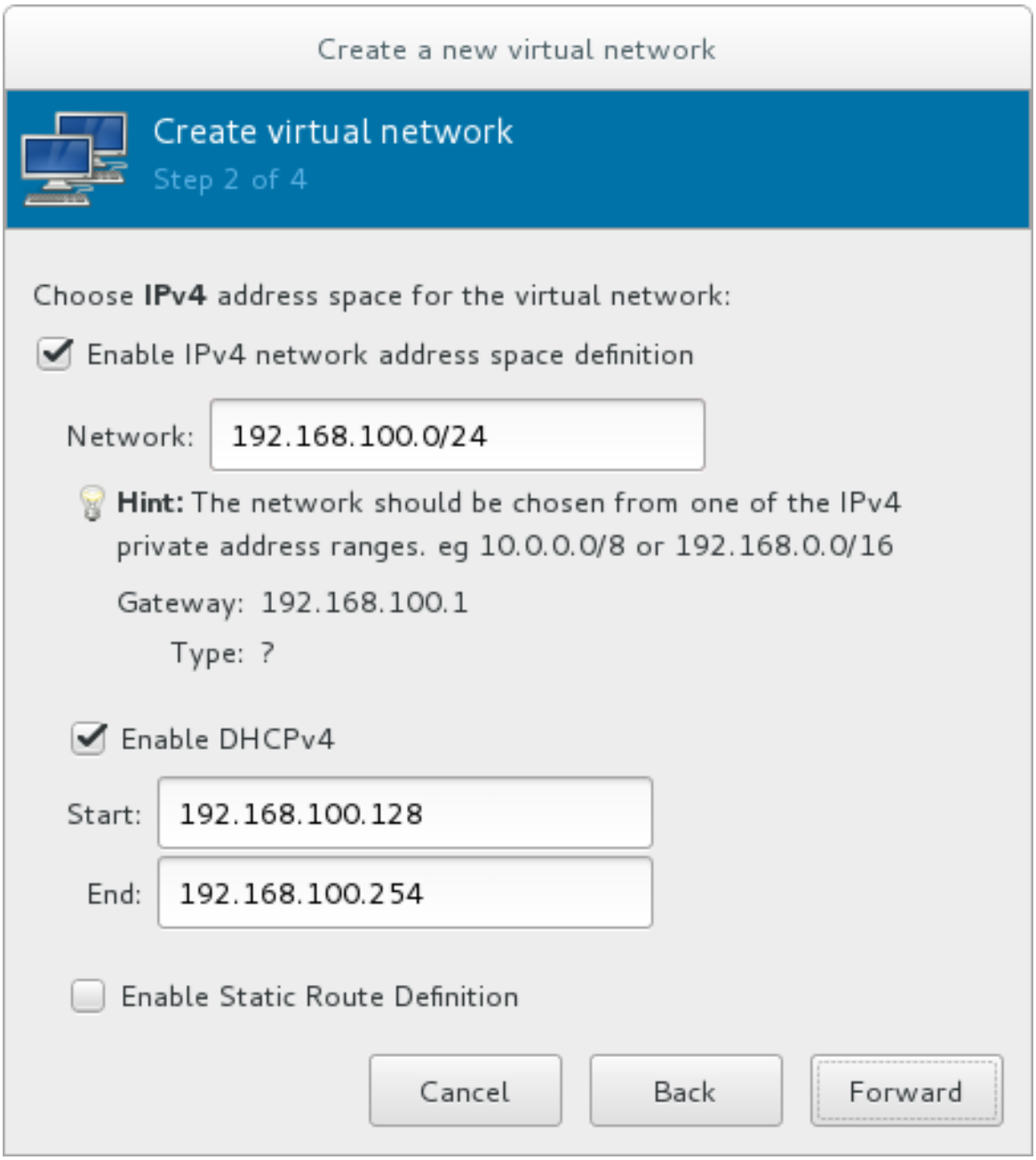
3. **Enable IPv4 network address space definition** チェックボックスをオンにします。

   Network フィールドに、仮想ネットワークの IPv4 アドレス空間を入力します。

   Enable DHCPv4 チェックボックスをオンにします。

   IP アドレスの **Start** 範囲および **End** 範囲を指定して、仮想ネットワークに DHCP 範囲を定義します。

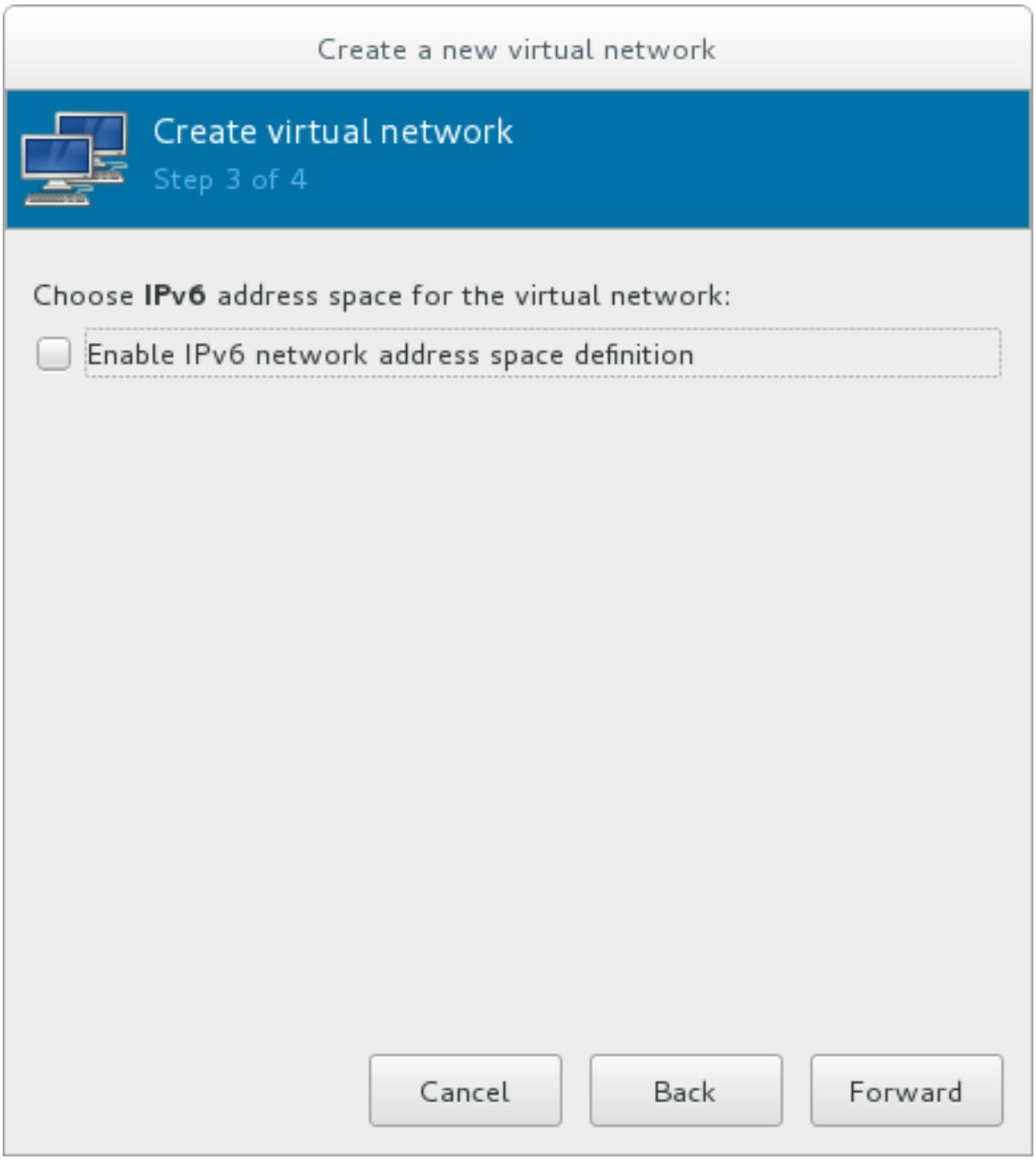
図17.14 IPv4 アドレス空間の選択



Forward をクリックして続けます。

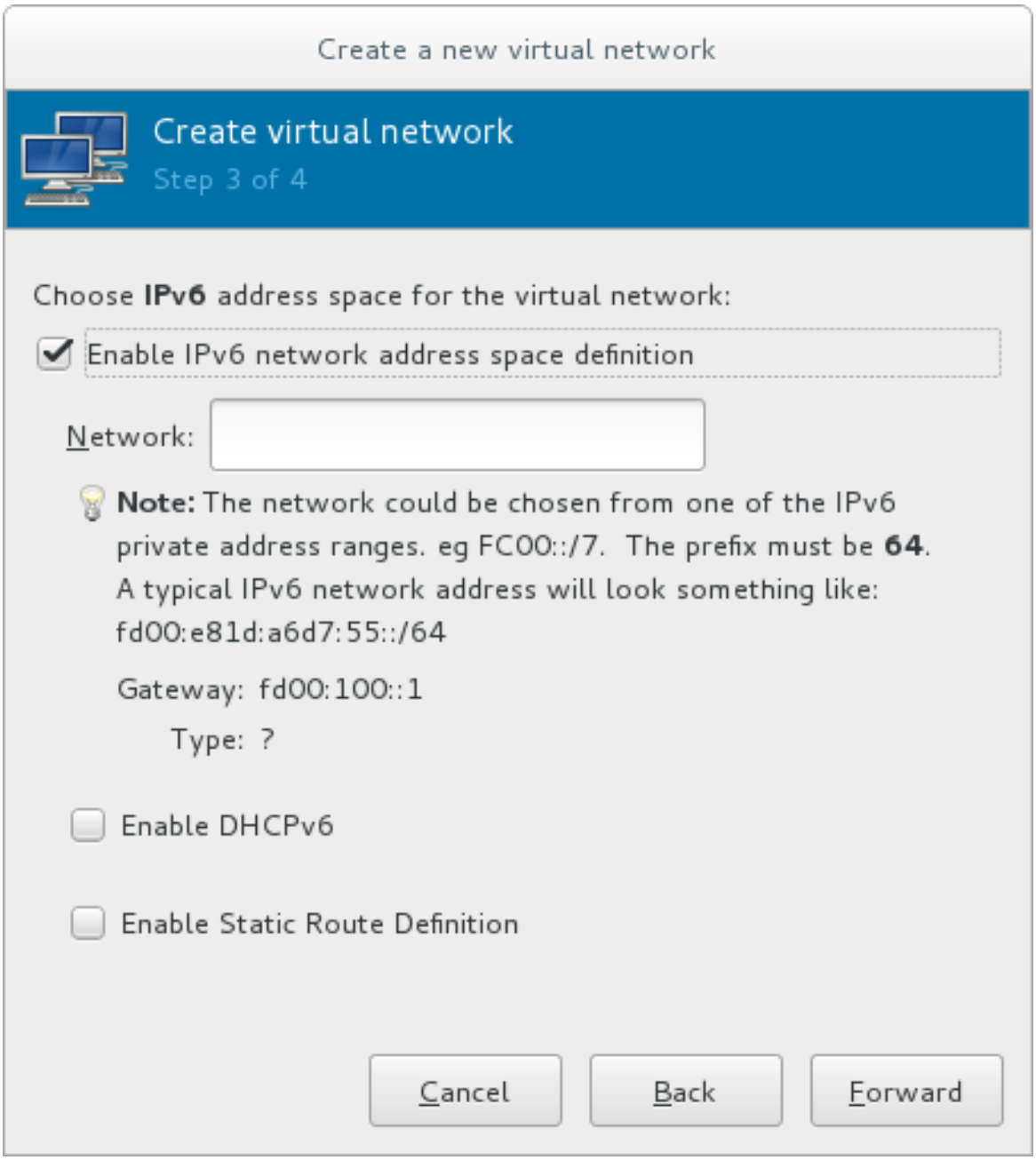
4. IPv6 を有効にする場合は、**Enable IPv6 network address space definition** チェックボックスをオンにします。

図17.15 IPv6 の有効化



追加のフィールドが、Create a new virtual network ウィンドウに表示されます。

図17.16 IPv6 の設定

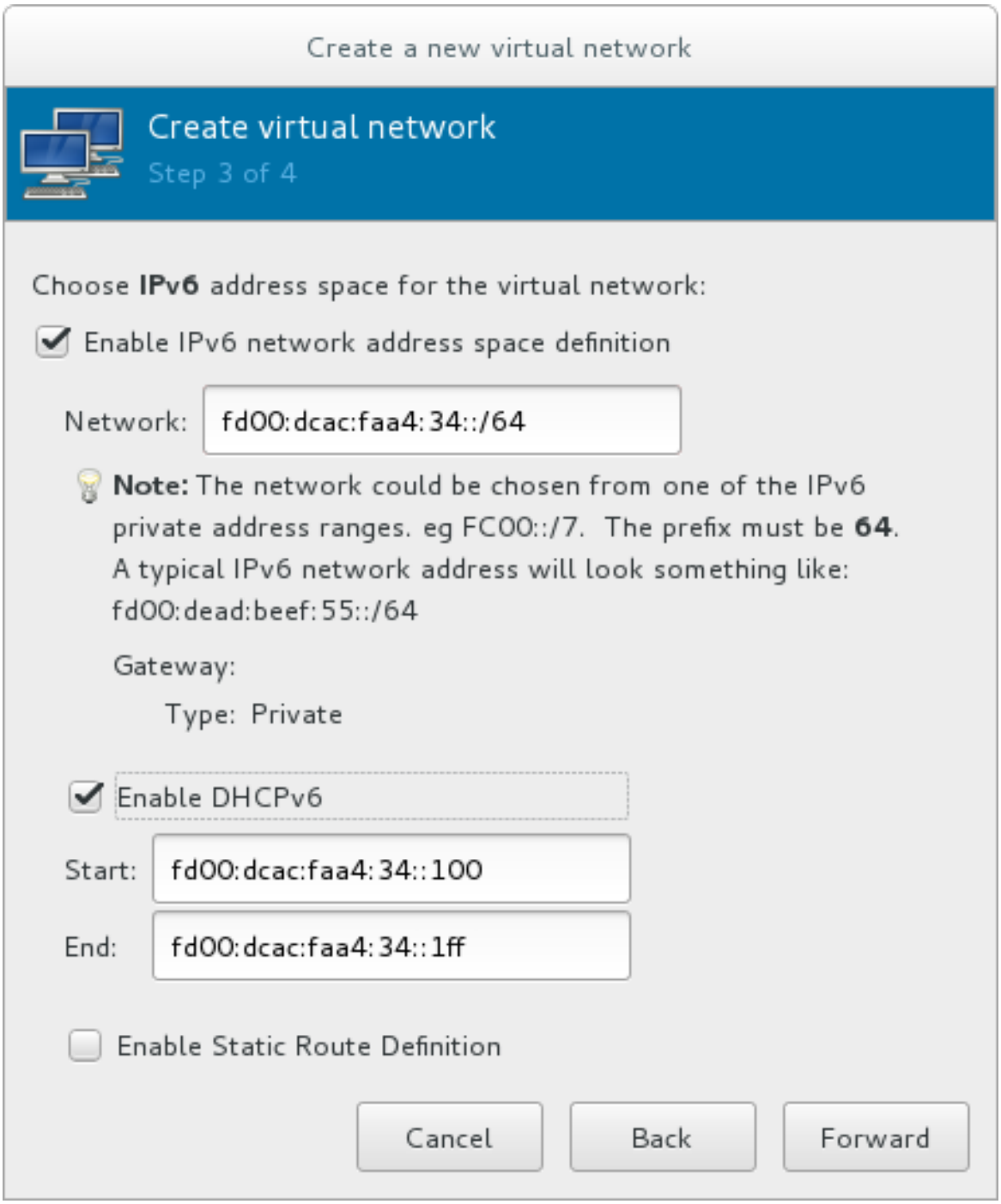


Network フィールドに IPv6 アドレスを入力します。

5. DHCPv6 を有効にする場合は、**Enable DHCPv6** チェックボックスをオンにします。

   追加のフィールドが、Create a new virtual network ウィンドウに表示されます。

図17.17 DHCPv6 の設定

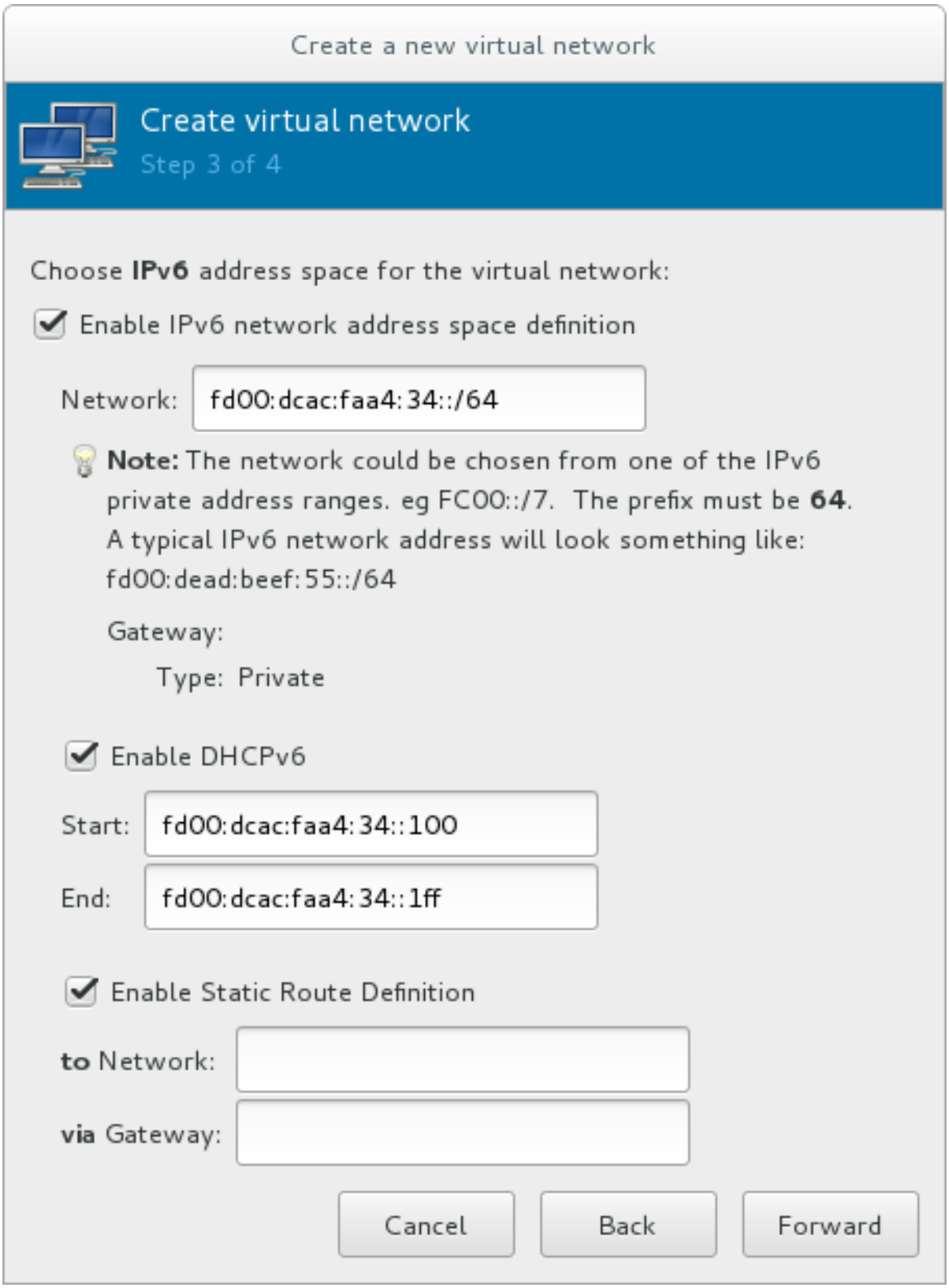


(必要に応じて) DHCPv6 範囲の開始と終了を編集します。

6. 静的ルート定義を有効にする場合は、**Enable Static Route Definition** チェックボックスをオンにします。

   追加のフィールドが、Create a new virtual network ウィンドウに表示されます。

図17.18 静的ルートの定義

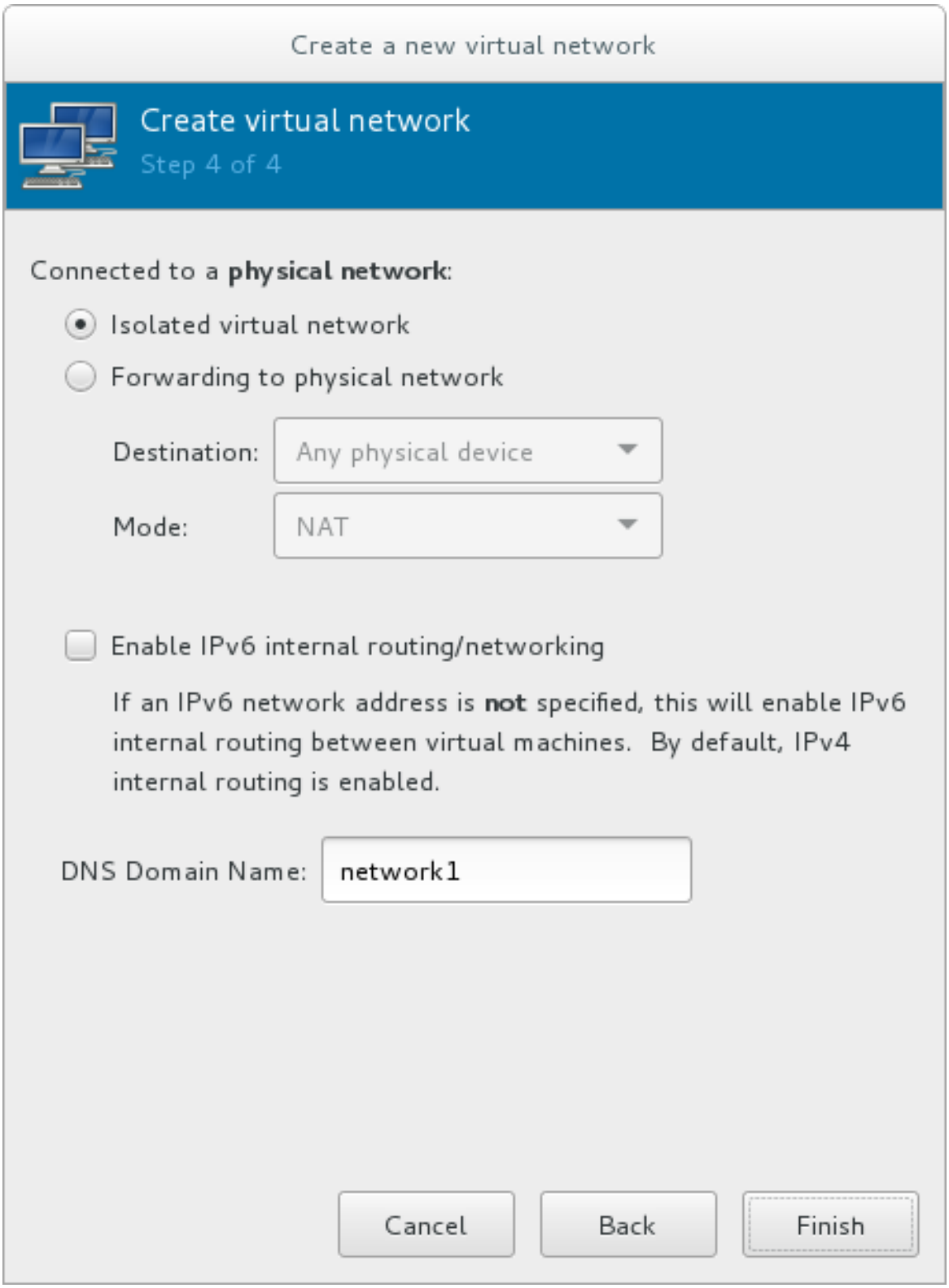


ネットワークアドレスと、ネットワークへのルートに使用されるゲートウェイを、適切なフィールドに入力します。

Forward をクリックします。

7. 仮想ネットワークを物理ネットワークに接続する方法を選択します。

図17.19 物理ネットワークへの接続



仮想ネットワークを分離する場合は、**Isolated virtual network** のラジオボタンを選択します。

仮想ネットワークを物理ネットワークに接続する場合は、**Forwarding to physical network** を選択し、**Destination** を **Any physical device** にするか、特定の物理デバイスにするかを選択します。また、**Mode**を **NAT** にするか、**Routed** にするかを選択します。

仮想ネットワーク内で IPv6 ルーティングを有効にする場合は、**Enable IPv6 internal routing/networking** チェックボックスをオンにします。

仮想ネットワークの DNS ドメイン名を入力します。

Finish をクリックして、仮想ネットワークを作成します。

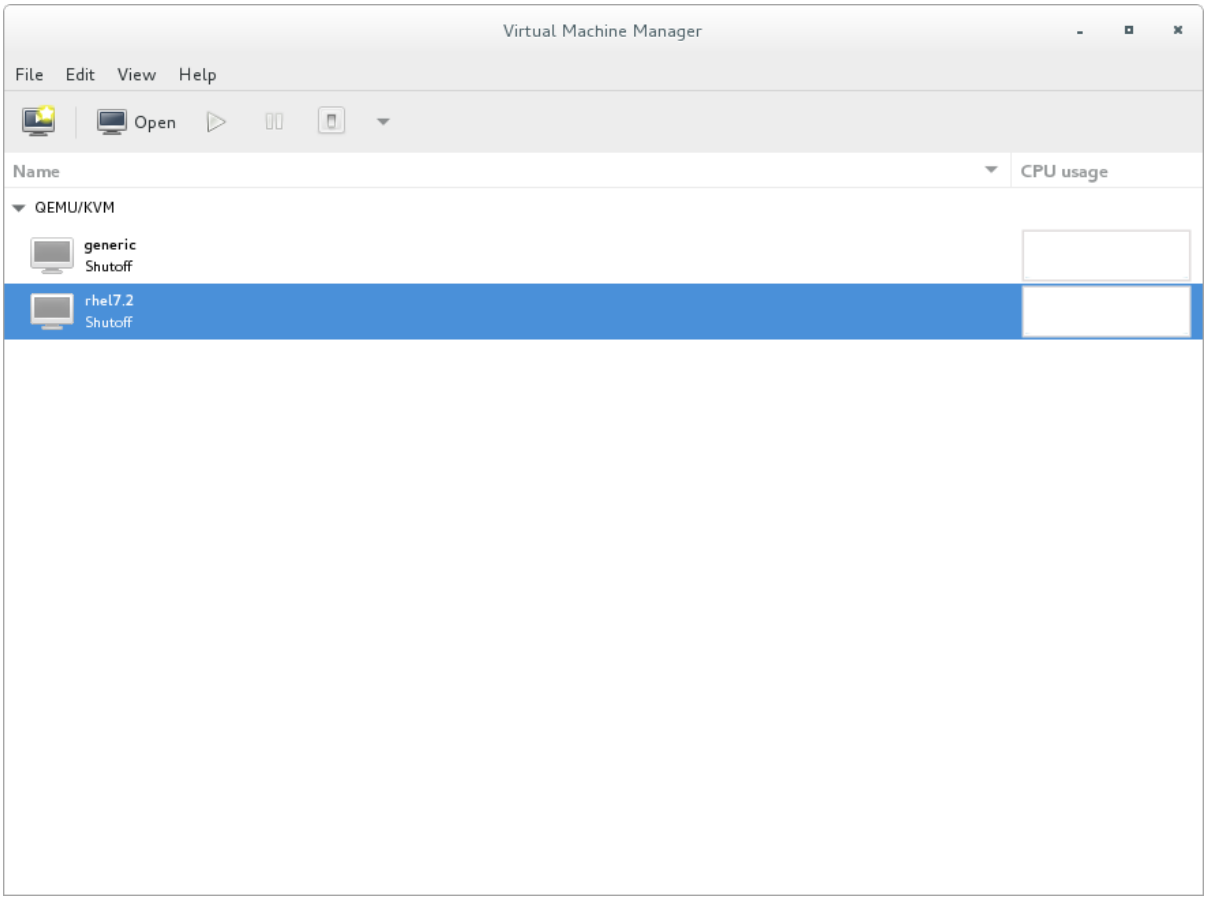
8. 新しい仮想ネットワークが、 **Connection Details** ウィンドウの **Virtual Networks** タブで利用できるようになります。

17.11. 仮想ネットワークのゲストへの割り当て

仮想ネットワークをゲストに接続するには、次のコマンドを実行します。

1. **Virtual Machine Manager** ウィンドウで、ネットワークが割り当てられるゲストを強調表示します。

図17.20 表示する仮想マシンの選択

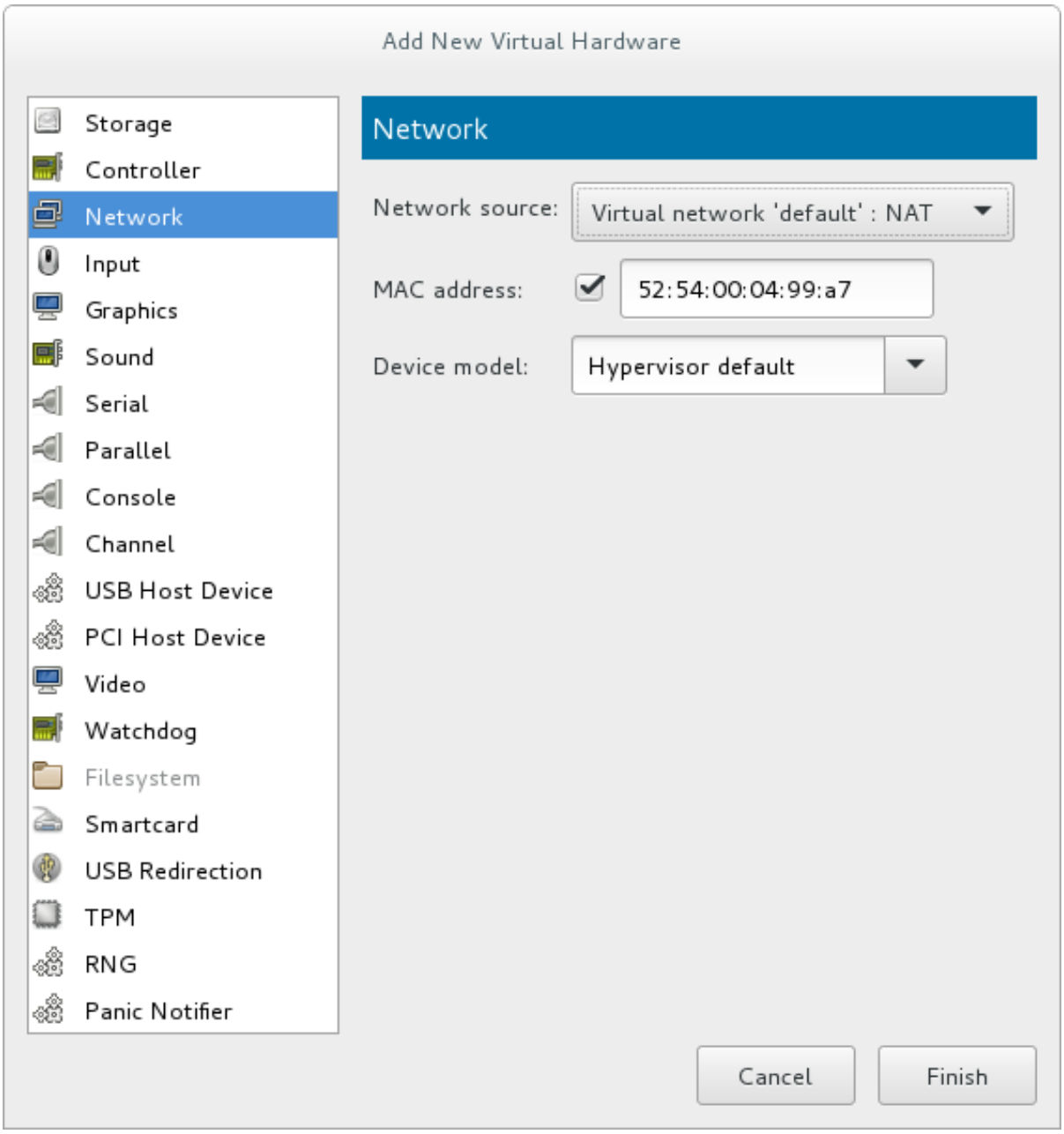


2. 仮想マシンマネージャーの**Edit** メニューから、**Virtual Machine Details** を選択します。

3. Virtual Machine Details ウィンドウの **Add Hardware** ボタンをクリックします。

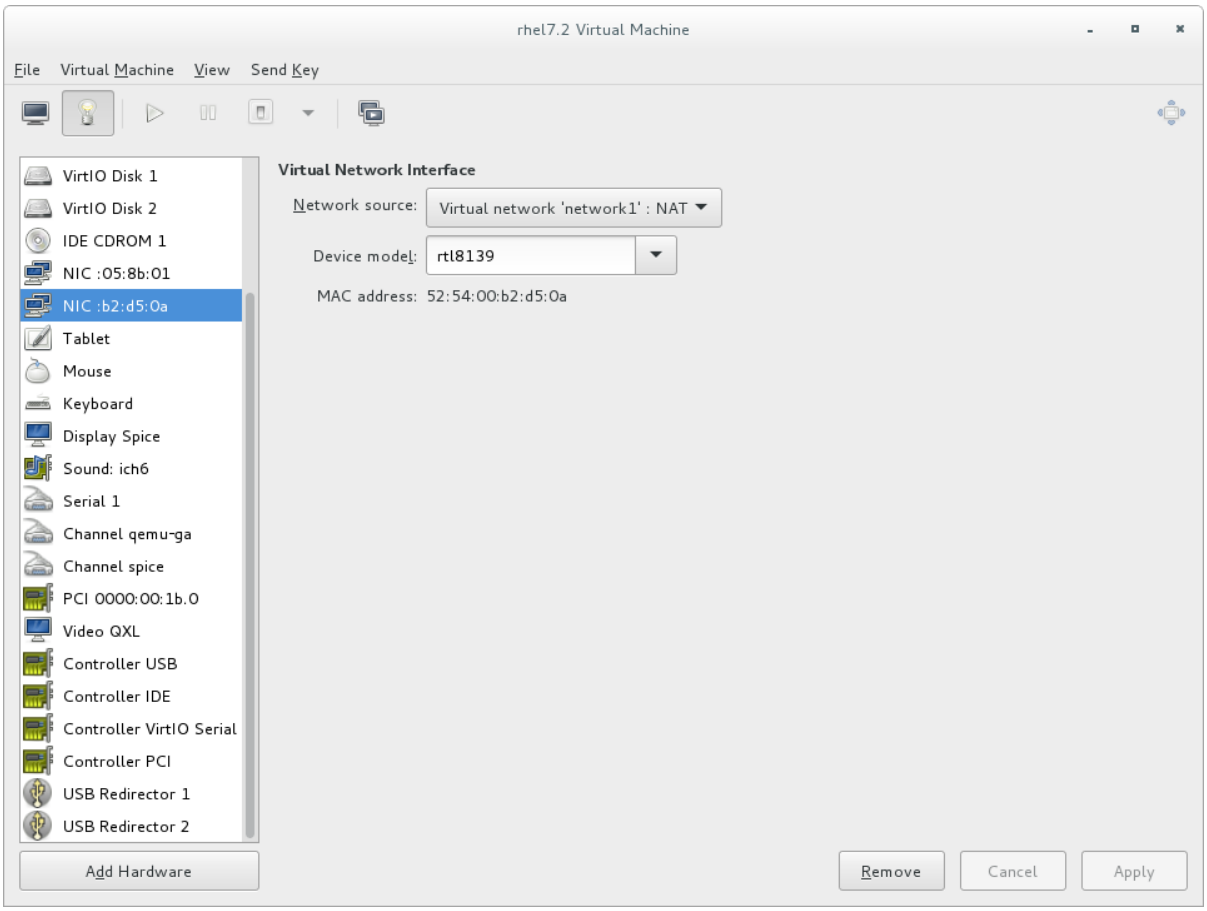
4. **Add new virtual hardware** ウィンドウで、左側のペインから **Network** を選択し、**Network source** メニューからネットワーク名 (この例では **network1**) を選択します。必要に応じて MAC アドレスを変更し、**Device model** を選択します。**Finish** をクリックします。

図17.21 Add new virtual hardware ウィンドウからネットワークを選択します



5. 新しいネットワークが、起動時にゲストに表示される仮想ネットワークインターフェイスとして表示されるようになりました。

図17.22 ゲストハードウェアリストに新しいネットワークが表示されました。



17.12. 物理インターフェイスへの仮想 NIC の直接接続

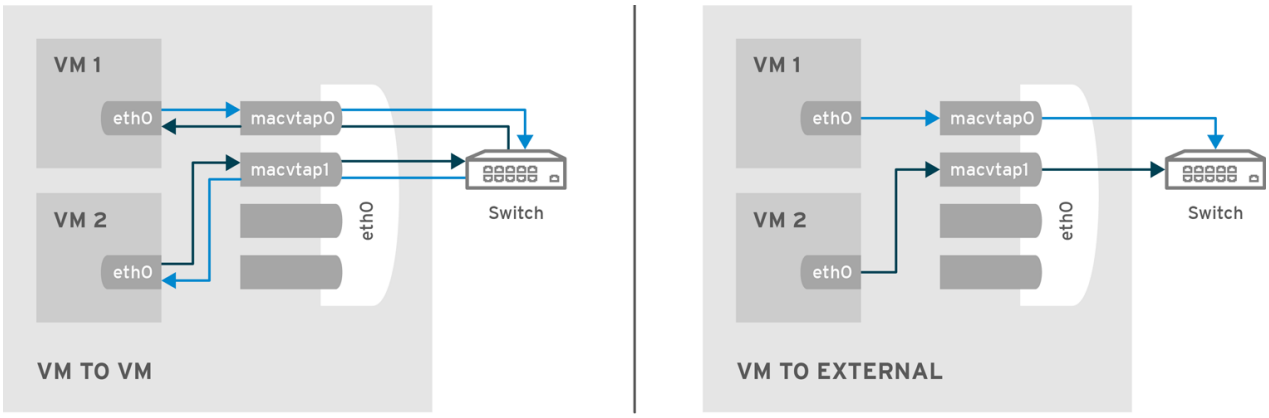
デフォルトの NAT 接続の代わりに、*macvtap* ドライバーを使用して、ゲストの NIC を、ホストマシンの指定された物理インターフェイスに直接接続できます。これを デバイスの割り当て (パススルーとも呼ばれます) と混同しないようにしてください。Macvtap 接続には次のモードがあり、それぞれに異なる利点とユースケースがあります。

物理インターフェイスの配信モード

VEPA

仮想イーサネットポートアグリゲーター (VEPA) モードでは、ゲストからのすべてのパケットが外部スイッチに送信されます。これにより、ユーザーはスイッチを介してゲストトラフィックを強制的に送信できます。VEPA モードが適切に機能するには、外部スイッチも **ヘアピンモード** に対応する必要があります。これにより、宛先がソースゲストと同じホストマシンのゲストであるパケットが、外部スイッチによりホストに返されます。

図17.23 VEPA モード

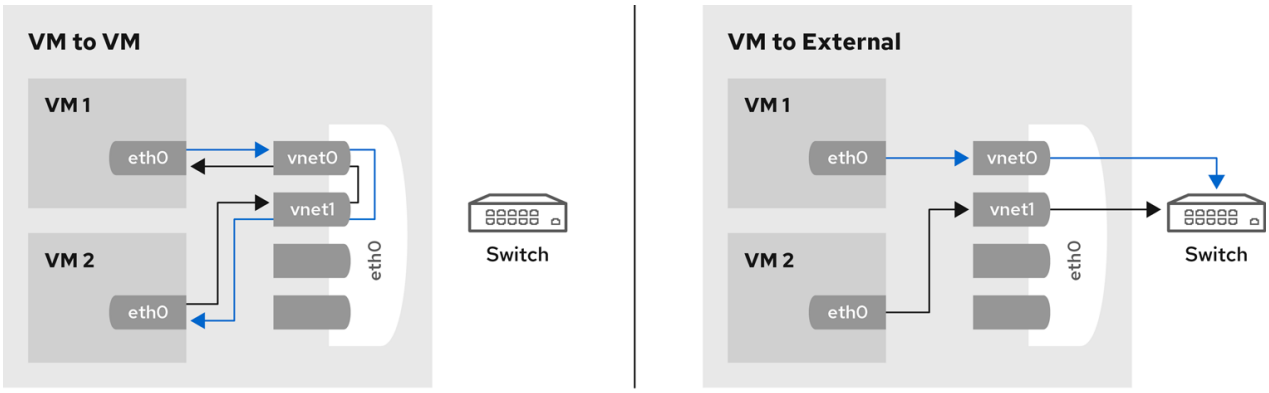


RHEL_437030_0417

bridge

宛先がソースゲストと同じホスト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。直接配信を成功させるには、ソースデバイスとターゲットデバイスの両方がブリッジモードになっている必要があります。いずれかのデバイスが VEPA モードの場合は、ヘアピン対応の外部スイッチが必要です。

図17.24 ブリッジモード

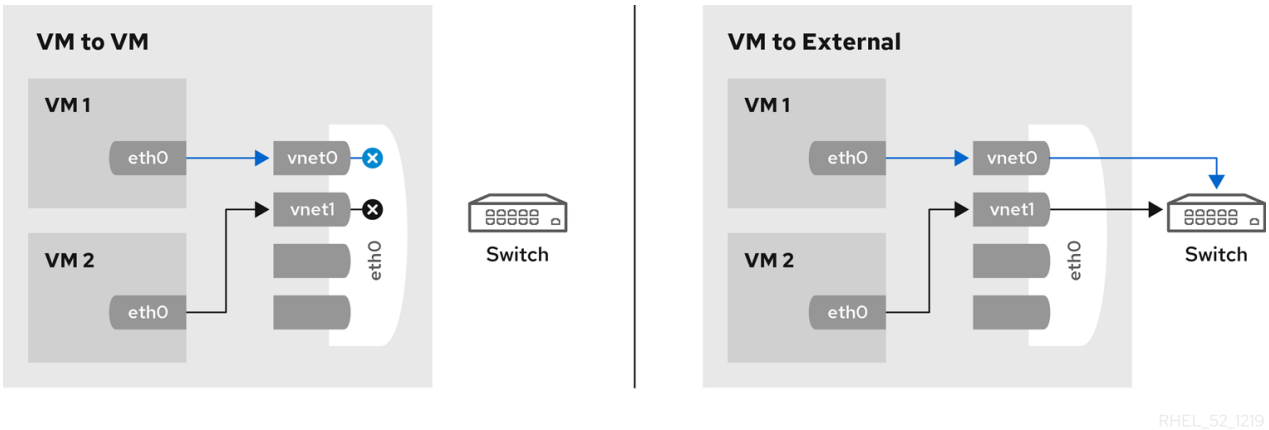


RHEL_52_1219

プライベート

すべてのパケットは外部スイッチに送信されます。また、外部ルーターまたはゲートウェイを介して送信され、これらがホストに送り返す場合に、同じホストマシン上のターゲットゲストにのみ配信されます。プライベートモードを使用すると、1 台のホスト上にある個別の仮想マシン間での通信を阻止することができます。移行元デバイスまたは移行先デバイスのいずれかがプライベートモードの場合は、以下の手順が行われます。

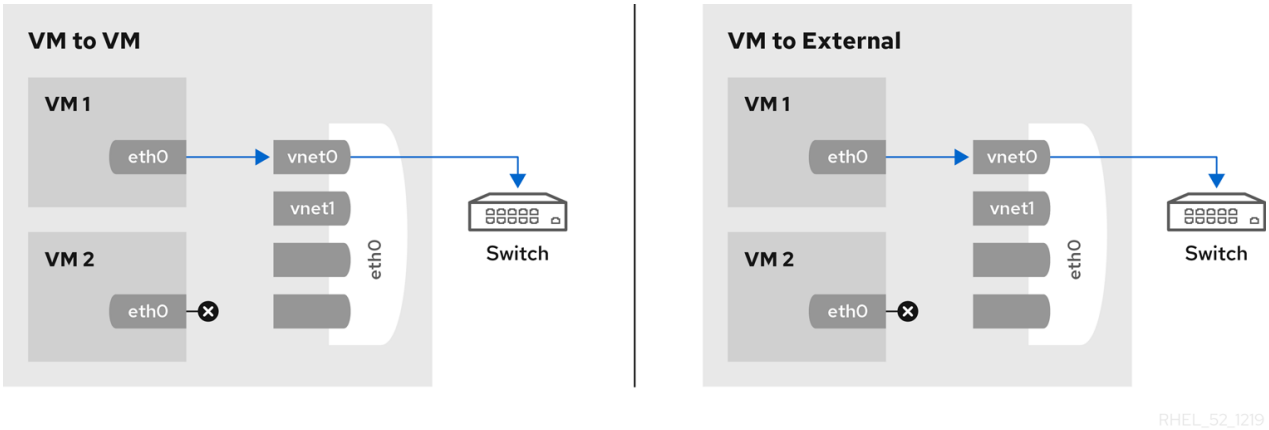
図17.25 プライベートモード



パススルー

この機能は、移行機能を失うことなく、物理インターフェイスデバイスまたはSR-IOV 仮想機能 (VF) をゲストに直接接続します。すべてのパケットは、指定されたネットワークデバイスに直接送信されます。パススルーモードではネットワークデバイスをゲスト間で共有できないため、ネットワークデバイスは 1 つのゲストにしか渡せないことに注意してください。

図17.26 passthrough モード



Macvtap の設定は、ドメインの XML ファイルを変更するか、**virt-manager** インターフェイスを使用します。

17.12.1. ドメイン XML を使用した macvtap の設定

ゲストのドメイン XML ファイルを開き、次のように **<devices>** 要素を変更します。

```
<devices>
 ...
 <interface type='direct'>
  <source dev='eth0' mode='vepa'/>
 </interface>
</devices>
```

直接接続の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イスが接続されているハードウェアスイッチによって管理できます。

スイッチが IEEE 802.1Qbg 規格に準拠している場合、インターフェイスには以下のような追加パラメー

ターを設定できます。virtualport 要素のパラメーターについては、IEEE 802.1Qbg 標準で詳細が説明されています。この値はネットワーク固有のもので、ネットワーク管理者が指定する必要があります。802.1Qbg の用語では、Virtual Station Interface (VSI) は仮想マシンの仮想インターフェイスを表します。また、IEEE 802.1Qbg では VLAN ID にゼロ以外の値が必要です。

Virtual Station Interface のタイプ

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識別します。これは整数値で、値 0 は予約されています。

typeid

VSI タイプ ID は、ネットワークアクセスを特徴付ける VSI タイプを識別します。VSI の種類は通常、ネットワーク管理者が管理します。これは整数値です。

typeidversion

VSI タイプバージョンでは、複数のバージョンの VSI タイプが許可されます。これは整数値です。

instanceid

VSI インスタンス ID は、VSI インスタンス (仮想マシンの仮想インターフェイス) が作成されると生成されます。これは、グローバルに一意の識別子です。

profileid

プロファイル ID には、このインターフェイ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データベースにより、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、これらのネットワークパラメーターはこのインターフェイスに適用されます。

4 つのタイプはそれぞれ、ドメイン XML ファイルを変更することによって設定されます。このファイルを開いたら、以下のようにモード設定を変更します。

```
<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 が表示されます。

```
<devices>
 ...
 <interface type='direct'>
  <source dev='eth0' mode='private'/>
   <virtualport type='802.1Qbh'>
    <parameters profileid='finance'/>
   </virtualport>
```

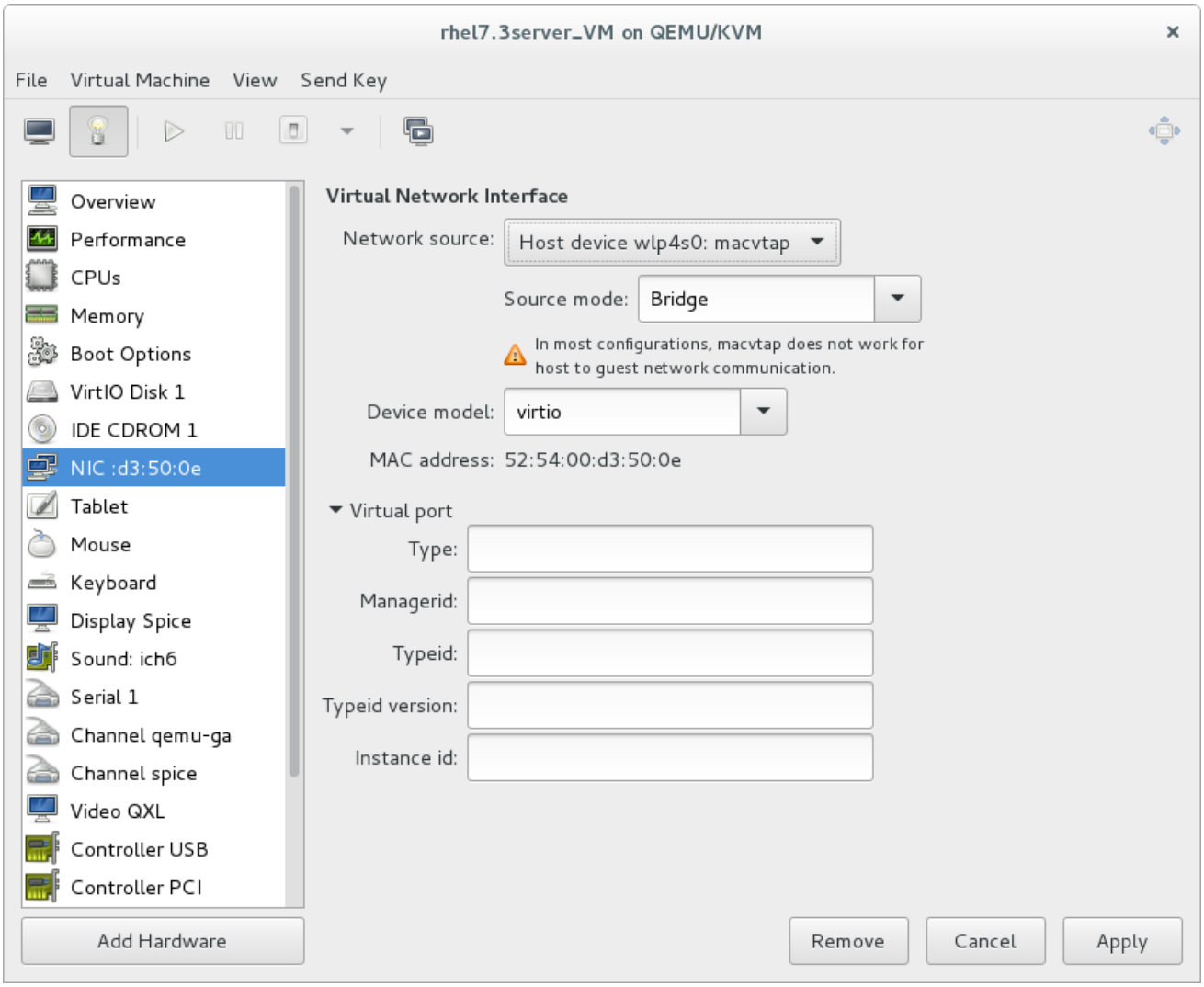
```
 </interface>
</devices>
...
```

17.12.2. virt-manager を使用した macvtap の設定

仮想ハードウェアの詳細ウィンドウ を開く⇒ メニューで NIC を選択 ⇒ Network source で host device *name*: macvtap を選択⇒ 目的の Source mode を選択。

仮想ステーションのインターフェイスタイプは、Virtual port サブメニューで設定できます。

図17.27 virt-manager での macvtap の設定



17.13. 仮想 NIC に接続しているホスト物理マシンまたはネットワークブリッジの動的な変更

本セクションでは、ゲスト仮想マシンを危険にさらすことなく、ゲスト仮想マシンの実行中に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を説明します。

1. 次のような設定で、ゲスト仮想マシンを準備します。

```
<interface type='bridge'>
      <mac address='52:54:00:4a:c9:5e'/>
      <source bridge='virbr0'/>
```

```
    <model type='virtio'/>
</interface>
```

2. インターフェイスを更新するために 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、ゲスト仮想マシンの vnetX が指定した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. 次のコマンドを使用して、新しいインターフェイス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. ゲスト仮想マシンで **service network restart** を実行します。ゲスト仮想マシンは、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 vnet0 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
```

17.14. ネットワークフィルターの適用

本セクションは、libvirt のネットワークフィルターと、その目的、概念、および XML 形式の概要を説明します。

17.14.1. 導入部分

ネットワークフィルタリングの目的は、仮想システムの管理者が、仮想マシンでネットワークトラフィックフィルタリングルールを設定および適用し、仮想マシンが送受信でき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フィルタリングルールは、仮想マシンの起動時にホストの物理マシンに適用されます。フィルタリングルールは仮想マシン内からは回避できないため、仮想マシンユーザーの視点からは必須となります。

ゲスト仮想マシンの視点からは、ネットワークフィルタリングシステムにより、各仮想マシンのネットワークトラフィックフィルタリングルールをインターフェイスごとに個別に設定できます。このルールは、仮想マシンの起動時にホストの物理マシンに適用され、仮想マシンの実行中に変更できます。後者は、ネットワークフィルターの XML 記述を変更することで実現できます。

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使用できます。このフィルターを変更すると、このフィルターを参照する実行中の仮想マシンのネットワークトラフィックフィルタリングルールがすべて更新されます。起動時に、実行していないマシンが更新されます。

前述のように、ネットワークトラフィックフィルタリングルールを適用する場合は、特定のタイプのネットワーク設定に設定された個別のネットワークインターフェイスを使用できます。対応しているネットワークタイプは、次のとおりです。

- network
- ethernet -- bridging モードで使用する必要があります。
- bridge

例17.1 ネットワークフィルタリングの例

インターフェイス XML は、トップレベルフィルターを参照するために使用されます。以下の例では、インターフェイスの説明は、フィルターの clean-traffic を参照します。

```
<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 で記述されており、他のフィルターへの参照、トラフィックフィルタリングのルール、または両方の組み合わせを含むことができます。上記の参照フィルターの clean-traffic は、他のフィルターへの参照のみが含まれ、実際のフィルタールールがないフィルターです。他のフィルターへの参照も使用できるため、フィルターツリーを構築できます。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、複数の仮想マシンで参照できます。インターフェイスは通常、各トラフィックフィルタリングルールに関連付けられた個々のパラメーターを持つため、フィルターの XML で説明されているルールは、変数を使用して一般化できます。この場合、フィルター XML では変数名が使用され、フィルターが参照される場所に名前と値が提供されます。

例17.2 説明の拡張

以下の例では、インターフェイスの説明が、パラメーター IP と、ドット付き IP アドレスを値として拡張されています。

```
<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 で表されます。ルールでは、このインターフェイスからのすべてのトラフィックが常にソース IP アドレスとして 10.0.0.1 を使用することを規定しており、これはこの特定のフィルターの目的の 1 つになります。

17.14.2. チェーンのフィルター

フィルタリングルールはフィルターチェーンで編成されています。このようなチェーンは、パケットフィルタリングルールを持つツリー構造を、個々のチェーン (分岐) のエントリーとして持つと考えることができます。

パケットは root チェーンでフィルター評価を開始し、他のチェーンで評価を継続し、これらのチェーンから root チェーンに戻るか、トラバースされたチェーンの 1 つでフィルタールールによりドロップまたは許可されます。

libvirt のネットワークフィルタリングシステムは、ユーザーがトラフィックフィルタリングのアクティブ化を選択する仮想マシンのネットワークインターフェイスごとに、個々の root チェーンを自動的に作成します。ユーザーは、root チェーンで直接インスタンス化されるフィルタリングルールを作成するか、プロトコル固有のルールを効率的に評価するためにプロトコル固有のフィルタリングチェーンを作成できます。

以下のチェーンが存在します。

- root
- mac
- stp (スパニングツリープロトコル)
- vlan
- arp と rarp
- ipv4
- ipv6

mac、stp、vlan、arp、rarp、ipv4、または ipv6 のプロトコルを評価する複数のチェーンは、プロトコル名をチェーン名の接頭辞としてのみ使用して作成できます。

例17.3 ARP トラフィックフィルタリング

この例では、名前が arp-xyz または arp-test のチェーンを指定し、そのチェーンで ARP プロトコルパケットが評価されるようにします。

以下のフィルター XML は、arp チェーンでの ARP トラフィックのフィルタリング例を示しています。

```
<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>
```

root チェーンなどではなく、ARP 固有のルールを arp チェーンに配置すると、ARP 以外のパケットプロトコルは ARP プロトコル固有のルールで評価される必要がなくなります。これにより、トラフィックフィルタリングの効率が向上します。ただし、その他のルールは評価されないため、指定したプロトコルのフィルタリングルールのみをチェーンに組み込むことに注意する必要があります。たとえば、IPv4 プロトコルパケットは ARP チェーンを通過しないため、IPv4 ルールは ARP チェーンで評価されません。

17.14.3. チェーンの優先度のフィルタリング

前述のように、フィルタールールを作成すると、すべてのチェーンが root チェーンに接続されます。これらのチェーンへのアクセス順序は、チェーンの優先度の影響を受けます。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以下の表に示します。

表17.1 チェーンのデフォルトの優先度の値のフィルタリング

チェーン (接頭辞)	デフォルトの優先度
stp	-810
mac	-800
vlan	-750
ipv4	-700
ipv6	-600
arp	-500

チェーン (接頭辞)	デフォルトの優先度
rarp	-400

注記

優先度が低いチェーンは、優先度が高いチェーンの前にアクセスされます。

表17.1「チェーンのデフォルトの優先度の値のフィルタリング」 に記載されているチェーンは、[-1000 から 1000] の範囲の値をフィルターノードの priority (XML) 属性に書き込むことで、カスタム優先度を割り当てることもできます。「チェーンのフィルター」 フィルターは、arp チェーンのデフォルト優先度 -500 を表示します。以下に例を示します。

17.14.4. フィルターにおける変数の使用

ネットワークトラフィックフィルターサブシステム (MAC と IP) で使用するために予約されている変数には、次の 2 つがあります。

MAC は、ネットワークインターフェイスの MAC アドレスに指定されます。この変数を参照するフィルタリングルールは、自動的にインターフェイスの MAC アドレスに置き換えられます。これは、MAC パラメーターを明示的に指定する必要なく機能します。上記の IP パラメーターに似た MAC パラメーターを指定できる場合でも、libvirt は、インターフェイスが使用する MAC アドレスを認識しているため、推奨できません。

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、指定のインターフェイスで使用する IP アドレスを表します。パラメーターが明示的に指定されずに参照されている場合に、インターフェイスで使用されている IP アドレス (つまり IP パラメーターの値) を libvirt デーモンが判断しようとする限り、IP パラメーターは特別なものとなります。現在の IP アドレス検出の制限は、この機能の使用方法と、その使用時に想定される制限に関する「制限」を参照してください。「チェーンのフィルター」 に示す XML ファイルには、フィルター***no-arp-spoofing***が含まれています。これは、ネットワークフィルター XML を使用して MAC 変数および IP 変数を参照する例です。

参照変数の前には、常に ***$*** という文字が付くことに注意してください。変数の値の形式は、XML で識別されるフィルター属性で期待されるタイプである必要があります。上記の例では、***IP*** パラメーターは、標準形式の正しい IP アドレスを保持する必要があります。正しい構造を指定しないと、フィルター変数が値に置き換わりません。また、ホットプラグの使用時に、仮想マシンの起動が妨げられたり、インターフェイスの接続が妨げられたりします。各 XML 属性で期待されるタイプの一部を、サンプル例17.4「サンプルの変数の種類」 に示します。

例17.4 サンプルの変数の種類

変数には要素のリストが含まれる場合があります (変数 IP には、特定のインターフェイスで有効な複数 IP アドレスを指定できるなど)。IP 変数に複数の要素を指定する表記は以下のようになります。

```
<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、インターフェイスごとに複数の IP アドレスを有効にするフィルターを作成します。各 IP アドレスは、個別のフィルタリングルールになります。そのため、上記の XML および以下のルールを使用して、3 つの個別のフィルタリングルール (各 IP アドレスに 1 つ) が作成されます。

```
<rule action='accept' direction='in' priority='500'>
  <tcp srpipaddr='$IP'/>
</rule>
```

要素のリストを含む変数の個々の要素にアクセスできるように、以下のようなフィルタリングルールは、変数*DSTPORTS* 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17.5 さまざまな変数の使用

そのため、**$VARIABLE[@<iterator id="x">]** という表記を使用して、別のリストから許可されるルールをすべて表すフィルタールールールを作成できます。以下の規則では、仮想マシンが、*DSTPORTS* で指定された一連のポートで、*SRCIPADDRESSES* で指定された送信元 IP アドレスのセットからトラフィックを受信できるようにします。この規則では、2 つの独立したイテレーターを使用して、変数 *DSTPORTS* の要素と *SRCIPADDRESSES* の要素の組み合わせをすべて生成します。

```
<rule action='accept' direction='in' priority='500'>
  <ip srcipaddr='$SRCIPADDRESSES[@1]' dstportstart='$DSTPORTS[@2]'/>
</rule>
```

以下のように、*SRCIPADDRESSES* および *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 ]
```

$SRCIPADDRESSES[@1] および **$DSTPORTS[@2]** を使用して変数に値を割り当てると、次に示すように、アドレスおよびポートのバリアントがすべて作成されます。

- 10.0.0.1, 80
- 10.0.0.1, 8080
- 11.1.2.3, 80
- 11.1.2.3, 8080

表記法 **$SRCIPADDRESSES[@1]** と **$DSTPORTS[@1]** を使用するなど、1 つのイテレーターを使用して同じ変数にアクセスすると、両方のリストにパラレルアクセスが行われ、結果が以下のようになります。

- 10.0.0.1, 80
- 11.1.2.3, 8080

注記

$VARIABLE は、**$VARIABLE[@0]** の省略形です。以前の表記法は、このセクションの上部にある最初の段落に示されているように、**iterator id="0"** が追加されたイテレーターのロールを常に想定しています。

17.14.5. 自動 IP アドレス検出と DHCP スヌーピング

本セクションでは、自動 IP アドレス検出および DHCP スヌーピングに関する情報を提供します。

17.14.5.1. 導入部分

仮想マシンのインターフェイスで使用される IP アドレスの検出は、変数 IP が参照されているにもかかわらず、その変数に値が割り当てられていない場合に、自動的にアクティブになります。変数 ***CTRL_IP_LEARNING*** を使用して、使用する IP アドレス学習方法を指定できます。有効な値は、*any*、*dhcp*、または *none* です。

any 値は、仮想マシンが使用しているアドレスを判断するために、libvirt にパケットを使用するように指示します。これは、変数 ***CTRL_IP_LEARNING*** が設定されていない場合のデフォルト設定です。この方法では、インターフェイスごとに 1 つの IP アドレスのみが検出されます。ゲスト仮想マシンの IP アドレスが検出されると、IP アドレススプーフィングがフィルターのいずれかで阻止された場合などに、そのアドレスへの IP ネットワークトラフィックがロックされます。この場合、仮想マシンのユーザーはゲスト仮想マシン内のインターフェイスで IP アドレスを変更できません。このインターフェイスは、IP アドレスのスプーフィングと見なされます。ゲスト仮想マシンが別のホスト物理マシンに移行されるか、サスペンド操作後に再開されると、ゲスト仮想マシンによって送信される最初のパケットによって、ゲスト仮想マシンが特定のインターフェイスで使用できる IP アドレスが再度決定されます。

dhcp の値は、有効なリースを持つ DHCP サーバーが割り当てられたアドレスのみを有効にするように libvirt に指示します。この方法は、インターフェイスごとに複数の IP アドレスの検出と使用をサポートします。一時停止の操作後にゲスト仮想マシンが再開すると、有効な IP アドレスリースがそのフィルターに適用されます。これを行わないと、ゲスト仮想マシンは DHCP を使用して新しい IP アドレスを取得することが期待されます。ゲスト仮想マシンを別の物理ホストの物理マシンに移行する場合は、ゲスト仮想マシンが 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 は、明示的な値を割り当てずに IP アドレスの学習および参照を行いません。これはエラーになります。

17.14.5.2. DHCP スヌーピング

CTRL_IP_LEARNING=*dhcp* (DHCP スヌーピング) では、追加のスプーフィング対策セキュリティーが提供されます。これは、信頼できる DHCP サーバーのみが IP アドレスを割り当てることができるフィルターと併用できます。これを有効にするには、変数 ***DHCPSERVER*** を、有効な DHCP サーバーの IP アドレスに設定し、この変数を使用して着信 DHCP 応答をフィルタリングするフィルターを提供します。

DHCP スヌーピングが有効で、DHCP リースが期限切れになると、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 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は、(仮想マシンインターフェイスをダウンして再度起動するなどして) 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。

注記

自動 DHCP 検出は、ゲスト仮想マシンがインフラストラクチャーの DHCP サーバーと交換する DHCP トラフィックをリッスンします。libvirt に対するサービス拒否攻撃を回避するために、これらのパケットの評価の速度が制限されます。つまり、インターフェイスで 1 秒あたりに過剰な数の DHCP パケットを送信するゲスト仮想マシンでは、これらのパケットがすべて評価されないため、フィルターが調整されない可能性があります。通常の DHCP クライアントの動作は、1 秒あたりの DHCP パケット数が少ないことが想定されます。さらに、インフラストラクチャー内のすべてのゲスト仮想マシンに適切なフィルターを設定して、DHCP パケットを送信できないようにすることが重要です。したがって、ゲスト仮想マシンがポート 67 からポート 68 への UDP および TCP のトラフィックを送信しないようにするか、DHCPSERVER 変数をすべての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された DHCP サーバーからの送信のみを許可するように制限する必要があります。同時に、サブネット内のすべてのゲスト仮想マシンで スプーフィング対策を有効にする必要があります。

例17.6 DHCP スヌーピングに対する IP のアクティブ化

以下の XML は、DHCP スヌーピング方法を使用した IP アドレス学習のアクティベーションの例を示しています。

```
<interface type='bridge'>
  <source bridge='virbr0'/>
  <filterref filter='clean-traffic'>
    <parameter name='CTRL_IP_LEARNING' value='dhcp'/>
  </filterref>
</interface>
```

17.14.6. 予約されている変数

表17.2「予約変数」 は、予約済みと見なされ、libvirt が使用する変数を示しています。

表17.2 予約変数

変数名	定義
MAC	インターフェイスの MAC アドレス
IP	インターフェイスで使用されている IP アドレスのリスト
IPV6	現在実装されていません。インターフェイスで使用中の IPV6 アドレスのリストを表示します。
DHCPSERVER	信頼できる DHCP サーバーの IP アドレスのリスト

変数名	定義
DHCPSERVERV6	現在実装されていない: 信頼できる DHCP サーバーの IPv6 アドレスのリスト
CTRL_IP_LEARNING	IP アドレス検出モードの選択

17.14.7. 要素と属性の概要

すべてのネットワークフィルターに必要な root 要素には、2 つの属性を持つ **<filter>** という名前が付けられています。**name** 属性は、指定されたフィルターの一意の名前を提供します。**chain** 属性は任意ですが、基本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を使用すると、効率的に処理するために、特定のフィルターをより適切に編成できます。現在、システムがサポートしているのは、**root**、**ipv4**、**ipv6**、**arp**、および **rarp** のチェーンのみです。

17.14.8. その他のフィルターの参照

任意のフィルターは、他のフィルターへの参照を保持できます。各フィルターは、フィルターツリーで複数回参照できますが、フィルター間の参照ではループが発生しません。

例17.7 クリーンなトラフィックフィルターの例

以下は、その他のいくつかのフィルターを参照するクリーントラフィックネットワークフィルターの XML を示しています。

```
<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するには、フィルターノード内に XML ノード **<filterref>** を提供する必要があります。このノードには、参照するフィルターの名前が含まれる属性フィルターが必要です。

新しいネットワークフィルターはいつでも定義でき、libvirt には知られていないネットワークフィルターへの参照を含むことができます。ただし、仮想マシンを起動するか、フィルターを参照しているネットワークインターフェイスをホットプラグすると、フィルターツリーの全ネットワークフィルターが利用可能になります。そうしないと、仮想マシンが起動しなくなり、ネットワークインターフェイスが接続できなくなります。

17.14.9. フィルタールール

次の XML は、発信 IP パケットの IP アドレス (変数 IP の値によって提供される) が予期されたものではない場合に、トラフィックをドロップするルールを実装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、VM による IP アドレスのスプーフィングを防ぎます。

例17.8 ネットワークトラフィックフィルタリングの例

```
<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リングルールは、ルールノードで開始します。このノードには、次の属性のうち最大 3 つを含むことができます。

- action は必須で、次の値を指定できます。
  - drop (ルールに一致すると、分析は行われずにパケットが静かに破棄されます)
  - reject (ルールを一致させると、分析は行われずに ICMP 拒否メッセージが生成されます)
  - accept (ルールを一致させるとパケットを受け付けますが、これ以上分析は行いません)
  - return (ルールを一致させるとこのフィルターが渡されますが、制御を呼び出しフィルターに戻して詳細な分析を行います)
  - continue (ルールの一致は次のルールに進み、詳細な分析が行われます)
- direction は必須で、次の値を指定できます。
  - 着信トラフィックには in
  - 送信トラフィックには out
  - 着信および送信トラフィックには inout
- priority は任意です。ルールの優先度は、他のルールに関連してルールがインスタンス化される順序を制御します。値が低いルールは、値が大きいルールの前にインスタンス化されます。有効な値の範囲は、-1000 から 1000 です。この属性を指定しないと、優先度 500 がデフォルトで割り当てられます。root チェーンのフィルタリングルールは、優先順位に従って、root チェーンに接続されたフィルターでソートルールが分類されることに注意してください。これにより、フィルタリングルールとフィルターチェーンへのアクセスをインターリーブできます。詳細は、「チェーンの優先度のフィルタリング」を参照してください。
- statematch はオプションです。設定可能な値は 0 または false で、基本的な接続状態の一致を無効にします。デフォルトの設定は true または 1 です。

詳細は、「高度なフィルター設定のトピック」 を参照してください。

上記の例の例17.7「クリーンなトラフィックフィルターの例」は、*type ip* のトラフィックがチェーン*ipv4*に関連付けられ、ルールが **priority=*500*** であることを示しています。たとえば、別のフィルターが参照されており、そのトラフィックの *type ip* がチェーン*ipv4* にも関連付けられている場合は、そのフィルターのルールが、示されているルールの **priority=*500*** を基準にして順序付けされます。

ルールには、トラフィックをフィルタリングするためのルールを 1 つだけ含めることができます。上記の例は、タイプ ip のトラフィックがフィルタリングされることを示しています。

17.14.10. サポートされているプロトコル

以下のセクションでは、ネットワークフィルターサブシステムが対応しているプロトコルをリスト表示

し、詳細を示します。このタイプのトラフィックルールは、ルールノードでネストされたノードとして提供されます。ルールがフィルタリングするトラフィックタイプに応じて、属性は異なります。上記の例は、ip トラフィックフィルタリングノード内で有効な単一の属性 ***srcipaddr*** を示しています。以下のセクションでは、どの属性が有効で、どのタイプのデータが期待されるかを示します。使用可能なデータ型は以下のとおりです。

- UINT8 : 8 ビット整数。範囲 0〜255
- UINT16 : 16 ビット整数。範囲 0〜65535
- MAC_ADDR: ドット付き 10 進数形式の MAC アドレス (例: 00:11:22:33:44:55)
- MAC_MASK: MAC アドレス形式の MAC アドレスマスク (例: FF:FF:FF:FC:00:00)
- IP_ADDR: ドット付き 10 進形式の IP アドレス (10.1.2.3 など)
- P_MASK: ドット付き 10 進数形式 (255.255.248.0) または CIDR マスク (0-32) の IP アドレスマスク
- IPV6_ADDR: 数値形式の IPv6 アドレス (例: FFFF::1)
- IPV6_MASK: 数値形式の IPv6 マスク (FFFF:FFFF:FC00::) または CIDR マスク (0-128)
- STRING: 文字列
- BOOLEAN - 'true'、'yes'、'1'、または 'false'、'no'、'0'
- IPSETFLAGS: ipset の送信元フラグおよび宛先フラグで、パケットヘッダーの送信元部分または宛先部分から機能を選択する最大 6 つの 'src' 要素または 'dst' 要素 (例: src、src、dst) で記述されます。ここで提供する '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 型または ***IPV6_MASK*** 型の属性を除くすべての属性は、value *no* の match 属性を使用して否定できます。複数の否定属性をまとめてグループ化できます。以下の XML フラグメントは、抽象属性を使用したこのような例を示しています。

```
[...]
  <rule action='drop' direction='in'>
    <protocol match='no' attribute1='value1' attribute2='value2'/>
    <protocol attribute3='value3'/>
  </rule>
[...]
```

ルールの動作は、ルールを評価するとともに、指定のプロトコル属性の境界内で論理的に調べます。したがって、1 つの属性の値が、ルールで指定された値と一致しない場合は、評価プロセス中にルール全体がスキップされます。したがって、上記の例では、protocol プロパティーの***attribute1*** が ***value1*** と protocol プロパティーの***attribute2*** が ***value2*** と一致せず、protocol プロパティーの ***attribute3*** が ***value3*** と一致する場合にのみ、着信トラフィックが破棄されます。

17.14.10.1. MAC(イーサネット)

プロトコル ID: mac

このタイプのルールは、root チェーンに置く必要があります。

表17.3 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	送信元の 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。有効な文字列は、[arp, rarp, ipv4, ipv6] などです。
comment	STRING	最大 256 文字のテキスト文字列

このフィルターは以下のように記述できます。

```
[...]
<mac match='no' srcmacaddr='$MAC'/>
[...]
```

17.14.10.2. VLAN (802.1Q)

プロトコル ID: vlan

このタイプのルールは、root チェーンまたは vlan チェーンに置く必要があります。

表17.4 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	送信元の 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	カプセル化レイヤー 3 プロトコル ID で、有効な文字列は arp、ipv4、ipv6 です。

属性名	データタイプ	定義
comment	STRING	最大 256 文字のテキスト文字列

17.14.10.3. STP (スパニングツリープロトコル)

プロトコル ID: stp

このタイプのルールは、root チェーンまたは stp チェーンのいずれかに置く必要があります。

表17.5 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	送信元の MAC アドレスに適用されるマスク
type	UINT8	BPDU (Bridge Protocol Data Unit) のタイプ
flags	UINT8	BPDU flagdstmacmask
root-priority	UINT16	root 優先度の範囲の開始
root-priority-hi	UINT16 (0x0-0xfff, 0 - 4095)	root 優先度の範囲の終了
root-address	MAC _ADDRESS	root の MAC アドレス
root-address-mask	MAC _MASK	root の MAC アドレスマスク
roor-cost	UINT32	root パスコスト (範囲開始)
root-cost-hi	UINT32	root パスコスト範囲の終了
sender-priority-hi	UINT16	送信者優先度の範囲の終了
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 (範囲開始)
port_hi	UINT16	ポート識別子の範囲の終了

属性名	データタイプ	定義
msg-age	UINT16	メッセージエージタイマー (範囲開始)
msg-age-hi	UINT16	メッセージエージタイマーの範囲の終了
max-age-hi	UINT16	最大エージタイマーの範囲の終了
hello-time	UINT16	Hello time タイマー (範囲開始)
hello-time-hi	UINT16	Hello time タイマーの範囲の終了
forward-delay	UINT16	転送遅延 (範囲開始)
forward-delay-hi	UINT16	転送遅延の範囲の終了
comment	STRING	最大 256 文字のテキスト文字列

17.14.10.4. ARP/RARP

プロトコル ID: arp または rarp

このタイプのルールは、root チェーンまたは arp/rarp チェーンのいずれかに置く必要があります。

表17.6 ARP プロトコルタイプおよび RARP 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	送信元の 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	ハードウェアの種類
protocoltype	UINT16	プロトコルタイプ

属性名	データタイプ	定義
opcode	UINT16, STRING	Opcode の有効な文字列は、Request、Reply、Request_Reverse、Reply_Reverse、DRARP_Request、DRARP_Reply、DRARP_Error、InARP_Request、ARP_NAK です。
arpsrcmacaddr	MAC_ADDR	ARP/RARP パケットのソース MAC アドレス
arpdstmacaddr	MAC _ADDR	ARP/RARP パケットの宛先 MAC アドレス
arpsrcipaddr	IP_ADDR	ARP/RARP パケットのソース IP アドレス
arpdstipaddr	IP_ADDR	ARP/RARP パケットの宛先 IP アドレス
gratuitous	BOOLEAN	Gratuitous ARP パケットを確認するかどうかを示すブール値
comment	STRING	最大 256 文字のテキスト文字列

17.14.10.5. IPv4

プロトコル ID: ip

このタイプのルールは、root チェーンまたは ipv4 チェーンに存在する必要があります。

表17.7 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	送信元の MAC アドレスに適用されるマスク
dstmacaddr	MAC_ADDR	宛先の MAC アドレス
dstmacmask	MAC_MASK	宛先の MAC アドレスに適用されるマスク
srcipaddr	IP_ADDR	ソース IP アドレス

属性名	データタイプ	定義
srcipmask	IP_MASK	マスクがソース IP アドレスに適用されました。
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
protocol	UINT8, STRING	レイヤー 4 プロトコル識別子。protocol に有効な文字列は、tcp、udp、udplite、esp、ah、icmp、igmp、sctp などです。
srcportstart	UINT16	有効なソースポートの範囲の開始。プロトコルが必要です。
srcportend	UINT16	有効なソースポートの範囲の終了。プロトコルが必要です。
dstportstart	UNIT16	有効な宛先ポートの範囲の開始。プロトコルが必要です。
dstportend	UNIT16	有効な宛先ポートの範囲の終了。プロトコルが必要です。
comment	STRING	最大 256 文字のテキスト文字列

17.14.10.6. IPv6

プロトコル ID: ipv6

このタイプのルールは、root チェーンまたは ipv6 チェーンに存在する必要があります。

表17.8 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	送信元の MAC アドレスに適用されるマスク
dstmacaddr	MAC_ADDR	宛先の MAC アドレス
dstmacmask	MAC_MASK	宛先の MAC アドレスに適用されるマスク

属性名	データタイプ	定義
srcipaddr	IP_ADDR	ソース IP アドレス
srcipmask	IP_MASK	マスクがソース IP アドレスに適用されました。
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
protocol	UINT8, STRING	レイヤー 4 プロトコル識別子。protocol に有効な文字列は、tcp、udp、udplite、esp、ah、icmpv6、sctp などです。
scrportstart	UNIT16	有効なソースポートの範囲の開始。プロトコルが必要です。
srcportend	UINT16	有効なソースポートの範囲の終了。プロトコルが必要です。
dstportstart	UNIT16	有効な宛先ポートの範囲の開始。プロトコルが必要です。
dstportend	UNIT16	有効な宛先ポートの範囲の終了。プロトコルが必要です。
comment	STRING	最大 256 文字のテキスト文字列

17.14.10.7. TCP/UDP/SCTP

プロトコル ID: tcp、udp、sctp

chain パラメーターはこのタイプのトラフィックでは無視されるため、省略するか、root に設定してください。

表17.9 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	マスクがソース IP アドレスに適用されました。

属性名	データタイプ	定義
dstipaddr	IP_ADDR	宛先 IP アドレス
dstipmask	IP_MASK	宛先 IP アドレスに適用されるマスク
scripto	IP_ADDR	ソース IP アドレスの範囲の開始
srcipfrom	IP_ADDR	ソース IP アドレスの範囲の終了
dstipfrom	IP_ADDR	宛先 IP アドレスの範囲の開始
dstipto	IP_ADDR	宛先 IP アドレスの範囲の終了
scrportstart	UNIT16	有効なソースポートの範囲の開始。プロトコルが必要です。
srcportend	UINT16	有効なソースポートの範囲の終了。プロトコルが必要です。
dstportstart	UNIT16	有効な宛先ポートの範囲の開始。プロトコルが必要です。
dstportend	UNIT16	有効な宛先ポートの範囲の終了。プロトコルが必要です。
comment	STRING	最大 256 文字のテキスト文字列
state	STRING	NEW、ESTABLISHED、RELATED、INVALID、または NONE のコンマ区切りのリスト
flags	STRING	TCP のみ - mask/flags の形式で、mask と flags の各々が、SYN、ACK、URG、PSH、FIN、RST、NONE、ALL のコンマで区切りリストになります。
ipset	STRING	libvirt の外部で管理される IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要です。

17.14.10.8. ICMP

プロトコル ID: icmp

注記: chain パラメーターはこのタイプのトラフィックでは無視されるため、省略するか、root に設定してください。

表17.10 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	マスクがソース IP アドレスに適用されました。
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 アドレスの範囲の開始
scripto	IP_ADDR	ソース IP アドレスの範囲の終了
dstipfrom	IP_ADDR	宛先 IP アドレスの範囲の開始
dstipto	IP_ADDR	宛先 IP アドレスの範囲の終了
type	UNIT16	ICMP のタイプ
code	UNIT16	ICMP コード
comment	STRING	最大 256 文字のテキスト文字列
state	STRING	NEW、ESTABLISHED、RELATED、INVALID、または NONE のコンマ区切りのリスト
ipset	STRING	libvirt の外部で管理される IPSet の名前

属性名	データタイプ	定義
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要です。

17.14.10.9. IGMP、ESP、AH、UDPLITE、'ALL'

プロトコル ID - igmp、esp、ah、udplite、all

chain パラメーターはこのタイプのトラフィックでは無視されるため、省略するか、root に設定してください。

表17.11 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	マスクがソース IP アドレスに適用されました。
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 アドレスの範囲の開始
scripto	IP_ADDR	ソース IP アドレスの範囲の終了
dstipfrom	IP_ADDR	宛先 IP アドレスの範囲の開始
dstipto	IP_ADDR	宛先 IP アドレスの範囲の終了
comment	STRING	最大 256 文字のテキスト文字列

属性名	データタイプ	定義
state	STRING	NEW、ESTABLISHED、RELATED、INVALID、または NONE のコンマ区切りのリスト
ipset	STRING	libvirt の外部で管理される IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要です。

17.14.10.10. IPV6 経由の TCP/UDP/SCTP

プロトコル ID: tcp-ipv6、udp-ipv6、sctp-ipv6

chain パラメーターはこのタイプのトラフィックでは無視されるため、省略するか、root に設定してください。

表17.12 IPv6 のプロトコルタイプ経由の TCP、UDP、SCTP

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	マスクがソース IP アドレスに適用されました。
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 アドレスの範囲の開始
scripto	IP_ADDR	ソース IP アドレスの範囲の終了
dstipfrom	IP_ADDR	宛先 IP アドレスの範囲の開始
dstipto	IP_ADDR	宛先 IP アドレスの範囲の終了
srcportstart	UINT16	有効なソースポートの範囲の開始
srcportend	UINT16	有効なソースポートの範囲の終了
dstportstart	UINT16	有効な宛先ポートの範囲の開始

属性名	データタイプ	定義
dstportend	UINT16	有効な宛先ポートの範囲の終了
comment	STRING	最大 256 文字のテキスト文字列
state	STRING	NEW、ESTABLISHED、RELATED、INVALID、または NONE のコンマ区切りのリスト
ipset	STRING	libvirt の外部で管理される IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要です。

17.14.10.11. ICMPv6

プロトコル ID: icmpv6

chain パラメーターはこのタイプのトラフィックでは無視されるため、省略するか、root に設定してください。

表17.13 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	マスクがソース IP アドレスに適用されました。
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 アドレスの範囲の開始
scripto	IP_ADDR	ソース IP アドレスの範囲の終了
dstipfrom	IP_ADDR	宛先 IP アドレスの範囲の開始
dstipto	IP_ADDR	宛先 IP アドレスの範囲の終了
type	UINT16	ICMPv6 のタイプ

属性名	データタイプ	定義
code	UINT16	ICMPv6 コード
comment	STRING	最大 256 文字のテキスト文字列
state	STRING	NEW、ESTABLISHED、RELATED、INVALID、または NONE のコンマ区切りのリスト
ipset	STRING	libvirt の外部で管理される IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要です。

17.14.10.12. IPv6 経由の IGMP, ESP, AH, UDPLITE, 'ALL'

プロトコル ID - igmp-ipv6、esp-ipv6、ah-ipv6、udplite-ipv6、all-ipv6

chain パラメーターはこのタイプのトラフィックでは無視されるため、省略するか、root に設定してください。

表17.14 IPv6 プロトコルタイプ経由の 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
srcipaddr	IP_ADDR	ソース IP アドレス
srcipmask	IP_MASK	マスクがソース IP アドレスに適用されました。
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 アドレスの範囲の開始
scripto	IP_ADDR	ソース IP アドレスの範囲の終了
dstipfrom	IP_ADDR	宛先 IP アドレスの範囲の開始
dstipto	IP_ADDR	宛先 IP アドレスの範囲の終了
comment	STRING	最大 256 文字のテキスト文字列

属性名	データタイプ	定義
state	STRING	NEW、ESTABLISHED、RELATED、INVALID、または NONE のコンマ区切りのリスト
ipset	STRING	libvirt の外部で管理される IPSet の名前
ipsetflags	IPSETFLAGS	IPSet のフラグ。ipset 属性が必要です。

17.14.11. 高度なフィルター設定のトピック

次のセクションでは、高度なフィルター設定のトピックについて説明します。

17.14.11.1. 接続追跡

ネットワークフィルターサブシステム (Linux の場合) は、IP テーブルの接続追跡サポートを利用します。これにより、ネットワークトラフィックの方向を強制 (状態の一致) したり、ゲスト仮想マシンへの同時接続数をカウントして制限したりできます。たとえば、ゲスト仮想マシンで TCP ポート 8080 がサーバーとして開いていると、クライアントはポート 8080 でゲスト仮想マシンに接続できます。接続の追跡と方向の強制により、ゲスト仮想マシンが (TCP クライアント) ポート 8080 からホスト物理マシンへの接続を開始して、リモートホスト物理マシンに戻らないようにします。さらに重要なのは、トラッキングにより、リモートの攻撃者がゲスト仮想マシンへの接続を確立できないようにすることです。たとえば、ゲスト仮想マシン内のユーザーが攻撃者サイトのポート 80 への接続を確立すると、攻撃者は TCP ポート 80 からゲスト仮想マシンへの接続を開始できなくなります。デフォルトでは、接続追跡を有効にした後にトラフィックの方向を強制する接続状態一致が有効になります。

例17.9 TCP ポートへの接続を無効にするなどの XML

以下は、TCP ポート 12345 への着信接続でこの機能が無効になっている XML フラグメントの例を示しています。

```
[...]
 <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るようになりましたが、仮想マシン内の (クライアント) TCP ポート 12345 からの開始も有効になります。これは望ましい場合とそうでない場合があります。

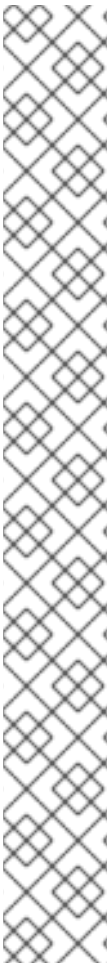
17.14.11.2. 接続数の制限

ゲスト仮想マシンが確立できる接続の数を制限するには、特定タイプのトラフィックの接続の制限を設定するルールを提供する必要があります。たとえば、仮想マシンが、一度に 1 つの IP アドレスに対してのみ ping を実行できるようにし、同時に 1 つの ssh 受信接続のみをアクティブにする必要がある場合など。

例17.10 接続に制限を設定する XML サンプルファイル

以下の XML フラグメントを使用して、接続を制限できます。

```
[...]
<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に先立ち、XML に記載する必要があります。例17.10「接続に制限を設定する XML サンプルファイル」 の XML ファイルに従い、ポート 22 に送信された DNS トラフィックがゲスト仮想マシンから送信されることを許可するための追加ルールが追加され、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いことを回避します。このルールを適用しないと、接続しようとしているときに ssh クライアントが予想外にハングする可能性があります。トラフィックの追跡に関連するタイムアウトを処理する際には、注意が必要です。ゲスト仮想マシン内でユーザーが終了した ICMP ping は、ホスト物理マシンの接続追跡システムでタイムアウトが長くなる可能性があるため、別の ICMP ping を実行できません。

最善の解決策は、# **echo 3 > /proc/sys/net/netfilter/nf_conntrack_icmp_timeout** で、ホスト物理マシンの **sysfs** のタイムアウトを調整することです。このコマンドは、ICMP 接続トラッキングのタイムアウトを 3 秒に設定します。これは、ある ping が終了すると、別の ping が 3 秒後に開始できることになります。

なんらかの理由でゲスト仮想マシンが TCP 接続を適切に閉じていないと、特にホストの物理マシンで TCP タイムアウト値が大量に設定されている場合に、接続が長期間開かれたままになります。また、アイドル状態の接続では、パケットが交換されると再度アクティブにできる接続追跡システムのタイムアウトが発生する場合があります。

ただし、この制限を低く設定しすぎると、新しく開始した接続により、アイドル状態の接続が強制的に TCP バックオフになる場合があります。したがって、接続の制限は、新しい TCP 接続の変動によって、アイドル状態の接続に関連して奇妙なトラフィック動作が発生しないように、かなり高く設定する必要があります。

17.14.11.3. コマンドラインツール

virsh は、ネットワークフィルターのライフサイクルサポートにより拡張されました。ネットワークフィルターサブシステムに関連するすべてのコマンドは、***nwfilter*** という接頭辞で始まります。以下のコマンドを使用できます。

- **nwfilter-list** - UUID と、すべてのネットワークフィルターの名前のリストを表示します。
- **nwfilter-define** - 新しいネットワークフィルターを定義するか、既存のフィルターを更新します (名前を指定する必要があります)。
- **nwfilter-undefine** - 指定したネットワークフィルターを削除します (名前を指定する必要があります)。現在使用しているネットワークフィルターを削除しないでください。
- **nwfilter-dumpxml** - 指定したネットワークフィルターを表示します (名前を指定する必要があります)。
- **nwfilter-edit** - 指定したネットワークフィルターを編集します (名前を指定する必要があります)。

17.14.11.4. 既存のネットワークフィルター

以下は、libvirt で自動的にインストールされるネットワークフィルターの例です。

表17.15 ICMPv6 プロトコルタイプ

プロトコル名	説明
allow-arp	ゲスト仮想マシンへのすべての送受信アドレス解決プロトコル (ARP) トラフィックを許可します。
no-arp-spoofing、no-arp-mac-spoofing、and no-arp-ip-spoofing	これらのフィルターは、ゲスト仮想マシンが ARP トラフィックをスプーフィングするのを防ぎます。また、ARP の要求メッセージおよび応答メッセージのみを許可し、これらのパケットには次の内容が含まれることを強制します。 • no-arp-spoofing - ゲストの MAC アドレスおよび IP アドレス • no-arp-mac-spoofing - ゲストの MAC アドレス • no-arp-ip-spoofing - ゲストの IP アドレス
low-dhcp	ゲスト仮想マシンが DHCP 経由で (任意の DHCP サーバーから) IP アドレスを要求できるようにします。
low-dhcp-server	ゲスト仮想マシンが、指定した DHCP サーバーから IP アドレスを要求できるようにします。DHCP サーバーの、ドットで区切られた 10 進数の IP アドレスは、このフィルターを参照する必要があります。変数の名前は *DHCPSERVER* でなければなりません。

プロトコル名	説明
low-ipv4	仮想マシンへのすべての送受信 IPv4 トラフィックを許可します。
low-incoming-ipv4	仮想マシンへの着信 IPv4 トラフィックのみを受け入れます。このフィルターは **clean-traffic** フィルターの一部です。
no-ip-spoofing	ゲスト仮想マシンが、パケット内の送信元 IP アドレスとは異なる IP パケットを送信しないようにします。このフィルターは **clean-traffic** フィルターの一部です。
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しないようにします。
no-mac-broadcast	指定された MAC アドレスへの送信 IPv4 トラフィックを防ぎます。このフィルターは **clean-traffic** フィルターの一部です。
no-other-l2-traffic	ネットワークが使用するその他のフィルターで指定したトラフィックを除き、すべてのレイヤー 2 ネットワークトラフィックを防ぎます。このフィルターは **clean-traffic** フィルターの一部です。
no-other-rarp-traffic、**qemu-announce-self**、**qemu-announce-self-rarp**	このフィルターは、QEMU のセルフアナウンスの RARP (Reverse Address Resolution Protocol) パケットを許可しますが、その他の RARP トラフィックは阻止します。これらのすべては、**clean-traffic** フィルターにも含まれます。
clean-traffic	MAC、IP、および ARP のスプーフィングを防ぎます。このフィルターは、他のいくつかのフィルターをビルディングブロックとして参照します。

これらのフィルターはビルディングブロックにすぎず、便利なネットワークトラフィックフィルターを提供するために、他のフィルターとの組み合わせが必要です。上記で最も使用されているのが *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て、仮想マシンが IP マルチキャストトラフィックを送信しないようにし、パケットスプーフィングを防ぐことができます。

17.14.11.5. 独自のフィルターの作成

libvirt はネットワークフィルターの例をいくつか提供しているだけなので、独自のフィルターを作成することを検討してください。これを計画する際に、ネットワークフィルターサブシステムと、それが内部でどのように機能するかについて知っておく必要があります。フィルターをかけるプロトコルを必ず熟知して理解しておく必要があります。それにより、通過するトラフィック以上のトラフィックがなくなり、実際に許可するトラフィックが通過するようになります。

ネットワークフィルターサブシステムは、現在 Linux ホストの物理マシンでのみ利用でき、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、ebtable、iptables、および ip6tables に対応し、その機能を利用します。「サポートされているプロトコル」 に記載されているリストを考慮し、ebtable を使用して以下のプロトコルを実装できます。

- mac
- stp (スパニングツリープロトコル)
- vlan (802.1Q)
- arp, rarp
- ipv4
- ipv6

IPv4 で実行するプロトコルはすべて iptables を使用してサポートされ、IPv6 で実行するプロトコルは ip6tables を使用して実装されます。

Linux ホストの物理マシンを使用して、libvirt のネットワークフィルタリングサブシステムが作成したすべてのトラフィックフィルタリングルールは、最初に ebtables が実装したフィルタリングサポートを通過し、その後で iptables フィルターまたは ip6tables フィルターを通過します。フィルターツリーに、mac、stp、vlan arp、rarp、ipv4、または ipv6 などのプロトコルを持つルールがある場合は、リスト表示されている ebtable ルールと値が自動的に使用されます。

同じプロトコルに複数のチェーンを作成できます。チェーンの名前は、以前に列挙されたプロトコルのいずれかの接頭辞を持つ必要があります。ARP トラフィックを処理するための追加のチェーンを作成する場合は、たとえば、名前が arp-test のチェーンを指定できます。

たとえば、ip プロトコルフィルターを使用し、受け入れる UDP パケットのソース、宛先 IP、およびポートの属性を指定して、送信元および宛先のポートで UDP トラフィックでフィルタリングできます。これにより、ebtables を使用した UDP トラフィックの早期フィルタリングが可能になります。ただし、UDP パケットなどの IP または IPv6 パケットが ebtables レイヤーを通過し、iptables ルールまたは ip6tables ルールをインスタンス化するフィルターツリーに少なくとも 1 つのルールがある場合、UDP パケットの通過を許可するルールもこれらのフィルターレイヤーに指定する必要があります。これは、適切な udp トラフィックフィルターノードまたは udp-ipv6 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17.11 カスタムフィルターの作成**
>
> 以下の要件のリストを満たすためにフィルターが必要であるとします。
>
> - 仮想マシンのインターフェイスによる MAC、IP、および ARP のスプーフィングの阻止
> - 仮想マシンのインターフェイスの TCP ポート 22 および 80 のみを開く
> - 仮想マシンがインターフェイスから ping トラフィックを送信できるようにしますが、インターフェイスで仮想マシンの ping を行わないようにします。
> - 仮想マシンが DNS ルックアップ (ポート 53 への UDP) を実行できるようにします。
>
> スプーフィングを防ぐための要件は、すでに存在する ***clean-traffic*** ネットワークフィルターにより満たされています。そのため、これを行う方法は、カスタムフィルターから参照することです。
>
> TCP ポート 22 および 80 のトラフィックを有効にするには、このタイプのトラフィックを有効にするために 2 つのルールが追加されました。ゲスト仮想マシンが ping トラフィックを送信できるようにするため、ICMP トラフィックにルールが追加されました。簡単にするために、一般的な ICMP ト

ラフィックはゲスト仮想マシンから開始することが許可され、ICMP エコー要求および応答メッセージには指定されません。その他のトラフィックはすべて、ゲスト仮想マシンに到達しないか、ゲスト仮想マシンにより開始されます。これを行うには、その他のトラフィックをすべて破棄するルールが追加されます。ゲスト仮想マシンが **test** と呼ばれ、フィルターを関連付けるインターフェイスが **eth0** と呼ばれていると、フィルターの名前は **test-eth0** となります。

この考慮事項の結果、ネットワークフィルターの XML が次のようになります。

```
<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>
```

17.14.11.6. サンプルのカスタムフィルター

上記の XML のルールのいずれかに、送信元アドレスまたは宛先アドレスとしてゲスト仮想マシンの 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リングは正しく機能します。理由は、ルールの評価はインターフェイスごとに内部的に行われるのに対し、ルールは、送信元または宛先の IP アドレスではなく、パケットを送信または受信する (タップ) インターフェイスに基づいて追加で評価されるためです。

例17.12 ネットワークインターフェイスの説明のサンプル XML

テストゲスト仮想マシンのドメイン XML 内にある可能性のあるネットワークインターフェイスの説明の XML フラグメントは、以下のようになります。

```
[...]
 <interface type='bridge'>
   <source bridge='mybridge'/>
   <filterref filter='test-eth0'/>
 </interface>
[...]
```

ICMP トラフィックをより厳格に制御し、ゲスト仮想マシンから送信できるのは ICMP エコー要求のみで、ゲスト仮想マシンが受信できるのは ICMP エコー応答のみとなるようにするため、上記の ICMP ルールを、以下の 2 つのルールに置き換えることができます。

```
<!- - 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17.13 2 番目の例のカスタムフィルター

この例は、上記の例と同様のフィルターを作成する方法を示していますが、ゲスト仮想マシン内にある ftp サーバーを使用して要件のリストを拡張しています。このフィルターの要件は以下のとおりです。

- ゲスト仮想マシンのインターフェイスの MAC、IP、および ARP スプーフィングを防ぎます。
- ゲスト仮想マシンのインターフェイスで TCP ポート 22 および 80 のみを開きます。
- ゲスト仮想マシンがインターフェイスから ping トラフィックを送信できるようにしますが、インターフェイスでゲスト仮想マシンへの ping は許可しません。
- ゲスト仮想マシンが DNS ルックアップ (ポート 53 への UDP) を実行できるようにします。
- ftp サーバーを有効にし (アクティブモード)、ゲスト仮想マシン内で実行できるようにします。

FTP サーバーをゲスト仮想マシン内で実行できるようにするという追加の要件は、ポート 21 が FTP 制御トラフィックに到達できるようにするという要件と、ゲスト仮想マシンが、ゲスト仮想マシンの TCP ポート 20 からの発信 TCP 接続を FTP クライアント (FTP アクティブモード) に確立できるようにする要件にマッピングします。このフィルターを作成する方法はいくつかあります。この例では、2 つの解決策を説明します。

最初のソリューションは、TCP プロトコルの state 属性を使用します。この属性は、Linux ホストの物理マシンの接続追跡フレームワークにフックを提供します。ゲスト仮想マシンが開始する FTP データ接続 (FTP アクティブモード) の場合は、RELATED 状態を使用して、ゲスト仮想マシンが開始する FTP データ接続が既存の FTP 制御接続の結果である (または関連している) ことを検出できるようにします。これにより、ファイアウォールを介して、パケットを渡すことができます。ただし、RELATED 状態は、FTP データパスの送信 TCP 接続の最初のパケットに対してのみ有効です。

その後、この状態は ESTABLISHED になり、着信方向および発信方向にも同様に適用されます。これはすべて、ゲスト仮想マシンの TCP ポート 20 から発信される FTP データトラフィックに関連します。これにより、以下のソリューションが得られます。

```
<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 状態を使用し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読み込ま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、

ゲスト仮想マシンと FTP 接続を確立する前に、次の 2 つのコマンドのいずれかを実行する必要があります。

- **modprobe nf_conntrack_ftp** (利用可能な場合)
- **modprobe ip_conntrack_ftp** (上記が利用可能でない場合)

FTP 以外のプロトコルを RELATED 状態と併用する場合は、対応するモジュールを読み込む必要があります。モジュールは、ftp、tftp、irc、sip、sctp、および amanda のプロトコルで利用できます。

2 番目のソリューションでは、以前のソリューションよりも多くの接続の状態フラグを使用します。このソリューションでは、トラフィックフローの最初のパケットが検出されると、接続の NEW 状態が有効であるという事実を利用します。その後、フローの最初のパケットが受け入れられると、フローは接続になるため、ESTABLISHED 状態になります。したがって、ESTABLISHED 接続のパケットをゲスト仮想マシンに到達できるようにしたり、ゲスト仮想マシンによって送信されるように、一般的なルールを記述することができます。これは、NEW 状態で識別される最も最初のパケットに特定のルールを書き込み、データが許容可能なポートを指定します。明示的に許可されていないポート向けのパケットはすべて破棄されるため、ESTABLISHED 状態に到達しません。そのポートから送信される後続のパケットもすべて破棄されます。

```
<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>
```

17.14.12. 制限

以下は、ネットワークフィルターサブシステムにおける現在知られている制限のリストです。

- 仮想マシンの移行は、ゲスト仮想マシンのトップレベルフィルターが参照するフィルターツリー全体が、ターゲットホストの物理マシンで利用可能な場合にのみ対応します。たとえば、ネットワークフィルターの **clean-traffic** は、すべての libvirt インストールで利用可能である必要があり、このフィルターを参照するゲスト仮想マシンの移行を有効にする必要があります。バージョンの互換性が問題ではないことを確認するには、パッケージを定期的に更新して、最新バージョンの libvirt を使用していることを確認してください。

- インターフェイスに関連付けられたネットワークトラフィックフィルターを失わないように、バージョン 0.8.1 以降の libvirt インストール間で移行を行う必要があります。

- ゲスト仮想マシンが VLAN (802.1Q) パケットを送信しても、プロトコル ID の arp、rarp、ipv4、ipv6 のルールでフィルターをかけることはできません。これらは、プロトコル ID、MAC、および VLAN でのみフィルターにかけることができます。そのため、フィルターの clean-traffic の例 例17.1「ネットワークフィルタリングの例」 は予想通りに機能しません。

17.15. トンネルの作成

本セクションでは、さまざまなトンネルシナリオを実装する方法を説明します。

17.15.1. マルチキャストトンネルの作成

マルチキャストグループは、仮想ネットワークを表すように設定されています。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、ホストの物理マシン全体であっても互いに通信できます。このモードは、特権を持たないユーザーでも使用できます。デフォルトの DNS または DHCP に対応せず、発信ネットワークアクセスもありません。外部への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 2 つ目の NIC が必要です。この NIC は、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、適切なルーティングを提供します。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。提供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のものである必要があることに注意してください。

マルチキャストトンネルを作成するには、次の XML ディテールを **<devices>** 要素に配置します。

図17.28 マルチキャストトンネルドメイン XMl の例

```
...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
    <source address='230.0.0.1' port='5558'/>
  </interface>
</devices>
...
```

17.15.2. TCP トンネルの作成

TCP のクライアント/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、1 つのゲスト仮想マシンがネットワークのサーバー側を提供し、その他のゲスト仮想マシンはすべてクライアントとして設定されます。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介してゲスト仮想マシンクライアント間でルーティングされます。このモードは、非特権ユーザーにも使用できます。このモードは、デフォルトの DNS または DHCP サポートを提供しておらず、送信ネットワークアクセスは提供されない点に注意してください。外部への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 2 つ目の NIC が必要です。この NIC は、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、適切なルーティングを提供します。

TCP トンネルを作成するには、以下の XML ディテールを **<devices>** 要素に配置します。

図17.29 TCP トンネルドメイン XMl の例

```
    ...
<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>
...
```

17.16. VLAN タグの設定

仮想ローカルエリアネットワーク (vLAN) タグは、**virsh net-edit** コマンドを使用して追加します。このタグは、SR-IOV デバイスで PCI デバイスの割り当てと併用することもできます。詳細は、「SR-IOV デバイスを使用した PCI 割り当ての設定」 を参照してください。

図17.30 vVLAN タグの設定 (対応しているネットワークタイプのみ)

```
<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>
```

ネットワークタイプが、ゲストに対して透過的な vlan タグ付けをサポートする場合にのみ、任意の **<vlan>** 要素は、このネットワークを使用して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 タグ付けをサポートします。標準の linux ブリッジや libvirt 自体の仮想ネットワークを含むその他の場合はすべて、これをサポートしません。802.1 Qbh (vn-link) スイッチおよび 802.1Qbg (VEPA) スイッチは、(libvirt の外部で) ゲストトラフィックを特定の vlan にタグ付けする独自の方法を提供します。) 想定どおりに、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は、指定されたすべてのタグを使用して VLAN トランクを実行することが想定されます。1 つのタグでの vlan トランクが必要な場合は、オプションの属性 trunk='yes' を vlan 要素に追加できます。

openvswitch を使用したネットワーク接続では、'native-tagged' モードおよび 'native-untagged' vlan モードを設定できます。**<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 に設定できます。要素の id 属性は、ネイティブ vlan を設定します。

<vlan> 要素は、**<portgroup>** 要素でも指定できます。また、ドメインの **<interface>** 要素でも指定できます。複数の場所で vlan タグが指定されている場合は、**<interface>** の設定が優先され、その後にインターフェイス設定で選択した**<portgroup>** の設定が優先されます。**<network>** の**<vlan>** は、**<portgroup>** または **<interface>** に指定がない場合にのみ選択されます。

17.17. 仮想ネットワークへの QOS 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最適なエクスペリエンスを保証するリソース制御システムを指し、遅延、ジッター、またはパケットロスがないことを確認します。QoS は、アプリケーション固有またはユーザー/グループ固有の場合があります。詳細は、「QoS (Quality of Service)」 を参照してください。

第18章 ゲストのリモート管理

本セクションでは、ゲストをリモートで管理する方法を説明します。

18.1. トランスポートモード

リモートマネジメントの場合、**libvirt** は以下のトランスポートモードに対応します。

トランスポートレイヤーセキュリティー (Transport Layer Security, TLS)

Transport Layer Security TLS 1.0 (SSL 3.1) が認証され、暗号化された TCP/IP ソケット。通常はパブリックポート番号をリッスンします。これを使用するには、クライアント証明書およびサーバー証明書を生成する必要があります。標準のポートは 16514 です。詳細な手順は、「TLS および SSL を使用したリモート管理」 を参照してください。

SSH

SSH (Secure Shell Protocol) 接続で転送されます。libvirt デーモン (**libvirtd**) は、リモートマシンで実行している必要があります。SSH アクセスには、ポート 22 を開いておく必要があります。ある種の SSH キー管理 (たとえば、**ssh-agent**ユーティリティー) を使用する必要があります。そうしないと、パスワードの入力を求められます。詳細な手順は、「SSH を使用したリモート管理」 を参照してください。

UNIX ソケット

UNIX ドメインソケットは、ローカルマシンでのみアクセスできます。ソケットは暗号化されず、UNIX の権限または SELinux を使用して認証を行います。通常のソケット名は **/var/run/libvirt/libvirt-sock** および **/var/run/libvirt/libvirt-sock-ro** です (読み取り専用接続の場合)。

ext

ext パラメーターは、libvirt のスコープ外の方法でリモートマシンに接続できる外部プログラムに使用されます。このパラメーターはサポートされていません。

TCP

暗号化されていない TCP/IP ソケット。実稼働環境での使用は推奨されていないため、これは通常無効になっていますが、管理者はこれをテストしたり、信頼できるネットワークで使用したりできます。デフォルトのポートは 16509 です。

デフォルトのトランスポートは、その他のトランスポートを指定しないと TLS になります。

リモート URI

URI (Uniform Resource Identifier) は、**virsh** および libvirt がリモートホストに接続するために使用します。URI は、**virsh** コマンドの **--connect** パラメーターとともに使用して、リモートホストで単一コマンドまたは移行を実行することもできます。リモート URI は、通常のローカル URI を取得し、ホスト名またはトランスポート名 (あるいはその両方) を追加することで形成されます。特別な場合として、remote の URI スキームを使用すると、リモートの libvirtd サーバーで、最適なハイパーバイザードライバーをプローブするように指示されます。これは、ローカル接続の NULL URI を渡すことに相当します。

libvirt の URI は一般的な形式をとります (角括弧内のコンテンツ "[]" は任意の関数を表します)。

```
driver[+transport]://[username@][hostname][:port]/path[?extraparameters]
```

ハイパーバイザー (ドライバー) が QEMU の場合は、パスが必須であることに注意してください。

以下は、有効なリモート URI の例になります。

- qemu://hostname/

外部の場所を対象とする場合は、トランスポート方法またはホスト名を指定する必要があります。詳細は、libvirt アップストリームのドキュメント を参照してください。

リモート管理パラメーターの例

- SSH トランスポートと、SSH ユーザーネーム **virtuser** を使用して、**host2** という名前のリモートの KVM ホストに接続します。各 connect コマンドは **[*URI*] [--readonly]** です。**virsh connect** コマンドの詳細は、「virsh Connect を使用したハイパーバイザーへの接続」 を参照してください。

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

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

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

テスト例

- 標準以外の UNIX ソケットを使用して、ローカルの KVM ハイパーバイザーに接続します。この場合は、UNIX ソケット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
```

- 暗号化されていない TCP/IP 接続を使用して、ポート 5000 の IP アドレス 10.1.1.10 のサーバーに libvirt デーモンを接続します。これは、デフォルト設定でテストドライバーを使用します。

```
test+tcp://10.1.1.10:5000/default
```

追加の URI パラメーター

リモートの URI には、追加のパラメーターを追加できます。以下の表では、認識されるパラメーターを説明します。その他のパラメーターはすべて無視されます。パラメーター値は URI エスケープする必要があります (つまり、パラメーターおよび特殊文字が URI 形式に変換される前に疑問符 (?) が追加されることを意味します)。

表18.1 追加の URI パラメーター

Name	トランスポートモード	説明	使用例

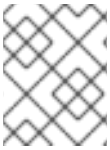
Name	トランスポートモード	説明	使用例
name	すべてのモード	リモート **virConnectOpen** 関数に渡される名前。名前は、通常、リモートの URI から ***transport***、***hostname***、***port number***、***username*** 、および追加のパラメーターを削除して生成されますが、非常に複雑なケースでは、名前を明示的に指定することを推奨します。	name=qemu:///system
command	ssh と ext	外部コマンド。ext トランスポートの場合はこれが必要です。ssh の場合、デフォルトは ssh です。コマンド用に PATH が検索されます。	command=/opt/openssh/bin/ssh
socket	unix と ssh	UNIX ドメインソケットへのパスで、デフォルトを上書きします。ssh トランスポートの場合は、これがリモートの **netcat** コマンド (netcat を参照) に渡されます。	socket=/opt/libvirt/run/libvirt/libvirt-sock
no_verify	tls	ゼロ以外の値に設定すると、サーバーの証明書のクライアントチェックが無効になります。クライアントの証明書または IP アドレスのサーバーチェックを無効にするには、libvirtd 設定を変更する必要があります。	no_verify=1
no_tty	ssh	0 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 がパスワードを要求できなくなります。ターミナルへのアクセスがない場合は、これを使用します。	no_tty=1

18.2. SSH を使用したリモート管理

ssh パッケージは、暗号化したネットワークプロトコルを提供し、管理機能をリモート仮想サーバーに安全に送信できます。以下の方法は、**SSH** 接続上でセキュアにトンネリングされた **libvirt** 管理接続を使用して、リモートマシンを管理します。すべての認証は、**SSH**の公開鍵暗号と、ローカルの**SSH**エージェントが収集したパスワードまたはパスフレーズを使用して行われます。さらに、各ゲストの **VNC** コンソールは、**SSH** を介してトンネリングされます。

SSH を使用して仮想マシンをリモートで管理する場合は、以下の問題に注意してください。

- 仮想マシンを管理するには、リモートマシンへの root ログインアクセスが必要です。
- 初期接続の設定プロセスが遅くなる可能性があります。
- すべてのホストまたはゲストで、ユーザーのキーを取り消すための標準的な方法または簡単な方法はありません。
- **SSH** は、リモートマシンの数が多いと適切にスケーリングされません。



注記

Red Hat Virtualization により、多数の仮想マシンのリモート管理が可能になります。詳細は、Red Hat Virtualization 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です。

- openssh
- openssh-askpass
- openssh-clients
- openssh-server

virt-manager 用にパスワードを使用しないまたはパスワードを使用したSSH アクセスの設定

以下の手順は、ゼロから開始しており、**SSH** キーが設定されていないことを前提としています。SSH 鍵を設定して別のシステムにコピーしている場合は、この手順をスキップできます。

重要

SSH 鍵はユーザーに依存するため、所有者のみが使用できます。キーの所有者は、キーを生成したユーザーです。キーは、複数のユーザーで共有できない場合があります。

リモートホストコンピューターに接続するには、キーを所有するユーザーが**virt-manager** を実行する必要があります。つまり、リモートシステムが root 以外のユーザーにより管理されている場合は、**virt-manager** を非特権モードで実行する必要があります。リモートシステムがローカルの root ユーザーで管理されている場合は、SSH 鍵を root が作成して所有する必要があります。

ローカルホストは、**virt-manager** を使用する非特権ユーザーとして管理できません。

1. **オプション: ユーザーの変更**
   必要に応じてユーザーを変更します。この例では、ローカルの root ユーザーを使用して、その他のホストとローカルホストをリモートで管理します。

   ```
   $ su -
   ```

2. **SSH 鍵ペアの生成**
   virt-manager が使用されているマシンで公開鍵ペアを生成します。この例では、~/**.ssh**/ ディレクトリーのデフォルトの鍵の場所を使用します。

```
# ssh-keygen -t rsa
```

3. **リモートホストへの鍵のコピー**
   パスワードなしのリモートログイン、またはパスフレーズを使用したリモートログインでは、マネージドシステムに SSH 鍵を配布する必要があります。**ssh-copy-id** コマンドを使用して、指定し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:
```

   その後、マシンにログインして、**.ssh/authorized_keys** ファイルを確認して、予期しない鍵が追加されていないことを確認します。

```
ssh root@host2.example.com
```

   必要に応じて、その他のシステムに対して繰り返します。

4. **オプション: パスフレーズを ssh-agent に追加します。**
   必要に応じて、SSH 鍵のパスフレーズを**ssh-agent**に追加します。ローカルホストで次のコマンドを使用して、パスフレーズ (存在する場合) を追加し、パスワードなしのログインを有効にします。

```
# ssh-add ~/.ssh/id_rsa
```

   ssh-agent が実行していないと、このコマンドの実行に失敗します。エラーや競合を回避するには、SSH パラメーターが正しく設定されていることを確認してください。詳細は、『Red Hat Enterprise System Administration Guide』 を参照してください。

libvirt デーモン (libvirtd)

libvirt デーモンは、仮想マシンを管理するインターフェイスを提供します。この方法で管理するすべてのリモートホストに、**libvirtd** デーモンをインストールして実行している。

```
$ ssh root@somehost
# systemctl enable libvirtd.service
# systemctl start libvirtd.service
```

libvirtd および **SSH** を設定したら、仮想マシンにリモートでアクセスして管理できるようになります。この時点で、**VNC** でゲストにアクセスできるようになるはずです。

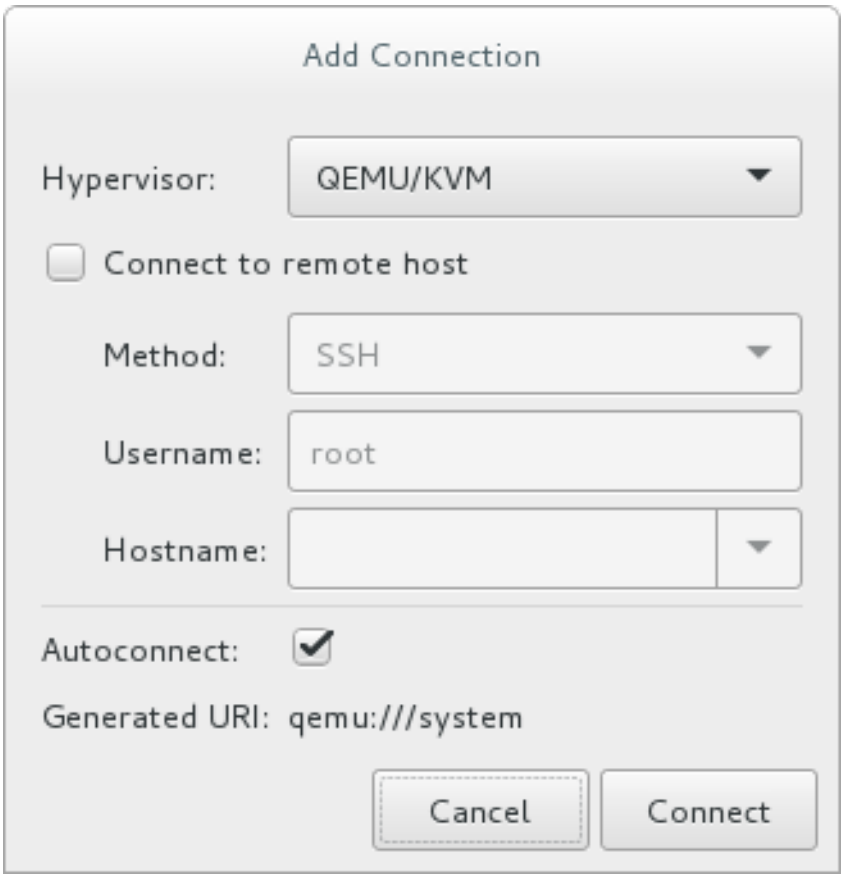
virt-manager を使用したリモートホストへのアクセス

リモートホストは、**virt-manager** GUI ツールで管理できます。SSH 鍵は、パスワードを使用しないログインを機能させるために、**virt-manager** を実行しているユーザーに属している必要があります。

1. **virt-manager** を起動します。

2. **File ⇒ Add Connection** メニューを開きます。

図18.1 Add Connection メニュー



3. ドロップダウンメニューを使用してハイパーバイザータイプを選択し、**Connect to remote host** チェックボックスをクリックして Connection **Method** (この場合は Remote tunnel over SSH) を開き、**User name** と **Hostname** を入力して **Connect** をクリックします。

18.3. TLS および SSL を使用したリモート管理

TLS および SSL プロトコルを使用して、仮想マシンを管理できます。TLS および SSL はスケーラビリティーは高くなりますが、SSH よりも複雑になります (「SSH を使用したリモート管理」 を参照)。TLS と SSL は、Web ブラウザーでセキュアな接続に使用されるものと同じ技術です。**libvirt** 管理接続は受信接続の TCP ポートを開きます。これは、x509 証明書に基づいて安全に暗号化および認証されます。以下の手順では、TLS および SSL 管理用の認証証明書を作成してデプロイする方法を説明します。

手順18.1 TLS 管理用の認証局 (CA) 鍵の作成

1. 始める前に、**gnutls-utils** がインストールされていることを確認します。インストールされていない場合は、インストールします。

```
# yum install gnutls-utils
```

2. 次のコマンドを使用して、秘密鍵を生成します。

```
# certtool --generate-privkey > cakey.pem
```

3. 鍵が生成されたら、鍵が自己署名できるように、署名ファイルを作成します。作成するには、署名の詳細が含まれるファイルを作成し、**ca.info** という名前を付けます。このファイルには、以下の内容を記述する必要があります。

```
cn = Name of your organization
ca
cert_signing_key
```

4. 以下のコマンドを使用して自己署名キーを生成します。

```
# certtool --generate-self-signed --load-privkey cakey.pem --template ca.info --outfile
cacert.pem
```

ファイルを生成したら、**rm** コマンドを使用して、**ca.info** ファイルを削除できます。生成プロセスの結果として生成されるファイルの名前は **cacert.pem** です。このファイルは公開鍵 (証明書) です。読み込んだ**cakey.pem**は秘密鍵です。セキュリティー上の理由から、このファイルはプライベートに保管し、共有領域には置かないでください。

5. **/etc/pki/CA/cacert.pem** ディレクトリー内のすべてのクライアントおよびサーバーに **cacert.pem** CA 証明書ファイルをインストールして、CA が発行した証明書が信頼でき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
```

これは、CA の設定に必要なものです。クライアントとサーバーの証明書を発行するために必要となるため、CA の秘密鍵を安全に保持します。

手順18.2 サーバー証明書の発行

この手順では、サーバーのホスト名に設定されている X.509 Common Name (CN) フィールドを使用して証明書を発行する方法を説明します。CN は、クライアントがサーバーへの接続に使用するホスト名と一致する必要があります。この例では、クライアントは URI: **qemu://*mycommonname*/system** を使用してサーバーに接続するため、CN 項目は、この例では "mycommoname" と同じになります。

1. サーバーの秘密鍵を作成します。

```
# certtool --generate-privkey > serverkey.pem
```

2. 最初に **server.info** というテンプレートファイルを作成して、CA の秘密鍵の署名を生成します。CN がサーバーのホスト名と同じに設定されていることを確認します。

```
organization = Name of your organization
cn = mycommonname
tls_www_server
encryption_key
signing_key
```

3. 証明書を作成します。

```
# certtool --generate-certificate --load-privkey serverkey.pem --load-ca-certificate cacert.pem
--load-ca-privkey cakey.pem \ --template server.info --outfile servercert.pem
```

これにより、2 つのファイルが生成されます。

- serverkey.pem - サーバーの秘密鍵
- servercert.pem - サーバーの公開鍵

4. 秘密鍵の場所は、秘密にしておきます。ファイルの内容を表示するには、次のコマンドを使用します。

```
# certtool -i --infile servercert.pem
```

このファイルを開く場合、**CN=** パラメーターは先に設定した CN と同じでなければなりません。(例:**mycommonname**)

5. 次の場所にある 2 つのファイルをインストールします。

- **serverkey.pem** - サーバーの秘密鍵。このファイルは、**/etc/pki/libvirt/private/serverkey.pem** に置きます。
- **servercert.pem** - サーバーの証明書。サーバーの **/etc/pki/libvirt/servercert.pem** にインストールします。

手順18.3 クライアント証明書の発行

1. すべてのクライアント (つまり、**virt-manager** などの libvirt にリンクしたプログラム) に、適切な名前を設定した X.509 識別名 (DN) フィールドを持つ証明書を発行する必要があります。これは、企業レベルで決定する必要があります。

   たとえば、以下の情報が使用されます。

```
C=USA,ST=North Carolina,L=Raleigh,O=Red Hat,CN=name_of_client
```

2. 秘密鍵を作成します。

```
# certtool --generate-privkey > clientkey.pem
```

3. 最初に **client.info** というテンプレートファイルを作成して、CA の秘密鍵の署名を生成します。ファイルには以下の内容が含まれている必要があります (フィールドは、お住まいの地域/場所に合わせてカスタマイズする必要があります)。

```
country = USA
state = North Carolina
locality = Raleigh
organization = Red Hat
cn = client1
tls_www_client
encryption_key
signing_key
```

4. 次のコマンドを使用して、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. クライアントマシンに証明書を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
```

18.4. VNC サーバーの設定

仮想ネットワークコンピューティング (VNC) を使用して、ホストとゲストマシン間でグラフィカルデスクトップ共有を設定するには、接続するゲストで VNC サーバーを設定する必要があります。これを行うには、ゲストの XML ファイルの **devices** 要素で、グラフィックタイプとして VNC を指定する必要があります。詳細は、「グラフィカルフレームバッファー」 を参照してください。

VNC サーバーに接続するには、**virt-viewer** ユーティリティーまたは **virt-manager** インターフェイスを使用します。

18.5. NSS を使用した仮想マシンのリモート管理の強化

Red Hat Enterprise Linux 7.3 以降では、libvirt Network Security Services (NSS) モジュールを使用すると、SSH、TLS、SSL、およびその他のリモートログインサービスを使用したゲストへの接続が容易になります。また、**ping** などのホスト名変換を使用するユーティリティーも利用できます。

この機能を使用するには、libvirt-nss パッケージをインストールします。

```
# yum install libvirt-nss
```

注記

libvirt-nss のインストールに失敗した場合は、Red Hat Enterprise Linux 用の **Optional** リポジトリーが有効になっていることを確認します。手順は、System Administrator's Guide を参照してください。

最後に、以下のように、**/etc/nsswitch.conf** ファイルーの **hosts** 行に **libvirt_guest** を追加して、モジュールを有効にします。

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

モジュールが **hosts** 行にリスト表示されている順序によって、指定されたリモートゲストを見つけるためにこれらのモジュールが参照される順序が決まります。その結果、libvirt の NSS モジュールが、ホストのドメイン名を IP アドレスに変換するモジュールに追加されました。たとえば、静的 IP アドレスを設定せずに、ゲストの名前のみを使用して、NAT モードでリモートゲストに接続できます。

```
# ssh root@guestname
root@guestname's password:
Last login: Thu Aug 10 09:12:31 2017 from 192.168.122.1
[root@guestname ~]#
```

注記

ゲストの名前がわからない場合は、**virsh list --all** コマンドを使用して取得できます。

第19章 VIRTUAL MACHINE MANAGER を使用したゲストの管理 (VIRT-MANAGER)

本章では、Virtual Machine Manager (**virt-manager**) のウィンドウ、ダイアログボックス、およびさまざまな GUI コントロール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ホストシステム上のハイパーバイザーとゲストのグラフィカルビューを提供します。**virt-manager** は、次のような仮想化管理タスクを実行できます。

- 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
- 運用パフォーマンスの監視
- ゲストの保存、復元、一時停止、再開、およびシャットダウンと起動
- テキストコンソールや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およびオフラインマイグレーション。

重要

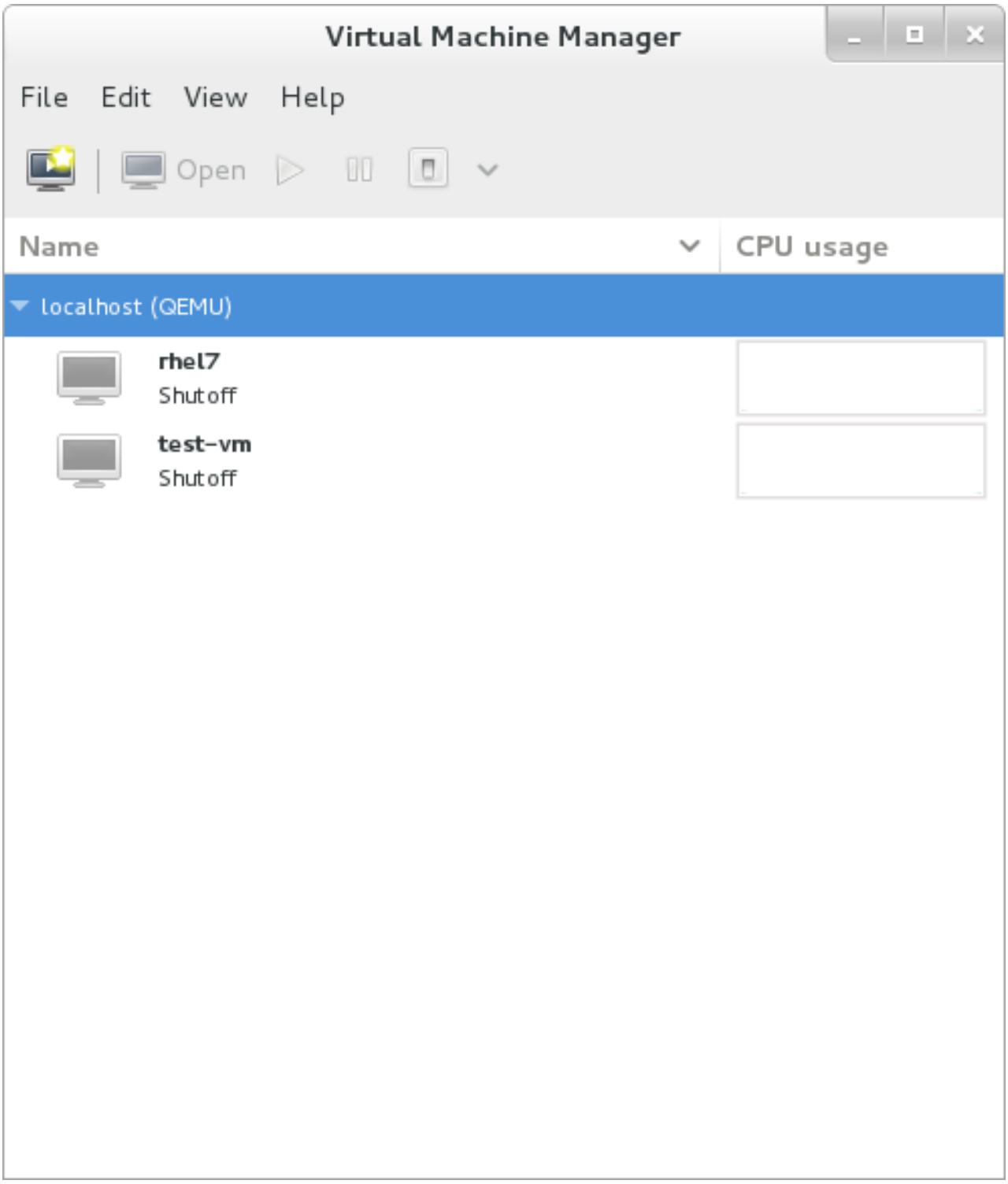
使用しているユーザーに気をつける必要があります。あるユーザーでゲスト仮想マシンを作成した場合、別のユーザーでそのユーザーに関する情報を取得することはできません。virt-manager で仮想マシンを作成する場合は、これが特に重要です。デフォルトのユーザーは、特に指定がない限り、この場合は root になります。**virsh list --all** コマンドを使用して仮想マシンのリストを表示できない場合は、仮想マシンの作成に使用したユーザーとは別のユーザーを使用してコマンドを実行している可能性が最も高いです。

19.1. VIRT-MANAGER の起動

virt-manager セッションを開始するには、Applications メニューを開き、System Tools メニューの Virtual Machine Manager (**virt-manager**) を選択します。

virt-manager のメインウィンドウが表示されます。

図19.1 virt-manager の起動



または、以下のコマンドで示すように、**ssh** を使用して、**virt-manager** をリモートで起動できます。

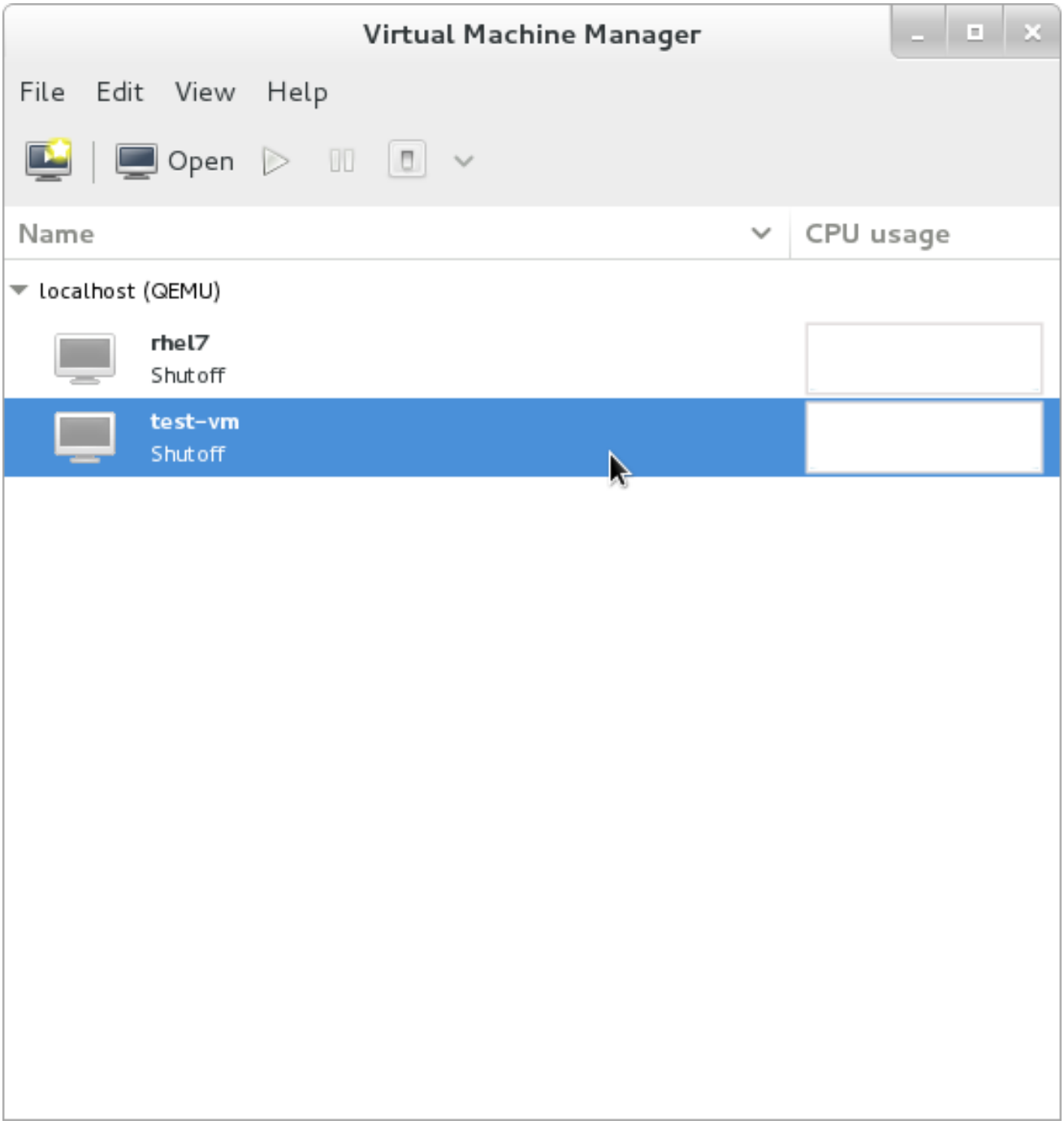
```
# ssh -X host's address
[remotehost]# virt-manager
```

ssh を使用した仮想マシンおよびホストの管理については、「SSH を使用したリモート管理」 を参照してください。

19.2. VIRTUAL MACHINE MANAGER のメインウィンドウ

このメインウィンドウには、実行中のゲストと、ゲストが使用するリソースがすべて表示されます。ゲストの名前をダブルクリックして、ゲストを選択します。

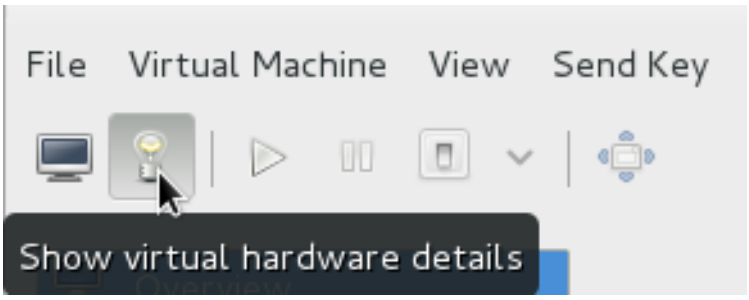
図19.2 Virtual Machine Manager のメインウィンドウ



19.3. 仮想ハードウェアの詳細ウィンドウ

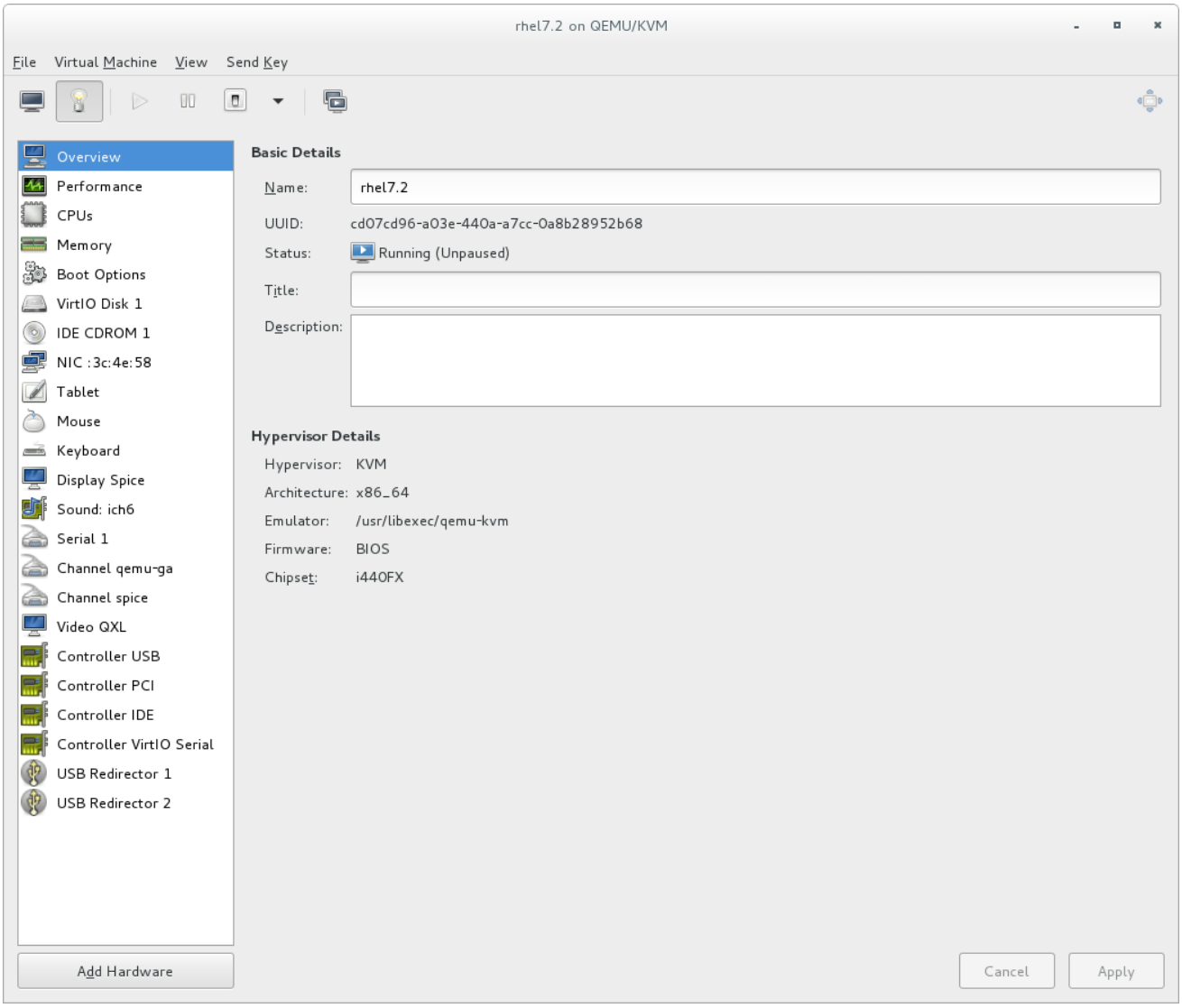
仮想ハードウェアの詳細ウィンドウには、ゲストに設定された仮想ハードウェアに関する情報が表示されます。仮想ハードウェアリソースは、このウィンドウで追加、削除、および変更できます。仮想ハードウェアの詳細ウィンドウにアクセスするには、ツールバーのアイコンをクリックします。

図19.3 仮想ハードウェアの詳細アイコン



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19.4 仮想ハードウェアの詳細ウィンドウ



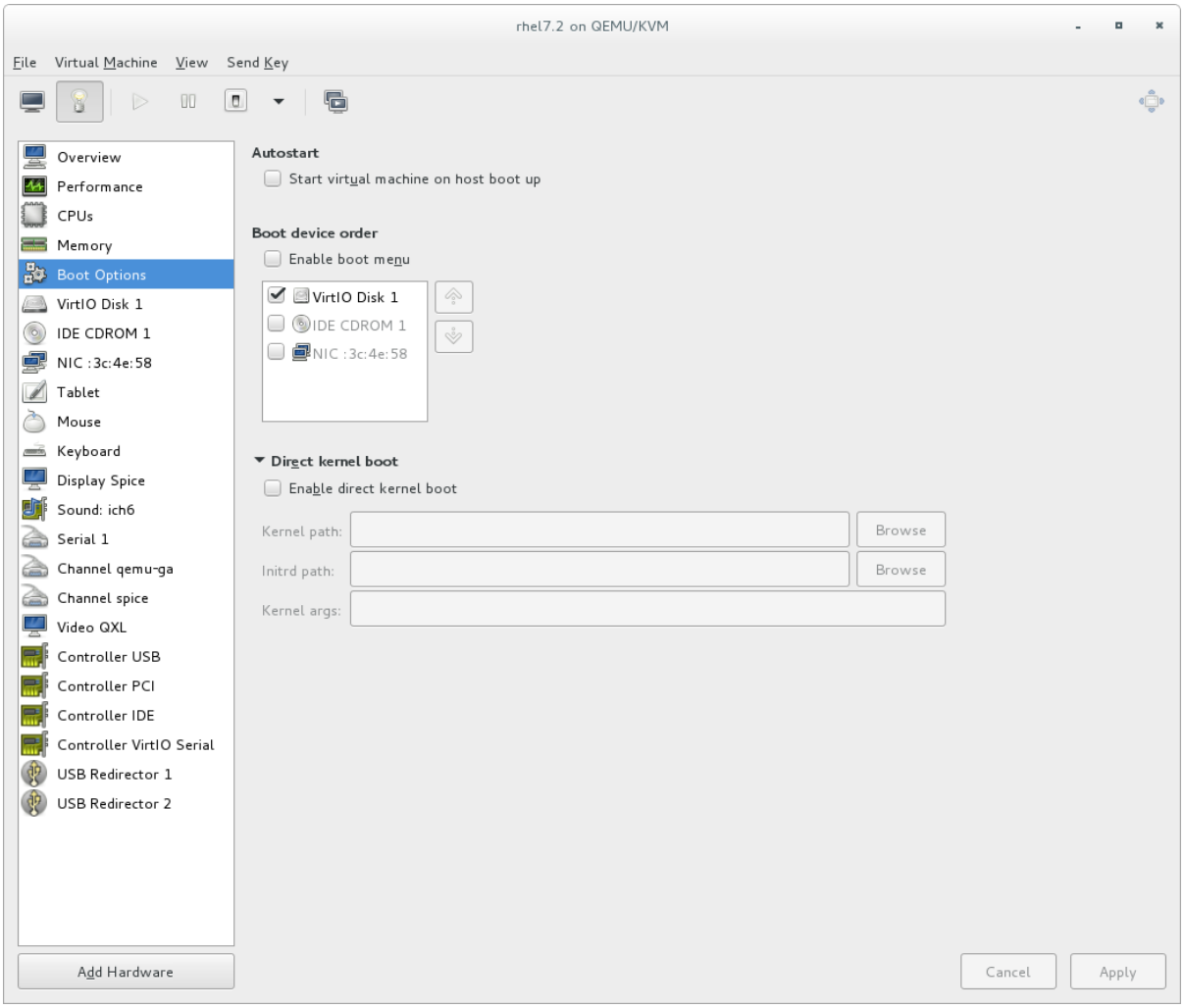
19.3.1. ゲスト仮想マシンへの起動オプションの適用

virt-manager を使用して、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するまで有効になりません。変更を行う前に仮想マシンの電源を切るか、後でマシンを再起動できます。これらのオプションのいずれかを実行しないと、ゲストが次回再起動したときに変更が行われます。

手順19.1 起動オプションの設定

1. 仮想マシンマネージャーのEdit メニューから、Virtual Machine Detailsを選択します。
2. サイドパネルから Boot Options を選択し、次のいずれかまたはすべての手順を完了します。
   a. ホストの物理マシンが起動するたびにこのゲスト仮想マシンを起動するように指定するには、Autostart チェックボックスを選択します。
   b. ゲスト仮想マシンの起動順序を指定するには、Enable boot menu のチェックボックスを選択します。これを選択してから、起動元のデバイスを選択し、矢印キーを使用して、ゲスト仮想マシンが起動時に使用する順序を変更します。
   c. Linux カーネルから直接起動する場合は、Direct kernel boot をデプロイメントします。Kernel path、Initrd path、および使用するKernel arguments を入力します。
3. Apply をクリックします。

図19.5 起動オプションの設定



19.3.2. ゲスト仮想マシンへの USB デバイスの接続

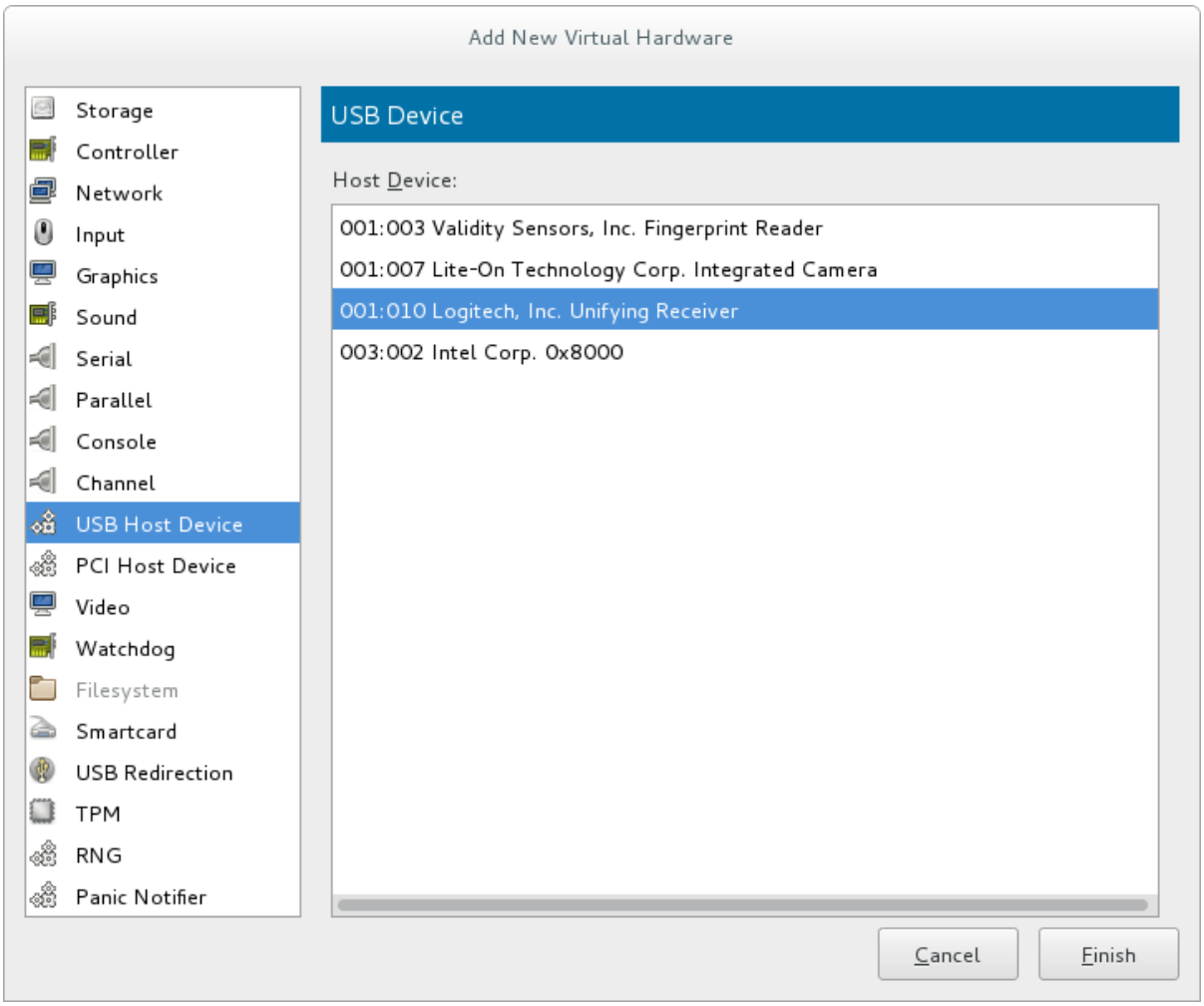
注記

USB デバイスをゲスト仮想マシンに接続するには、まずホストの物理マシンに接続し、デバイスが機能していることを確認します。ゲストを実行している場合は、続行する前にシャットダウンする必要があります。

手順19.2 Virt-Manager を使用した USB デバイスの接続

1. ゲスト仮想マシンの Virtual Machine Details 画面を開きます。
2. **Add Hardware** をクリックします。
3. **Add New Virtual Hardware** ポップアップで、**USB Host Device** を選択し、リストから接続するデバイスを選択して、**Finish** をクリックします。

   図19.6 USB デバイスの追加



4. ゲスト仮想マシンで USB デバイスを使用するには、ゲスト仮想マシンを起動します。

19.3.3. USB リダイレクト

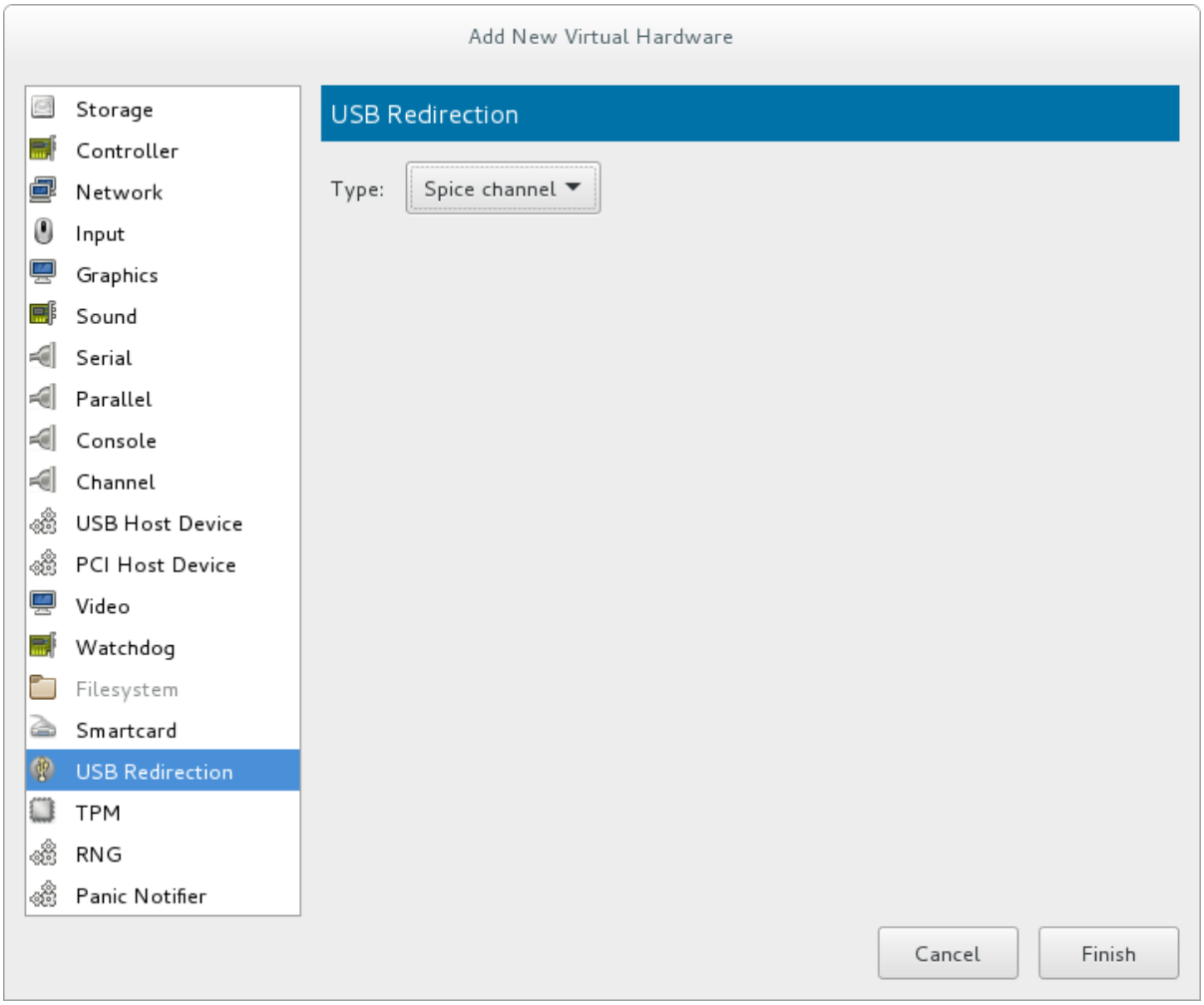
USB リダイレクトは、データセンターで実行しているホストの物理マシンがある場合に最も適しています。ユーザーは、ローカルマシンまたはシンクライアントからゲスト仮想マシンに接続します。このローカルマシンには SPICE クライアントがあります。ユーザーは、任意の USB デバイスをシンクライアントに接続できます。SPICE クライアントは、デバイスをデータセンターのホスト物理マシンにリダイレクトするため、シンクライアントで実行している仮想マシンで使用できます。

手順19.3 USB デバイスのリダイレクト

1. ゲスト仮想マシンの Virtual Machine Details 画面を開きます。
2. **Add Hardware** をクリックします。

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

図19.7 Add New Virtual Hardware ウィンドウ



4. **Virtual Machine** を開き、**Redirect USB device** を選択します。USB デバイスのリストが記載されたポップアップウィンドウが開きます。

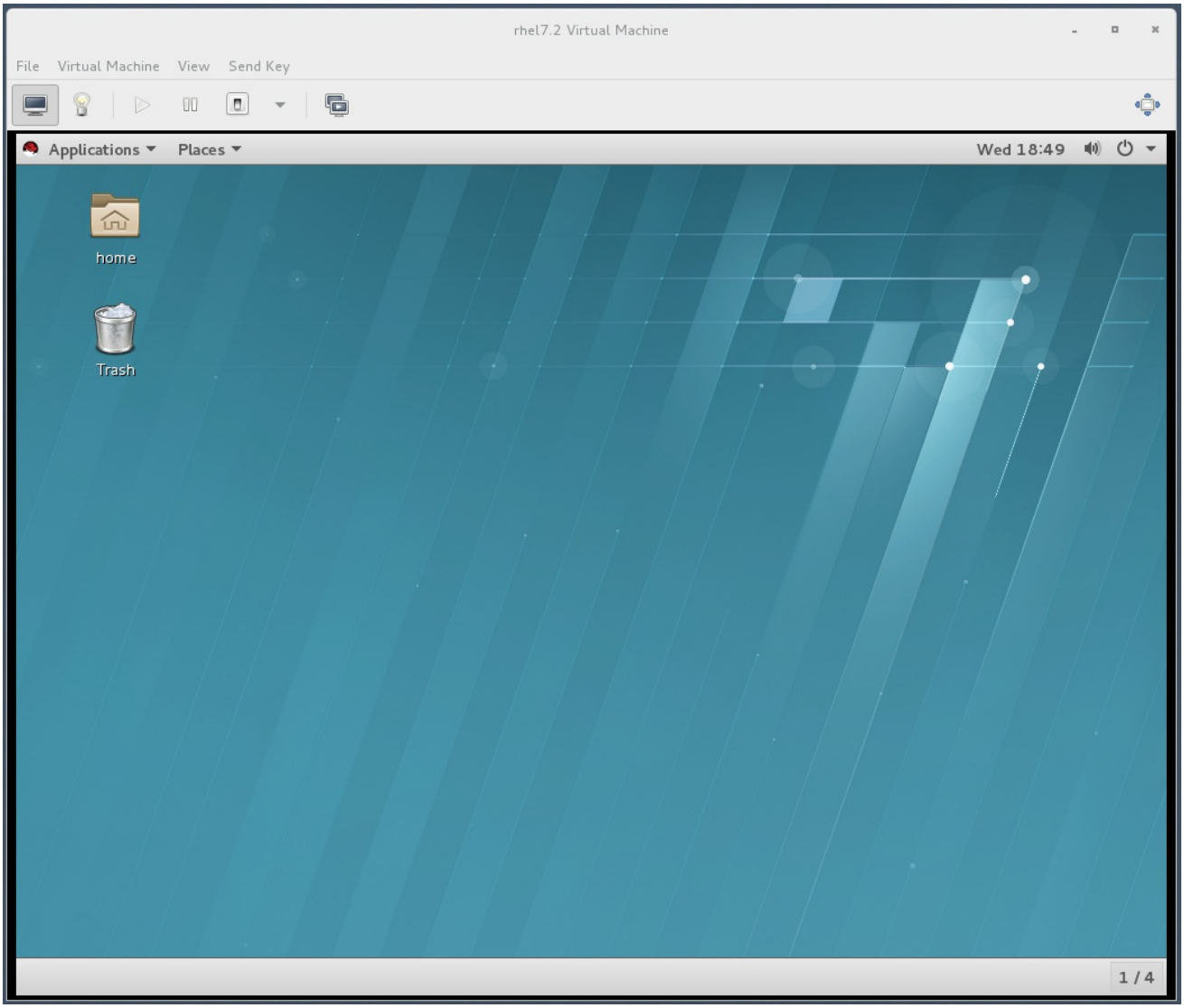
図19.8 USB デバイスを選択します。

5. チェックボックスをチェックして、リダイレクトする USB デバイスを選択し、**OK** をクリックします。

19.4. 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。ゲストは、複数のプロトコルを使用してグラフィカルフレームバッファーをエクスポートできます。**virt-manager** は、**VNC** および **SPICE** をサポートします。仮想マシンが認証を要求するように設定されている場合、仮想マシンのグラフィカルコンソールは、ディスプレイが表示される前にパスワードの入力を求めます。

図19.9 グラフィカルコンソールウィンドウ



注記

VNC は多くのセキュリティー専門家によって安全ではないと見なされますが、Red Hat Enterprise Linux の仮想化で VNC のセキュアな使用を可能にするためにいくつかの変更が加えられました。ゲストマシンは、ローカルホストのループバックアドレス (**127.0.0.1**) のみをリッスンします。これにより、ホストでシェル特権を持つユーザーのみが、VNC を介して virt-manager および仮想マシンにアクセスできるようになります。virt-manager は、他のパブリックネットワークインターフェイスをリッスンするように設定されており、別の方法を設定することもできますが、推奨されません。

リモート管理は、トラフィックを暗号化する SSH 経由のトンネルにより実行できます。VNC は、SSH 経由でのトンネルを使用せずにリモートからアクセスするように設定できますが、セキュリティー上の理由から推奨されません。ゲストをリモートで管理するには、18章 *ゲストのリモート管理* の手順に従います。TLS は、ゲストおよびホストシステムを管理するためのエンタープライズレベルのセキュリティーを提供できます。

ローカルデスクトップは、ゲストマシンに送信しないようにキーの組み合わせ (Ctrl+Alt+F1 キーなど) を傍受できます。**Send key** メニューオプションを使用すると、これらのシーケンスを送信できます。ゲストマシンウィンドウで **Send key** メニューをクリックし、送信するキーシーケンスを選択します。また、このメニューから画面出力をキャプチャーすることもできます。

SPICE は、Red Hat Enterprise Linux で利用可能な VNC の代替手段です。

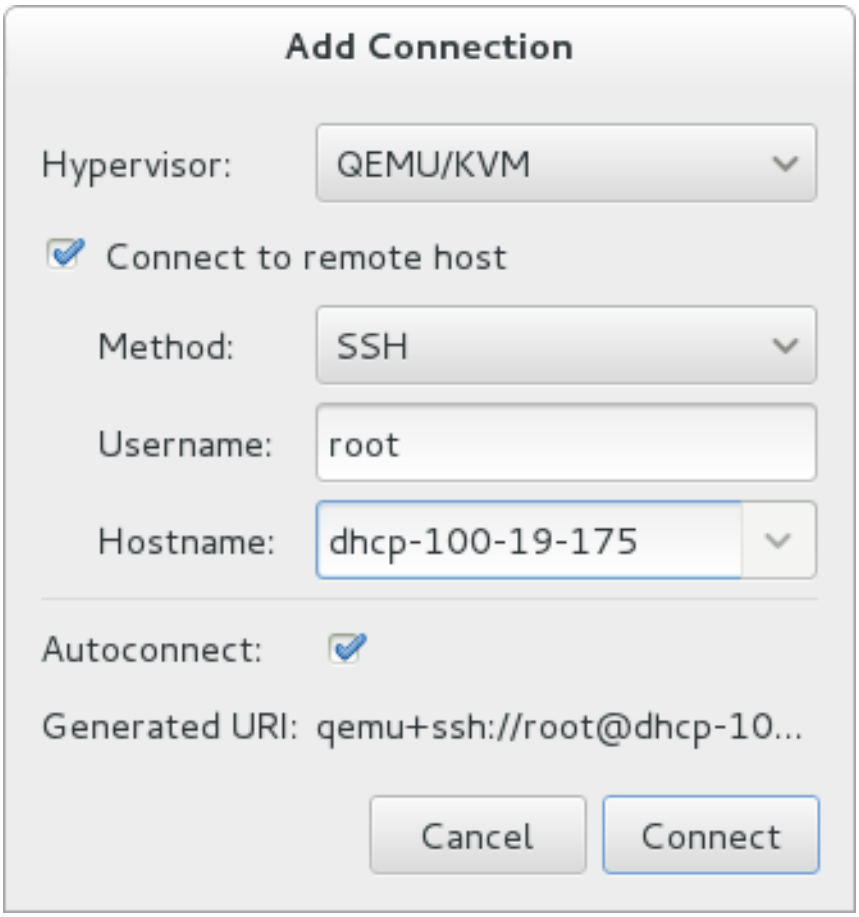
19.5. リモート接続の追加

この手順では、**virt-manager** を使用し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. 新規接続を作成するには、**File** メニューを開き、**Add Connection** メニューアイテムを選択します。

2. **Add Connection** 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 の場合は、システムで **QEMU/KVM** を選択します。ローカルシステムの **Local**、またはリモート接続オプションのいずれかを選択し、**Connect** をクリックします。この例では、SSH 経由でリモートトンネルを使用します。これは、デフォルトのインストールで機能します。リモート接続の設定の詳細は、18章 *ゲストのリモート管理* を参照してください。

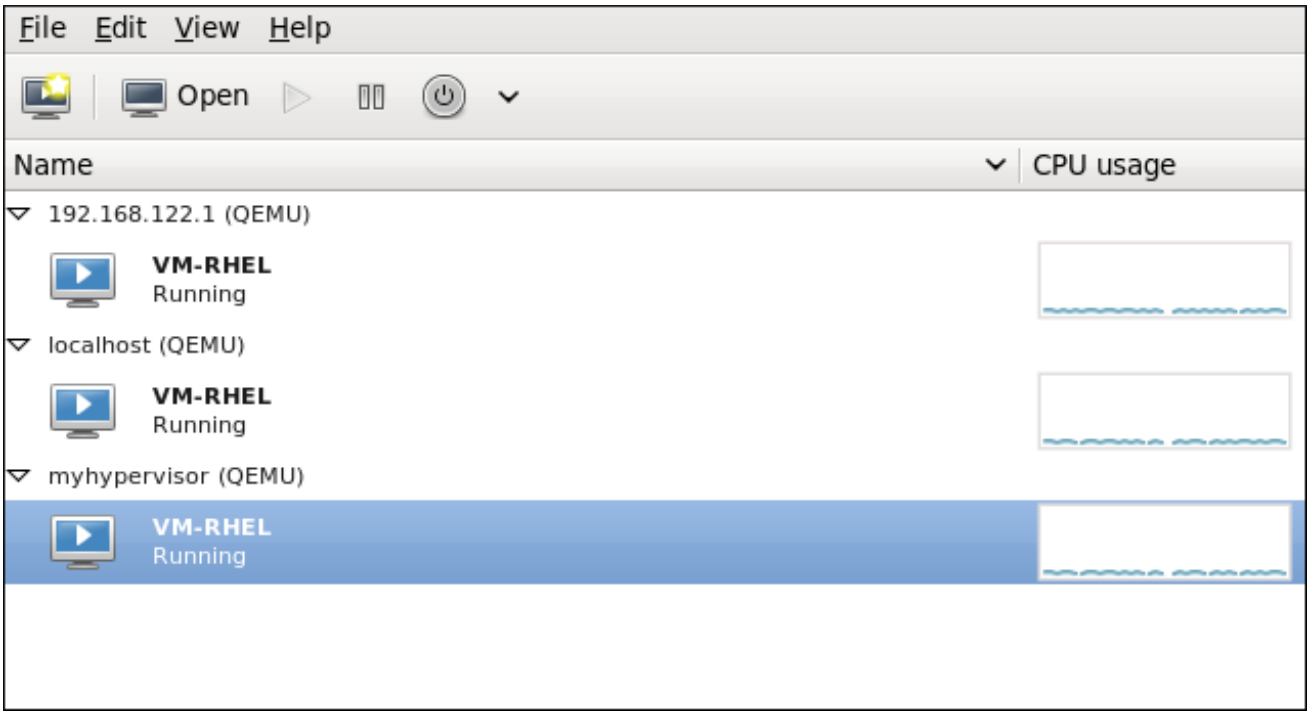
図19.10 接続の追加



3. プロンプトが表示されたら、選択したホストの root パスワードを入力します。

これで、リモートホストコンピューターが接続され、メイン**virt-manager** ウィンドウに表示されるようになります。

図19.11 メインの virt-manager ウィンドウのリモートホスト



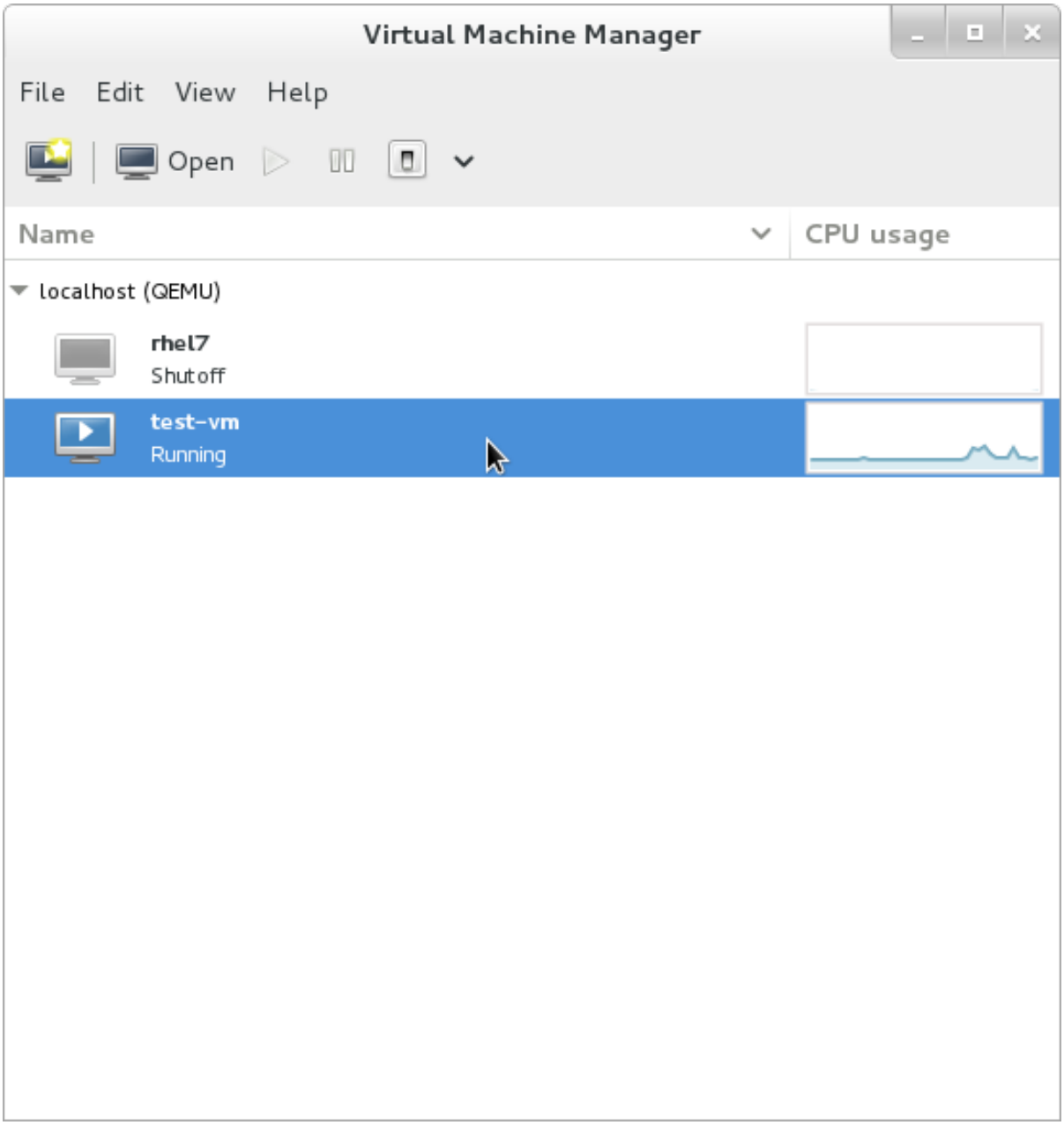
19.6. ゲストの詳細の表示

Virtual Machine Monitor 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には、次のコマンドを実行します。

1. Virtual Machine Manager 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19.12 表示する仮想マシンの選択

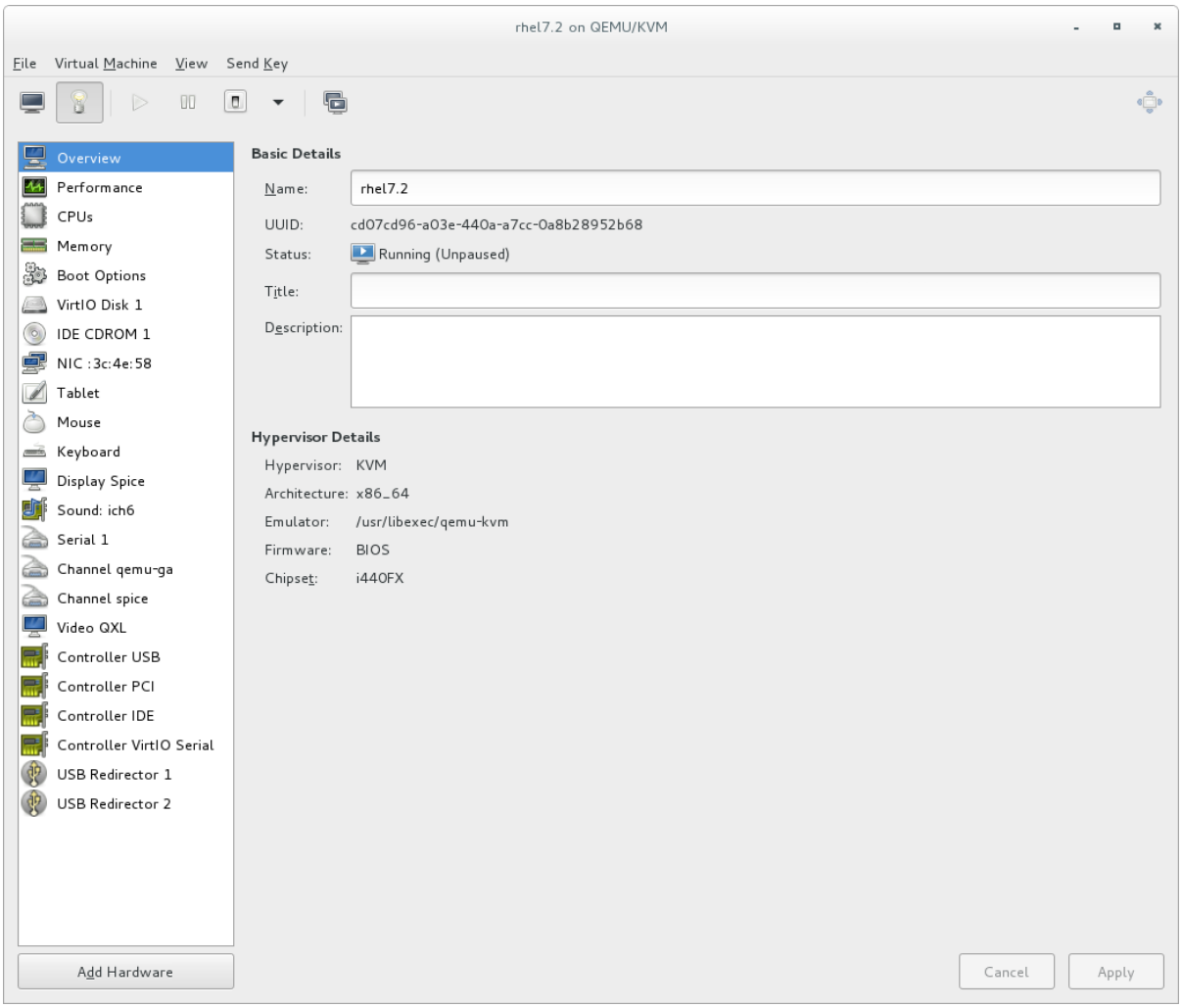


2. 仮想マシンマネージャーの**Edit** メニューから、**Virtual Machine Details** を選択します。

   仮想マシンの詳細ウィンドウが開くと、コンソールが表示される場合があります。これが発生した場合は、**View** をクリックし、**Details** を選択します。デフォルトでは、Overview ウィンドウが最初に開きます。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から **Overview** を選択します。

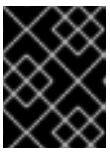
   Overview ビューには、ゲストの設定詳細の概要が表示されます。

図19.13 ゲストの詳細の概要の表示



3. 左側のナビゲーションペインから **CPU** を選択します。**CPU** ビューでは、現在のプロセッサー割り当てを表示または変更できます。

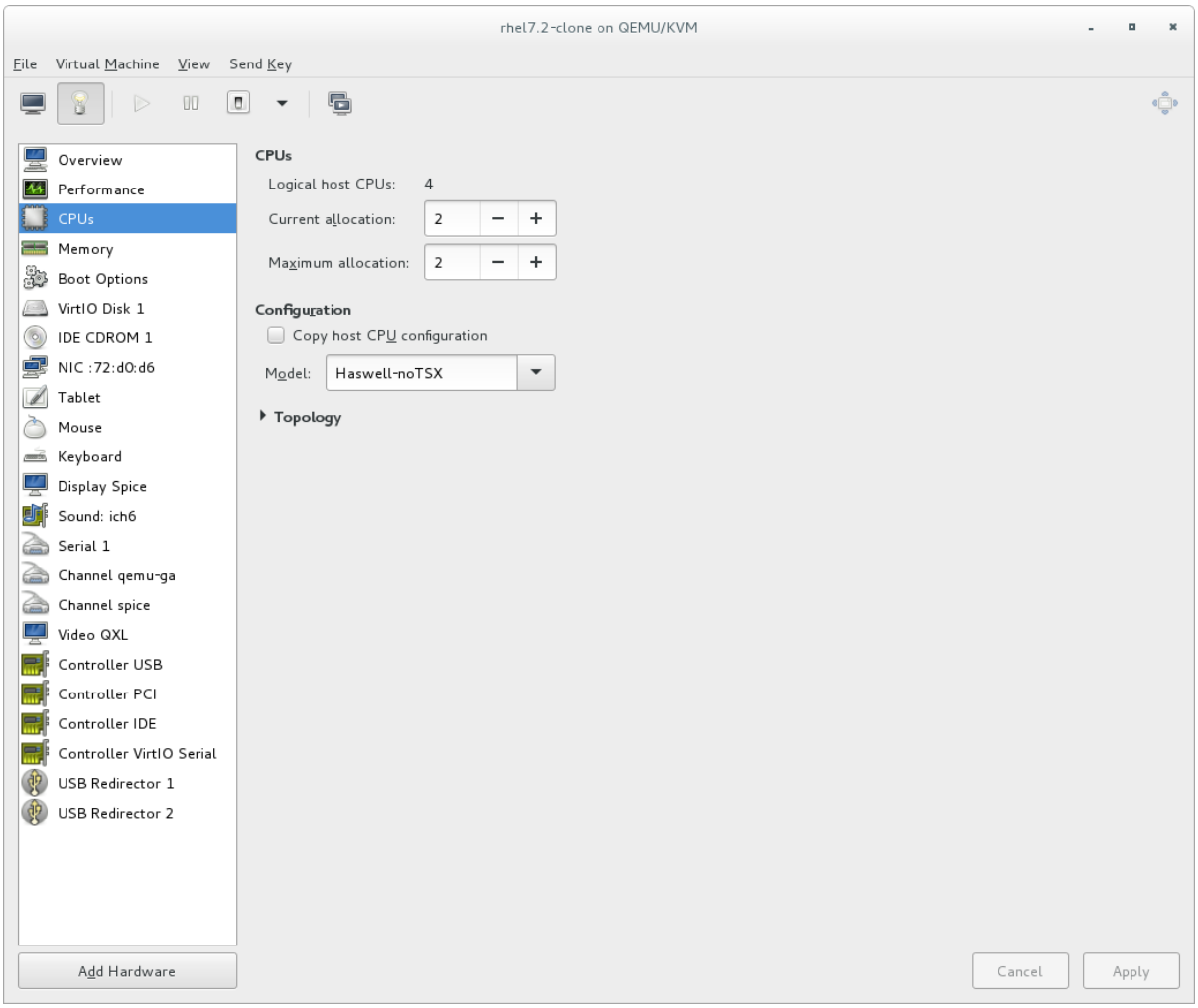
   仮想マシンの実行中に仮想 CPU (vCPU) の数を増やすこともできます。これは*hot plugging* と呼ばれます。



重要

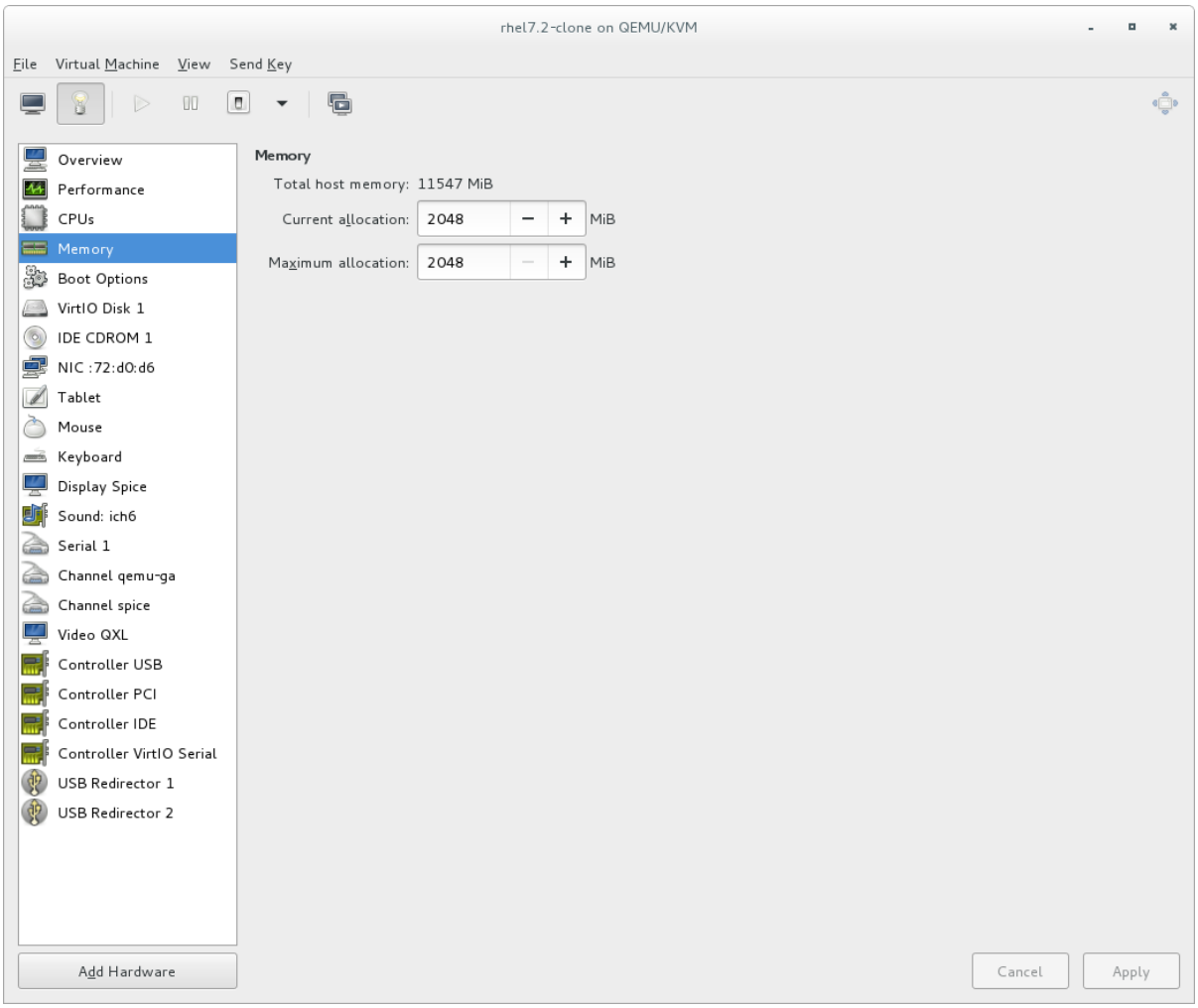
Hot unplugging vCPU は、Red Hat Enterprise Linux 7 ではサポートされていません。

図19.14 プロセッサー割り当てパネル



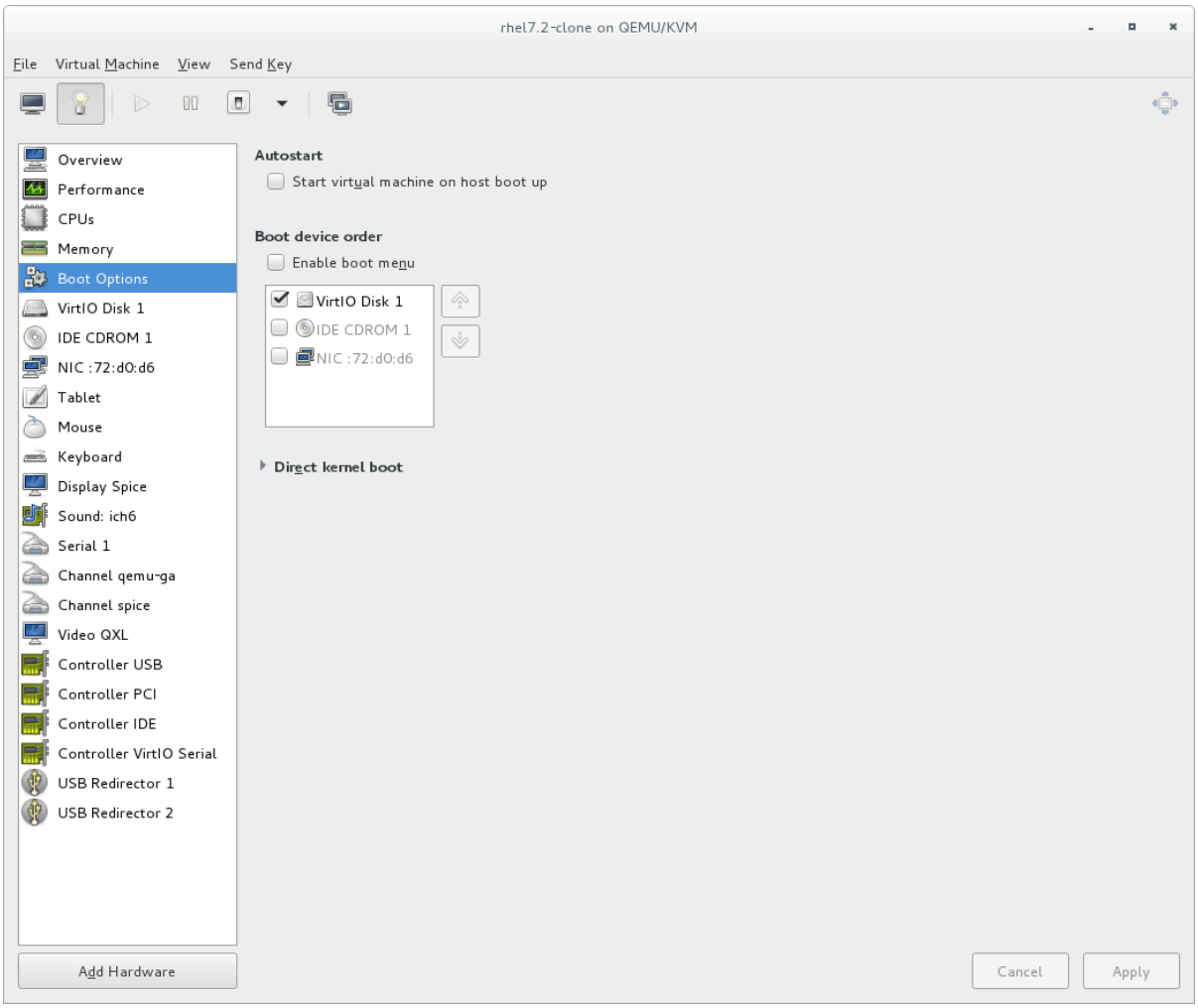
4. 左側のナビゲーションペインから **Memory** を選択します。**Memory** ビューでは、現在のメモリー割り当てを表示または変更できます。

図19.15 メモリー割り当ての表示



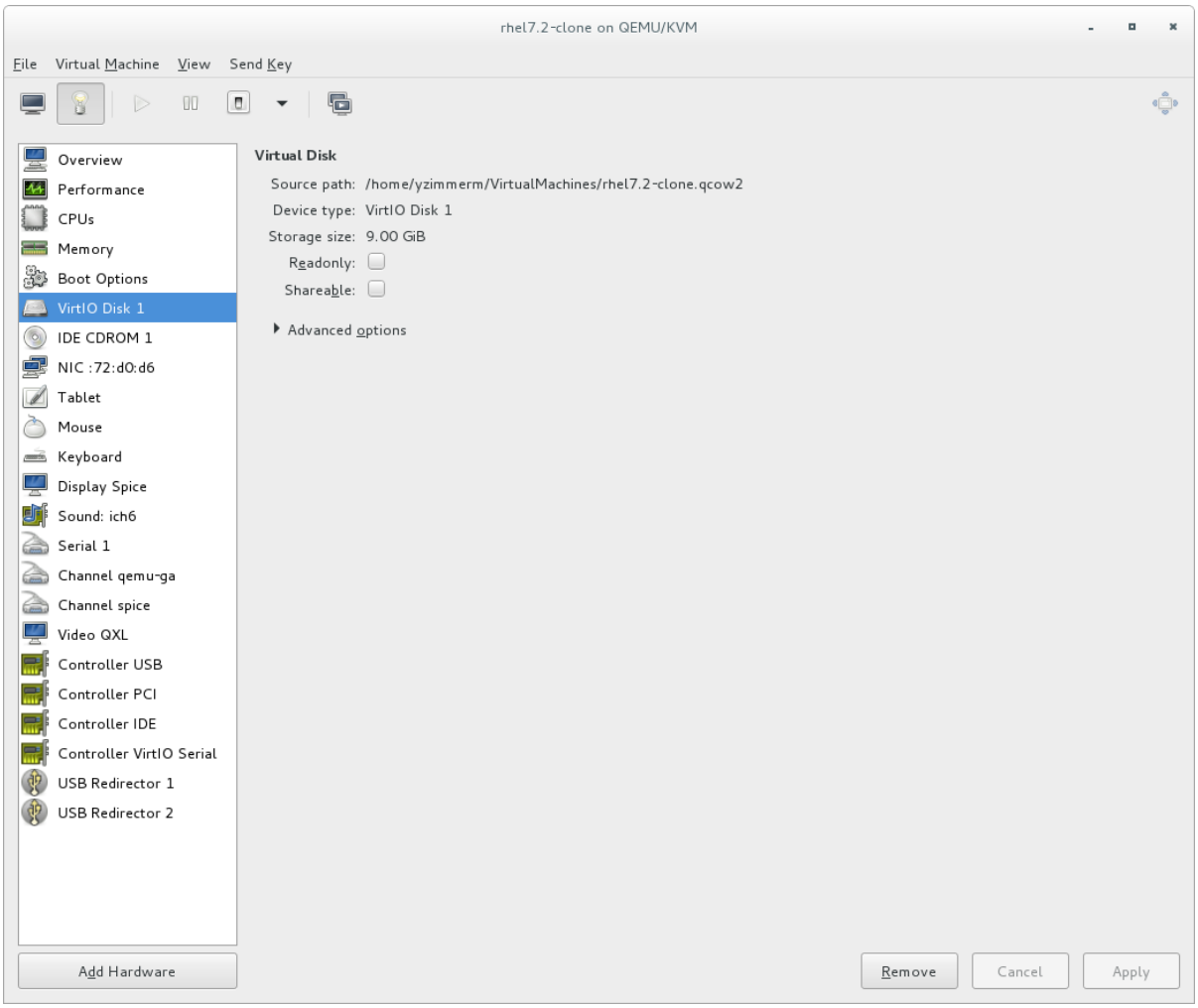
5. 左側のナビゲーションペインから **Boot Options** を選択します。**Boot Options** ビューを使用すると、ホストの起動時に仮想マシンが起動しているかどうかや、仮想マシンのブートデバイスの順序など、起動オプションを表示または変更できます。

図19.16 起動オプションの表示



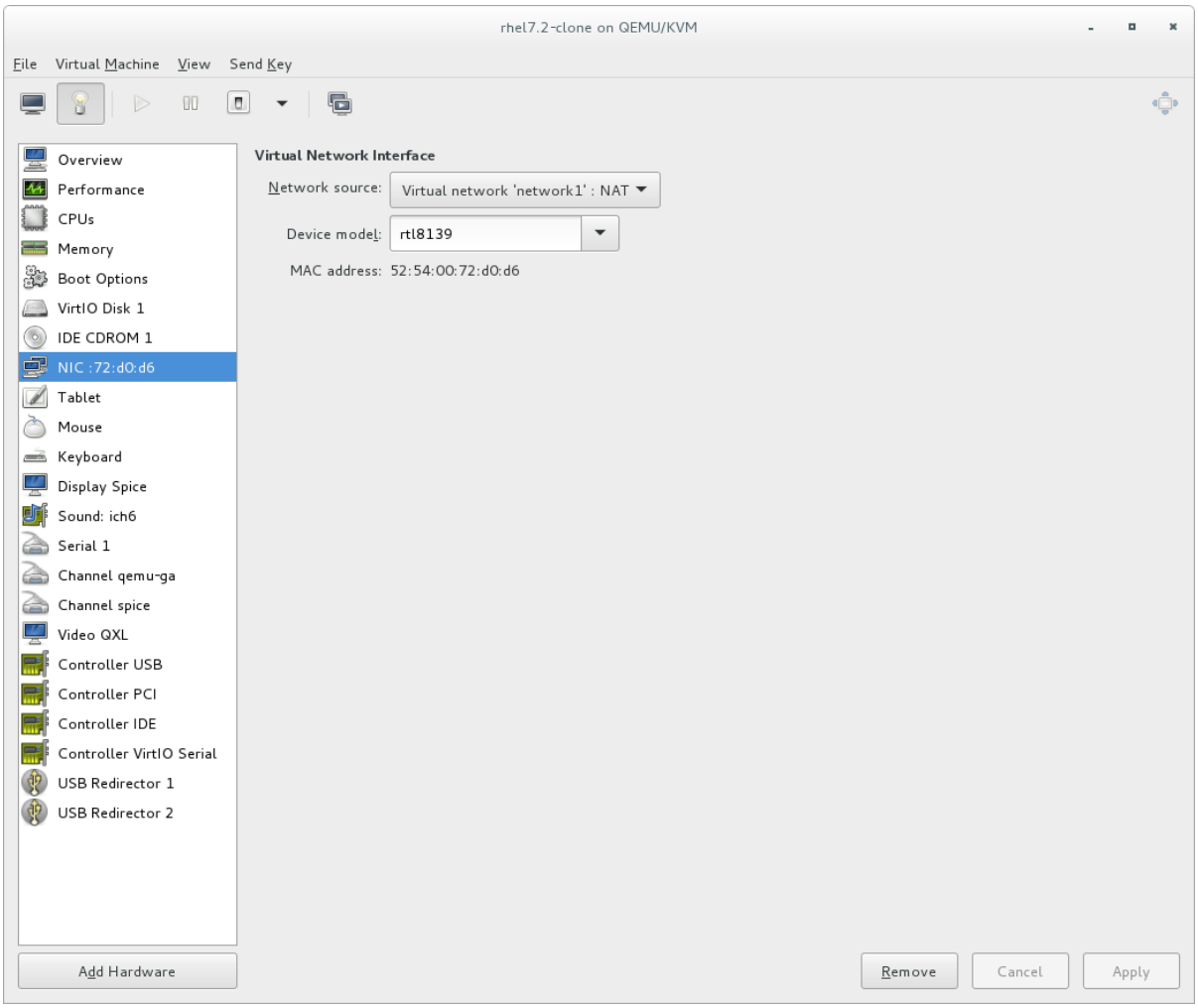
6. 仮想マシンに接続されている各仮想ディスクがナビゲーションペインに表示されます。仮想ディスクをクリックして、仮想ディスクを変更または削除します。

図19.17 ディスク設定の表示



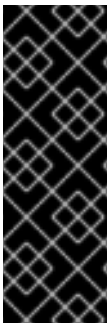
7. 仮想マシンに接続されている各仮想ネットワークインターフェイスがナビゲーションペインに表示されます。仮想ネットワークインターフェイスをクリックして、仮想ネットワークインターフェイスを変更または削除します。

図19.18 ネットワーク設定の表示



19.7. スナップショットの管理

virt-manager を使用すると、ゲスト *snapshots* を作成、実行、および削除できます。スナップショットは、ゲストのハードディスク、メモリー、およびデバイスの状態を 1 つの時点で保存したイメージです。スナップショットを作成したら、ゲストをスナップショットの設定にいつでも戻すことができます。

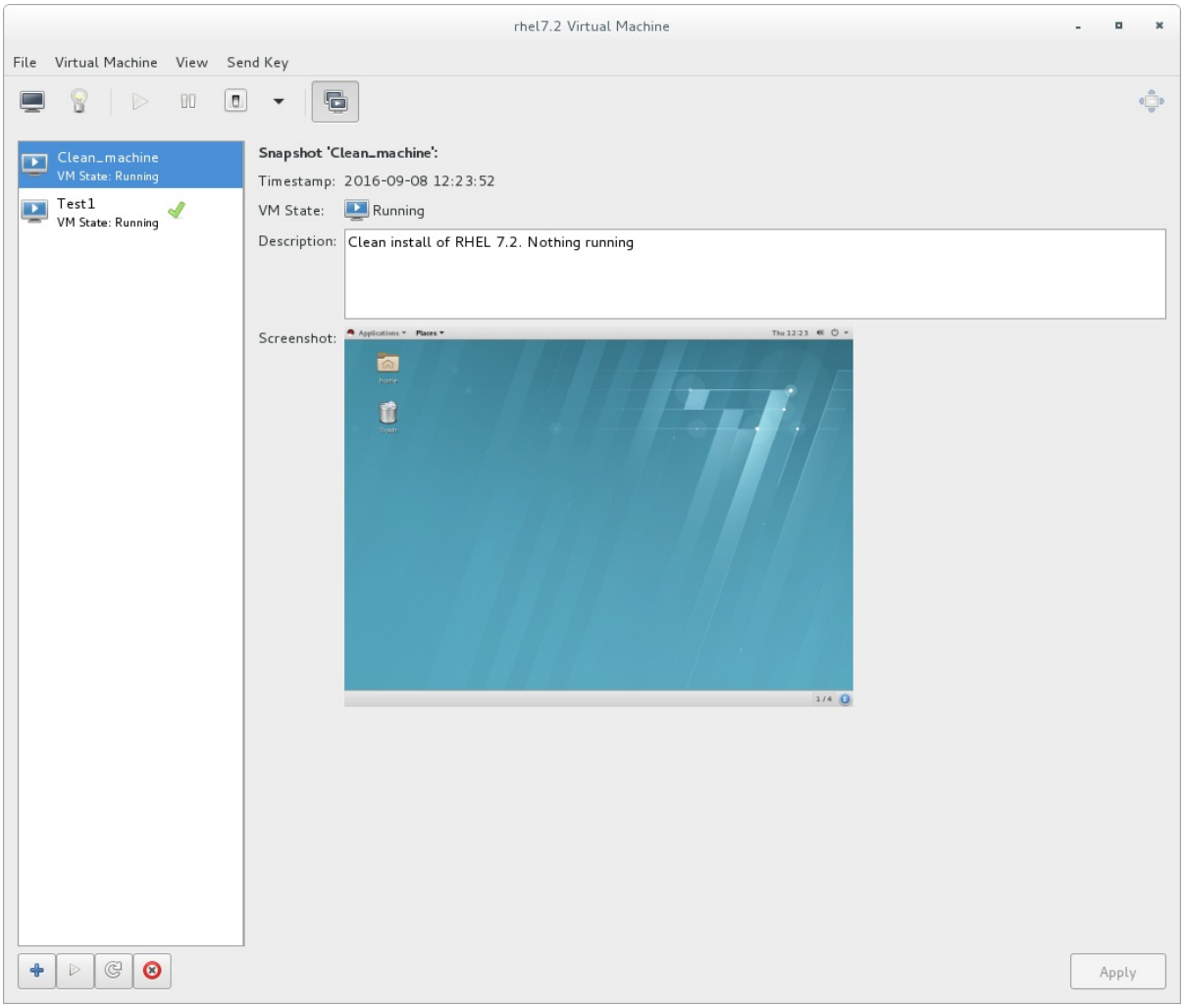


重要

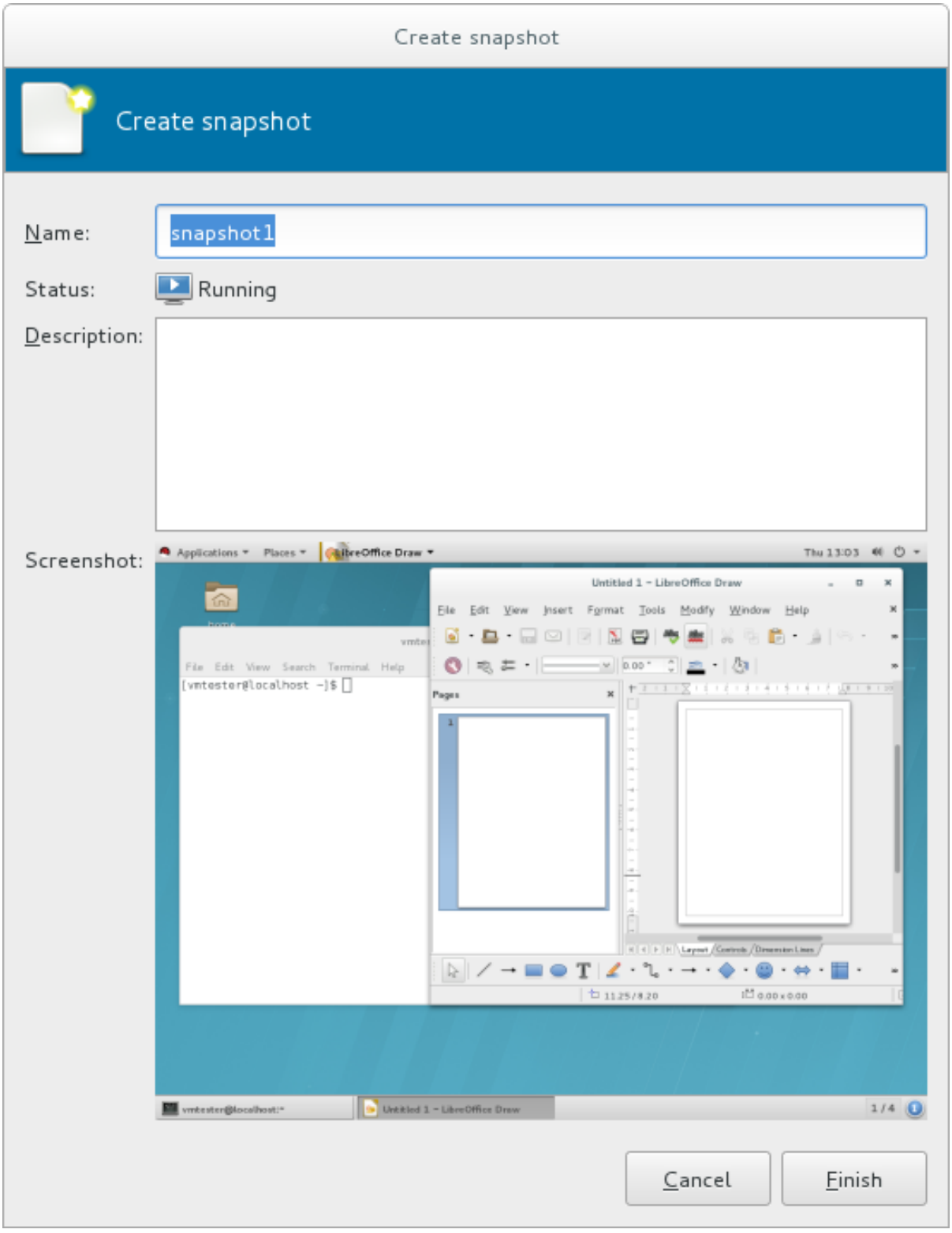
Red Hat は、外部スナップショットを使用することを推奨します。外部スナップショットは、他の仮想化ツールで処理する場合、より柔軟で信頼性が高いためです。ただし、現在のところ、**virt-manager** では外部スナップショットを作成できません。

外部スナップショットを作成する場合は、**--diskspec vda,snapshot=external** で **virsh snapshot-create-as** コマンドを実行します。詳細は、「libvirt を使用して外部スナップショットを作成するための回避策」 を参照してください。

- virt-manager でスナップショットを管理するには、ゲストコンソールの  をクリックして、スナップショット管理インターフェイスを開きます。



- 新しいスナップショットを作成する場合は、スナップショットリストの下にある  をクリックします。スナップショット作成インターフェイスで、スナップショットの名前と、必要に応じて説明を入力して、**Finish** をクリックします。



- ゲストをスナップショットの設定に戻すには、スナップショットを選択し、 をクリックします。

- 選択したスナップショットを削除するには、 をクリックします。

警告

仮想マシンの実行中にスナップショットを作成して読み込む (**ライブスナップショット** とも呼ばれる) ことは、qcow2 ディスクイメージでのみ対応しています。

詳細なスナップショット管理には、**virsh snapshot-create** コマンドを使用します。**virsh** でスナップショットを管理する方法は、「スナップショットの管理」 を参照してください。

第20章 VIRSH を使用したゲスト仮想マシンの管理

virsh は、ゲスト仮想マシンを管理するコマンドラインインターフェイスツールで、Red Hat Enterprise Linux 7 で仮想化を制御する主要な手段として機能します。**virsh** コマンドラインツールは、libvirt 管理 API に構築され、ゲスト仮想マシンの作成、デプロイ、および管理に使用できます。**virsh** ユーティリティーは、仮想化管理スクリプトを作成するために最適で、root 権限のないユーザーは読み取り専用モードで使用できます。**virsh** パッケージは、libvirt-client パッケージの一部として **yum** とともにインストールされます。

インストール手順については、「仮想化パッケージの手動インストール」 を参照してください。virsh の概要 (実用的なデモンストレーションを含む) は、Virtualization Getting Started Guide を参照してください。本章の残りのセクションでは、使用方法に基づいた論理的な順序で設定された **virsh** コマンドについて説明します。

注記

ヘルプを使用する場合、または man ページを読み込む場合は、ゲスト仮想マシンという用語の代わりに、'domain' という用語が使用されることに注意してください。これは、libvirt が使用する用語です。画面の出力が表示され、'domain' が使用されている場合は、ゲストまたはゲスト仮想マシンに切り替わりません。すべての例では、ゲスト仮想マシン 'guest1' が使用されます。いずれの場合も、これをゲスト仮想マシンの名前に置き換える必要があります。ゲスト仮想マシンの名前を作成する場合は、短くて覚えやすい整数 (0,1,2...) やテキスト文字列名を使用するか、すべての場合で仮想マシンの完全な UUID を使用することもできます。

重要

使用しているユーザーに気をつける必要があります。あるユーザーを使用してゲスト仮想マシンを作成した場合、別のユーザーを使用してそのユーザーに関する情報を取得することはできません。virt-manager で仮想マシンを作成する場合は、これが特に重要です。デフォルトのユーザーは、特に指定がない限り、この場合は root になります。**virsh list --all** コマンドを使用して仮想マシンのリストを表示できない場合は、仮想マシンの作成に使用したユーザーとは別のユーザーを使用してコマンドを実行している可能性が最も高いです。詳細は、重要 を参照してください。

20.1. ゲスト仮想マシンの状態とタイプ

virsh コマンドの一部は、ゲスト仮想マシンのステータスの影響を受けます。

- *Transient* - 一時的なゲストは再起動しても存続しません。
- *Persistent* - 永続的なゲスト仮想マシンは再起動しても存続し、削除されるまで持続します。

仮想マシンのライフサイクル期間中、libvirt は、以下の状態のいずれかでゲストを分類します。

- Undefined - これは、定義されていないか作成されていないゲスト仮想マシンです。したがって、libvirt はこの状態のゲストを認識せず、この状態のゲスト仮想マシンについて報告しません。
- Shut off - これは定義されているものの、実行していないゲスト仮想マシンです。永続ゲストのみが停止と見なされます。このため、一時的なゲスト仮想マシンがこの状態になると、ゲスト仮想マシンは存在しなくなります。
- Running - この状態のゲスト仮想マシンは定義されており、現在機能しています。この状態は、永続ゲスト仮想マシンおよび一時ゲスト仮想マシンの両方で使用できます。

- Paused - ハイパーバイザーでのゲスト仮想マシンの実行が一時停止しているか、状態が再開するまで一時的に保存されています。この状態のゲスト仮想マシンは、一時停止していることを認識しません。また、再開時に時間が経過していることにも気づきません。

- Saved - この状態は paused 状態に似ていますが、ゲスト仮想マシンの設定は永続ストレージに保存されます。この状態のゲスト仮想マシンは一時停止していることを認識しません。また、復元後に時間が経過したことも通知しません。

20.2. VIRSH バージョンの表示

virsh version コマンドは、現在の libvirt バージョンを表示し、ローカルの virsh クライアントに関する情報を表示します。以下に例を示します。

```
$ virsh version
Compiled against library: libvirt 1.2.8
Using library: libvirt 1.2.8
Using API: QEMU 1.2.8
Running hypervisor: QEMU 1.5.3
```

virsh version --daemon は、ホストで実行している libvirt デーモンの情報など、 **libvirtd** バージョンとパッケージ情報を取得する場合に役立ちます。

```
$ virsh version --daemon
Compiled against library: libvirt 1.2.8
Using library: libvirt 1.2.8
Using API: QEMU 1.2.8
Running hypervisor: QEMU 1.5.3
Running against daemon: 1.2.8
```

20.3. ECHO を使用したコマンドの送信

virsh echo [--shell][--xml] ***arguments*** コマンドは、指定した形式で、指定した引数を表示します。使用できる形式は、**--shell** と **--xml** です。クエリーされる各引数は、空白で区切ります。**--shell** オプションは、必要に応じて一重引用符でフォーマットされた出力を生成するため、コマンドとして bash モードにコピーアンドペーストする場合に適しています。**--xml** 引数を使用すると、出力は XML ファイルで使用するようにフォーマットされ、ゲストの設定に保存または使用できます。

20.4. VIRSH CONNECT を使用したハイパーバイザーへの接続

virsh connect [*hostname-or-URI*] [--readonly] コマンドは、virsh を使用してローカルのハイパーバイザーセッションを開始します。このコマンドを初めて実行すると、virsh シェルが実行するたびに自動的に実行されます。ハイパーバイザー接続 URI は、ハイパーバイザーへの接続方法を指定します。最も一般的に使用される URI は以下のとおりです。

- **qemu:///system** - KVM ハイパーバイザーのゲスト仮想マシンを監視するデーモンに、root ユーザーとしてローカルに接続します。

- **qemu:///session** - KVM ハイパーバイザーを使用して、ユーザーのゲストローカルマシンセットに、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。

このコマンドは、次のように、マシン名 (hostname) またはハイパーバイザーの URL(**virsh uri** コマンドの出力) のいずれかで指定されているターゲットゲストを指定して実行できます。

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

たとえば、一連のゲスト仮想マシンに接続するセッションを確立するには、ユーザーをローカルユーザーにします。

```
$ virsh connect qemu:///session
```

読み取り専用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 の詳細は、リモート URI を参照してください。URI がわからない場合は、**virsh uri** コマンドを実行すると以下のようなメッセージが表示されます。

20.5. ゲスト仮想マシンおよびハイパーバイザーに関する情報の表示

virsh list コマンドは、ハイパーバイザーに接続され、要求された検索パラメーターに適合するゲスト仮想マシンをリスト表示します。 コマンドの出力には、テーブルに 3 つの列があります。各ゲスト仮想マシンが、ID、名前、状態 とともにリスト表示されます。

virsh listには、さまざまな検索パラメーターが利用できます。このオプションは、man ページ、**man virsh** の実行、または **virsh list --help** コマンドの実行で利用できます。

注記

このコマンドを実行しても、root ユーザーが作成したゲスト仮想マシンのみが表示されることに注意してください。作成した仮想マシンが表示されない場合は、root で仮想マシンを作成していない可能性があります。

virt-manager インターフェイスを使用して作成されたゲストは、デフォルトでは root により作成されます。

例20.1 ローカルに接続したすべての仮想マシンのリストを表示する方法

以下の例は、ハイパーバイザーが接続している仮想マシンのリストを表示します。このコマンドは、永続的な仮想マシンと 一時的な 仮想マシンの両方をリスト表示することに留意してください。

```
# virsh list --all

Id          Name                State
----------------------------------------------------
8  guest1   running
22  guest2   paused
35  guest3   shut off
38           guest4   shut off
```

例20.2 非アクティブのゲスト仮想マシンのリストを表示する方法

以下の例は、現在非アクティブであるか、実行していないゲストのリストを表示します。このリストには永続的な仮想マシンのみが含まれていることに注意してください。

```
# virsh list --inactive

Id          Name                State
```

```
----------------------------------------------------
35  guest3   shut off
38  guest4   shut off
```

さらに、次のコマンドを使用して、ハイパーバイザーの基本情報を表示することもできます。

- # **virsh hostname** - ハイパーバイザーのホストネームを表示します。以下に例を示します。

```
# virsh hostname
dhcp-2-157.eus.myhost.com
```

- # **virsh sysinfo** - ハイパーバイザーシステム情報の XML 表現を表示します (存在する場合)。以下に例を示します。

```
# virsh sysinfo
<sysinfo type='smbios'>
  <bios>
    <entry name='vendor'>LENOVO</entry>
    <entry name='version'>GJET71WW (2.21 )</entry>
[...]
```

20.6. 仮想マシンの起動、再開、および復元

20.6.1. ゲスト仮想マシンの起動

virsh start *domain*; [--console] [--paused] [--autodestroy] [--bypass-cache] [--force-boot] コマンドは、定義済みで非アクティブの仮想マシンを起動します。この仮想マシンのステータスは、最後の管理保存状態または起動直後から非アクティブになっています。デフォルトでは、ドメインが **virsh managedsave** コマンドにより保存された場合、ドメインは以前の状態に復元されます。これを行わないと、新たに起動されます。このコマンドには次の引数を使用でき、仮想マシンの名前が必要になります。

- **--console** - **virsh** を実行している端末をドメインのコンソールデバイスに接続します。これは、ランレベル 3 です。
- **--paused** - これがドライバーでサポートされている場合は、ゲスト仮想マシンを一時停止状態で起動します。
- **--autodestroy** - virsh の接続が切断されると、ゲスト仮想マシンが自動的に破棄されます。
- **--bypass-cache** - ゲスト仮想マシンが **managedsave** 内にある場合に使用されます。
- **--force-boot** - **managedsave** オプションをすべて破棄し、新規ブートが実行されます。

例20.3 仮想マシンの起動方法

以下の例では、作成済みで現在非アクティブな状態にある*guest1*仮想マシンを起動します。さらに、このコマンドは、ゲストのコンソールを virsh を実行しているターミナルに割り当てます。

```
# virsh start guest1 --console
Domain guest1 started
Connected to domain guest1
```

```
Escape character is ^]
```

20.6.2. システムの起動時に仮想マシンを自動的に起動する設定

virsh autostart [--disable] ***domain*** コマンドは、ホストマシンの起動時にゲスト仮想マシンを自動的に起動します。このコマンドに **--disable** 引数を追加すると、自動起動が無効になります。この場合のゲストは、ホストの物理マシンの起動時には自動的に起動しません。

例20.4 ホストの物理マシンの起動時に仮想マシンを自動的に起動する方法

以下の例では、ホストの起動時に自動起動するためにすでに作成した *guest1* 仮想マシンを設定します。

```
# virsh autostart guest1
```

20.6.3. ゲスト仮想マシンの再起動

virsh reboot ***domain*** **[--mode** ***modename*****]** コマンドを使用して、ゲスト仮想マシンを再起動します。このアクションは再起動を実行した後にのみ返されるため、その時点からゲスト仮想マシンが実際に再起動するまでに時間がかかる場合があることに注意してください。ゲスト仮想マシンの XML 設定ファイルの **on_reboot** 要素を変更することで、再起動中のゲスト仮想マシンの動作を制御できます。デフォルトでは、ハイパーバイザーは適切なシャットダウン方法を自動的に選択しようとします。別の方法を指定する場合は、**--mode** 引数を使用して、**acpi** と **agent** を含むコンマ区切りのリストを指定できます。ドライバーが各モードを試行する順序は定義されておらず、virsh で指定された順序とは無関係です。順序を厳格に制御するには、一度に 1 つのモードを使用し、コマンドを繰り返します。

例20.5 ゲスト仮想マシンを再起動する方法

次の例では、*guest1* という名前のゲスト仮想マシンを再起動します。この例では、再起動で **initctl** メソッドが使用されますが、必要に応じて任意のモードを選択できます。

```
# virsh reboot guest1 --mode initctl
```

20.6.4. ゲスト仮想マシンの復元

virsh restore <file> [--bypass-cache] [--xml ***/path/to/file*****] [--running] [--paused]** コマンドは、**virsh save** コマンドで保存したゲスト仮想マシンを復元します。**virsh save** コマンドの詳細は、「ゲスト仮想マシンの設定の保存」 を参照してください。復元操作により、保存したゲスト仮想マシンが再起動します。これには時間がかかる場合があります。ゲスト仮想マシンの名前と UUID は保持されますが、その ID が、仮想マシンを保存したときに保持されていた ID と一致するとは限りません。

virsh restore コマンドには、以下の引数を使用できます。

- **--bypass-cache** - 復元を実行してファイルシステムキャッシュを回避しますが、このフラグを使用すると復元動作が遅くなる可能性があることに注意してください。
- **--xml** - この引数は、XML ファイル名とともに使用する必要があります。通常、この引数は省略されますが、ドメイン XML のホスト固有の部分のみを変更し、復元したゲスト仮想マシンで使用するための代替 XML ファイルを提供するために使用できます。たとえば、ゲストの保存後

に取得されたディスクスナップショットによる、基に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** - 保存イメージに記録された状態を上書きして、ゲスト仮想マシンを実行時に起動します。
- **--paused** - 保存イメージに記録された状態を上書きし、ゲスト仮想マシンを一時停止として起動します。

例20.6 ゲスト仮想マシンを復元する方法

以下の例では、ゲスト仮想マシンと、その実行設定ファイルの*guest1-config.xml*を復元します。

```
# virsh restore guest1-config.xml --running
```

20.6.5. ゲスト仮想マシンの再開

virsh resume *domain* コマンドは、サスペンドされたドメインの CPU を再起動します。この操作は即座に行われます。ゲスト仮想マシンは、一時停止した時点から実行を再開します。このアクションは、定義されていないゲスト仮想マシンを再開しないことに注意してください。この操作は、transient 仮想マシンを再開せず、永続仮想マシンでのみ機能します。

例20.7 一時停止しているゲスト仮想マシンを復元する方法

以下の例では、*guest1* 仮想マシンを復元します。

```
# virsh resume guest1
```

20.7. 仮想マシンの設定の管理

本セクションでは、仮想マシンの設定を管理する方法を説明します。

20.7.1. ゲスト仮想マシンの設定の保存

virsh save [--bypass-cache] *domain* file [--xml *string*] [--running] [--paused] [--verbose] コマンドは、指定したドメインを停止し、ゲスト仮想マシンのシステムメモリーの現在の状態を指定したファイルに保存します。ゲスト仮想マシンが使用するメモリー量によっては、かなりの時間がかかる可能性があります。**virsh restore** (「ゲスト仮想マシンの復元」) コマンドを使用すると、ゲスト仮想マシンの状態を復元できます。

virsh save コマンドと **virsh suspend** コマンドの違いは、**virsh suspend** はドメイン CPU を停止しますが、ドメインの **qemu** プロセスは実行中のままにし、そのメモリーイメージはホストシステムに常駐します。このメモリーイメージは、ホストシステムを再起動すると失われます。

virsh save コマンドは、ホストシステムのハードディスクにあるドメインの状態を保存し、**qemu** プロセスを終了します。これにより、保存した状態からドメインを再起動できます。

virsh domjobinfo コマンドを使用して**virsh save** のプロセスを監視し、**virsh domjobabort** コマンドを使用して取り消すことができます。

virsh save コマンドには、以下の引数を使用できます。

- **--bypass-cache** - 復元を実行してファイルシステムキャッシュを回避しますが、このフラグを使用すると復元動作が遅くなる可能性があることに注意してください。
- **--xml** - この引数は、XML ファイル名とともに使用する必要があります。通常、この引数は省略されますが、ドメイン XML のホスト固有の部分のみを変更し、復元したゲスト仮想マシンで使用するための代替 XML ファイルを提供するために使用できます。たとえば、ゲストの保存後に取得されたディスクスナップショットによる、基に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** - 保存イメージに記録された状態を上書きして、ゲスト仮想マシンを実行時に起動します。
- **--paused** - 保存イメージに記録された状態を上書きし、ゲスト仮想マシンを一時停止として起動します。
- **--verbose** - 保存の進捗を表示します。

例20.8 設定を実行しているゲスト仮想マシンを保存する方法

以下の例では、*guest1* 仮想マシンの実行設定を **guest1-config.xml** ファイルに保存します。

```
# virsh save guest1 guest1-config.xml --running
```

20.7.2. XML ファイルを使用したゲスト仮想マシンの定義

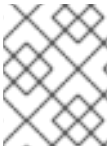
virsh define *filename* コマンドは、XML ファイルからゲスト仮想マシンを定義します。この場合のゲスト仮想マシンの定義は登録されていますが、起動していません。ゲスト仮想マシンがすでに実行している場合は、ドメインをシャットダウンして再起動すると変更が有効になります。

例20.9 XML ファイルからゲスト仮想マシンを作成する方法

以下の例では、既存の*guest1-config.xml* XML ファイルから仮想マシンを作成します。このファイルには、仮想マシンの設定が含まれます。

```
# virsh define guest1-config.xml
```

20.7.3. ゲスト仮想マシンの復元に使用される XML ファイルの更新



注記

このコマンドは、ゲスト仮想マシンが正しく実行されない状況から回復するためにのみ使用する必要があります。これは一般的な使用を目的としていません。

virsh save-image-define *filename* [--xml */path/to/file*] [--running] [--paused] コマンドは、ゲスト仮想マシンの XML ファイルを更新します。これは、**virsh restore** コマンドを使用して仮想マシンを復元する際に使用されます。*--xml* 引数は、ゲスト仮想マシンの XML 用の代替 XML 要素を含む XML ファイル名である必要があります。たとえば、ゲストを保存した後に、基となるストレージのディスクスナップショットを作成することで生じるファイルの命名の相違点を説明するために使用できます。ゲスト仮想マシンを実行中または一時停止の状態に復元する場合は、イメージレコードを保存します。引数 **--running** または **--paused** を使用すると、使用される状態が決定します。

例20.10 ゲスト仮想マシンの実行設定を保存する方法

以下の例では、*guest1-config.xml* 設定ファイルを、対応する実行中のゲストの状態で更新します。

```
# virsh save-image-define guest1-config.xml --running
```

20.7.4. ゲスト仮想マシンの XML ファイルの抽出

注記

このコマンドは、ゲスト仮想マシンが正しく実行されない状況から回復するためにのみ使用する必要があります。これは一般的な使用を目的としていません。

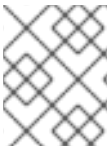
virsh save-image-dumpxml *file* --security-info コマンドは、保存された状態ファイル (**virsh save** コマンドで使用) が参照された時に有効だったゲスト仮想マシンの XML ファイルを抽出します。**--security-info** 引数を使用すると、ファイル内に機密情報が含まれます。

例20.11 最後に保存した XML 設定をプルする方法

次の例では、ゲスト仮想マシンが最後に保存されたときに作成された設定ファイルのダンプをトリガーします。この例では、生成されるダンプファイルの名前は *guest1-config-xml* です。

```
# virsh save-image-dumpxml guest1-config.xml
```

20.7.5. ゲスト仮想マシンの設定の編集



注記

このコマンドは、ゲスト仮想マシンが正しく実行されない状況から回復するためにのみ使用する必要があります。これは一般的な使用を目的としていません。

virsh save-image-edit <file> [--running] [--paused] コマンドは、**virsh save** コマンドが作成した XML 設定ファイルを編集します。**virsh save** コマンドの詳細は、「ゲスト仮想マシンの設定の保存」 を参照してください。

ゲスト仮想マシンが保存されると、作成されるイメージファイルは、仮想マシンを **--running** または **--paused** 状態に復元するかどうかを示します。**save-image-edit** コマンドでこれらの引数を使用しない場合は、状態はイメージファイル自体により決定されます。**--running** (実行中の状態を選択) または **--paused** (一時停止した状態を選択) を選択すると、**virsh restore** が使用する状態を上書きできます。

例20.12 ゲスト仮想マシンの設定を編集し、マシンを実行状態に復元する方法

以下の例では、デフォルトエディターで編集するために、ゲスト仮想マシンの設定ファイル *guest1-config.xml* を開きます。編集を保存すると、仮想マシンが新しい設定で起動します。

```
# virsh save-image-edit guest1-config.xml --running
```

20.8. ゲスト仮想マシンのシャットオフ、シャットダウン、再起動、および強制的なシャットダウン

20.8.1. ゲスト仮想マシンのシャットダウン

virsh shutdown *domain* [--mode *modename*] コマンドは、ゲスト仮想マシンをシャットダウンします。ゲスト仮想マシンの設定ファイルの **on_shutdown** パラメーターを変更することで、ゲスト仮想マシンの再起動方法の動作を制御できます。**on_shutdown** パラメーターの変更は、ドメインをシャットダウンして再起動しないと反映されません。

virsh shutdown コマンドには、以下のオプションを指定できます。

- **--mode** が シャットダウンモードを選択します。これは、**acpi**、**agent**、**initctl**、**signal**、または **paravirt** のいずれかになります。

例20.13 ゲスト仮想マシンをシャットダウンする方法

以下の例では、**acpi** モードを使用して、*guest1* 仮想マシンをシャットダウンします。

```
# virsh shutdown guest1 --mode acpi
Domain guest1 is being shutdown
```

20.8.2. ゲスト仮想マシンの一時停止

virsh suspend *domain* コマンドは、ゲスト仮想マシンを一時停止します。

ゲスト仮想マシンが一時停止状態の場合、システム RAM は消費されますが、プロセッサーリソースは消費されません。ゲスト仮想マシンが一時停止されている間は、ディスクとネットワークの I/O は発生しません。この操作は即座に行われるため、ゲスト仮想マシンは **virsh resume** コマンドでしか再起動できません。一時的な仮想マシンでこのコマンドを実行すると、コマンドは削除されます。

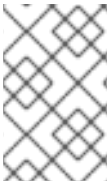
例20.14 ゲスト仮想マシンを一時停止する方法

以下の例では、*guest1*仮想マシンを一時停止します。

```
# virsh suspend guest1
```

20.8.3. 仮想マシンのリセット

virsh reset *domain* は、ゲストをシャットダウンせずにすぐにゲスト仮想マシンをリセットします。リセットは、マシンのリセットボタンをエミュレートします。すべてのゲストハードウェアが RST 行を認識し、内部状態を再初期化します。ゲストの仮想マシン OS がシャットダウンしないと、データが失われる可能性があることに注意してください。



注記

仮想マシンをリセットしても、ドメイン設定の保留中の変更は適用されません。ドメインの設定の変更は、ドメインの完全なシャットダウンおよび再起動後にのみ反映されます。

例20.15 ゲスト仮想マシンをリセットする方法

以下の例では、*guest1* 仮想マシンをリセットします。

```
# virsh reset guest1
```

20.8.4. 後で再起動するために実行中のゲスト仮想マシンの停止

virsh managedsave *domain* --bypass-cache --running | --paused | --verbose コマンドは、実行中のゲスト仮想マシンを保存および破棄 (停止) し、後で同じ状態から再起動できるようにします。**virsh start** コマンドと併用すると、この保存ポイントから自動的に開始されます。**--bypass-cache** 引数と一緒に使用すると、ファイルシステムキャッシュが回避されます。このオプションを使用すると、保存プロセスの速度が遅くなり、**--verbose** オプションを使用したダンププロセスの進捗が表示されることに注意してください。通常の条件では、管理保存は、保存が完了したときにゲスト仮想マシンが置かれている状態によって決定される実行状態と一時停止状態のどちらを使用するかを決定します。ただし、これは、**--running** オプションを使用して、実行状態のままにする必要があることを示したり、**--paused** オプションを使用して、一時停止状態のままにすることを示したりして上書きできます。管理保存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、ゲスト仮想マシンが次に起動したときに強制的に完全起動します。管理保存プロセス全体は、**domjobinfo** コマンドを使用して監視できます。**domjobabort** コマンドを使用して監視を中止することもできます。

例20.16 実行中のゲストを停止し、設定を保存する方法

以下の例では、*guest1* 仮想マシンを停止し、実行設定を保存して再起動できるようにしています。

```
# virsh managedsave guest1 --running
```

20.9. 仮想マシンの削除

20.9.1. 仮想マシンの定義解除

virsh undefine *domain* [--managed-save] [*storage*] [--remove-all-storage] [--wipe-storage] [--snapshots-metadata] [--nvram] コマンドは、ドメインの定義を解除します。ドメインが非アクティブの場合は、設定が完全に削除されます。ドメインがアクティブ (実行中) の場合は、一時的 ドメインに変換されます。ゲスト仮想マシンが非アクティブになると、設定は完全に削除されます。

このコマンドには、以下の引数を使用できます。

- **--managed-save** - この引数は、管理保存イメージもクリーンアップされることを保証します。この引数を使用しないと、管理保存でゲスト仮想マシンの定義を解除しようとすると失敗します。
- **--snapshots-metadata** - この引数は、非アクティブなゲスト仮想マシンを定義する際に、(**snapshot-list** で表示される) スナップショットもクリーンアップすることを保証します。スナップショットのメタデータを使用して、アクティブでないゲスト仮想マシンの定義を解除しようとすると、失敗することに注意してください。この引数を使用し、ゲスト仮想マシンがアクティブな場合は無視されます。
- **--storage** - この引数を使用する場合は、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をコンマで区切って指定し、未定義のドメインと一緒に削除する必要があります。このアクションでは、ストレージボリュームを削除する前にそのストレージボリュームの

定義が解除されます。これは、非アクティブのゲスト仮想マシンでのみ可能であり、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注意してください。

- **--remove-all-storage** - ゲスト仮想マシンの定義を解除するほかに、関連するすべてのストレージボリュームが削除されます。仮想マシンを削除する場合は、関連付けられた同じストレージを使用する仮想マシンがほかにないときに限り、このオプションを選択します。別の方法として、**virsh vol-delete** を使用する方法があります。詳細は、「ストレージボリュームの削除」 を参照してください。

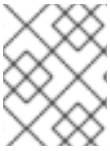
- **--wipe-storage** - ストレージボリュームを削除する以外にも、コンテンツをワイプします。

例20.17 ゲスト仮想マシンを削除し、そのストレージボリュームを削除する方法

以下の例では、*guest1* 仮想マシンの定義を解除し、関連するすべてのストレージボリュームを削除します。定義されていないゲストは一時的なものになり、シャットダウンすると削除されます。

```
# virsh undefine guest1 --remove-all-storage
```

20.9.2. ゲスト仮想マシンの強制停止



注記

このコマンドは、他の方法で仮想ゲストマシンをシャットダウンできない場合にのみ使用してください。

virsh destroy コマンドは、通常とは異なる即時のシャットダウンを開始し、指定したゲスト仮想マシンを停止します。**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。**virsh destroy** コマンドは、ゲスト仮想マシンが応答しない場合にのみ使用してください。**--graceful** オプションを指定した **virsh destroy** コマンドは、仮想マシンの電源を切る前にディスクイメージファイルのキャッシュをフラッシュしようとします。

例20.18 ハードシャットダウンしてゲスト仮想マシンをすぐにシャットダウンする方法

以下の例では、*guest1* 仮想マシンが (おそらく応答しないため) すぐにシャットダウンします。

```
# virsh destroy guest1
```

これには、**virsh undefine** コマンドを使用します。例20.17「ゲスト仮想マシンを削除し、そのストレージボリュームを削除する方法」 を参照

20.10. ゲスト仮想マシンのシリアルコンソールの接続

virsh console *domain* [--devname *devicename*] [--force] [--safe] コマンドは、ゲスト仮想マシンの仮想シリアルコンソールを接続します。これは、VNC プロトコルまたは SPICE プロトコルを提供しない (つまりGUI ツール 用のビデオディスプレイを提供しない) ゲストや、ネットワークに接続されていない (そのため SSH を使用して対話できない) ゲストにとても役立ちます。

オプションの **--devname** パラメーターは、ゲスト仮想マシンに設定された代替コンソール、シリアル、またはパラレルデバイスのデバイスエイリアスを参照します。このパラメーターを省略すると、プライマリーコンソールが開きます。**--safe** オプションが指定されている場合、ドライバーが安全なコン

ソール処理に対応している場合にのみ接続を試みます。このオプションは、サーバーがコンソールデバイスへの排他アクセスを確保する必要があることを指定します。必要に応じて、**force** オプションを指定します。これは、接続が切断された場合など、既存のセッションの接続を解除するように要求します。

例20.19 コンソールモードでゲスト仮想マシンを起動する方法

以下の例では、作成済みの*guest1* 仮想マシンを起動し、セーフコンソール処理を使用してシリアルコンソールに接続します。

```
# virsh console guest1 --safe
```

20.11. マスク不可割り込みの挿入

virsh inject-nmi *domain* は、マスク不可割り込み (NMI) メッセージをゲスト仮想マシンに挿入します。これは、回復不能なハードウェアエラーなど、応答時間が重要な場合に使用されます。また、**virsh inject-nmi** は、Windows ゲストでクラッシュダンプをトリガーする場合に役立ちます。

例20.20 ゲスト仮想マシンに NMI を挿入する方法

以下の例では、*guest1* 仮想マシンに NMI を送信します。

```
# virsh inject-nmi guest1
```

20.12. 仮想マシンに関する情報の取得

20.12.1. デバイスブロック統計の表示

デフォルトでは、**virsh domblkstat** コマンドは、ドメインに定義されている最初のブロックデバイスのブロック統計を表示します。他のブロックデバイスの統計情報を表示するには、**virsh domblklist *domain*** コマンドを使用してすべてのブロックデバイスのリストを表示し、特定のブロックデバイスを選択してから、ドメイン名の後の **virsh domblklist** コマンドの出力から **ターゲット** 名または **ソース** 名を指定して表示します。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できるわけではないことに注意してください。出力が最も読みやすい形式で表示されるようにするには、**--human** 引数を使用します。

例20.21 ゲスト仮想マシンのブロック統計を表示する方法

以下の例は、*guest1* 仮想マシンに定義されたデバイスを表示し、そのデバイスのブロック統計をリスト表示します。

```
# virsh domblklist guest1

Target     Source
------------------------------------------------
vda        /VirtualMachines/guest1.img
hdc        -

# virsh domblkstat guest1 vda --human
Device: vda
```

```
number of read operations:      174670
number of bytes read:           3219440128
number of write operations:     23897
number of bytes written:        164849664
number of flush operations:     11577
total duration of reads (ns):   1005410244506
total duration of writes (ns):  1085306686457
total duration of flushes (ns): 340645193294
```

20.12.2. ネットワークインターフェイス統計の取得

virsh domifstat *domain interface-device* コマンドは、指定されたゲスト仮想マシンで実行している指定されたデバイスのネットワークインターフェイス統計情報を表示します。

ドメインに定義されているインターフェイスデバイスを特定するには、**virsh domiflist** コマンドを使用し、Interface 列の出力を使用します。

例20.22 ゲスト仮想マシンのネットワーク統計を表示する方法

以下の例では、*guest1* 仮想マシンに定義されているネットワークインターフェイスを取得した後、取得したインターフェイス (*macvtap0*) のネットワーク統計量を表示します。

```
# virsh domiflist guest1
Interface  Type       Source     Model       MAC
-------------------------------------------------------
macvtap0   direct     em1        rtl8139     12:34:00:0f:8a:4a
# virsh domifstat guest1 macvtap0
macvtap0 rx_bytes 51120
macvtap0 rx_packets 440
macvtap0 rx_errs 0
macvtap0 rx_drop 0
macvtap0 tx_bytes 231666
macvtap0 tx_packets 520
macvtap0 tx_errs 0
macvtap0 tx_drop 0
```

20.12.3. ゲスト仮想マシンの仮想インターフェイスのリンク状態の変更

virsh domif-setlink *domain interface-device state* コマンドは、指定したインターフェイスデバイスリンクのステータスを **up** または **down** のいずれかに設定します。ドメインに定義されているインターフェイスデバイスを特定するには、**virsh domiflist** コマンドを使用し、インターフェイスデバイスオプションとして **Interface** 列または **MAC** 列のいずれかを使用します。デフォルトでは、**virsh domif-setlink** は、実行中のドメインのリンクステートを変更します。ドメインの永続的な設定を変更するには、**--config** 引数を使用します。

例20.23 ゲスト仮想マシンインターフェイスを有効にする方法

次の例は、*rhel7* ドメインのインターフェイスデバイスを決定し、リンクを**down**として設定し、最後に**up**として設定する方法を示しています。

```
# virsh domiflist rhel7
Interface  Type       Source     Model       MAC
```

```
----------------------------------------------------
vnet0      network    default    virtio      52:54:00:01:1d:d0

# virsh domif-setlink rhel7 vnet0 down
Device updated successfully

# virsh domif-setlink rhel7 52:54:00:01:1d:d0 up
Device updated successfully
```

20.12.4. ゲスト仮想マシンの仮想インターフェイスのリンク状態のリスト表示

virsh domif-getlink *domain interface-device* コマンドは、指定したインターフェイスデバイスリンクステートを取得します。ドメインに定義されているインターフェイスデバイスを特定するには、**virsh domiflist** コマンドを使用し、インターフェイスデバイスオプションとして **Interface** 列または **MAC** 列のいずれかを使用します。デフォルトでは、**virsh domif-getlink** は、実行中のドメインのリンクステートを取得します。ドメインの永続的な設定を取得するには、 **--config option** を使用します。

例20.24 ゲスト仮想マシンのインターフェイスのリンク状態を表示する方法

以下の例は、*rhel7* ドメインのインターフェイスデバイスを判別し、その状態を **up** として決定し、続いて状態を **down** に変更してから、変更が正常に行われたことを確認します。

```
# virsh domiflist rhel7
Interface  Type       Source     Model       MAC
----------------------------------------------------
vnet0      network    default    virtio      52:54:00:01:1d:d0

# virsh domif-getlink rhel7 52:54:00:01:1d:d0
52:54:00:01:1d:d0 up

# virsh domif-setlink rhel7 vnet0 down
Device updated successfully

# virsh domif-getlink rhel7 vnet0
vnet0 down
```

20.12.5. ネットワークインターフェイスの帯域幅パラメーターの設定

virsh domiftune *domain interface-device* コマンドは、指定したドメインのインターフェイスの帯域幅パラメーターを取得または設定します。ドメインに定義されているインターフェイスデバイスを特定するには、**virsh domiflist** コマンドを使用し、インターフェイスデバイスオプションとして **Interface** 列または **MAC** 列のいずれかを使用します。以下の形式を使用する必要があります。

```
# virsh domiftune domain interface [--inbound] [--outbound] [--config] [--live] [--current]
```

--config、**--live**、および **--current** のオプションは、「スケジュールパラメーターの設定」 で説明されています。**--inbound** または **--outbound** を指定しない場合は、**virsh domiftune** が指定したネットワークインターフェイスを照会し、帯域幅設定を表示します。**--inbound** または **--outbound** のいずれか、両方を指定し、平均、ピーク、およびバーストの値を指定すると、**virsh domiftune** は帯域幅設定

を設定します。少なくとも平均値が必要です。帯域幅設定を削除する場合は、0 (ゼロ) を指定します。平均値、ピーク値、およびバースト値の説明については、「インターフェイスデバイスの接続」 を参照してください。

例20.25 ゲスト仮想マシンのネットワークインターフェイスパラメーターの設定方法

以下の例では、*guest1* という名前のゲスト仮想マシンの *eth0* パラメーターを設定します。

```
# virsh domiftune guest1 eth0 outbound --live
```

20.12.6. メモリー統計の取得

virsh dommemstat *domain* [<period in seconds>] [--config] [--live] [--current] コマンドは、実行中のゲスト仮想マシンのメモリー統計を表示します。オプションの **period** スイッチを使用するには、秒単位の期間が必要です。このオプションを 0 より大きい値に設定すると、バルーンドライバーは追加の統計を返します。これは、後続の **dommemstat** コマンドを実行すると表示されます。**period** オプションを 0 に設定すると、バルーンドライバーの収集を停止しますが、バルーンドライバーにある統計は消去されません。**period** オプションも設定しないと、**--live** オプション、**--config** オプション、または **--current** オプションを使用できません。**--live** オプションを指定すると、ゲストの実行中の統計のみが収集されます。**--config** オプションを使用すると、永続的なゲストの統計が収集されますが、次回の起動後にのみ収集されます。**--current** オプションを使用すると、現在の統計が収集されます。

--live オプションおよび **--config** オプションの両方を使用できますが、**--current** は使用できません。フラグが指定されていない場合は、ゲストの状態が統計収集の動作 (実行中かどうか) を決定します。

例20.26 実行中のゲスト仮想マシンのメモリー統計を収集する方法

以下の例は、*rhel7* ドメインのメモリー統計を表示しています。

```
# virsh dommemstat rhel7
actual 1048576
swap_in 0
swap_out 0
major_fault 2974
minor_fault 1272454
unused 246020
available 1011248
rss 865172
```

20.12.7. ブロックデバイスのエラーの表示

virsh domblkerror *domain* コマンドは、**エラー** 状態のすべてのブロックデバイスと、各ブロックデバイスで検出されたエラーをリスト表示します。このコマンドは、**virsh domstate** コマンドが I/O エラーのためにゲスト仮想マシンが一時停止したことを報告した後に使用することを推奨します。

例20.27 仮想マシンのブロックデバイスエラーを表示する方法

以下の例では、*guest1* 仮想マシンのブロックデバイスエラーを表示します。

```
# virsh domblkerror guest1
```

20.12.8. ブロックデバイスのサイズの表示

virsh domblkinfo *domain* コマンドは、仮想マシン上の特定のブロックデバイスの容量、割り当て、および物理ブロックサイズのリストを表示します。virsh domblklist コマンドを使用して、すべてのブロックデバイスのリストを表示し、ドメイン名の後の **virsh domblklist** 出力から **ターゲット** 名または **ソース** 名を指定して、特定のブロックデバイスを表示するように選択します。

例20.28 ブロックデバイスのサイズの表示方法

この例では、*rhel7* 仮想マシンのブロックデバイスのリストを表示し、各デバイスのブロックサイズを表示します。

```
# virsh domblklist rhel7
Target     Source
------------------------------------------------
vda        /home/vm-images/rhel7-os
vdb        /home/vm-images/rhel7-data

# virsh domblkinfo rhel7 vda
Capacity:       10737418240
Allocation:     8211980288
Physical:       10737418240

# virsh domblkinfo rhel7 /home/vm-images/rhel7-data
Capacity:       104857600
Allocation:     104857600
Physical:       104857600
```

20.12.9. ゲスト仮想マシンに関連付けられたブロックデバイスの表示

virsh domblklist *domain* [--inactive] [--details] コマンドは、指定されたゲスト仮想マシンに関連付けられたすべてのブロックデバイスの表を表示します。

--inactive を指定すると、次回のシステムの起動時に使用されるデバイスが結果に表示されます。また、実行中のゲスト仮想マシンで現在使用中のデバイスは表示されません。**--details** を指定すると、ディスクのタイプとデバイス値がテーブルに含まれます。この表に表示される情報は、**virsh domblkinfo** および **virsh snapshot-create** など、ブロックデバイスを提供する必要があるその他のコマンドで使用できます。**virsh snapshot-create** コマンドで xmlfile コンテキスト情報を生成する場合は、ディスク **ターゲット** コンテキストまたは **ソース** コンテキストも使用できます。

例20.29 仮想マシンに関連付けられているブロックデバイスを表示する方法

以下の例では、*rhel7* 仮想マシンに関連付けられたブロックデバイスの詳細を表示します。

```
# virsh domblklist rhel7 --details
Type       Device     Target     Source
------------------------------------------------
file       disk       vda        /home/vm-images/rhel7-os
file       disk       vdb        /home/vm-images/rhel7-data
```

20.12.10. ゲスト仮想マシンに関連付けられた仮想インターフェイスの表示

virsh domblklist *domain* コマンドは、指定し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イスのテーブルを表示します。**virsh domiflist** コマンドには仮想マシン (または *domain*) の名前が必要で、オプションで **--inactive** 引数を指定することもできます。後者は、実行中の設定ではなく非アクティブの設定を取得します。これは、デフォルト設定で取得されます。**--inactive** を指定すると、結果は、次回の起動時に使用されるデバイスを示し、実行中のゲストが現在使用しているデバイスは示しません。仮想インターフェイス (**detach-interface**、**domif-setlink**、**domif-getlink**、**domifstat**、および **domiftune** など) の MAC アドレスを必要とする Virsh コマンドは、このコマンドが表示する出力を受け付けます。

例20.30 ゲスト仮想マシンに関連付けられた仮想インターフェイスを表示する方法

以下の例では、*rhel7* 仮想マシンに関連付けられている仮想インターフェイスを表示した後、*vnet0* デバイスのネットワークインターフェイス統計情報を表示します。

```
# virsh domiflist rhel7
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet0      network    default    virtio      52:54:00:01:1d:d0

# virsh domifstat rhel7 vnet0
vnet0 rx_bytes 55308
vnet0 rx_packets 969
vnet0 rx_errs 0
vnet0 rx_drop 0
vnet0 tx_bytes 14341
vnet0 tx_packets 148
vnet0 tx_errs 0
vnet0 tx_drop 0
```

20.13. スナップショットの使用

20.13.1. データのコピーによるバッキングチェーンの短縮

このセクションでは、**virsh blockcommit *domain* <path> [<bandwidth>] [<base>] [--shallow] [<top>] [--active] [--delete] [--wait] [--verbose] [--timeout <number>] [--pivot] [--keep-overlay] [--async] [--keep-relative]** コマンドを使用して、バッキングチェーンを短縮する方法について説明します。コマンドには、ヘルプメニューまたは man ページに記載されている多数のオプションがあります。

virsh blockcommit コマンドは、チェーンの一部からバッキングファイルにデータをコピーし、コミットされた部分をバイパスするためにチェーンの残りの部分をピボットできるようにします。たとえば、これが現在の状態であるとします。

```
base ← snap1 ← snap2 ← active.
```

virsh blockcommit を使用すると、snap2 のコンテンツが snap1 に移動します。これにより、チェーンから snap2 を削除できるため、バックアップが非常に速くなります。

手順20.1 バッキングチェーンの短縮方法

- *guest1* をゲスト仮想マシンの名前に置き換え、*disk1* をディスクの名前に置き換えます。

```
# virsh blockcommit guest1 disk1 --base snap1 --top snap2 --wait --verbose
```

snap2 の内容が snap1 に移動し、以下のような結果になります。

base ← **snap1** ← **active**.Snap2 が無効になり、削除できるようになりました。

警告

virsh blockcommit は、***--base*** 引数に依存するファイルを破損します。(***--top*** 引数に依存するファイルはベースを参照するようになったため、除きます)。これを回避するには、複数のゲストが共有するファイルに変更をコミットしないでください。***--verbose*** オプションでは、進行状況をスクリーンに表示できます。

20.13.2. イメージの平坦化によるバッキングチェーンの短縮

virsh blockpull は、以下のアプリケーションで使用できます。

1. バッキングイメージチェーンからのデータをイメージに入力して、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自己完結型になり、背面イメージに依存しなくなり、以下のようになります。

   - 前: base.img ← active
   - 後: base.img がゲストで使用されなくなり、Active にすべてのデータが含まれるようになりました。

2. バッキングイメージチェーンの一部を平坦化します。これを使用すると、スナップショットをトップレベルイメージに平坦化し、以下のようになります。

   - 前: base ← sn1 ←sn2 ← active
   - 後: base.img ← active.**アクティブ** には、**sn1** および **sn2** のすべてのデータが含まれるようになりました。また、ゲストでは sn1 も sn2 も使用されないことに注意してください。

3. ディスクイメージを、ホストの新しい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。

   - 前 (元のイメージファイル) - **/fs1/base.vm.img**
   - 後: **/fs2/active.vm.qcow2** が新しいファイルシステムになり、**/fs1/base.vm.img** が使用されなくなります。

4. コピー後のストレージ移行を使用したライブマイグレーションに役立ちます。ディスクイメージは、ライブマイグレーションの完了後に、移行元ホストから移行先ホストにコピーされます。

   要するに、前: **/source-host/base.vm.img** 後: **/destination-host/active.vm.qcow2** となります。**/source-host/base.vm.img** は使用されなくなりました。

手順20.2 データを平坦化してバッキングチェーンを短縮する方法

1. **virsh blockpull** を実行する前にスナップショットを作成すると便利な場合があります。これを行うには、**virsh snapshot-create-as** コマンドを使用します。以下の例では、*guest1* をゲスト仮想マシンの名前に置き換え、*snap1* をスナップショットの名前に置き換えます。

```
# virsh snapshot-create-as guest1 snap1 --disk-only
```

2. チェーンが **base** ← **snap1** ← **snap2** ← **active** のようになる場合は、以下のコマンドを入力し、*guest1* をゲスト仮想マシンの名前に置き換え、*path1* をディスクへのソースパス (**/home/username/VirtualMachines/***) に置き換えます。

```
# virsh blockpull guest1 path1
```

このコマンドは、*snap2* からデータをアクティブにプルすることで、*snap1* のバッキングファイルをアクティブな状態にします。その結果、**base** ← **snap1** ← **active** になります。

3. **virsh blockpull**が完了すると、チェーンに追加イメージを作成したスナップショットのlibvirt追跡は役に立ちなくなります。古いスナップショットのトラッキングを削除する場合は、次のコマンドを実行します。guest1 をゲスト仮想マシンの名前に置き換え、snap1 をスナップショットの名前に置き換えます。

```
# virsh snapshot-delete guest1 snap1 --metadata
```

virsh blockpull の追加アプリケーションは、以下のように実行できます。

例20.31 つのイメージを平坦化し、バッキングイメージチェーンのデータを取り込む方法

次の例では、ゲスト *guest1* で *vda* 仮想ディスクを平坦化し、イメージをバッキングイメージチェーンのデータで取り込み、取り込み操作が完了するまで待機します。

```
# virsh blockpull guest1 vda --wait
```

例20.32 バッキングイメージチェーンの一部を平坦化する方法

次の例では、*/path/to/base.img* ディスクイメージに基づいて、ゲスト *guest1* の *vda* 仮想ディスクを平坦化します。

```
# virsh blockpull guest1 vda /path/to/base.img --base --wait
```

例20.33 ディスクイメージをホストの新しいファイルシステムに移動する方法

ディスクイメージをホストの新しいファイルシステムに移動するには、次の 2 つのコマンドを実行します。各コマンドで、*guest1* をゲスト仮想マシンの名前に置き換え、*disk1* を仮想ディスクの名前に置き換えます。XML ファイル名およびパスを、スナップショットの場所と名前に変更します。

```
# virsh snapshot-create guest1 --xmlfile /path/to/snap1.xml --disk-only
```

```
# virsh blockpull guest1 disk1 --wait
```

例20.34 コピー後のストレージ移行でライブ移行を使用する方法

コピー後のストレージの移行でライブマイグレーションを使用するには、次のコマンドを実行します。

宛先で、次のコマンドを入力します。バッキングファイルを、ホスト上のバッキングファイルの名前と場所に置き換えます。

```
# qemu-img create -f qcow2 -o backing_file=/source-host/vm.img /destination-host/vm.qcow2
```

ソースで次のコマンドを入力します。*guest1* を、ゲスト仮想マシンの名前に置き換えます。

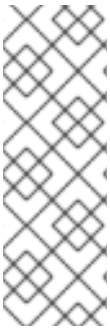
```
# virsh migrate guest1
```

宛先で以下のコマンドを入力します。*guest1* はゲスト仮想マシンの名前に、*disk1* は仮想ディスクの名前に置き換えます。

```
# virsh blockpull guest1 disk1 --wait
```

20.13.3. ゲスト仮想マシンのブロックデバイスのサイズの変更

virsh blockresize コマンドは、ゲスト仮想マシンの実行中に、ブロックデバイスの絶対パスを使用して、ゲスト仮想マシンのブロックデバイスのサイズを変更できます。これは、一意のターゲット名 (**<target dev="name"/>**) またはソースファイル (**<source file="name"/>**) にも対応します。ゲスト仮想マシンに接続されているディスクデバイスの 1 つに適用できます (コマンド**virsh domblklist** を使用すると、指定のゲスト仮想マシンに関連付けられているすべてのブロックデバイスの概要を示す表を出力できます)。



注記

ライブイメージのサイズを変更すると、イメージのサイズは常に変更されますが、ゲストはすぐにはサイズを変更しない場合があります。最近のゲストカーネルでは、virtio-blk デバイスのサイズが自動的に更新されます (古いカーネルではゲストを再起動する必要があります)。SCSI デバイスの場合は、**echo > /sys/class/scsi_device/0:0:0:0/device/rescan** コマンドを使用して、ゲストの再スキャンを手動でトリガーする必要があります。さらに、IDE では、新しいサイズを取得する前にゲストを再起動する必要があります。

例20.35 ゲスト仮想マシンのブロックデバイスのサイズを変更する方法

以下の例では、*guest1* 仮想マシンのブロックデバイスのサイズを 90 バイトに変更します。

```
# virsh blockresize guest1 90 B
```

20.14. グラフィック表示への接続の URI の表示

virsh domdisplay を実行すると、URI が出力されます。URI を使用すると、VNC、SPICE、または RDP を介してゲスト仮想マシンのグラフィカルディスプレイに接続できます。オプションの**--type**を使

用すると、グラフィカルディスプレイの種類を指定できます。引数 **--include-password** を使用すると、SPICE チャンネルパスワードが URI に含まれます。

例20.36 URI for SPICE の表示方法

以下の例では、SPICE の URI を表示しています。これは、仮想マシンの*guest1*が使用しているグラフィカルディスプレイです。

```
# virsh domdisplay --type spice guest1
spice://192.0.2.1:5900
```

コネクション URI の詳細は、the libvirt upstream pages を参照してください。

20.15. VNC ディスプレイの IP アドレスとポート番号の表示

virsh vncdisplay コマンドは、指定されたゲスト仮想マシンの VNC ディスプレイの IP アドレスとポート番号を返します。ゲストで情報が利用できない場合は、終了コード**1**が表示されます。

このコマンドを機能させるには、ゲストの XML ファイルの **devices** 要素で VNC をグラフィックタイプとして指定する必要があります。詳細は、「グラフィカルフレームバッファー」 を参照してください。

例20.37 VNC の IP アドレスとポート番号を表示する方法

以下の例では、*guest1* 仮想マシンの VNC ディスプレイのポート番号を表示します。

```
# virsh vncdisplay guest1
127.0.0.1:0
```

20.16. 使用されていないブロックの破棄

virsh domfstrim *domain* [--minimum *bytes*] [--mountpoint *mountPoint*] コマンドは、指定した実行中のゲスト仮想マシン内で、マウントされたすべてのファイルシステムに対して **fstrim** ユーティリティーを起動します。これは、ファイルシステムが使用していないブロックを破棄します。引数**--minimum** を使用する場合は、バイト単位で指定する必要があります。この量は、連続する空き範囲の長さとしてゲストカーネルに送信されます。この量より小さい値は無視されます。この値を増やすと、空き領域がひどく断片化されたファイルシステムとの競合が発生します。この場合はすべてのブロックが破棄されるわけ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、空きブロックはすべて破棄されます。この値をゼロより大きくすると、fstrim 操作は、フラグメント化された空き領域が不良なファイルシステムでより速く完了します。ただし、すべてのブロックが破棄されるわけではありません。特定のマウントポイントのみをトリムする場合は、**--mountpoint** 引数を使用し、マウントポイントを指定する必要があります。

例20.38 使用していないブロックを破棄する方法

以下の例では、*guest1* という名前のゲスト仮想マシンで実行しているファイルシステムをトリムします。

```
# virsh domfstrim guest1 --minimum 0
```

20.17. ゲスト仮想マシンの取得コマンド

20.17.1. ホストの物理マシン名の表示

virsh domhostname *domain* コマンドは、ハイパーバイザーが公開できる場合に限り、指定したゲスト仮想マシンの物理ホスト名を表示します。

例20.39 ホストの物理マシン名を表示する方法

以下の例では、ハイパーバイザーが*guest1* 仮想マシンを利用可能にしている場合に、そのホスト物理マシン名を表示します。

```
# virsh domhostname guest1
```

20.17.2. 仮想マシンの一般情報の表示

virsh dominfo *domain* コマンドは、指定したゲスト仮想マシンの基本情報を表示します。このコマンドは、**[--domain] *guestname*** オプションと併用できます。

例20.40 ゲスト仮想マシンの一般情報を表示する方法

以下の例では、*guest1* という名前のゲスト仮想マシンの概要を表示します。

```
# virsh dominfo guest1
Id:             8
Name:           guest1
UUID:           90e0d63e-d5c1-4735-91f6-20a32ca22c40
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       271.9s
Max memory:     1048576 KiB
Used memory:    1048576 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c422,c469 (enforcing)
```

20.17.3. 仮想マシンの ID 番号の表示

virsh list の出力には ID が含まれますが、 **virsh domid *domain>|<ID*** は、実行中のゲスト仮想マシンの ID を表示します。ID は、仮想マシンを実行するたびに変更します。ゲスト仮想マシンがシャットダウンすると、マシン名が一連のダッシュ ('-----') として表示されます。このコマンドは、**[--domain *guestname*]** オプションとともに使用することもできます。

例20.41 仮想マシンの ID 番号を表示する方法

このコマンドを実行して使用可能な出力を受信するには、仮想マシンが実行している必要があります。以下の例では、*guest1* 仮想マシンの ID 番号を生成します。

```
# virsh domid guest1
8
```

20.17.4. ゲスト仮想マシンでのジョブの実行の中止

virsh domjobabort *domain* コマンドは、指定されたゲスト仮想マシンで現在実行中のジョブを中止します。このコマンドは、**[--domain *guestname*]** オプションとともに使用することもできます。

例20.42 ゲスト仮想マシンで実行中のジョブを中止する方法

この例では、中断する *guest1* 仮想マシンでジョブが実行されます。コマンドを実行して、*guest1* を仮想マシンの名前に変更します。

```
# virsh domjobabort guest1
```

20.17.5. ゲスト仮想マシンで実行しているジョブの情報表示

virsh domjobinfo *domain* コマンドは、指定されたゲスト仮想マシンで実行しているジョブの情報 (移行統計など) を表示します。このコマンドは **[--domain *guestname*]** オプションと併用したり、**--completed** オプションを指定して、最近完了したジョブの統計情報を返すこともできます。

例20.43 統計的フィードバックの表示方法

以下の例では、*guest1* 仮想マシンの統計情報をリスト表示します。

```
# virsh domjobinfo guest1
Job type:         Unbounded
Time elapsed:     1603        ms
Data processed:   47.004 MiB
Data remaining:   658.633 MiB
Data total:       1.125 GiB
Memory processed: 47.004 MiB
Memory remaining: 658.633 MiB
Memory total:     1.125 GiB
Constant pages:   114382
Normal pages:     12005
Normal data:      46.895 MiB
Expected downtime: 0           ms
Compression cache: 64.000 MiB
Compressed data:  0.000 B
Compressed pages: 0
Compression cache misses: 12005
Compression overflows: 0
```

20.17.6. ゲスト仮想マシンの名前の表示

virsh domname *domainID* コマンドは、ID または UUID を指定して、ゲスト仮想マシン名を表示します。**virsh list --all** コマンドはゲスト仮想マシンの名前も表示しますが、このコマンドではゲスト名のみが表示されます。

例20.44 ゲスト仮想マシンの名前を表示する方法

次の例では、ドメイン ID が *8* のゲスト仮想マシンの名前を表示します。

```
# virsh domname 8
guest1
```

20.17.7. 仮想マシンの状態の表示

virsh domstate *domain* コマンドは、指定したゲスト仮想マシンの状態を表示します。**--reason** 引数を使用すると、表示されているステートの理由も表示されます。このコマンドは **[--domain *guestname*]** オプションでも使用できます。また、その状態の原因を表示する **--reason** オプションも使用できます。コマンドにエラーが表示された場合は、コマンド**virsh domblkerror**を実行してください。詳細は、「ブロックデバイスのエラーの表示」 を参照してください。

例20.45 ゲスト仮想マシンの現在の状態を表示する方法

以下の例では、*guest1* 仮想マシンの現在の状態を表示します。

```
# virsh domstate guest1
running
```

20.17.8. 仮想マシンへの接続状態の表示

virsh domcontrol *domain* は、指定されたゲスト仮想マシンの制御に使用されるハイパーバイザーへのインターフェイスの状態を表示します。OK でも Error でもない状態の場合は、制御インターフェイスが表示状態に入ってから経過した秒数も出力されます。

例20.46 ゲスト仮想マシンのインターフェイス状態を表示する方法

以下の例では、*guest1* 仮想マシンのインターフェイスの現在の状態を表示します。

```
# virsh domcontrol guest1
ok
```

20.18. QEMU 引数のドメイン XML への変換

virsh domxml-from-native コマンドは、既存の QEMU 引数セットを、libvirt が使用できる Domain XML 設定ファイルに変換する方法を提供します。このコマンドは、libvirt で管理できるように、コマンドラインから起動した既存の QEMU ゲストを変換する場合にのみ使用することを目的としています。したがって、ここで説明する方法を使用して、新規ゲストを作成することはできません。新しいゲストは、virsh、virt-install、または virt-manager を使用して作成する必要があります。詳細は、on the libvirt upstream website を参照してください。

手順20.3 QEMU ゲストを libvirt に変換する方法

1. 引数ファイル (ファイルタイプ***.args**) を使用して QEMU ゲストで起動します。この例では、*demo.args* と名前が付けられています。

```
$ cat demo.args
LC_ALL=C
PATH=/bin
HOME=/home/test
USER=test
LOGNAME=test /usr/bin/qemu -S -M pc -m 214 -smp 1 -nographic -monitor pty -no-acpi -
boot c -hda /dev/HostVG/QEMUGuest1 -net none -serial none -parallel none -usb
```

2. このファイルをドメインの XML ファイルに変換して、ゲストを libvirt が管理できるようにするには、次のコマンドを実行します。*qemu-guest1* は、ゲスト仮想マシンの名前に置き換え、*demo.args* は QEMU 引数ファイルのファイル名に置き換えることを忘れないでください。

```
# virsh domxml-from-native qemu-guest1 demo.args
```

このコマンドにより、*demo.args* ファイルが次のドメイン XML ファイルに変換されます。

図20.1 ゲスト仮想マシンの新規設定ファイル

```
<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>
```

20.19. VIRSH DUMP を使用したゲスト仮想マシンのコアのダンプファイルの作成

ゲスト仮想マシン (kdump および pvpanic に加えて) のトラブルシューティングを行う方法の 1 つは、**virsh dump *domain corefilepath* [--bypass-cache] {--live | --crash | --reset} [--verbose] [--memory-only] [--format=*format*]** コマンドを使用することです。これにより、ゲスト仮想マシンのコアを含むダンプファイルが作成され、クラッシュユーティリティーなどで解析できるようになります。

具体的には、**virsh dump** コマンドを実行すると、ゲスト仮想マシンのコアが、指定したコアファイルパスで指定されたファイルにダンプされます。ハイパーバイザーによっては、この動作に制限を加え、*corefilepath* パラメーターで指定したファイルおよびパスに対する適切なパーミッションをユー

ザーが手動で確保することが必要になる場合があります。このコマンドは、SR-IOV デバイスやその他のパススルー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、以下のような効果があります。

- **--bypass-cache** - 保存したファイルは、ホストファイルシステムキャッシュを迂回しません。ファイルの内容には影響を及ぼしません。このオプションを選択すると、ダンプ操作が遅くなる可能性があることに注意してください。
- **--live** は、ゲスト仮想マシンの実行を継続するためファイルを保存します。ゲスト仮想マシンは一時停止または停止しません。
- **--crash** は、ダンプファイルの保存時にゲスト仮想マシンを一時停止したままにするのではなく、クラッシュした状態にします。ゲスト仮想マシンは "Shut off" としてリスト表示され、理由は "Crashed" になります。
- **--reset** - ダンプファイルが正常に保存されると、ゲスト仮想マシンがリセットされます。
- **--verbose** は、ダンププロセスの進捗を表示します。
- **--memory-only** - このオプションを使用してダンプを実行すると、ダンプファイルが作成されます。このダンプファイルの内容には、ゲスト仮想マシンのメモリーと CPU 共通レジスターファイルのみが含まれます。このオプションは、完全ダンプの実行に失敗する場合に使用する必要があります。これは、ゲスト仮想マシンがライブマイグレーションできない場合 (パススルー PCI デバイスが原因) に発生することがあります。

  --format=*format* オプションを使用すると、メモリーのみのダンプを保存できます。使用できる形式は次のとおりです。

  - **elf** - デフォルトの非圧縮形式です。
  - **kdump-zlib** - zlib 圧縮した kdump 圧縮形式です。
  - **kdump-lzo** - LZO 圧縮による kdump 圧縮形式です。
  - **kdump-snappy** - Snappy 圧縮による kdump 圧縮形式です。

重要

crash ユーティリティーが、**virsh dump** コマンドのデフォルトのコアダンプファイル形式に対応しなくなりました。**crash** を使用して **virsh dump** が作成したコアダンプファイルを分析する場合は、**--memory-only** オプションを使用する必要があります。

また、Red Hat サポートケースに添付するコアダンプファイルを作成する場合は、**--memory-only** オプションを使用する必要があります。

virsh domjobinfo コマンドを使用すると、プロセス全体を監視できます。**virsh domjobabort** コマンドを使用すると、プロセスをキャンセルできます。

例20.47 virsh でダンプファイルを作成する方法

次の例では、*guest1* 仮想マシンのコアのダンプファイルを作成し、それを**core/file/path.file** ファイルに保存してから、ゲストをリセットします。このコマンドを使用する最も一般的なシナリオは、ゲスト仮想マシンが適切に動作していない場合です。

```
# virsh dump guest1 core/file/path.file --memory-only --reset
```

20.20. 仮想マシンの XML ダンプの作成 (設定ファイル)

virsh dumpxml コマンドは、ゲスト仮想マシンの XML 設定ファイルを返します。これは、必要に応じて使用、保存、または変更できます。

その後、XML ファイル (**guest.xml**) を使用して、ゲスト仮想マシンを再作成できます (「ゲスト仮想マシンの XML 設定の編集」 を参照)。この XML 設定ファイルを編集して、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デプロイしたりできます。

例20.48 ゲスト仮想マシンの XML ファイルを取得する方法

以下の例では、*guest1* 仮想マシンの XML 設定を取得し、これを *guest1.xml* ファイルに書き込んでから、プロセスが正常に完了したことを確認します。

```
# virsh dumpxml guest1 > guest1.xml
# cat guest1.xml
<domain type='kvm'>
  <name>guest1-rhel6-64</name>
  <uuid>b8d7388a-bbf2-db3a-e962-b97ca6e514bd</uuid>
  <memory>2097152</memory>
  <currentMemory>2097152</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
    <boot dev='hd'/>
  </os>
[...]
```

20.21. 設定ファイルから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できます。以前に作成したゲスト仮想マシンから XML をコピーするか、**virsh dumpxml** を使用します。

例20.49 XML ファイルからゲスト仮想マシンを作成する方法

以下の例では、既存の*guest1.xml* 設定ファイルから新しい仮想マシンを作成します。作業を開始する前に、このファイルが必要です。**virsh dumpxml** コマンドを使用してファイルを取得できます。手順は 例20.48「ゲスト仮想マシンの XML ファイルを取得する方法」 を参照してください。

```
# virsh create guest1.xml
```

20.22. ゲスト仮想マシンの XML 設定の編集

virsh edit コマンドを使用すると、指定したゲストのドメイン XML 設定ファイルを変更できます。このコマンドを実行すると、**$EDITOR** シェルパラメーターで指定したテキストエディターで XML ファイルが開きます (デフォルトでは **vi** に設定されています)。

例20.50 ゲスト仮想マシンの XML 設定を編集する方法

以下の例では、*guest1* 仮想マシンに関連付けられた XML 設定ファイルを、デフォルトのテキストエディターで開きます。

```
# virsh edit guest1
```

20.23. KVM ゲスト仮想マシンへの複合 PCI デバイスの追加

複合機能の PCI デバイスを KVM ゲスト仮想マシンに追加するには、次のコマンドを実行します。

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>** 要素に、**multifunction='on'** 属性を追加します。これにより、特定のマルチファンクション PCI デバイスに対して別の機能を使用できるようになります。

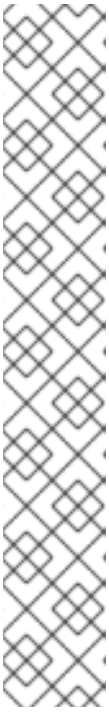
```
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none'/>
<source file='/var/lib/libvirt/images/rhel62-1.img'/>
<target dev='vda' bus='virtio'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'
multifunction='on'/>
</disk>
```

2 つの機能を持つ PCI デバイスの場合は、XML 設定ファイルを修正して、最初のデバイスと同じスロット番号と、別の機能番号 (**function='0x1'** など) を持つ 2 番目のデバイスを追加します。以下に例を示します。

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none'/>
<source file='/var/lib/libvirt/images/rhel62-1.img'/>
<target dev='vda' bus='virtio'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'
multifunction='on'/>
</disk>
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none'/>
<source file='/var/lib/libvirt/images/rhel62-2.img'/>
<target dev='vdb' bus='virtio'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x1'/>
</disk>
```

3. **lspci** コマンドを実行します。KVM ゲスト仮想マシンからの出力は、virtio ブロックデバイスを示しています。

```
$ 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 インターフェイスとの互換性のためにリアルモードで実行します。これにより、利用可能なメモリーの量が制限されます。これにより、SeaBIOS は、限られた数のディスクのみを処理できるようになりました。現在、対応しているディスク数は次のとおりです。

- virtio-scsi – 64
- virtio-blk – 4
- ahci/sata - 24 (6 ポートすべてが接続されている 4 つのコントローラー)
- usb-storage – 4

この問題を回避するには、仮想マシンに多数のディスクを割り当てる場合に、システムディスクの pci スロット番号が小さいことを確認してください。そのため、pci バスをスキャンすると、SeaBIOS が最初にそれを認識するようになります。ディスクごとのメモリーオーバーヘッドが小さいため、virtio-blk の代わりに virtio-scsi デバイスを使用することも推奨されます。

20.24. 指定したゲスト仮想マシンの CPU 統計の表示

virsh cpu-stats *domain* --total start count コマンドは、指定したゲスト仮想マシンの CPU 統計情報を提供します。デフォルトでは、すべての CPU の統計および合計が表示されます。**--total** オプションは、合計統計のみを表示します。**--count** オプションは、*count* CPU の統計のみを表示します。

例20.51 ゲスト仮想マシンの CPU 統計を生成する方法

以下の例では、*guest1* という名前のゲスト仮想マシンの CPU 統計情報を生成します。

```
# virsh cpu-stats guest1

CPU0:
 cpu_time         242.054322158 seconds
 vcpu_time        110.969228362 seconds
CPU1:
 cpu_time         170.450478364 seconds
 vcpu_time        106.889510980 seconds
CPU2:
 cpu_time         332.899774780 seconds
 vcpu_time        192.059921774 seconds
CPU3:
 cpu_time         163.451025019 seconds
 vcpu_time         88.008556137 seconds
Total:
 cpu_time         908.855600321 seconds
 user_time         22.110000000 seconds
 system_time       35.830000000 seconds
```

20.25. ゲストコンソールのスクリーンショットの撮り方

virsh screenshot *guestname* [imagefilepath] コマンドは、現在のゲスト仮想マシンコンソールのスク

リーンショットを取り、ファイルに保存します。ファイルパスが指定されていない場合は、スクリーンショットが現在のディレクトリーに保存されます。ハイパーバイザーがゲスト仮想マシンの複数のディスプレイに対応している場合は、**--*screen* screenID** オプションを使用してキャプチャーする画面を指定します。

例20.52 ゲストマシンのコンソールのスクリーンショットを撮る方法

以下の例では、*guest1* マシンのコンソールのスクリーンショットを撮り、**/home/username/pics/guest1-screen.png** として保存します。

```
# virsh screenshot guest1 /home/username/pics/guest1-screen.ppm
Screenshot saved to /home/username/pics/guest1-screen.ppm, with type of image/x-portable-pixmap
```

20.26. 指定したゲスト仮想マシンへのキーストロークの組み合わせの送信

virsh send-key *ドメイン* **--codeset --holdtime** *キーコード* コマンドを使用すると、特定のゲスト仮想マシンに、*キーコード* としてシーケンスを送信できます。各 *キーコード* は、数値または以下の対応する *コードセット* のシンボリック名になります。

--holdtime を指定すると、各キーストロークは指定した時間 (ミリ秒単位) 保持されます。**--codeset**ではコードセットを指定できます。デフォルトは **Linux** ですが、以下のオプションを使用できます。

- **linux** - このオプションを選択すると、シンボリック名が対応する Linux キー定数マクロ名に一致するようになります。数値は、Linux 汎用入力イベントサブシステムが提供するものになります。
- **xt** - XT キーボードコントローラーで定義する値を送信します。シンボリック名は提供されていません。
- **atset1** - 数値は、AT キーボードコントローラー、set1 (XT 互換セット) で定義されるものです。atset1 から拡張されたキーコードは、XT コードセットの拡張キーコードとは異なる場合があります。シンボリック名は記載されていません。
- **atset2** - 数値は、AT キーボードコントローラー、set 2 で定義されるものです。シンボリック名は記載されていません。
- **atset3** - 数値は、AT キーボードコントローラー、set 3 (PS/2 互換) で定義されるものです。シンボリック名は記載されていません。
- **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 の鍵定数マクロ名と一致します。
- **xt_kbd** - 数値は、Linux KBD デバイスーで定義されるものです。これは、元の XT コードセットのバリアントですが、拡張キーコードのエンコードが異なります。シンボリック名は記載されていません。
- **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 鍵定数マクロ名と一致します。
- **usb** - 数値は、キーボード入力の USB HID で定義されるものです。シンボリック名は記載されていません。
- **rfb** - 数値は、raw キーコードを送信するために RFB 拡張で定義されるものです。これは XT

コードセットのバリアントですが、拡張キーコードでは、最初のバイトの上位ビットではなく、2 番目のビットセットの下位ビットが使用されます。シンボリック名は記載されていません。

例20.53 ゲスト仮想マシンにキーストロークの組み合わせを送信する方法

以下の例では、Linux エンコードの **Left Ctrl**、**Left Alt**、および **Delete** を*guest1* 仮想マシンに送信し、それらを 1 秒間保持します。これらの鍵はすべて同時に送信されるため、ゲストがランダムな順序で受信できます。

```
# virsh send-key guest1 --codeset Linux --holdtime 1000 KEY_LEFTCTRL KEY_LEFTALT
KEY_DELETE
```

注記

複数の *キーコード*を指定すると、すべてがゲスト仮想マシンに同時に送信されるため、順不同で受信されることがあります。別のキーコードが必要な場合は、シーケンスを送信する順序で **virsh send-key** コマンドを複数回実行する必要があります。

20.27. ホストマシンの管理

本セクションには、ホストシステムを管理するために必要なコマンド (コマンドにより*node* と呼ばれる) が記載されています。

20.27.1. ホスト情報の表示

virsh nodeinfo コマンドは、モデル番号、CPU 数、CPU の種類、物理メモリーのサイズなど、ホストに関する基本情報を表示します。出力は、**virNodeInfo** 構造に対応します。具体的には、"CPU socket(s)" フィールドは NUMA セルごとの CPU ソケット数を示します。

例20.54 ホストマシンの情報を表示する方法

以下の例では、ホストに関する情報を取得します。

```
$ 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
```

20.27.2. NUMA パラメーターの設定

virsh numatune コマンドは、指定したゲスト仮想マシンの NUMA パラメーターを設定または取得できます。ゲスト仮想マシンの設定 XML ファイル内では、このパラメーターは **<numatune>** 要素にネストされています。フラグを使用しないと、現在の設定のみが表示されます。**numatune *domain*** コマンドには、指定したゲスト仮想マシン名が必要で、次の引数を使用できます。

- **--mode** - モードは、**strict**、**インターリーブ**、または **推奨** のいずれかに設定できます。稼働中のドメインでは、ゲスト仮想マシンが **strict** モードで起動しない限り、モードを変更できません。
- **--nodeset** には、ゲスト仮想マシンの実行にホスト物理マシンが使用する NUMA ノードのリストが含まれています。このリストには、それぞれがコンマで区切られたノードが含まれ、ノード範囲に使用されるダッシュ**-**と、ノードの除外に使用されるカレット**^** も含まれています。
- インスタンスごとに、以下の 3 つのフラグのいずれかを使用できます。
  - **--config** は、永続ゲスト仮想マシンの次回の起動時に影響を及ぼします。
  - **--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を有効にします。

例20.55 ゲスト仮想マシンの NUMA パラメーターを設定する方法

以下の例では、実行中の*guest1* 仮想マシンのノード 0、2、および 3 の NUMA モードを **strict** に設定します。

```
# virsh numatune guest1 --mode strict --nodeset 0,2-3 --live
```

このコマンドを実行すると、guest1 の実行中の設定が、その XML ファイルの以下の設定に変更されます。

```
<numatune>
      <memory mode='strict' nodeset='0,2-3'/>
</numatune>
```

20.27.3. NUMA セルの空きメモリー量の表示

virsh freecell コマンドは、指定された NUMA セル内に、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、指定したオプションに応じて、マシンで利用可能なメモリーを、3 つの異なるディスプレイのいずれかに表示します。 指定されたセル。

例20.56 仮想マシンおよび NUMA セルのメモリープロパティーを表示する方法

以下のコマンドは、すべてのセルで利用可能なメモリーの合計量を表示します。

```
# virsh freecell
Total: 684096 KiB
```

個々のセルに使用可能なメモリー容量も表示するには、**--all** オプションを使用します。

```
# virsh freecell --all
   0:     804676 KiB
--------------------
Total:     804676 KiB
```

指定したセルの個々のメモリー容量を表示するには、**--cellno** オプションを実行します。

```
# virsh freecell --cellno 0
0: 772496 KiB
```

20.27.4. CPU リストの表示

virsh nodecpumap コマンドは、ホストマシンで利用可能な CPU の数と、現在オンラインになっている数を表示します。

例20.57 ホストで利用可能な CPU の数を表示する方法

以下の例では、ホストで利用可能な CPU の数を表示します。

```
# virsh nodecpumap
  CPUs present: 4
  CPUs online: 1
  CPU map: y
```

20.27.5. CPU 統計の表示

virsh nodecpustats [*cpu_number*] [--percent] コマンドは、ホストコンピューターの CPU 負荷ステータスに関する統計情報を表示します。CPU を指定した場合は、指定した CPU のみが統計情報となります。**percent** オプションを指定すると、1 秒間隔で記録された各タイプの CPU 統計の割合がコマンドで表示されます。

例20.58 CPU 使用率の統計情報を表示する方法

以下の例は、ホスト CPU の負荷に関する一般的な統計を返します。

```
# virsh nodecpustats
user:                1056442260000000
system:               401675280000000
idle:                7549613380000000
iowait:                94593570000000
```

この例では、CPU 番号 2 の統計をパーセンテージで表示します。

```
# virsh nodecpustats 2 --percent
usage:           2.0%
user:            1.0%
system:          1.0%
idle:           98.0%
iowait:          0.0%
```

20.27.6. デバイスの管理

20.27.6.1. virsh を使用したデバイスの接続および更新

ストレージデバイスの接続の詳細は、「ゲストへのストレージデバイスの追加」 を参照してください。

手順20.4 ゲスト仮想マシンが使用するホットプラグ USB デバイス

USB デバイスは、ホットプラグにより実行している仮想マシンに接続するか、ゲストのシャットダウン中に接続できます。ゲストで使用するデバイスをホストマシンに接続している必要があります。

1. 次のコマンドを実行して、接続する 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、論理名 (**usb_device.xml** など) を付けます。検索で表示されたとおりに、ベンダーと製品 ID 番号 (16 進数) をコピーします。図20.2「USB デバイスの XML スニペット」 に示すように、この情報を XML ファイルに追加します。このファイルは次の手順で必要となるため、名前を覚えておいてください。

   図20.2 USB デバイスの XML スニペット

```
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x17ef'/>
    <product id='0x480f'/>
  </source>
</hostdev>
```

3. 次のコマンドを実行して、デバイスを接続します。コマンドを実行したら、*guest1* を仮想マシンの名前に置き換え、*usb_device.xml* を、前の手順で作成したデバイスのベンダーと製品 ID が含まれる XML ファイルの名前に置き換えます。次回のシステムの再起動時に変更が反映されるようにするには、**--config** 引数を使用します。現在のゲスト仮想マシンで変更を有効にするには、**--current** 引数を使用します。追加の引数については、virsh の man ページを参照してください。

```
# virsh attach-device guest1 --file usb_device.xml --config
```

例20.59 ゲスト仮想マシンからデバイスをホットアンプラグする方法

以下の例では、*usb_device1.xml* ファイル で設定した USB デバイスを、 *guest1* 仮想マシンから切り離します。

```
# virsh detach-device guest1 --file usb_device.xml
```

20.27.6.2. インターフェイスデバイスの接続

virsh attach-interface *domain type source* [<target>] [<mac>] [<script>] [<model>] [<inbound>] [<outbound>] [--config] [--live] [--current] コマンドには、以下の引数を使用できます。

- **--type** - インターフェイスタイプの設定を許可します。
- **--source** - ネットワークインターフェイスの送信元を設定できます。
- **--live** - ゲスト仮想マシンの設定を実行してその値を取得します。

- **--config** - システムの次回起動時に有効になります。
- **--current** - 現在の設定に従って値を取得します。
- **--target** - ゲスト仮想マシン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イスの MAC アドレスを指定します。
- **--script** - このオプションは、デフォルトのパスの代わりに、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します。
- **--model** - このオプションを使用して、モデルタイプを指定します。
- **--inbound** - インターフェイスの着信帯域幅を制御します。指定できる値は、**average**、**peak**、および **burst** です。
- **--outbound** - インターフェイスのアウトバウンド帯域幅を制御します。指定できる値は、**average**、**peak**、および **burst** です。

注記

平均値およびピーク値は 1 秒あたりのキロバイト数で表されます。バースト値は、ネットワーク XML アップストリームのドキュメント の説明に従って、ピーク速度における単一バーストのキロバイト数で表されます。

type は、物理ネットワークデバイスを示す **network**、またはデバイスへの **bridge** を示すブリッジのいずれかです。*source* はデバイスのソースです。接続したデバイスを削除する場合は、**virsh detach-device** コマンドを実行します。

例20.60 デバイスをゲスト仮想マシンに接続する方法

次の例では、*networkw* ネットワークデバイスを *guest1* 仮想マシンに割り当てます。インターフェイスモデルは、**virtio** としてゲストに表示されます。

```
# virsh attach-interface guest1 networkw --model virtio
```

20.27.6.3. CDROM のメディアの変更

virsh change-media コマンドは、CD-ROM のメディアを別のソースまたはフォーマットに変更します。このコマンドには、以下の引数が使用されます。これらの引数の例や説明については、man ページも併せて参照してください。

- **--path** - ディスクデバイスの完全修飾パスまたはターゲットが含まれる文字列
- **--source** - メディアのソースが含まれる文字列
- **--eject** - メディアをイジェクトします。
- **--insert** - メディアを挿入します。
- **--update** - メディアを更新します。
- **--current** - **--live** および **--config** のいずれかまたは両方を指定できます。これは、ハイパーバイザードライバーの実装に依存します。

- **--live** - 実行中のゲスト仮想マシンのライブ設定を変更します。
- **--config** - 永続的な設定を変更し、システムの次回起動時に影響が確認されます。
- **--force** - メディアの変更を強制します。

20.27.7. ノードメモリーパラメーターの設定および表示

virsh node-memory-tune [shm-pages-to-scan] [shm-sleep-milisecs] [shm-merge-across-nodes] コマンドが表示され、ノードメモリーパラメーターを設定できるようになります。このコマンドでは、以下のパラメーターを設定できます。

- **--shm-pages-to-scan** - Kernel Samepage Merging (KSM) サービスがスリープ状態に切り替わる前にスキャンするページ数を設定します。
- **--shm-sleep-milisecs** - KSM が次回のスキャンの前にスリープするミリ秒数を設定します。
- **--shm-merge-across-nodes** - 別の NUMA ノードのページをマージできるかどうかを指定します。

例20.61 NUMA ノード間でメモリーページをマージする方法

以下の例では、すべての NUMA ノードのメモリーページをすべてマージします。

```
# virsh node-memory-tune --shm-merge-across-nodes 1
```

20.27.8. ホストでのデバイスのリスト表示

virsh nodedev-list *--cap* --tree コマンドは、**libvirt** サービスに認識されている、ホストで使用可能なすべてのデバイスを一覧表示します。*--cap* は、機能タイプでリストをフィルタリングするために使用されます。各タイプはコンマで区切られ、**--tree** と一緒に使用することはできません。引数**--tree** を使用して、出力をツリー構造に置きます。

例20.62 ホストで利用可能なデバイスを表示する方法

次の例では、ホストで利用可能なデバイスをツリー形式でリスト表示します。このリストは切り捨てられていることに注意してください。

```
# virsh nodedev-list --tree
computer
  |
  +- net_lo_00_00_00_00_00_00
  +- net_macvtap0_52_54_00_12_fe_50
  +- net_tun0
  +- net_virbr0_nic_52_54_00_03_7d_cb
  +- pci_0000_00_00_0
  +- pci_0000_00_02_0
  +- pci_0000_00_16_0
  +- pci_0000_00_19_0
  |   |
  |   +- net_eth0_f0_de_f1_3a_35_4f
 [...]
```

この例では、ホストで利用可能な SCSI デバイスのリストを表示します。

```
# virsh nodedev-list --cap scsi
scsi_0_0_0_0
```

20.27.9. ホストマシンでのデバイスの作成

virsh nodedev-create *file* コマンドを使用すると、ホストの物理マシンにデバイスを作成し、ゲスト仮想マシンに割り当てることができます。**libvirt** は、使用可能なホストノードを自動的に検出しますが、このコマンドを使用すると、**libvirt** が検出しなかったハードウェアを登録できます。指定するファイルには、ホストデバイスのトップレベル**<デバイス>**の XML 説明が含まれている必要があります。このようなファイルの例は、例20.65「デバイスの XML ファイルを取得する方法」 を参照してください。

例20.63 XML ファイルからデバイスを作成する方法

この例では、PCI デバイス用の XML ファイルを作成し、それを **scsi_host2.xml** として保存しています。次のコマンドは、このデバイスをゲストに割り当てることができます。

```
# virsh nodedev-create scsi_host2.xml
```

20.27.10. デバイスの削除

virsh nodedev-destroy コマンドは、ホストコンピューターからデバイスを削除します。virsh ノードデバイスドライバーは永続的な設定をサポートしないため、ホストマシンを再起動すると、デバイスが再び使用可能になることに注意してください。

また、割り当てが異なると、デバイスは別のバックエンドドライバー (vfio、kvm) にバインドされることが想定されます。**--driver** 引数を使用すると、目的のバックエンドドライバーを指定できます。

例20.64 ホストの物理マシンからデバイスを削除する方法

次の例では、ホストマシンから *scsi_host2* という名前の SCSI デバイスを削除します。

```
# virsh nodedev-destroy scsi_host2
```

20.27.11. デバイス設定の収集

virsh nodedev-dumpxml *device* コマンドは、指定したホストデバイスの XML 表現を出力します。出力には、デバイス名、デバイスが接続しているバス、ベンダー、プロダクト ID、機能などの情報と、**libvirt** が使用できる情報が含まれます。引数の *デバイス* は、デバイス名にすることも、WWNN、WWPN 形式の WWN の組み合わせにすることもできます (HBA のみ)。

例20.65 デバイスの XML ファイルを取得する方法

次の例では、*scsi_host2* と識別される SCSI デバイスの XML ファイルを取得します。名前は、 virsh nodedev-list コマンドを使用して取得しました。

```
# virsh nodedev-dumpxml scsi_host2 <device> <name>scsi_host2</name>
<parent>scsi_host1</parent> <capability type='scsi_host'> <capability type='fc_host'>
```

```
<wwnn>2001001b32a9da5b</wwnn> <wwpn>2101001b32a9da5b</wwpn> </capability>
</capability> </device>
```

20.27.12. デバイスのリセットのトリガー

virsh nodedev-reset *device* コマンドを実行すると、指定したデバイスのデバイスリセットがトリガーされます。ゲスト仮想マシンパススルーまたはホスト物理マシンの間でノードデバイスを転送する前に、このコマンドを実行すると役立ちます。**libvirt** は必要に応じてこのアクションを自動的に実行しますが、このコマンドは必要に応じて明示的なリセットを許可します。

例20.66 ゲスト仮想マシンのデバイスをリセットする方法

以下の例では、*scsi_host2* という名前のゲスト仮想マシンのデバイスをリセットします。

```
# virsh nodedev-reset scsi_host2
```

20.28. ゲストの仮想マシン情報の取得

20.28.1. ゲスト仮想マシンのドメイン ID の取得

virsh domid コマンドは、ゲスト仮想マシンの ID を返します。これは、ゲストが起動または再起動するたびに変更することに注意してください。このコマンドには、仮想マシンの名前または仮想マシンの UUID のいずれかが必要です。

例20.67 ゲスト仮想マシンのドメイン ID を取得する方法

以下の例では、*guest1* という名前のゲスト仮想マシンのドメイン ID を取得します。

```
# virsh domid guest1
8
```

domid は、シャットダウン状態のゲスト仮想マシン に **-** を返します。仮想マシンがシャットダウンしていることを確認する場合は、**virsh list --all** コマンドを実行できます。

20.28.2. ゲスト仮想マシンのドメイン名の取得

virsh domname は、ID または UUID を指定したゲスト仮想マシンの名前を返します。ゲストが起動するたびに ID が変更されることに注意してください。

例20.68 仮想マシンの ID を取得する方法

以下の例では、ID が *8* であるゲスト仮想マシンの名前を取得します。

```
# virsh domname 8
guest1
```

20.28.3. ゲスト仮想マシンの UUID の取得

virsh domuuid は、指定したゲスト仮想マシンの UUID または ID の *Universally Unique Identifier* を返します。

例20.69 ゲスト仮想マシンの UUID を表示する方法

以下の例では、*guest1* という名前のゲスト仮想マシンの UUID を取得します。

```
# virsh domuuid guest1
r5b2-mySQL01 4a4c59a7-ee3f-c781-96e4-288f2862f011
```

20.28.4. ゲスト仮想マシン情報の表示

virsh dominfo コマンドは、仮想マシンの名前、ID、または UUID を指定して、そのゲスト仮想マシンに関する情報を表示します。仮想マシンの起動ごとに ID が変更されることに注意してください。

例20.70 ゲスト仮想マシンの一般的な詳細を表示する方法

以下の例では、*guest1* という名前のゲスト仮想マシンの概要を表示します。

```
# virsh dominfo guest1
Id:             8
Name:           guest1
UUID:           90e0d63e-d5c1-4735-91f6-20a32ca22c48
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       32.6s
Max memory:     1048576 KiB
Used memory:    1048576 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c552,c818 (enforcing)
```

20.29. ストレージプールコマンド

libvirt を使用すると、仮想マシン内のデバイスとして表示されるストレージボリュームを提供するために使用されるファイル、raw パーティション、ドメイン固有の形式など、さまざまなストレージソリューションを管理できます。詳細は、libvirt アップストリームページ を参照してください。ストレージプールを管理するコマンドの多くは、ゲスト仮想マシンに使用されるコマンドと似ています。

20.29.1. ストレージプール XML の検索

virsh find-storage-pool-sources *type* コマンドは、検出された 1 つのソースのすべてのストレージプールを説明する XML を表示します。タイプには、netfs、disk、dir、fs、iscsi、logical、および gluster が含まれます。すべてのタイプがストレージバックエンドドライバーに対応しており、さらに

多くのタイプが利用可能であることに注意してください (詳細については、man ページを参照してください)。*--srcSpec* を使用してテンプレートソースの XML ファイルを提供することで、プールのクエリーをさらに制限することもできます。

例20.71 利用可能なストレージプールの XML 設定のリストを表示する方法

以下の例では、システムで利用可能なすべての論理ストレージプールの XML 設定を出力します。

```
# virsh find-storage-pool-sources logical
<sources>
  <source>
    <device path='/dev/mapper/luks-7a6bfc59-e7ed-4666-a2ed-6dcbff287149'/>
    <name>RHEL_dhcp-2-157</name>
    <format type='lvm2'/>
  </source>
</sources>
```

20.29.2. ストレージプールの検索

virsh find-storage-pool-sources-as *type* コマンドは、特定のタイプを指定して、潜在的なストレージプールのソースを見つけます。タイプには、netfs、disk、dir、fs、iscsi、logical、および gluster が含まれます。すべてのタイプがストレージバックエンドドライバーに対応しており、さらに多くのタイプが利用可能であることに注意してください (詳細については、man ページを参照してください)。また、オプションの引数 *host*、*port*、*initiator* も指定できます。このオプションのそれぞれが、クエリーを受けるものを決定します。

例20.72 潜在的なストレージプールソースを見つける方法

次の例では、指定したホストマシンのディスクベースのストレージプールを検索します。ホスト名がわからない場合は、**virsh hostname** コマンドを最初に実行してください。

```
# virsh find-storage-pool-sources-as disk --host myhost.example.com
```

20.29.3. ストレージプール情報のリスト表示

virsh pool-info *pool* コマンドは、指定したストレージプールオブジェクトの基本情報をリスト表示します。このコマンドには、ストレージプールの名前または UUID が必要です。この情報を取得するには、**pool-list** コマンドを使用します。

例20.73 ストレージプールの情報を取得する方法

以下の例は、*vdisk* という名前のストレージプールに関する情報を取得します。

```
# virsh pool-info vdisk

Name:           vdisk
UUID:
State:          running
Persistent:     yes
Autostart:      no
Capacity:       125 GB
```

```
Allocation:     0.00
Available:      125 GB
```

20.29.4. 利用可能なストレージプールのリスト表示

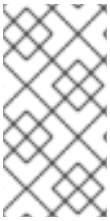
virsh pool-list は、libvirt が認識しているすべてのストレージプールオブジェクトをリスト表示します。デフォルトでは、アクティブなプールのみがリスト表示されます。**--inactive** 引数を使用すると非アクティブなプールのみがリスト表示され、**--all** 引数を使用するとすべてのストレージプールがリスト表示されます。このコマンドは、以下のオプション引数を取ります。これにより、検索結果をフィルターにかけます。

- **--inactive** - 非アクティブなストレージプールのリストを表示します。
- **--all** - アクティブなストレージプールと非アクティブなストレージプールの両方をリスト表示します。
- **--persistent** - 永続ストレージプールのリストを表示します。
- **--transient** - 一時的なストレージプールのリストを表示します。
- **--autostart** - 自動起動が有効になっているストレージプールのリストを表示します。
- **--no-autostart** - 自動起動が無効になっているストレージプールのリストを表示します。
- **--type *type*** - 指定したタイプのプールのみをリスト表示します。
- **--details** - ストレージプールの拡張内容のリストを表示します。

上記の引数に加えて、リストのコンテンツをフィルタリングするために使用できるフィルタリングフラグのセットがいくつかあります。**--persistent** は、リストを永続プールに制限します。**--transient** はリストを一時的なプールに制限し、**--autostart** はリストをプールの自動起動に制限し、最後に **--no-autostart** は、自動起動が無効になっているストレージプールにリストを制限します。

*--type*を必要とするすべてのストレージプールコマンドでは、プールタイプをコンマで区切る必要があります。有効なプールタイプには、**dir**、**fs**、**netfs**、**logical**、**disk**、**iscsi**、**scsi**、**mpath**、**rbd**、**sheepdog**、および **gluster** が含まれます。

--details オプションは、**virsh** に対して、プール永続性と利用可能な容量関連の情報を追加で表示するように指示します。



注記

このコマンドを古いサーバーで使用すると、固有の競合で一連の API 呼び出しの使用が強制されます。ここでは、リストの収集中にプールの状態が呼び出し間で変更した場合に、プールがリストに表示されなかったり、複数回表示されたりすることがあります。ただし、新しいサーバーにはこの問題はありません。

例20.74 すべてのストレージプールのリストを表示する方法

この例では、アクティブおよび非アクティブの両方のストレージプールのリストを表示します。

```
# virsh pool-list --all
```

```
Name                 State      Autostart
-----------------------------------------
default              active     yes
vdisk                active     no
```

20.29.5. ストレージプールリストの更新

virsh pool-refresh *pool* コマンドは、ストレージプールに含まれるストレージボリュームのリストを更新します。

例20.75 ストレージプール内のストレージボリュームのリストを更新する方法

以下の例では、*vdisk* という名前のストレージボリュームのリストを更新します。

```
# virsh pool-refresh vdisk

Pool vdisk refreshed
```

20.29.6. ストレージプールの作成、定義、および起動

20.29.6.1. ストレージプールの構築

virsh pool-build *pool* コマンドは、コマンドに指定した名前を使用してストレージプールを構築します。オプションの引数 **--overwrite** および **--no-overwrite** は、FS ストレージプール、またはディスクタイプベースのストレージプール、または論理タイプベースのストレージプールでのみ使用できます。[--overwrite] または [--no-overwrite] がなく、使用されるプールが FS の場合は、タイプが実際にディレクトリーベースであることを前提としています。プール名のほかに、ストレージプール UUID も使用することが可能です。

--no-overwrite を指定すると、プローブにより、ファイルシステムがターゲットデバイスにすでに存在するかどうかを判断したり、存在する場合はエラーを返すか、存在しない場合は **mkfs** を使用してターゲットデバイスをフォーマットします。**--overwrite** を指定した場合は、**mkfs** コマンドが実行され、ターゲットデバイスに存在するデータがすべて上書きされます。

例20.76 ストレージプールを構築する方法

以下の例では、*vdisk* という名前のディスクベースのストレージプールを作成します。

```
# virsh pool-build vdisk

Pool vdisk built
```

20.29.6.2. XML ファイルからのストレージプールの定義

virsh pool-define *file* コマンドは、XML *file* からストレージプールオブジェクトを作成するが、起動しません。

例20.77 XML ファイルからストレージプールを定義する方法

この例では、ストレージプールの設定を含む XML ファイルが作成されていることを前提としています。以下に例を示します。

```
<pool type="dir">
 <name>vdisk</name>
 <target>
        <path>/var/lib/libvirt/images</path>
 </target>
</pool>
```

以下のコマンドは、XML ファイル (この例では *vdisk.xml*) からディレクトリータイプのストレージプールを構築します。

```
# virsh pool-define vdisk.xml

Pool vdisk defined
```

ストレージプールが定義されたことを確認するには、例20.74「すべてのストレージプールのリストを表示する方法」 に示されているように **virsh pool-list --all** コマンドを実行します。ただし、コマンドを実行すると、プールが起動していないとステータスが inactive と表示されます。ストレージプールを起動する方法は、例20.81「ストレージプールを起動する方法」を参照してください。

20.29.6.3. ストレージプールの作成

virsh pool-create *file* コマンドは、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て起動します。

例20.78 XML ファイルからストレージプールを作成する方法

この例では、ストレージプールの設定を含む XML ファイルがすでに作成されていることを前提としています。以下に例を示します。

```
<pool type="dir">
 <name>vdisk</name>
 <target>
        <path>/var/lib/libvirt/images</path>
 </target>
</pool>
```

以下の例では、XML ファイル (この例では *vdisk.xml*) に基づいて、ディレクトリータイプのストレージプールを構築します。

```
# virsh pool-create vdisk.xml

Pool vdisk created
```

ストレージプールが作成されたことを確認するには、例20.74「すべてのストレージプールのリストを表示する方法」 に示されているように **virsh pool-list --all** コマンドを実行します。ただし、コマンドを実行すると、プールが起動していないとステータスが inactive と表示されます。ストレージプールを起動する方法は、例20.81「ストレージプールを起動する方法」を参照してください。

20.29.6.4. ストレージプールの作成

virsh pool-create-as *name* コマンドは、指定した raw パラメーターからプールオブジェクト名を作成して開始します。このコマンドには、以下のオプションがあります。

- **--print-xml** - XML ファイルの内容を表示しますが、そこからストレージプールを定義または作成しません。
- **--type** *type* は、ストレージプールタイプを定義します。使用できるタイプは、「利用可能なストレージプールのリスト表示」 を参照してください。
- **--source-host** *hostname* - 基となるストレージのソースホスト物理マシン
- **--source-path** *path* - 基となるストレージの場所
- **--source-dev** *path* - 基となるストレージのデバイス
- **--source-name** *name* - ソースの基となるストレージの名前
- **--source-format** *format* - ソースの基となるストレージの形式
- **--target** *path* - 基となるストレージのターゲット

例20.79 ストレージプールを作成して起動する方法

以下の例では、**/mnt** ディレクトリーに、*vdisk* という名前のストレージプールを作成して起動します。

```
# virsh pool-create-as --name vdisk --type dir --target /mnt

Pool vdisk created
```

20.29.6.5. ストレージプールの定義

virsh pool-define-as *<name>* コマンドは、指定した 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、起動しません。このコマンドには、以下のオプションが使用できます。

- **--print-xml** - XML ファイルの内容を表示しますが、そこからストレージプールを定義または作成しません。
- **--type** *type* は、ストレージプールタイプを定義します。使用できるタイプは、「利用可能なストレージプールのリスト表示」 を参照してください。
- **--source-host** *hostname* - 基となるストレージのソースホスト物理マシン
- **--source-path** *path* - 基となるストレージの場所
- **--source-dev** *devicename* - 基となるストレージのデバイス
- **--source-name** *sourcename* - 基となるストレージの名前
- **--source-format** *format* - 基となるストレージの形式
- **--target** *targetname* - 基となるストレージのターゲット

--print-xml を指定すると、プールを作成または定義せずに、プールオブジェクトの XML を出力します。それ以外の場合は、プールを構築するためには指定の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、プールタイプをコンマで区切る必要があります。有効なプールタイプには、**dir**、**fs**、**netfs**、**logical**、**disk**、**iscsi**、**scsi**、**mpath**、**rbd**、**sheepdog**、および **gluster** が含まれます。

例20.80 ストレージプールを定義する方法

以下の例では、*vdisk* という名前のストレージプールを定義しますが、起動はしません。このコマンドを実行したら、**virsh pool-start** コマンドを使用してストレージプールをアクティブにします。

```
# virsh pool-define-as --name vdisk --type dir --target /mnt

Pool vdisk defined
```

20.29.6.6. ストレージプールの起動

virsh pool-start *pool* コマンドは、指定したストレージプールを開始します。このストレージプールは以前定義されていましたが、非アクティブです。このコマンドは、ストレージプールの UUID とプールの名前を使用することもできます。

例20.81 ストレージプールを起動する方法

以下の例では、例20.78「XML ファイルからストレージプールを作成する方法」 で作成した *vdisk* ストレージプールを起動します。

```
# virsh pool-start vdisk

Pool vdisk started
```

プールが開始したことを確認するには、例20.74「すべてのストレージプールのリストを表示する方法」 に示すように、**virsh pool-list --all** コマンドを実行し、ステータスが有効であることを確認します。

20.29.6.7. ストレージプールの自動起動

virsh pool-autostart *pool* コマンドを使用すると、システムの起動時にストレージプールが自動的に起動します。このコマンドには、プール名または UUID が必要です。**pool-autostart** コマンドを無効にするには、コマンドに **--disable** 引数を使用します。

例20.82 ストレージプールの自動起動方法

以下の例では、例20.78「XML ファイルからストレージプールを作成する方法」 で作成した *vdisk* ストレージプールを自動起動します。

```
# virsh pool-autostart vdisk

Pool vdisk autostarted
```

20.29.7. ストレージプールの停止および削除

virsh pool-destroy *pool* コマンドは、ストレージプールを停止します。停止すると、libvirt はプールを管理しなくなりますが、プールに含まれる生データは変更されず、後で **pool-create** コマンドを使用して復元できます。

例20.83 ストレージプールを停止する方法

以下の例では、例20.78「XML ファイルからストレージプールを作成する方法」 で作成した *vdisk* ストレージプールを停止します。

```
# virsh pool-destroy vdisk

Pool vdisk destroyed
```

virsh pool-delete *pool* コマンドは、指定したストレージプールが使用しているリソースを破壊します。この操作は復元できず、元に戻せないことに注意してください。ただし、プール構造はこのコマンドの後に依然として存在し、新しいストレージボリュームの作成を受け入れる準備ができています。

例20.84 ストレージプールを削除する方法

以下の例では、例20.78「XML ファイルからストレージプールを作成する方法」 に組み込まれている*vdisk* ストレージプールを削除します。

```
# virsh pool-delete vdisk

Pool vdisk deleted
```

virsh pool-undefine *pool* コマンドは、非アクティブなプールの設定を定義します。

例20.85 ストレージプールの定義を解除する方法

以下の例では、例20.78「XML ファイルからストレージプールを作成する方法」 で作成した*vdisk* ストレージプールの定義を解除します。これにより、ストレージプールが一時的になります。

```
# virsh pool-undefine vdisk

Pool vdisk undefined
```

20.29.8. プール用の XML ダンプファイルの作成

virsh pool-dumpxml *pool* コマンドは、指定されたストレージプールオブジェクトの XML 情報を返します。オプション **--inactive** を使用すると、現在のプール設定の代わりに、次回のプール開始時に使用される設定がダンプされます。

例20.86 ストレージプールの設定を取得する方法

以下の例では、例20.78「XML ファイルからストレージプールを作成する方法」 で作成した *vdisk* ストレージプールの設定を取得します。コマンドを実行すると、設定ファイルはターミナルで開きます。

```
# virsh pool-dumpxml vdisk
<pool type="dir">
 <name>vdisk</name>
 <target>
        <path>/var/lib/libvirt/images</path>
 </target>
</pool>
```

20.29.9. ストレージプールの設定ファイルの編集

pool-edit *pool* コマンドは、指定したストレージプールの XML 設定ファイルを開いて編集します。

この方法は、適用する前にエラーをチェックするため、XML 設定ファイルの編集に使用すべき唯一の方法です。

例20.87 ストレージプールの設定を編集する方法

以下の例では、例20.78「XML ファイルからストレージプールを作成する方法」 で作成した *vdisk* ストレージプールの設定を編集します。コマンドを実行すると、設定ファイルがデフォルトエディターで開きます。

```
# virsh pool-edit vdisk
<pool type="dir">
 <name>vdisk</name>
 <target>
        <path>/var/lib/libvirt/images</path>
 </target>
</pool>
```

20.30. ストレージボリュームコマンド

本セクションでは、ストレージボリュームの作成、削除、および管理を行うコマンドを説明します。ストレージボリュームを作成するには、少なくとも 1 つのストレージプールが必要です。ストレージプールの作成方法の例は、例20.78「XML ファイルからストレージプールを作成する方法」 を参照してください。ストレージプールの詳細は、「ストレージプールの使用」 を参照してください。ストレージボリュームの詳細は、「ストレージボリュームの使用」 を参照してください。

20.30.1. ストレージボリュームの作成

virsh vol-create-from *pool file vol* コマンドは、別のボリュームを入力として使用してボリュームを作成します。このコマンドには、ストレージプール名またはストレージプール UUID のいずれかが必要で、次のパラメーターとオプションが必要です。

- **--pool *string*** - 必須 - ストレージボリュームに割り当てるストレージプールまたはストレージプールの UUID の名前が含まれます。このストレージプールは、この新規ストレージボリュームのベースに使用するストレージボリュームに関連付けられたものと同じストレージプールである必要はありません。
- **--file *string*** - 必須 - ストレージボリュームのパラメーターを含む XML ファイルの名前が含まれます。

- **--vol *string*** - 必須 - この新しいストレージボリュームの基となるストレージボリュームの名前が含まれます。
- **--inputpool *string*** - 任意 - 新しいストレージボリュームの入力として使用するストレージボリュームに関連付けられたストレージプールに名前を付けることができます。
- **--prealloc-metadata** - 任意 - 新しいストレージボリュームに、(完全割り当てではなく qcow2 の) メタデータを事前に割り当てます。

例については、「ストレージボリュームの作成」 を参照してください。

20.30.2. パラメーターからのストレージボリュームの作成

virsh vol-create-as *pool name capacity* コマンドは、引数のセットからボリュームを作成します。*pool* 引数は、ボリュームを作成するストレージプールの名前または UUID を含みます。このコマンドには、以下の必須パラメーターおよびオプションがあります。

- **[--pool] *string*** - 必須 - 関連付けられたストレージプールの名前が含まれます。
- **[--name] *string*** - 必須 - 新しいストレージボリュームの名前が含まれます。
- **[--capacity] *string*** - 必須 - ストレージボリュームのサイズで、整数で表されます。指定がない場合は、デフォルトはバイトになります。バイト、キロバイト、メガバイト、ギガバイト、およびテラバイトには、それぞれ接尾辞 b、k、M、G、T を使用します。
- **--allocation *string*** - 任意 - 最初の割り当てサイズで、整数で表されます。指定がない場合は、デフォルトはバイトになります。
- **--format *string*** - 任意 - ファイル形式の種類が含まれます。指定できるタイプは、raw、bochs、qcow、qcow2、qed、host_device、および vmdk です。ただし、これはファイルベースのストレージプールのみを対象としています。デフォルトで使用される 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場合は、「target 要素の設定」 を参照してください。
- **--backing-vol *string*** - 任意 - バッキングボリュームが含まれます。これは、スナップショットを撮る場合に使用されます。
- **--backing-vol-format *string*** - 任意 - バッキングボリュームの形式が含まれます。これは、スナップショットを撮る場合に使用されます。
- **--prealloc-metadata** - 任意 - メタデータを事前に割り当てることができます (完全割り当てではなく qcow2)。

例20.88 一連のパラメーターからストレージボリュームを作成する方法

以下の例では、*vol-new* という名前の 100MB のストレージボリュームを作成します。これには、例20.78「XML ファイルからストレージプールを作成する方法」 で作成した*vdisk* ストレージプールが含まれます。

```
# virsh vol-create-as vdisk vol-new 100M


vol vol-new created
```

20.30.3. XML ファイルからのストレージボリュームの作成

virsh vol-create *pool file* コマンドは、ストレージボリュームパラメーターを含む XML ファイルから新しいストレージボリュームを作成します。

例20.89 既存の XML ファイルからストレージボリュームを作成する方法

以下の例では、以下のように、ファイル*vol-new.xml*に基づいてストレージボリュームを作成します。

```
<volume>
 <name>vol-new</name>
 <allocation>0</allocation>
 <capacity unit="M">100</capacity>
 <target>
  <path>/var/lib/libvirt/images/vol-new</path>
  <permissions>
   <owner>107</owner>
   <group>107</group>
            <mode>0744</mode>
            <label>virt_image_t</label>
          </permissions>
        </target>
</volume>
```

ストレージボリュームは、ストレージプール*vdisk*に関連付けられています。イメージへのパスは **/var/lib/libvirt/images/vol-new** です。

```
# virsh vol-create vdisk vol-new.xml

vol vol-new created
```

20.30.4. ストレージボリュームのクローンの作成

virsh vol-clone *vol-name new-vol-name* コマンドは、既存のストレージボリュームのクローンを作成します。**virsh vol-create-from** コマンドも使用できますが、ストレージボリュームのクローンを作成することは推奨されません。このコマンドは、**--pool *string*** オプションを受け入れます。これにより、新しいストレージボリュームに関連付けられるストレージプールを指定できます。*vol* 引数は、ソースストレージボリュームの名前またはキーまたはパスです。*name* 引数は、新しいストレージボリュームの名前を指します。追加情報は、「virsh を使用したストレージボリュームの作成」 を参照してください。

例20.90 ストレージボリュームのクローンを作成する方法

以下の例では、*vol-new* という名前のストレージボリュームを、*vol-clone* という名前の新しいボリュームにクローンします。

```
# virsh vol-clone vol-new vol-clone


vol vol-clone cloned from vol-new
```

–

20.31. ストレージボリュームの削除

virsh vol-delete *vol* pool コマンドは、指定したボリュームを削除します。このコマンドには、ボリュームが存在するストレージプールの名前または UUID と、ストレージボリュームの名前が必要です。ボリューム名の代わりに、削除するボリュームのキーまたはパスを使用することもできます。

例20.91 ストレージボリュームを削除する方法

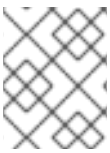
以下の例では、ストレージプール*vdisk*を含む *new-vol* という名前のストレージボリュームを削除します。

```
# virsh vol-delete new-vol vdisk

vol new-vol deleted
```

20.32. ストレージボリュームのコンテンツの削除

virsh vol-wipe *vol pool* コマンドはボリュームをワイプし、そのボリューム上のデータが今後の読み取りからアクセスできないようにします。このコマンドには、ボリュームが存在するストレージプールの名前または UUID である**--pool *pool*** と、消去するボリュームの名前または鍵またはパスである *pool* が必要です。ボリュームをゼロで再書き込みする代わりに、引数 **--algorithm** と、以下のサポートされるアルゴリズムのいずれかを使用して、別のワイプアルゴリズムを選択できることに注意してください。

- **zero** - 1-pass all zeroes
- **nnsa** - 削除可能/削除不可のハードディスクをサニタイズするための 4-pass NNSA Policy Letter NAP-14.1-C (XVI-8) (つまり、random x2, 0x00, verify)
- **dod** - 削除可能/削除不可の rigid ディスクをサニタイズするための 4-pass DoD 5220.22-M section 8-306 procedure (つまり、0x00, 0xff, verify)
- **bsi** - ドイツ情報技術セキュリティーセンター (http://www.bsi.bund.de) が推奨する 9-pass メソッド: 0xff、0xfe、0xfd、0xfb、0xf7、0xef、0xdf、0xbf、0x7f
- **gutmann** - Gutmann の論文で説明されている標準的な 35-pass sequence
- **schneier** - Bruce Schneier が "Applied Cryptography" (1996) で説明した 7-pass メソッド: 0x00、0xff、random x5
- **pfitzner7** - Roy Pfitzner によって提案された 7-random-pass メソッド: random x7
- **pfitzner33** - Roy Pfitzner によって提案された 33-random-pass メソッド: random x33
- **random** - 1-pass pattern: random.s



注記

アルゴリズムの可用性は、ホストにインストールされている "scrub" バイナリーのバージョンにより制限される場合があります。

例20.92 ストレージボリュームのコンテンツを削除する方法 (ストレージボリュームをワイプする方法)

以下の例では、ストレージボリューム*new-vol*のコンテンツを消去します。このコンテンツには、ストレージプール*vdisk*が関連付けられています。

```
# virsh vol-wipe new-vol vdisk

vol new-vol wiped
```

20.33. ストレージボリューム情報の XML ファイルへのダンプ

virsh vol-dumpxml *vol* コマンドは、ボリューム情報を取得し、その内容を含む XML ファイルを作成し、標準出力ストリームに設定されている設定に出力します。オプションで、**--pool** オプションを使用して、関連付けられたストレージプールの名前を指定できます。

例20.93 ストレージボリュームのコンテンツをダンプする方法

以下の例では、*vol-new* という名前のストレージボリュームのコンテンツを XML ファイルにダンプします。

```
# virsh vol-dumpxml vol-new
```

20.34. ボリューム情報のリスト表示

virsh vol-info *vol* コマンドは、指定したストレージボリュームの基本情報を表示します。ストレージボリューム名、キーまたはパスのいずれかを指定する必要があります。このコマンドは、オプション **--pool** も受け入れます。ここで、ストレージボリュームに関連付けられるストレージプールを指定できます。プール名または UUID のいずれかを指定できます。

例20.94 ストレージボリュームの情報を表示する方法

以下の例では、*vol-new* という名前のストレージボリュームに関する情報を取得します。このコマンドを実行して、ストレージボリュームの名前を、ストレージボリュームの名前に変更する必要があります。

```
# virsh vol-info vol-new
```

virsh vol-list *pool* コマンドは、指定したストレージプールに関連付けられているすべてのボリュームをリスト表示します。このコマンドには、ストレージプールの名前または UUID が必要です。**--details** オプションは、**virsh** に、ボリュームタイプおよびキャパシティー関連の情報 (利用可能な場合) を追加で表示するように指示します。

例20.95 ストレージボリュームに関連付けられているストレージプールを表示する方法

以下の例は、ストレージプール*vdisk*に関連付けられているすべてのストレージボリュームのリストを表示します。

```
# virsh vol-list vdisk
```

-

20.35. ストレージボリュームの情報の取得

virsh vol-pool *vol* コマンドは、指定したストレージボリュームのプール名または UUID を返します。デフォルトでは、ストレージプール名が返されます。**--uuid** オプションを使用した場合は、UUID が返されます。このコマンドには、要求された情報を返すストレージボリュームのキーまたはパスが必要です。

例20.96 ストレージボリュームの名前または UUID を表示する方法

以下の例では、パス**/var/lib/libvirt/images/vol-new**にあるストレージボリュームの名前を取得します。

```
# virsh vol-pool /var/lib/libvirt/images/vol-new

vol-new
```

vol-path --pool *pool-or-uuid vol-name-or-key* コマンドは、指定したボリュームのパスを返します。このコマンドには、**--pool *pool-or-uuid*** が必要です。これは、ボリュームが存在するストレージプールの名前または UUID です。また、パスが要求されているボリュームの名前または鍵である *vol-name-or-key* も必要です。

vol-name *vol-key-or-path* コマンドは、指定したボリュームの名前を返します。*vol-key-or-path* は、名前を返す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、指定したボリュームのボリュームキーを返します。 **--pool *pool-or-uuid*** はストレージプールの名前または UUID で、 *vol-name-or-path* はボリュームキーを返すボリュームの名前またはパスになります。

20.36. ゲストごとの仮想マシン情報の表示

20.36.1. ゲスト仮想マシンの表示

アクティブゲスト仮想マシンのリストと、現在のその状態を **virsh** で表示するには、次のコマンドを実行します。

```
# virsh list
```

使用可能なその他のオプションは以下のとおりです。

- **--all** - ゲスト仮想マシンのリストを表示します。以下に例を示します。

```
# virsh list --all
 Id Name                 State
----------------------------------
  0 Domain-0             running
  1 Domain202            paused
  2 Domain010            shut off
  3 Domain9600           crashed
```

注記

virsh list --all の実行時に結果が表示されない場合は、root ユーザーで仮想マシンを作成していない可能性があります。

virsh list --all コマンドは、次の状態を認識します。

- running - **running** の状態は、現在 CPU でアクティブなゲスト仮想マシンを指します。
- idle - **idle** の状態は、ゲスト仮想マシンがアイドル状態であり、実行していないか、実行できない可能性があることを示しています。これは、ゲスト仮想マシンが I/O を待機している (従来の待機状態) か、その他の処理が行われていないためにスリープ状態になっている場合に発生します。
- paused: ゲスト仮想マシンが一時停止すると、メモリーとその他のリソースを消費しますが、ハイパーバイザーから CPU リソースのスケジューリングには対応していません。**paused** 状態は、**virt-manager** または **virsh suspend** コマンドで **paused** ボタンを使用すると発生します。
- in shutdown - **in shutdown**の状態は、シャットダウン中のゲスト仮想マシン用です。ゲスト仮想マシンにシャットダウンシグナルが送信され、操作を正常に停止するプロセスにあるはずです。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可能性があり、一部のオペレーティングシステムはこれらのシグナルに応答しない場合があります。
- shut off - **shut off** 状態は、ゲスト仮想マシンが実行していないことを示しています。これは、ゲスト仮想マシンが完全にシャットダウンしているか、起動していない場合に発生します。
- crashed - **crashed** の状態は、ゲスト仮想マシンがクラッシュしたことを示しており、クラッシュ時にゲスト仮想マシンが再起動しないように設定されている場合にのみ発生します。
- pmsuspended - ゲストは、ゲストの電源管理により一時停止されています。

- **--inactive** - 定義されているものの、現在アクティブではないゲスト仮想マシンのリストを表示します。これには、**shut off** および **crashed** であるマシンが含まれます。
- **--managed-save** - 管理保存状態が有効になっているゲストは、**saved** としてリスト表示されます。この項目でゲストをフィルターにかけるには、**--all** オプションまたは **--inactive** オプションも使用する必要があります。
- **--name** - このコマンドは、デフォルトのテーブル形式ではなく、ゲストの名前をリスト表示します。このオプションは、**--uuid** オプション (ゲスト UUID のリストのみを出力) と **--table** オプション (テーブルスタイルの出力を使用すべきと判断) と相互に排他的です。
- **--title** - ゲスト **title** フィールドもリスト表示します。これには、ゲストの簡単な説明が含まれます。この項目は、デフォルト (**--table**) の出力形式で使用する必要があります。以下に例を示します。

```
$ virsh list --title

 Id    Name                           State    Title
----------------------------------------------------------------------
 0     Domain-0                       running  Mailserver1
 2     rhelvm                         paused
```

- **--persistent** - 永続ゲストのみがリストに含まれます。一時ゲストをリスト表示するには、**--transient** 引数を使用します。
- **--with-managed-save** - 管理保存で設定されているゲストのリストを表示します。ゲストのリストを表示するには、**--without-managed-save** を使用します。
- **--state-running** - 実行中のゲストのみをリスト表示します。同様に、一時停止したゲストには **--state-paused** を使用し、無効にしたゲストには **--state-shutoff** を使用します。**--state-other** は、すべての状態をフォールバックとしてリスト表示します。
- **--autostart** - 自動起動ゲストのみがリスト表示されます。この機能を無効にしたゲストのリストを表示するには、引数 **--no-autostart** を使用します。
- **--with-snapshot** - スナップショットイメージをリスト表示できるゲストのリストを表示します。スナップショットなしでゲストに絞り込むには、**--without-snapshot** オプションを使用します。

20.36.2. 仮想 CPU 情報の表示

virsh を使用してゲスト仮想マシンから仮想 CPU を表示するには、次のコマンドを実行します。

```
# virsh vcpuinfo {domain-id, domain-name or domain-uuid}
```

virsh vcpuinfo の出力例を以下に示します。

```
# virsh vcpuinfo guest1
VCPU:           0
CPU:            2
State:          running
CPU time:       7152.4s
CPU Affinity:   yyyy

VCPU:           1
CPU:            2
State:          running
CPU time:       10889.1s
CPU Affinity:   yyyy
```

20.36.3. vCPU のホストの物理マシンの CPU へのピニング

virsh vcpupin コマンドは、仮想 CPU を物理 CPU に割り当てます。

```
# virsh vcpupin guest1
VCPU: CPU Affinity
----------------------------------
  0: 0-3
  1: 0-3
```

vcpupin コマンドには、以下の引数を使用できます。

- **--vcpu** には vcpu 番号が必要です。
- **[--cpulist]** ***string*** は、設定するホスト物理マシンの CPU 番号のリストを表示します。

- **--config** は次回のブートに影響します。
- **--live** は、実行中のゲスト仮想マシンに影響します。
- **--current** は、現在のゲスト仮想マシンのステータスに影響を及ぼします。

20.36.4. 指定したドメインの仮想 CPU 数に関する情報の表示

virsh vcpucount コマンドには、*domain*名またはドメイン ID が必要です。

```
# virsh vcpucount guest1
maximum      config         2
maximum      live           2
current      config         2
current      live           2
```

vcpucountには、以下の引数を指定できます。

- **--maximum** は、vcpus の最大容量を取得します。
- **--active** は、現在アクティブな vcpu の数を取得します。
- 実行中のゲスト仮想マシンから**--live** を取得します。
- **--config** は、システムの次回起動時に使用する値を取得します。
- **--current** は、現在のゲスト仮想マシンの状態に応じて値を取得します。
- 返される **--guest** 数は、ゲストの視点から見たものです。

20.36.5. 仮想 CPU アフィニティーの設定

物理 CPU を使用して仮想 CPU のアフィニティーを設定するには、以下を実行します。

```
# virsh vcpupin domain-id vcpu cpulist
```

domain-id パラメーターは、ゲスト仮想マシンの ID 番号または名前です。

vcpu パラメーターは、ゲスト仮想マシンに割り当てられている仮想 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 ID 番号のリストで、コンマで区切ります。 **cpulist** パラメーターは、VCPU を実行できる物理 CPU を指定します。

--config などの追加パラメーターは次回の起動に影響を及ぼしますが、**--live** はゲスト仮想マシンの実行中および**- -current** に影響を及ぼします。

20.36.6. 仮想 CPU 数の設定

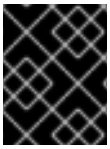
このコマンドを使用して、ゲスト仮想マシンでアクティブな仮想 CPU の数を変更します。デフォルトでは、このコマンドはアクティブなゲスト仮想マシンで機能します。次にゲスト仮想マシンを起動したときに使用される非アクティブの設定を変更するには、**--config** フラグを使用します。**virsh** でゲスト仮想マシンに割り当てられている CPU の数を変更するには、次のコマンドを実行します。

```
# virsh setvcpus {domain-name, domain-id or domain-uuid} count [[--config] [--live] | [--current]] [--maximum] [--guest]
```

以下に例を示します。

```
# virsh setvcpus guestVM1 2 --live
```

guestVM1 への vCPU の数は 2 に設定され、このアクションは guestVM1 の実行中に実行されます。



重要

vCPU のホットアンプラグは、Red Hat Enterprise Linux 7 ではサポートされていません。

数値は、ホスト、ハイパーバイザー、またはゲスト仮想マシンの元の説明から制限できます。

--config フラグを指定すると、ゲスト仮想マシンの保存されている XML 設定が変更され、ゲストが起動しないと有効になりません。

--live を指定する場合は、ゲスト仮想マシンがアクティブである必要があり、変更が即座に行われます。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。ハイパーバイザーで対応している場合は、**--config** フラグと **--live** フラグの両方を指定できます。

--current を指定した場合は、フラグが現在のゲスト仮想マシンの状態に影響を及ぼします。

フラグが指定されていない場合は、**--live** フラグが指定されたと見なされます。ゲスト仮想マシンがアクティブでないと、コマンドが失敗します。さらに、フラグが指定されていない場合は、**--config** フラグも想定されるかどうかはハイパーバイザーによる影響を受けます。これにより、変更を永続化するために XML 設定を調整するかどうかを決定します。

--maximum フラグは、ゲスト仮想マシンを次に起動したときにホットプラグできる仮想 CPU の最大数を制御します。したがって、これは **--config** フラグでのみ使用でき、**--live** フラグでは使用できません。

count は、ゲスト仮想マシンに割り当てられている CPU 数を超えることはできないことに注意してください。

--guest を指定すると、フラグが、現在のゲスト仮想マシンの CPU 状態を変更します。

20.36.7. メモリー割り当ての設定

virsh でゲスト仮想マシンのメモリー割り当てを変更するには、次のコマンドを実行します。

```
# virsh setmem {domain-id or domain-name} count
```

以下に例を示します。

```
# virsh setmem vr-rhel6u1-x86_64-kvm --kilobytes 1025000
```

countはキロバイト単位で指定する必要があります。新しい count の値は、ゲスト仮想マシンに指定した量を超えることはできません。64MB 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使用できない可能性があります。最大メモリー値を上げても、アクティブなゲスト仮想マシンには影響しません。新しい値が利用可能なメモリーよりも小さい場合は、縮小するため、ゲスト仮想マシンがクラッシュする可能性があります。

このコマンドには、以下のオプションがあります。

- *domain* - ドメインネーム、ID、または UUID で指定されます。
- *size* - 新しいメモリーサイズを、スケール変換した整数で指定します。デフォルトの単位は KiB ですが、別の単位を指定できます。

  有効なメモリーユニットは以下のとおりです。

  - ***b*** または ***bytes*** (バイト)
  - ***KB*** (キロバイト) (10^3 または 1,000 バイトのブロック)
  - ***k*** または ***KiB*** (キビバイト) (2^{10} または 1024 バイトのブロック)
  - ***MB*** (メガバイト) (10^6 または 1,000,000 バイトのブロック)
  - ***M*** または ***MiB*** (メビバイト) (2^{20} または 1,048,576 バイトのブロック)
  - ***GB*** (ギガバイト) (10^9 または 1,000,000,000 バイトのブロック)
  - ***G*** または ***GiB*** (ギビバイト) (2^{30} または 1,073,741,824 バイトのブロック)
  - ***TB*** (テラバイト) (10^{12} または 1,000,000,000,000 バイトのブロック)
  - ***T*** または ***TiB*** (テビバイト) (2^{40}または 1,099,511,627,776 バイトのブロック)

  すべての値は、libvirt により、最も近い kibibyte に丸められます。また、ハイパーバイザーが対応する粒度に丸めることもできます。一部のハイパーバイザーは、4000KiB(4000 x 2^{10} または 4,096,000 バイト) などの最小値も適用します。この値の単位は、オプションの属性 ***memory unit*** により決定されます。デフォルトは、測定単位としての KiB (キビバイト) になります。この値は、2^{10} または 1024 バイトのブロックで乗算されます。

- **--config** - コマンドは次回のシステムの起動時に有効になります。
- **--live** - 実行中のゲスト仮想マシンのメモリーを制御します。
- **--current** - このコマンドは、現在のゲスト仮想マシンのメモリーを制御します。

20.36.8. ドメインのメモリー割り当ての変更

virsh setmaxmem *domain size* --config --live --current コマンドを使用すると、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量を設定できます。

```
# virsh setmaxmem guest1 1024 --current
```

最大メモリーサイズに指定できるサイズは、拡張された整数です。拡張子に対応するものを除き、デフォルトではキビバイト単位で表されます。このコマンドには、以下の引数を使用できます。

- **--config** - 次回の起動に影響します
- **--live** - 実行中のゲスト仮想マシンのメモリーを制御します。これにより、すべてのハイパーバイザーで、最大メモリー制限のライブ変更が許可されないため、ハイパーバイザーがこの動作に対応できるようになります。

- **--current** - 現在のゲスト仮想マシンのメモリーを制御します。

20.36.9. ゲスト仮想マシンブロックのデバイス情報の表示

virsh domblkstat コマンドを使用して、実行中のゲスト仮想マシンのブロックデバイス統計情報を表示します。**--human**を使用すると、よりユーザーフレンドリーな方法で統計情報を表示できます。

```
# virsh domblkstat GuestName block-device
```

20.36.10. ゲスト仮想マシンのネットワークデバイス情報の表示

virsh domifstat コマンドを使用すると、実行中のゲスト仮想マシンのネットワークインターフェイス統計情報が表示されます。

```
# virsh domifstat GuestName interface-device
```

20.37. 仮想ネットワークの管理

本セクションでは、**virsh** コマンドを使用した仮想ネットワークの管理を説明します。仮想ネットワークのリストを表示するには、

```
# virsh net-list
```

このコマンドは、以下のような出力を生成します。

```
# 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、次のコマンドを実行します。

```
# virsh net-dumpxml NetworkName
```

指定した仮想ネットワークに関する情報を XML 形式で表示します。

```
# 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**virsh** コマンドは、以下のとおりです。

- **virsh net-autostart *network-name***: **libvirt** デーモンの起動時に自動的に起動する *network-name* をマークします。**--disable** オプションは、*network-name* のマークを解除します。
- **virsh net-create *XMLfile***: 既存のファイルの XML 定義を使用して、新しい (一時的な) ネットワークを起動します。
- **virsh net-define *XMLfile***: 既存のファイルの XML 定義を使用して、新しいネットワークを起動はせずに定義します。
- **virsh net-destroy *network-name***: *network-name* として指定されたネットワークを破棄します。
- **virsh net-name *networkUUID***: 指定された *networkUUID* をネットワーク名に変換します。
- **virsh net-uuid *network-name***: 指定された *network-name* をネットワーク UUID に変換します。
- **virsh net-start *nameOfInactiveNetwork***: 非アクティブなネットワークを起動します。
- **virsh net-undefine *nameOfInactiveNetwork***: ネットワークの非アクティブな XML 定義を削除します。ネットワークの状態には影響を及ぼしません。このコマンドの実行時にドメインが実行していた場合、ネットワークは実行を継続します。ただし、ネットワークは永続的ではなく一時的になります。

libvirt には、ドメインが使用でき、実際のネットワークデバイスにリンクできる仮想ネットワークを定義する機能があります。この機能の詳細は、 libvirt アップストリーム Web サイト のドキュメントを参照してください。仮想ネットワーク用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、名前または UUID のいずれかになります。

20.37.1. 仮想ネットワークの自動起動

virsh net-autostart コマンドは、ゲスト仮想マシンの起動時に仮想ネットワークを自動的に起動するように設定します。

```
# virsh net-autostart network [--disable]
```

このコマンドは、**--disable** オプションを受け入れます。これにより、自動開始コマンドが無効になります。

20.37.2. XML ファイルからの仮想ネットワークの作成

virsh net-create コマンドは、XML ファイルから仮想ネットワークを作成します。libvirt が使用する XML ネットワークフォーマットの詳細は、libvirt アップストリームの Web サイト を参照してください。このコマンド *file* は、XML ファイルへのパスになります。XML ファイルから仮想ネットワークを作成するには、次のコマンドを実行します。

```
# virsh net-create file
```

20.37.3. XML ファイルからの仮想ネットワークの定義

virsh net-define は、XML ファイルから仮想ネットワークを定義します。ネットワークは単に定義されているだけで、インスタンス化されません。

```
# virsh net-define file
```

20.37.4. 仮想ネットワークの停止

virsh net-destroy コマンドは、仮想ネットワークの名前または UUID を指定して、その仮想ネットワークを破壊 (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* が必要です。

```
# virsh net-destroy network
```

20.37.5. ダンプファイルの作成

virsh net-dumpxml は、指定した仮想ネットワークの stdout に、仮想ネットワーク情報を XML ダンプとして出力します。**--inactive** を指定すると、物理機能は、関連付けられた仮想機能にデプロイメントされません。

```
# virsh net-dumpxml network [--inactive]
```

20.37.6. 仮想ネットワークの XML 設定ファイルの編集

以下のコマンドは、ネットワークの XML 設定ファイルを編集します。

```
# virsh net-edit network
```

XML ファイルの編集に使用するエディターは、$VISUAL 環境変数または $EDITOR 環境変数で提供できます。デフォルトは **vi** です。

20.37.7. 仮想ネットワークに関する情報の取得

virsh net-info は、*network* オブジェクトの基本情報を戻します。

```
# virsh net-info network
```

20.37.8. 仮想ネットワークに関する情報のリスト表示

virsh net-list コマンドは、動作しているネットワークのリストを返します。**--all** を指定すると、定義済みで非アクティブなネットワークも含まれます。**--inactive** を指定すると、無効なもののみがリスト表示されます。返されたネットワークを **--persistent** でフィルタリングして永続的なネットワークを、**--transient** でフィルタリングして一時的なネットワークを、**--autostart** でフィルタリングして自動起動が有効になっているネットワークを、そして **--no-autostart** でフィルタリングして自動起動が無効になっているネットワークをリスト表示することも推奨されます。

注記: 古いサーバーと通信する場合、このコマンドは一連の API コールと固有の競合を使用することを強制されます。ここでは、リストの収集中にプールの状態が変更した場合に、プールがリストに表示されなかったり、複数回表示されたりすることがあります。新しいサーバーにはこの問題がありません。

仮想ネットワークのリストを表示するには、次のコマンドを実行します。

```
# virsh net-list [--inactive | --all] [--persistent] [<--transient>] [--autostart] [<--no-autostart>]
```

20.37.9. ネットワーク UUID のネットワーク名への変換

virsh net-name は、ネットワークの UUID をネットワーク名に変換します。

-

```
# virsh net-name network-UUID
```

20.37.10. ネットワーク名のネットワーク UUID への変換

virsh net-uuid コマンドは、ネットワーク名をネットワーク UUID に変換します。

```
# virsh net-uuid network-name
```

20.37.11. 定義済みの非アクティブなネットワークの起動

virsh net-start コマンドは、(定義済みの) 非アクティブネットワークを開始します。

```
# virsh net-start network
```

20.37.12. 非アクティブなネットワークの設定の定義解除

virsh net-undefine コマンドは、非アクティブなネットワークの設定を定義しません。

```
# virsh net-undefine network
```

20.37.13. 既存のネットワーク定義ファイルの更新

```
# virsh net-update network directive section XML [--parent-index index] [[--live] [--config] | [--current]]
```

virsh net-update コマンドは、以下のいずれかの*directives*をセクションに実行することにより、既存のネットワーク定義の指定されたセクションを更新します。

- **add-first**
- **add-last** または **add** (これらは同義)
- **削除**
- **modify**

section は、以下のいずれかになります。

- **bridge**
- **domain**
- **ip**
- **ip-dhcp-host**
- **ip-dhcp-range**
- **forward**
- **forward interface**
- **forward-pf**

- **portgroup**
- **dns-host**
- **dns-txt**
- **dns-srv**

各セクションは、変更される要素につながる XML 要素階層の連結によって名前が付けられます。たとえば、**ip-dhcp-host** は、ネットワークの **<ip>** 要素内の **<dhcp>** 要素に含まれる **<host>** 要素を変更します。

XML は、変更するタイプの XML 要素全体 (**<host mac="00:11:22:33:44:55' ip='1.2.3.4'/>** など) のテキスト、または XML 要素がすべて含まれるファイル名のいずれかになります。あいまいさの除去は、提供されているテキストの最初の文字を調べて行います。最初の文字が **<** の場合は XML テキストで、最初の文字が **>** でない場合は、使用される xml テキストが含まれるファイル名になります。***--parent-index*** オプションは、要求された要素が、複数の親要素のうちどの要素にあるのかを指定するために使用します (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>** 要素のいずれかに置くことができます。parent-index が指定されていない場合は最も適切な **<ip>** 要素が選択されます (通常は、**<dhcp>** 要素がすでに存在するものだけが対象となります)。ただし、***--parent-index*** が指定されていると、**<ip>** の特定のインスタンスが修正されます。***--live*** を指定すると、実行中のネットワークに影響を及ぼします。***--config*** を指定した場合は、次に永続的なネットワークを起動する際に影響を及ぼします。***--current*** を指定した場合は、現在のネットワークステータスに影響を及ぼします。***--live*** フラグと ***--config*** フラグの両方が指定できますが、**--current** は排他的になります。フラグを指定しないことは、***--current*** を指定することと同じです。

20.37.14. virsh を使用したゲスト仮想マシンの移行

virsh を使用した移行に関する情報は、Live KVM Migration with virsh というタイトルのセクションにあります。「virsh を使用した KVM のライブ移行」 を参照してください。

20.37.15. ゲスト仮想マシンの静的 IP アドレスの設定

ゲスト仮想マシンが DHCP から IP アドレスを取得するように設定されているにもかかわらず、予測可能な静的 IP アドレスが必要な場合は、以下の手順に従って libvirt が使用する DHCP サーバー設定を変更できます。この手順では、この変更を行うために、ゲストのインターフェイスの MAC アドレスを把握しておく必要があります。したがって、ゲストの作成後に操作を実行するか、ゲストの作成前にその MAC アドレスを決定する必要があります。その後、ゲスト仮想マシンの作成時に同じアドレスを手動で設定する必要があります。

また、この手順は、転送モードが **"nat"**、**"route"**、または 転送モードが何もない libvirt 仮想ネットワークに接続されているゲストインターフェイスにのみ有効であることに注意してください。ネットワークが **forward mode="bridge"** または **"hostdev"** で設定されている場合は、この手順が正しく機能しません。この場合、DCHP サーバーはネットワークの別の場所にあるため、libvirt の制御下にはありません。この場合は、リモートの DHCP サーバーで静的 IP エントリーを作成する必要があります。これを行うには、サーバーに同梱されているドキュメントを参照してください。

手順20.5 静的 IP アドレスの設定

この手順は、ホストの物理マシンで実行します。

1. **ゲスト XML 設定ファイルの確認**
   virsh domiflist *guest1* コマンドを実行して、ゲストのネットワーク設定を表示します。*guest1* の代わりに仮想マシンの名前を使用してください。テーブルが表示されます。Source 列を調べ

ます。これが、お使いのネットワークの名前になります。この例では、ネットワークは default と呼ばれています。この名前は、残りの手順および MAC アドレスに使用されます。

```
# virsh domiflist guest1
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet4      network    default    virtio      52:54:00:48:27:1D
```

2. **DHCP 範囲を確認します。**
設定する IP アドレスは、ネットワークに指定される dhcp の範囲内で設定する必要があります。また、ネットワーク上のその他の既存の静的 IP アドレスと競合しないようにする必要があります。利用可能なアドレスと、使用されるアドレスの範囲を確認するには、ホストマシンで以下のコマンドを使用します。

```
# virsh net-dumpxml default | egrep 'range|host\ mac'

<range start='198.51.100.2' end='198.51.100.254'/>
<host mac='52:54:00:48:27:1C:1D' ip='198.51.100.2'/>
```

表示される出力は例とは異なり、より多くの行とホストの mac 行を表示できます。各ゲストの静的 IP アドレスには、1 行が含まれます。

3. **静的 IP アドレスを設定します。**
ホストマシンで以下のコマンドを実行し、*default* をネットワークの名前に置き換えます。

```
# virsh net-update default add ip-dhcp-host '<host mac="52:54:00:48:27:1D"
ip="198.51.100.3"/>' --live --config
```

--live オプションでは、この変更を即座に有効にし、**--config** オプションでは変更を永続化します。このコマンドは、有効な IP アドレスおよび MAC アドレスを使用する限り、作成していないゲスト仮想マシンにも機能します。MAC アドレスは、有効なユニキャスト MAC アドレス (**:** で区切られた 6 つの 16 進数のペア。最初の数字のペアは偶数) にしてください。libvirt が新しいランダム MAC アドレスを作成する場合は、最初の 3 桁のペアに **52:54:00** を使用するため、この規則に従うことが推奨されます。

4. **インターフェイスの再起動 (オプション)**
ゲスト仮想マシンが現在実行中の場合は、ゲスト仮想マシンに DHCP アドレスの再要求を強制する必要があります。ゲストが実行していない場合は、次にゲストを起動したときに、新しい IP アドレスが実装されます。インターフェイスを再起動するには、ホストマシンで以下のコマンドを入力します。

```
# virsh domif-setlink guest1 52:54:00:48:27:1D down
# sleep 10
# virsh domif-setlink guest1 52:54:00:48:27:1D up
```

このコマンドにより、ゲスト仮想マシンのオペレーティングシステムは、イーサネットケーブルが接続されていないと認識し、10 秒後に再接続します。**sleep** コマンドは、IP アドレスを再要求せずに、多くの DHCP クライアントが短い間ケーブルを切断できるため、重要です。10 秒経過すると、DHCP クライアントは古い IP アドレスを忘れ、**up** コマンドが実行されると新しい IP アドレスを要求します。何らかの理由でこのコマンドが失敗した場合は、ゲストオペレーティングシステムの管理インターフェイスからゲストのインターフェイスをリセットする必要があります。

20.38. インターフェイスコマンド

以下のコマンドはホストインターフェイスを操作するため、ゲスト仮想マシンからは実行しないでください。これらのコマンドは、ホストの物理マシンにある端末から実行する必要があります。

警告

本セクションのコマンドは、マシンで NetworkManager サービスが無効になっており、代わりに **network** サービスを使用している場合に限りサポートされます。

多くの場合、このようなホストインターフェイスは、ゲスト仮想マシンの **<interface>** 要素 (システムが作成したブリッジインターフェイスなど) 内で名前で使用できますが、ホストインターフェイスが特定のゲスト設定 XML と完全に関連付けられる必要はありません。ホストインターフェイスのコマンドの多くは、ゲスト仮想マシンに使用されるコマンドと似ており、インターフェイスの名前はその名前または MAC アドレスになります。ただし、**iface** 引数に MAC アドレスを使用する場合、そのアドレスが一意である場合にのみ機能します (多くの場合、インターフェイスとブリッジが同じ MAC アドレスを共有しており、その場合、その MAC アドレスを使用すると曖昧さによりエラーが発生し、代わりに名前を使用する必要があります)。

20.38.1. XML ファイルを使用したホストの物理マシンインターフェイスの定義および起動

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イスを定義します。このコマンドはインターフェイスのみを定義し、起動はしません。

```
# virsh iface-define iface.xml
```

定義済みのインターフェイスを起動する場合は、**iface-start *interface*** を実行します。*interface* は、インターフェイスの名前になります。

20.38.2. ホストインターフェイスの XML 設定ファイルの編集

コマンド **virsh iface-edit *interface*** は、ホストインターフェイスの XML 設定ファイルを編集します。これは、XML 設定ファイルを変更する際に推奨される **唯一** の方法です。(これらのファイルの詳細は、23章 *ドメイン XML の操作* を参照してください。)

20.38.3. ホストインターフェイスのリスト表示

virsh iface-listには、アクティブホストインターフェイスのリストが表示されます。**--all** が指定されている場合、このリストには、定義されているが非アクティブなインターフェイスも含まれます。**--inactive** を指定すると、非アクティブのインターフェイスのみがリスト表示されます。

20.38.4. MAC アドレスのインターフェイス名への変換

virsh iface-name *interface* コマンドは、ホストインターフェイスの MAC アドレスをインターフェイス名に変換します。ただし、MAC アドレスがホストインターフェイス間で固有のものである必要があります。このコマンドには、インターフェイスの MAC アドレスである *interface* が必要です。

virsh iface-mac *interface* コマンドは、ホストのインターフェイス名を MAC アドレスに変換します。この場合、*interface* がインターフェイス名になります。

20.38.5. 特定のホスト物理マシンインターフェイスの停止および定義解除

virsh iface-destroy *interface* コマンドは、指定したホストインターフェイスを破壊 (停止) します。これは、ホストで **virsh if-down** を実行するのと同じです。このコマンドは、そのインターフェイスのアクティブな使用を無効にし、すぐに有効にします。

インターフェイスの定義を解除する場合は、インターフェイス名を指定して **virsh iface-undefine *interface*** を実行します。

20.38.6. ホスト設定ファイルの表示

virsh iface-dumpxml *interface* --inactive コマンドは、ホストインターフェイスの情報を、stdout への XML ダンプとして表示します。**--inactive** 引数を指定すると、出力には、次回起動したときに使用されるインターフェイスの永続的な状態が反映されます。

20.38.7. ブリッジデバイスの作成

virsh iface-bridge コマンドは、*bridge* という名前のブリッジデバイスを作成し、既存のネットワークデバイス*interface*を新しいブリッジに割り当てます。この新しいブリッジはすぐに機能し、STP が有効になり、遅延は 0 になります。

```
# virsh iface-bridge interface bridge
```

これらの設定は、**--no-stp** オプション、**--no-start** オプション、および遅延秒数で変更できることに注意してください。インターフェイスの IP アドレス設定は、新しいブリッジデバイスに移動します。ブリッジの切断の詳細は、「ブリッジデバイスの破損」 を参照してください。

20.38.8. ブリッジデバイスの破損

virsh iface-unbridge *bridge* --no-start コマンドは、指定したブリッジデバイス *bridge* を削除し、その基盤となるインターフェイスを通常の使用に戻し、すべての IP アドレスの設定をブリッジデバイスから基盤となるデバイスに移動します。**--no-start** 引数を使用しない限り、基本となるインターフェイスが再起動しますが、通常は再起動しないことを念頭に置いてください。ブリッジを作成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。

20.38.9. インターフェイススナップショットの操作

virsh iface-begin コマンドは、現在のホストインターフェイス設定のスナップショットを作成します。これは、後でコミット (**virsh iface-commit** を使用) または復元 (**virsh iface-rollback** を使用) できます。これは、新しいホストインターフェイスの定義および起動で問題が発生し、システムが正しく設定されていない場合に役立ちます。スナップショットがすでに存在する場合は、以前のスナップショットがコミットまたは復元されるまで、このコマンドは失敗します。スナップショットの作成時と、その最終的なコミットまたはロールバックの間に、libvirt API 外でホストインターフェイスに外部変更が加えられると、定義されていない動作が発生します。

virsh iface-commit を実行して、前回の**virsh iface-begin**以降に加えた変更をすべて動作として宣言し、ロールバックポイントを削除します。**virsh iface-begin** を使用してインターフェイススナップショットが開始していない場合は、このコマンドは失敗します。

virsh iface-rollback を使用して、ホストインターフェイスの設定を、**virsh iface-begin** コマンドの最終実行時を記録した状態に戻します。**virsh iface-begin** コマンドが以前に実行されていない場

合、**virsh iface-rollback** は失敗します。**virsh iface-commit** の実行前にホストの物理マシンを再起動すると、自動ロールバックが実行され、ホストの設定が、**virsh iface-begin**の実行時の状態に復元されることに注意してください。これは、ネットワーク設定に不適切な変更があると、変更を元に戻す目的でホストに到達できない場合に役に立ちますが、ホストは電源サイクルが設定されているか、そうでなければ強制的に再起動されます。

例20.97 スナップショットの使用例

新しいホストインターフェイスを定義して起動します。

```
# virsh iface-begin
# virsh iface-define eth4-if.xml
# virsh if-start eth4
```

問題が発生し、ネットワークの実行が停止した場合は、変更をロールバックします。

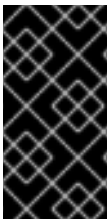
```
# virsh iface-rollback
```

すべてが適切に機能する場合は、変更をコミットします。

```
# virsh iface-commit
```

20.39. スナップショットの管理

以下のセクションでは、ゲスト仮想マシンのスナップショットを操作するために実行できるアクションを説明します。*Snapshots* は、指定した時点でゲスト仮想マシンのディスク、メモリー、およびデバイス状態を取得して、将来使用できるように保存します。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、破壊的な操作になる可能性がある前にゲスト仮想マシンの状態を保存することまで、多くの用途があります。スナップショットは、一意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のに使用される XML 形式のドキュメントは、libvirt アップストリームの Web サイト を参照してください。



重要

Red Hat Enterprise Linux 7 では、ゲスト仮想マシンの一時停止または電源切断時のスナップショットの作成のみに対応しています。実行中のゲスト (*ライブスナップショット* としても知られている) のスナップショットの作成は、Red Hat Virtualization™ で利用できます。詳細はサービス担当者にお問い合わせください。

20.39.1. スナップショットの作成

virsh snapshot-create コマンドは、ゲスト仮想マシンの XML ファイル (**<name>** 要素や **<description>** 要素、**<disks>** など) で指定されたプロパティーを使用して、ゲスト仮想マシンのスナップショットを作成します。スナップショットを作成するには、以下のコマンドを実行します。

```
# virsh snapshot-create domain XML file [--redefine [--current] [--no-metadata] [--halt] [--disk-only] [--reuse-external] [--quiesce] [--atomic]
```

ゲスト仮想マシンの名前、ID、または uid は、ゲスト仮想マシンの要件として使用できます。XML 要件は、最低限でも *name*、*description*、および *disks* の各要素を含む必要がある文字列です。

残りのオプション引数は以下のとおりです。

- **--disk-only** - ゲスト仮想マシンのメモリー状態はスナップショットに含まれません。

- XML ファイル文字列が完全に省略されている場合は、libvirt がすべてのフィールドの値を選択します。snapshot-current によってリスト表示されるように、新しいスナップショットが最新になります。また、スナップショットには、ゲスト仮想マシンの状態で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速くなりますが、電源コードが突然引っ張られたときのディスクの状態と同様であるため、ディスクスナップショットに戻すには **fsck** またはジャーナル再生が必要になる場合があります。**--halt** と **--disk-only** を混在させると、現時点ではディスクにフラッシュされていない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、ゲスト仮想マシンは非アクティブのままになります。**--halt** と **--disk-only** を混在させると、現時点ではディスクにフラッシュされていないデータおよびメモリー状態が失われます。

- **--redefine** は、**virsh snapshot-dumpxml** によって生成されたすべての XML 要素が有効であるかどうかを指定します。これは、スナップショット階層をあるマシンから別のマシンに移行したり、一時的な ゲスト仮想マシンがなくなって後で同じ名前と UUID で再作成された場合の階層を再作成したり、スナップショットメタデータにわずかな変更を加えたりするために使用できます (スナップショットに埋め込まれたゲスト仮想マシン XML のホスト固有の側面など)。このフラグを指定すると、**xmlfile** 引数が必須になります。**--current** フラグも指定されていない場合は、ゲスト仮想マシンの現在のスナップショットは変更されません。

- **--no-metadata** はスナップショットを作成しますが、メタデータは即座に破棄されます (つまり、libvirt は、スナップショットを現在のものとして扱いません。また、メタデータに関するlibvirtを教えるために **--redefine** を使用しない限り、スナップショットに戻ることはできません)。

- **--reuse-external** を使用すると、スナップショット XML は既存のファイルの宛先を使用して外部スナップショットを要求すると、宛先が存在している必要があり、再利用されます。そうしないと、既存ファイルの内容が失われないようにスナップショットは拒否されます。

- **--quiesce** libvirt は、ゲストエージェントを使用して、ゲスト仮想マシンがマウントしたファイルシステムのフリーズとフリーズ解除を試みます。ただし、ゲスト仮想マシンにゲストエージェントがない場合は、スナップショットの作成に失敗します。スナップショットには、仮想ゲストマシンのメモリー状態を含めることができます。スナップショットは外部にある必要があります。

- **--atomic** を指定すると、libvirt はスナップショットが成功するか、変更なしで失敗するかのいずれかを保証します。すべてのハイパーバイザーがこれをサポートしているわけではないことに注意してください。このフラグを指定しないと、動作の一部を実行した後にハイパーバイザーの中には失敗するものがあります。このため、**virsh dumpxml** を使用して、部分的な変更が行われたかどうかを確認する必要があります。

スナップショットメタデータが存在すると、永続的なゲスト仮想マシンの定義を解除しようとすることを防ぐことができます。ただし、一時的な ゲスト仮想マシンでは、ゲスト仮想マシンが実行を終了すると (**destroy** などのコマンドまたは内部ゲストアクションによって) スナップショットメタデータが警告なしで失われます。

20.39.2. 現在のゲスト仮想マシンのスナップショットの作成

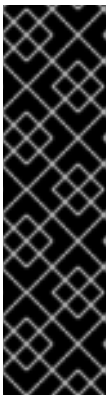
virsh snapshot-create-as コマンドは、ドメイン XML ファイルで指定されたプロパティー (**name** 要素や **description** 要素など) を使用して、ゲスト仮想マシンのスナップショットを作成します。この値が XML ストリングに含まれていない場合、libvirt は値を選択します。スナップショットを作成するには、以下のコマンドを実行します。

```
# snapshot-create-as domain {[--print-xml] | [--no-metadata] [--halt] [--reuse-external]} [name]
[description] [--disk-only [--quiesce]] [--atomic] [[--memspec memspec]] [--diskspec] diskspec]
```

残りのオプション引数は以下のとおりです。

- **--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** に適切な XML を作成します。
- **--halt** では、スナップショットの作成後も、ゲスト仮想マシンを非アクティブのままにします。
- **--disk-only** は、ゲスト仮想マシンのステータスを含まないスナップショットを作成します。
- **--memspec** を使用すると、チェックポイントが内外のどちらにあるかを制御できます。このフラグは必須で、その後に**[file=]name[,snapshot=type]** 形式の**memspec**が続きます。type は、none、internal、または external のいずれかになります。file=name にリテラルのコンマを含める場合は、2 番目のコンマでエスケープします。
- **--diskspec** オプションを使用して、**--disk-only** および外部チェックポイントによる外部ファイルの作成方法を制御できます。このオプションは、ドメイン XML 内の**<disk>**要素の数に従って、複数回指定できます。それぞれの**<diskspec>**は、ディスク **[,snapshot=type] [,driver=type][,file=name]** の形式になります。特定のディスクについて **--diskspec** を省略すると、仮想マシン設定のデフォルトの動作が使用されます。ディスクまたは**file=name**にリテラルのコンマを含める場合は、2 番目のコンマでエスケープします。*domain*、*name*、および *description* の 3 つすべてが含まれている場合を除き、ディスクスペックの前にリテラル**--diskspec**を指定する必要があります。たとえば、ディスクスペックが **vda,snapshot=external,file=/path/to,,new** の場合は、以下の XML になります。

```
<disk name='vda' snapshot='external'>
  <source file='/path/to,new'/>
</disk>
```



重要

Red Hat は、外部スナップショットを使用することを推奨します。外部スナップショットは、他の仮想化ツールで処理する場合、より柔軟で信頼性が高いためです。外部スナップショットを作成する場合は、**--diskspec vda,snapshot=external** で **virsh-create-as** を実行します。

この項目を使用しないと、**virsh** により内部スナップショットが作成されます。このスナップショットは安定性に欠け、最適化されていないため、使用することは推奨されていません。詳細は、「libvirt を使用して外部スナップショットを作成するための回避策」 を参照してください。

- **--reuse-external** が指定され、ドメイン XML または diskspec オプションは既存のファイルの宛先で外部スナップショットを要求すると、宛先が存在している必要があり、これは再利用されます。そうしないと、既存ファイルの内容が失われないようにスナップショットが拒否されます。
- **--quiesce** を指定すると、libvirt はゲストエージェントを使用して、ゲスト仮想マシンにマウントされているファイルシステムのフリーズとフリーズ解除を試みます。ただし、ドメインにゲストエージェントがない場合は、スナップショットの作成に失敗します。現在、これには**--disk-only** も渡される必要があります。

- **--no-metadata** はスナップショットデータを作成しますが、メタデータはすぐに破棄されます (つまり、libvirt はスナップショットを最新として処理せず、また、snapshot-create が、その後、libvirt にメタデータを再度教えるために使用されない限り、スナップショットに戻ることはできません)。このフラグは **--print-xml** と互換性がありません。

- **--atomic** を実行すると、libvirt はスナップショットの成功を保証するか、変更を行わずに失敗します。すべてのハイパーバイザーがこれをサポートしているわけではないことに注意してください。このフラグを指定しないと、動作の一部を実行した後にハイパーバイザーの中には失敗するものがあります。このため、**virsh dumpxml** を使用して、部分的な変更が行われたかどうかを確認する必要があります。

警告

現在、64 ビット ARM プラットフォーム ホストで実行している KVM ゲストのスナップショットを作成することはできません。64 ビット ARM の KVM は、Red Hat ではサポートされていないことに注意してください。

20.39.3. 現在使用しているスナップショットの表示

virsh snapshot-current コマンドを使用して、現在使用中のスナップショットを照会します。

```
# virsh snapshot-current domain {[--name] | [--security-info] | [snapshotname]}
```

snapshotname を使用しない場合は、ゲスト仮想マシンの現在のスナップショット (存在する場合) の snapshot XML が出力として表示されます。**--name** を指定すると、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** を指定すると、セキュリティーの機密情報が XML に含まれます。**snapshotname** を使用して、既存の名前付きスナップショットをゲスト仮想マシンに戻さずに、現在のスナップショットになるようにリクエストを生成します。

20.39.4. snapshot-edit

このコマンドは、現在使用中のスナップショットを編集するために使用されます。

```
# virsh snapshot-edit domain [snapshotname] [--current] {[--rename] [--clone]}
```

snapshotname と **--current** の両方を指定すると、編集したスナップショットが現在のスナップショットになります。**snapshotname** を省略する場合は、現在のスナップショットを編集するには、**--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、いくつかのエラーチェックも含まれます。

```
# virsh snapshot-dumpxml dom name > snapshot.xml
# vi snapshot.xml [note - this can be any editor]
# virsh snapshot-create dom snapshot.xml --redefine [--current]
```

--renameを指定すると、スナップショットの名前が変更されます。**--clone** を指定してスナップショット名を変更すると、スナップショット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は変更されません。スナップショット名の変更には十分な注意

が必要です。1 つの 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内容には、作成元のスナップショット名からしかアクセスできないためです。

20.39.5. snapshot-info

snapshot-info *domain* コマンドは、スナップショットに関する情報を表示します。使用するには、以下を実行します

```
# snapshot-info domain {snapshot | --current}
```

指定した**snapshot**、または**--current** で作成中のスナップショットの基本情報を出力します。

20.39.6. snapshot-list

指定したゲスト仮想マシンで利用可能なスナップショットのリストを表示します。デフォルトでは、スナップショット名、作成時間、およびゲスト仮想マシンの状態の列が表示されます。使用するには、以下を実行します

```
# virsh snapshot-list domain [{--parent | --roots | --tree}] [{[--from] snapshot | --current} [--
descendants]] [--metadata] [--no-metadata] [--leaves] [--no-leaves] [--inactive] [--active] [--disk-only]
[--internal] [--external]
```

オプションの引数は以下のとおりです。

- **--parent** は、各スナップショットの親の名前を示す列を、出力テーブルに追加します。このオプションは、**--roots** や **--tree** と併用できません。
- **--roots** は、リストをフィルタリングして、親がないスナップショットのみを表示します。このオプションは、**--parent** または **--tree** では使用できません。
- **--tree** は、スナップショット名のみをリスト表示し、出力をツリー形式で表示します。このオプションは、**--roots** や **--parent** と併用できません。
- **--from** は、指定したスナップショットの子であるスナップショットのリストをフィルタリングします。または、**--current** が指定されている場合は、リストを現在のスナップショットから開始するようにします。分離時または **--parent** と併用する場合、**--descendants** が存在しない限り、リストは直接の子に制限されます。**--tree** と一緒に使用すると、**--descendants** の使用が暗黙的になります。このオプションは、**--roots** とは互換性がありません。**--from** または **--current** の開始点は、**--tree** オプションが指定されていない限り、リストに含まれていないことに注意してください。
- **--leaves** を指定すると、このリストは、子がないスナップショットのみを対象にフィルター処理されます。同様に、**--no-leaves** を指定すると、リストは子を持つスナップショットのみにフィルタリングされます。(両方のオプションを省略するとフィルタリングはされませんが、両方のオプションを指定すると、サーバーがフラグを認識するかどうかに応じて同じリストまたはエラーが生成されることに注意してください) フィルタリングオプションは **--tree** と互換性がありません。
- **--metadata** を指定すると、このリストは、libvirt メタデータに関与するスナップショットのみを対象としてフィルター処理されるため、永続的なゲスト仮想マシンの定義が解除されたり、一時的なゲスト仮想マシンの破棄時に失われてしまう可能性があります。同様に、**--no-metadata** が指定されている場合には、リストが libvirt メタデータを必要としないスナップショットだけにフィルタリングされます。
- **--inactive** を指定すると、ゲスト仮想マシンのシャットダウン時に取得したスナップショット

に対してリストがフィルター処理されます。**--active** を指定すると、ゲスト仮想マシンの実行中に取得したスナップショットにリストがフィルタリングされ、スナップショットには、その稼働状態に戻すためのメモリーの状態が含まれます。**--disk-only** を指定すると、ゲスト仮想マシンの実行時に撮られたスナップショット (ただし、スナップショットにはディスクステートのみが含まれる) に対して、リストがフィルタリングされます。

- **--internal** を指定すると、リストは、既存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リングされます。--external を指定すると、リストは、ディスクイメージまたはメモリー状態に外部ファイルを使用するスナップショットにフィルタリングされます。

20.39.7. snapshot-dumpxml

virsh snapshot-dumpxml *domain snapshot* コマンドは、スナップショットという名前のゲスト仮想マシンのスナップショット XML を出力します。使用するには、以下を実行します

```
# virsh snapshot-dumpxml domain snapshot [--security-info]
```

--security-info オプションには、セキュリティーの機密情報も含まれます。**virsh snapshot-current** を使用すると、現行スナップショットの XML に簡単にアクセスできます。

20.39.8. snapshot-parent

指定したスナップショットについて、親スナップショットが存在する場合はその名前を出力し、存在しない場合は**--current** で現在のスナップショットの名前を出力します。使用するには、以下を実行します

```
# virsh snapshot-parent domain {snapshot | --current}
```

20.39.9. snapshot-revert

指定したドメインを、**snapshot** で指定したスナップショットか、**--current** で指定した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を取得してからドメイン内に加えられた変更はすべて失われます。また、**snapshot-revert** の完了後のドメインの状態は、元のスナップショット取得時のドメインの状態であることに注意してください。

スナップショットを元に戻すには、次のコマンドを実行します。

```
# virsh snapshot-revert domain {snapshot | --current} [{--running | --paused}] [--force]
```

通常、スナップショットに戻すと、ドメインはスナップショット作成時の状態のままになります。ただし、ゲスト仮想マシンの状態がないディスクスナップショットでは、ドメインは非アクティブの状態のままになります。**--running** または **--paused** のいずれかを渡すと、状態の変更 (非アクティブなドメイ

ンの起動や、実行中のドメインの一時停止など) が行われます。一時ドメインは非アクティブの状態にできないため、一時ドメインのディスクスナップショットに戻す場合には、このフラグのいずれかを使用する必要があります。

snapshot revertに追加の危険が伴う場合が 2 つあり、その場合は**--force** を使用して続行する必要があります。1 つ目は、設定を戻すための完全なドメイン情報がないスナップショットの場合です。libvirt は、現在の設定がスナップショット時の使用状況と一致することを証明できないため、**--force** を提供することで、スナップショットが現在の設定と互換性があることをlibvirtに保証します (一致しないとドメインの実行に失敗する可能性が高くなります)。もう 1 つは、実行中のドメインから、既存のハイパーバイザーを再利用するのではなく、新しいハイパーバイザーを作成する必要があるアクティブなドメインに戻す場合です。これは、既存の VNC 接続または Spice 接続を壊すなどの欠点を意味するからです。これは、証明可能な互換性のない設定を使用するアクティブなスナップショットと、**--start** フラグまたは **--pause** フラグを組み合わせた非アクティブなスナップショットで発生します。

20.39.10. snapshot-delete

virsh snapshot-delete *domain* コマンドは、指定し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に対するスナップショット、または **--current** を使用した現行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スナップショットが含まれる場合は、このスナップショットの変更は子スナップショットにマージされます。**--children** を使用すると、このスナップショットと、その子スナップショットがすべて削除されます。**--children-only** を使用すると、このスナップショットの子はすべて削除されますが、このスナップショットはそのままになります。この 2 つのフラグは相互に排他的です。

この**--metadata**を使用すると、libvirt が管理するスナップショットのメタデータが削除されます。外部ツールによるアクセス用に、スナップショットのコンテンツはそのまま残されます。それ以外の場合は、スナップショットを削除すると、その時点のデータコンテンツも削除されます。

20.40. ゲスト仮想マシンの CPU モデル設定

20.40.1. 導入部分

すべてのハイパーバイザーには、ゲスト仮想マシンがデフォルトで CPU に表示するものに関する独自のポリシーがあります。一部のハイパーバイザーは、ゲスト仮想マシンで使用できる CPU ホスト物理マシンの機能を決定します。一方、QEMU/KVM は、ゲスト仮想マシンを、**qemu32** または **qemu64** という名前の一般的なモデルで表示します。このようなハイパーバイザーは、より高度なフィルタリングを実行し、すべての物理 CPU を少数のグループに分類し、ゲスト仮想マシンに提示される各グループに対してベースライン CPU モデルを 1 つ用意します。このような動作により、ホストの物理マシン間でゲスト仮想マシンを安全に移行できます。ただし、ゲスト仮想マシンすべてに、同じグループに分類される物理 CPU がある場合に限ります。libvirt は通常、ポリシー自体を適用せず、より高いレイヤーで必要なポリシーを定義するメカニズムを提供します。ホストの物理マシン間でゲスト仮想マシンの移行が正常に実行されるようにするには、CPU モデル情報を取得し、適切なゲスト仮想マシンの CPU モデルを定義する方法を理解することが重要になります。ハイパーバイザーは、認識している機能のみをエミュレートでき、ハイパーバイザーがリリースされてから作成された機能はエミュレートできない場合がある点に注意してください。

20.40.2. ホスト物理マシンの CPU モデルの学習

virsh capabilities コマンドは、ハイパーバイザー接続とホスト物理マシンの機能を説明する XML ドキュメントを表示します。表示されている XML スキーマが拡張され、ホスト物理マシンの CPU モデル

に関する情報が提供されるようになりました。CPU モデルを説明する際の課題の 1 つは、すべてのアーキテクチャーで、その機能を公開するアプローチが異なることです。QEMU/KVM および **libvirt** では、CPU モデル名文字列と、名前付きフラグのセットを組み合わせたスキームが使用されます。

既知の CPU モデルをすべてリスト表示するデータベースは実用的ではないので、libvirt のベースライン CPU モデル名のリストは少ないです。CPUID ビットの最大数を実際のホストマシン CPU と共有し、残りのビットを名前付き機能としてリスト表示するものを選択します。libvirt では、ベースライン CPU に含まれる機能が表示されないことに注意してください。一見、これは欠陥のように思われますが、本セクションで説明するように、この情報を実際に把握する必要はありません。

20.40.3. VFIO IOMU デバイスのサポートの決定

virsh domcapabilities コマンドを使用して、VFIO に対応するかどうかを判断します。以下の出力例を参照してください。

図20.3 VFIO のサポートの決定

```
# virsh domcapabilities

[...output truncated...]

<enum name='pciBackend'>
        <value>default</value>
        <value>vfio</value>

[...output truncated...]
```

20.40.4. ホスト物理マシンのプールに対応するための互換性のある CPU モデルの決定

1 台のホスト物理マシンに搭載されている CPU 機能を確認できるようになりました。次の手順は、ゲスト仮想マシンに公開するのに最も適した CPU 機能を判断することです。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ないことが分かっている場合は、ホスト物理マシンの CPU モデルを変更せずにそのまま渡すことができます。仮想化データセンターには、すべてのサーバーで 100% 同一の CPU が保証されるように設定されている場合があります。ここでも、ホスト物理マシンの CPU モデルは、変更せずにそのまま渡すことができます。ただし、より一般的なケースは、ホストの物理マシン間で CPU にバリエーションがある場合です。この混合 CPU 環境では、最小公倍数の CPU を決定する必要があります。これは完全に単純なものではないため、libvirt はこのタスクの API を提供します。libvirt が、ホストの物理マシンの CPU モデルをそれぞれ説明する XML ドキュメントのリストを提供している場合、libvirt は、これらを CPUID マスクに内部的に変換し、その共通部分を計算して、CPUID マスクの結果を XML CPU 説明に戻します。

以下は、**virsh capabilities** の実行時に、libvirt が基本的なワークステーションの機能として報告する内容の例になります。

図20.4 ホストの物理マシンの CPU モデル情報をプルする

```
<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>
```

次に、同じ **virsh capabilities** コマンドを使用して、別のサーバーと比較します。

図20.5 ランダムなサーバーから CPU 説明を生成する

```
<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...
```

この CPU 説明が、以前のワークステーションの CPU 説明と互換性があるかどうかを確認するには、**virsh cpu-compare** コマンドを使用します。

縮小した内容は、**virsh-caps-workstation-cpu-only.xml** という名前のファイルに保存されており、このファイルで **virsh cpu-compare** コマンドを実行できます。

```
# 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
```

この出力から分かるように、libvirt は、CPU との厳密な互換性がないことを正しく報告しています。これは、クライアント CPU にないサーバー CPU の機能が複数あるためです。クライアントとサーバー間で移行を行うには、XML ファイルを開いて、一部の機能をコメントアウトする必要があります。削除する機能を特定するには、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、以下が行われます。

図20.6 複合 CPU ベースライン

```
<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>
```

この複合ファイルでは、共通する要素が示されます。共通していないものはすべてコメントアウトする必要があります。

20.41. ゲスト仮想マシンの CPU モデルの設定

単純なデフォルトの場合、ゲスト仮想マシンの CPU 設定は、ホストの物理マシンの capabilities XML が公開するものと同じ基本的な XML 表現を受け入れます。つまり、**virsh cpu-baseline** コマンドーの XML を、*domain* 要素の最上位にあるゲスト仮想マシン XML に直接コピーできるようになりました。上記の XML スニペットでは、ゲスト仮想マシン XML で CPU を記述する際に利用できる追加の属性がいくつかあります。これらはほとんど無視してもかまいませんが、ここではその実行内容を簡単に説明します。トップレベルの **<cpu>** 要素には、**match** という名前の属性があります。この属性には、以下の値を指定できます。

- match='minimum' - ホストの物理マシン CPU には、ゲスト仮想マシン XML で説明されている CPU 機能が少なくとも必要です。ホストの物理マシンに、ゲスト仮想マシンの設定以外の機能がある場合は、ゲスト仮想マシンにも公開されます。

- match='exact' - ホストの物理マシンの CPU には、ゲスト仮想マシンの XML で説明されている CPU 機能が少なくとも必要です。ホストの物理マシンに、ゲスト仮想マシンの設定以外の機能がある場合は、ゲスト仮想マシンからマスクアウトされます。

- match='strict' - ホストの物理マシン CPU には、ゲスト仮想マシン XML で説明されている CPU 機能とまったく同じものが必要です。

次の機能拡張では、<feature>要素はそれぞれ、以下の可能な値を持つ追加の 'policy' 属性を持つことができます。

- policy='force' - ホストの物理マシンに機能がない場合でも、ゲスト仮想マシンにその機能を公開します。通常、これはソフトウェアエミュレーションの場合にのみ役立ちます。

注記

forceポリシーを使用しても、ハイパーバイザーが特定の機能をエミュレートできない場合があります。

- policy='require' - 機能をゲスト仮想マシンに公開し、ホストの物理マシンにその機能がない場合には失敗します。これは妥当なデフォルトです。

- policy='optional' - 機能に対応している場合にゲスト仮想マシンに公開します。

- policy='disable' - ホストの物理マシンにこの機能がある場合は、ゲスト仮想マシンに表示しないようにします。
- policy='forbid' - ホストの物理マシンにこの機能がある場合は、ゲスト仮想マシンの起動に失敗して拒否されます。

'forbid' ポリシーは、機能が正しくないアプリケーションが CPUID マスクになくても機能を使用しようとし、その機能を持つホストの物理マシンで誤ってゲスト仮想マシンを実行しないようにするニッチのシナリオ向けです。'optional' ポリシーには、移行に関する特別な動作があります。ゲスト仮想マシンが最初に起動するときにこのフラグはオプションですが、ゲスト仮想マシンがライブマイグレーションされるときには、機能が移行中に消えないため、このポリシーは必須になります。

20.42. ゲスト仮想マシンのリソースの管理

virsh では、ゲスト仮想マシンベースでリソースをグループ化し、割り当てることができます。これは、*cgroups* を作成し、ゲスト仮想マシンの代わりにこれを管理する libvirt デーモンによって管理されます。システム管理者ができることは、指定したゲスト仮想マシンに対して調整可能パラメーターを照会するか設定することだけです。**libvirt** サービスは、以下の cgroups を使用して仮想マシンのチューニングと監視を行います。

- **memory** - メモリーコントローラーにより、RAM の制限やスワップの使用量を設定し、グループ内の全プロセスの累積使用量を照会できます。
- **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に対する CPU 使用率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順位を制御します。これは、**nice** レベルの権限を付与するのと似ています。
- **devices** - デバイスコントローラーは、文字デバイス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を一時停止して再開します。これは、グループ全体の **SIGSTOP** と似ています。
- **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で、ネットワークの使用率を管理します。

cgroup は、**libvirt** で **systemd**によって設定されています。以下の **virsh** 調整コマンドは、cgroup の設定方法に影響を及ぼします。

- **schedinfo** - 「スケジュールパラメーターの設定」 で説明しています。
- **blkdeviotune** - 「ディスク I/O スロットリング」 で説明しています。
- **blkiotune** - 「ブロック I/O パラメーターの表示または設定」 で説明しています。
- **domiftune** - 「ネットワークインターフェイスの帯域幅パラメーターの設定」 で説明しています。
- **memtune** - 「メモリーの調整の設定」 で説明しています。

cgroups の詳細は、Red Hat Enterprise Linux 7 Resource Management Guide を参照してください。

20.43. スケジュールパラメーターの設定

virsh schedinfo コマンドは、ホストマシンの仮想マシンプロセスのホストスケジューリングパラメーターを変更します。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 --current --config --live
```

各パラメーターの説明を以下に示します。

- *domain* - ゲスト仮想マシンドメイン
- **--set** - ここに置かれるストリングは、呼び出されるコントローラーまたはアクションです。このストリングは、*parameter=value* 形式を使用します。必要な場合には、追加のパラメーターまたは値も追加する必要があります。
- **--current** - **--set** と一緒に使用すると、指定した**set**文字列を現在のスケジューラー情報として使用します。一緒に使用しない場合は、現在のスケジューラー情報が表示されます。
- **--config** - - **--set** と併用すると、次回のシステムの再起動時に指定した**set** 文字列を使用します。一緒に使用しない場合は、設定ファイルに保存されているスケジューラー情報が表示されます。
- **--live** - **--set** と一緒に使用すると、現在実行しているゲスト仮想マシンで指定した**set** 文字列を使用します。一緒に使用しない場合は、実行中の仮想マシンが現在使用している設定が表示されます。

スケジューラーは、***cpu_shares***、***vcpu_period***、および ***vcpu_quota*** のパラメーターのいずれかで設定できます。このパラメーターは、vCPU スレッドに適用されます。

以下は、パラメーターが cgroup フィールド名にマッピングされる方法を示しています。

- ***cpu_shares***:cpu.shares
- ***vcpu_period***:cpu.cfs_period_us
- ***vcpu_quota***:cpu.cfs_quota_us

例20.98 schedinfo show

この例は、シェルゲスト仮想マシンのスケジュール情報を示しています。

```
# virsh schedinfo shell
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
```

例20.99 schedinfo set

この例では、cpu_shares が 2046 に変更されています。これは現在の状態に影響しますが、設定ファイルには影響しません。

```
# virsh schedinfo --set cpu_shares=2046 shell
Scheduler      : posix
```

```
cpu_shares    : 2046
vcpu_period   : 100000
vcpu_quota    : -1
```

libvirt は、エミュレータープロセスの設定を変更する ***emulator_period*** パラメーターおよび ***emulator_quota*** パラメーターにも対応しています。

20.44. ディスク I/O スロットリング

virsh blkdeviotune コマンドは、指定されたゲスト仮想マシンのディスク I/O スロットリングを設定します。これにより、ゲスト仮想マシンが共有リソースを過剰に使用しなくなり、他のゲスト仮想マシンのパフォーマンスに影響を与える可能性があります。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。ドメイン名のリストを表示するには、**virsh domblklist** コマンドを実行します。**--config**、**--live**、および **--current** の引数は、「スケジュールパラメーターの設定」 と同じように機能します。制限を指定しないと、現在の I/O 制限設定のクエリーが実行されます。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** - 合計スループット制限を 1 秒あたりのバイト数で指定します。
- **--read-bytes-sec** - 読み取りスループットの上限をバイト/秒単位で指定します。
- **--write-bytes-sec** - 書き込みスループットの制限をバイト/秒単位で指定します。
- **--total-iops-sec** - 1 秒あたりの I/O 操作の制限合計を指定します。
- **--read-iops-sec** - 1 秒あたりの読み取り I/O 操作の制限を指定します。
- **--write-iops-sec** - 1 秒あたりの書き込み I/O 操作の制限を指定します。

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。サンプルドメイン XML は、図23.27「デバイス - ハードドライブ、フロッピーディスク、CD-ROM の例」 を参照してください。

20.45. ブロック I/O パラメーターの表示または設定

blkiotune コマンドは、指定されたゲスト仮想マシンの I/O パラメーターを設定または表示します。以下の形式を使用する必要があります。

```
# virsh blkiotune domain [--weight weight] [--device-weights device-weights] [---device-read-iops-sec -device-read-iops-sec] [--device-write-iops-sec device-write-iops-sec] [--device-read-bytes-sec device-read-bytes-sec] [--device-write-bytes-sec device-write-bytes-sec] [[--config] [--live] | [--current]]
```

このコマンドの詳細は、『Virtualization Tuning and Optimization Guide』 を参照してください。

20.46. メモリーの調整の設定

virsh memtune virtual_machine --parameter size は、『Virtualization Tuning and Optimization Guide』 で説明しています。

第21章 オフラインツールを使用したゲスト仮想マシンのディスクアクセス

21.1. 導入部分

Red Hat Enterprise Linux 7 には、ゲスト仮想マシンのディスクやその他のディスクイメージのアクセス、編集、および作成を可能にする **libguestfs** ユーティリティーが多数用意されています。このツールには、以下のような複数の用途があります。

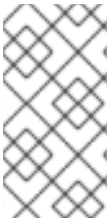
- ゲスト仮想マシンのディスクにあるファイルの表示またはダウンロード
- ゲスト仮想マシンのディスクでのファイルの編集またはアップロード。
- ゲスト仮想マシンの設定の読み取りまたは書き込み。
- ファイル、ディレクトリー、ファイルシステム、パーティション、論理ボリューム、およびその他のオプションを含む新しいディスクイメージの準備
- 起動に失敗したゲスト仮想マシン、または起動設定の変更が必要なゲスト仮想マシンの復旧および修復
- ゲスト仮想マシンのディスク使用状況の監視
- 組織のセキュリティー標準など、ゲスト仮想マシンのコンプライアンスの監査
- テンプレートのクローンを作成して変更し、ゲスト仮想マシンをデプロイする。
- CD および DVD ISO のイメージおよびフロッピーディスクイメージの読み取り。

警告

この章に記載されているユーティリティーを使用して、実行中の仮想マシンに接続され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ったり、書き込みモードでそのようなディスクイメージを開いたり **しないでください**。

実行すると、ゲスト仮想マシンのディスクが破損します。このツールはこれを防ぐように試みますが、すべてのケースを保護するわけではありません。ゲスト仮想マシンが実行中だと疑われる場合は、Red Hat はユーティリティーを使用しないことを強く推奨します。

安全性を向上させるために、特定のユーティリティーは、(**--ro** オプションを使用して) 読み取り専用モードで使用できますが、変更を保存しません。



注記

libguestfs ドキュメントと関連ユーティリティーの主なソースは、Linux の man ページです。API は **guestfs(3)**、guestfish は **guestfish(1)**、仮想化ユーティリティーは独自の man ページ (**virt-df(1)** など) で説明されています。トラブルシューティング情報は、「libguestfs トラブルシューティング」 を参照してください。

21.1.1. リモート接続の使用に関する注意

Red Hat Enterprise Linux 7 の仮想化コマンドの中には、リモートの libvirt 接続を指定できるものがあります。以下に例を示します。

```
# virt-df -c  qemu://remote/system -d Guest
```

ただし、Red Hat Enterprise Linux 7 の libguestfs ユーティリティーは、リモートの libvirt ゲストのディスクにはアクセスできません。また、上記のようにリモート URL を使用するコマンドは想定どおりに機能しません。

ただし、Red Hat Enterprise Linux 7 以降、libguestfs はネットワークブロックデバイス (NBD) を介してリモートディスクソースにアクセスできます。**qemu-nbd** コマンドを使用して、リモートマシンからディスクイメージをエクスポートし、**nbd:**// URL を使用してそのイメージ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ります。

リモートシステムの場合: **qemu-nbd -t disk.img**

ローカルシステムの場合: **virt-df -a nbd://remote**

影響を受けるのは、次の libguestfs コマンドの場合です。

- 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

21.2. 用語

本セクションでは、本章全体で使用される用語を説明します。

- **libguestfs (GUEST FileSystem LIBrary)**- ディスクイメージを開いたり、ファイルを読み書きしたりするために使用される基本的な機能を提供する C ライブラリーです。C のプログラムは、この API に直接書き込むことができます.。
- **guestfish (GUEST Filesystem Interactive SHell)**は、コマンドラインまたはシェルスクリプトから使用できる対話式のシェルです。これは、libguestfs API のすべての機能を公開します。
- さまざまな virt ツールが libguestfs の上に構築され、コマンドラインから特定の 1 つのタスクを実行する方法を提供します。このようなツールには、**virt-df**、**virt-rescue**、**virt-resize**、および **virt-edit** が含まれます。
- **augeas** は、Linux 設定ファイルを編集するためのライブラリーです。libguestfs とは異なりますが、libguestfs の値のほとんどは、このツールを使用した組み合わせによるものです。
- **guestmount** は、libguestfs と FUSE との間のインターフェイスです。これは、主に、ホストの物理マシン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使用されます。この機能は必須ではありませんが、便利です。

21.3. インストール

libguestfs、guestfish、libguestfs ツール、および guestmount 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 libguestfs-tools
```

言語バインディングを含むすべての libguestfs 関連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 '*guestf*'
```

21.4. GUESTFISH シェル

guestfish は、コマンドラインまたはシェルスクリプトから、ゲスト仮想マシンファイルシステムにアクセスするために使用できるインタラクティブなシェルです。libguestfs API の機能はすべて、シェルから利用できます。

仮想マシンのディスクイメージの表示または編集を開始するには、次のコマンドを実行して、使用するディスクイメージのパスを置き換えます。

```
$ guestfish --ro -a /path/to/disk/image
```

--ro は、ディスクイメージが読み取り専用で開かれていることを意味します。このモードは常に安全ですが、書き込みアクセスは許可しません。ゲスト仮想マシンが実行されていないことが**確実**な場合、またはディスクイメージがライブゲスト仮想マシンに接続されていない場合にのみ、このオプションを省略してください。**libguestfs** を使用してライブゲスト仮想マシンを編集することはできません。これを試みると、ディスクが破損し、元に戻せなくなります。

/path/to/disk/image は、ディスクへのパスです。これには、ファイル、ホストの物理マシン論理ボリューム (/dev/VG/LV など)、または SAN LUN (/dev/sdf3) があります。

注記

libguestfs および guestfish には root 権限は必要ありません。アクセスされているディスクイメージが root から読み取りまたは書き込み、あるいは両方を必要とする場合にのみ、これらを root として実行する必要があります。

guestfish を対話的に起動すると、以下のプロンプトが表示されます。

```
$ guestfish --ro -a /path/to/disk/image

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

><fs>
```

プロンプトで **実行** と入力して、ライブラリーを開始し、ディスクイメージを割り当てます。これは、最初に実行するときに最大 30 秒かかることがあります。その後の起動は、非常に速く完了します。

注記

libguestfs は、KVM (利用可能な場合) などのハードウェア仮想化アクセラレーションを使用して、このプロセスを高速化します。

run コマンドを入力すると、以下のセクションで示すように、他のコマンドを使用できます。

21.4.1. guestfish を使用したファイルシステムの表示

このセクションでは、guestfish でファイルシステムを表示する方法を説明します。

21.4.1.1. 手動によるリスト表示と表示

list-filesystems コマンドは、libguestfs が検出したファイルシステムのリストを表示します。この出力は、Red Hat Enterprise Linux 4 ディスクイメージを示しています。

```
><fs> run
><fs> list-filesystems
/dev/vda1: ext3
/dev/VolGroup00/LogVol00: ext3
/dev/VolGroup00/LogVol01: swap
```

その他の便利なコマンドは、**list-devices**、**list-partitions**、**lvs**、**pvs**、**vfs-type**、および **file** です。**help** *command* と入力すると、以下の出力のように、コマンドの詳細とヘルプを表示できます。

```
><fs> help vfs-type
 NAME
    vfs-type - get the Linux VFS type corresponding to a mounted device

 SYNOPSIS
     vfs-type mountable
```

```
DESCRIPTION
    This command gets the filesystem type corresponding to the filesystem on
    "device".

    For most filesystems, the result is the name of the Linux VFS module
    which would be used to mount this filesystem if you mounted it without
    specifying the filesystem type. For example a string such as "ext3" or
    "ntfs".
```

ファイルシステムの実際の内容を表示するには、最初にマウントする必要があります。

ls、**ll**、**cat**、その**more**、**download**、**tar-out** などの guestfish コマンドを使用して、ファイルおよびディレクトリーを表示およびダウンロードできます。

注記

このシェルには、現在の作業ディレクトリーの概念はありません。通常のシェルとは異なり、**cd** コマンドを使用してディレクトリーを変更することはできません。すべてのパスは、先頭のスラッシュ (/) 文字で始まる完全修飾する必要があります。**Tab** 鍵を使用して、パスを完成させます。

guestfish シェルを終了するには、**exit** と入力するか、**Ctrl+d** と入力します。

21.4.1.2. guestfish 検査経由

ファイルシステムを手動でリスト表示してマウントする代わりに、guestfish 自体がイメージを検証して、ゲスト仮想マシンのようにファイルシステムをマウントできます。これを行うには、コマンドラインに **-i** を追加します。

```
$ guestfish --ro -a /path/to/disk/image -i

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

 Operating system: Red Hat Enterprise Linux AS release 4 (Nahant Update 8)
/dev/VolGroup00/LogVol00 mounted on /
/dev/vda1 mounted on /boot

><fs> ll /
total 210
drwxr-xr-x. 24 root root  4096 Oct 28 09:09 .
drwxr-xr-x  21 root root  4096 Nov 17 15:10 ..
drwxr-xr-x.  2 root root  4096 Oct 27 22:37 bin
drwxr-xr-x.  4 root root  1024 Oct 27 21:52 boot
drwxr-xr-x.  4 root root  4096 Oct 27 21:21 dev
drwxr-xr-x. 86 root root 12288 Oct 28 09:09 etc
...
```

インスペクションおよびマウントを実行するために、guestfish 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** コマンドを使用する必要はありません。**-i** オプションは、多くの一般的な Linux ゲスト仮想マシンで機能します。

21.4.1.3. 名前によるゲスト仮想マシンのアクセス

ゲスト仮想マシンは、libvirt として知られる名前 (つまり **virsh list --all** にあるように) を指定するとコマンドラインからアクセスできます。**-d** オプションを使用して、ゲスト仮想マシンに名前 (**-i** オプションの有無にかかわらず) でアクセスします。

```
$ guestfish --ro -d GuestName -i
```

21.4.2. guestfish を使用したファイルの追加

guestfish でファイルを追加するには、完全な URI が必要です。ファイルは、ローカルファイル、ネットワークブロックデバイス (NBD) またはリモートブロックデバイス (RBD) に置かれたファイルのいずれかになります。

URI に使用する形式は、以下のいずれかになります。ローカルファイルの場合は ///: を使用します。

- **guestfish -a *disk*.img**
- **guestfish -a file:///*directory*/*disk*.img**
- **guestfish -a nbd://*example.com*[:*port*]**
- **guestfish -a nbd://*example.com*[:*port*]/*exportname***
- **guestfish -a nbd://?socket=/*socket***
- **guestfish -a nbd:///*exportname*?socket=/*socket***
- **guestfish -a rbd:///*pool*/*disk***
- **guestfish -a rbd://*example.com*[:*port*]/*pool*/*disk***

21.4.3. guestfish を使用したファイルの変更

ファイルを変更したり、ディレクトリーを作成したり、ゲスト仮想マシンにその他の変更を加えたりする場合は、最初に本セクションの冒頭にある ゲスト仮想マシンをシャットダウンする必要がある に関する警告に留意してください。guestfish で実行中のディスクを編集または変更すると、ディスクが **破損します**。ここでは、**/boot/grub/grub.conf** ファイルーを編集する例を説明します。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たら、以下のようなコマンドを使用して書き込みアクセスを取得するために **--ro** フラグを省略できます。

```
$ guestfish -d RHEL3 -i

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

 Operating system: Red Hat Enterprise Linux AS release 3 (Taroon Update 9)
```

```
 /dev/vda2 mounted on /
 /dev/vda1 mounted on /boot

><fs> edit /boot/grub/grub.conf
```

ファイルを編集するコマンドには、**edit**、**vi**、および **emacs** が含まれます。**write**、**mkdir**、**upload**、**tar-in** など、ファイルとディレクトリーを作成する多くのコマンドも存在します。

21.4.4. guestfish でのその他のアクション

また、**mkfs**、**part-add**、**lvresize**、**lvcreate**、**vgcreate**、**pvcreate** などのコマンドを使用して、ファイルシステムのフォーマットを設定し、パーティションを作成し、LVM 論理ボリュームなどのサイズを変更できます。

21.4.5. guestfish によるシェルスクリプト設定

guestfish を対話的に使用することに慣れたら、必要に応じてシェルスクリプトの記述が役に立つ場合があります。以下は、新しい MOTD (message of the day) をゲストに追加する単純なシェルスクリプトです。

```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$guestname" -i <<'EOF'
   write /etc/motd "Welcome to Acme Incorporated."
   chmod 0644 /etc/motd
 EOF
```

21.4.6. Augeas スクリプトと libguestfs スクリプト

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役立ちます。たとえば、次のスクリプトは、Augeas を使用し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。この例は、Red Hat Enterprise Linux を実行してい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、設定ファイルの変更にも使用できます。上記のスクリプトを修正して、キーボードレイアウトを変更できます。

```
#!/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
```

2 つのスクリプト間で、以下の 3 つの変更が行われたことに注意してください。

1. 2 番目の例では **--ro** オプションが削除され、ゲスト仮想マシンに書き込む機能が利用できるようになりました。
2. **aug-get** コマンドが、フェッチではなく値を変更するように **aug-set** に変更されました。新しい値は**"gb"** (引用符を含む) になります。
3. ここでは **aug-save** コマンドを使用しているため、Augeas は変更をディスクに書き込みます。

注記

Augeas の詳細は、Web サイト http://augeas.net を参照してください。

guestfish には、本書で説明する以上の機能があります。たとえば、最初からディスクイメージを作成する場合は、次のコマンドを実行します。

```
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する場合は、次のコマンドを実行します。

```
><fs> copy-out /home /tmp/home
```

詳細は、man ページの guestfish(1) を参照してください。

21.5. その他のコマンド

本セクションでは、guestfish を使用してゲスト仮想マシンのディスクイメージを表示および編集するのと同等の、簡単なツールを説明します。

- **virt-cat** は、guestfish **download** コマンドに似ています。ゲスト仮想マシンに 1 つのファイルをダウンロードして表示します。以下に例を示します。

  ```
  # virt-cat RHEL3 /etc/ntp.conf | grep ^server
   server     127.127.1.0       # local clock
  ```

- **virt-edit** は guestfish **edit** コマンドと似ています。これを使用すると、ゲスト仮想マシン内の 1 つのファイルを対話形式で編集できます。たとえば、Linux ベースのゲスト仮想マシンで、起動しない**grub.conf** ファイルを修正する必要がある場合があります。

  ```
  # virt-edit LinuxGuest /boot/grub/grub.conf
  ```

  virt-edit には別のモードがあり、このモードを使用すると、1 つのファイルにシンプルな非対話的な変更を加えることができます。これには、**-e** オプションが使用されます。たとえば、次のコマンドは、Linux ゲスト仮想マシンの root パスワードを、パスワードを持たないように変更します。

  ```
  # virt-edit LinuxGuest /etc/passwd -e 's/^root:.*?:/root::/'
  ```

- **virt-ls** は guestfish の **ls** コマンド、**ll** コマンド、および **find** コマンドに似ています。これは、ディレクトリーを (再帰的に) リスト表示するために使用されます。たとえば、以下のコマンドは、Linux ゲスト仮想マシンの /home 配下にファイルとディレクトリーを再帰的にリスト表示します。

```
# virt-ls -R LinuxGuest /home/ | less
```

21.6. VIRT-RESCUE: レスキューシェル

このセクションは、レスキューシェルに関する情報を提供します。

21.6.1. 導入部分

本セクションでは、仮想マシンのレスキュー CD と似ていると思われる **virt-rescue** を説明します。ゲスト仮想マシンをレスキューシェルで起動して、メンテナンスを実行してエラーを修正し、ゲスト仮想マシンを修復できるようにします。

virt-rescue と guestfish には、重複する部分があります。virt-rescue は、通常の Linux ファイルシステムツールを使用して対話型のアドホックな変更を行うために使用されます。これは、障害が発生したゲスト仮想マシンのレスキューにとりわけ適しています。virt-rescue はスクリプトを使用できません。

一方、guestfish は対話的に使用することもできますが、正式なコマンドセット (libguestfs API) を使用して、スクリプトを使用し、構造化した変更を行う場合にとりわけ役立ちます。

21.6.2. virt-rescue の実行

ゲスト仮想マシンで **virt-rescue** を使用する前に、ゲスト仮想マシンが実行していないことを確認してください。そうでないと、ディスクが破損します。ゲスト仮想マシンが稼働していないことを確認したら、次のコマンドを実行します。

```
$ virt-rescue -d GuestName
```

(GuestName は、libvirt が認識しているゲスト名です。) または、以下のコマンドを実行します。

```
$ virt-rescue -a /path/to/disk/image
```

(パスは、ゲスト仮想マシンのディスクを含むファイル、論理ボリューム、LUN などを指定できます)。

virt-rescue がレスキュー仮想マシンを起動すると、最初に出力がスクロールして過去のものが表示されます。最後に、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通常の bash シェルで、通常の Red Hat Enterprise Linux コマンドを減らしたものが利用できます。たとえば、次のように入力します。

```
><rescue> fdisk -l /dev/vda
```

上記のコマンドは、ディスクパーティションのリストを表示します。ファイルシステムをマウントするには、**/sysroot** にマウントすることが推奨されます。これは、ユーザーが好きなものをマウントするためのレスキューマシンの空のディレクトリーです。/ のファイルは、レスキュー仮想マシン自体のファイルであ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レスキューが終了したら、**exit** または **Ctrl+d** を入力してシェルを終了します。

virt-rescue には多くのコマンドラインオプションがあります。最もよく使用されるオプションは以下のとおりです。

- **--ro**: ゲスト仮想マシンで読み取り専用モードで操作します。変更は保存されません。これを使用して、ゲスト仮想マシンを実験できます。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有効にします。RPM またはその他のファイルをゲスト仮想マシンにダウンロードする必要がある場合などに使用します。

21.7. VIRT-DF: ディスク使用量のモニタリング

このセクションでは、ディスクの使用状況の監視に関する情報を説明します。

21.7.1. 導入部分

本セクションでは、ディスクイメージまたはゲスト仮想マシンからファイルシステムの使用状況を表示する **virt-df** を説明します。Linux の **df** コマンドと似ていますが、これは仮想マシン用です。

21.7.2. virt-df の実行

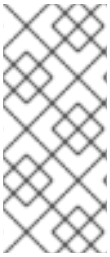
ディスクイメージにあるすべてのファイルシステムのファイルシステム使用状況を表示するには、次のコマンドを実行します。

```
# 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 ゲスト仮想マシンディスクイメージです。この場合のパスは、このディスクイメージが置かれているホストの物理マシンの論理ボリュームです。)

virt-df のみを使用して、libvirt が認識しているゲスト仮想マシンの情報をすべてリスト表示することもできます。**virt-df** は、**-h** (人間が判読できる) や **-i** (ブロックの代わりに inode を表示) など、通常の**df**と同じオプションの一部を認識します。

```
# virt-df -h -d domname
 Filesystem                      Size       Used  Available  Use%
 F14x64:/dev/sda1                  484.2M      66.3M     392.9M   14%
 F14x64:/dev/vg_f14x64/lv_root    7.4G       3.0G       4.4G   41%
 RHEL6brewx64:/dev/sda1          484.2M      52.6M     406.6M   11%
 RHEL6brewx64:/dev/vg_rhel6brewx64/lv_root
                              13.3G       3.4G       9.2G   26%
```



注記

virt-df は読み取り専用アクセスのみが必要であるため、ライブゲスト仮想マシンでは安全に使用できます。ただし、ゲスト仮想マシン内で実行している **df** コマンドと同じ数字になることを想定することはできません。これは、ディスク上のものが、ライブゲスト仮想マシンの状態とわずかに同期が外れるためです。それでも、分析と監視の目的には十分な近似値となります。

virt-df は、統計を監視ツールやデータベースなどに統合できるように設計されています。これにより、システム管理者は、ディスク使用状況の傾向に関するレポートを生成し、ゲスト仮想マシンでディスク領域が不足するとアラートを生成できます。これを行うには、**--csv** を使用して、マシンが判読可能なコンマ区切り値 (CSV) 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、スプレッドシートソフトウェア、およびその他のさまざまなツールとプログラミング言語で読み取り可能です。raw の CSV は以下のようになります。

```
# virt-df --csv -d RHEL6Guest
 Virtual Machine,Filesystem,1K-blocks,Used,Available,Use%
 RHEL6brewx64,/dev/sda1,102396,24712,77684,24.1%
 RHEL6brewx64,/dev/sda2,20866940,7786652,13080288,37.3%
```

21.8. VIRT-RESIZE: ゲスト仮想マシンのオフラインでのサイズ変更

本セクションでは、オフラインのゲスト仮想マシンのサイズを変更する方法を説明します。

21.8.1. 導入部分

本セクションでは、ゲスト仮想マシンを拡張または縮小するツールである **virt-resize** を説明します。これは、オフラインのゲスト仮想マシン (シャットダウン) でのみ機能します。これは、ゲストの仮想マシンイメージをコピーして、元のディスクイメージをそのままにして動作します。元のイメージをバックアップとして使用できるため、これは理想的ですが、2 倍のディスク容量が必要になるというトレードオフがあります。

21.8.2. ディスクイメージの拡張

このセクションでは、ディスクイメージを拡張する簡単な例を示します。

1. サイズを変更するディスクイメージを探します。コマンド **virsh dumpxml GuestName** は、libvirt ゲスト仮想マシンに使用できます。

2. ゲスト仮想マシンを拡張する方法を決定します。以下のように、ゲスト仮想マシンディスクで **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
```

以下の例は、以下の方法を示しています。

- 最初の (ブート) パーティションのサイズを、約 100MB から 500MB に増やします。
- ディスクの合計サイズを 8GB から 16GB に増やします。
- 2 番目のパーティションをデプロイメントして、残りの領域を埋めます。
- **/dev/VolGroup00/LogVol00** をデプロイメントし、2 番目のパーティションの新しい領域を埋めます。

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. 元のディスクの名前をバックアップとして変更します。これを行う方法は、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。ファイルとして保存されている場合は、**mv** コマンドを実行します。(この例で示しているように) 論理ボリュームの場合は、**lvrename** を使用します。

   ```
   # lvrename /dev/vg_guests/RHEL6 /dev/vg_guests/RHEL6.backup
   ```

3. 新しいディスクを作成します。この例の要件は、ディスクの合計サイズを最大 16GB まで拡張することです。論理ボリュームが使用されるため、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. 手順 2 の要件は、以下のコマンドで表されます。

   ```
   # virt-resize \
         /dev/vg_guests/RHEL6.backup /dev/vg_guests/RHEL6 \
         --resize /dev/sda1=500M \
         --expand /dev/sda2 \
         --LV-expand /dev/VolGroup00/LogVol00
   ```

   最初の 2 つの引数は、入力ディスクと出力ディスクです。**--resize /dev/sda1=500M** は、最初のパーティションのサイズを最大 500 MB に変更します。**--expand /dev/sda2** は、残りのすべての領域を埋めるために 2 番目のパーティションを拡張します。**--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て、2 番目のパーティションの余分な領域を埋めます。

   virt-resize は、出力で実行している処理を説明します。

   ```
   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てください。正常に機能している場合 (および徹底的なテストを行った後) は、バックアップディスクを削除できます。失敗した場合は、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、バックアップディスクの名前を元の名前に戻します。

6. **virt-df** または **virt-filesystems** を使用して、新しいサイズを表示します。

```
# 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を変更すると、問題が発生する場合があることに注意してください。**virt-resize** が失敗した場合は、virt-resize(1) の man ページに、確認して試すことができるヒントが複数あります。一部の古い Red Hat Enterprise Linux ゲスト仮想マシンに場合は、GRUB に関するヒントに特に注意を払う必要がある場合があります。

21.9. VIRT-INSPECTOR: ゲスト仮想マシンの検査

このセクションでは、ゲスト仮想マシンの検証について説明します。

21.9.1. 導入部分

virt-inspector は、ディスクイメージを調べて、そこに含まれるオペレーティングシステムを確認するツールです。

21.9.2. インストール

virt-inspector とドキュメントをインストールするには、次のコマンドを入力します。

```
# yum install libguestfs-tools
```

サンプルの XML 出力や、出力の Relax-NG スキーマを含むドキュメントが、**/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えられます。

21.9.3. virt-inspector の実行

以下の例に示すように、**virt-inspector** は任意のディスクイメージまたは libvirt ゲスト仮想マシンに対して実行できます。

```
$ virt-inspector -a disk.img > report.xml
```

または、以下のようになります。

```
$ virt-inspector -d GuestName > report.xml
```

結果は XML レポート (**report.xml**) になります。XML ファイルの主なコンポーネントは、以下のような、通常 1 つの**<operatingsystem>** 要素を含むトップレベルの**<operatingsytems>** 要素です。

```
<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>
```

これらのレポートの処理は、W3C 標準の XPath クエリーを使用して行うのが最適です。Red Hat Enterprise Linux 7 には、シンプルなインスタンスに使用できる **xpath** コマンドラインプログラムが同梱されています。ただし、長期間使用し、高度な使用方法を利用する場合は、XPath ライブラリーと、お気に入りのプログラミング言語を使用することを検討してください。

たとえば、以下の XPath クエリーを使用して、すべてのファイルシステムデバイスをリスト表示できます。

```
$ virt-inspector GuestName | xpath //filesystem/@dev
 Found 3 nodes:
 -- NODE --
 dev="/dev/sda1"
 -- NODE --
 dev="/dev/vg_f12x64/lv_root"
 -- NODE --
 dev="/dev/vg_f12x64/lv_swap"
```

または、次のコマンドを実行してインストールしたすべてのアプリケーションの名前をリスト表示します。

```
$ virt-inspector GuestName | xpath //application/name
 [...long list...]
```

21.10. プログラミング言語からの API の使用

libguestfs API は、Red Hat Enterprise Linux 7 の次の言語から直接使用できます: C、C ++、Perl、Python、Java、Ruby、および OCaml

- C バインディングおよび C++ バインディングをインストールするには、以下のコマンドを実行します。

  ```
  # yum install libguestfs-devel
  ```

- Perl バインディングをインストールするには、以下のコマンドを実行します。

  ```
  # yum install 'perl(Sys::Guestfs)'
  ```

- Python バインディングをインストールするには、以下のコマンドを実行します。

  ```
  # yum install python-libguestfs
  ```

- Java バインディングをインストールするには、以下のコマンドを実行します。

  ```
  # 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には、以下のコマンドを実行します。

  ```
  # yum install ruby-libguestfs
  ```

- OCaml バインディングをインストールするには、以下のコマンドを実行します。

  ```
  # yum install ocaml-libguestfs ocaml-libguestfs-devel
  ```

各言語のバインディングは基本的に同じですが、構文が少し変更されています。C ステートメント:

```
guestfs_launch (g);
```

Perl では、以下のように表示されます。

```
$g->launch ()
```

または、OCaml で以下のようになります。

```
g#launch ()
```

本セクションでは、C の API のみを説明します。

C バインディングおよび C++ バインディングでは、エラーを手動で確認する必要があります。他のバインディングでは、エラーは例外に変換されます。以下の例で示している追加のエラーチェックは他の言語には必要ありませんが、逆に例外をキャッチするコードを追加することもできます。libguestfs API のアーキテクチャーに関する肝心な点については、以下のリストを参照してください。

- libguestfs API は同期的です。各呼び出しは、完了するまでブロックされます。非同期で呼び出しを行う場合は、スレッドを作成する必要があります。
- libguestfs の API はスレッドセーフではありません。各ハンドルは、1 つのスレッドからのみ使用する必要があります。または、スレッド間でハンドルを共有する場合は、独自のミューテックスを実装して、2 つのスレッドが同時に 1 つのハンドルでコマンドを実行できないようにします。
- 同じディスクイメージで複数のハンドルを開くことはできません。すべてのハンドルが読み取り専用である場合は許容されますが、推奨されません。
- そのディスクイメージ (たとえば、ライブ仮想マシン) を使用するものがない場合は、書き込み用にディスクイメージを追加しないでください。これを行うと、ディスクが破損します。
- 現在使用中のディスクイメージ (ライブ仮想マシンなど) で読み取り専用ハンドルを開くことができます。ただし、特にディスクイメージの読み取り時にそのディスクイメージが大量に書き込まれている場合は、結果が予測できなかったり、矛盾することがあります。

21.10.1. C プログラムを使用した API との相互作用

C プログラムは、**<guestfs.h>** ヘッダーファイルを含めることから開始して、ハンドルを作成する必要があります。

```
#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、以下の 2 つのコマンドを実行します。

```
gcc -Wall test.c -o test -lguestfs
./test
```

この段階では、出力はありません。このセクションの残りの部分では、このプログラムを拡張して新しいディスクイメージを作成し、パーティションを作成し、ext4 ファイルシステムでフォーマットして、ファイルシステムにいくつかのファイルを作成する方法を示す例を紹介します。ディスクイメージは**disk.img** と呼ばれ、現在のディレクトリーに作成されます。

このプログラムの概要は、以下のとおりです。

- ハンドルを作成します。
- ハンドルにディスクを追加します。
- libguestfs バックエンドを起動します。
- パーティション、ファイルシステム、およびファイルを作成します。
- ハンドルを閉じて、終了します。

変更したプログラムを以下に示します。

```
#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);
}
```

このプログラムをコンパイルして、以下の 2 つのコマンドを実行します。

```
gcc -Wall test.c -o test -lguestfs
./test
```

プログラムが正常に完了すると、**disk.img** というディスクイメージが残ります。これは 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 バインディングおよび C++ バインディングのみ)、libguestfs は stderr にエラーを出力します。この動作は、エラーハンドラーを設定することで変更できます。man ページの guestfs(3) では、これを詳細に説明しています。

21.11. VIRT-SYSPREP: 仮想マシン設定のリセット

virt-sysprep コマンドラインツールを使用すると、ゲスト仮想マシンをリセットまたは設定解除して、クローンを作成できます。このプロセスには、SSH ホストの鍵の削除、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。virt-sysprep は、SSH キー、ユーザー、ロゴなどを追加して、仮想マシンをカスタマイズすることもできます。各手順は、必要に応じて有効または無効にできます。

virt-sysprep を使用するには、ゲスト仮想マシンがオフラインである必要があるため、シャットダウンしてからコマンドを実行してください。**virt-sysprep**は、ゲストまたはディスクイメージをコピーせずに、その場で変更することに注意してください。ゲスト仮想マシンの既存のコンテンツを保持する場合は、最初にディスクのスナップショット、コピー、またはクローンを作成する必要があります。ディスクのコピーおよびクローン作成の詳細は、libguestfs.org を参照してください。

ディスクイメージにアクセスするために root が必要でない限り、root として**virt-sysprep** を使用しないことが推奨されます。ただし、このような場合は、**virt-sysprep** を実行している root 以外のユーザーがディスクイメージに書き込み可能なパーミッションを変更することが推奨されます。

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。

```
# yum install /usr/bin/virt-sysprep
```

virt-sysprep では、以下のコマンドオプションを使用できます。

表21.1virt-sysprepコマンド

コマンド	説明	例
--help	特定のコマンドまたは **virt-sysprep** コマンドに関する簡単なヘルプエントリーを表示します。詳細なヘルプは、man ページの virt-sysprep を参照してください。	**virt-sysprep --help**
-a [*file*] or --add [*file*]	指定された*file*を追加します。これは、ゲスト仮想マシンからのディスクイメージである必要があります。ディスクイメージの形式は自動検出されます。これを上書きして、特定の形式に強制する場合は、**--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 rbd://example.com[:port]/pool/disk**

コマンド	説明	例
-c [*URI*] or --connect [*URI*]	libvirt を使用する場合は、指定した URI に接続します。省略した場合は、KVM ハイパーバイザーを介して接続します。ゲストブロックデバイスを直接指定 (**virt-sysprep -a**) すると、libvirt はまったく使用されません。	**virt-sysprep -c qemu:///system**
-d [*guest*] or --domain [*guest*]	指定したゲスト仮想マシンのすべてのディスクを追加します。ドメイン名の代わりにドメイン UUID を使用できます。	**virt-sysprep --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***
-n or --dry-run	ゲスト仮想マシンで、読み取り専用の "dry run" sysprep 操作を実行します。これにより sysprep 操作が実行されますが、末尾のディスクに対する変更はすべて破棄されます。	**virt-sysprep -n**
--enable [*operations*]	指定した*operations*を有効にします。可能な操作のリストを表示するには、--list コマンドを使用します。	**virt-sysprep --enable ssh-hostkeys,udev-persistent-net**
--operation または --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 の操作を有効にし、tmp-files の操作を無効にします。有効な操作のリストは、libguestfs.org を参照してください。

コマンド	説明	例
--format [**raw**|**qcow2**|**auto**]	-a オプションのデフォルトは、ディスクイメージの形式を自動検出することです。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されます。--format auto を使用すると、後続の -a オプションの自動検出に戻ります (上述の -a コマンドを参照)。	**virt-sysprep --format raw -a *disk.img*** は、disk.img に対して 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** に対して raw 形式 (自動検出なし) を強制し、**another.img** では自動検出に戻ります。信頼できない raw 形式のゲストディスクイメージがある場合は、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、悪意のあるゲストで発生する可能性のあるセキュリティーの問題を回避できます。*
--list-operations	virt-sysprep プログラムでサポートされている操作をリスト表示します。これらは、1 行に 1 つずつ表示され、1 つ以上の空白で区切られたフィールドがあります。出力の最初のフィールドは操作名であり、**--enable** フラグに指定できます。2 番目のフィールドは、操作がデフォルトで有効になっている場合は * 文字で、そうでない場合は空白になります。同じ行のその他のフィールドには、操作の説明が記載されています。	**virt-sysprep --list-operations**
--mount-options	ゲスト仮想マシンの各マウントポイントに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リストを使用します。このリストをシェルから保護するために、このリストの周囲に引用符を付ける必要がある場合があります。	**virt-sysprep --mount-options "/:notime"** は、**notime** 操作で root ディレクトリーをマウントします。
-q or --quiet	ログメッセージを出力しないようにします。	**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 または --version	virt-sysprep のバージョン番号を表示し、終了します。	**virt-sysprep -V**

コマンド	説明	例
--root-password	root パスワードを設定します。新しいパスワードを明示的に指定する場合や、選択したファイルの最初の行の文字列を使用する場合に使用できます。これにより、安全性が高まります。	**virt-sysprep --root-password password:***123456* **-a** *guest.img* または **virt-sysprep --root-password file:***SOURCE_FILE_PATH* **-a** *guest.img*

詳細は、libguestfs のドキュメント を参照してください。

21.12. VIRT-CUSTOMIZE: 仮想マシン設定のカスタマイズ

virt-customize コマンドラインツールを使用すると、仮想マシンをカスタマイズできます。たとえば、パッケージのインストールや設定ファイルの編集を行います。

virt-customize を使用するには、ゲスト仮想マシンがオフラインである必要があるため、シャットダウンしてからコマンドを実行してください。**virt-customize**は、ゲストまたはディスクイメージをコピーせずに、その場で変更することに注意してください。ゲスト仮想マシンの既存のコンテンツを保持する場合は、最初にディスクのスナップショット、コピー、またはクローンを作成する必要があります。ディスクのコピーおよびクローン作成の詳細は、libguestfs.org を参照してください。

警告

稼働中の仮想マシンで **virt-customize** を使用したり、他のディスク編集ツールを併用すると、ディスクが破損する可能性があります。このコマンドを使用する前に、仮想マシンをシャットダウンする**必要**があります。また、ディスクイメージは同時に編集しないでください。

virt-customize は、root では実行しないことが推奨されます。

virt-customize をインストールするには、次のいずれかのコマンドを実行します。

```
# yum install /usr/bin/virt-customize
```

または

```
# yum install libguestfs-tools-c
```

virt-customize では、以下のコマンドオプションを使用できます。

表21.2 virt-customizeオプション

コマンド	説明	例

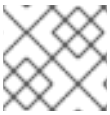
コマンド	説明	例
--help	特定のコマンドまたは **virt-customize** ユーティリティーに関する簡単なヘルプエントリーを表示します。詳細なヘルプは、man ページの virt-customize を参照してください。	**virt-customize --help**
-a [*file*] or --add [*file*]	指定された*file*を追加します。これは、ゲスト仮想マシンからのディスクイメージである必要があります。ディスクイメージの形式は自動検出されます。これを上書きして、特定の形式に強制する場合は、**--format** オプションを使用します。	**virt-customize --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-customize -a rbd://example.com[:port]/pool/disk**
-c [*URI*] or --connect [*URI*]	libvirt を使用する場合は、指定した URI に接続します。省略した場合は、KVM ハイパーバイザーを介して接続します。ゲストブロックデバイスを直接指定 (**virt-customize -a**) すると、libvirt はまったく使用されません。	**virt-customize -c qemu:///system**
-d [*guest*] or --domain [*guest*]	指定したゲスト仮想マシンのすべてのディスクを追加します。ドメイン名の代わりにドメイン UUID を使用できます。	**virt-customize --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***
-n or --dry-run	ゲスト仮想マシンで、読み取り専用の "dry run" カスタマイズ操作を実行します。これによりカスタマイズ操作が実行されますが、末尾のディスクに対する変更はすべて破棄されます。	**virt-customize -n**

コマンド	説明	例
--format [**raw**|**qcow2**|**auto**]	-a オプションのデフォルトは、ディスクイメージの形式を自動検出することです。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されます。**--format auto** スイッチを使用すると、後続の -a オプションの自動検出に戻ります (上述の **-a** コマンドを参照してください)。	**virt-customize --format raw -a *disk.img*** は、disk.img に対して raw 形式 (自動検出なし) を強制しますが、**virt-customize --format raw -a *disk.img* --format auto -a *another.img*** *は、**disk.img** に対して raw 形式 (自動検出なし) を強制し、**another.img** では自動検出に戻ります。信頼できない raw 形式のゲストディスクイメージがある場合は、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、悪意のあるゲストで発生する可能性のあるセキュリティーの問題を回避できます。*
-m [*MB*] または --memsize [*MB*]	**--run** スクリプトに割り当てるメモリー量を変更します。**--run** スクリプトや **--install** オプションでメモリー不足の問題が発生する場合は、メモリーの割り当てを増やします。	**virt-customize --memsize 1024**
--network または --no-network	インストール時にゲストからのネットワークアクセスを有効または無効にします。デフォルトでは有効になっています。**--no-network** を使用して、アクセスを無効にします。このコマンドは、システムの起動後のネットワークへのゲストアクセスには影響しません。詳細は、libguestfs documentation を参照してください。	**virt-customize -a http://[user@]example.com[:port]/disk.img**
-q or --quiet	ログメッセージを出力しないようにします。	**virt-customize -q**
-smp [*N*]	**--install** スクリプトで使用できる*N* 仮想 CPU を有効にします。*N* は 2 以上である必要があります。	**virt-customize -smp 4**
-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-customize --verbose**
-V または --version	**virt-customize**のバージョン番号を表示し、終了します。	**virt-customize --V**

コマンド	説明	例
-x	libguestfs API 呼び出しの追跡を有効にします。	**virt-customize -x**

virt-customize コマンドは、カスタマイズオプションを使用して、ゲストのカスタマイズ方法を設定します。ここでは、**--selinux-relabel** のカスタマイズオプションを説明します。

--selinux-relabel のカスタマイズオプションでは、正しい SELinux ラベルが付くように、ゲストのファイルに再ラベル付けされます。このオプションは、ファイルの再ラベル付けを即座に試行します。失敗すると、イメージで **/.autorelabel** がアクティブになります。これにより、次回イメージを起動したときに再ラベル付け操作が行われるようにスケジュールされます。



注記

このオプションは、SELinux に対応するゲストにのみ使用してください。

以下の例では、GIMP パッケージおよび Inkscape パッケージをゲストにインストールし、ゲストの次回起動時に SELinux ラベルが正しいことを確認します。

例21.1 virt-customize を使用したゲストへのパッケージのインストール

```
virt-customize -a disk.img --install gimp,inkscape --selinux-relabel
```

カスタマイズオプションなどの詳細は、libguestfs.org を参照してください。

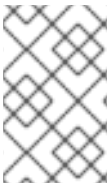
21.13. VIRT-DIFF: 仮想マシンファイル間の相違点の一覧表示

virt-diff コマンドラインツールを使用すると、2 つの仮想マシンのディスクイメージにあるファイル間の相違点をリスト表示できます。この出力は、仮想マシンのディスクイメージの実行後の変更を示しています。このコマンドを使用して、オーバーレイ間の違いを表示することもできます。

注記

virt-diff は読み取り専用アクセスのみが必要であるため、ライブゲスト仮想マシンでは安全に使用できます。

このツールは、実行中の仮想マシンと選択したイメージで、ファイル名、ファイルサイズ、チェックサム、拡張属性、ファイルコンテンツなどの相違を検出します。



注記

virt-diff コマンドでは、ブートローダー、パーティション間、ファイルシステム内の未使用領域、または非表示セクターの確認は行いません。したがって、これをセキュリティーまたはフォレンジックツールとして使用しないことが推奨されます。

virt-diff をインストールするには、次のいずれかのコマンドを実行します。

```
# yum install /usr/bin/virt-diff
```

または

```
# yum install libguestfs-tools-c
```

2 つのゲストを指定するには、最初のゲストに **-a** オプションまたは **-d** オプションを使用し、2 番目のゲストに **-A** オプションまたは **-D** オプションを使用する必要があります。以下に例を示します。

```
$ virt-diff -a old.img -A new.img
```

libvirt が認識している名前を使用することもできます。以下に例を示します。

```
$ virt-diff -d oldguest -D newguest
```

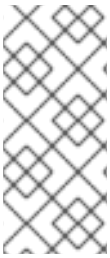
virt-diff では、以下のコマンドオプションを使用できます。

表21.3 virt-diffオプション

コマンド	説明	例
--help	特定のコマンドまたは **virt-diff** ユーティリティーに関する簡単なヘルプエントリーを表示します。詳細なヘルプは、man ページの virt-diff を参照してください。	**virt-diff --help**
-a [*file*] or --add [*file*]	指定した *ファイル* を追加します。これは、最初の仮想マシンのディスクイメージになります。仮想マシンに複数のブロックデバイスがある場合は、すべてのブロックデバイスに個別の **-a** オプションを指定する必要があります。 ディスクイメージの形式は自動検出されます。これを上書きして、特定の形式に強制する場合は、**--format** オプションを使用します。	**virt-customize --add /dev/vms/original.img -A /dev/vms/new.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-diff -a rbd://example.com[:port]/pool/newdisk -A rbd://example.com[:port]/pool/olddisk**
--all	**--extra-stats --times --uids --xattrs** と同じです。	**virt-diff --all**

コマンド	説明	例
--atime	初期設定では、**virt-diff** はファイルアクセス時間の変更を無視します。これは、ファイルアクセス時間の変更が興味深いものではないためです。**--atime** オプションを使用して、アクセス時間の違いを表示します。	**virt-diff --atime**
-A [*file*]	指定した *ファイル* または*URI* を追加します。これは、2 番目の仮想マシンのディスクイメージになります。	**virt-diff --add /dev/vms/original.img -A /dev/vms/new.img**
-c [*URI*] or --connect [*URI*]	libvirt を使用する場合は、指定した URI に接続します。省略した場合は、デフォルトの libvirt ハイパーバイザーに接続します。ゲストブロックデバイスを直接指定 (**virt-diff -a**) すると、libvirt はまったく使用されません。	**virt-diff -c qemu:///system**
--csv	結果をコンマ区切り値 (CSV) 形式で提供します。この形式は、データベースやスプレッドシートに簡単にインポートできます。詳細は、注記 参照してください。	virt-diff --csv
-d [*guest*] or --domain [*guest*]	指定したゲスト仮想マシンからすべてのディスクを最初のゲスト仮想マシンとして追加します。ドメイン名の代わりにドメイン UUID を使用できます。	$ **virt-diff --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***
-D [*guest*]	指定したゲスト仮想マシンのすべてのディスクを 2 番目のゲスト仮想マシンとして追加します。ドメイン名の代わりにドメイン UUID を使用できます。	**virt-diff --D *90df2f3f-8857-5ba9-2714-7d95907b1cd4***
--extra-stats	追加の統計情報を表示します。	**virt-diff --extra-stats**

コマンド	説明	例
--format または --format=[**raw**|**qcow2**]	**-a**/**-A** オプションのデフォルトは、ディスクイメージの形式を自動検出することです。これを使用すると、コマンドラインで **-a**/**-A** オプションのディスク形式が強制されます。**--format auto** スイッチを使用すると、後続の -a オプションの自動検出に戻ります (上述の **-a** コマンドを参照してください)。	**virt-diff --format raw -a *new.img* -A *old.img*** は、new.img および old.img に対して raw 形式 (自動検出なし) を強制しますが、**virt-diff --format raw -a *new.img* --format auto -a *old.img*** は、**new.img** に対して raw 形式 (自動検出なし) を強制し、**old.img** では自動検出に戻ります。信頼できない raw 形式のゲストディスクイメージがある場合は、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、悪意のあるゲストで発生する可能性のあるセキュリティーの問題を回避できます。
-h または --human-readable	ファイルのサイズを人間が判読できる形式で表示します。	**virt-diff -h**
--time-days	変更されたファイルの時間フィールドを、現在の日数 (将来の場合は負) で表示します。 出力内の**0**は、現在より 86,399 秒 (23 時間、59 分、59 秒) 前から、今後の 86,399 秒後までを意味することに注意してください。	**virt-diff --time-days**
-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-diff --verbose**
-V または --version	**virt-diff**のバージョン番号を表示し、終了します。	**virt-diff -V**
-x	libguestfs API 呼び出しの追跡を有効にします。	**virt-diff -x**



注記

コンマ区切り値 (CSV) 形式では解析が困難な場合があります。そのため、シェルスクリプトには csvtool を使用し、その他の言語には CSV 処理ライブラリー (Perl の場合は Text::CSV、Python の組み込み csv ライブラリーなど) を使用することが推奨されます。また、ほとんどのスプレッドシートおよびデータベースでは、CSV を直接インポートできます。

詳細および追加オプションは、libguestfs.org を参照してください。

21.14. VIRT-SPARSIFY: 空きディスク容量の再利用

virt-sparsify コマンドラインツールを使用すると、仮想マシンのディスク (またはディスクイメージ) をスパースにできます。これは、シンプロビジョニングとも呼ばれます。ディスクイメージの空きディスク領域は、ホストの空き領域に変換されます。

virt-sparsify コマンドは、ext2、ext3、ext4、btrfs、NTFS などのほとんどのファイルシステムで機能します。また、LVM 物理ボリュームでも動作します。**virt-sparsify** は、仮想マシンのディスクイメージだけでなく、任意のディスクイメージで動作できます。

警告

稼働中の仮想マシンで **virt-sparsify** を使用したり、他のディスク編集ツールを併用すると、ディスクが破損する可能性があります。このコマンドを使用する前に、仮想マシンをシャットダウンする**必要**があります。また、ディスクイメージは同時に編集しないでください。

このコマンドは、一部のディスク形式の間での変換にも使用できます。たとえば、**virt-sparsify** は、未加工のディスクイメージを、シンプロビジョニングの qcow2 イメージに変換できます。

注記

仮想マシンに複数のディスクがあり、ボリュームマネージメントを使用する場合は、**virt-sparsify** が機能しますが、それほど効果はありません。

入力が raw の場合、デフォルトの出力は raw sparse になります。出力イメージのサイズは、スパース性を理解するツールを使用して確認する必要があります。

```
$ ls -lh test1.img
-rw-rw-r--. 1 rjones rjones 100M Aug  8 08:08 test1.img
$ du -sh test1.img
3.6M   test1.img
```

なお、**ls** コマンドでは、イメージのサイズは 100M であることを示しています。ただし、**du** コマンドは、イメージサイズが 3.6M であることを正しく示しています。

重要な制限

以下は、重要な制限のリストです。

- **virt-sparsify** を使用する前に、仮想マシンを **シャットダウンする必要があります**。
- 最悪の場合は、**virt-sparsify** はソースディスクイメージの最大 2 倍の仮想サイズを必要とする場合があります。1 つは一時コピー用で、もう 1 つは宛先イメージ用です。

  --in-place オプションを使用すると、大量の一時領域が必要なくなります。
- **virt-sparsify** を使用して、ディスクイメージのサイズを変更することはできません。ディスクイメージのサイズを変更するには、**virt-resize** を使用します。**virt-resize** の詳細は、「virt-resize: ゲスト仮想マシンのオフラインでのサイズ変更」 を参照してください。
- 暗号化されたディスクはスパース化できないため、**virt-sparsify** は暗号化されたディスクでは機能しません。

- **virt-sparsify** では、パーティション間の領域をスパースにすることはできません。この領域はブートローダーなどの重要なアイテムによく使用されるため、実際には未使用のスペースではありません。
- **copy** モードでは、qcow2 の内部スナップショットは、宛先イメージにコピーされません。

例
virt-sparsify をインストールするには、次のいずれかのコマンドを実行します。

```
# yum install /usr/bin/virt-sparsify
```

または

```
# yum install libguestfs-tools-c
```

ディスクのスパース化

```
# virt-sparsify /dev/sda1 /dev/device
```

/dev/sda1 のコンテンツを**/dev/device** にコピーして、出力をスパースにします。**/dev/device** がすでに存在する場合は上書きされます。**/dev/sda1** の形式が検出され、**/dev/device** の形式として使用されます。

形式を変換するには、次のコマンドを実行します。

```
# virt-sparsify disk.raw --convert qcow2 disk.qcow2
```

ソースディスクイメージ内にあるすべてのファイルシステムの空き領域をゼロに設定し、スパース化します。

特定のファイルシステムで、空き領域がゼロで上書きされないようにするには、次のコマンドを実行します。

```
# virt-sparsify --ignore /dev/device /dev/sda1 /dev/device
```

ファイルシステムの空き領域をゼロで上書きすることなく、ディスクイメージ内のすべてのファイルシステムからスパース化されたディスクイメージを作成します。

一時コピーを作成せずにディスクイメージをスパースにするには、次のコマンドを実行します。

```
# virt-sparsify --in-place disk.img
```

指定したディスクイメージをスパースにし、イメージファイルを上書きします。

virt-sparsifyオプション
virt-sparsify では、以下のコマンドオプションを使用できます。

表21.4 virt-sparsifyオプション

コマンド	説明	例

コマンド	説明	例
--help	特定のコマンドまたは **virt-sparsify** ユーティリティーに関する簡単なヘルプエントリーを表示します。詳細なヘルプは、man ページの virt-sparsify を参照してください。	**virt-sparsify --help**
--check-tmpdir **ignore**|**continue**|**warn**|**fail**	*tmpdir* に、操作を完了するのに十分な領域があるかどうかを推定します。操作を完了するのに十分な領域がない場合の動作を指定するには、このオプションを使用します。 ● **ignore**: この問題を無視し、操作を続行します。 ● **continue**: エラーを報告し、操作を続行します。 ● **warn**: エラーを報告し、ユーザーが Enter キーを押すのを待ちます。 ● **fail**: エラーを報告し、操作を中止します。 このオプションは、**--in-place** オプションとは併用できません。	**virt-sparsify --check-tmpdir ignore /dev/sda1 /dev/device** **virt-sparsify --check-tmpdir continue /dev/sda1 /dev/device** **virt-sparsify --check-tmpdir warn /dev/sda1 /dev/device** **virt-sparsify --check-tmpdir fail /dev/sda1 /dev/device**
--compress	出力ファイルを圧縮します。これは、出力形式が qcow2 の場合に ***のみ*** 機能します。このオプションは、**--in-place** オプションとは併用できません。	**virt-sparsify --compress /dev/sda1 /dev/device**
--convert	指定した形式を使用してスパースイメージを作成します。形式を指定しない場合は、入力形式が使用されます。 raw、qcow、vd の出力形式がサポートされ、動作が確認されています。 QEMU エミュレーターで対応している形式はどれでも使用できます。 **--convert** オプションを使用することが推奨されます。この方法では、**virt-sparsify** が入力形式を推測する必要がありません。 このオプションは、**--in-place** オプションとは併用できません。	**virt-sparsify --convert raw /dev/sda1 /dev/device** **virt-sparsify --convert qcow2 /dev/sda1 /dev/device** **virt-sparsify --convert *other_format indisk outdisk***

コマンド	説明	例
--format	入力ディスクイメージの形式を指定します。指定しない場合は、イメージから形式が検出されます。信頼できない raw 形式のゲストディスクイメージを使用する場合は、形式を指定してください。	**virt-sparsify --format raw /dev/sda1 /dev/device** **virt-sparsify --format qcow2 /dev/sda1 /dev/device**
--ignore	指定したファイルシステムまたはボリュームグループを無視します。 ファイルシステムを指定し、**--in-place** オプションを指定しない場合は、ファイルシステムの空き領域はゼロになりません。ただし、ゼロの既存ブロックはスパース化されています。**--in-place** オプションを指定しても、ファイルシステムは完全に無視されます。 ボリュームグループを指定しても、ボリュームグループは無視されます。ボリュームグループ名には、**/dev/** 接頭辞を使用しないでください。たとえば、**--ignore vg_foo** です。 **--ignore** オプションは、複数回指定できます。	**virt-sparsify --ignore filesystem1 /dev/sda1 /dev/device** **virt-sparsify --ignore volume_group/dev/sda1 /dev/device**

コマンド	説明	例
--in-place	一時的なコピーを作成する代わりに、イメージをインプレースでスパースにします。インプレーススパース化は、スパース化をコピーするよりも効率的ですが、スパース化をコピーするのと同じディスク領域は復旧できません。インプレーススパース化は、破棄 (トリムまたはアンマップとしても知られている) サポートを使用して機能します。 インプレーススパース化を使用するには、インプレースでスパース化されるディスクイメージを指定します。 インプレーススパース化を指定する場合は、以下のオプションを使用できません。 ● **--convert** および **--compress** は、ディスクフォーマットの大規模な変更が必要なため、使用できません。 **--check-tmpdir** は、大量の一時領域が必要ないため、使用できません。	**virt-sparsify --in-place disk.img**
-x	libguestfs API 呼び出しの追跡を有効にします。	**virt-sparsify -x filesystem1 /dev/sda1 /dev/device**

詳細および追加オプションは、libguestfs.org を参照してください。

第22章 ゲスト仮想マシン管理用のグラフィカルユーザーインターフェイスツール

virt-manager のほかに、Red Hat Enterprise Linux 7 には、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以下のツールが備わっています。

22.1. VIRT-VIEWER

virt-viewer は、ゲスト仮想マシンのグラフィカルコンソールを表示するための最小限のコマンドラインユーティリティーです。コンソールには、VNC または SPICE プロトコルを使用してアクセスします。ゲストは、名前、ID、または UUID で参照できます。ゲストが実行していない場合は、ビューアーが起動するまで待機してからコンソールに接続するように設定できます。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、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も接続できます。

virt-manager と比較すると、**virt-viewer** では機能セットが小さくなりますが、リソースへの要求は低くなります。また、**virt-manager** とは異なり、ほとんどの場合、**virt-viewer** は、libvirt への読み取りと書き込みのパーミッションを必要としません。したがって、ゲストに接続して表示できる非特権ユーザーよって使用されることが可能ですが、設定することはできません。

virt-viewer をインストールするには、以下を実行します

```
# yum install virt-viewer
```

Syntax

virt-viewer コマンドラインの基本的な構文は次のようになります。

```
# virt-viewer [OPTIONS] {guest-name|id|uuid}
```

virt-viewer で使用できるオプションのリストは、man ページの **virt-viewer** を参照してください。

ゲスト仮想マシンへの接続

オプションを付けずに使用すると、**virt-viewer** は、ローカルシステムのデフォルトハイパーバイザーで接続できるゲストのリストを表示します。

デフォルトのハイパーバイザーを使用する、指定したゲスト仮想マシンに接続するには、次のコマンドを実行します。

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

KVM-QEMU ハイパーバイザーを使用するゲスト仮想マシンに接続するには、次のコマンドを実行します。

```
# virt-viewer --connect qemu:///system guest-name
```

TLS を使用してリモートコンソールに接続するには、次のコマンドを実行します。

```
# virt-viewer --connect qemu://example.org/ guest-name
```

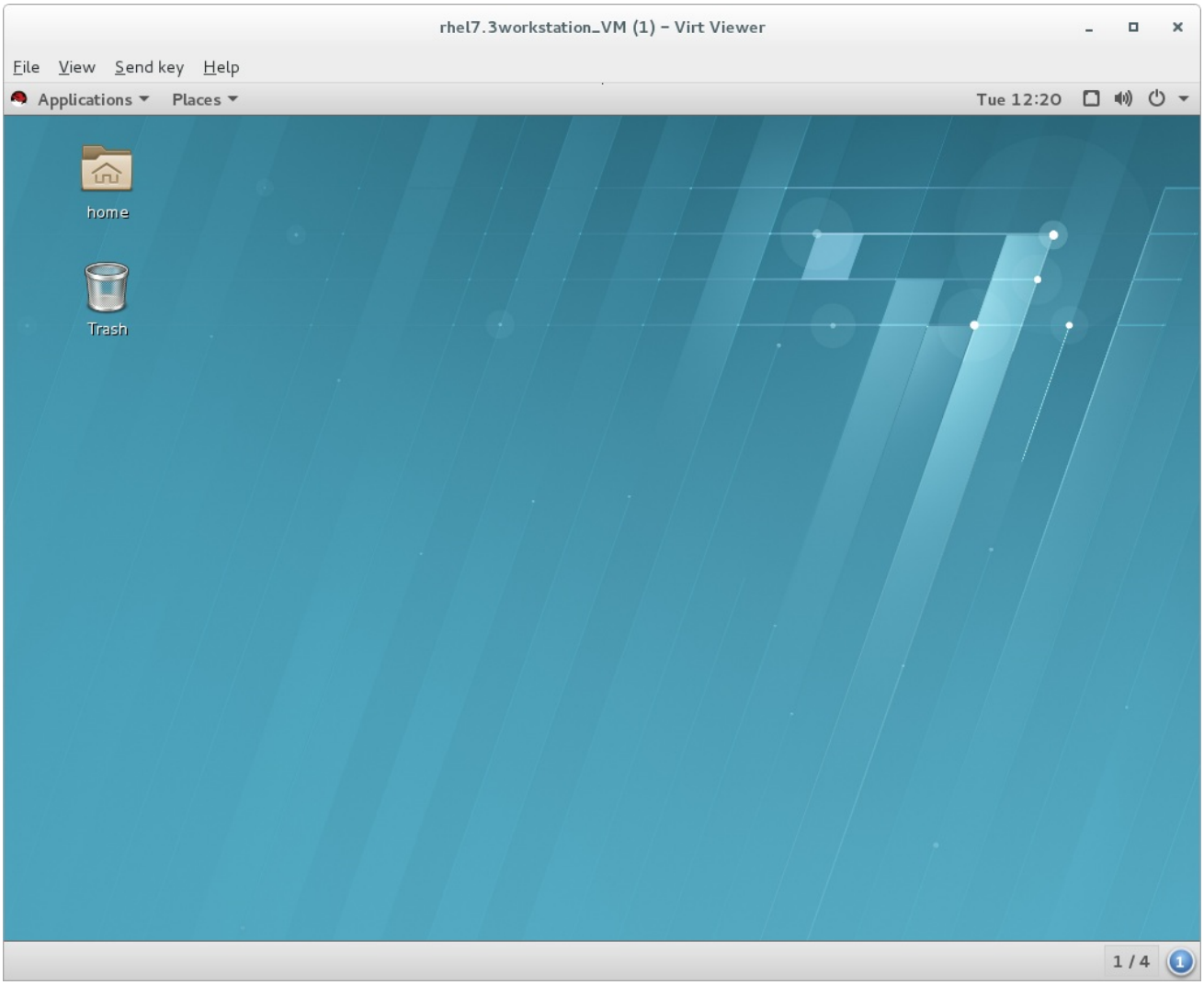
SSH を使用してリモートホストのコンソールに接続するには、ゲスト設定を調べてから、コンソールにトンネリングされていない直接接続を確立します。

```
# virt-viewer --direct --connect qemu+ssh://root@example.org/ guest-name
```

インターフェイス

デフォルトでは、**virt-viewer** インターフェイスは、ゲストと対話するための基本的なツールのみを提供します。

図22.1 サンプルの virt-viewer インターフェイス



ホットキーの設定

virt-viewerセッション用にカスタマイズしたキーボードショートカット (ホットキーとも呼ばれます) を作成する場合は、**--hotkeys** オプションを使用します。

```
# virt-viewer --hotkeys=action1=key-combination1[,action2=key-combination2] guest-name
```

ホットキーに割り当てることができるアクションは、次のとおりです。

- toggle-fullscreen
- release-cursor
- smartcard-insert
- smartcard-remove

キーと名前の組み合わせのホットキーでは、大文字と小文字が区別されません。ホットキーの設定は、今後の **virt-viewer** セッションには引き継がれないことに注意してください。

例22.1 virt-viewer ホットキーの設定

KVM-QEMU ゲストに接続して testguest と呼ばれる 全画面モードに変更するホットキーを追加するには、次のコマンドを実行します。

```
# virt-viewer --hotkeys=toggle-fullscreen=shift+f11 qemu:///system testguest
```

キオスクモード

キオスクモードでは、**virt-viewer** により、ユーザーは接続したデスクトップとのみ対話でき、ゲストがシャットダウンしない限り、ゲスト設定またはホストシステムと対話するオプションは提供されません。これは、管理者がユーザーのアクションの範囲を指定したゲストに制限する場合などに役立ちます。

キオスクモードを使用するには、**-k** または **--kiosk** オプションでゲストに接続します。

例22.2 キオスクモードでの virt-viewer の使用

マシンのシャットダウン後に終了する KVM-QEMU 仮想マシンにキオスクモードで接続する場合は、次のコマンドを使用します。

```
# virt-viewer --connect qemu:///system guest-name --kiosk --kiosk-quit on-disconnect
```

ただし、キオスクモードのみでは、ゲストのシャットダウン後に、ユーザーがホストシステムやゲスト設定と対話しないようにすることはできません。これには、ホストのウィンドウマネージャーを無効にするなど、さらなるセキュリティー対策が必要になります。

22.2. REMOTE-VIEWER

remote-viewer は、SPICE および VNC に対応するシンプルなリモートデスクトップディスプレイクライアントです。これは、機能のほとんどと制限を virt-viewer と共有します。

ただし、**virt-viewer** とは異なり、**remote-viewer** では、リモートゲストディスプレイに接続するのに libvirt が必要ありません。このため、**remote-viewer** は、たとえば、libvirt と対話したり SSH 接続を使用したりするためのパーミッションを提供しないリモートホスト上の仮想マシンに接続するために使用できます。

remote-viewer ユーティリティーをインストールするには、以下を実行します。

```
# yum install virt-viewer
```

Syntax

remote-viewer コマンドラインの基本的な構文は次のようになります。

```
# remote-viewer [OPTIONS] {guest-name|id|uuid}
```

remote-viewer で使用できるオプションのリストは、man ページの **remote-viewer** を参照してください。

ゲスト仮想マシンへの接続

オプションを付けずに使用すると、**remote-viewer** は、ローカルシステムのデフォルトの URI で接続できるゲストのリストを表示します。

remote-viewer を使用して特定のゲストに接続するには、VNC/SPICE URI を使用します。URI の取得方法は、「グラフィック表示への接続の URI の表示」 を参照してください。

例22.3 SPICE を使用したゲストディスプレイへの接続

以下を使用して、SPICE 通信にポート 5900 を使用する "testguest" と呼ばれるマシンの SPICE サーバーに接続します。

```
# remote-viewer spice://testguest:5900
```

例22.4 VNC を使用したゲストディスプレイへの接続

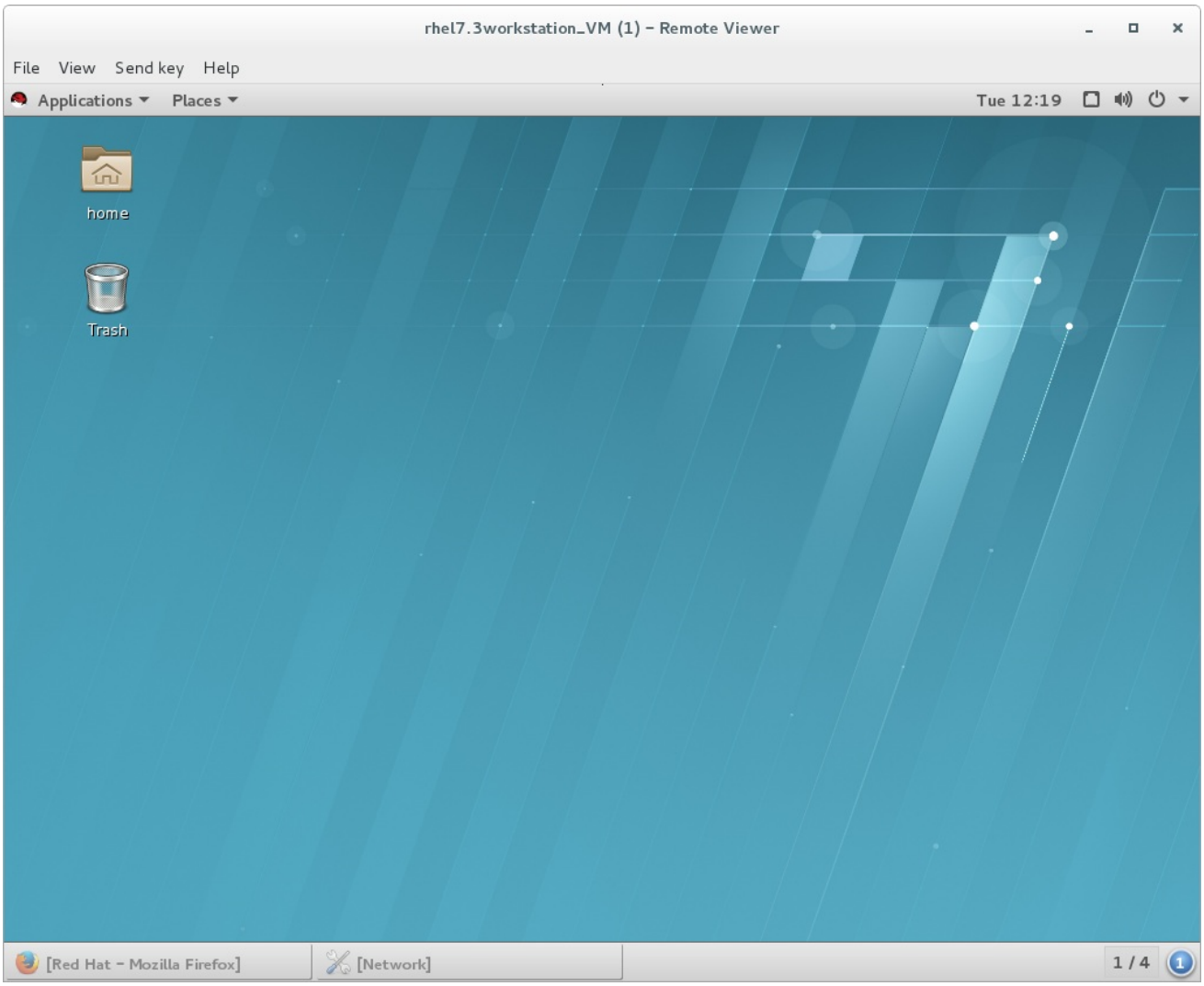
以下を使用して、VNC 通信にポート 5900 を使用する **testguest2** と呼ばれるマシンの VNC サーバーに接続します。

```
# remote-viewer vnc://testguest2:5900
```

インターフェイス

デフォルトでは、**remote-viewer** インターフェイスは、ゲストと対話するための基本的なツールのみを提供します。

図22.2 サンプルのremote-viewer インターフェイス



22.3. GNOME BOXES

Boxes は、仮想マシンおよびリモートシステムの表示およびアクセスに使用される軽量なグラフィカルデスクトップ仮想化ツールです。

virt-viewer および remote-viewer とは異なり、Boxes では virt-manager と同様に、ゲスト仮想マシンの表示だけでなく、ゲスト仮想マシンの作成および設定も行えます。ただし、virt-manager と比較すると、Boxes では、管理オプションおよび機能が少なくなりますが、使用しやすくなります。

Boxes をインストールするには、以下を実行します

```
# yum install gnome-boxes
```

Applications ⇒ System Tools で 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、以下の 2 つのボタンがあります。

-  - 検索ボタン (名前でゲスト仮想マシンを検索) および

-  - 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でき、操作を個別に、またはグループとして実行できます。利用可能な操作は、操作バーの画面下部に表示されます。

図22.3 操作バー

実行可能な操作は、以下の 4 つです。

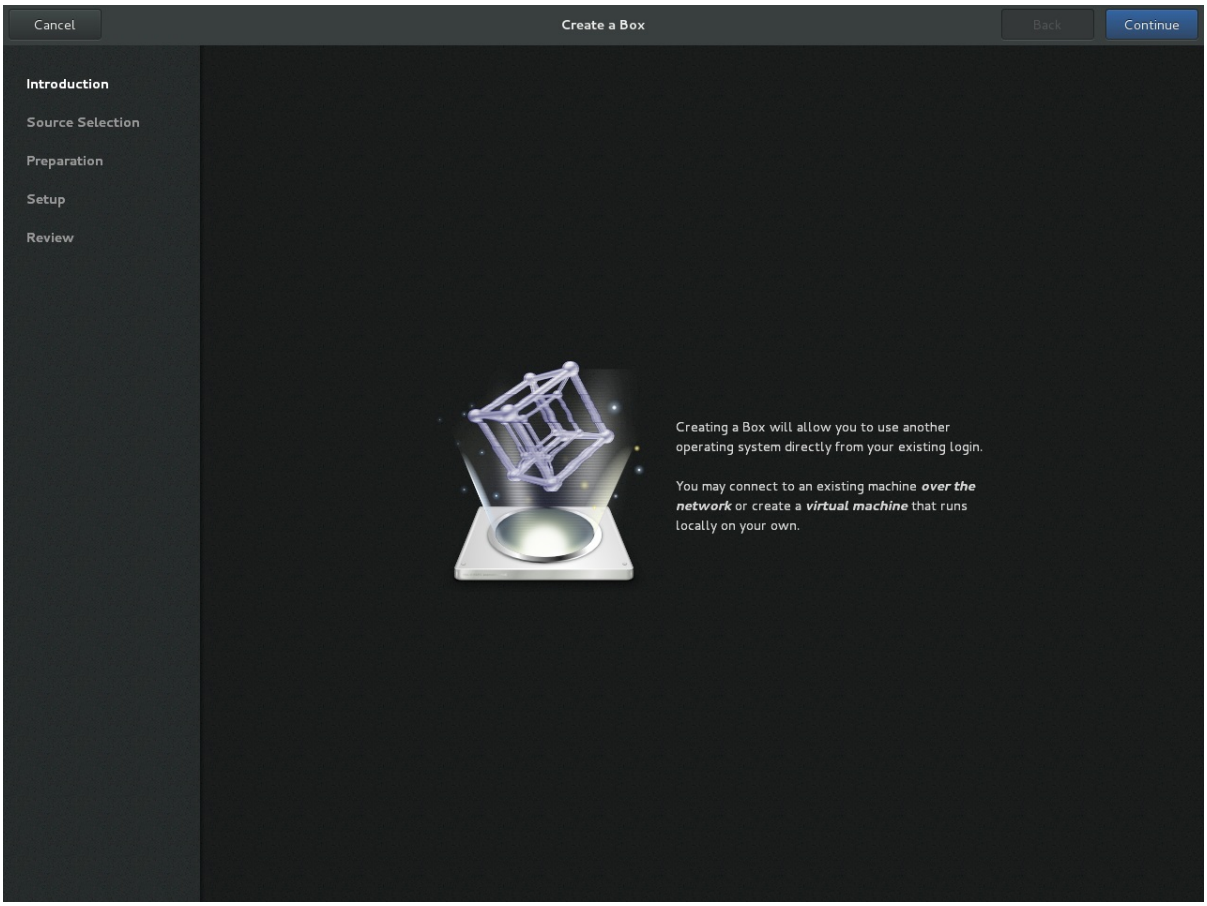
- **Favorite**: 選択したゲスト仮想マシンにハートを追加し、ゲストリストの一番上に移動します。これは、ゲストの数が増えるにつれ、非常に役に立ちます。
- **Pause**: 選択したゲスト仮想マシンが実行を停止します。
- **Delete**: 選択したゲスト仮想マシンを削除します。
- **Properties**: 選択したゲスト仮想マシンのプロパティーを表示します。

メイン画面の左側にある **New** ボタンを使用して、新しいゲスト仮想マシンを作成します。

手順22.1 Boxes を使用した新しい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22.4 Introduction 画面



2. **ソースの選択**
Source Selection 画面には、3 つのオプションがあります。

   - 利用可能なメディア: 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。いずれかをクリックすると、Review 画面が表示されます。

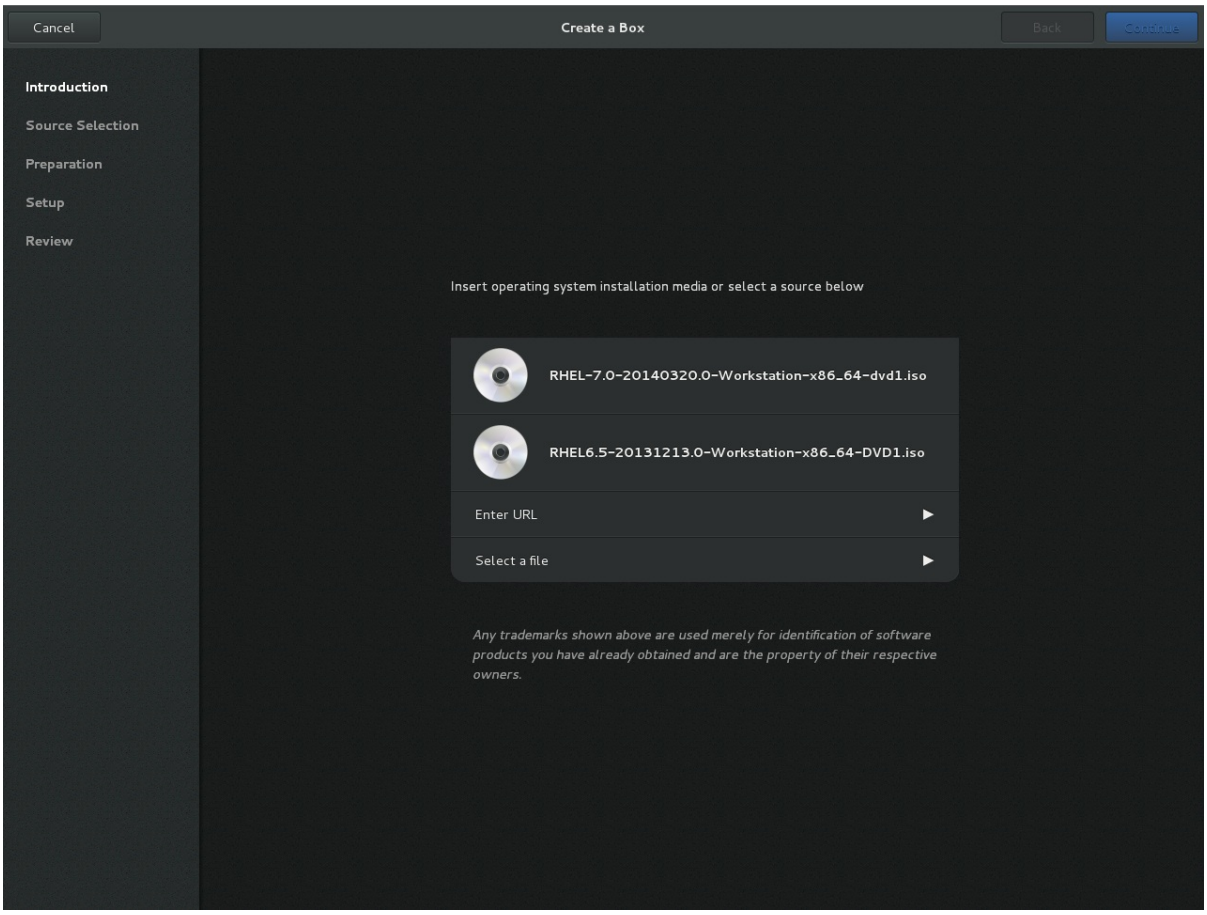
   - **Enter a URL**: URL を入力して、ISO ファイルへのローカル URI または パスを指定します。これは、リモートマシンへのアクセスにも使用できます。このアドレスは、***protocol*://*IPaddress*?*port*;** のパターンに従います。以下に例を示します。

     ```
     spice://192.168.122.1?port=5906;
     ```

     プロトコルは、**spice://**、**qemu://**、または **vnc://** になります。

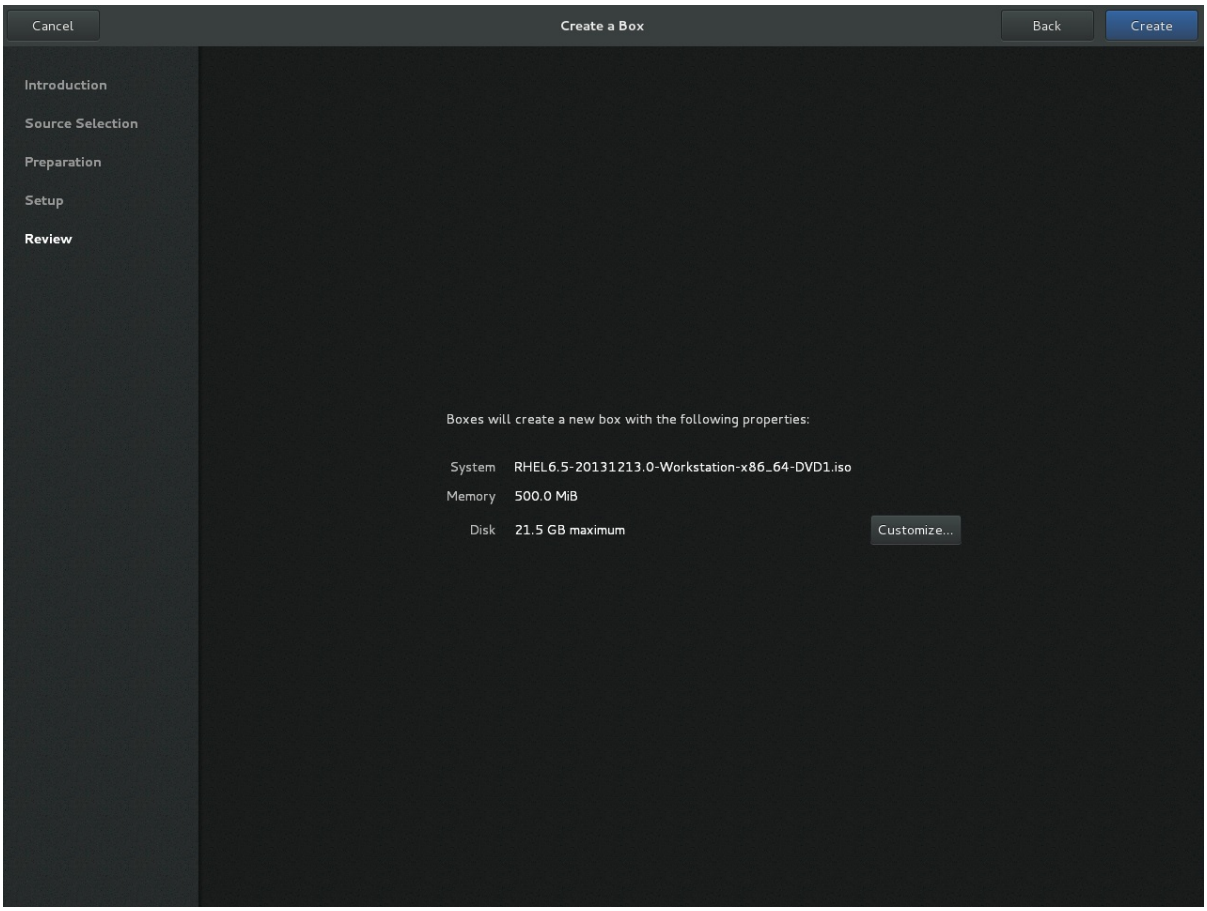
   - **Select a file**: ファイルディレクトリーを開いて、インストールメディアを手動で検索します。

図22.5 Source Selection 画面



3. **詳細の確認**
   Review 画面には、ゲスト仮想マシンの詳細が表示されます。

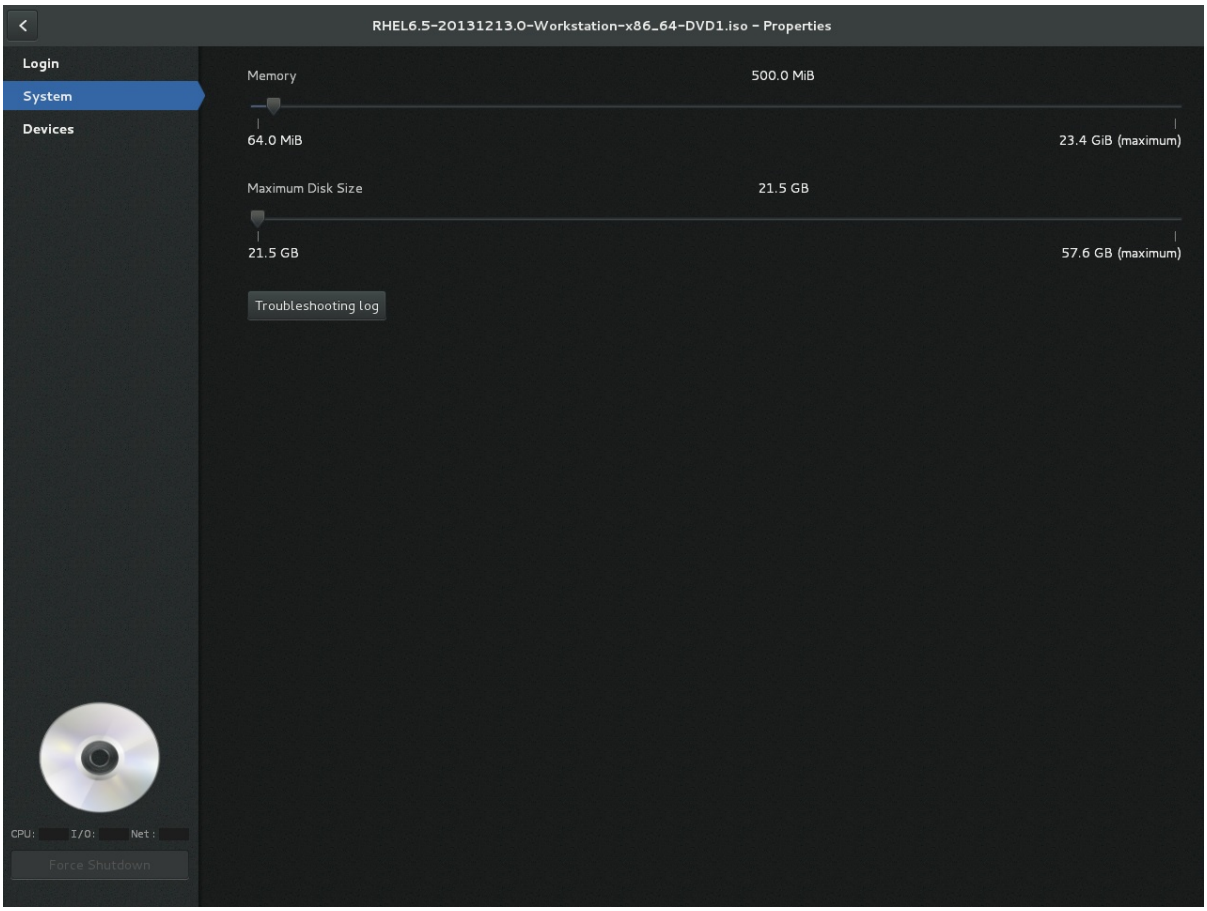
図22.6 Review 画面



これらの詳細はそのままにしておくことができます。その場合は、最終ステップに進むか、次のようにします。

4. **必要に応じて、詳細をカスタマイズします。**
Customize をクリックすると、メモリーやディスクサイズなど、ゲスト仮想マシンの設定を調整できます。

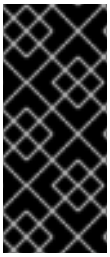
図22.7 Customization 画面



5. Create
   Create をクリックします。新しいゲスト仮想マシンが開きます。

第23章 ドメイン XML の操作

本章では、ゲスト仮想マシンの XML 設定ファイル (*ドメイン XML* とも呼ばれます) のコンポーネントの詳細を説明します。本章では、*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、ドメインの実行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固有ですが、**KVM** などが含まれます。**id** は、実行中のゲスト仮想マシンの一意の整数識別子です。アクティブでないマシンには、**id** 値が設定されていません。本章のセクションでは、ドメイン XML のコンポーネントを説明します。ドメイン XML の操作が必要な場合は、本書のその他の章を参照してください。



重要

対応している管理インターフェイス (**virsh**、Virtual Machine Manager など) およびコマンド (**virt-xml** など) のみを使用して、ドメイン XML ファイルのコンポーネントを編集します。テキストエディターでドメインの XML ファイルを直接開いて編集しないでください。ドメイン XML ファイルを直接編集する必要がある場合は、**virsh edit** コマンドを使用します。

注記

本章は libvirt アップストリームのドキュメント に基づいています。

23.1. 一般情報およびメタデータ

この情報は、ドメイン XML の以下の部分にあります。

図23.1 ドメイン XML メタデータ

```
<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>
```

ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.1 一般的なメタデータ要素

要素	説明
<name>	仮想マシンの名前を割り当てます。この名前は英数字のみで設定され、1 台のホストの物理マシンの範囲内で固有である必要があります。永続的な設定ファイルを保存するためにファイル名を形成するためによく使用されます。

要素	説明
<uuid>	仮想マシンのグローバルに一意識別子を割り当てます。この形式は、RFC 4122 準拠 (**3e3fce45-4f53-4fa7-bb32-11f34168b82b** など) である必要があります。新しいマシンの定義時または作成時に省略した場合は、ランダムな UUID が生成されます。**sysinfo** 仕様を使用して、UUID を提供することもできます。
<title>	ドメインの簡単な説明のための領域を作成します。タイトルには改行しないでください。
<description>	タイトルとは異なり、このデータは libvirt では使用されません。これには、ユーザーが表示を選択した情報を含めることができます。
<metadata>	アプリケーションで使用できるため、カスタムメタデータを XML ノード/ツリーの形式で保存できます。アプリケーションは、XML ノード/ツリーでカスタム名前空間を使用し、名前空間ごとにトップレベル要素を 1 つのみ使用する必要があります (アプリケーションが構造を必要とする場合は、その名前空間要素のサブ要素を持つ必要があります)。

23.2. オペレーティングシステムの起動

BIOS ブートローダー、ホスト物理マシンのブートローダー、ダイレクトカーネルブート、コンテナーブートなど、仮想マシンを起動する方法は多数あります。

23.2.1. BIOS ブートローダー

BIOS の起動は、完全仮想化をサポートするハイパーバイザーで利用できます。この場合、BIOS は、ブートイメージの場所を決定する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) を持ちます。ドメイン XML の **<os>** セクションには、以下の内容が含まれます。

図23.2 BIOS ブートローダードメイン XML

```
...
<os>
  <type>hvm</type>
  <boot dev='fd'/>
  <boot dev='hd'/>
  <boot dev='cdrom'/>
  <boot dev='network'/>
  <bootmenu enable='yes'/>
  <smbios mode='sysinfo'/>
  <bios useserial='yes' rebootTimeout='0'/>
</os>
...
```

ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.2 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するオペレーティングシステムの種類を指定します。**hvm** は、オペレーティングシステムがベアメタルで実行するように設計されており、完全な仮想化が必要であることを示します。**linux** は、KVM ハイパーバイザーゲスト ABI をサポートするオペレーティングシステムを指します。オプションの属性も 2 つあります。**arch** は仮想化に対する CPU アーキテクチャーを指定し、**machine** はマシンタイプを参照します。詳細は、『libvirt upstream documentation』 を参照してください。
<boot>	次回の起動時に考慮するデバイスを指定します。値は、**fd**、**hd**、**cdrom**、または **network** のいずれかになります。ブート要素を複数回繰り返して、順番に試行するブートデバイスの優先順位リストを設定できます。同じタイプの複数のデバイスは、バスの順序を維持しながら、ターゲットに従ってソートされます。ドメインを定義すると、libvirt が返す XML 設定により、デバイスがソートされた順にリスト表示されます。ソートされると、最初のデバイスが起動可能としてマークされます。詳細は、libvirt upstream documentation を参照してください。
<bootmenu>	ゲスト仮想マシンの起動時にインタラクティブな起動メニュープロンプトを有効にするかどうかを設定します。**enable** 属性は、**yes** または **no のいずれかになり** ます。指定しない場合は、ハイパーバイザーのデフォルトが使用されます。
<smbios>	ゲスト仮想マシンで SMBIOS 情報をどのように表示するかを指定します。**mode** 属性は、**emulate** (ハイパーバイザーがすべての値を生成できるようにする)、**host** (UUID を除くブロック 0 とブロック 1 のすべてをホスト物理マシンの SMBIOS 値からコピーします。**virConnectGetSysinfo** 呼び出しを使用して、どの値がコピーされているかを確認することができます)、または **sysinfo** (sysinfo 要素の値を使用) のいずれかとして指定する必要があります。指定しない場合は、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** の値を持つ属性 **useserial** があります。この属性は、シリアルグラフィックスアダプターを有効または無効にします。これにより、ユーザーはシリアルポートで BIOS メッセージを表示できるようになります。したがって、シリアルポートを定義する必要があります。**rebootTimeout** 属性は、(BIOS に応じて) システムの起動に失敗した場合にゲスト仮想マシンが再起動するかどうかと、その起動後どのくらいの時間がかかるかを制御します。この値はミリ秒単位で設定し、上限は **65535** です。**-1** を設定すると再起動が無効になります。

23.2.2. ダイレクトカーネルブート

新しいゲスト仮想マシンのオペレーティングシステムをインストールする場合は、多くの場合、ホストの物理マシンのオペレーティングシステムに保存されているカーネルと **initrd** から直接起動して、コマンドラインの引数をインストーラーに渡せるようにすると便利です。この機能は、通常、完全仮想化ゲスト仮想マシンおよび準仮想化ゲスト仮想マシンの両方で利用できます。

図23.3 ダイレクトカーネルブート

```
...
<os>
  <type>hvm</type>
  <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>
...
```

ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.3 ダイレクトカーネルブート要素

要素	説明
<type>	BIOS ブートのセクションで説明されているものと同じです。
<kernel>	ホスト物理マシンのオペレーティングシステムのカーネルイメージへの完全修飾パスを指定します。
<initrd>	ホスト物理マシンのオペレーティングシステムの (任意) ramdisk イメージへの完全修飾パスを指定します。

要素	説明
<cmdline>	システムの起動時にカーネル (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、代替のプライマリーコンソール (シリアルポートなど)、またはインストールメディアソースや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。

23.2.3. コンテナーブート

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は、カーネルまたはブートイメージの代わりに、**init** 要素を使用して**init** バイナリーへのパスを指定する必要があります。デフォルトでは、引数を指定せずに起動します。最初の**argv**を指定するには、**initarg** 要素を必要な回数だけ繰り返します。**cmdline** 要素は、**/proc/cmdline** と同等の機能を提供しますが、**<initarg>**には影響を及ぼしません。

図23.4 コンテナーブート

```
...
<os>
  <type arch='x86_64'>exe</type>
  <init>/bin/systemd</init>
  <initarg>--unit</initarg>
  <initarg>emergency.service</initarg>
</os>
...
```

23.3. SMBIOS システム情報

ハイパーバイザーの中には、ゲスト仮想マシンに表示されるシステム情報を制御できるものもあります (たとえば、SMBIOS フィールドはハイパーバイザーで入力し、ゲスト仮想マシンの **dmidecode** コマンドを使用して検証できます)。オプションの **sysinfo** 要素は、このようなカテゴリーの情報をすべてカバーします。

図23.5 SMBIOS システム情報

```
...
<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>
...
```

<sysinfo> 要素には、サブ要素のレイアウトを決定する必須の属性**type**があり、以下のように定義されます。

- **<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** サブ要素と併用するとゲスト仮想マシンに影響します。**<sysinfo>** の各サブ要素は、SMBIOS ブロックの名前を付け、その中の要素は、ブロック内のフィールドを記述するエントリー要素のリストになります。以下のブロックおよびエントリーが認識されます。

  - **<bios>** - SMBIOS の ブロック 0 で、エントリー名は **vendor**、**version**、**date**、および **release** から取得されます。

  - **<system>** - 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** から取得します。**uuid** エントリーが最上位の **uuid** 要素とともに指定されている場合は、その 2 つの値が一致している必要があります。

23.4. CPU の割り当て

図23.6 CPU の割り当て

```
<domain>
  ...
  <vcpu placement='static' cpuset="1-4,^3,6" current="1">2</vcpu>
  ...
</domain>
```

<vcpu> 要素は、ゲスト仮想マシンのオペレーティングシステムに割り当てる仮想 CPU の最大数を定義します。この最大数は 1 から、ハイパーバイザーで対応している最大数までの間でなければなりません。この要素には、**cpuset** 属性を指定できます。属性は、ドメインプロセスおよび仮想 CPU をデフォルトで固定できる物理 CPU 番号のコンマ区切りのリストです。

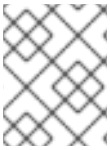
ドメインプロセスおよび仮想 CPU のピン留めポリシーは、**cputune** 属性を使用して個別に指定できることに注意してください。**emulatorpin** 属性が **<cputune>** で指定されている場合、**<vcpu>** で指定された **cpuset** は無視されます。

同様に、**vcpupin** に値を設定した仮想 CPU では、 **cpuset** 設定が無視されます。**vcpupin** が指定されていない仮想 CPU の場合は、**cpuset** で指定された物理 CPU に固定されます。 **cpuset** リストの各要素は、単一の CPU 番号、CPU 番号の範囲、または前の範囲から除外される CPU 番号が後に続くキャレット (^) のいずれかになります。属性 **current** を使用して、有効にする仮想 CPU の最大数より少ない数を指定することができます。

placement オプション属性を使用して、ドメインプロセスの CPU 配置モードを示すことができます。**placement** は、以下のいずれかに設定できます。

- **static** - vCPU を、 **cpuset** 属性で定義された物理 CPU にピン留めします。 **cpuset** が定義されていない場合は、利用可能なすべての物理 CPU にドメインプロセスが固定されます。

- **auto** - ドメインプロセスが、numad のクエリーからアドバイザリーノードセットにピン留めされ、指定されている場合は、属性 **cpuset** の値が無視されます。



注記

cpuset 属性が **placement** とともに使用される場合、**placement** の値は、デフォルトで **<numatune>** 要素 (使用されている場合) の値か、**static** になります。

23.5. CPU チューニング

図23.7 CPU 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>
```

すべてオプションですが、ドメイン XML のこのセクションのコンポーネントは次のとおりです。

表23.4 CPU チューニング要素

要素	説明
<cputune>	ドメインの CPU 調整可能パラメーターに関する詳細を提供します。これは任意です。
<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。省略した場合や、**<vcpu>** 要素の **cpuset** 属性が指定されていない場合は、vCPU がすべての物理 CPU にデフォルトで固定されます。必要な属性が 2 つ含まれています。**<vcpu>** 属性は **id** を指定し、**cpuset** 属性は **<vcpu>** 要素の **cpuset** 属性と同じです。
<emulatorpin>	エミュレーター (**<vcpu>** を含まないドメインのサブセット) を固定するホスト物理マシンの CPU を指定します。省略した場合に、**<vcpu>** 要素の **cpuset** 属性を指定しないと、すべての物理 CPU に、デフォルトでエミュレーターが固定されます。固定する物理 CPU を指定する 1 つの必須 **cpuset** 属性が含まれています。**<vcpu>** 要素の **placement** 属性が **auto** に設定されている場合、**emulatorpin** は許可されません。

要素	説明
<共有>	ドメインの比例的な加重共有を指定します。省略した場合は、オペレーティングシステムのデフォルト値が使用されます。値の単位がない場合は、その他のゲスト仮想マシンの設定を基準にして計算されます。たとえば、**<共有>** の値が 2048 で設定されたゲスト仮想マシンは、**<共有>** の値が 1024 で設定されたゲスト仮想マシンの 2 倍の CPU 時間を受け取ります。
<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、ドメインの各 vCPU は、割り当てられたクォータに相当する実行時間を超えて消費することはできなくなります。この値は、以下の範囲内である必要があります: **1000-1000000**。**0** の値を持つ**<period>**は、値がないことを意味します。
<quota>	最大許容帯域幅をマイクロ秒単位で指定します。**<quota>** が負の値のドメインは、ドメインの帯域幅が無限であることを示します。つまり、帯域幅は制御されません。値は次の範囲内である必要があります: **1000 - 18446744073709551** または **0** 未満。**0** の値を持つ**quota** は、値がないことを意味します。この機能を使用すると、すべての vCPU で同じ速度で実行できます。
<emulator_period>	施行間隔をマイクロ秒単位で指定します。**<emulator_period>** 内では、ドメインのエミュレータースレッド (vCPU を除く) が、**<emulator_quota>** 相当の実行時間より多くを消費することはできません。**<emulator_period>** は、次の範囲内でなければなりません: **1000 - 1000000**。**0** の**<emulator_period>**は、値がないことを意味します。
<emulator_quota>	ドメインのエミュレータースレッド (vCPU を除く) で許可される最大帯域幅をマイクロ秒単位で指定します。**<emulator_quota>** が負の値のドメインは、ドメインがエミュレータースレッド (vCPU を除く) に無限の帯域幅を持つことを示します。これは、帯域幅が制御されないことを示しています。値は次の範囲内である必要があります: **1000～18446744073709551**、または **0**未満。**0** の**<emulator_quota>**は、値がないことを意味します。

23.6. メモリーバッキング

メモリーバッキングにより、ハイパーバイザーはゲスト仮想マシン内の大きなページを適切に管理できます。

図23.8 メモリーバッキング

```
<domain>
  ...
  <memoryBacking>
    <hugepages>
   <page size="1" unit="G" nodeset="0-3,5"/>
   <page size="2" unit="M" nodeset="4"/>
  </hugepages>
  <nosharepages/>
  <locked/>
  </memoryBacking>
  ...
</domain>
```

memoryBacking 要素の詳細は、libvirt のアップストリームドキュメント を参照してください。

23.7. メモリーの調整

図23.9 メモリーの調整

```
<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>
```

<memtune> は任意ですが、ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.5 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パラメーターに関する詳細を提供します。省略した場合は、オペレーティングシステムのデフォルト値が使用されます。パラメーターはプロセス全体に適用されるため、制限を設定する場合は、ゲスト仮想マシンの RAM をゲスト仮想マシンのビデオ RAM に追加して値を決定するため、メモリーオーバーヘッドがある程度発生します。調整可能パラメーターごとに、**<memory>** と同じ値を使用して、入力中のユニット番号を指定できます。後方互換性の場合は、出力は常にキビバイト (KiB) 単位で行われます。

要素	説明
<hard_limit>	ゲスト仮想マシンが使用できる最大メモリー量。単位は **kibibytes** (1024 バイトのブロック) で表されます。
<soft_limit>	メモリーの競合中に強制されるメモリー制限。この値は、キビバイト (1024 バイトのブロック) で表されます。
<swap_hard_limit>	最大メモリーと、ゲスト仮想マシンが使用できるスワップの合計です。この値は、キビバイト (1024 バイトのブロック) で表されます。これは、**<hard_limit>** 値以上である必要があります。
<min_guarantee>	ゲスト仮想マシンに保証される最小メモリー割り当て。この値は、キビバイト (1024 バイトのブロック) で表されます。

23.8. MEMORY ALLOCATION

ゲスト仮想マシンがクラッシュした場合に、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 (***dumpCore='on'***) に含めるか (***dumpCore='off'***) 含まないかを制御できます。デフォルト設定は ***on*** であるため、このパラメーターが ***off*** に設定されていない場合は、ゲスト仮想マシンのメモリーがコアダンプファイルに含まれます。

<maxMemory> 要素は、ゲストのランタイムメモリーの最大割り当てを決定します。***slots*** 属性は、ゲストにメモリーを追加するために使用できるスロットの数を指定します。

<memory> 要素は、システムの起動時にゲストに割り当てられるメモリーの上限を指定します。また、NUMA セルサイズ設定を使用して設定することもできます。また、メモリーをホットプラグして、***maxMemory*** で指定された制限に増やすこともできます。

<currentMemory> 要素は、ゲスト仮想マシンの実際のメモリー割り当てを決定します。この値は、ゲスト仮想マシンのメモリーを必要に応じてバルーンできるように、最大割り当て (**<memory>** で設定) より小さく設定できます。省略した場合は、**<memory>** 要素と同じ値がデフォルトになります。unit 属性の動作は、メモリーと同じです。

図23.10 メモリーユニット

```
<domain>
  <maxMemory slots='16' unit='KiB'>1524288</maxMemory>
  <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 core dumpfile -->
  <currentMemory unit='KiB'>524288</currentMemory>
  <!-- makes the current memory unit 524288 KiB -->
  ...
</domain>
```

23.9. NUMA ノードのチューニング

virsh edit を使用して NUMA ノードの調整を行った後、以下のドメイン XML パラメーターが影響を受けます。

図23.11 NUMA ノードのチューニング

```
<domain>
  ...
  <numatune>
    <memory mode="strict" nodeset="1-4,^3"/>
  </numatune>
  ...
</domain>
```

すべてオプションですが、ドメイン XML のこのセクションのコンポーネントは次のとおりです。

表23.6 NUMA ノードのチューニング要素

要素	説明
<numatune>	ドメインプロセスの NUMA ポリシーを制御することで、NUMA ホストの物理マシンのパフォーマンスを調整する方法を説明します。
<memory>	NUMA ホストの物理マシンで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。これにはいくつかのオプション属性が含まれます。**mode** 属性は、**interleave**、**strict**、または **preferred** に設定できます。値が指定されていない場合、デフォルトは **strict** に設定されます。**nodeset** 属性は、**<vcpu>** 要素の **cpuset** 属性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示すために使用できます。その値は、**static** または **auto** のいずれかになります。**<nodeset>** 属性が指定されている場合、デフォルトは **<vcpu>** または **static** の **<placement>** になります。**auto** は、ドメインプロセスが numad のクエリーから返されたアドバイザリーノードセットからのみメモリーを割り当てることを示し、**nodeset** 属性の値が指定されている場合は無視されます。**vcpu** の **<placement>** 属性を **auto** に設定し、**<numatune>** 属性を指定しない場合は、**<placement> auto** および **strict** モードのデフォルト **<numatune>** が暗黙的に追加されます。

23.10. ブロック I/O チューニング

図23.12 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>
```

すべてオプションですが、ドメイン XML のこのセクションのコンポーネントは次のとおりです。

表23.7 ブロック I/O チューニング要素

要素	説明
<blkiotune>	このオプション要素は、ドメインの**blkio** cgroup 調整可能パラメーターを調整する機能を提供します。省略した場合は、オペレーティングシステムのデフォルト値が使用されます。
<weight>	この任意の weight 要素は、ゲスト仮想マシンの全体的な I/O ウェイトです。この値は、100 - 1000 の範囲で指定してください。
<device>	ドメインには、ドメインが使用している各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のディスクが、1 つのホストの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。また、このパラメーターは、同じホストの物理マシンファイルシステム内のファイルでバッキングされているため、各ゲスト仮想マシンの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設定します (1 つの**<disk>**に適用できる **<iotune>** 要素との違いを確認してください)。各デバイス要素には、2 つの必須サブ要素があります。**<path>**はデバイスの絶対パスを表し、**<weight>** はそのデバイスの相対的な重みを示します。許容範囲は 100 - 1000 になります。

23.11. リソースのパーティション設定

ハイパーバイザーを使用すると、仮想マシンをリソースパーティションに配置できます。この場合、上

記パーティションのネストが考えられます。**<resource>** 要素は、リソースのパーティション設定に関連する設定をグループ化します。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の子要素パーティションをサポートしています。リストにパーティションがない場合は、ドメインがデフォルトパーティションに置かれます。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する必要があります。デフォルトでは、(ハイパーバイザー固有の) デフォルトパーティションのみが存在すると想定できます。

図23.13 リソースのパーティション設定

```
<resource>
  <partition>/virtualmachines/production</partition>
</resource>
```

現在、リソースパーティションは、KVM ドライバーおよび LXC ドライバーで対応しています。これは、マウントされたすべてのコントローラーで、パーティションパスを cgroups ディレクトリーにマッピングします。

23.12. CPU モデルとトポロジー

このセクションでは、CPU モデルの要件を説明します。すべてのハイパーバイザーには、ゲストにデフォルトで表示される CPU 機能に関する独自のポリシーがあることに注意してください。KVM によりゲストに提示される CPU 機能のセットは、ゲスト仮想マシンの設定で選択された CPU モデルによって異なります。**qemu32** と **qemu64** は基本的な CPU モデルですが、他のモデル (追加機能付き) も利用できます。各モデルとそのトポロジーは、ドメイン XML の以下の要素を使用して指定されます。

図23.14 CPU モデルとトポロジーの例 1

```
<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23.15 CPU モデルとトポロジーの例 2

```
<cpu mode='host-model'>
   <model fallback='forbid'/>
   <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

図23.16 CPU モデルとトポロジーの例 3

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

CPU モデルやその機能に制限を設けない場合は、以下のような簡単な **<cpu>** 要素を使用できます。

図23.17 CPU モデルとトポロジーの例 4

```
<cpu>
   <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

図23.18 PPC64/PSeries CPU モデルの例

```
<cpu mode='custom'>
  <model>POWER8</model>
</cpu>
```

図23.19 aarch64/virt CPU モデルの例

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

ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.8 CPU モデルとトポロジー要素

要素	説明
<cpu>	これは、ゲスト仮想マシンの CPU 要件を説明するためのメインコンテナーです。
<match>	ゲスト仮想マシンに仮想 CPU を提供する方法を指定する場合は、この要件を満たす必要があります。トポロジーが **<cpu>** 内で唯一の要素である場合は、**match** 属性を省略できます。**match** 属性に使用できる値は、以下のとおりです。 ● **minimum** - 指定する CPU モデルおよび機能では、最低限必要な CPU が説明されます。 ● **exact** - ゲスト仮想マシンに提供される仮想 CPU は、仕様と完全に一致します。 ● **strict** - ホストの物理マシンの CPU が仕様と完全に一致しない限り、ゲスト仮想マシンが作成されません。 **match** は省略可能で、デフォルトは **exact** になります。

要素	説明
<モード>	このオプション属性を使用すると、ゲスト仮想マシンの CPU を、ホストの物理マシンの CPU にできるだけ近づけるように設定できます。**mode** 属性に使用できる値は、以下のとおりです。 ● **custom** - CPU がゲスト仮想マシンにどのように表示されるかを説明します。**mode** 属性が指定されていない場合のデフォルト設定です。このモードでは、ゲスト仮想マシンを起動しているホストの物理マシンがどのホストの物理マシンであっても、永続ゲスト仮想マシンが同じハードウェアを認識できるようになります。 ● **host-model** - ホストの物理マシンの CPU 定義を capabilities XML からドメイン XML にコピーするためのショートカット。CPU 定義はドメインの起動直前にコピーされるため、同じ XML を異なるホストの物理マシンで使用することも、各ホストの物理マシンがサポートする最適なゲスト仮想マシンの CPU を提供することもできます。**match** 属性および任意の機能要素は、このモードでは使用できません。詳細は、libvirt アップストリームの Web サイト を参照してください。 ● **host-passthrough** このモードでは、ゲスト仮想マシンから見える CPU は、libvirt 内のエラーの原因となる要素を含め、ホストの物理マシン CPU とまったく同じです。このモードのデメリットは、ゲストの仮想マシン環境を異なるハードウェアで再生できないため、このモードを使用する場合は十分注意して行うことが推奨されます。**model** 要素および **feature** 要素はこのモードでは使用できません。
<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 モデルとその定義のリストは、libvirt の データディレクトリーにインストールされている **cpu_map.xml** ファイルを参照してください。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 機能のリストを維持しながら、ハイパーバイザーが対応する最も近いモデルに自動的にフォールバックします。オプションの **fallback** 属性を使用すると、この動作を禁止できます。この場合、対応していない CPU モデルを要求するドメインを起動しようとすると失敗します。フォールバック属性で対応している値は、**allow** (デフォルト) と **forbid** です。オプションの **vendor_id** 属性を使用すると、ゲスト仮想マシンが認識するベンダー ID を設定できます。ちょうど 12 文字の長さである必要があります。設定しない場合は、ホスト物理マシンのベンダー iID が使用されます。一般的な値は、**AuthenticAMD** および **GenuineIntel** です。

要素	説明
<vendor>	ゲスト仮想マシンが要求する CPU ベンダーを指定します。この要素がないと、ゲスト仮想マシンは、ベンダーに関係なく、指定された機能と一致する CPU で実行します。サポートされているベンダーのリストは、**cpu_map.xml** を参照してください。
<topology>	ゲスト仮想マシンに提供される仮想 CPU に対して要求されるトポロジーを指定します。ソケット、コア、およびスレッドには、それぞれゼロ以外の値を 3 つ指定する必要があります。つまり、CPU ソケットの合計数、ソケットごとのコア数、およびコアごとのスレッド数です。
<機能>	選択した CPU モデルが提供する機能を微調整するために使用する、ゼロ以上の要素を含むことができます。既知の機能名のリストは、**cpu_map.xml** ファイルに記載されています。各機能要素の意味は、ポリシー属性により異なります。この属性は、次のいずれかの値に設定する必要があります。 ● **force** - 仮想マシンがホスト物理マシンの CPU で実際に対応しているかどうかに関係なく、仮想マシンを強制的に対応させます。 ● **require** - この機能がホストの物理マシンの CPU で対応していない場合は、ゲスト仮想マシンの作成に失敗することを指示します。これはデフォルト設定です。 ● **optional** - この機能は仮想 CPU で対応していますが、ホストの物理マシン CPU で対応している場合に限ります。 ● **disable** - これは仮想 CPU では対応していません。 ● **forbid** - この機能がホストの物理マシンの CPU でサポートされていると、ゲスト仮想マシンの作成に失敗します。

23.12.1. 指定した CPU の機能セットの変更

CPU モデルには固有の機能セットがありますが、個々の機能コンポーネントは機能ごとに許可または禁止することができるため、CPU のより個別化された設定が可能となります。

手順23.1 CPU 機能の有効化と無効化

1. 開始するには、ゲスト仮想マシンをシャットダウンします。
2. **virsh edit [domain]** 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変更して、属性値 **'allow'** を追加して機能を強制的に許可するか、**'forbid'** を追加して機能のサポートを拒否します。

図23.20 CPU 機能の有効化または無効化の例

```
<!-- 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23.21 例 2 CPU 機能の有効化または無効化

```
<!--original feature set-->
<cpu match='exact'>
    <model fallback='allow'>core2duo</model>
    <vendor>Intel</vendor>
    <topology sockets='1' cores='2' threads='1'/>
    <feature policy='disable' name='lahf_lm'/>
</cpu>

<!--changed feature set-->
<cpu match='exact'>
    <model fallback='allow'>core2duo</model>
    <vendor>Intel</vendor>
    <topology sockets='1' cores='2' threads='1'/>
    <feature policy='enable' name='lahf_lm'/>
  </cpu>
```

4. 変更が完了したら、設定ファイルを保存して、ゲスト仮想マシンを起動します。

23.12.2. 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。

図23.22 ゲスト仮想マシンの NUMA トポロジー

```
<cpu>
  <numa>
    <cell cpus='0-3' memory='512000'/>
    <cell cpus='4-7' memory='512000'/>
  </numa>
</cpu>
...
```

各セル要素は、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 kibibytes (1024 バイトのブロック) で指定します。各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。

23.13. イベントの設定

ドメイン XML の以下のセクションを使用すると、さまざまなイベントのデフォルトのアクションを上書きできます。

図23.23 イベントの設定

```
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<on_lockfailure>poweroff</on_lockfailure>
```

以下の要素のコレクションでは、ゲスト仮想マシンのオペレーティングシステムが、ライフサイクル操作をトリガーする場合にアクションを指定できます。一般的なユースケースは、オペレーティングシステムの初期インストール時に、再起動を強制的に電源オフとして扱うことです。これにより、仮想マシンをインストール後の初回起動時に再設定できます。

ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.9 イベント設定要素

State	説明

State	説明
<on_poweroff>	ゲスト仮想マシンがパワーオフを要求した場合に実行するアクションを指定します。4 つの引数が可能です。 ● **destroy** - この操作によりドメインが完全に終了し、すべてのリソースが解放されます。 ● **restart** - この操作によりドメインが完全に終了し、同じ設定で再起動します。 ● **preserve** - この操作によりドメインは完全に終了しますが、リソースは保持されるため、今後の解析に使用できます。 ● **rename-restart** - この操作によりドメインが完全に終了し、続いて新しい名前で再起動します。
<on_reboot>	ゲスト仮想マシンが再起動を要求した場合に実行するアクションを指定します。4 つの引数が可能です。 ● **destroy** - この操作によりドメインが完全に終了し、すべてのリソースが解放されます。 ● **restart** - この操作によりドメインが完全に終了し、同じ設定で再起動します。 ● **preserve** - この操作によりドメインは完全に終了しますが、リソースは保持されるため、今後の解析に使用できます。 ● **rename-restart** - この操作によりドメインが完全に終了し、続いて新しい名前で再起動します。

State	説明
<on_crash>	ゲスト仮想マシンがクラッシュした場合に実行するアクションを指定します。さらに、以下の追加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がダンプされ、ドメインが完全に終了し、すべてのリソースが解放されます。 ● **coredump-restart** - クラッシュしたドメインのコアがダンプされ、同じ設定でドメインが再起動します。 4 つの引数が可能です。 ● **destroy** - この操作によりドメインが完全に終了し、すべてのリソースが解放されます。 ● **restart** - この操作によりドメインが完全に終了し、同じ設定で再起動します。 ● **preserve** - この操作によりドメインは完全に終了しますが、リソースは保持されるため、今後の解析に使用できます。 ● **rename-restart** - この操作によりドメインが完全に終了し、続いて新しい名前で再起動します。
<on_lockfailure>	ロックマネージャーがリソースロックを失った場合に行うアクションを指定します。以下のアクションは、libvirt によって認識されますが、すべてが個々のロックマネージャーによってサポートされる必要はありません。アクションを指定しないと、各ロックマネージャーはデフォルトのアクションを実行します。以下の引数を使用できます。 ● **poweroff** - ドメインの電源を強制的に切断します。 ● **restart** - ドメインを再起動して、ロックを再取得します。 ● **pause** - ロックの問題が解決したときに手動で再開できるように、ドメインを一時停止します。 ● **ignore** - 何も起こらなかったかのようにドメインを実行し続けます。

23.14. ハイパーバイザーの機能

ハイパーバイザーを使用すると、特定の CPU 機能またはマシン機能を有効 (**state='on'**) または無効 (**state='off'**) にできます。

図23.24 ハイパーバイザーの機能

```
...
<features>
  <pae/>
  <acpi/>
  <apic/>
  <hap/>
  <privnet/>
  <hyperv>
    <relaxed state='on'/>
  </hyperv>
</features>
...
```

すべての機能は、**<features>** 要素にリスト表示されます。**<state>**を指定しないと無効になります。利用可能な機能は、**capabilities** XML を呼び出して確認できますが、完全仮想化ドメインの一般的な設定は次のとおりです。

表23.10 ハイパーバイザーの機能要素

State	説明
<pae>	物理アドレス拡張モードでは、32 ビットのゲスト仮想マシンが 4GB を超えるメモリーにアドレスを指定できます。
<acpi>	電源管理に役立ちます。たとえば、KVM ゲスト仮想マシンでは、正常なシャットダウンを機能させるために必要です。
<apic>	プログラム可能な IRQ 管理を使用できるようにします。この要素には、**on** および **off** の値を持つオプションの属性**eoi** があります。これは、ゲスト仮想マシンの EOI (割り込みの終了) の可用性を設定します。
<hap>	ハードウェアで利用可能な場合に、ハードウェア支援ページングの使用を有効にします。

23.15. 時間管理

ゲスト仮想マシンのクロックは、通常、ホストの物理マシンのクロックから初期化されます。ほとんどのオペレーティングシステムでは、ハードウェアクロックは UTC (デフォルト設定) に保持されることが想定されています。

ゲスト仮想マシンでの正確な時間管理は、仮想プラットフォームにおける鍵となる課題です。さまざまなハイパーバイザーが、さまざまな方法で時間管理の問題を処理しようとしています。**libvirt** は、ドメイン XML の **<clock>** と **<timer>** 要素を使用して、ハイパーバイザーに依存しない時間管理用の設定を

提供します。ドメイン XML は、**virsh edit** コマンドを使用して編集できます。詳細は、「ゲスト仮想マシンの XML 設定の編集」 を参照してください。

図23.25 時間管理

```
...
<clock offset='localtime'>
  <timer name='rtc' tickpolicy='catchup' track='guest'>
    <catchup threshold='123' slew='120' limit='10000'/>
  </timer>
  <timer name='pit' tickpolicy='delay'/>
</clock>
...
```

ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.11 時間管理要素

State	説明

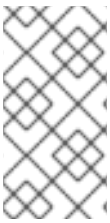
State	説明
<clock>	**<clock>** 要素は、ゲスト仮想マシンのクロックをホスト物理マシンのクロックと同期させる方法を決定するために使用されます。**offset** 属性には 4 つの値を指定できます。これにより、ゲスト仮想マシンのクロックをホスト物理マシンに同期させる方法を詳細に制御できます。ハイパーバイザーは、常にすべてのポリシーをサポートする必要はないことに注意してください。 ● **utc** - 起動時にクロックを UTC に同期します。**utc** モードは、adjustment 属性を使用して制御できる **variable** モードに変換できます。値が **reset** の場合、変換は行われません。数値を入力すると、その値を初期調整として使用し、強制的に **variable** モードに変換されます。デフォルトの調整は、ハイパーバイザー固有です。 ● **localtime** - ゲスト仮想マシンのクロックを、起動時にホスト物理マシンに設定されたタイムゾーンと同期します。調整属性の動作は、**utc** モードと同じです。 ● **timezone** - ゲスト仮想マシンのクロックを、要求されたタイムゾーンに同期します。 ● **variable** - ゲスト仮想マシンクロックに、basis 属性に応じて UTC または **localtime** を基準にした任意のオフセットを適用します。UTC (または **localtime**) に対する差分は、**adjustment** 属性を使用して秒単位で指定します。ゲスト仮想マシンは、時間の経過とともに RTC を調整する自由があり、次回のシステムの再起動時に有効になると予想されます。これは、**utc** モードおよび **localtime** モード (オプション属性の **adjustment='reset'** を使用) とは対照的です。ここでは、システムの再起動ごとに RTC 調整が失われます。また、**basis** 属性は **utc** (デフォルト) または **localtime** のいずれかになります。**clock** 要素には、0 個以上の**<timer>** 要素を指定できます。
<timer>	下記を参照してください。
<あり>	ゲスト仮想マシンで特定のタイマーが使用可能であるかどうかを指定します。**yes** または **no** に設定できます。

注記

<clock> 要素には、子としての**<timer>** 要素を 0 個以上指定できます。**<timer>** 要素は、ゲスト仮想マシンのクロック同期に使用されるタイムソースを指定します。

各**<timer>** 要素では、**name** のみが必要で、その他の属性は任意になります。

- **name** - 変更する**timer**を選択します。指定できる値は、**kvmclock**、**pit**、または **rtc** です。
- **track** - タイマートラックを指定します。次の値を使用できます: **boot**、**guest**、または **wall**。**track** は **name="rtc"** に対してのみ有効です。
- **tickpolicy** - ゲスト仮想マシンにティックを挿入する期限が過ぎた場合の動作を指定します。設定可能な値は、以下のとおりです。
  - **delay** - 通常の速度でティックを配信し続けます。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、高いレートでティックを配信しますキャッチアップが完了すると、ゲスト仮想マシンの時間は表示されません。さらに、3 つのオプション属性 (それぞれ正の整数) (threshold、slew、および limit) があります。
  - **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。マージの実行方法によっては、ゲスト仮想マシンの時間が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今後の挿入を続行します。失われたティックを処理する明示的なステートメントがない限り、ゲスト仮想マシンの時間が遅延する場合があります。



注記

値 **utc** は、デフォルトで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を **localtime** の値で実行している場合は、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と同期させるため、クロックオフセットを変更して別の値にする必要があります。

例23.1 常に UTC に同期する

```
<clock offset="utc" />
```

例23.2 常にホストの物理マシンのタイムゾーンに同期する

```
<clock offset="localtime" />
```

例23.3 任意のタイムゾーンに同期する

```
<clock offset="timezone" timezone="Europe/Paris" />
```

例23.4 UTC + 任意のオフセットに同期する

```
<clock offset="variable" adjustment="123456" />
```

23.16. タイマー要素の属性

name 要素には、使用するタイムソースの名前が含まれます。値は、以下のいずれかになります。

表23.12 名前属性値

値	説明
pit	Programmable Interval Timer - 周期的な割り込みがあるタイマーです。この属性を使用すると、**tickpolicy** 遅延がデフォルト設定になります。
rtc	Real Time Clock - 周期的な割り込みがある継続的に実行するタイマー。この属性は、**tickpolicy catchup** サブ要素に対応します。
kvmclock	KVM clock - KVM ゲスト仮想マシンに推奨されるクロックソースです。KVM pvclock または kvm-clock を使用すると、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時間を読み取ることができます。

track 属性は、タイマーが追跡するものを指定し、***rtc*** の **name** 値に対してのみ有効です。

表23.13 属性値を追跡する

値	説明
boot	古い **ホストの物理マシン** オプションに対応します。これはサポート対象外の追跡オプションです。
guest	RTC は、ゲストの仮想マシンの時間を常に追跡します。
wall	RTC は常にホスト時間を追跡します。

tickpolicy 属性と値は、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23.14 tickpolicy 属性値

値	説明
delay	通常のレートで配信を継続します (ティックは遅延します)。

値	説明
catchup	キャッチアップするために、より高いレートで配信されます。
merge	ティックが 1 つのティックにマージされます。
discard	不明なティックはすべて破棄されます。

present 属性は、ゲスト仮想マシンに表示されるデフォルトのタイマーセットを上書きするために使用されます。**present** 属性には、以下の値を指定できます。

表23.15 present 属性値

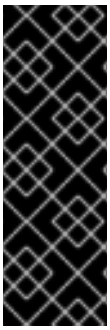
値	説明
はい	このタイマーをゲスト仮想マシンに強制的に表示します。
いいえ	このタイマーをゲスト仮想マシンに表示しないように強制します。

23.17. DEVICES

この XML 要素のセットはすべて、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。以下のデバイスはすべて、主な**<devices>** 要素の子として示されます。

以下の仮想デバイスに対応しています。

- virtio-scsi-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 バス仮想乱数発生器



重要

virtio デバイスを作成し、ベクトル数を 32 より大きい値に設定すると、デバイスは、Red Hat Enterprise Linux 6 ではゼロに設定されているかのように動作しますが、Enterprise Linux 7 では動作しません。生成されるベクトル設定の不一致により、いずれかのプラットフォームの virtio デバイスのベクトル数が 33 以上に設定されていると、移行エラーが発生します。したがって、vector の値を 32 より大きく設定することは推奨されません。**virtio-balloon-pci** および **virtio-rng-pci** を除くすべての virtio デバイスは、**vector** 引数を受け入れます。

図23.26 デバイス - 子要素

```
...
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
</devices>
...
```

<emulator> 要素の内容は、デバイスモデルエミュレーターバイナリーへの完全修飾パスを指定します。capabilities XML は、各特定の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使用する推奨されるデフォルトエミュレーターを指定します。

23.17.1. ハードドライブ、フロッピーディスク、および CD-ROM

ドメイン XML のこのセクションでは、**<disk>** 要素で指定されている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など、ディスクのように見えるデバイスを指定します。

図23.27 デバイス - ハードドライブ、フロッピーディスク、CD-ROM の例

```
<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23.28 デバイス - ハードドライブ、フロッピーディスク、CD-ROM の例 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23.29 デバイス - ハードドライブ、フロッピーディスク、CD-ROM の例 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23.30 デバイス - ハードドライブ、フロッピーディスク、CD-ROM の例 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23.31 デバイス - ハードドライブ、フロッピーディスク、CD-ROM の例 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23.32 デバイス - ハードドライブ、フロッピーディスク、CD-ROM の例 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23.33 デバイス - ハードドライブ、フロッピーディスク、CD-ROM の例 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23.34 デバイス - ハードドライブ、フロッピーディスク、CD-ROM の例 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>
...
```

23.17.1.1. ディスク要素

<disk> 要素は、ディスクを記述するための主要なコンテナーです。属性**type** は、**<disk>** 要素で使用できます。以下のタイプが許可されています。

- **file**
- **block**
- **dir**
- **network**

詳細は、libvirt upstream pages を参照してください。

23.17.1.2. ソース要素

ディスクソースを表します。ディスクソースは、次のようにディスクタイプ属性に従います。

- **<file>** - **file** 属性は、ディスクが置かれているファイルへの完全修飾パスを指定します。
- **<block>** - **dev** 属性は、ディスクとして機能するホストデバイスへの完全修飾パスを指定します。
- **<dir>** - **dir** 属性は、ディスクとして使用されるディレクトリーの完全修飾パスを指定します。
- **<network>** - **protocol** 属性は、要求されたイメージへのアクセスに使用されるプロトコルを指定します。指定できる値は、**nbd**、**isci**、**rbd**、**sheepdog**、および **gluster** です。
  - **protocol** 属性が **rbd**、**sheepdog**、または **gluster** の場合は、追加の属性である **name** が必須となります。この属性は、使用されるボリュームおよびイメージを指定します。
  - **protocol** 属性が **nbd** の場合、**name** 属性はオプションになります。
  - **protocol** 属性が **isci** の場合、**name** 属性には、ターゲットの名前とスラッシュを区切った論理ユニット番号を含めることができます。たとえば、iqn.2013-07.com.example:iscsi-pool/1 のようになります。指定しない場合、デフォルトの LUN はゼロになります。
- **<volume>** - 基となるディスクソースは、**pool** 属性および **volume** 属性で表されます。
  - **<pool>** - ディスクソースが存在するストレージプール (**libvirt** が管理) の名前です。
  - **<volume>** - ディスクソースとして使用されるストレージボリューム (**libvirt** が管理) の名前です。

    volume 属性の値は、**virsh vol-list [pool-name]** の Name 列の出力です。

ディスクのタイプが **network** の場合、**source** には、接続するホスト物理マシンを指定するために使用される **type='dir'** および **type='network'** を含む、ゼロ以上の **host** サブ要素が含まれることがあります。CD-ROM またはフロッピー (デバイス属性) を表す**file**ディスクタイプで、ソースファイルにアクセスできない場合にディスクを使用する動作のポリシーを定義できます。これは、次のいずれかの値で **startupPolicy** 属性を設定します。

- **mandatory** は、何らかの理由で欠落している場合に障害の原因となります。これはデフォルト設定です。
- **requisite** は、起動時に欠落している場合は失敗し、移行、復元、または復帰時に欠落している場合はドロップします。
- **optional** は、開始試行時に欠落している場合はドロップします。

23.17.1.3. ミラー要素

この要素は、ハイパーバイザーが **BlockCopy** 操作を起動し、属性ファイルの **<mirror>** の場所が最終的にソースと同じ内容を持ち、属性形式のファイル形式 (ソースの形式とは異なる場合あり) の場合に存在します。属性 ready が存在する場合は、ディスクがピボットする準備ができていることが分かっています。存在しない場合は、ディスクが依然としてコピー中である可能性があります。現在、この要素は出力でのみ有効で、入力では無視されます。

23.17.1.4. ターゲット要素

<target> 要素は、ディスクがゲスト仮想マシンのオペレーティングシステムに公開されるバスまたはデバイスを制御します。**dev** 属性は、論理デバイスの名前を示します。実際のデバイス名が、ゲスト仮想マシンのオペレーティングシステムのデバイス名にマッピングされる保証はありません。オプションのバス属性では、エミュレートするディスクデバイスの種類を指定します。指定できる値はドライバー固有で、一般的な値は **ide**、**scsi**、**virtio**、**kvm**、**usb**、または **sata** です。省略した場合は、デバイス名のスタイルからバスタイプが推測されます。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。オプションの属性 **tray** は、リムーバブルディスク (CD-ROM やフロッピーディスクなど) のトレイステータスを示します。値は**open** または **closed** のいずれかです。デフォルト設定は **closed** です。

23.17.1.5. iotune 要素

オプションの **<iotune>** 要素は、デバイスごとに異なる値を使用して、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 (これを、ドメインにグローバルに適用する**blkiotune**要素と比較してください)。この要素には、以下のオプションのサブ要素があります (まったく指定されていないサブ要素、または **0** の値で指定されているサブ要素は、制限がないことを意味する点に注意してください)。

- **<total_bytes_sec>** - 合計スループット制限 (バイト/秒)。この要素は、**<read_bytes_sec>** または **<write_bytes_sec>** とは併用できません。
- **<read_bytes_sec>** - 読み取りスループットの制限 (バイト/秒)。
- **<write_bytes_sec>** - 書き込みスループットの制限 (バイト/秒)。
- **<total_iops_sec>** - 1秒あたりの I/O 操作の合計です。この要素は、 **<read_iops_sec>** または **<write_iops_sec>** とは併用できません。
- **<read_iops_sec>** - 1秒あたりの読み取り I/O 操作数。
- **<write_iops_sec>** - 1秒あたりの書き込み I/O 操作数。

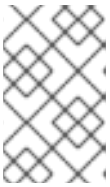
23.17.1.6. ドライバー要素

任意の **<driver>** 要素を使用すると、ディスクの提供に使用されるハイパーバイザードライバーに関する詳細を指定できます。以下のオプションが使用できます。

- 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名を選択し、任意の type 属性はサブタイプを提供します。
- オプションの **cache** 属性は、キャッシュメカニズムを制御します。使用できる値は、**default**、**none**、**writethrough**、**writeback**、**directsync** (**writethrough** と同じだが、ホスト物理マシンのページキャッシュをバイパスする) および **unsafe** (ホスト物理マシンがすべてのディスク I/O をキャッシュし、ゲスト仮想マシンからの同期要求は無視される) です。
- オプションの **error_policy** 属性は、ディスクの読み取りエラー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。読み取りエラーのみに対する動作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** は読み取りおよび書き込みエラーの両方に使用されます。**rerror_policy** が指定されている場合は、読み取りエラーの **error_policy** をオーバーライドします。また、**enospace** は読み取りエラーの有効なポリシーではないため、**error_policy** が **enospace** に設定され、**rerror_policy** が指定されていない場合は、読み取りエラーのデフォルト設定である **report** が使用される点に注意してください。

- オプションの **io** 属性は、I/O 上で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** に対応します。オプションの **ioeventfd** 属性を使用すると、virtio ディスクデバイスのドメイン I/O の非同期処理を設定できます。デフォルトはハイパーバイザーにより決定します。指定できる値は **on** と **off** です。これを有効にすると、別のスレッドが I/O を処理している間にゲスト仮想マシンを実行できます。通常、I/O の実行中にシステム CPU の使用率が高くなったゲスト仮想マシンは、この恩恵を受けます。一方、ホストの物理マシンが過負荷になると、ゲスト仮想マシンの I/O レイテンシーが増加します。ただし、デフォルト設定を変更せず、ハイパーバイザーによる設定の決定を許可することが推奨されます。



注記

ioeventfd 属性は、**disk** XML セクションの **<driver>** 要素と、**device** XML セクションの **<driver>** 要素に含まれます。前者の場合は virtIO ディスク、後者の場合は SCSI ディスクに影響を及ぼします。

- オプションの **event_idx** 属性は、デバイスイベント処理の一部の側面を制御するもので、**on** または **off** に設定できます。**on** に設定すると、ゲスト仮想マシンの中断および終了の回数が減ります。デフォルトではハイパーバイザーが決定し、デフォルトの設定は **on** になります。この動作が必要ない場合は、**off** を設定すると強制的に機能が無効になります。ただし、デフォルト設定を変更せず、ハイパーバイザーによる設定の指示を許可することが強く推奨されます。

- オプションの **copy_on_read** 属性は、リードバック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。使用できる値は、**on** または **<off>** のいずれかです。**copy-on-read** は、同じバッキングファイルセクターに繰り返しアクセスすることを回避し、バッキングファイルが低速のネットワーク上にある場合に役立ちます。デフォルトでは、**copy-on-read**は **off** です。

- この**discard='unmap'**は、破棄に対応するように設定できます。同じ行を、**discard='ignore'** に置き換えて無効にすることができます。デフォルト設定は、**discard='ignore'** です。

23.17.1.7. 追加のデバイス要素

device 要素内では、以下の属性を使用できます。

- **<boot>** - ディスクが起動可能であることを指定します。

  追加の起動値

  - **<order>** - システムの起動時にデバイスを試行する順序を指定します。
  - **<per-device>** ブート要素は、BIOS ブートローダーセクションの一般的なブート要素とは併用できません。

- **<encryption>** - ボリュームの暗号化方法を指定します。

- **<readonly>** - ゲスト仮想マシンがデバイスを変更できないことを示します。この設定は、**attribute <device='cdrom'>** を使用するディスクのデフォルト値です。

- **<shareable>** デバイスがドメイン間で共有されることが期待されることを示します (ハイパーバイザーおよびオペレーティングシステムがこれに対応している場合)。**shareable** を使用すると、そのデバイスに **cache='no'** が指定されます。

- **<transient>** - デバイスのコンテンツの変更は、ゲスト仮想マシンの終了時に自動的に元に戻す必要があることを示します。一部のハイパーバイザーでは、ディスクに **transient** マークを付けると、ドメインが移行またはスナップショットに参加できなくなります。

- **<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。(例: **<serial>**WD-WMAP9A966149**</serial>**)

- **<wwn>** - 仮想ハードディスクまたは CD-ROM ドライブのワールドワイドネーム (WWN) を指定します。16 桁の 16 進数で設定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。印刷可能な 8 文字を超えてはなりません。

- **<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。印刷可能な 16 文字を超えてはなりません。

- **<host>** - 以下の属性をサポートします。

  - **name** - ホスト名を指定します。

  - **port** - ポート番号を指定します。

  - **transport** - トランスポートの種類を指定します。

  - **socket** - ソケットのパスを指定します。

  この要素の意味と要素数は、プロトコルに基づく追加のホスト属性 に示すように **protocol** 属性により異なります。

  プロトコルに基づく追加のホスト属性

  - **nbd** - ***nbd-server*** を実行しているサーバーを指定します。使用できるのは、1 台のホスト物理マシンだけです。このプロトコルのデフォルトポートは 10809 です。

  - **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で使用できます。

  - **sheepdog** - ***sheepdog***サーバーのいずれかを指定し (デフォルトは localhost:7000)、ホスト物理マシンの 1 つと使用するか、いずれとも使用することができません。

  - **gluster** - glusterd デーモンを実行しているサーバーを指定します。使用できる物理マシンは 1 つだけです。トランスポート属性の有効な値は、**tcp**、**rdma**、または **unix** です。何も指定しないと、**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したスロットに関連付けます。実際の **<controller>** デバイスは多くの場合、推測できますが、明示的に指定することもできます。**type** 属性は必須で、通常は **pci** または **drive** です。**pci** コントローラーの場合、**bus**、**slot**、および **function** の追加属性と、オプションの **domain** および **multifunction** が存在する必要があります。**multifunction** のデフォルトは **off** です。**drive** コントローラーでは、追加の属性 **controller**、**bus**、**target**、および **unit** が利用できます。それぞれの属性のデフォルト設定は **0** です。

- **auth** - ソースへのアクセスに必要な認証情報を提供します。これには、認証時に使用するユーザー名を識別する必須属性**username**と、必須属性**type**を持つサブ要素**secret** が含まれます。
- **geometry** - ジオメトリー設定を上書きする機能を提供します。これは、主に S390 DASD ディスクまたは古い DOS ディスクに役立ちます。これには、以下のパラメーターを指定できます。
  - **cyls** - シリンダーの数を指定します。
  - **heads** - ヘッドの数を指定します。
  - **secs** - トラックごとのセクター数を指定します。
  - **trans** - BIOS-Translation-Modes を指定し、 **none**、**lba**、または **auto** の値を設定できます。
- **blockio** - ブロックデバイスを、以下のブロックデバイスプロパティーのいずれかで上書きできます。

  blockio **オプション**

  - **logical_block_size** - ゲスト仮想マシンのオペレーティングシステムを報告し、ディスク I/O の最小ユニットを説明します。
  - **physical_block_size** - ゲスト仮想マシンのオペレーティングシステムを報告し、ディスクのハードウェアセクターサイズを説明します。これは、ディスクデータの調整に関連付けることができます。

23.17.2. デバイスアドレス

多くのデバイスには、仮想バスに配置されたデバイスがゲスト仮想マシンに提示される場所を説明するオプションの **<address>** サブ要素があります。入力でアドレス (またはアドレス内の任意の属性) が省略された場合、libvirt は適切なアドレスを生成します。レイアウトをさらに制御する必要がある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の例は、以下を参照してください。

各アドレスには、デバイスが稼働しているバスを説明する必須の属性 ***type*** があります。特定のデバイスに使用するアドレスの選択は、デバイスやゲスト仮想マシンのアーキテクチャーによって一部が制約されます。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、32 ビット AMD および Intel の **type='pci'**、または AMD64 と Intel 64、ゲスト仮想マシン、または PowerPC64 pseries ゲスト仮想マシンの **type='spapr-vio'** を使用します。各アドレス**<type>**には、デバイスの配置先を制御する追加のオプション属性があります。追加の属性は、以下のとおりです。

- **type='pci'** - PCI アドレスには、以下の追加属性があります。
  - **domain** (2 バイトの 16 進数の整数で、現在 qemu で使用されていません)
  - **bus** (0 から 0xff までの 16 進数の値)
  - **slot** (0x0 から 0x1f までの 16 進数の値)
  - **function** (0 から 7 までの値)
  - また、**multi-function** 属性も利用できます。これは、PCI 制御レジスターの特定のスロットまたは機能に対して、多機能ビットをオンにすることを制御します。この多機能属性は、デフォルトでは **'off'** ですが、多機能を使用するスロットの機能 0 の場合は **'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 進数の値)
  - **port** - (1.2 または 2.1.3.1 などの最大 4 オクテットのドット表記)
- **type='spapr-vio'** - PowerPC pseries ゲスト仮想マシンでは、SPAPR-VIO バスにデバイスを割り当てることができます。フラットな 64 ビットのアドレス空間を持ちます。通常、デバイスは 0x1000 の倍数 (ゼロ以外) で割り当てられますが、その他のアドレスは有効で、**libvirt**で許可されています。開始レジスターの 16 進数のアドレスを決定する追加の **reg** 属性を、この属性に割り当てることができます。

23.17.3. 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っては、1 つのバスに多くの仮想デバイスを割り当てることができます。通常の状況では、libvirt はバスに使用するコントローラーを自動的に推測できます。ただし、ゲスト仮想マシン XML で明示的な **<controller>** 要素を指定する必要がある場合があります。

図23.35 コントローラー要素

```
...
<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>
...
```

各コントローラーには、必須属性**type** (**"ide", "fdc", "scsi", "sata", "usb", "ccid", or "virtio-serial"** のいずれか) と、(**address** 要素のコントローラー属性で使用するために) バスコントローラーが検出される順序を表す 10 進数の整数である必須属性**index** があります。**"virtio-serial"** コントローラーには、さらに 2 つのオプション属性 (**ports** および **vectors**) があり、コントローラーを介して接続できるデバイスの数を制御します。

<controller type='scsi'> には、**"auto", "buslogic", "ibmvscsi", "lsilogic", "lsias1068", "virtio-scsi or "vmpvscsi"** の 1 つであるオプションの属性**model** があります。**<controller type='scsi'>** には、指定したキュー数のマルチキューサポートを有効にする属性 **num_queues** もあります。また、**ioeventfd** 属性を使用できます。これは、コントローラーが、SCSI ディスクで非同期処理を使用するかどうかを指定します。許可される値は "on" と "off" です。

"usb" コントローラーには、**"piix3-uhci", "piix4-uhci", "ehci", "ich9-ehci1", "ich9-uhci1", "ich9-uhci2", "ich9-uhci3", "vt82c686b-uhci", "pci-ohci" or "nec-xhci"** の 1 つであるオプションの属性**model** があります。また、ゲスト仮想マシンで USB バスを明示的に無効にする必要がある場合は、**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。

コントローラー自体が PCI バスまたは USB バスにある場合は、オプションのサブ要素 **address** は、上記のセマンティクスを使用して、コントローラーとマスターバス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完全なリレーション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マスターと同じバスにあるため、コンパニオンインデックスの値は同じである必要があります。

図23.36 デバイス - コントローラー - USB

```
...
<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>
...
```

23.17.4. デバイスリース

ロックマネージャーを使用する場合は、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、リースを取得できない限り、ゲスト仮想マシンを起動しないようにします。従来の管理ツールを使用して設定すると、ドメイン XML の以下のセクションが影響を受けます。

図23.37 デバイス - デバイスリース

```
...
<devices>
  ...
  <lease>
    <lockspace>somearea</lockspace>
    <key>somekey</key>
    <target path='/some/lease/path' offset='1024'/>
  </lease>
  ...
</devices>
...
```

lease セクションには、以下の引数を指定できます。

- **lockspace** - 鍵が保持されているロック領域を識別する任意の文字列です。ロックマネージャーは、ロックスペース名の形式や長さに追加の制限を課す場合があります。
- **key** - 取得するリースを一意に識別する任意の文字列。ロックマネージャーは、キーの形式や長さに追加の制限を課す場合があります。
- **target** - ロックスペースに関連付けられたファイルの完全修飾パス。オフセットは、リースがファイル内に保存される場所を指定します。ロックマネージャーがオフセットを必要としない場合は、この値を **0** に設定します。

23.17.5. ホスト物理マシンのデバイス割り当て

23.17.5.1. 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、**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。管理ツールを使用してホスト物理マシンを変更し、ドメイン XML ファイルの以下のセクションを設定します。

図23.38 デバイス - ホスト物理マシンのデバイス割り当て

```
...
<devices>
  <hostdev mode='subsystem' type='usb'>
    <source startupPolicy='optional'>
      <vendor id='0x1234'/>
      <product id='0xbeef'/>
    </source>
    <boot order='2'/>
  </hostdev>
</devices>
...
```

または、以下を実行することもできます。

図23.39 デバイス - ホスト物理マシンのデバイス割り当ての代替案

```
...
<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>
...
```

または、以下を実行することもできます。

図23.40 デバイス - ホスト物理マシン scsi デバイスの割り当ての代替案

```
...
<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>
..
```

ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.16 ホスト物理マシンの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、ホストの物理マシンデバイスを記述する主な要素です。以下のオプションを取ります。 ● **mode** - この値は、USB デバイスおよび PCI デバイスでは常に**subsystem**です。 ● **type** - USB デバイスの場合は **usb**、PCI デバイスの場合は **pci**。 ● **managed** - デバイスの*管理モード*を切り替えます。 ○ PCI デバイスに対して **yes** に設定すると、ゲストマシンに接続され、必要に応じてゲストマシンから切り離され、ホストマシンに再接続されます。 デバイス割り当ての一般的な使用には、**managed='yes'** を推奨します。 ○ PCI および USB デバイスの場合は **no** に設定するか、省略しても、デバイスはゲストに接続したままになります。ホストでデバイスを使用できるようにするには、ゲストを起動したり、デバイスをホットプラグしたりする前に、引数 **virNodeDeviceDettach** または **virsh nodedev-dettach** を使用する必要があります。さらに、デバイスのホットアンプラグまたはゲストの停止後に、**virNodeDeviceReAttach** または **virsh nodedev-reattach** を使用する必要があります。**managed='no'** は、主に特定のゲスト専用のデバイスに推奨されます。

パラメーター	説明
比較元	ホストの物理マシンから見たデバイスを説明します。USB デバイスは、**vendor** 要素および **product** 要素を使用してベンダーまたはプロダクト ID でアドレスを指定するか、**address** 要素を使用して、ホスト物理マシンのデバイスのアドレスを指定します。一方、PCI デバイスは、アドレスによってのみ記述できます。USB デバイスのソース要素には、**startupPolicy** 属性が含まれる場合があります。これを使用すると、指定したホスト物理マシンの USB デバイスが見つからない場合の対処方法に関するルールを定義できます。この属性は、次の値を受け入れます。 ● **mandatory** - 何らかの理由でない場合は失敗します (デフォルト)。 ● **requisite** - システムの起動時にない場合は失敗し、migrate/restore/revert にない場合はドロップします。 ● **optional** - 起動の試行時にない場合はドロップします。
vendor, product	このような要素には、それぞれ USB ベンダーと製品 ID を指定する **id** 属性があります。ID は、10 進数、16 進数 (0x で始まる)、または 8 進数 (0 で始まる) 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。この属性の順序により、システムの起動シーケンスでデバイスが試行される順序が決定します。デバイスごとのブート要素は、BIOS ブートローダーセクションの一般的なブート要素とは併用できません。
rom	PCI デバイスの ROM がゲスト仮想マシンに表示される方法を変更する場合に使用されます。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で表示されるかどうかを決定します。(PCI のドキュメントでは、**rom bar** 設定により、ROM のベースアドレスレジスターの存在が制御されます。)**rom bar**を指定しないと、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指定するために使用されます。これは、たとえば、SR-IOV 対応イーサネットデバイスの仮想機能に PXE ブート ROM を提供する場合に役立ちます (VF にはブート ROM がありません)。

パラメーター	説明
address	また、デバイスがホストの物理マシンに表示される USB バスとデバイス番号を指定する **bus** および **bus** 属性もあります。この属性の値は、10 進数、16 進数 (0x で始まる)、または 8 進数 (0 で始まる) で指定できます。PCI デバイスの場合、この要素は、**lspci** または **virsh nodedev-list** で検出されるデバイスを指定できるように、3 つの属性を持ちます。

23.17.5.2. ブロック / キャラクターデバイス

ホストの物理マシンのブロック / キャラクターデバイスは、マネジメントツールを使用してドメイン XML **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でのみ可能であることに注意してください。

図23.41 デバイス - ホスト物理マシンのデバイス割り当てブロックキャラクターデバイス

```
...
<hostdev mode='capabilities' type='storage'>
  <source>
    <block>/dev/sdf1</block>
  </source>
</hostdev>
...
```

別のアプローチは以下のとおりです。

図23.42 デバイス - ホスト物理マシンの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案 1

```
...
<hostdev mode='capabilities' type='misc'>
  <source>
    <char>/dev/input/event3</char>
  </source>
</hostdev>
...
```

また別のアプローチは以下のとおりです。

図23.43 デバイス - ホスト物理マシンの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案 2

```
...
<hostdev mode='capabilities' type='net'>
  <source>
    <interface>eth0</interface>
  </source>
</hostdev>
...
```

ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.17 ブロック/ キャラクターデバイス要素

パラメーター	説明
hostdev	これは、ホスト物理マシンデバイスを説明する主要なコンテナーです。ブロックデバイス/キャラクターデバイスの場合、パススルー**mode**は常に**capabilities** になります。ブロックデバイスの場合は **type** は **block** に、キャラクターデバイスの場合は **char** になります。
比較元	これは、ホストの物理マシンから見たデバイスを説明します。ブロックデバイスの場合は、ホスト物理マシンのオペレーティングシステムのブロックデバイスへのパスが、ネストされた **block** 要素で提供され、キャラクターデバイスの場合は **char** 要素が使用されます。

23.17.6. リダイレクトされたデバイス

キャラクターデバイスを介した USB デバイスのリダイレクトは、ドメイン XML の次のセクションを変更して設定します。

図23.44 デバイス - リダイレクトされたデバイス

```
...
<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>
...
```

ドメイン XML のこのセクションのコンポーネントは以下のとおりです。

表23.18 リダイレクトされたデバイス要素

パラメーター	説明
redirdev	これは、リダイレクトされたデバイスを記述するためのメインコンテナーです。USB デバイスの場合、**bus** は **usb** である必要があります。追加の属性タイプが必要で、対応しているシリアルデバイスタイプのいずれかと一致するものが、トンネルのホスト物理マシン側の説明に使用されます。通常は、**type='tcp'** または **type='spicevmc'** (SPICE グラフィックデバイスの usbredir チャンネルを使用) です。**redirdev** 要素には、オプションのサブ要素 **address** があります。これは、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。**source** などのサブ要素は、**targe** サブ要素は必要ありませんが、指定の**type**に応じて必要となる場合があります (キャラクターデバイスのコンシューマーは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、システムの起動順序においてデバイスが試行される順序を決定します。デバイスごとのブート要素は、BIOS ブートローダーセクションの一般的なブート要素とは併用できません。
redirfilter	これは、特定のデバイスをリダイレクトから除外するフィルタールールを作成するために使用されます。サブ要素 **usbdev** を使用して、各フィルタールールを定義します。**class** 属性は USB クラスコードです。

23.17.7. スマートカードデバイス

仮想スマートカードデバイスは、**smartcard** 要素を介してゲスト仮想マシンに提供することができます。ホスト物理マシンにある USB スマートカードリーダーデバイスは、デバイスパススルー機能を持つゲスト仮想マシンでは使用できません。これは、ホスト物理マシンとゲスト仮想マシンの両方で使用可能にすることができず、ゲスト仮想マシンから削除された場合にホスト物理マシンコンピューターをロックすることができるためです。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イスを提示できる特別な仮想デバイスを提供します。これには、ホストの物理マシンから、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作成されたチャンネルから認証情報を取得する方法を説明するいくつかのモードがあります。

管理ツールを使用して、キャラクターデバイス経由で USB デバイスリダイレクトを設定し、ドメイン XML の次のセクションを変更します。

図23.45 デバイス - スマートカードデバイス

```
...
<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>
...
```

smartcard 要素には必須の属性**mode** があります。各モードで、ゲスト仮想マシンは、物理 USB CCID (Chip/Smart Card Interface Device) カードのように動作する USB バス上のデバイスを認識します。

モードの属性は、以下のとおりです。

表23.19 スマートカードモードの要素

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のすべての要求を、NSS を介して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ように中継します。その他の属性やサブ要素は必要ありません。オプションの **address** サブ要素の使用方法は、以下を参照してください。

パラメーター	説明
mode='host-certificates'	このモードでは、スマートカードをホスト物理マシンに接続させることなく、ホスト物理マシンのデータベースに存在する 3 つの NSS 証明書名を指定できます。これらの証明書は、**certutil -d /etc/pki/nssdb -x -t CT,CT,CT -S -s CN=cert1 -n cert1,** コマンドを使用して生成できます。生成される 3 つの証明書名は、それぞれ 3 つの **certificate** サブ要素の内容として提供する必要があります。追加のサブ要素の**database**では、代替ディレクトリー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)。指定しないと **/etc/pki/nssdb** にデフォルトが設定されます。
mode='passthrough'	このモードを使用すると、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、セカンダリーキャラクターデバイスを介してすべての要求をサードパーティープロバイダーにトンネリングできます。サードパーティープロバイダーは、スマートカードと通信するか、3 つの証明書ファイルを使用します。操作のこのモードでは、トンネルのホスト物理マシン側を説明するために、対応しているシリアルデバイスタイプの 1 つに一致する追加の属性**type**が必要になります。**type='tcp'** または **type='spicevmc'** (SPICE グラフィックデバイスのスマートカードチャネルを使用) が一般的です。**target** サブ要素は必要ありませんが、**source** などのサブ要素は、指定のタイプに応じて必要になる場合があります (キャラクターデバイスのコンシューマーは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)。

各モードは、オプションのサブ要素**address**をサポートしています。これは、スマートカードと ccid バスコントローラーとの間の関連を微調整します。詳細は、「デバイスアドレス」 を参照してください。

23.17.8. ネットワークインターフェイス

管理ツールを使用してネットワークインターフェイスデバイスを変更し、ドメイン XML の以下の部分を設定します。

図23.46 デバイス - ネットワークインターフェイス

```
...
<devices>
  <interface type='direct' trustGuestRxFilters='yes'>
    <source dev='eth0'/>
    <mac address='52:54:00:5d:c7:9e'/>
    <boot order='1'/>
    <rom bar='off'/>
  </interface>
</devices>
...
```

ゲスト仮想マシンのネットワークインターフェイスを設定する方法はいくつかあります。これは、interface 要素の type 属性に値を設定することで行われます。以下の値を使用できます。

- **"direct"** - ゲスト仮想マシンの NIC を、ホスト物理マシンの物理 NIC に割り当てます。詳細およびサンプルは、「物理インターフェイスへの直接接続」 を参照してください。
- **"network"** - これは、動的ネットワーク設定またはワイヤレスネットワーク設定のホスト物理マシンにおける一般的なゲスト仮想マシンの接続に推奨される設定です。詳細およびサンプルは、「仮想ネットワーク」 を参照してください。
- **"bridge"** - 静的有線ネットワーク設定を使用するホスト物理マシンのゲスト仮想マシンの接続に推奨される設定です。詳細およびサンプルは、「LAN へのブリッジ」 を参照してください。
- **"ethernet"** - 管理者が任意のスクリプトを実行して、ゲスト仮想マシンのネットワークを LAN に接続する手段を提供します。詳細およびサンプルは、「一般的なイーサネット接続」 を参照してください。
- **"hostdev"** - 汎用デバイスパススルーを使用して、PCI ネットワークデバイスをゲスト仮想マシンに直接割り当てることを許可します。詳細およびサンプルは、「PCI パススルー」 を参照してください。
- **"mcast"** - 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使用できます。詳細およびサンプルは、「マルチキャストトンネル」 を参照してください。
- **"user"** - ユーザーオプションを使用して、ユーザー空間の SLIRP スタックパラメーターを設定すると、外界に NAT を持つ仮想 LAN が提供されます。詳細およびサンプルは、「ユーザー空間の SLIRP スタック」 を参照してください。
- **"server"** - サーバーオプションを使用して TCP クライアントサーバーアーキテクチャーを作成し、1 つのゲスト仮想マシンがネットワークのサーバー側を提供し、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します。詳細およびサンプルは、「TCP トンネル」 を参照してください。

これらのオプションごとに、詳細情報を示すリンクがあります。また、各 **<interface>** 要素はオプションの **<trustGuestRxFilters>** 属性で定義できます。これにより、ホストの物理マシンが、ゲスト仮想マシンから受信したレポートを検出して信頼できるようになります。このようなレポートは、インターフェイスがフィルターへの変更を受信するたびに送信されます。これには、プライマリー MAC アドレス、デバイスアドレスフィルター、または vlan 設定の変更が含まれます。セキュリティー上の理由から、**<trustGuestRxFilters>** 属性はデフォルトで無効になっています。また、この属性のサポートは、ゲストネットワークデバイスモデルや、ホストの物理マシンの接続タイプによって異なります。現在、virtio デバイスモデルと、ホストの物理マシンの macvtap 接続でのみサポートされています。オプショ

ンのパラメーター **<trustGuestRxFilters>** を設定することが推奨される単純なユースケースは、ゲストによって設定されたフィルターもホストでミラーリングされるため、ゲスト仮想マシンにホストの物理マシン側フィルターを制御するパーミッションを付与する場合になります。

上記の属性のほかに、各 **<interface>** 要素にはオプションの **<address>** サブ要素を指定できます。このサブ要素には、属性**type='pci'** を使用して、インターフェイスを特定の PCI スロットに関連付けることができます。詳細は、「デバイスアドレス」 を参照してください。

23.17.8.1. 仮想ネットワーク

これは、動的またはワイヤレスネットワーク設定のホスト物理マシンにおける一般的なゲスト仮想マシンの接続に推奨される設定です (または、ホスト物理マシンのハードウェアの詳細が**<ネットワーク>** 定義で個別に説明されている複数ホストの物理マシン環境)。さらに、名前付きのネットワーク定義で記述されている詳細を含む接続を提供します。仮想ネットワークの**forward mode**設定によっては、ネットワークが完全に分離している (**<forward>** 要素が指定されていない) 場合と、NAT を使用して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接続している (**forward mode='nat'**) 場合、NAT を使用せずにルーティングしている (**forward mode='route'**) 場合、またはホスト物理マシンのネットワークインターフェイス (macvtap を使用) またはブリッジデバイス (**forward mode='bridge|private|vepa|passthrough'**) のいずれかに直接接続している場合があります。

bridge、**private**、**vepa**、および **passthrough** の転送モードを持つネットワークでは、ホストの物理マシンに、必要な DNS サービスおよび DHCP サービスが、libvirt の範囲外に設定されていることを前提とします。分離ネットワーク、nat ネットワーク、ルーティングネットワークの場合、DHCP および DNS は、libvirt により仮想ネットワークで提供され、IP 範囲は、**virsh net-dumpxml [networkname]** で仮想ネットワーク設定を調べることで決定できます。追加設定なしで設定された 'default' 仮想ネットワークは、NAT を使用してデフォルトルートに接続し、IP 範囲は 192.168.122.0/255.255.255.0 になります。各ゲスト仮想マシンには、vnetN という名前で作成された tun デバイスが関連付けられます。これは、**<target>** 要素 (「ターゲット要素の上書き」 を参照) で上書きできます。

インターフェイスのソースがネットワークの場合、ポートグループをネットワークの名前と共に指定できます。1 つのネットワークに複数のポートグループが定義されており、各ポートグループには、ネットワーク接続のさまざまなクラスに対する若干異なる設定情報が含まれています。また、**<direct>** ネットワーク接続 (以下で説明) と同様に、**network** タイプの接続では**<virtualport>**要素を指定できます。設定データは、802.1Qbg または 802.1Qbh 準拠の *Virtual Ethernet Port Aggregator* (VEPA)switch スイッチ、あるいは Open vSwitch 仮想スイッチに転送されます。

スイッチの種類は、ホスト物理マシンの **<network>** 要素の設定に依存するため、**<virtualport type>** 属性を省略することができます。**<virtualport type>**は一度または複数回指定する必要があります。ドメインが起動すると、定義されているタイプと属性をマージして、完全な **<virtualport>** 要素を構築します。これにより、新たに再構築された仮想ポートが作成されます。下位の仮想ポートの属性は、上位の仮想ポートで定義された属性を変更できないことに注意してください。インターフェイスの優先度が最も高く、ポートグループの優先度が最も低くなります。

たとえば、802.1Qbh スイッチおよび Open vSwitch スイッチの両方で適切に機能するネットワークを作成する場合は、タイプを指定せず、**profileid** と **interfaceid** の両方を提供する必要があります。**managerid**、**typeid**、**profileid** などの仮想ポートから入力するその他の属性は任意です。

ゲスト仮想マシンの接続を特定タイプのスイッチのみに制限する場合は、virtualport タイプを指定でき、指定したポートタイプのスイッチのみが接続されます。追加のパラメーターを指定することで、スイッチの接続をさらに制限することもできます。その結果、ポートが指定されており、ホストの物理マシンのネットワークに別のタイプの virtualport があると、インターフェイスの接続に失敗します。仮想ネットワークパラメーターは、ドメイン XML の以下の部分を変更する管理ツールを使用して定義します。

図23.47 デバイス - ネットワークインターフェイス - 仮想ネットワーク

```
...
<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>
...
```

23.17.8.2. LAN へのブリッジ

「ネットワークインターフェイス」 で説明されているように、これは、静的有線ネットワーク設定を使用するホスト物理マシンのゲスト仮想マシンの接続に推奨される設定です。

LAN へのブリッジは、ゲスト仮想マシンから LAN に直接ブリッジを提供します。ホスト物理マシン上にブリッジデバイスがあり、1 つ以上のホスト物理マシンの物理 NIC がスレーブであることを前提としています。ゲスト仮想マシンには、**<vnetN>** の名前で作成された関連する **tun** デバイスがあります。これは **<target>** 要素 (「ターゲット要素の上書き」 を参照) で上書きすることもできます。**<tun>** デバイスは、ブリッジにスレーブされます。IP 範囲またはネットワーク設定は、LAN で使用されるものと同じです。これにより、物理マシンと同様に、ゲスト仮想マシンの完全な着信ネットワークアクセスと発信ネットワークアクセスが提供されます。

Linux システムでは、ブリッジデバイスは通常、標準的な Linux ホストの物理マシンブリッジです。Open vSwitch に対応するホストの物理マシンでは、インターフェイス定義に **virtualport type='openvswitch'/** を追加することで、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**virtualport** は、**parameters** 要素に 2 つのパラメーターを指定できます。1 つは、この特定のインターフェイスを Open vSwitch に固有に識別するために使用される標準の UUID である**interfaceid** (指定しないと、インターフェイスの最初の定義時にランダムな**interfaceid**が生成されます) で、もう 1 つは Open vSwitch にインターフェイス**<port-profile>** として送信されるオプションの**profileid** です。ブリッジを LAN 設定に設定するには、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。

図23.48 デバイス - ネットワークインターフェイス - LAN へのブリッジ

```
...
<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>
```

23.17.8.3. ポートマスカレードの範囲の設定

ポートマスカレードの範囲を設定する場合は、以下のようにポートを設定します。

図23.49 ポートマスカレードの範囲

```
<forward mode='nat'>
   <address start='1.2.3.4' end='1.2.3.10'/>
</forward>  ...
```

この値は、以下のように **iptable** コマンドを使用して設定する必要があります。 「ネットワークアドレス変換」

23.17.8.4. ユーザー空間の SLIRP スタック

ユーザー空間の SLIRP スタックパラメーターを設定すると、外部への NAT を備えた仮想 LAN が提供されます。仮想ネットワークには DHCP サービスおよび DNS サービスがあり、ゲスト仮想マシンに 10.0.2.15 以降の IP アドレスを割り当てます。デフォルトのルーターは 10.0.2.2 で、DNS サーバーは 10.0.2.3 です。このネットワークは、ゲスト仮想マシンに外部アクセスを行う必要がある特権のないユーザーにとって唯一のオプションです。

ユーザー空間の SLIRP スタックパラメーターは、ドメイン XML の以下の部分で定義されています。

図23.50 デバイス - ネットワークインターフェイス - ユーザー空間の SLIRP スタック

```
...
<devices>
  <interface type='user'/>
  ...
  <interface type='user'>
    <mac address="00:11:22:33:44:55"/>
  </interface>
</devices>
...
```

23.17.8.5. 一般的なイーサネット接続

これにより、管理者が任意のスクリプトを実行してゲスト仮想マシンのネットワークを LAN に接続できるようになります。ゲスト仮想マシンには、**vnetN** の名前で作成された**<tun>** デバイスがありますが、**<target>** 要素で上書きすることもできます。**tun** デバイスを作成すると、シェルスクリプトが実行され、必要なホスト物理マシンネットワークインテグレーションが完了します。初期設定では、このスクリプトは **/etc/qemu-ifup** と呼ばれていますが、上書きできます (「ターゲット要素の上書き」 を参照)。

一般的なイーサネット接続パラメーターは、ドメイン XML の以下の部分で定義されています。

図23.51 デバイス - ネットワークインターフェイス - 汎用イーサネット接続

```
...
<devices>
  <interface type='ethernet'/>
  ...
  <interface type='ethernet'>
    <target dev='vnet7'/>
    <script path='/etc/qemu-ifup-mynet'/>
  </interface>
</devices>
...
```

23.17.8.6. 物理インターフェイスへの直接接続

物理インターフェイスが指定されている場合は、これにより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イスに直接接続されます。

これには、Linux macvtap ドライバーが利用可能である必要があります。macvtap デバイスの操作モードは、**mode** 属性値の **vepa** ('Virtual Ethernet Port Aggregator')、 **bridge** または **private** のいずれかを選択できます。デフォルトモードは **vepa** です。

物理インターフェイスへの直接接続を操作するには、ドメイン XML のこのセクションで以下のパラメーターを設定します。

図23.52 デバイス - ネットワークインターフェイス - 物理インターフェイスへの直接接続

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='vepa'/>
  </interface>
</devices>
...
```

各モードにより、パケットの配信は表23.20「物理インターフェイス要素への直接接続」 のように動作します。

表23.20 物理インターフェイス要素への直接接続

要素	説明
vepa	ゲスト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。パケットの送信先が、パケットの送信元と同じホスト物理マシン上のゲスト仮想マシンであるパケットは、VEPA 対応のブリッジによりホスト物理マシンに返されます (現在のブリッジは、通常 VEPA 対応ではありません)。
bridge	宛先が、送信元と同じホストの物理マシンにあるパケットは、ターゲットの macvtap デバイスに直接配信されます。直接配信する場合は、作成元デバイスと宛先デバイスの両方がブリッジモードになっている必要があります。いずれかが **vepa** モードにある場合は、VEPA 対応のブリッジが必要です。
プライベート	すべてのパケットは外部ブリッジに送信されます。また、外部ルーターまたはゲートウェイを介して送信され、そのデバイスがホストの物理マシンに返す場合は、同じホストの物理マシンのターゲット仮想マシンにのみ配信されます。移行元デバイスまたは移行先デバイスのいずれかがプライベートモードの場合は、以下の手順が行われます。
パススルー	この機能は、移行機能を失うことなく、SR-IOV 対応 NIC の仮想機能をゲスト仮想マシンに直接接続します。すべてのパケットは、設定したネットワークデバイスの VF/IF に送信されます。デバイスの機能によっては、追加の前提条件や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 以降が必要です。

直接接続された仮想マシンのネットワークアクセスは、ホストの物理マシンの物理インターフェイスが接続されているハードウェアスイッチにより管理できます。

スイッチが IEEE 802.1Qbg 規格に準拠している場合、インターフェイスには以下のような追加パラメーターを設定できます。virtualport 要素のパラメーターについては、IEEE 802.1Qbg 標準で詳細が説明されています。この値はネットワーク固有のもので、ネットワーク管理者が指定する必要があります。802.1Qbg の用語では、Virtual Station Interface (VSI) は仮想マシンの仮想インターフェイスを表します。

また、IEEE 802.1Qbg では VLAN ID にゼロ以外の値が必要です。

操作可能な追加の要素は、表23.21「物理インターフェイスの追加要素への直接接続」 で説明されています。

表23.21 物理インターフェイスの追加要素への直接接続

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識別します。これは整数値で、値 **0** が予約されます。
typeid	VSI タイプ ID は、ネットワークアクセスを特徴付ける VSI タイプを識別します。VSI の種類は通常、ネットワーク管理者が管理します。これは整数値です。
typeidversion	VSI タイプバージョンでは、複数のバージョンの VSI タイプが許可されます。これは整数値です。
instanceid	VSI インスタンス ID 識別子は、VSI インスタンス (仮想マシンの仮想インターフェイス) が作成されると生成されます。これは、グローバルに一意の識別子です。
profileid	プロファイル ID には、このインターフェイ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データベースにより、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、これらのネットワークパラメーターはこのインターフェイスに適用されます。

ドメイン XML の追加パラメーターには、以下が含まれます。

図23.53 デバイス - ネットワークインターフェイス - 物理インターフェイスへの直接接続追加パラメーター

```
...
<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>
...
```

スイッチが IEEE 802.1Qbh 規格に準拠している場合、インターフェイスには以下のような追加パラメーターを設定できます。この値はネットワーク固有のもので、ネットワーク管理者が指定する必要があります。

ドメイン XML の追加パラメーターには、以下が含まれます。

図23.54 デバイス - ネットワークインターフェイス - 物理インターフェイスへの直接接続、さらなる追加パラメーター

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private'/>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance'/>
    </virtualport>
  </interface>
</devices>
...
```

profileid 属性には、このインターフェイ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データベースにより、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、これらのネットワークパラメーターはこのインターフェイスに適用されます。

23.17.8.7. PCI パススルー

source 要素で指定される PCI ネットワークデバイスは、ジェネリックデバイスパススルーを使用してゲスト仮想マシンに直接割り当てられます。このデバイスの MAC アドレスは、最初にオプションで設定された値に設定され、そのデバイスの MAC アドレスがオプションで指定された **virtualport** 要素を使用して 802.1Qbh 対応スイッチに関連付けられます (**type='direct'** ネットワークデバイスの場合は、上記の仮想ポートの例を参照してください)。標準のシングルポート PCI イーサネットカードドライバー設計の制限により、この方法で割り当てることができるのは SR-IOV (Single Root I/O Virtualization) デバイスのみであることに注意してください。標準のシングルポート 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における "intelligent passthrough" は、標準の **hostdev** デバイスの機能と非常に似ています。相違点は、パススルーデバイスに MAC アドレスと**virtualport**を指定できることです。この機能が不要な場合、SR-IOV に対応していない標準のシングルポート PCI カード、PCIe カード、または USB ネットワークカードがある場合 (そのため、ゲスト仮想マシンドメインに割り当てられた後にリセット中に設定済みの MAC アドレスを失います)、または 0.9.11 よりも古い libvirt バージョンを使用している場合は、標準の**hostdev**定義を使用して、**interface type='hostdev'** の代わりに、ゲスト仮想マシンにデバイスを割り当てます。

図23.55 デバイス - ネットワークインターフェイス - PCI パススルー

```
...
<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>
...
```

23.17.8.8. 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すために使用できます。同じマルチキャストグループ内にネットワークデバイスがあるゲスト仮想マシンは、複数の物理ホストの物理マシンにまたがる場合でも、互いに通信します。このモードは、非特権ユーザーとして使用できます。デフォルトの DNS または DHCP に対応せず、発信ネットワークアクセスもありません。外部への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 2 番目の NIC が必要です。この NIC は、最初の 4 つのネットワークタイプのいずれかに接続し、適切なルーティングを提供します。マルチキャストプロトコルは、**user mode** Linux ゲスト仮想マシンが使用するプロトコルと互換性があります。使用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からのものである必要があることに注意してください。マルチキャストトンネルは、マネジメントツールを使用して**interface type**を操作し、これを **mcast** に設定して作成されます。そして、**mac address** および **source address** を提供します。以下に例を示します。

図23.56 デバイス - ネットワークインターフェイス - マルチキャストトンネル

```
...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
    <source address='230.0.0.1' port='5558'/>
  </interface>
</devices>
...
```

23.17.8.9. 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、あるゲスト仮想マシンがネットワークのサーバー側を提供し、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ているゲスト仮想マシンを介してルーティングされます。このモデルは、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 に対応せず、発信ネットワークアクセスもありません。外部への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 2 番目の NIC が必要です。この NIC は、最初の 4 つのネットワークタイプのいずれかに接続し、適切なルーティングを提供します。TCP トンネルは、マネジメントツールを使用して**interface type**を操作し、これを **mcast** に設定して作成されます。そして、**mac address** および **source address** を提供します。以下に例を示します。

図23.57 デバイス - ネットワークインターフェイス - TCP トンネル

```
...
<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>
...
```

23.17.8.10. NIC ドライバー固有のオプションの設定

NIC によっては、調整可能なドライバー固有のオプションが含まれる場合があります。このオプションは、インターフェイス定義の **driver** サブ要素の属性として設定されます。このようなオプションは、管理ツールを使用して設定し、ドメイン XML の以下のセクションを設定します。

図23.58 デバイス - ネットワークインターフェイス - NIC ドライバー固有のオプションの設定

```
<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>
...
```

virtio の NIC ドライバーでは、以下の属性を使用できます。

表23.22 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** 属性では、使用するバックエンドドライバーの種類が強制されます。値は、**kvm** (ユーザー空間のバックエンド) または **vhost** (カーネルにより vhost モジュールが提供されることが必要なカーネルバックエンド) のいずれかになります。カーネルサポートのない vhost ドライバーを要求しようとする場合は拒否されます。デフォルト設定は、vhost ドライバーが存在する場合は **vhost** になりますが、存在しない場合は警告なしに **kvm** に戻ります。
txmode	送信バッファーが満杯になった場合にパケットの送信を処理する方法を指定します。値は、**iothread** または **timer** のいずれかになります。**iothread** に設定すると、パケット tx は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、**kvm** コマンドラインの "-device" virtio-net-pci オプションに **"tx=bh"** を追加することに変換されます)。**timer** に設定すると、KVM で tx 処理が行われます。現在送信可能な tx データよりも多くの tx データが存在する場合は、KVM が他の処理を行うために移動する前にタイマーが設定されます。タイマーが実行されると、さらなるデータを送信するために別の試みが行われます。この値を変更することは推奨されません。
ioeventfd	インターフェイ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、ハイパーバイザーの判断に任されます。指定できる値は **on** と **off** です。このオプションを有効にすると、KVM は、別のスレッドが I/O を処理している間にゲスト仮想マシンを実行できます。通常、I/O の実行中にシステム CPU の使用率が高くなったゲスト仮想マシンは、この恩恵を受けます。一方、物理ホストマシンのオーバーロードは、ゲスト仮想マシンの I/O レイテンシーを増加させる可能性もあります。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一部の側面を制御します。値は **on** または **off** のいずれかになります。**on** がデフォルトで、ゲスト仮想マシンの割り込みと終了の数を減らします。この動作が最適ではない状況では、この属性を使用すると機能を強制的にオフにできます。この値を変更することは推奨されません。

23.17.8.11. ターゲット要素の上書き

ターゲット要素を上書きする場合は、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。

図23.59 デバイス - ネットワークインターフェイス - ターゲット要素の上書き

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
  </interface>
</devices>
...
```

ターゲットを指定しないと、特定のハイパーバイザーにより、作成された tun デバイスの名前が自動的に生成されます。この名前は手動で指定できますが、**vnet** または **vif** で始まる名前は使用できません。これらの接頭辞は、libvirt および特定のハイパーバイザーが予約するものです。この接頭辞を使用して手動で指定したターゲットは無視されます。

23.17.8.12. 起動順序の指定

起動順序を指定するには、管理ツールを使用して、ドメイン XML に以下の変更を行います。

図23.60 起動順序の指定

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <boot order='1'/>
  </interface>
</devices>
...
```

これに対応するハイパーバイザーでは、ネットワーク起動に使用する特定の NIC を設定できます。属性の順序により、システムの起動シーケンスでデバイスが試行される順序が決定します。デバイスごとのブート要素は、BIOS ブートローダーセクションの一般的なブート要素とは併用できないことに注意してください。

23.17.8.13. インターフェイス 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 に以下の変更を加えます。

図23.61 インターフェイス ROM BIOS 設定

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </interface>
</devices>
...
```

これをサポートするハイパーバイザーの場合は、PCI Network デバイスの ROM がゲスト仮想マシンにどのように提供されるかを変更できます。**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で表示されるかどうかを決定します。(PCI のドキュメントでは、**rom bar** 設定により、ROM のベースアドレスレジスターの存在が制御されます。)**rom bar** を指定しないと、KVM のデフォルトが使用されます (古いバージョンの KVM はデフォルトで **off** を使用し、新しい KVM ハイパーバイザーはデフォルトで **on** を使用します)。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指定するために使用されます。これは、ネットワークデバイスに別のブート ROM を提供する際に役立ちます。

23.17.8.14. QoS (Quality of Service)

送受信トラフィックは、QoS (Quality of Service) を設定するために、独立してシェイプできます。**bandwidth** 要素には、最大 1 つのインバウンドおよび 1 つのアウトバウンド子要素を設定できます。このような子要素のいずれかを外した場合は、トラフィックの方向に QoS が適用されません。そのため、ドメインの着信トラフィックのみを形成する場合はインバウンドのみを使用し、その逆も同様にします。

これらの各要素には、必須の属性**average** (または以下のような **floor**) が 1 つずつあります。**Average** は、シェイプされるインターフェイスの平均ビットレートを指定します。さらに、オプションの属性が 2 つあります。

- **peak** - この属性は、ブリッジがデータを送信できる最大レートをキロバイト (秒) 単位で指定します。この実装の制限は、Linux Ingress フィルターがまだ認識していないため、アウトバウンド要素のこの属性は無視されます。

- **burst**: ピークの速度でバーストできるバイト数を指定します。属性に使用できる値は整数です。

average 属性および **peak** 属性の単位は キロバイト/秒 で、 **burst** はキロバイト単位でのみ設定されます。また、着信トラフィックには**floor** 属性を指定できます。これにより、シェーピングのインターフェイスでは最小限のスループットが保証されます。**floor** を使用する場合は、すべてのトラフィックが QoS の決定が行われる 1 つのポイントを通過する必要があります。このため、**forward**タイプが **route**、**nat**、または no forward at all である **interface type='network'**/ の場合のみに使用できます。仮想ネットワーク内では、接続されているすべてのインターフェイスに、少なくとも着信 QoS セット (**average** 以上) が必要ですが、floor 属性では **average** を指定する必要がありません。ただし、**peak** 属性および **burst** 属性には依然として**average** が必要です。現時点では、Ingress qdiscs にクラスがないため、**floor** は受信トラフィックにのみ適用でき、送信トラフィックには適用されません。

QoS 設定を指定するには、管理ツールを使用して、ドメイン XML に以下の変更を行います。

図23.62 QoS (Quality of Service)

```
...
<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>
...
```

23.17.8.15. VLAN タグの設定 (サポートされているネットワークタイプのみ)

VLAN タグ設定を指定するには、管理ツールを使用して、ドメイン XML に以下の変更を行います。

図23.63 VLAN タグの設定 (サポートされているネットワークタイプのみ)

```
...
<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>
...
```

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 タグ付けに対応している場合、任意の **vlan** 要素でゲスト仮想マシンのネットワークトラフィックに適用する 1 つ以上の VLAN タグを指定できます。OpenvSwitch インターフェイスおよび **type='hostdev'** SR-IOV インターフェイスのみが、ゲスト仮想マシントラフィックの透過的な VLAN タグ付けに対応します。標準的な Linux ブリッジや libvirt 独自の仮想ネットワークを含むその他のインターフェイスでは対応していません。802.1 Qbh (vn-link) スイッチおよび 802.1Qbg (VEPA) スイッチは、(libvirt の外部で) ゲスト仮想マシンのトラフィックを特定の VLAN にタグ付けする独自の方法を提供します。複数のタグを指定できるようにするには (VLAN トランクの場合)、**tag** サブ要素で使用する VLAN タグを指定します (**tag id='42'**/ など)。インターフェイスに複数の **vlan** 要素が定義されている場合は、指定されたすべてのタグを使用して VLAN トランクを実行することが想定されます。タグが 1 つの VLAN トランクが必要な場合は、オプションの属性**trunk='yes'** をトップレベルの **vlan** 要素に追加できます。

23.17.8.16. 仮想リンクの状態の変更

この要素は、仮想ネットワークのリンク状態を設定します。属性 **state** に指定できる値は、**up** および **down** です。**down** が値として指定された場合、インターフェイスはネットワークケーブルが切断されているように動作します。この要素が指定されていない場合のデフォルトの動作は**up** です。

仮想リンク状態の設定を指定するには、管理ツールを使用して、ドメイン XML に以下の変更を行います。

図23.64 仮想リンクの状態の変更

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
    <link state='down'/>
  </interface>
<devices>
...
```

23.17.9. 入力デバイス

入力デバイスを使用すると、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相互作用が可能になります。フレームバッファーを有効にすると、入力デバイスが自動的に提供されます。デバイスを明示的に追加することもできます。たとえば、絶対的なカーソルの動きにグラフィックタブレットを使用することもできます。

入力デバイスの設定を指定するには、管理ツールを使用して、ドメイン XML に以下の変更を行います。

図23.65 入力デバイス

```
...
<devices>
  <input type='mouse' bus='usb'/>
</devices>
...
```

<input> 要素には 1 つの必須属性があります: **type** は **mouse** または **tablet** に設定するすることができます。**tablet** はカーソルの絶対移動を提供しますが、**mouse** は相対移動を使用します。オプションの **bus** 属性を使用すると、デバイスタイプを詳細に指定できます。また、**kvm** (準仮想化)、**ps2**、および **usb** に設定できます。

インプット要素にはオプションのサブ要素**<address>**があります。これは、上記で説明されているように、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。

23.17.10. ハブデバイス

ハブは、1 つのポートを複数の場所に拡張し、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可能なデバイスです。

ハブデバイスの設定を指定するには、管理ツールを使用して、ドメイン XML に以下の変更を行います。

図23.66 ハブデバイス

```
...
<devices>
  <hub type='usb'/>
</devices>
...
```

hub 要素には、**type** という必須属性が 1 つあります。これは、**usb** にのみ設定できます。**hub** 要素には、**type='usb'** をオプションに持つオプションのサブ要素 **address** があり、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。

23.17.11. グラフィカルフレームバッファー

グラフィックデバイスを使用すると、ゲスト仮想マシンのオペレーティングシステムとのグラフィカルな相互作用が可能になります。ゲスト仮想マシンには通常、ユーザーとの対話を許可するようにフレームバッファーまたはテキストコンソールのいずれかが設定されます。

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、管理ツールを使用して、ドメイン XML に次の変更を行います。

図23.67 グラフィカルフレームバッファー

```
...
<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>
...
```

graphics 要素には必須の**type** 属性があります。この属性は、以下の表の説明に従って、**sdl**、**vnc**、**rdp**、**desktop**、または **spice** になります。

表23.23 グラフィカルフレームバッファーのメイン要素

パラメーター	説明

パラメーター	説明
sdl	これにより、ホストの物理マシンデスクトップにウィンドウが表示されます。これには、以下のオプション引数を使用できます。 ● 使用するディスプレイの**display**属性 ● 認証 ID の **xauth** 属性 ● 値 **yes** または **no** を受け付けるオプションの **fullscreen** 属性
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 (**-1** は、自動割り当てが必要であることを示す従来の構文)。 ● **autoport** 属性は、使用する TCP ポートの自動割り当てを指定する際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。**timestamp passwdValidTo='2010-04-09T15:51:00'** が UTC であると想定し、パスワードの有効性に制限を設定することができます。 ● **connected** 属性を使用すると、パスワードの変更時に接続先を制御できます。VNC は **keep** 値のみを受け入れます。すべてのハイパーバイザーでサポートされていないことに注意してください。 ● KVM は、listen/port を使用する代わりに、UNIX ドメインソケットパスをリッスンするソケット属性をサポートします。

パラメーター	説明
spice	SPICE サーバーを起動します。 ● **port** 属性は TCP ポート番号を指定します (自動割り当ての必要があることを示す従来の構文として -1 を使用)。**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指定する際に推奨される新しい構文です。 ● **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 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。**timestamp passwdValidTo='2010-04-09T15:51:00'** が UTC であると想定し、パスワードの有効性に制限を設定することができます。 ● **connected** 属性を使用すると、パスワードの変更時に接続先のクライアントを制御できます。SPICE は、クライアントとの接続を維持するための **keep**、クライアントとの接続を解除する **disconnect**、そしてパスワードの変更に失敗する **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るわけ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ます。有効な値は、**secure**、**insecure**、およびデフォルトの**any** になります (可能な場合は**secure** になりますが、安全なパスが利用できない場合にエラーになるのではなく、**insecure** にフォールバックします)。

SPICE に通常の TCP ポートと、TLS で保護された TCP ポートの両方が設定されている場合は、各ポートで実行できるチャネルを制限することが望ましい場合があります。これを行うには、主な **graphics** 要素内に 1 つ以上の**channel** 要素を追加します。有効なチャンネル名は、**main**、**display**、**inputs**、**cursor**、**playback**、**record**、**smartcard**、および **usbredir** などです。

SPICE 設定を指定するには、管理ツールを使用して、ドメイン XML に以下の変更を行います。

図23.68 SPICE 設定の例

```
<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>
```

SPICE は、オーディオ、イメージ、およびストリーミングの変数圧縮設定に対応しています。この設定は、以下の要素の **compression** 属性を使用して設定されます。

- イメージの圧縮を設定する**image** (**auto_glz**、**auto_lz**、**quic**、**glz**、**lz**、**off** を受け入れます)
- WAN を介したイメージの JPEG 圧縮の**jpeg** (**auto**、**never**、**always** を受け入れます)
- WAN イメージ圧縮の設定用の **zlib** (**auto**、**never**、**always** を受け入れます)、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** を受け入れます)

streaming 要素は、ストリーミングモードを設定します。**mode** 属性は、**filter**、**all**、または **off** に設定できます。

さらに、(SPICE エージェントを介して) コピーアンドペースト機能は、**clipboard** 要素により設定されます。これはデフォルトで有効になっており、**copypaste** プロパティーを **no** に設定することで無効にできます。

mouse 要素は、マウスモードを設定します。**mode** 属性は、**server** または **client** に設定できます。mode を指定しない場合は、KVM のデフォルトが使用されます (**client** モード)。

追加の要素は以下のとおりです。

表23.24 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 (自動割り当ての必要があることを示す従来の構文として -1 を使用)。 • **autoport** 属性は、使用する TCP ポートの自動割り当てを指定する際に推奨される構文です。 • **replaceUser** 属性は、仮想マシンへの同時接続を複数許可するかどうかを決定するブール値です。 • 新規クライアントがシングル接続モードで接続する場合に、**multiUser** 属性で既存の接続をドロップする必要があるか、また、新規の接続を VRDP サーバーで確立する必要があるかを決定します。

パラメーター	説明
desktop	この値は、現在 VirtualBox ドメインに予約されています。**sdl** と同様に、ホスト物理マシンデスクトップ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** と同様に、オプション属性の **display** および **fullscreen** を受け入れます。
listen	グラフィックスタイプ **vnc** および **spice** のリッスンソケットを設定するために使用されるアドレス情報を入力する代わりに、**graphics** の別のサブ要素である **listen** 属性を指定することができます (上記の例を参照)。**listen** は次の属性を受け入れます。 ● **type** - **address** または **network** に設定します。これは、このリッスン要素が、直接使用するアドレスを指定するか、ネットワークに名前を付けるか (これを使用して、リッスンする適切なアドレスを判断します) 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 クエリーで IP アドレスに解決されます) のいずれかが含まれます。実行中のドメインの"live" XML では、この属性は、たとえ**type='network'**であっても、リッスンに使用される IP アドレスに設定されます。 ● **network** - **type='network'** の場合、network 属性には、libvirt の設定済みのネットワークリストにネットワークの名前が含まれます。名前の付いたネットワーク設定を検証し、適切なリッスンアドレスを決定します。たとえば、ネットワークの設定に IPv4 アドレスがある場合 (正引きタイプのルート、NAT、または分離タイプがある場合など) は、ネットワークの設定にリスト表示されている最初の IPv4 アドレスが使用されます。ネットワークがホストの物理マシンブリッジを表している場合は、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ます。ネットワークが 'direct' (macvtap) モードの 1 つを記述している場合は、最初の forward dev の 1 番目の IPv4 アドレスが使用されます。

23.17.12. ビデオデバイス

ビデオデバイスの設定を指定するには、管理ツールを使用して、ドメイン XML に次の変更を行います。

図23.69 ビデオデバイス

```
...
<devices>
  <video>
    <model type='vga' vram='8192' heads='1'>
      <acceleration accel3d='yes' accel2d='yes'/>
    </model>
  </video>
</devices>
...
```

graphics 要素には必須の**type** 属性があり、以下の説明に従って、"sdl"、"vnc"、"rdp"、または "desktop" の値をとります。

表23.25 グラフィカルフレームバッファー要素

パラメーター	説明
video	**video** 要素は、ビデオデバイスを説明するコンテナーです。後方互換性のために、動画が設定されておらず、ドメイン XML に **graphics** 要素がある場合は、ゲスト仮想マシンの種類に応じたデフォルトの **video** が追加されます。ram または vram が指定されていない場合は、デフォルト値が使用されます。
model	これには、利用可能なハイパーバイザー機能に応じて、**vga**、**cirrus**、**vmvga**、**kvm**、**vbox**、または **qxl** の値を取る必須の **type** 属性があります。また、vram と、ヘッドを使用した数値を使用して、ビデオメモリーの量をキビバイト (1024 バイトのブロック) で提供することもできます。
アクセラレーション	アクセラレーションに対応している場合は、**acceleration** 要素の **accel3d** 属性および **accel2d** 属性を使用して有効にする必要があります。
address	オプションの address サブ要素を使用すると、ビデオデバイスを特定の PCI スロットに関連付けることができます。

23.17.13. コンソール、シリアル、およびチャネルデバイス

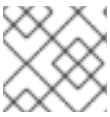
キャラクターデバイスは、仮想マシンと対話する方法を提供します。準仮想化コンソール、シリアルポート、およびチャネルはすべてキャラクターデバイスとして分類され、同じ構文で表されます。

コンソール、チャネル、およびその他のデバイス設定を指定する場合は、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。

図23.70 コンソール、シリアル、およびチャネルデバイス

```
...
<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>
...
```

このディレクティブの各々では、トップレベル要素の名前 (**serial**、**console**、**channel**) でデバイスがゲスト仮想マシンにどのように提示されるかを説明します。ゲスト仮想マシンインターフェイスは、**target** 要素で設定されます。ホスト物理マシンに提示されるインターフェイスは、トップレベル要素の **type** 属性で指定されます。ホスト物理マシンインターフェイスは、source 要素で設定されます。**source** 要素には、ソケットパスでのラベリングの実行方法を上書きするオプションの**seclabel**を指定できます。この要素がない場合は、ドメインごとの設定からセキュリティーラベルが継承されます。各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** があり、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** デバイスに対応しなくなりました。

23.17.14. ゲスト仮想マシンのインターフェイス

キャラクターデバイスは、次のいずれかの形式でゲスト仮想マシンに表示されます。

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。

図23.71 ゲスト仮想マシンインターフェイスのシリアルポート

```
...
<devices>
  <serial type='pty'>
    <source path='/dev/pts/3'/>
    <target port='0'/>
  </serial>
</devices>
...
```

<target> には、ポート番号を指定する **port** 属性を指定できます。ポートには 0 から始まる番号が付けられています。通常、シリアルポートは 0、1、または 2 です。オプションの **type** 属性もあります。この属性には、値として **isa-serial** または **usb-serial** の 2 つの選択肢があります。 **type** がない場合

は、**isa-serial** がデフォルトで使用されます。**usb-serial** の場合、**type='usb'** の任意のサブ要素 **<address>** で、上記の特定のコントローラーにデバイスを関連付けることができます。

<console> 要素は、対話式コンソールを表すために使用されます。以下のルールに従い、使用中のゲスト仮想マシンのタイプによって、コンソールは準仮想化デバイスである場合と、シリアルデバイスのクローンである場合があります。

- **targetType** 属性が設定されていない場合、デフォルトのデバイス**type**はハイパーバイザーのルールに従います。libvirt に渡された XML を再クエリーする際に、デフォルトの**type**が追加されます。完全に仮想化された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、シリアルポートです。

- **targetType** 属性が **serial** で、**<serial>** 要素が存在しない場合は、console 要素が **<serial>** 要素にコピーされます。**<serial>** 要素がすでに存在する場合は、console 要素は無視されます。

- **targetType** 属性が **serial** でない場合は、通常処理されます。

- 最初の **<console>** 要素のみが、**serial** の**targetType**を使用できます。セカンダリー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 要素が **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 コンソールには関連付けられたコントローラーがありません。

以下の例では、virtio コンソールデバイスが **/dev/hvc[0-7]** としてゲスト仮想マシンに公開されます (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)。

図23.72 ゲスト仮想マシンインターフェイス - virtio コンソールデバイス

```
...
<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>
...
```

コンソールがシリアルポートとして提示される場合、**<target>** 要素にはシリアルポートと同じ属性があります。通常、コンソールは 1 つだけです。

23.17.15. Channel

これは、ホストの物理マシンとゲスト仮想マシン間のプライベート通信チャンネルを表します。これは、管理ツールを使用してゲスト仮想マシンに変更を加え、ドメイン XML の以下のセクションを編集することで操作します。

図23.73 Channel

```
...
<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>
...
```

これは、さまざまな方法で実装できます。**<channel>** の特定のタイプは、**<target>** 要素の **type** 属性で指定します。各チャンネルタイプには、以下のように異なるターゲット属性があります。

- **guestfwd** - ゲスト仮想マシンが指定の IP アドレスおよびポートに送信する TCP トラフィックが、ホスト物理マシンのチャネルデバイスに転送されることを指定します。**target** 要素には、address 属性および port 属性が必要です。

- **virtio** - 準仮想化された virtio チャネル。Linux ゲストオペレーティングシステムでは、 **<チャネル>** 設定が **/dev/vport*** ファイルの内容を変更します。オプションの要素 **名** を指定すると、代わりに **/dev/virtio-ports/$name** ファイルが使用されます。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。オプションの要素 **address** では、チャンネルを特定の**type='virtio-serial'** コントローラーに関連付けることができます (上記を参照)。KVM では、name 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静止などの操作で、ゲスト仮想マシンにインストールされているゲストエージェントと対話できます。

- **spicevmc** - 準仮想化 SPICE チャンネル。ドメインには、グラフィックデバイスとしての SPICE サーバーも必要です。この時点で、ホストの物理マシンがメインチャネル全体でメッセージをピギーバックします。属性が **type='virtio';** の **target** 要素が必要です。オプションの属性 **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、デフォルトは **name='com.redhat.spice.0'** になります。オプションの **<address>** 要素は、チャンネルを特定の **type='virtio-serial'** コントローラーに関連付けることができます。

23.17.16. ホストの物理マシンインターフェイス

キャラクターデバイスは、ホストの物理マシンに、次のいずれかの形式で表示されます。

表23.26 キャラクターデバイスの要素

パラメーター	説明	XML スニペット
ドメインログファイル	キャラクターデバイスのすべての入力を無効にし、仮想マシンのログファイルに出力を送信します。	<devices> <console type='stdio'> <target port='1'/> </console> </devices>
デバイスログファイル	ファイルが開かれ、キャラクターデバイスに送信したすべてのデータがファイルに書き込まれます。この設定でゲストを正常に起動するには、移行先ディレクトリーに、ゲストの **virt_log_t** SELinux ラベルが必要です。	<devices> <serial type="file"> <source path="/var/log/vm/vm-serial.log"/> <target port="1"/> </serial> </devices>
仮想コンソール	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、これには "ctrl+alt+3" のような特別なホットキーシーケンスを使用してアクセスします。	<devices> <serial type='vc'> <target port="1"/> </serial> </devices>
Null デバイス	キャラクターデバイスをボイドに接続します。入力にデータが提供されることはありません。書き込まれたデータはすべて破棄されます。	<devices> <serial type='null'> <target port="1"/> </serial> </devices>
擬似 TTY	疑似 TTY は、**/dev/ptmx** を使用して割り当てられます。**virsh console** などの適切なクライアントは、接続してローカルのシリアルポートと対話できます。	<devices> <serial type="pty"> <source path="/dev/pts/3"/> <target port="1"/> </serial> </devices>

パラメーター	説明	XML スニペット
NB 特殊ケース	NB 特殊なケースで、**<console type='pty'>** の場合、TTY パスもトップレベルの **<console>** タグの属性 **tty='/dev/pts/3'** として複製されます。これにより、**<console>** タグの構文が簡略化されました。	
ホスト物理マシンのデバイスプロキシー	キャラクターデバイスは、基本となる物理キャラクターデバイスに渡されます。エミュレート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するなど、デバイスの種類が一致する必要があります。シリアルポートを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>
名前付きパイプ	キャラクターデバイスは、名前付きパイプに出力を書き込みます。詳細は、man ページの pipe(7) を参照してください。	<devices> <serial type="pipe"> <source path="/tmp/mypipe"/> <target port="1"/> </serial> </devices>

パラメーター	説明	XML スニペット
TCP クライアントサーバー	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"/> <target port="1"/> </serial> </devices> または、クライアント接続を待機している 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 の TCP の代わりに telnet を使用できます。また、telnets (セキュアな telnet) および tls も使用できます。 <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 ネットワークコンソール	キャラクターデバイスは UDP netconsole サービスとしてパケットの送受信を行います。これは不可逆のサービスです。	<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 ドメインソケット 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>

23.17.17. サウンドデバイス

仮想サウンドカードは、**sound** 要素を使用してホスト物理マシンに接続できます。

図23.74 仮想サウンドカード

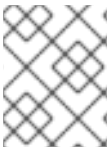
```
...
<devices>
  <sound model='ac97'/>
</devices>
...
```

sound 要素には、必須の属性 **model** が 1 つあります。これは、実際のサウンドデバイスをエミュレートするものを指定します。有効な値は、基本となるハイパーバイザーに固有のものですが、一般的な選択肢は **'sb16'**、**'ac97'**、および **'ich6'** です。また、**'ich6'** モデルセットを持つ **sound** 要素には、さまざまなオーディオコーデックをオーディオデバイスに割り当てるためのオプションの **codec** サブ要素を追加できます。指定しない場合は、再生および録画を許可するために、デフォルトのコーデックが割り当てられます。有効な値は **'duplex'** (ラインインおよびラインアウトをアドバタイズ) および **'micro'** (スピーカーおよびマイクをアドバタイズ) です。

図23.75 サウンドデバイス

```
...
<devices>
  <sound model='ich6'>
    <codec type='micro'/>
  <sound/>
</devices>
...
```

各サウンド要素には、オプションのサブ要素**<address>**があります。このサブ要素は、デバイスを特定の PCI スロットに接続できます (上記を参照)。



注記

es1370 サウンドデバイスは、Red Hat Enterprise Linux 7 ではサポートされなくなりました。代わりに ac97 を使用します。

23.17.18. 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>** 要素を使用して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には、ゲスト仮想マシンに追加のドライバーおよび管理デーモンが必要です。現在、ウォッチドッグの実行時にサポート通知はありません。

図23.76 ウォッチドッグデバイス

```
...
<devices>
  <watchdog model='i6300esb'/>
</devices>
...

...
<devices>
  <watchdog model='i6300esb' action='poweroff'/>
</devices>
...
```

この XML では、以下の属性が宣言されています。

- **model** - 必要な **model** 属性は、実際のウォッチドッグデバイスをエミュレートするものを指定します。有効な値は、基になるハイパーバイザーに固有のものです。
- **model** 属性は、以下の値をとることができます。
  - **i6300esb** - 推奨されるデバイスで、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、ウォッチドッグの期限が切れたときに行うアクションを説明します。有効な値は、基になるハイパーバイザーに固有のものです。**action** 属性には、以下の値を指定できます。

- **reset**: デフォルト設定で、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 シグナルに応答することが必要になります。ウォッチドッグが期限切れとなった状況では、ゲスト仮想マシンは通常、ACPI シグナルに応答できません。したがって、'shutdown' の使用は推奨されません。また、ダンプファイルを保存するディレクトリーは、/etc/libvirt/kvm.conf ファイルの auto_dump_path で設定できます。

23.17.19. パニックデバイスの設定

Red Hat Enterprise Linux 7 ハイパーバイザーは、**pvpanic** メカニズムを使用して Linux ゲスト仮想マシンカーネルパニックを検出できます。**pvpanic** が起動すると、**libvirtd** デーモンにメッセージを送信します。これにより、事前設定された対応が開始します。

pvpanic デバイスを有効にするには、以下を実行します。

- ホストマシンの**/etc/libvirt/qemu.conf** ファイルにある次の行を追加またはコメント解除します。

```
auto_dump_path = "/var/lib/libvirt/qemu/dump"
```

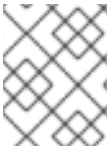
- **virsh edit** コマンドを実行して、指定したゲストのドメイン XML ファイルを編集し、**panic**を**devices** 親要素に追加します。

```
<devices>
  <panic>
    <address type='isa' iobase='0x505'/>
  </panic>
</devices>
```

<address> 要素は、パニックのアドレスを指定します。デフォルトの ioport は 0x505 です。ほとんどの場合、アドレスを指定する必要はありません。

libvirtd がクラッシュに反応する方法は、ドメイン XML の **<on_crash>** 要素により決定します。可能なアクションは以下のとおりです。

- **coredump-destroy** - ゲスト仮想マシンのコアダンプをキャプチャーして、ゲストをシャットダウンします。
- **coredump-restart** - ゲスト仮想マシンのコアダンプをキャプチャーして、ゲストを再起動します。
- **preserve**: ゲスト仮想マシンを停止して、次のアクションを待ちます。



注記

kdump サービスが有効な場合は、**<on_crash>** の設定よりも優先され、選択した **<on_crash>** の動作は実行されません。

pvpanic の詳細は、関連するナレッジベースの記事 を参照してください。

23.17.20. メモリーバルーンデバイス

バルーンデバイスは、仮想マシンの RAM の一部が使用されていないことを指定できます (バルーンの **膨張** として知られているプロセス)。これにより、ホストや、そのホストのその他の仮想マシンが使用するメモリーを解放できます。仮想マシンにメモリーが再度必要になると、バルーンが**収縮** され、ホストは RAM を仮想マシンに分散して戻すことができます。

メモリーバルーンのサイズは、**<currentMemory>** および **<memory>** 設定の違いにより決定します。たとえば、**<memory>** を 2 GiB に設定し、 **<currentMemory>** を 1 GiB に設定すると、バルーンには 1 GiB が含まれます。手動設定が必要な場合は、**virsh setmem** コマンドを使用して**<currentMemory>** 値を設定し、**virsh setmaxmem** コマンドを使用して **<memory>** 値を設定できます。

警告

<currentMemory> の値を変更する場合には、ゲスト OS が適切に機能するように十分なメモリーを残してください。設定値を低くしすぎると、ゲストが不安定になる可能性があります。

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自動的に追加されます。XML 設定では、これは **<memballoon>** 要素により表されます。メモリーバルーニングは **libvirt** サービスにより管理され、必要に応じて自動的に追加されます。そのため、特定の PCI スロットを割り当てる必要がない限り、ゲスト仮想マシン XML にこの要素を明示的に追加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'** を使用できることに留意してください。

以下の例は、**libvirt** が自動的に追加する memballoon デバイスを示しています。

図23.77 メモリーバルーンデバイス

```
...
<devices>
  <memballoon model='virtio'/>
</devices>
...
```

以下の例は、静的 PCI スロット 2 を要求された状態で手動で追加されたデバイスを示しています。

図23.78 手動で追加したメモリーバルーンデバイス

```
...
<devices>
  <memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
  </memballoon>
</devices>
...
```

必要な **model** 属性は、どのようなバルーンデバイスが提供されるかを指定します。有効な値は仮想プラットフォーム固有です。KVM ハイパーバイザーでは、**'virtio'** がデフォルト設定です。

23.18. 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、機能のレベルは異なります。ボリュームの作成を許可するものと、既存のボリュームのみを使用できるものがあります。ボリュームのサイズや配置に制限がある場合があります。

ストレージプールドキュメントのトップレベル要素は **<pool>** です。これには、**dir, fs, netfs, disk, iscsi, logical, scsi, mpath, rbd, sheepdog** または **gluster** の値をとる 1 つの属性 **type** があります。

23.18.1. ストレージプールのメタデータの提供

以下の XML の例は、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、プールが iSCSI ストレージプールです。

図23.79 一般的なメタデータタグ

```
<pool type="iscsi">
  <name>virtimages</name>
  <uuid>3e3fce45-4f53-4fa7-bb32-11f34168b82b</uuid>
  <allocation>10000000</allocation>
  <capacity>50000000</capacity>
  <available>40000000</available>
  ...
</pool>
```

この例で使用される要素は、表23.27「**virt-sysprep**コマンド」 で説明されています。

表23.27 virt-sysprepコマンド

要素	説明
<name>	ホスト物理マシンに固有である必要があるストレージプールの名前を指定します。これは、ストレージプールを定義する際に必須になります。

要素	説明
<uuid>	ストレージプールの識別子を指定します。この識別子はグローバルに一意である必要があります。UUID の提供は任意ですが、ストレージプールの作成時に UUID が提供されていない場合は、UUID が自動的に生成されます。
<allocation>	ストレージプールへのストレージ割り当ての合計を提供します。メタデータのオーバーヘッドにより、これは、すべてのストレージボリュームにわたる割り当て合計より大きくなることがあります。この値はバイト単位で表されます。この要素は読み取り専用であり、値は変更しないでください。
<capacity>	プールの合計ストレージ容量を提供します。デバイスの制約により、ストレージボリュームの容量が不足している可能性があります。この値はバイト単位です。この要素は読み取り専用であり、値は変更しないでください。
<available>	ストレージプールに新しいストレージボリュームを割り当てるための空き領域を提供します。基となるデバイスの制約により、すべての空き領域を 1 つの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。この値はバイト単位です。この要素は読み取り専用であり、値は変更しないでください。

23.18.2. ソース要素

<pool> 要素内には、単一の **<source>** 要素が定義されます (1 つのみ)。**<source>** の子要素は、ストレージプールタイプによって異なります。使用できる XML の例の一部を以下に示します。

図23.80 ソース要素オプション 1

```
  ...
<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23.81 ソース要素オプション 2

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

<source> が使用できる子要素は、表23.28「ソースの子要素のコマンド」 で説明されています。

表23.28 ソースの子要素のコマンド

要素	説明
<device>	ホスト物理マシンデバイスによってバッキングされているストレージプールのソースを提供します (「ストレージプール」 に示すように、**<pool type=>**に基づいています)。バックエンドドライバーによっては複数回繰り返される場合があります。ブロックデバイスノードへの完全修飾パスである 1 つの属性**path**が含まれます。
<dir>	ディレクトリー (**<pool type='dir'>**) がサポートするストレージプールのソースを提供します。またはオプションで、ファイルシステム (**<pool type='gluster'>**) に基づくストレージプール内のサブディレクトリーを選択します。この要素は、(**<pool>**) ごとに 1 回のみ発生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のフルパスである 1 つの属性 (**<path>**) を受け入れます。

要素	説明
<adapter>	SCSI アダプター (**<pool type='scsi'>**) がサポートするストレージプールのソースを提供します。この要素は、(**<pool>**) ごとに 1 回のみ発生する可能性があります。属性名は SCSI アダプター名 (例: "scsi_host1") になります。後方互換のために "host1" は引き続きサポートされていますが、推奨されません。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'** です。省略し、**name** 属性を指定すると、デフォルトの **type='scsi_host'** になります。後方互換性を維持するため、属性の**type**は**type='scsi_host'** アダプターではオプションですが、**type='fc_host'** アダプターでは必須です。**wwnn** (Word Wide Node Name) 属性および **wwpn** (Word Wide Port Name) 属性は、**type='fc_host'** アダプターにより使用され、ファイバーチャネルストレージファブリックのデバイスを一意に識別します (デバイスは HBA または vHBA のいずれかです)。**wnn** と **wpn** の両方を指定する必要があります。(v)HBA の wnn/wwpn を取得する方法は、「デバイス設定の収集」 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。
<ホスト>	リモートサーバー (**type='netfs'|'iscsi'|'rbd'|'sheepdog'|'gluster'**) からのストレージによってバッキングされるストレージプールのソースを提供します。この要素は、**<directory>** 要素または **<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。必要に応じて、プロトコル固有のポート番号の **port** 属性を含めることができます。
<auth>	存在する場合は、**<auth>** 要素は、**type** 属性 (プール **type='iscsi'|'rbd'**) の設定により、ソースへのアクセスに必要な認証情報を提供します。**type**は、**type='chap'** または **type='ceph'** 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 "ceph" を使用し、iSCSI ターゲットの CHAP (Challenge-Handshake Authentication Protocol) に "iscsi" を使用します。また、必須属性**username**は、認証時に使用するユーザー名と、必須属性タイプのサブ要素の secret を識別して、実際のパスワードまたはその他の認証情報を保持する libvirt 秘密オブジェクトに戻します。ドメインの XML は、パスワードを意図的に公開せず、パスワードを管理するオブジェクトへの参照のみを公開します。secret 要素は、秘密オブジェクトの UUID を持つ uuid 属性、または secret オブジェクトで指定された鍵に一致する usage 属性のいずれかを必要とします。

要素	説明
<name>	指定した名前の要素**<type>**から、ストレージデバイスによりバッキングされるストレージプールのソースを提供します。これは、(**type='logical'|'rbd'|'sheepdog','gluster'**) の値をとることができます。
<format>	ストレージプール**<type>**の形式に関する情報を提供します。この形式は、**type='logical'|'disk'|'fs'|'netfs'** の値を使用できます。この値はバックエンド固有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タイプ、パーティションテーブルタイプ、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にはデフォルト値が必要なため、この要素はオプションとなります。
<vendor>	ストレージデバイスのベンダーに関するオプション情報を提供します。これには、バックエンド固有の値を持つ1つの属性**<name>**が含まれます。
<product>	ストレージデバイスの製品名に関するオプション情報を提供します。これには、バックエンド固有の値を持つ1つの属性**<name>**が含まれます。

23.18.3. ターゲット要素の作成

以下のタイプ (**type='dir'|'fs'|'netfs'|'logical'|'disk'|'iscsi'|'scsi'|'mpath'**) のトップレベル内の **<pool>** 要素には、1つの **<target>** 要素が含まれます。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説明するために使用されます。以下の子要素を含めることができます。

図23.82 ターゲット要素 XML の例

```
<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>
```

表 (表23.29「ターゲットの子要素」) は、親 **<target>** 要素に有効な子要素を説明しています。

表23.29 ターゲットの子要素

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ピングされる場所を指定します。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、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/**は論理的な選択のように見えるかもしれませんが、デバイスのノードはオンデマンドで割り当てられるため、再起動後も安定しているとは限りません。**/dev/disk/by-{path,id,uuid,label}** の場所など、安定した場所を使用することが推奨されます。

要素	説明
<permissions>	現在、これはディレクトリーベースまたはファイルシステムベースのストレージプールでのみ役に立ちます。これは、ディレクトリーとしてローカルファイルシステムの名前空間にマッピングされます。これは、ストレージプールの構築時に、最終ディレクトリーに使用するパーミッションに関する情報を提供します。**<mode>** 要素には、8 進数のパーミッションセットが含まれます。**<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 (SELinux など) ラベル文字列が含まれます。
<timestamps>	ストレージボリュームのタイミング情報を提供します。最大 4 つのサブ要素が含まれます。**timestamps='atime'|'btime|'ctime'|'mtime'** には、保存ボリュームのアクセス時間、作成時間、変更時間、および修正時間 (既知の場合) が格納されます。使用される時間形式は **<seconds>** です。エポックの開始 (1970 年 1 月 1 日) からの**<nanoseconds>**。ナノ秒の解像度が 0 であるか、ホストのオペレーティングシステムまたはファイルシステムで対応していない場合は、ナノ秒の部分が省略されます。これは読み取り専用属性で、ストレージボリュームの作成時には無視されます。
<暗号化 (encryption)>	存在する場合は、ストレージボリュームの暗号化方法を指定します。詳細は、libvirt upstream pages を参照してください。

23.18.4. デバイスエクステントの設定

ストレージプールで、基本となる配置または割り当て方法が公開される場合は、**<source>** 要素の **<device>** 要素に、利用可能なエクステントに関する情報が含まれることがあります。ストレージプールによっては、ストレージボリュームを 1 つの制約 (ディスクパーティションプールなど) 内で完全に割り当てる必要があるという制約があります。したがって、エクステント情報を使用すると、アプリケーションが新しいストレージボリュームに可能な最大サイズを判断できます。

エクステント情報に対応するストレージプールの場合、各 **<device>** 要素には、**<freeExtent>** 要素が 0 個以上含まれます。これらの各要素には、デバイス上のエクステントの境界を提供する 2 つの属性 (**<start>** および **<end>**) が含まれます (バイト単位)。

23.19. ストレージボリューム

ストレージボリュームは通常、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 以降では、出力専用のオプションの属性タイプは実際のタイプ (ファイル、ブロック、dir、ネットワーク、または netdir) のリストを表示するようになりました。

23.19.1. 一般的なメタデータ

<volume> 要素の最上部には、この XML の例に示すように、メタデータと呼ばれる情報が含まれています。

図23.83 ストレージボリュームの一般的なメタデータ

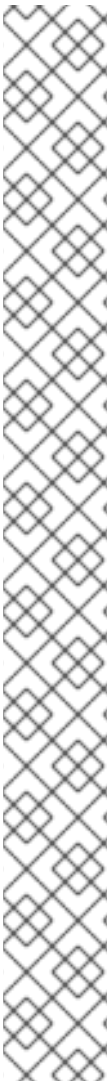
```
...
<volume type='file'>
  <name>sparse.img</name>
  <key>/var/lib/libvirt/images/sparse.img</key>
  <allocation>0</allocation>
  <capacity unit="T">1</capacity>
  ...
</volume>
```

この表 (表23.30「ボリュームの子要素」) は、親 **<volume>** 要素に有効な子要素を説明します。

表23.30 ボリュームの子要素

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。ストレージボリュームを定義する場合に必須です。
<鍵 (key)>	1 つの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、1 つのストレージボリュームを識別する 2 つの異なる鍵を持つことができます。ストレージボリュームは常に生成されるため、このフィールドは、ストレージボリュームの作成時には設定できません。
<allocation>	ストレージボリュームへのストレージの合計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ると、論理容量よりも小さい可能性があります。ストレージボリュームに、メタデータのオーバーヘッドが大量にある場合は、論理容量よりも大きくなることもあります。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した場合は、作成時にストレージボリュームが完全に割り当てられます。容量より小さい値に設定すると、ストレージプールでストレージボリュームのスパース割り当てを選択できるようになります。ストレージプールのタイプによって、スパースストレージボリュームの扱い方が異なります。たとえば、論理プールが満杯になっても、ストレージボリュームの割り当ては自動的に拡張されません。ユーザーは、ストレージボリュームを設定したり、自動的に割り当てるように **dmeventd** を設定したりします。デフォルトでは、これはバイト単位で指定されます。注記 を参照

要素	説明
<capacity>	ストレージボリュームの論理容量を提供します。この値は、デフォルトではバイト単位ですが、**<unit>** 属性は、注記 で説明されている **<allocation>** と同じセマンティックで指定できます。ストレージボリュームを作成する場合は必須です。
<比較元>	ストレージボリュームの基本的なストレージ割り当てに関する情報を提供します。ストレージプールのタイプによっては、利用できない場合があります。
<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現に関する情報を提供します。



注記

必要に応じて、オプションの属性**unit** を指定して、渡される値を調整できます。この属性は、**<allocation>** 要素および **<capacity>** 要素で使用できます。属性 **unit** に指定できる値は、以下のとおりです。

- **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** (ピビバイト)
- **EB** (エクサバイト)
- **E** または **EiB** (エクスビバイト)

23.19.2. target 要素の設定

<target> 要素は、**<volume>** のトップレベル要素に配置できます。これは、ストレージボリュームで、ホスト物理マシンのファイルシステムにマッピングする方法を説明するために使用されます。この要素は、以下の子要素をとることができます。

図23.84 ターゲットの子要素

```
<target>
  <path>/var/lib/libvirt/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>
```

<target> の具体的な子要素は、表23.31「ターゲットの子要素」 で説明されています。

表23.31 ターゲットの子要素

要素	説明
<path>	ローカルファイルシステムでストレージボリュームにアクセスできる場所を、絶対パスで指定します。これは読み取り専用属性であるため、ボリュームの作成時に指定することはできません。
<format>	プール固有のボリューム形式に関する情報を提供します。ディスクベースのストレージプールの場合は、パーティションタイプが提供されます。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、ファイル形式の種類 (cow、qcow、vmdk、raw など) が提供されます。ストレージボリュームの作成時に省略した場合は、ストレージプールのデフォルトの形式が使用されます。実際の形式は、**type** 属性で指定されます。有効な値のリストは、「ストレージプールの使用」 にある特定のストレージプールのセクションを参照してください。

要素	説明
<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パーミッションに関する情報を提供します。これは現在、割り当てられた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に対してのみ、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パーミッションを決定します。子要素が 4 つ含まれています。**<mode>** 要素には、8 進数のパーミッションセットが含まれます。**<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 (SELinux など) ラベル文字列が含まれます。
<compat>	互換性レベルを指定します。これまでは、**<type='qcow2'>** ボリュームにのみ使用されていました。有効な値は、qcow2 (バージョン 2) では **<compat>**0.10**</compat>** で、イメージが互換性がある QEMU バージョンを指定するための qcow2 (バージョン 3) では **<compat>**1.1**</compat>** です。**<feature>** 要素が存在する場合は、**<compat>**1.1**</compat>** が使用されます。省略した場合は、qemu-img のデフォルトが使用されます。
<features>	形式固有の機能。現在、**<format type='qcow2'/>** (バージョン 3) でのみ使用されています。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やフラッシュの量が減るため、初期書き込みパフォーマンスが改善されます。この改善は、特にライトスルーキャッシュモードで見られますが、クラッシュ後にイメージを修復する必要があり、参照カウンターの更新に時間がかかります。この機能は実装されると速くなるため、qcow2 (バージョン 3) でこの機能を使用することが推奨されます。

23.19.3. バッキングストア要素の設定

トップレベルの **<volume>** 要素には、1 つの **<backingStore>** 要素が含まれます。このタグは、ストレージボリュームのオプションの copy-on-write バッキングストアを説明するために使用されます。以下の子要素を含めることができます。

図23.85 バッキングストアの子要素

```
<backingStore>
  <path>/var/lib/libvirt/images/master.img</path>
  <format type='raw'/>
  <permissions>
    <owner>107</owner>
    <group>107</group>
    <mode>0744</mode>
    <label>virt_image_t</label>
  </permissions>
</backingStore>
```

表23.32 バッキングストアの子要素

要素	説明
<path>	ローカルファイルシステムでバッキングストアにアクセスできる場所を、絶対パスで指定します。省略した場合は、このストレージボリュームにバッキングストアはありません。
<format>	プール固有のバッキングストア形式に関する情報を提供します。ディスクベースのストレージプールの場合は、パーティションタイプが提供されます。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、ファイル形式の種類 (cow、qcow、vmdk、raw など) が提供されます。実際の出力形式は、**<type>** 属性で指定します。有効な値のリストは、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では、同じ形式のバッキングストアが必要ですが、qcow2 形式では、異なるバッキングストア形式が使用できます。
<permissions>	バッキングファイルのパーミッションに関する情報を提供します。子要素が 4 つ含まれています。**<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 (SELinux など) ラベル文字列が含まれます。

23.20. セキュリティーラベル

<seclabel> 要素により、セキュリティードライバーの動作を制御できます。基本的な動作モードは、**'dynamic'** で、libvirt が一意のセキュリティーラベルを自動的に生成します。**'static'** では、アプリケーションまたは管理者がラベルを選択します。**'none'** では、制限が無効になっています。動的ラベル生成では、libvirt は、仮想マシンに関連付けられたリソースに常に自動的に再ラベル付けします。静的ラベルの割り当てでは、デフォルトで管理者またはアプリケーションがすべてのリソースでラベルを正しく設定する必要がありますが、必要に応じて自動再ラベル付けを有効にすることができます。

libvirt が複数のセキュリティードライバーを使用する場合は、複数の seclabel タグを使用できます。各ドライバーに 1 つずつ、および各タグが参照するセキュリティードライバーを、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、以下のとおりです。

図23.86 セキュリティーラベル

```
<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'/>
```

入力 XML に**'type'** 属性が指定されていない場合は、セキュリティードライバーのデフォルト設定 (**'none'** または **'dynamic'**) が使用されます。**<baselabel>** を設定し、**'type'** を設定しないと、型が **'dynamic'** であると想定されます。自動リソース再ラベル付けが有効な実行中のゲスト仮想マシンの XML を表示する場合は、追加の XML 要素 **imagelabel** が含まれます。これは出力専用の要素であるため、ユーザー提供の XML ドキュメントでは無視されます。

以下の要素は、以下の値で操作できます。

- **type** - libvirt が一意のセキュリティーラベルを自動的に生成するかどうかを判断するために、**static**、**dynamic**、または **none** を指定します。
- **model** - 有効なセキュリティーモデル名で、現在アクティブなセキュリティーモデルと一致します。
- **relabel** - **yes** または **no** のいずれかです。動的ラベル割り当てを使用する場合は、これを常に**yes** にする必要があります。静的ラベル割り当てでは、デフォルトは **no** になります。
- **<label>** - 静的ラベリングを使用する場合は、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必要があります。コンテンツの形式は、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 のコンテキストです。
  - **AppArmor** - AppArmor プロファイル。
  - **DAC** - 所有者とグループをコロンで区切ります。ユーザー/グループ名または UID/GID の両方として定義できます。ドライバーはまずこれらの値を名前として解析しようとしますが、先頭にあるプラス記号を使用すると、ドライバーが値を UID または GID として解析するように強制できます。
- **<baselabel>** - 動的ラベル付けを使用する場合は、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。コンテンツの形式は、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>** - 出力専用の要素です。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示します。コンテンツの形式は、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。再ラベル付けが有効な場合は、ラベル付けを無効にする (ファイルが NFS またはセキュリティーラベル付けのない他のファイルシステムに存在する場合に役立ちます)、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ですべてではなく一部のリソースを共有できるように特殊なラベルを作成する場合に役立ちます) ことで、特定のソースファイル名に対して行われるラベル付けを微調整することもできます。seclabel 要素がトップレベルドメイン割り当てではなく特定のパスに割り当てられている場合は、属性の relabel または sub-element ラベルのみがサポートされます。

23.21. 仮想マシンの XML 設定例

以下の表は、ゲスト仮想マシン (VM) の XML 設定例 (*ドメイン XML* とも呼ばれる) を示しています。また、設定の内容についても説明しています。

仮想マシンの XML 設定を取得するには、**virsh dumpxml** コマンドを使用します。仮想マシン設定の編集に関する詳細は、仮想化スタートガイド を参照してください。

表23.33 ドメイン XML 設定の例

ドメイン XML セクション	説明
<domain type='kvm'> <name>Testguest1</name> <uuid>ec6fbaa1-3eb4-49da-bf61-bb02fbec4967</uuid> <memory unit='KiB'>1048576</memory> <currentMemory unit='KiB'>1048576</currentMemory> <vcpu placement='static'>1</vcpu>	これは、1024 MiB の割り当て RAM を持つ Testguest1 と呼ばれる KVM です。仮想マシンの全般パラメーターの設定方法は、「一般情報およびメタデータ」 を参照してください。
<vcpu placement='static'>1</vcpu>	ゲスト仮想マシンに 1 つの vCPU が割り当てられています。CPU 割り当ての詳細は、「CPU の割り当て」 を参照してください。
<os> <type arch='x86_64' machine='pc-i440fx-2.9'>hvm</type> <boot dev='hd'/> </os>	マシンアーキテクチャーは AMD64 および Intel 64 のアーキテクチャーに設定され、Intel 440FX マシンタイプを使用して機能の互換性を判断します。OS はハードドライブから起動します。オペレーティングシステムのパラメーターの変更方法は、「オペレーティングシステムの起動」 を参照してください。

ドメイン XML セクション	説明
<features> <acpi/> <apic/> <vmport state='off'/> </features>	ハイパーバイザー機能の **acpi** および **apic** が無効になり、VMware IO ポートが無効になります。ハイパーバイザー機能の変更方法は、「ハイパーバイザーの機能」 を参照してください。
<cpu mode='host-passthrough' check='none'/>	ゲスト CPU の機能は、ホスト CPU の機能と同じに設定されます。CPU 機能の変更の詳細は、「CPU モデルとトポロジー」 を参照してください。
<clock offset='utc'> <timer name='rtc' tickpolicy='catchup'/> <timer name='pit' tickpolicy='delay'/> <timer name='hpet' present='no'/> </clock>	ゲストの仮想ハードウェアクロックは UTC タイムゾーンを使用します。また、QEMU ハイパーバイザーと同期するために、異なるタイマーが 3 つ設定されます。時間管理設定の変更の詳細は、「時間管理」 を参照してください。
<on_poweroff>destroy</on_poweroff> <on_reboot>restart</on_reboot> <on_crash>destroy</on_crash>	仮想マシンの電源がオフの場合や、その OS が予期せず終了すると、libvirt がゲストを終了して、割り当てられたリソースをすべて解放します。ゲストは、再起動すると同じ設定で再起動します。この設定の詳細は、「イベントの設定」 を参照してください。
<pm> <suspend-to-mem enabled='no'/> <suspend-to-disk enabled='no'/> </pm>	このゲスト仮想マシンの S3 および S4 の ACPI スリープ状態は無効化されています。" />.

ドメイン XML セクション	説明
<devices> <emulator>/usr/bin/qemu-kvm</emulator> <disk type='file' device='disk'> <driver name='qemu' type='qcow2'/> <source file='/var/lib/libvirt/images/Testguest.qcow2'/> <target dev='hda' bus='ide'/> <address type='drive' controller='0' bus='0' target='0' unit='0'/> </disk> <disk type='file' device='cdrom'> <driver name='qemu' type='raw'/> <target dev='hdb' bus='ide'/> <readonly/> <address type='drive' controller='0' bus='0' target='0' unit='1'/> </disk>	仮想マシンは、エミュレーションに **/usr/bin/qemu-kvm** バイナリーファイルを使用します。また、2 つのディスクが割り当てられています。最初のディスクは、ホストに格納されている **/var/lib/libvirt/images/Testguest.qcow2** をベースにした仮想ハードドライブで、このドライブの論理デバイス名は **hda** に設定されています。ディスクの管理の詳細は、「ハードドライブ、フロッピーディスク、および CD-ROM」 を参照してください。
<controller type='usb' index='0' model='ich9-ehci1'> <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x7'/> </controller> <controller type='usb' index='0' model='ich9-uhci1'> <master startport='0'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0' multifunction='on'/> </controller> <controller type='usb' index='0' model='ich9-uhci2'> <master startport='2'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x1'/> </controller> <controller type='usb' index='0' model='ich9-uhci3'> <master startport='4'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x2'/> </controller> <controller type='pci' index='0' model='pci-root'/> <controller type='ide' index='0'> <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/> </controller> <controller type='virtio-serial' index='0'> <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/> </controller>	仮想マシンは、USB デバイスの接続に 4 つのコントローラーを使用し、PCIe (PCI-Express) デバイスには root コントローラーを使用します。さらに、**virtio-serial** コントローラーが利用できるため、仮想マシンは、シリアルコンソールなど、各種方法でホストを操作できます。コントローラーの設定の詳細は、「コントローラー」 を参照してください。

ドメイン XML セクション	説明
<interface type='network'> <mac address='52:54:00:65:29:21'/> <source network='default'/> <model type='rtl8139'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/> </interface>	ネットワークインターフェイスは、**default** の仮想ネットワークおよび **rtl8139** ネットワークデバイスモデルを使用する仮想マシンにセットアップされます。ネットワークインターフェイスの設定の詳細は、「ネットワークインターフェイス」 を参照してください。
<serial type='pty'> <target port='0'/> </serial> <console type='pty'> <target type='serial' port='0'/> </console> <channel type='spicevmc'> <target type='virtio' name='com.redhat.spice.0'/> <address type='virtio-serial' controller='0' bus='0' port='1'/> </channel>	仮想マシンには**pty** シリアルコンソールが設定されており、これにより、ホストと最も基本的な仮想マシンの通信が可能になります。準仮想化**SPICE** チャンネルを使用します。この設定は自動で設定されており、設定の変更は推奨されません。キャラクターデバイスの概要は、「ネットワークインターフェイス」 を参照してください。**serial ports** および **consoles** の詳細は、「ゲスト仮想マシンのインターフェイス」 を参照してください。**channels** の詳細は、「Channel」 を参照してください。
<input type='mouse' bus='ps2'/> <input type='keyboard' bus='ps2'/>	仮想マシンは、マウスおよびキーボードを受信するように設定されている仮想**ps2** ポートを使用します。この設定は自動で設定されており、設定の変更は推奨されません。詳細は、「入力デバイス」 を参照してください。
<graphics type='spice' autoport='yes'> <listen type='address'/> <image compression='off'/> </graphics>	仮想マシンは、**SPICE** プロトコルを使用して、自動割り当てポート番号とイメージ圧縮が無効になっているグラフィカル出力をレンダリングします。グラフィックデバイスの設定方法は、「グラフィカルフレームバッファー」 を参照してください。

ドメイン XML セクション	説明
<sound model='ich6'> <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/> </sound> <video> <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1' primary='yes'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/> </video>	**ICH6** HDA サウンドデバイスが仮想マシン用にセットアップされ、QEMU **QXL** 準仮想化フレームバッファーデバイスがビデオアクセラレーターとしてセットアップされます。この設定は自動で設定されており、設定の変更は推奨されません。**sound devices** の設定方法は、「サウンドデバイス」 を参照してください。**video devices** の設定は、「ビデオデバイス」 を参照してください。
<redirdev bus='usb' type='spicevmc'> <address type='usb' bus='0' port='1'/> </redirdev> <redirdev bus='usb' type='spicevmc'> <address type='usb' bus='0' port='2'/> </redirdev> <memballoon model='virtio'> <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/> </memballoon> </devices> </domain>	仮想マシンには、USB デバイスをリモートで接続するためのリダイレクトが 2 つあり、メモリーバルーニング がオンになっています。この設定は自動で設定されており、設定の変更は推奨されません。詳細は、「リダイレクトされたデバイス」 を参照してください。

パート III. 付録

ここでは、仮想化の問題の一般的な問題と解決策を説明します。また、複数のアーキテクチャーで KVM 仮想化を使用する方法と、IOMMU グループを使用する方法を説明します。この部分には、仮想化パッケージの追加サポートおよび製品の制限に関する情報も含まれます。

付録A トラブルシューティング

本章では、Red Hat Enterprise Linux 7 の仮想化に関する一般的な問題と解決策を説明します。

この章を読むことで、仮想化テクノロジーに関連する一般的な問題の一部を理解することができます。トラブルシューティングスキルを向上させるために、Red Hat Enterprise Linux 7 で仮想化を実験してテストすることが推奨されます。

本書で回答が見つからない場合は、オンライン上の仮想化コミュニティーに回答があるかもしれません。Linux 仮想化 Web サイトのリストは、「オンラインリソース」 を参照してください。

また、Red Hat ナレッジベース の RHEL 7 で仮想化のトラブルシューティングに関する詳細も記載されています。

A.1. デバッグおよびトラブルシューティングツール

本セクションでは、システム管理者アプリケーション、ネットワークユーティリティー、およびデバッグツールの概要を説明します。標準のシステム管理ツールおよびログを使用して、トラブルシューティングを支援できます。

- **kvm_stat** - KVM ランタイム統計を取得します。詳細は、「kvm_stat」 を参照してください。
- **ftrace** - カーネルイベントを追跡します。詳細は、『What is ftrace and how do I use it? 』 のソリューション記事 (サブスクリプションが必要) を参照してください。
- **vmstat** - 仮想メモリーの使用状況を表示します。詳細は、**man vmstat** コマンドを使用します。
- **iostat** - I/O 負荷統計を提供します。詳細は、Red Hat Enterprise Linux Performance Tuning Guide を参照してください。
- **lsof** - 開いているファイルのリストを表示します。詳細は、**man lsof** コマンドを使用します。
- **systemtap** - オペレーティングシステムを監視するスクリプトユーティリティーです。詳細は、Red Hat Enterprise Linux Developer Guide を参照してください。
- **crash** - カーネルクラッシュダンプデータまたはライブシステムを分析します。詳細は、Red Hat Enterprise Linux Kernel Crash Dump Guide を参照してください。
- **sysrq** - コンソールが応答しなくても、カーネルが応答する鍵の組み合わせです。詳細は Red Hat ナレッジベース を参照してください。

これらのネットワークユーティリティーは、以下の仮想化ネットワークの問題のトラブルシューティングに役立ちます。

- **ip addr**、**ip route**、および **ip monitor**。
- **tcpdump** - ネットワーク上のパケットトラフィックを診断します。このコマンドは、ネットワーク異常やネットワーク認証の問題を見つける際に役立ちます。**tcpdump** のグラフィカルバージョンでは、**wireshark** という名前のグラフィカルバージョンがあります。
- **brctl** - Linux カーネル内のイーサネットブリッジ設定を検証して設定するネットワークユーティリティーです。以下に例を示します。

```
$ 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
```

仮想化のトラブルシューティングに役立つコマンドを以下に示します。

- **strace** は、システムコールと、別のプロセスが受信して使用した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で実行している VNC サーバーに接続します。 **yum install tigervnc** を使用して **vncviewer** をインストールします。
- **vncserver** は、サーバーでリモートデスクトップを起動します。リモートセッションを使用して、virt-manager などのグラフィカルユーザーインターフェイスを実行できます。**yum install tigervnc-server** を使用して **vncserver** をインストールします。

上記のコマンドのほかに、仮想化ログを調べると便利です。詳細は、「仮想化ログ」 を参照してください。

A.2. ダンプファイルの作成

ゲスト仮想マシンのコアのダンプをファイルに要求できるため、**crash**ユーティリティー などにより仮想マシンのエラーを診断できます。

警告

Red Hat Enterprise Linux 7.5 以降では、Kernel Address Space Randomization (KASLR) 機能により、ゲストダンプファイルが **crash** で読み込みできなくなります。これを修正するには、ゲストの XML 設定ファイルの **<features>** セクションに **<vmcoreinfo/>** 要素を追加します。

宛先のホストが **<vmcoreinfo/>** をサポートしない OS を使用している場合は、**<vmcoreinfo/>** を使用したゲストの 移行 に失敗する点に留意してください。これには、Red Hat Enterprise Linux 7.4 以前のバージョンと、Red Hat Enterprise Linux 6.9 以前のバージョンが該当します。

A.2.1. virsh ダンプファイルの作成

virsh dump コマンドを実行すると、ゲスト仮想マシンのコアをファイルにダンプする要求が送信され、仮想マシンのエラーを診断できます。このコマンドを実行すると、引数 **corefilepath** で指定したファイルおよびパスに対する適切なパーミッションを手動で確認する必要があります。**virsh dump** コマンドは、コアダンプ (または **crash** ユーティリティー) と似ています。

詳細は、Creating a Dump File of a Guest Virtual Machine's Core を参照してください。

A.2.2. Python スクリプトを使用したコアダンプの保存

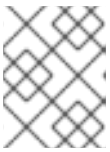
dump-guest-memory.py python スクリプトは、GNU デバッガー (GDB) 拡張機能を実装します。この拡張機能は、**qemu-kvm** プロセスがホストでクラッシュした後、コアダンプからゲスト仮想マシンのメモリーを抽出して保存します。ホスト側の QEMU プロセスがクラッシュすることがゲストのアクションに関連している場合は、QEMU プロセスがクラッシュしたときのゲストの状態を調べると役に立つ場合があります。

python スクリプトは、GDB 拡張機能を実装します。これは、GDB の新しいコマンドです。GDB で、元の (クラッシュした) QEMU プロセスのコアダンプファイルを開いた後、python スクリプトを GDB に読み込むことができます。その後、GDB プロンプトから新しいコマンドを実行できます。これにより、QEMU コアダンプトからゲストメモリーダンプを抽出し、新しいローカルファイルを作成します。

dump-guest-memory.py python スクリプトを使用するには、以下を実行します。

1. qemu-kvm-debuginfo パッケージをインストールします。
2. GDB を起動し、クラッシュした**/usr/libexec/qemu-kvm** バイナリー用に保存されているコアダンプファイルを開きます。デバッグシンボルは自動的に読み込まれます。
3. GDB で新しいコマンドを読み込みます。

```
# source /usr/share/qemu-kvm/dump-guest-memory.py
```



注記

python スクリプトを読み込んだ後、組み込みの GDB **help** コマンドを使用すると、**dump-guest-memory** 拡張機能の詳細を確認できます。

4. GDB でコマンドを実行します。以下に例を示します。

```
# dump-guest-memory /home/user/extracted-vmcore X86_64
```

5. ゲストカーネル解析用に **crash** ユーティリティーで **/home/user/extracted-vmcore** を開きます。

crash ユーティリティーで使用する QEMU コアファイルからゲスト仮想マシンコアを抽出する方法は、『How to extract ELF cores from 'gcore' generated qemu core files for use with the 'crash' utility 』を参照してください。

A.3. SYSTEMTAP FLIGHT RECORDER を使用した定常ベースでのトレースデータのキャプチャー

qemu-kvm パッケージで提供される systemtap initscript を使用して、QEMU トレースデータを常に

キャプチャーできます。このパッケージでは、SystemTap のフライトレコーダーモードを使用して、実行中のゲスト仮想マシンをすべて追跡し、結果をホストの固定サイズのバッファーに保存します。古いトレースエントリーは、バッファーが満杯になると新しいエントリーにより上書きされます。

手順A.1 systemtap の設定および実行

1. **パッケージのインストール**
   以下のコマンドを実行して、systemtap-init スクリプトパッケージをインストールします。

   ```
   # yum install systemtap-initscript
   ```

2. **設定ファイルのコピー**
   以下のコマンドを実行して、systemtap スクリプトと設定ファイルを 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は、qemu_kvm.stp で指定されています。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を追加または削除するようにカスタマイズできます。

   SystemTap のカスタマイズを **qemu_kvm.conf** にすると、フライトレコーダーバッファーサイズや、メモリーのみまたはディスクにもトレースを保存するかどうかを制御できます。

3. **サービスを起動します。**
   以下のコマンドを実行して、systemtap サービスを開始します。

   ```
   # systemctl start systemtap qemu_kvm
   ```

4. **システムの起動時に systemtap を有効にして実行できるようにします。**
   システムの起動時に、次のコマンドを実行して、systemtap サービスを有効にします。

   ```
   # systemctl enable systemtap qemu_kvm
   ```

5. **サービスの実行の確認**
   次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証

1. **トレースバッファーダンプファイルの作成**
   trace.log という名前のトレースバッファーダンプファイルを作成し、次のコマンドを実行して、tmp ディレクトリーに置きます。

   ```
   # staprun -A qemu_kvm >/tmp/trace.log
   ```

   ファイル名と場所は変更できます。

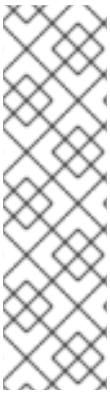
2. **サービスを起動します。**

前の手順でサービスが停止した場合は、以下のコマンドを実行してサービスを再起動します。

```
# systemctl start systemtap qemu_kvm
```

3. **トレースの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

以下の注意事項および制限事項が記載されている必要があります。

- systemtap サービスは、デフォルトでは無効になっています。
- このサービスを有効にすると、パフォーマンスが低下しますが、これは、どのイベントを完全に有効化するかによって異なります。
- **/usr/share/doc/qemu-kvm-*/README.systemtap** に 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得する python スクリプトです。**kvm_stat** コマンドは、**kvm** に表示されるゲストの動作を診断するために使用できます。特に、ゲストのパフォーマンスに関連する問題です。現在、報告されている統計はシステム全体を対象としており、実行中のすべてのゲストの動作が報告されます。このスクリプトを実行するには、qemu-kvm-tools パッケージをインストールする必要があります。詳細は、「既存の Red Hat Enterprise Linux システム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 コマンドでは、**kvm** カーネルモジュールが読み込まれ、**debugfs** がマウントされている必要があります。上記のいずれかの機能が有効になっていない場合は、**debugfs** または **kvm** モジュールを有効にするために必要な手順が表示されます。以下に例を示します。

```
# kvm_stat
Please mount debugfs ('mount -t debugfs debugfs /sys/kernel/debug')
and ensure the kvm modules are loaded
```

必要に応じて**debugfs** をマウントします。

```
# mount -t debugfs debugfs /sys/kernel/debug
```

kvm_stat 出力

kvm_stat コマンドは、すべてのゲストとホストの統計情報を出力します。(**Ctrl**+**c** または **q** を使用して) コマンドが終了するまで、出力は更新されます。画面に表示される出力は異なる場合があることに注意してください。出力要素の説明については、用語のいずれかをクリックすると、リンク先の定義が表示されます。

```
# kvm_stat

kvm statistics
```

```
kvm_exit                    17724       66
 Individual exit reasons follow, see kvm_exit (NAME) for more information.
kvm_exit(CLGI)                  0        0
kvm_exit(CPUID)                 0        0
kvm_exit(CR0_SEL_WRITE)         0        0
kvm_exit(EXCP_BASE)             0        0
kvm_exit(FERR_FREEZE)           0        0
kvm_exit(GDTR_READ)             0        0
kvm_exit(GDTR_WRITE)            0        0
kvm_exit(HLT)                  11       11
kvm_exit(ICEBP)                 0        0
kvm_exit(IDTR_READ)             0        0
kvm_exit(IDTR_WRITE)            0        0
kvm_exit(INIT)                  0        0
kvm_exit(INTR)                  0        0
kvm_exit(INVD)                  0        0
kvm_exit(INVLPG)                0        0
kvm_exit(INVLPGA)               0        0
kvm_exit(IOIO)                  0        0
kvm_exit(IRET)                  0        0
kvm_exit(LDTR_READ)             0        0
kvm_exit(LDTR_WRITE)            0        0
kvm_exit(MONITOR)               0        0
kvm_exit(MSR)                  40       40
kvm_exit(MWAIT)                 0        0
kvm_exit(MWAIT_COND)            0        0
kvm_exit(NMI)                   0        0
kvm_exit(NPF)                   0        0
kvm_exit(PAUSE)                 0        0
kvm_exit(POPF)                  0        0
kvm_exit(PUSHF)                 0        0
kvm_exit(RDPMC)                 0        0
kvm_exit(RDTSC)                 0        0
kvm_exit(RDTSCP)                0        0
kvm_exit(READ_CR0)              0        0
kvm_exit(READ_CR3)              0        0
kvm_exit(READ_CR4)              0        0
kvm_exit(READ_CR8)              0        0
kvm_exit(READ_DR0)              0        0
kvm_exit(READ_DR1)              0        0
kvm_exit(READ_DR2)              0        0
kvm_exit(READ_DR3)              0        0
kvm_exit(READ_DR4)              0        0
kvm_exit(READ_DR5)              0        0
kvm_exit(READ_DR6)              0        0
kvm_exit(READ_DR7)              0        0
kvm_exit(RSM)                   0        0
kvm_exit(SHUTDOWN)              0        0
kvm_exit(SKINIT)                0        0
kvm_exit(SMI)                   0        0
kvm_exit(STGI)                  0        0
kvm_exit(SWINT)                 0        0
kvm_exit(TASK_SWITCH)           0        0
kvm_exit(TR_READ)               0        0
kvm_exit(TR_WRITE)              0        0
kvm_exit(VINTR)                 1        1
```

```
kvm_exit(VMLOAD)              0        0
kvm_exit(VMMCALL)             0        0
kvm_exit(VMRUN)               0        0
kvm_exit(VMSAVE)              0        0
kvm_exit(WBINVD)              0        0
kvm_exit(WRITE_CR0)           2        2
kvm_exit(WRITE_CR3)           0        0
kvm_exit(WRITE_CR4)           0        0
kvm_exit(WRITE_CR8)           0        0
kvm_exit(WRITE_DR0)           0        0
kvm_exit(WRITE_DR1)           0        0
kvm_exit(WRITE_DR2)           0        0
kvm_exit(WRITE_DR3)           0        0
kvm_exit(WRITE_DR4)           0        0
kvm_exit(WRITE_DR5)           0        0
kvm_exit(WRITE_DR6)           0        0
kvm_exit(WRITE_DR7)           0        0
kvm_entry                 17724       66
kvm_apic                  13935       51
kvm_emulate_insn          13924       51
kvm_mmio                  13897       50
varl-kvm_eoi               3222       12
kvm_inj_virq               3222       12
kvm_apic_accept_irq        3222       12
kvm_pv_eoi                 3184       12
kvm_fpu                     376        2
kvm_cr                      177        1
kvm_apic_ipi                278        1
kvm_msi_set_irq             295        0
kvm_pio                      79        0
kvm_userspace_exit           52        0
kvm_set_irq                  50        0
kvm_pic_set_irq              50        0
kvm_ioapic_set_irq           50        0
kvm_ack_irq                  25        0
kvm_cpuid                    90        0
kvm_msr                      12        0
```

変数の説明

- **kvm_ack_irq** - 割り込みコントローラー (PIC/IOAPIC) の割り込み確認応答の回数。
- **kvm_age_page** - メモリー管理ユニット (MMU) 通知機能によるページエージの反復回数。
- **kvm_apic** - APIC レジスターのアクセス回数。
- **kvm_apic_accept_irq** - ローカル APIC に許可されている割り込みの数。
- **kvm_apic_ipi** - インタープロセッサー割り込みの数。
- **kvm_async_pf_completed** - 非同期ページフォルトの補完数。
- **kvm_async_pf_doublefault** - 非同期ページフォルト停止回数。
- **kvm_async_pf_not_present** - 非同期ページフォルトの初期化回数。

- **kvm_async_pf_ready** - 非同期ページフォルトの補完数。
- **kvm_cpuid** - 実行された CPUID 命令の数。
- **kvm_cr** - トラップおよびエミュレートされた制御レジスター (CR) のアクセス数 (CR0、CR3、CR4、CR8)。
- **kvm_emulate_insn** - エミュレートされた命令の数。
- **kvm_entry** - エミュレートされた命令の数。
- **kvm_eoi** - APIC (Advanced Programmable Interrupt Controller) end of interrupt (EOI) 通知の数。
- **kvm_exit** - **VM-exits** の数。
- **kvm_exit (NAME)** - プロセッサー固有の個別の出口。詳細は、プロセッサーのドキュメントを参照してください。
- **kvm_fpu** - KVM 浮動小数点ユニット (FPU) の再読み込みの回数。
- **kvm_hv_hypercall** - Hyper-V ハイパーコールの数。
- **kvm_hypercall** - Hyper-V 以外のハイパーコールの数。
- **kvm_inj_exception** - ゲストに挿入された例外の数。
- **kvm_inj_virq** - ゲストに挿入された割り込みの数。
- **kvm_invlpga** - 傍受された INVLPGA 命令の数。
- **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。
- **kvm_mmio** - エミュレートされたメモリーマップ I/O (MMIO) 操作の数。
- **kvm_msi_set_irq** - message-signaled interrupts (MSI) の数。
- **kvm_msr** - モデル固有レジスター (MSR) のアクセス回数。
- **kvm_nested_intercepts** - L1 ⇒ L2 ネストされた SVM スイッチの数。
- **kvm_nested_vmrun** - L1 ⇒ L2 ネストされた SVM スイッチの数。
- **kvm_nested_intr_vmexit** - 割り込みウィンドウによるネストされた VM-exit 挿入の回数。
- **kvm_nested_vmexit** - ネストされた (L2) ゲストの実行中のハイパーバイザーの終了。
- **kvm_nested_vmexit_inject** - L2 ⇒ L1 ネストされたスイッチの数。
- **kvm_page_fault** - ハイパーバイザーが処理しているページフォルトの数。
- **kvm_pic_set_irq** - 仮想 Programmable Interrupt Controller (PIC) への割り込みレベルの変更回数。
- **kvm_pio** - エミュレートされたプログラム I/O (PIO) 操作の回数。
- **kvm_pv_eoi** - 準仮想化 End of Input (EOI) イベントの数。

- **kvm_set_irq** - ジェネリック IRQ コントローラーレベルでの割り込みレベル変更の回数 (PIC、IOAPIC、および MSI のカウント)。
- **kvm_skinit** - 仮想マシンスキニットの終了回数。
- **kvm_track_tsc** - タイムスタンプカウンター (TSC) 書き込み回数。
- **kvm_try_async_get_page** - 非同期ページフォルトの試行回数。
- **kvm_update_master_clock** - pvclock マスタークロックの更新回数。
- **kvm_userspace_exit** - ユーザー空間を終了した回数。
- **kvm_write_tsc_offset** - TSC オフセット書き込み回数。
- **vcpu_match_mmio** - SPTE キャッシュされたメモリーマップ I/O (MMIO) のヒット回数。

kvm_stat コマンドからの出力は、KVM ハイパーバイザーにより、**/sys/kernel/debug/tracing/events/kvm/** ディレクトリーにある疑似ファイルとしてエクスポートされます。

A.5. シリアルコンソール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、ビデオデバイスまたはヘッドレスサーバーでカーネルパニックおよびハードウェアの問題のデバッグに役立ちます。

KVM ゲストのシリアルコンソール出力を有効にするには、次のコマンドを実行します。

1. ゲストのドメイン XML ファイルに、シリアルコンソールの設定が含まれていることを確認します。以下に例を示します。

```
<console type='pty'>
  <source path='/dev/pts/16'/>
  <target type='virtio' port='1'/>
  <alias name='console1'/>
</console>
```

2. ゲストで、Red Hat ナレッジベースの記事 How can I enable serial console for Red Hat Enterprise Linux 7? に従います。

ホストで次のコマンドを実行すると、シリアルコンソールにアクセスできます。*guestname* はゲスト仮想マシンの名前です。

```
# virsh console guestname
```

virt-manager を使用して、仮想テキストコンソールを表示することもできます。ゲストコンソールウィンドウで、**View** メニューから **Text Consoles** の **Serial 1** を選択します。

A.6. 仮想化ログ

以下の方法を使用すると、ハイパーバイザーおよびゲストのイベントに関するログデータにアクセスできます。これは、システムの仮想化のトラブルシューティングに役立ちます。

- 各ゲストには、**/var/log/libvirt/qemu/** ディレクトリーに保存されているログがあります。ログには *GuestName*.log という名前が付けられ、容量の制限に到達すると定期的に圧縮されます。
- **systemd** Journal で **libvirt** イベントを表示するには、

  ```
  # journalctl _SYSTEMD_UNIT=libvirtd.service
  ```

  のコマンドを使用します。
- **auvirt** コマンドは、ハイパーバイザーのゲストに関連する監査結果を表示します。表示されるデータは、特定のゲスト、時間枠、および情報形式を選択することで、絞り込むことができます。たとえば、次のコマンドは、当日の**testguest** 仮想マシンで発生したイベントの概要を提供します。

  ```
  # auvirt --start today --vm testguest --summary
  Range of time for report:       Mon Sep  4 16:44 - Mon Sep  4 17:04
  Number of guest starts:         2
  Number of guest stops:          1
  Number of resource assignments: 14
  Number of related AVCs:         0
  Number of related anomalies:    0
  Number of host shutdowns:       0
  Number of failed operations:    0
  ```

  また、auvirt の情報を **systemd** ジャーナルに自動的に追加するように設定することもできます。これを行うには、**/etc/libvirt/libvirtd.conf** ファイルを編集して、***audit_logging*** パラメーターの値を **1** に設定します。

  詳細は、man ページの **auvirt** を参照してください。
- 仮想マシンマネージャーでエラーが発生した場合は、**$HOME/.virt-manager/** ディレクトリーの **virt-manager.log** ファイルで生成されたデータを確認できます。
- ハイパーバイザーシステムの監査ログは、**/var/log/audit/audit.log** ファイルを参照してください。
- ゲストオペレーティングシステムによっては、さまざまなシステムログファイルがゲストに保存される場合もあります。

Red Hat Enterprise Linux でのログインの詳細は、System Administrator's Guide を参照してください。

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を使用する場合は、設定されているループデバイスの数を増やす必要がある場合があります。デフォルト設定では、最大 8 つのアクティブループデバイスが使用できます。8 個以上のファイルベースのゲストまたはループデバイスが必要な場合は、設定されたループデバイスを **/etc/modprobe.d/**ディレクトリーで調整できます。以下の行を追加します。

```
options loop max_loop=64
```

この例では 64 を使用しますが、別の数値を指定して最大ループ値を設定できます。また、システムにループデバイスバッキングゲストを実装する必要がある場合もあります。完全仮想化システムでループデバイスバッキングゲストを使用するには、**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エラー

ゲストのメモリー変更が速すぎて、ライブマイグレーションプロセスでメモリーを何度も転送する必要があり、終了 (収束) に失敗する場合があります。

現在のライブマイグレーション実装では、デフォルトのマイグレーション時間が 30ms に設定されています。この値は、残りを転送するために、移行の最後にゲストの一時停止時間を決定します。値を大きくすると、ライブマイグレーションが収束する可能性が高くなります

A.9. BIOS での INTEL VT-X および AMD-V VIRTUALIZATION ハードウェア拡張の有効化

注記

専門知識を深めるには、Red Hat Virtualization (RH318) のトレーニングコースの受講を推奨します。

本セクションでは、ハードウェア仮想化拡張機能を識別し、無効になっている場合に、その拡張機能を BIOS で有効にする方法を説明します。

Intel VT-x 拡張機能は、BIOS で無効にできます。特定のラップトップベンダーは、CPU で Intel VT-x 拡張機能をデフォルトで無効にしています。

AMD-V の BIOS では、仮想化拡張機能を無効にできません。

無効にした仮想化拡張機能を有効にする手順は、次のセクションを参照してください。

BIOS で、仮想化拡張機能が有効になっていることを確認します。Intel VT または AMD-V の BIOS 設定は、通常 **Chipset** メニューまたは **Processor** メニューにあります。メニュー名は、このガイドとは異なる場合があります。仮想化拡張機能の設定は、**Security Settings** や、他の非標準のメニュー名などに記載されている場合があります。

手順A.3 BIOS での仮想化拡張機能の有効化

1. コンピューターを再起動し、システムの BIOS メニューを開きます。通常、これは **delete** キー、**F1** キー、または **Alt** キーと **F4** キーを押して実行できます。
2. **BIOS での仮想化拡張機能の有効化**

注記

以下の手順の多くは、使用しているマザーボード、プロセッサーのタイプ、チップセット、および OEM により異なる場合があります。システムの設定に関する正しい情報は、システムに付属のドキュメントを参照してください。

a. **Processor** サブメニューを開きます。プロセッサー設定メニューは、**Chipset**、**Chipset**、または **Northbridge** で非表示にすることができます。

b. **Intel Virtualization Technology** を有効にします (Intel VT-x としても知られています)。**AMD-V** 拡張機能は、BIOS では無効にできないため、事前に有効にしておく必要があります。仮想化拡張機能は、OEM やシステム BIOS によっては、**Virtualization Extensions**、**Vanderpool**、またはその他の名前にラベルが付けられている場合があります。

c. オプションが利用できる場合は、Intel VT-d または AMD IOMMU を有効にします。PCI デバイスの割り当てには、Intel VT-d および AMD IOMMU を使用します。

d. **Save & Exit**を選択します。

3. マシンを再起動します。

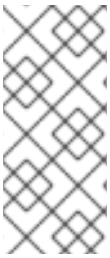
4. マシンが起動したら、**grep -E "vmx|svm" /proc/cpuinfo** を実行します。***--color*** の指定は任意ですが、検索語を強調表示する場合に便利です。コマンドが出力すると、仮想化拡張機能が有効になります。出力がない場合は、システムの仮想化拡張機能や、正しい BIOS 設定が有効になっていない可能性があります。

A.10. RED HAT ENTERPRISE LINUX 7 ホストでの RED HAT ENTERPRISE LINUX 6 ゲストのシャットダウン

Minimal installation を使用して Red Hat Enterprise Linux 6 ゲスト仮想マシンをインストールしても、acpid (acpi デーモン) はインストールされません。Red Hat Enterprise Linux 7 は、 **systemd** に引き継がれたため、このパッケージを必要としなくなりました。ただし、Red Hat Enterprise Linux 7 ホストで実行されている Red Hat Enterprise Linux 6 ゲスト仮想マシンには引き続き必要です。

acpid パッケージがないと、**virsh shutdown** コマンドの実行時に Red Hat Enterprise Linux 6 ゲスト仮想マシンはシャットダウンしません。**virsh shutdown** は、ゲスト仮想マシンを適切にシャットダウンするように設計されています。

virsh shutdown コマンドを使用すると、システム管理が簡単かつ安全になります。**virsh shutdown** コマンドを使用して正常にシャットダウンしない場合は、システム管理者がゲスト仮想マシンに手動でログインするか、**Ctrl**-**Alt**-**Del** のキーの組み合わせを各ゲスト仮想マシンに送信する必要があります。



注記

その他の仮想化オペレーティングシステムは、この問題の影響を受ける可能性があります。**virsh shutdown** コマンドでは、ACPI のシャットダウン要求を処理するようにゲスト仮想マシンのオペレーティングシステムを設定する必要があります。多くのオペレーティングシステムでは、ACPI のシャットダウン要求を受け入れるために、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です。

手順A.4 Red Hat Enterprise Linux 6 ゲストの回避策

1. **acpid パッケージのインストール**
acpid サービスは、ACPI 要求をリッスンして処理します。

   ゲスト仮想マシンにログインし、ゲスト仮想マシンに acpid パッケージをインストールします。

   ```
   # yum install acpid
   ```

2. **ゲストで acpid サービスを有効にします。**
ゲスト仮想マシンの起動シーケンス中に**acpid**サービスを開始するように設定し、サービスを開始します。

   ```
   # chkconfig acpid on
   # service acpid start
   ```

3. **ゲストドメイン XML の準備**

ドメインの XML ファイルを編集して、次の要素を追加します。virtio シリアルポートを **org.qemu.guest_agent.0** に置き換え、表示されている名前の代わりにゲストの名前を使用します。この例では、ゲストは guest1 です。ファイルを保存することを忘れないでください。

図A.1 ゲスト XML の置き換え

```
<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/guest1.agent'/>
  <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

4. **QEMU ゲストエージェントのインストール**
   QEMU ゲストエージェント (QEMU-GA) をインストールし、『Red Hat Enterprise Linux 6 Virtualization Administration Guide』 の指示に従ってサービスを起動します。

5. **ゲストをシャットダウンします。**

   a. 既知のゲスト仮想マシンのリストを表示して、シャットダウンするゲスト仮想マシンの名前を取得できます。

   ```
   # virsh list --all
     Id Name                 State
   ----------------------------------
    14 guest1                running
   ```

   b. ゲスト仮想マシンをシャットダウンします。

   ```
   # virsh shutdown guest1

   guest virtual machine guest1 is being shutdown
   ```

   c. ゲスト仮想マシンがシャットダウンするまで数秒待ちます。シャットダウンしていることを確認します。

   ```
   # virsh list --all
    Id Name                 State
   ----------------------------------
    14 guest1                shut off
   ```

   d. 編集した XML ファイルを使用して、*guest1* という名前のゲスト仮想マシンを起動します。

   ```
   # virsh start guest1
   ```

   e. *guest1* ゲスト仮想マシンで acpi をシャットダウンします。

   ```
   # virsh shutdown --mode acpi guest1
   ```

   f. 再度すべてのゲスト仮想マシンをリスト表示します。*guest1* は引き続きリストにあり、シャットダウンされていることが示されているはずです。

```
#  virsh list --all
  Id Name                 State
----------------------------------
  14 guest1               shut off
```

g. 編集した XML ファイルを使用して、*guest1* という名前のゲスト仮想マシンを起動します。

```
# virsh start guest1
```

h. *guest1* ゲスト仮想マシンゲストエージェントをシャットダウンします。

```
# virsh shutdown --mode agent guest1
```

i. ゲスト仮想マシンを一覧表示します。*guest1* はまだリストにあるはずで、シャットオフされていることを示しているはずです。

```
# virsh list --all
  Id Name                 State
----------------------------------
  guest1                shut off
```

ゲスト仮想マシンは、上記の回避策を使用せずに、連続したシャットダウンの **virsh shutdown** コマンドを使用してシャットダウンします。

上記の方法の他に、**libvirt-guests** サービスを停止することで、ゲストを自動的にシャットダウンできます。このメソッドの詳細は、「(オプション) 正常なシャットダウンを許可する回避策」 を参照してください。

A.11. (オプション) 正常なシャットダウンを許可する回避策

libvirt-guests サービスには、ゲストが適切にシャットダウンできるように設定できるパラメーターがあります。これは、libvirt インストールの一部で、デフォルトでインストールされるパッケージです。このサービスは、ホストのシャットダウン時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ゲストをシャットダウン前の状態に復元します。デフォルトでは、この設定はゲストを一時停止するように設定されています。ゲストを適切にシャットダウンするには、**libvirt-guests** 設定ファイルのパラメーターのいずれかを変更する必要があります。

手順A.5 ゲストの正常なシャットダウンを可能にするための libvirt-guests サービスパラメーターの変更

ここで説明する手順では、ホストの物理マシンが停止している場合、電源が切れている場合、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できるようにします。

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 (#) を削除して、***ON_SHUTDOWN=suspend***を ***ON_SHUTDOWN=shutdown*** に変更します。変更を保存することを忘れないでください。

```
$ 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
# - ignore  libvirt-guests init script won't start any guest on boot, however,
#           guests marked as autostart will still be automatically started by
#           libvirtd
#ON_BOOT=start

# Number of seconds to wait between each guest start. Set to 0 to allow
# 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
```

???

URIS - checks the specified connections for a running guest. The ***Default*** setting functions in the same manner as **virsh** does when no explicit URI is set In addition, one can explicitly set the URI from **/etc/libvirt/libvirt.conf**. Note that when using the libvirt configuration file default setting, no probing will be used.

???

ON_BOOT - specifies the action to be done to / on the guests when the host boots. The ***start*** option starts all guests that were running prior to shutdown regardless on their autostart settings. The ***ignore*** option will not start the formally running guest on boot, however, any guest marked as autostart will still be automatically started by **libvirtd**.

???

The ***START_DELAY*** - sets a delay interval in between starting up the guests. This time period is set in seconds. Use the 0 time setting to make sure there is no delay and that all guests are started simultaneously.

???

ON_SHUTDOWN - specifies the action taken when a host shuts down. Options that can be set include: ***suspend*** which suspends all running guests using **virsh managedsave** and ***shutdown*** which shuts down all running guests. It is best to be careful with using the **shutdown** option as there is no way to distinguish between a guest which is stuck or ignores shutdown requests and a guest that just needs a longer time to shutdown. When setting the ***ON_SHUTDOWN=shutdown***, you must also set ***SHUTDOWN_TIMEOUT*** to a value suitable for the guests.

???

PARALLEL_SHUTDOWN Dictates that the number of guests on shutdown at any time will not exceed number set in this variable and the guests will be suspended concurrently. If set to **0**, then guests are not shutdown concurrently.

???

Number of seconds to wait for a guest to shut down. If ***SHUTDOWN_TIMEOUT*** is enabled, this timeout applies as a timeout for shutting down all guests on a single URI defined in the variable URIS. If ***SHUTDOWN_TIMEOUT*** is set to ***0***, then there is no timeout (use with caution, as guests might not respond to a shutdown request). The default value is 300 seconds (5 minutes).

???

BYPASS_CACHE can have 2 values, 0 to disable and 1 to enable. If enabled it will by-pass the file system cache when guests are restored. Note that setting this may effect performance and may cause slower operation for some file systems.

2. **libvirt-guests サービスを開始します。**
   サービスを開始していない場合は、**libvirt-guests** サービスを開始します。サービスを再起動しないでください。再起動すると、実行中のゲスト仮想マシンがすべてシャットダウンします。

A.12. KVM ネットワークパフォーマンス

デフォルトでは、KVM 仮想マシンには、仮想 Realtek 8139 (rtl8139) の NIC (ネットワークインターフェイスコントローラー) が割り当てられています。

rtl8139 仮想化 NIC はほとんどの環境で問題なく機能しますが、このデバイスは、10 ギガバイトイーサネットなど、一部のネットワークでパフォーマンス低下の問題が発生する可能性があります。

パフォーマンスを向上させるために、準仮想化ネットワークドライバーに切り替えることができます。

注記

仮想化 Intel PRO/1000 (**e1000**) ドライバーも、エミュレートされたドライバーとしてサポートされていることに注意してください。**e1000** ドライバーを使用する場合は、以下の手順の **virtio** を **e1000** に置き換えます。最善のパフォーマンスを実現するには、**virtio** ドライバーを使用することが推奨されます。

手順A.6 virtio ドライバーへの切り替え

1. ゲストオペレーティングシステムをシャットダウンします。
2. **virsh** コマンドでゲストの設定ファイルを編集します (***GUEST*** はゲストの名前です)。

```
# virsh edit GUEST
```

virsh edit コマンドは、**$EDITOR** シェル変数を使用して、使用するエディターを決定します。

3. 設定のネットワークインターフェイスセクションを見つけます。このセクションは、以下のスニペットと似ています。

```
<interface type='network'>
  [output truncated]
  <model type='rtl8139' />
</interface>
```

4. モデル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 ドライバーから virtio ドライバーに変更されます。

```
<interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

5. 変更を保存し、テキストエディターを終了します。

6. ゲストオペレーティングシステムを再起動します。

その他のネットワークドライバーを使用した新しいゲストの作成

別のネットワークドライバーを使用して、新しいゲストを作成することもできます。ネットワーク接続でゲストをインストールするのが困難な場合に必要になることがあります。この方法では、テンプレートとして使用するために、少なくとも 1 つのゲストが作成されている (CD または DVD からインストールされている可能性あり) 必要があります。

1. 既存のゲスト (この例では *Guest1*) 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て編集し、一意のフィールド (仮想マシン名、UUID、ディスクイメージ、MAC アドレス、およびその他の一意のパラメーター) を更新します。UUID および MAC アドレス行を削除できることに注意してください。virsh を実行すると、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イスセクションにモデル行を追加します。

```
 <interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

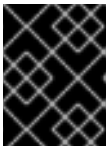
3. 新しい仮想マシンを作成します。

```
# virsh define /tmp/new-guest.xml
# virsh start new-guest
```

A.13. LIBVIRT を使用して外部スナップショットを作成するための回避策

KVM ゲストのスナップショットには、以下の 2 つのクラスがあります。

- **Internal snapshots** は qcow2 ファイルに完全に含まれ、libvirt で完全に対応しているため、スナップショットの作成、削除、および復帰が可能になります。これは、特にオプションが指定されていない場合に、スナップショットの作成時に libvirt が使用するデフォルトの設定です。このファイルタイプは、スナップショットの作成に他のタイプよりも若干時間がかかります。また、qcow2 ディスクが必要という欠点があります。

  重要

  内部スナップショットは積極的に開発されていないため、Red Hat では使用を推奨していません。

- **外部スナップショット** は、任意のタイプの元のディスクイメージと連携し、ゲストのダウンタイムなしに取得でき、より安定し、信頼性も高いです。このため、KVM ゲスト仮想マシンでは外部スナップショットの使用が推奨されます。ただし、外部スナップショットは現在、Red Hat Enterprise Linux 7 には完全に実装されておらず、**virt-manager** を使用する際には利用できません。

  外部スナップショットを作成するには、**--diskspec vda,snapshot=external** オプションを指定して、**snapshot-create-as** を使用するか、スナップショットの XML ファイルで次の **disk** 行を使用します。

  ```
  <disk name='vda' snapshot='external'>
    <source file='/path/to,new'/>
  </disk>
  ```

  現時点では、libvirt が作成できるものの、それ以上操作を行うことができないため、外部スナップショットは一方向操作となります。回避策は、libvirt アップストリームページで 説明されています。

A.14. 日本語キーボードを使用したゲストコンソールでの文字の欠落

Red Hat Enterprise Linux 7 ホストで、日本語キーボードをマシンにローカルに接続すると、ゲストコンソールでアンダースコア (_文字) などの文字が正しく表示されない場合があります。これは、必要なキーマップがデフォルトで正しく設定されていないために発生します。

Red Hat Enterprise Linux 6 および Red Hat Enterprise Linux 7 ゲストでは、通常、関連するキーを押したときにエラーメッセージが表示されません。ただし、Red Hat Enterprise Linux 4 および Red Hat Enterprise Linux 5 のゲストでは、以下のようなエラーが表示される場合があります。

```
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 でこの問題を修正するには、次の手順を実行します。

- 影響を受けるゲストを **virt-manager** で開きます。
- **View → Details** をクリックします。
- リストで **Display VNC** を選択します。

- **Keymap** プルダウンメニューで、**Auto** を **ja** に変更します。
- **Apply** ボタンをクリックします。

または、ターゲットゲストで **virsh edit** コマンドを使用してこの問題を修正する場合は、次のコマンドを実行します。

- **virsh edit *guestname***を実行します。
- <graphics> タグに以下の属性を追加します。**keymap='ja'**。以下に例を示します。

```
<graphics type='vnc' port='-1' autoport='yes' keymap='ja'/>
```

A.15. ゲスト仮想マシンのシャットダウンに失敗する

従来は、**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るため、物理マシンの電源ボタンを押したときと同じ動作がコピーされていました。すべての物理マシン内で、このイベントを処理するかどうかは OS 次第になります。以前は、オペレーティングシステムは単にサイレントシャットダウンしていました。現在、最も一般的なアクションは、実行すべき内容を尋ねるダイアログを表示することです。オペレーティングシステムによっては、特にユーザーがログインしていない場合に、このイベントを完全に無視するものもあります。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** は機能しません (無視されるか、仮想ディスプレイにダイアログが表示されます)。ただし、ゲスト仮想マシンに qemu-guest-agent チャンネルが追加されており、このエージェントがゲスト仮想マシンの OS 内で実行している場合は、**virsh shutdown** コマンドにより 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求められます。エージェントはゲスト仮想マシン OS 内からシャットダウンを呼び出し、すべてが想定どおりに機能します。

手順A.7 ゲスト仮想マシンで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. ゲスト仮想マシンの ドメイン XML を開き、以下のスニペットを追加します。

   図A.2 ゲストエージェントチャネルの設定

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

3. **virsh start [domain]** を実行して、ゲスト仮想マシンを起動します。
4. ゲスト仮想マシン (**yum install qemu-guest-agent**) に qemu-guest-agent をインストールし、サービス (qemu-guest-agent.service) としてシステムを起動するたびに自動的に実行するようにします。

A.16. ゲスト仮想マシンの SMART ディスク監視の無効化

SMART ディスクの監視は、仮想ディスクと物理ストレージデバイスがホストの物理マシンにより管理されているため、安全に無効にできます。

```
# service smartd stop
# systemctl --del smartd
```

A.17. LIBGUESTFS トラブルシューティング

libguestfs が機能しているかどうかを確認するテストツールを利用できます。libguestfs (root アクセスは必要ありません) をインストールした後、以下のコマンドを入力して、通常の操作をテストします。

```
$ libguestfs-test-tool
```

このツールは、libguestfs の動作をテストするために、大量のテキストを出力します。テストが成功すると、出力の末尾に以下のテキストが表示されます。

```
===== TEST FINISHED OK =====
```

A.18. SR-IOV のトラブルシューティング

本セクションは、SR-IOV に影響を与える可能性がある問題の解決策を記載します。追加のヘルプが必要な場合は、「SR-IOV 仮想機能のプールからの PCI デバイス割り当ての設定」 を参照してください。

ゲストの起動エラー

設定した仮想マシンを起動すると、次のようなエラーが発生します。

```
# 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、別のゲストまたはホスト自体に割り当てられているデバイスが原因で発生します。

ゲストの移行、保存、またはダンプのエラー

仮想マシンの移行およびダンプを試行すると、以下のようなエラーが発生します。

```
# virsh dump rhel7/tmp/rhel7.dump

error: Failed to core dump domain rhel7 to /tmp/rhel7.dump
error: internal error: unable to execute QEMU command 'migrate': State blocked by non-
migratable device '0000:00:03.0/vfio-pci'
```

デバイスの割り当ては、仮想マシンが起動した特定ホストのハードウェアを使用するため、デバイスの割り当てが使用中の場合は、ゲストの移行と保存に対応しません。現在では、ゲストのコアダンプにも同じ制限が適用されます。これは将来変更される可能性があります。QEMU は、現在、**--memory-only** が指定されていない限り、PCI デバイスが接続されているゲスト仮想マシンでの移行、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、このアクションは USB デバイスでのみ対応しています。現在、これを改善するための作業が行われています。

A.19. 一般的なLIBVIRTエラーとトラブルシューティング

この付録では、一般的な **libvirt** 関連の問題とエラー、およびそれらに対処するための手順について説明します。

トラブルシューティングの詳細は、以下の表でエラーを特定し、**Solution** の対応するリンクを参照してください。

表A.1 一般的なlibvirtエラー

エラー	問題の説明	解決方法
libvirtd の起動に失敗しました。	libvirt デーモンの起動に失敗しました。ただし、**/var/log/messages** にはこのエラーに関する情報はありません。	「libvirtd が起動しない」
Cannot read CA certificate	これは、URI がハイパーバイザーに接続できない場合に発生するいくつかのエラーのいずれかになります。	「URI のハイパーバイザー接続に失敗する」
その他の接続エラー	これは、URI がハイパーバイザーに接続できない場合に発生するその他のエラーです。	「URI のハイパーバイザー接続に失敗する」
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起動しますが、DHCP、PXE プロトコルを使用した起動、またはその両方から IP アドレスを取得することはできません。これは、多くの場合、ブリッジの転送遅延時間が長く設定されている場合、または iptables パッケージとカーネルがチェックサムの難号化 (mangle) 規則をサポートしない場合に発生します。	「ゲスト上の PXE ブート (または DHCP) が失敗」
ゲストは外部ネットワークにアクセスできるが、macvtap インターフェイ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ますが、macvtap (または ***type='direct'***) ネットワークインターフェイスを使用するように設定するとホストマシンに接続できません。 これは、実際にはエラーではなく、macvtap の定義済みの動作です。	「ゲストは外部ネットワークにアクセスできるが、macvtap インターフェイスの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ぼ常に無害ですが、間違って問題の証拠と見なされることがよくあります。	「Could not add rule to fixup DHCP response checksums on network *'default'*」

エラー	問題の説明	解決方法
Unable to add bridge br0 port vnet0: No such device	このエラーメッセージまたは類似のメッセージ **Failed to add tap interface to bridge *'br0'*: No such device** は、ゲスト (あるいはドメイン) 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」
Unable to resolve address *name_of_host* service '49155': Name or service not known	QEMU ゲストの移行が失敗し、このエラーメッセージが見慣れないホスト名で表示されます。	「**error: unable to resolve address** で移行が失敗する」
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ません。	「**Unable to allow access for disk path: No such file or directory** を表示して移行が失敗する」
libvirtd の開始時に存在するゲスト仮想マシンがない	libvirt デーモンは正常に起動しますが、**virsh list --all** の実行時にゲスト仮想マシンが存在しないように見えます。	「libvirtd の起動時に存在するゲスト仮想マシンがない」
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 ドキュメントが、API を介して**libvirt** に渡されると、いくつかの一般的なエラーが発生します。このエントリーでは、ゲスト XML 定義の編集方法と、XML 構文および設定における一般的なエラーの詳細を説明します。	「一般的な XML エラー」

A.19.1. libvirtd が起動しない

現象

libvirt デーモンが自動的に起動しない。**libvirt** デーモンの手動による起動も失敗する。

```
# systemctl start libvirtd.service
* Caching service dependencies ...                                [ ok ]
* Starting libvirtd ...
/usr/sbin/libvirtd: error: Unable to initialize network sockets. Check /var/log/messages or run
without --daemon for more info.
* start-stop-daemon: failed to start `/usr/sbin/libvirtd'        [ !! ]
* ERROR: libvirtd failed to start
```

また、**/var/log/messages** ではこのエラーに関する**'more info'**はありません。

調査

以下の行を有効にして、**/etc/libvirt/libvirtd.conf** のlibvirt のログを変更します。この行を有効にするには、テキストエディターで**/etc/libvirt/libvirtd.conf** ファイルを開き、次の行の先頭からハッシュ (**#**) 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。問題を診断したら、**/etc/libvirt/libvirtd.conf** ファイルーでこの行を再度コメント入力することが推奨されます。

libvirt を再起動し、問題が解決されたかどうかを確認します。

それでも**libvirtd** が引き続き正常に起動しない場合は、以下のようなエラーが表示されます。

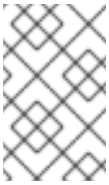
```
# systemctl restart libvirtd
Job for libvirtd.service failed because the control process exited with error code. See "systemctl
status libvirtd.service" and "journalctl -xe" for details.

Sep 19 16:06:02 jsrh libvirtd[30708]: 2017-09-19 14:06:02.097+0000: 30708: info : libvirt version:
3.7.0, package: 1.el7 (Unknown, 2017-09-06-09:01:55, js
Sep 19 16:06:02 jsrh libvirtd[30708]: 2017-09-19 14:06:02.097+0000: 30708: info : hostname: jsrh
Sep 19 16:06:02 jsrh libvirtd[30708]: 2017-09-19 14:06:02.097+0000: 30708: error :
daemonSetupNetworking:502 : unsupported configuration: No server certif
Sep 19 16:06:02 jsrh systemd[1]: libvirtd.service: main process exited, code=exited,
status=6/NOTCONFIGURED
Sep 19 16:06:02 jsrh systemd[1]: Failed to start Virtualization daemon.

-- Subject: Unit libvirtd.service has failed
-- Defined-By: systemd
-- Support: http://lists.freedesktop.org/mailman/listinfo/systemd-devel
--
-- Unit libvirtd.service has failed.
--
-- The result is failed.
```

libvirtd の man ページでは、 **libvirt** を **Listen for TCP/IP connections** モードで実行すると、足りない **cacert.pem** ファイルが TLS 認証局として使用されることを示しています。これは、 ***--listen*** パラメーターが渡されることを意味します。

解決方法

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 Domain Identity, Authentication, and Policy Guide』 の Managing Certificates および Certificate Authorities の章を参照してください。

- TLS は使用せずに TCP を使用してください。**/etc/libvirt/libvirtd.conf** で、**listen_tls = 0** および **listen_tcp = 1** を設定します。デフォルト値は **listen_tls = 1** および **listen_tcp = 0** です。
- ***--listen*** パラメーターを渡さないでください。**/etc/sysconfig/libvirtd.conf** で、***LIBVIRTD_ARGS*** 変数を変更します。

A.19.2. URI のハイパーバイザー接続に失敗する

サーバーへの接続時に、いくつかの異なるエラーが発生する可能性があります (**virsh**を実行する場合など)。

A.19.2.1. Cannot read CA certificate

現象

コマンドの実行中に、以下のエラー (または同様のエラー) が表示される。

```
$ virsh -c qemu://$hostname/system_list
error: failed to connect to the hypervisor
error: Cannot read CA certificate '/etc/pki/CA/cacert.pem': No such file or directory
```

調査

このエラーメッセージは、実際の原因に関して誤解を招くものです。このエラーは、誤って指定された URI や未設定の接続など様々な要素によって起こり得ます。

解決方法

誤って指定された URI

接続 URI に ***qemu://system*** または ***qemu://session*** を指定すると、**virsh** は、ホスト名の ***system*** または ***session*** にそれぞれ接続しようとします。これは、**virsh** が、2 番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。

前方スラッシュを 3 つ使用してローカルホストに接続します。たとえば、***qemu:///system*** を指定すると、**virsh** はローカルホストコンピューターの libvirtd の***system*** インスタンスに接続します。

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** に設定されます。これは証明書になります。

接続が未設定

URI は適切ですが (***qemu[+tls]://server/system*** など)、証明書がマシンに正しく設定されていません。TLS の設定に関する詳細は、アップストリームの **libvirt** web サイト を参照してください。

A.19.2.2. 'host:16509' でサーバーに接続できず、接続が拒否される。

現象

接続のためには **libvirtd** がTCP ポートをリッスンする必要があるが、接続が失敗する。

```
# virsh -c qemu+tcp://host/system
error: failed to connect to the hypervisor
error: unable to connect to server at 'host:16509': Connection refused
```

/etc/libvirt/libvirtd.conf で設定を変更した後も、**libvirt** デーモンは 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、**libvirt** の TCP ポートは設定変更後も開いていない。

```
# netstat -lntp | grep libvirtd
#
```

調査

libvirt デーモンは、**--listen** オプションなしで起動されました。以下のコマンドを実行してこれを確認します。

```
# ps aux | grep libvirtd
root     10749  0.1  0.2 558276 18280 ?        Ssl  23:21   0:00 /usr/sbin/libvirtd
```

出力に **--listen** オプションが含まれていません。

解決方法

--listen オプションでデーモンを起動します。

これを行うには、**/etc/sysconfig/libvirtd** ファイルを修正して、以下の行のコメントを解除します。

```
# LIBVIRTD_ARGS="--listen"
```

次に、以下のコマンドで **libvirtd** サービスを再起動します。

```
# /bin/systemctl restart libvirtd.service
```

A.19.2.3. Authentication Failed

現象

コマンドの実行中に、以下のエラー (または同様のエラー) が表示される。

```
$ virsh -c qemu://$hostname/system_list
error: failed to connect to the hypervisor
error: authentication failed: authentication failed
```

調査

正しい認証情報を使用しても認証に失敗する場合は、SASL 認証が設定されていない可能性があります。

解決方法

1. **/etc/libvirt/libvirtd.conf** ファイルを編集し、***auth_tcp*** パラメーターの値を **sasl** に設定します。確認するには、以下を実行します。

```
# cat /etc/libvirt/libvirtd.conf | grep auth_tcp
auth_tcp = "sasl"
```

2. **/etc/sasl2/libvirt.conf** ファイルを編集し、次の行をファイルに追加します。

```
mech_list: digest-md5
sasldb_path: /etc/libvirt/passwd.db
```

3. cyrus-sasl-md5 パッケージがインストールされていることを確認します。

```
# yum install cyrus-sasl-md5
```

4. **libvirtd** サービスを再起動します。

```
# systemctl restart libvirtd
```

5. libvirt SASL のユーザー名およびパスワードを設定します。

```
# saslpasswd2 -a libvirt 1
```

A.19.2.4. Permission Denied

現象

root 以外のユーザーで **virsh** コマンドを実行すると、以下のエラー (または同様のもの) が表示されます。

```
$ virsh -c qemu://$hostname/system_list
error: Failed to connect socket to '/var/run/libvirt/libvirt-sock': Permission denied
error: failed to connect to the hypervisor
```

解決方法

1. **/etc/libvirt/libvirt.conf** ファイルを編集し、次の行をファイルに追加します。

```
#unix_sock_group = "libvirt"
#unix_sock_ro_perms = "0777"
#unix_sock_rw_perms = "0770"
```

2. **libvirtd** サービスを再起動します。

```
# systemctl restart libvirtd
```

A.19.3. ゲスト上の PXE ブート (または DHCP) が失敗

現象

ゲスト仮想マシンは正常に起動するが、DHCP から IP アドレスを取得できないか、PXE プロトコルを使用してブートを実行できない、またはその両方。このエラーには、ブリッジの転送遅延時間が長く設定されている場合と iptables パッケージとカーネルがチェックサムの難号化 (mangle) 規則をサポートしない場合という 2 つの一般的な原因があります。

ブリッジの転送遅延時間が長い

調査

これは、このエラーの最も一般的な原因になります。ゲストのネットワークインターフェイスが STP (Spanning Tree Protocol) 対応のブリッジデバイスに接続しており、かつ長時間の転送遅延時間が設定されていると、ゲストがブリッジに接続してからその転送遅延時間が経過してからでなければゲスト仮想マシンからブリッジにネットワークパケットを転送しません。この遅延により、インターフェイスからのトラフィックの監視、背後での MAC アドレスの決定、ネットワークトポロジー内の転送ループ防止がブリッジ時間で可能になります。

転送遅延がゲストの PXE や DHCP クライアントのタイムアウトよりも長い場合、クライアントの操作は失敗し、ゲストは (PXE の場合) 起動に失敗するか、(DHCP の場合) IP アドレスの取得に失敗します。

解決方法

これが原因の場合は、ブリッジにおける転送遅延を 0 に変更するか、ブリッジの STP を無効にするか、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への接続に使用されておらず、単に複数のエンドポイントを単一のネットワークに接続するために使用されている (**libvirt** で使用されるブリッジの最も一般的なユースケース) 場合にのみ適用されます 。

ゲストが **libvirt** が管理する仮想ネットワークに接続するインターフェイスを備えている場合、そのネットワークの定義を編集し、再起動します。たとえば、以下のコマンドでデフォルトのネットワークを編集します。

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

<bridge> 要素に以下の属性を追加します。

```
<name_of_bridge='virbr0' delay='0' stp='on'/>
```

注記

delay='0' および **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イスが **libvirt** 外で設定されているホストブリッジに接続されている場合は、遅延設定を変更します。

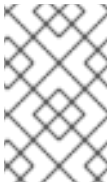
/etc/sysconfig/network-scripts/ifcfg-*name_of_bridge* ファイルで以下の行を追加または編集し、0 秒の遅延で STP を有効にします。

```
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 がネットワークのルートブリッジでない場合、そのブリッジの遅延は最終的にルートブリッジに設定された遅延時間に再設定されます。この発生を防ぐには、*name_of_bridge* の STP を無効にします。

iptable パッケージおよびカーネルは、チェックサム難号化ルールをサポートしません。

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'** が含まれています。

- ホストに **vhost-net** モジュールがロードされている。

  これは、**ls /dev/vhost-net** が空の結果を返さない場合に該当します。

- ゲストがホスト上で直接実行されている DHCP サーバーから IP アドレスを取得しようとしている。

- ホストコンピューターの iptables バージョンが 1.4.10 より古い。

  iptables 1.4.10 は、**libxt_CHECKSUM** 拡張機能を追加する最初のバージョンです。**libvirtd** ログに以下のメッセージが表示される場合は、これに該当します。

  ```
  warning: Could not add rule to fixup DHCP response checksums on network default
  warning: May need to update iptables package and kernel to support CHECKSUM
  rule.
  ```

  重要

  このリストの最初の 3 つの条件すべてが該当していなければ上記の警告メッセージは問題を示すものではなく、無視することができます。

これらの条件が当てはまる場合、ホストからゲストへ送信される UDP パケットには未算出のチェックサムがあります。これにより、ホストの UDP パケットがゲストのネットワークスタックには無効のように表示されます。

解決方法

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。最善のソリューションは、ホストiptablesとカーネルを、できるだけiptables-1.4.10 以降に更新することです。または、この特定のゲストの **vhost-net** ドライバーを無効にすることが最も具体的な修正方法になります。これを実行するには、以下のコマンドでゲストの設定を編集します。

-

```
virsh edit name_of_guest
```

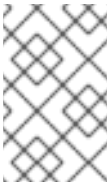
<driver> の行を変更するか、**<interface>** セクションに追加します。

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

変更を保存し、ゲストをシャットダウンしてから再起動します。

この問題が解決されない場合、**firewalld** とデフォルトの **libvirt** ネットワーク間に競合があることが原因として考えられます。

これを修正するには、**service firewalld stop** コマンドで firewalld を停止し、**service libvirtd restart** コマンドで libvirt を再起動します。



注記

また、**/etc/sysconfig/network-scripts/ifcfg-*network_name*** ファイルが正しく設定されている場合は、**dhclient** コマンドを root で使用することで、ゲストが IP アドレスを取得していることを確認できます。

A.19.4. ゲストは外部ネットワークにアクセスできるが、macvtap インターフェイスの使用時にはホストにアクセスできない

現象

ゲスト仮想マシンは他のゲストと通信できますが、macvtap (***type='direct'***) ネットワークインターフェイスを使用するように設定すると、ホストマシンには接続できなくなります。

調査

Virtual Ethernet Port Aggregator (VEPA) や VN-Link 対応スイッチに接続していない場合でも、macvtap インターフェイスは役に立ちます。このようなインターフェイスのモードを **ブ bridge** に設定すると、従来のホストブリッジデバイスの使用に伴う設定問題 (またはNetworkManagerの非互換性) がなく、ゲストが非常に簡単に物理ネットワークに直接接続できるようになります。

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イ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ありません。これは、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、物理インターフェイスに転送されるゲストからそのブリッジへのトラフィックは、ホストの IP スタックに送り返されることはありません。さらに、物理インターフェイスに送信されたホストの IP スタックからのトラフィックも macvtap ブリッジに送り返されず、ゲストに転送することができません。

解決方法

libvirt を使って、分離されたネットワークと、このネットワークに接続する各ゲスト仮想マシンの 2 つ目のインターフェイスを作成します。その後、ホストとゲストは、この分離したネットワークを介して直接通信できるようになります。一方で、NetworkManager との互換性も維持します。

手順A.8 libvirt で分離されたネットワークを作成する

1. **/tmp/isolated.xml** ファイルに以下の XML を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 がすでに使われている場合、別のネットワークを選びます。

   図A.3 分離されたネットワーク XML

   ```
   ...
   <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. 次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を自動起動するように設定します。

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*** を使用して、ネットワーク接続に macvtap を使用する各ゲストの設定を変更し、次のような **<devices>** に新しい**<interface>**を追加します (**<model type='virtio'/>** 行はオプションで追加できます)。

   図A.4 インターフェイスデバイス XML

   ```
   ...
   <interface type='network' trustGuestRxFilters='yes'>
     <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 インターフェイスを使用することになります。詳細は、「ネットワークインターフェイス」 を参照してください。

A.19.5. 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、ほとんどの場合は問題ありません。

解決方法

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。

このような場合の詳細は 「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.19.6. Unable to add bridge br0 port vnet0: No such device

現象

以下のエラーメッセージが表示されます。

> Unable to add bridge *name_of_bridge* port vnet0: No such device

たとえばブリッジ名が *br0* の場合、エラーメッセージは以下のようになります。

> Unable to add bridge br0 port vnet0: No such device

libvirt のバージョン 0.9.6 以前では、以下のようなメッセージになります。

> Failed to add tap interface to bridge *name_of_bridge*: No such device

たとえば、ブリッジの名前が *br0* の場合、エラーは次のようになります。

> 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。

エラーメッセージに記載されているブリッジデバイスが存在しないことを確認するには、シェルコマンド **ip addr show *br0*** を使用します。

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)
```

解決方法

既存のブリッジを編集するか、virsh で新しいブリッジを作成します。

virsh を使用し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 を使用して、既存のブリッジ設定を編集します。

virsh edit *name_of_guest* を使用して、すでに存在するブリッジまたはネットワークを使用するように **<interface>** 定義を変更します。

たとえば、**type='bridge'** を **type='network'** に変更し、**<source bridge='br0'/>** を **<source network='default'/>** に変更します。

virsh を使用したホストブリッジの作成

libvirt バージョン 0.9.8 以降の場合は、**virsh iface-bridge** コマンドでブリッジデバイスを作成できます。これにより、**eth0** のあるブリッジデバイス *br0* が作成され、ブリッジの一部として設定された物理ネットワークインターフェイスが割り当てられます。

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

オプション: 必要に応じて、このブリッジを削除し、次のコマンドを実行して元の**eth0** 設定を復元します。

```
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、ホスト上にブリッジデバイスを手動で作成することができます。手順は、「libvirt を使用したブリッジネットワーク」を参照してください。

A.19.7. error: unable to resolve address で移行が失敗する

現象

QEMU ゲストの移行が失敗し、以下のエラーメッセージが表示される。

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
  error: Unable to resolve address name_of_host service '49155': Name or service not known
```

たとえば、宛先ホスト名が **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** はどこにも使用していないため、このエラーは奇妙です。

調査

移行時に、宛先ホスト上で実行されている **libvirtd** は移行データの受信が予想されるアドレスおよびポートから URI を作成し、これをソースホスト上で実行されている **libvirtd** に送信します。

上記の例では、宛先ホスト (**192.168.122.12**) は名前を *'newyork'* に設定しています。何らかの理由で、そのホスト上で実行中の **libvirtd** は 返されても使用できる IP アドレスに名前を解決できません。このため、移行元の **libvirtd** が名前を解決することを予期して、ホスト名 *'newyork'* を返しました。これは、DNS が適切に設定されていないか、**/etc/hosts** にローカルループバックアドレス (**127.0.0.1**) に関連付けられたホスト名がある場合に発生します。

移行データに使用するアドレスは、移行先**libvirtd**への接続に使用するアドレス (**qemu+tcp://192.168.122.12/system** など) から自動的に決定されないことに注意してください。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。

解決方法

最善の解決法として、DNS を正しく設定し、移行に関連するすべてのホストがすべてのホスト名を解決できるようにすることができます。

DNS をこのように設定できない場合は、各ホスト上の **/etc/hosts** ファイルに、移行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。

いずれの方法でもホスト名を解決できない場合、**virsh migrate** は移行ホストの指定をサポートします。

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system tcp://192.168.122.12
```

移行先の **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 (たとえば、ファイアウォール設定との関連により適切でない場合)、ポート番号は以下のコマンドで指定できます。

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system tcp://192.168.122.12:12345
```

別のオプションとして、トンネル化した移行を使用することもできます。トンネル化した移行では移行データ用に別の接続を作成しませんが、その代わりに移行先 **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.19.8. **Unable to allow access for disk path: No such file or directory** を表示して移行が失敗する

現象

libvirt がディスクイメージにアクセスできないため、ゲスト仮想マシン (またはドメイン) を移行できない。

```
# virsh migrate qemu qemu+tcp://name_of_host/system
error: Unable to allow access for disk path /var/lib/libvirt/images/qemu.img: No such file or
directory
```

たとえば、宛先ホスト名が **newyork** の場合、エラーメッセージは以下のようになります。

```
# virsh migrate qemu qemu+tcp://newyork/system
error: Unable to allow access for disk path /var/lib/libvirt/images/qemu.img: No such file or
directory
```

調査

デフォルトでは、移行で移動するのは実行中のゲストのメモリー内の状態のみです (メモリー-または CPU 状態など)。移行中にはディスクイメージは移動しませんが、両方のホストから同じパスでアクセスできる状態である必要があります。

解決方法

両方のホストの同じ位置に共有ストレージをセットアップし、マウントします。最も簡単な方法として、NFS を使用することができます。

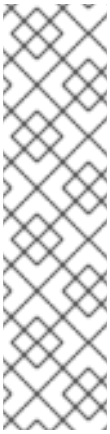
手順A.9 共有ストレージのセットアップ

1. ホスト上に共有ストレージとして機能する NFS サーバーをセットアップします。関連するすべてのホストが NFS で共有ストレージにアクセスしている限り、NFS サーバーには移行関連のいずれかのホストを使用できます。

```
# mkdir -p /exports/images
# cat >>/etc/exports <<EOF
/exports/images    192.168.122.0/24(rw,no_root_squash)
EOF
```

2. **libvirt** を実行しているすべてのホスト上の共通のロケーションに、エクスポートされたディレクトリーをマウントします。たとえば、NFS サーバーの IP アドレスが 192.168.122.1 の場合は、以下のコマンドでディレクトリーをマウントします。

```
# cat >>/etc/fstab <<EOF
192.168.122.1:/exports/images  /var/lib/libvirt/images  nfs  auto  0 0
EOF
# mount /var/lib/libvirt/images
```



注記

NFS を使用してあ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、両方のホストで共有ストレージからマウントされる必要があります。これが正確に設定されていない場合、ゲスト仮想マシンは移行時にそのディスクイメージへのアクセスを失う可能性があります。これは、ゲストを移行先に正常に移行した後に、移行元ホストの **libvirt** デーモンがディスクイメージ上の所有者、権限および SELinux ラベルを変更する可能性があるからです。

libvirt は、ディスクイメージが共有ストレージのロケーション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.19.9. libvirtd の起動時に存在するゲスト仮想マシンがない

現象

libvirt デーモンは正常に起動したが、ゲスト仮想マシンが見当たらない。

-

```
# virsh list --all
 Id    Name                           State
----------------------------------------------------
```

調査

この問題の原因としていくつもの原因が考えられます。以下のテストを実施して原因を特定します。

KVM カーネルモジュールの確認

KVM カーネルモジュールがカーネルに挿入されていることを確認する。

```
# 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 シェルの同様のコマンド**lsmod | grep kvm_amd**を使用して、**kvm_amd**のカーネルモジュールがカーネルに挿入されていることを確認します。

モジュールがない場合は、**modprobe *<modulename>*** を使用して挿入します。

注記

一般的ではありませんが、KVM 仮想化サポートがカーネルにコンパイルされている場合もあります。その場合は、モジュールは必要ありません。

仮想化拡張機能の確認

仮想化拡張機能がホスト上でサポートされ、有効にされていることを確認します。

```
# egrep "(vmx|svm)" /proc/cpuinfo
flags  : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv svm_lock nrip_save
flags  : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv svm_lock nrip_save
```

BIOS 設定内のファームウェア設定で仮想化拡張機能を有効にします。詳細は、お使いのハードウェアの資料を参照してください。

クライアント URI 設定の確認

クライアントの URI が適切に設定されていることを確認します。

```
# virsh uri
vbox:///system
```

たとえば、このメッセージは URI が **QEMU** ではなく **VirtualBox** ハイパーバイザーに接続されていることを示し、URI の設定エラーであることがわかります。本来は **QEMU** ハイパーバイザーへの接続として設定されているはずです。URI が **QEMU** に正常に接続している場合は、メッセージは以下のようになります。

```
# virsh uri
qemu:///system
```

他のハイパーバイザーが存在し、**libvirt** がこのハイパーバイザーとデフォルトで通信する場合に、この状況が発生します。

解決方法

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します。

```
# virsh list --all
```

A.19.10. 一般的な XML エラー

libvirt では、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 ドキュメントが API を介して **libvirt** に渡されると、さまざまな一般的なエラーが発生します。エラーのある XML タグ、不適切な値、欠落している要素など、一般的な XML エラーの一部を以下に示します。

A.19.10.1. ドメイン定義の編集

これは推奨されていませんが、ゲスト仮想マシン (またはドメインの XML ファイル) を手動で編集することが必要になる場合があります。ゲストの XML にアクセスして編集するには、次のコマンドを使用します。

```
# virsh edit name_of_guest.xml
```

このコマンドは、ゲスト仮想マシンの現在の定義を持つテキストエディターでファイルを開きます。編集を終了して変更を保存したら、**libvirt** により XML が再読み込みされ、解析されます。XML が正しい場合は、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場合は、すべての変更を保存してから、エディターを終了します。

XML ファイルを保存したら、**xmllint** コマンドを使用して XML の形式が正しいことを確認するか、**virt-xml-validate** コマンドを使用して使用上の問題を確認します。

```
# xmllint --noout config.xml
```

```
# virt-xml-validate config.xml
```

エラーが返されない場合、XML 記述は正しく設定され、**libvirt** スキーマと一致します。スキーマがすべての制約を把握していない場合は、報告されたエラーを修正するとさらにトラブルシューティングが行われます。

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、ゲストの状態と設定の定義が記載されています。これらのドキュメントは自動的に生成されるため、手動で編集しないでください。これらのドキュメントのエラーには、破損したドキュメントのファイル名が含まれています。ファイル名は、URI により定義された

ホストマシンでのみ有効です。このホストマシンでは、コマンドが実行されたマシンが表示される場合があります。

libvirt が作成したファイルでエラーが発生することはまれです。ただし、このエラーの原因の 1 つとして、**libvirt** のダウングレードが考えられます。新しいバージョンの **libvirt** では、古いバージョンで生成された XML を読み取ることができるのに対し、古いバージョンの **libvirt** では、新しいバージョンで追加された XML 要素により混同される可能性があります。

A.19.10.2. XML 構文エラー

構文エラーは XML パーサーで検出されます。エラーメッセージには、問題を特定するための情報が記載されています。

XML パーサーのこのエラーメッセージの例は 3 行で設定されています。最初の行はエラーメッセージを表し、以降の 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ます。3 行目には、エラーがその上の行のどこにあるかを示すインジケーターが含まれています。

```
error: (name_of_guest.xml):6: StartTag: invalid element name
<vcpu>2</vcpu><
-----------------^
```

このメッセージに含まれる情報

(*name_of_guest.xml*)

エラーが含まれる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る XML ドキュメントを説明するシンボリック名であり、ディスク上のファイルに直接対応しません。括弧内に含まれないファイル名は、接続のターゲットにあるローカルファイルです。

6

エラーが含まれる XML ファイルの行番号です。

StartTag: 無効な要素名

これは、特定の XML エラーを説明する**libxml2**パーサーからのエラーメッセージです。

A.19.10.2.1. ドキュメントの迷子の <

現象

以下のエラーが発生します。

```
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、パーサーがゲストの XML ファイルの 6 行目の**<** 記号の後に新しい要素名を想定していることを示しています。

テキストエディターで行番号の表示が有効になっていることを確認します。XML ファイルを開き、6 行目のテキストを探します。

```
<domain type='kvm'>
  <name>name_of_guest</name>
<memory>524288</memory>
<vcpu>2</vcpu><
```

ゲストの XML ファイルのこのスニペットには、ドキュメントに余分な**<**が含まれています。

解決方法

余分な**<**を削除するか、新しい要素を終了します。

A.19.10.2.2. 終了していない属性

現象

以下のエラーが発生します。

```
error: (name_of_guest.xml):2: Unescaped '<' not allowed in attributes values
<name>name_of_guest</name>
--^
```

調査

ゲストの XML ファイルのこのスニペットには、終了していない要素の属性値が含まれます。

```
<domain type='kvm>
<name>name_of_guest</name>
```

この例では、**'kvm'**に 2 番目の引用符が欠けています。属性値は、XML の開始タグおよび終了タグと同様に、引用符やアポストロフィで開いたり閉じたりする必要があります。

解決方法

すべての属性値文字列を正しく開いたり閉じたりします。

A.19.10.2.3. タグの開始と終了の不一致

現象

以下のエラーが発生します。

```
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、問題のあるタグを識別するためのヒントが 3 つ含まれています。

最後のコロンに続くメッセージ **clock line 16 and domain** は、**<clock>** にはドキュメントの 16 行目に一致しないタグが含まれてい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のポインターで、2 番目の問題のあるタグを識別します。

ペアになっていないタグは、**/>** で閉じる必要があります。以下のスニペットはこのルールに従わず、上記のエラーメッセージが表示されます。

```
<domain type='kvm'>
  ...
    <clock offset='utc'>
```

このエラーは、ファイル内で XML タグが一致しないために発生します。すべての XML タグに、一致する開始タグと終了タグが必要です。

一致しない XML タグのその他の例

以下の例では、同様のエラーメッセージと、一致しない XML タグのバリエーションを示しています。

終了タグ (**</name>**) がないため、このスニペットに **<features>** の不一致エラーが含まれています。

```
<domain type='kvm'>
 ...
 <features>
   <acpi/>
   <pae/>
 ...
 </domain>
```

このスニペットには、対応する開始タグのない終了タグ (**</name>**) が含まれます。

```
<domain type='kvm'>
  </name>
  ...
</domain>
```

解決方法

すべての XML タグが正しく開始および終了していることを確認します。

A.19.10.2.4. タグの書式エラー

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、簡単なタイプミスが原因で簡単に発生します。このエラーメッセージでは、XML エラー (この例では **type** という単語の真ん中に余計な空白があります) をポインターで強調表示します。

```
<domain ty pe='kvm'>
```

これらの XML の例は、特殊文字の欠落や追加の文字などの誤植が原因で正しく解析されません。

```
<domain type 'kvm'>
```

```
<dom#ain type='kvm'>
```

解決方法

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み、ポインターでエラーを特定します。XML を修正し、変更を保存します。

A.19.10.3. ロジックおよび設定エラー

フォーマットが適切な XML ドキュメントには、構文は正しいものの、**libvirt** が解析できないエラーが含まれる場合があります。このようなエラーは多数存在します。以下では、最も一般的な 2 つのケースについて説明しています。

A.19.10.3.1. バニッシュ部分

現象

加えた変更の一部が表示されず、ドメインの編集または定義後も反映されません。**define** または **edit** コマンドは機能しますが、再度 XML をダンプすると変更が消えます。

調査

このエラーは、libvirt が解析しない設定または構文の破損が原因で発生します。**libvirt** ツールは通常、認識している設定のみを探しますが、その他のすべての設定は無視します。その結果、**libvirt** が入力を解析すると、XML の変更の一部が消えます。

解決方法

XML 入力を検証してから、**edit** コマンドまたは **define** コマンドに渡します。**libvirt** 開発者は、**libvirt** にバンドルされた XML スキーマのセットを維持します。このセットは、**libvirt** が使用する XML ドキュメントで許可されている設定の大半を定義します。

以下のコマンドを使用して、**libvirt** XML ファイルを検証します。

```
# virt-xml-validate libvirt.xml
```

このコマンドが成功すると、**libvirt** は、おそらく XML のすべての設定を理解します。ただし、スキーマが、指定されたハイパーバイザーに対してのみ有効なオプションを検出できない場合を除きます。たとえば、**virsh dump** コマンドの結果として **libvirt** が生成した XML は、エラーなしで検証する必要があります。

A.19.10.3.2. ドライブのデバイスタイプが間違っている

現象

CD-ROM 仮想ドライブのソースイメージの定義は追加されていますが、ここには記載されていません。

```
# virsh dumpxml domain
<domain type='kvm'>
  ...
  <disk type='block' device='cdrom'>
```

```
    <driver name='qemu' type='raw'/>
    <target dev='hdc' bus='ide'/>
    <readonly/>
  </disk>
  ...
</domain>
```

解決方法

欠落している **<source>** パラメーターを次のように追加して、XML を修正します。

```
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source file='/path/to/image.iso'/>
  <target dev='hdc' bus='ide'/>
  <readonly/>
</disk>
```

type='block' ディスクデバイスでは、ソースが物理デバイスであることが想定されます。イメージファイルでディスクを使用する場合は、代わりに **type='file'** を使用します。

付録B 複数のアーキテクチャー上での KVM 仮想化の使用

デフォルトでは、Red Hat Enterprise Linux 7 での KVM 仮想化は AMD64 および Intel 64 アーキテクチャーと互換性を維持します。しかし、Red Hat Enterprise Linux 7.5 以上では kernel-alt パッケージが導入されたため、以下のアーキテクチャーもサポートされるようになりました。

- IBM POWER
- IBM Z
- ARM システム (非対応)

これらのアーキテクチャー上で仮想化を使用する場合、インストール、使用方法、および機能のサポートは若干 AMD64 や Intel 64 とは異なることに注意してください。詳細は、以下を参照してください。

B.1. IBM POWER SYSTEMS での KVM 仮想化の使用

Red Hat Enterprise Linux 7.5 より、IBM POWER8 Systems および IBM POWER9 Systems で KVM 仮想化がサポートされるようになりました。ただし、IBM POWER8 はkernel-alt を使用しません。つまり、この 2 つのアーキテクチャーは特定の側面で異なります。

インストール
IBM POWER8 および POWER9 Systems 向けの Red Hat Enterprise Linux 7 に KVM 仮想化をインストールするには、以下を行います。

1. カスタマーポータルのブート可能なイメージからホストシステムをインストールします。
   - IBM POWER8
   - IBM POWER9

   詳細な手順については、Red Hat Enterprise Linux 7 インストールガイド を参照してください。
2. お使いのホストシステムがハイパーバイザーの要件を満たしているようにしてください。
   - お使いのマシンが適切なタイプであることを確認します。

     ```
     # grep ^platform /proc/cpuinfo
     ```

     このコマンドの出力には、サポートされる PowerNV マシンタイプで実行されていることを示す **PowerNV** エントリーが含まれている必要があります。

     ```
     platform        : PowerNV
     ```

   - KVM-HV カーネルモジュールをロードします。

     ```
     # modprobe kvm_hv
     ```

   - KVM-HV カーネルモジュールがロードされたことを確認します。

     ```
     # lsmod | grep kvm
     ```

     KVM-HV が正常にロードされた場合は、このコマンドの出力に **kvm_hv** が含まれます。

3. 2章 *仮想化パッケージのインストール* で説明されているその他の仮想化パッケージに加え、qemu-kvm-ma パッケージをインストールします。

アーキテクチャーの関連事項

IBM POWER 向けの Red Hat Enterprise Linux 7.5 上の KVM 仮想化は、AMD64 や Intel 64 システムの KVM とは以下の点が異なります。

- IBM POWER ホスト上のゲストに推奨される**最低メモリー容量** の割り当ては **2GB の RAM** です。
- IBM POWER システムでは、SPICE プロトコルに対応していません。ゲストのグラフィカル出力を表示するには、VNC プロトコルを使用します。さらに、次の仮想 グラフィックスカードデバイス にのみ対応しています。
  - **vga** - **-vga std** モードでのみサポートされ、**-vga cirrus** モードではサポートされません。
  - **virtio-vga**
  - **virtio-gpu**
- 以下の仮想化機能は、AMD64 および Intel 64 ホストでは無効になっていますが、IBM POWER 上では動作します。しかし、Red Hat ではサポートしていないため、使用は推奨されません。
  - I/O スレッド
- SMBIOS 設定は利用できません。
- 互換モードのゲストを含む POWER8 ゲストは、以下のようなエラーで起動に失敗することがあります。

  ```
  qemu-kvm: Failed to allocate KVM HPT of order 33 (try smaller maxmem?): Cannot allocate memory
  ```

  これは、Red Hat Enterprise Linux 7.3 またはそれ以前を使用するゲストで発生する可能性が高くなります。

  この問題を解決するには、ホストのカーネルコマンドラインに **kvm_cma_resv_ratio=*memory*** を追加して、ゲストのハッシュページテーブル (HPT) に使用できる CMA メモリープールを増やします。*memory* は、CMA プールに予約する必要があるホストメモリーの割合 (デフォルトは 5) です。
- 現在、THP (transparent huge page) を利用しても IBM POWER8 ゲストのパフォーマンスは著しく改善されません。

  また、IBM POWER8 Systems の静的な ヒュージページ のサイズは 16MiB と 16GiB で、IBM POWER9 の AMD64 および Intel 64 は 2MiB および 1GiB であることに注意してください。そのため、ゲストに静的なヒュージページが設定されている場合、ゲストを IBM POWER8 ホストから IBM POWER9 ホストに移行できません。

  さらに、IBM POWER8 ゲストで 静的なヒュージページまたは THP を使用 できるようにするには、最初に ホスト上でヒュージページを設定 する必要があります。
- AMD64 および Intel64 システムでサポートされる仮想 周辺デバイス の一部は、IBM POWER Systems ではサポートされず、異なるデバイスが代替としてサポートされます。

- **ioh3420** や **xio3130-downstream** デバイスを含む PCI-E 階層に使用されるデバイスはサポートされません。この機能は、**spapr-pci-host-bridge** デバイスが提供する複数の独立した PCI root ブリッジに置き換えられます。
- UHCI コントローラーおよび EHCI PCI コントローラーには対応していません。代わりに OHCI コントローラーおよび xHCI コントローラーを使用してください。
- 仮想 IDE CD-ROM (**ide-cd**) および仮想 IDE ディスク (**ide-hd**) を含む IDE デバイスはサポートされません。代わりに **virtio-scsi** デバイスおよび **virtio-blk** デバイスを使用します。
- エミュレートされた PCI NIC (**rtl8139**) はサポートされません。代わりに **virtio-net** デバイスを使用します。
- **intel-hda**、**hda-output**、および **AC97** を含むサウンドデバイスはサポートされません。
- **usb-redir**、**usb-tablet** などの USB リダイレクトデバイスはサポートされません。

- **kvm-clock** サービスを IBM POWER システムの 時間管理 のために設定する必要はありません。
- **pvpanic** デバイスは IBM POWER システムではサポートされません。しかし、このアーキテクチャーではデフォルトで有効になっている同等の機能を使用できます。この機能をゲストで有効にするには、**<on_crash>** 設定要素で **preserve** の値を使用します。さらに、**<devices>** セクションの **<panic>** 要素を削除するようにしてください。 削除しないと、ゲストが IBM POWER システムで起動しない原因となります。
- IBM POWER8 システムでは、ゲストに対応するために、ホストマシンをシングルスレッドモードで稼働する必要があります。これは、qemu-kvm-ma パッケージがインストールされていると、自動的に設定されます。シングルスレッドのホストで実行しているゲストは複数のスレッドを使用することができます。
- RHEL 7 ホストで実行している IBM POWER 仮想マシンが、ゼロメモリー (**memory='0'**) を使用する NUMA ノードで設定されていると、仮想マシンが正しく動作しません。したがって、Red Hat は、RHEL 7 ではゼロメモリー NUMA ノードを持つ IBM POWER 仮想マシンに対応しません。

B.2. IBM Z での KVM 仮想化の使用

インストール

IBM Z ホストでは、ハイパーバイザーを専用の論理パーティション (LPAR) にインストールする必要があります。z/VM OS における KVM の実行はサポートされていません。また、LPAR は *SIE* (start-interpretive execution) 仮想化拡張をサポートする必要もあります。

Red Hat Enterprise Linux 7 for IBM Z に KVM 仮想化をインストールするには、以下を行います。

1. カスタマーポータルのブート可能イメージ からホストシステムをインストールします。詳細な手順は、インストールガイドを参照してください。
2. お使いのシステムがハイパーバイザーの要件を満たしているようにしてください。
   - CPU の仮想化拡張が利用できることを確認します。

     ```
     # grep sie /proc/cpuinfo
     ```

このコマンドの出力には、プロセッサーに必要な仮想化拡張があることを示す **sie** エントリーが含まれている必要があります。

```
features	: esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te sie
```

- KVM カーネルモジュールを読み込みます。

```
# modprobe kvm
```

- KVM カーネルモジュールが読み込まれていることを確認します。

```
# lsmod | grep kvm
```

KVM が正常にロードされた場合は、このコマンドの出力に **kvm** が含まれます。そうでない場合は、Red Hat Enterprise Linux 7 のカーネルのkernel-altバージョンを使用していることを確認します。

3. 2章 *仮想化パッケージのインストール* で説明されているその他の仮想化パッケージに加え、qemu-kvm-ma パッケージをインストールします。

4. ゲストの設定時、ゲストを "Spectre" 脆弱性から保護するために、以下の方法の 1 つで CPU を設定 することが推奨されます。

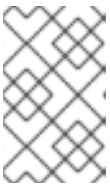
- 以下のように、ホストの CPU モデルを使用します。

```
<cpu mode='host-model' check='partial'>
  <model fallback='allow'/>
</cpu>
```

これにより、ホストが **ppa15** 機能および **bpb** 機能をサポートしている場合は、ゲストでも利用できます。

- 特定のホストモデルを使用している場合、**ppa15** および **bpb** 機能を追加します。次の例では、**zEC12** CPU モデルを使用します。

```
<cpu mode='custom' match='exact' check='partial'>
    <model fallback='allow'>zEC12</model>
    <feature policy='force' name='ppa15'/>
    <feature policy='force' name='bpb'/>
</cpu>
```



注記

z12 ホストマシンで **z114** および **z196** CPU モデルで **ppa15** 機能を使用すると、最新のマイクロコードレベル (バンドル 95 またはそれ以降) を使用するようにしてください。

アーキテクチャーの関連事項

IBM Z 向けの Red Hat Enterprise Linux 7.5 上の KVM 仮想化は、AMD64 や Intel 64 システムの KVM とは以下の点が異なります。

- IBM Z では SPICE および VNC プロトコルは利用できず、仮想グラフィカルカードデバイス はサポートされません。そのため、ゲストのグラフィカル出力は表示できません。
- IBM Z は、仮想 PCI デバイスおよび USB デバイスをサポートしていません。したがって、*virtio-*-pci* デバイスはサポートされていないため、代わりに ***virtio-*-ccw*** デバイスを使用してください。たとえば、**virtio-net-pci** の代わりに **virtio-net-ccw** を使用します。
- <boot dev='*device*'/> XML 設定要素は IBM Z ではサポートされていません。デバイスの起動順序を定義するには、<devices> セクションの <boot order='*number*'/> 要素を使用します。アップストリームの libvirt ドキュメント を参照してください。

注記

AMD64 および Intel 64 ホストでも、起動順序の管理に <boot order='*number*'/> の使用が推奨されます。

- SMBIOS 設定は利用できません。
- IBM Z で使用される ウォッチドックデバイス モデルは **diag288** である必要があります。
- IBM Z で 入れ子になった仮想化 機能を有効にするには、以下を行います。入れ子になった仮想化は、IBM Z では AMD64 および Intel 64 システムと同様にテクノロジープレビューとして利用できるため、本番環境での使用は推奨されていないことに注意してください。

1. 1. 入れ子になった仮想化がシステムで利用できるかどうかを確認します。

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

このコマンドによって **1** が返された場合、この機能は有効になっています。

このコマンドによって **0** が返された場合、以下の手順に従って有効にします。

2. **kvm** モジュールをアンロードします。

```
# modprobe -r kvm
```

3. ネスト機能をアクティブにします。

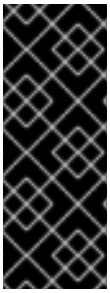
```
# modprobe kvm nested=1
```

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

```
options kvm nested=1
```

- **kvm-clock** サービスは、AMD64 システムおよび Intel 64 システムに固有のものであり、IBM Z の 時間管理 用に設定する必要はありません。

B.3. ARM システムでの KVM 仮想化の使用



重要

KVM 仮想化は、64 ビット ARM アーキテクチャー向けに、Red Hat Enterprise Linux 7.5 以降で提供されます。そのため、ARM システムの KVM 仮想化は Red Hat ではサポートされておらず、実稼働環境での使用を目的としていません。そのため、既知のセキュリティー脆弱性に対応していない可能性があります。また、ARM における KVM 仮想化は急速な開発段階であるため、以下の情報が正確または完全であることを保証できません。

インストール

ARM 向けに Red Hat Enterprise Linux 7.5 に仮想化をインストールするには、以下を行います。

1. カスタマーポータルのブート可能なイメージからホストシステムをインストールします。
2. システムがインストールされた後、次のコマンドを使用して、システム上に仮想化スタックをインストールします。

```
# yum install qemu-kvm-ma libvirt libvirt-client virt-install AAVMF
```

正常にインストールするには、**Optional** チャンネルが有効になっている必要があります。

アーキテクチャーの関連事項

64 ビット ARM アーキテクチャー向けの Red Hat Enterprise Linux 7.5 上の KVM 仮想化は、AMD64 や Intel 64 システムの KVM とは以下の点が異なります。

- PXE ブートは、**virtio-net-device** および **virtio-net-pci** ネットワークインターフェイスコントローラー (NIC) でのみサポートされます。また、PXE ブートには ARM Architecture Virtual Machine Firmware (AAVMF) のビルトイン **VirtioNetDxe** ドライバーを使用する必要があります。iPXE オプションの ROM はサポートされません。
- 最大 123 個の仮想 CPU (vCPU) のみを単一のゲストに割り当てできます。

付録C 仮想化の制限

本付録では、Red Hat Enterprise Linux 7 における仮想化パッケージに関する追加のサポートと製品の制限について説明します。

C.1. システムの制限

ホストシステム

KVM を使用する Red Hat Enterprise Linux は、以下のホストアーキテクチャーでのみサポートされます。

- AMD64 および Intel 64
- IBM Z
- IBM POWER8
- IBM POWER9

本書は、主に AMD64 および Intel 64 の機能について説明しますが、他のサポート対象のアーキテクチャーはほぼ同じように機能します。詳細は、付録B *複数のアーキテクチャー上での KVM 仮想化の使用* を参照してください。

ゲストシステム

Red Hat Enterprise Linux 7 では、Microsoft Windows ゲスト仮想マシンは、AMC (Advanced Mission Critical) などの特定のサブスクリプションプログラムでのみサポートされます。サブスクリプションモデルに Windows ゲストのサポートが含まれているかどうか不明な場合は、カスタマーサポートにお問い合わせください。

Red Hat Enterprise Linux 7 上の Windows ゲスト仮想マシンの詳細は、Windows Guest Virtual Machines on Red Hat Enterprise Linux 7 のナレッジベースの記事 を参照してください。

C.2. 機能の制限

Red Hat Enterprise Linux に同梱されるハイパーバイザーパッケージは qemu-kvm です。これは、Red Hat Virtualization (RHV) および Red Hat OpenStack (RHOS) 製品に同梱される qemu-kvm-rhev パッケージとは異なります。qemu-kvm に適用される制限の多くは、qemu-kvm-rhev には適用されません。

qemu-kvm パッケージと qemu-kvm-rhev パッケージの相違点の詳細は、What are the differences between qemu-kvm and qemu-kvm-rhev and all sub-packages? を参照してください。

Red Hat Enterprise Linux に同梱される KVM ハイパーバイザーには、以下の制限が適用されます。

ゲストごとの vCPU の最大数

Red Hat Enterprise Linux 7.2 以降は、ゲストごとに 240 個の vCPU をサポートしています (Red Hat Enterprise Linux 7.0 の 160 個から増加)。

ネストされた仮想化

ネストされた仮想化は、Red Hat Enterprise Linux 7.2 以降ではテクノロジープレビューとして利用できます。この機能により、KVM はハイパーバイザーとして機能し、独自のゲストを作成できるゲストを起動できます。

TCG のサポート

QEMU および **libvirt** には、QEMU Tiny Code Generator (TCG) を使用した動的な変換モードが含まれます。このモードでは、ハードウェアの仮想化のサポートは必要ありません。ただし、TCG は Red Hat ではサポートされていません。

qemu-kvm パッケージーを使用して仮想マシンにネストされたゲストを作成する場合は、親仮想マシンでネストされた仮想化が有効になっていない限り TCG を使用します。ネストされた仮想化は現時点でテクノロジープレビューであることに注意してください。詳細は、12章*Nested Virtualization* を参照してください。

TCG ベースのゲストは、次のコマンドを使用して認識できます。

- ゲストのドメイン XML ファイルには **<domain type='qemu'>** 行が含まれますが、KVM ゲストには **<domain type='kvm'>** が含まれます。
- 仮想ハードウェアの詳細ビューの概要ペインでは、**virt-manager** は、**KVM** ではなく、**QEMU TCG** として仮想マシンのタイプを表示します。

定数 TSC ビット

Constant TSC (Time Stamp Counter) がないシステムには、追加の設定が必要です。Constant タイムスタンプカウンターがあるかどうかの判断と、関連する問題を修正するための設定手順は、8章*KVM ゲストのタイミング管理* を参照してください。

エミュレートされた SCSI アダプター

SCSI デバイスエミュレーションは、virtio-scsi 準仮想化ホストバスアダプター (HBA) でのみ対応しています。エミュレートされた SCSI HBA は、Red Hat Enterprise Linux では KVM に対応していません。

エミュレートされた IDE デバイス

KVM は、仮想マシンごとに、最大 4 つの仮想 (エミュレート) IDE デバイスに制限されます。

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、純粋に仮想デバイスで、仮想マシンで最適に機能するように設計されています。

Red Hat Enterprise Linux 7 は、仮想マシンバスごとに 32 の PCI デバイススロットをサポートし、デバイススロットごとに 8 つの PCI 機能をサポートします。これにより、仮想マシンで多機能の性能が有効になり、PCI ブリッジが使用されていない場合に、理論上は 1 つのバスあたり最大 256 個の PCI 機能が提供されます。各 PCI ブリッジは新しいバスを追加します。これにより、別の 256 個のデバイスアドレスが有効になる可能性があります。ただし、一部のバスでは、256 個のデバイスアドレスすべてがユーザーに利用できるようになっていません。たとえば、ルートバスには、スロットを占有する複数の組み込みデバイスがあります。

デバイスの詳細は 16章 *ゲスト仮想マシンのデバイス設定* を、PCI ブリッジの詳細は 「PCI ブリッジ」 を参照してください。

移行の制限

デバイスの割り当ては、仮想マシンに公開されている物理デバイスを、その仮想マシン専用で使用することを指します。デバイスの割り当ては、仮想マシンが実行される特定のホストのハードウェアを使用するため、デバイスの割り当てを使用中の際は、移行と保存/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に対応している場合は、移行前に割り当てたデバイスを削除し、この機能を有効にするために保存/復元を行うことができます。

ライブマイグレーションは、同じ CPU タイプ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、両方のホストで、NX (No eXecution) ビットが同じ値 (**on** または **off** のいずれか) に設定され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いているすべてのブロックデバイスに **cache=none** を指定する必要があります。

警告

cache=none を指定しないと、ディスクが破損する可能性があります。

ストレージの制限

ゲスト仮想マシンに、ディスク全体またはブロックデバイス (**/dev/sdb** など) への書き込みアクセスを提供すると、それに関連するリスクが発生します。ゲスト仮想マシンがブロックデバイス全体にアクセスできる場合は、ホストマシンとボリュームラベルまたはパーティションテーブルを共有できます。ホストシステムのパーティション認識コードにバグが存在すると、セキュリティーリスクが発生する可能性があります。ゲスト仮想マシンに割り当てられ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、このリスクを回避します。

警告

ストレージの制限に従わないと、セキュリティーのリスクが発生する可能性があります。

ライブスナップショット

Red Hat Enterprise Linux の KVM のバックアップおよび復元の API は、ライブスナップショットに対応していません。

ストリーミング、ミラーリング、およびライブマージ

ストリーミング、ミラーリング、およびライブマージには対応していません。これにより、ブロックジョブが阻止されます。

I/O スロットリング

Red Hat Enterprise Linux は、仮想ディスクでの操作の最大入力および出力レベルの設定をサポートしていません。

I/O スレッド

Red Hat Enterprise Linux は、VirtIO インターフェイスを備えたディスクでの入出力操作用の個別のスレッドの作成をサポートしていません。

メモリーのホットプラグおよびホットアンプラグ

Red Hat Enterprise Linux は、仮想マシンからのメモリーのホットプラグまたはホットアンプラグをサポートしていません。

vhost-user

Red Hat Enterprise Linux は、ユーザースペース vhost インターフェイスの実装をサポートしていません。

CPU のホットアンプラグ

Red Hat Enterprise Linux は、仮想マシンからの CPU のホットアンプラグをサポートしていません。

PCIe の NUMA ゲストの場所

Red Hat Enterprise Linux は、仮想 PCIe デバイスを特定の NUMA ノードにバインドすることには対応していません。

コアダンプの制限

現在、コアダンプは移行に実装されているため、デバイスの割り当てが使用されている場合はサポートされません。

リアルタイムカーネル

現在、KVM はリアルタイムカーネルに対応していないため、Red Hat Enterprise Linux for Real Time では使用できません。

C.3. アプリケーションの制限

仮想化には、特定の種類のアプリケーションには適さない側面があります。

I/O スループットの要件が高いアプリケーションでは、完全に仮想化されたゲストに KVM の準仮想化ドライバー (virtio ドライバー) を使用する必要があります。virtio ドライバーがないと、特定のアプリケーションは、高い I/O 負荷では予測できない場合があります。

I/O 要件が高いため、以下のアプリケーションは使用しないでください。

- **kdump**サーバー
- **netdump**サーバー

I/O を多用したり、リアルタイムパフォーマンスを必要としたりするアプリケーションやツールを慎重に評価する必要があります。I/O パフォーマンスを向上させるには、virtio ドライバーまたは PCI デバイスの割り当てを検討してください。完全に仮想化したゲストの virtio ドライバーの詳細は、*5章KVM 準仮想化 (virtio) ドライバー* を参照してください。PCI デバイスの割り当ての詳細は、*16章 ゲスト仮想マシンのデバイス設定* を参照してください。

仮想環境でアプリケーションを実行すると、パフォーマンスがわずかに低下します。新しいハードウェアおよび高速ハードウェアへの統合による仮想化のパフォーマンス上の利点は、仮想化の使用に伴う潜在的なアプリケーションパフォーマンスの問題に対して評価する必要があります。

C.4. その他の制限

仮想化に影響するその他すべての制限および問題のリストについては、『Red Hat Enterprise Linux 7 Release Notes』 を参照してください。『Red Hat Enterprise Linux 7 リリースノート 』 では、現在の新機能、既知の問題、および更新または検出された制限について説明しています。

C.5. ストレージのサポート

仮想マシンでは、以下の保存方法に対応しています。

- ローカルストレージのファイル
- 物理ディスクのパーティション
- ローカルに接続されている物理 LUN
- LVM パーティション
- NFS 共有ファイルシステム
- iSCSI
- GFS2 クラスターファイルシステム
- 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)

C.6. USB 3 / XHCI のサポート

USB 3(xHCI)USB ホストアダプターエミュレーションは、Red Hat Enterprise Linux 7.2 以降でサポートされています。すべての USB 速度が利用可能です。つまり、あらゆる種類の USB デバイスを xHCI バスにプラグインできます。また、コンパニオンコントローラー (USB 1 デバイス用) は必要ありません。ただし、USB 3 バルクストリームには対応していません。

xHCI の利点:

- 仮想化互換のハードウェア設計。つまり、xHCI エミュレーションではポーリングオーバーヘッドが低減するため、以前のバージョンよりも少ない CPU リソースが必要になります。
- USB 3 デバイスの USB パススルーが利用できます。

xHCI の制限事項:

- Red Hat Enterprise Linux 5 ゲストではサポートされません。

xHCI デバイスのドメイン XML デバイスについては、図16.19「USB3/xHCI デバイスのドメイン XML の例」 を参照してください。

付録D 関連情報

仮想化および Red Hat Enterprise Linux の詳細は、以下のリソースを参照してください。

D.1. オンラインリソース

- http://www.libvirt.org/ は、**libvirt** 仮想化 API のアップストリーム公式 Web サイトです。
- https://virt-manager.org/ は、仮想マシンを管理するグラフィカルアプリケーションである **Virtual Machine Manager** (virt-manager) のアップストリームプロジェクトの Web サイトです。
- Red Hat Virtualization - http://www.redhat.com/products/cloud-computing/virtualization/
- Red Hat 製品ドキュメント - https://access.redhat.com/documentation/en/
- 仮想化テクノロジーの概要 - http://virt.kernelnewbies.org

D.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ーのサブコマンドとオプション、および **libvirt** 仮想ライブラリー API の包括的な情報が含まれます。
- **/usr/share/doc/gnome-applet-vm-*version-number*** - ローカルで実行している仮想マシンを監視および管理する GNOME グラフィカルパネルアプレットのドキュメント。
- **/usr/share/doc/libvirt-python-*version-number*** - **libvirt** ライブラリーの Python バインディングの詳細を説明します。**libvirt-python** パッケージにより、python の開発者は、**libvirt** 仮想化管理ライブラリーとインターフェイスするプログラムを作成できます。
- **/usr/share/doc/virt-install-*version-number*** - 仮想マシン内で Fedora および 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際に役に立つ、**virt-install** コマンドに関するドキュメントを提供します。
- **/usr/share/doc/virt-manager-*version-number*** - Virtual Machine Manager に関するドキュメントを提供します。このドキュメントは、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。
- 

  注記

  その他の Red Hat Enterprise Linux コンポーネントの詳細は、**usr/share/doc/** の適切な **man** ページまたはファイルを参照してください。

付録E IOMMU グループの使用[1]

Red Hat Enterprise Linux 7 で導入された *VFIO* (Virtual Function I/O) は、ユーザー空間のドライバーフレームワークを提供する一連の Linux カーネルモジュールです。このフレームワークにより、ユーザー空間ドライバーのセキュアなデバイスアクセスを有効にするために IOMMU (Input–output memory management unit) 保護が使用されます。VFIO は、*Data Plane Development Kit (DPDK)* やより一般的な PCI デバイス割り当て などのユーザー空間ドライバーを有効にします。

VFIO は IOMMU グループを使用してデバイスを分離し、ホスト物理マシン上で実行される 2 つのデバイス間で意図しない *DMA (Direct Memory Access、直接メモリーアクセス)* が発生しないようにします。意図しない DAM はホストとゲストの機能に影響を与える可能性があります。Red Hat Enterprise Linux 7 で利用できる IOMMU グループは、Red Hat Enterprise Linux 6 で利用可能なレガシー KVM デバイス割り当てよりも大幅に改良されています。この付録では、以下を重点的に取り上げます。

- IOMMU グループの概要
- デバイス分離の重要性
- VFIO の利点

E.1. IOMMU の概要

IOMMU はデバイスの仮想アドレス空間を作成します。ここで、各 I/O Virtual Address (IOVA) は物理システムメモリーの異なるアドレスに変換されます。この変換が完了すると、デバイスは物理システムのメモリー内の異なるアドレスに接続されます。OMMU がない場合、すべてのデバイスは、メモリーアドレス変換がないために物理メモリーの共有されたフラットビューを持ちます。IIOMMU がある場合、デバイスは、デバイスの割り当てに役立つ新規のアドレス空間である IOVA 空間を受信します。

IOMMU が異なると機能レベルも異なります。過去において、IOMMU は変換のみを提供することに制限され、アドレス空間の小規模な枠にのみ使用されることが多くありました。たとえば、IOMMU は複数デバイスで共有されるローメモリー (low memory) の IOVA 空間の小規模な枠 (1 GB 以下) のみを予約しました。AMD GART (graphics address remapping table) が汎用 IOMMU として使用される場合がこのモデルの例になります。これらの従来の IOMMU は、*バウンスバッファー (bounce buffer)* および *アドレスコアレッシング (address coalescing)* という 2 つの機能を主に提供してきました。

- **バウンスバッファー**は、デバイスのアドレス機能がプラットフォームの同機能よりも小さい場合に必要になります。たとえば、デバイスのアドレス空間がメモリーの 4 GB (32 ビット) に制限され、ドライバーが 4 GB を超えるバッファーに割り当てようとする場合に、デバイスはバッファーに直接アクセスできなくなります。このような状況ではバウンスバッファーを使用する必要があります。これは下部メモリーに位置するバッファー容量です。デバイスが DMA 操作を行うことができます。ローメモリー (low memory) にあるバッファー領域では、デバイスが DMA 操作を実行できます。バッファーのデータは操作の完了時にドライバーの割り当て済みバッファーにのみコピーされます。つまり、バッファーはローメモリー (low memory) アドレスからハイメモリー (high memory) アドレスにバウンスされます。IOMMU は、バッファーがデバイスの物理アドレスデバイス容量を超えても、アドレス空間内で IOVA 変換を提供してバウンスバッファーを防止します。これにより、デバイスは、バッファーがデバイスの物理アドレス空間を超えても DMA 操作を直接バッファーで実行します。 これまでこの IOMMU 機能は、IOMMU での使用に制限されてきましたが、*PCI-Express* (PCIe) の導入により、4GB を超えるアドレスのサポート機能がレガシー以外のすべてのエンドポイントで必要になりました。

- 従来のメモリー割り当てでは、メモリーのブロックはアプリケーションのニーズに応じて割り当てや解放が実行されます。この方法を使用することにより、物理アドレス空間全体に分散するメモリーギャップが生成されます。メモリーギャップの**コアレッシング**、つまり簡単に言えばメモリーギャップが 1 つにまとめられることにより、メモリーをより効率的に使用できるようにすることが望ましいと言えます。IOMMU は、scatter-gather リストと呼ばれることもある分散したメモリーのリストのコアレッシングを IOVA 空間全体で実行します。これにより、

IOMMU は連続的な DMA 操作を作成し、最終的には I/O パフォーマンスの効率を高めます。最も簡単な例として、ドライバーは物理メモリー領域の連続しない 2 つの 4KB バッファーを割り当てることができます。IOMMU は、これらのバッファーの連続する範囲を割り当て、I/O デバイスが 2 つの 4 KB DMA ではなく単一の 8KB DVA を実行できるようにします。

メモリーコアレッシングおよびバウンスバッファーはホストの高パフォーマンス I/O において重要ですが、仮想化環境に不可欠な IOMMU 機能は最新 IOMMU の **分離機能**です。従来の PCI は要求側のデバイスの ID (リクエスター ID) でトランザクションにタグ付けしないため、PCI-Express 以前には分離を広範囲に実行することはできませんでした。PCI-X にはある程度のリクエスター ID が含まれていましたが、デバイスを相互に接続し、トランザクションの所有権を持たせるルールでは、デバイスを分離するための完全なサポートは提供されませんでした。

PCIe では、各デバイスのトランザクションは、デバイスに固有のリクエスター ID (BDF と省略されることの多い PCI バス/デバイス/機能番号) でタグ付けされ、これはデバイスの固有の IOVA テーブルを参照するために使用されます。分離が可能になることにより、IOVA 空間は接続できないメモリーのオフロードやメモリーコアレッシングなどの変換操作のみならず、デバイスから DMA アクセスを制限するためにも使用できるようになりました。これにより、デバイスを相互に分離し、メモリー領域の重複した割り当てを防ぐことができます。これは適切なゲスト仮想マシンデバイス管理に不可欠です。これらの機能をゲスト仮想マシンで使用することにより、割り当て済みデバイスの IOVA 空間には仮想マシンについてのゲスト物理およびホスト物理メモリーのマッピングが設定されます。これがいったん実行されると、デバイスはゲスト仮想マシンのアドレス空間で DMA 操作を透過的に実行します。

E.2. IOMMU グループの詳細

IOMMU グループは、IOMMU の観点から分離されているとみなされる最も小さなデバイスセットとして定義されます。分離を実行するために必要な最初の手順は、詳細度 (granularity) の使用です。IOMMU がデバイスを別個の IOVA 空間に区分できない場合、それらは分離されません。たとえば、複数のデバイスが同一 IOVA 空間のエイリアスになる場合、IOMMU はそれらを区別することができません。これは、通常の x86 PC がすべての従来型の PCI デバイスに同一リクエスター ID に対するエイリアスを設定してそれらを 1 つにまとめる方法になります (PCIe-to-PCI ブリッジ)。レガシー KVM デバイスの割り当てでは、ユーザーはこれらの従来型 PCI デバイスを別々に割り当てるため、IOMMU はデバイス間を区別できずに設定は失敗します。VFIO は IOMMU グループで管理されるため、VFIO は IOMMU の詳細度の使用というこの最も基本的な要件を満たさない設定を許可しません。

次の手順として、デバイスからのトランザクションが IOMMU に実際に到達するかどうかを判別する必要があります。PCIe 仕様では、トランザクションを相互接続ファブリック内に再度ルート指定できます。PCIe ダウンストリームポートは、あるダウンストリームデバイスから別のデバイスへのトランザクションを再度ルート指定できます。PCIe スイッチのダウンストリームポートは相互に接続し、1 つのポートから別のポートへの再ルート指定を可能にします。マルチファンクションエンドポイントデバイス内でも、1 つの機能からのトランザクションは別のファンクションに直接送信できます。これらの 1 つのデバイスから別のデバイスへのトランザクションはピアツーピアトランザクションと呼ばれ、別々の IOV 空間で稼働しているデバイスの分離を破棄する可能性があります。たとえば、ゲスト仮想マシンに割り当てられているネットワークインターフェイスカードが DMA の書き込み操作を独自の IOV 空間内の仮想アドレスに試行するとします。ただし、物理空間ではその同じアドレスはホストによって所有されているピアディスクコントローラーに属します。この場合、デバイスについての物理変換に対する IOVA は IOMMU のみで実行されるため、トランザクションのデータパスの最適化を試行する相互接続により、ディスクへの DMA 書き込み操作が、変換用に IOMMU に到達する前に間違ってリダイレクトされる可能性があります。

この問題を解決するために、PCI Express 仕様には、これらのリダイレクトの可視性とコントロールを提供する PCIe Access Control Services (ACS) のサポートが含まれます。これは相互接続やマルチファンクションエンドポイントに欠落していることの多い、デバイスを相互に分離するために必要なコンポーネントです。デバイスから IOMMU へのすべてのレベルで ACS サポートがない場合も、リダイレクトが生じることを想定する必要があります。このため、ACS サポートがない場合、PCI トポロジーのすべてのデバイスの分離に影響があります。PCI 環境の IOMMU グループはこの分離を考慮に入れて、変換されないピアツーピア DMA が可能な複数のデバイスを 1 つにグループ化します。

要約すると、IOMMU グループは、IOMMU が可視性を持ち、他のグループから分離される最小単位のデバイスセットを表します。VFIO はこの情報を使用してユーザー空間についてデバイスの安全な所有権を確保します。ブリッジ、root ポートおよびスイッチ (相互接続ファブリックの例すべて) の例外を除き、IOMMU グループ内のすべてのデバイスは VFIO デバイスドライバーにバインドされるか、安全なスタブドライバーとして認識されます。PCI の場合、これらのドライバーは vfio-pci と pci-stub です。pci-stub は、ホストがこのドライバーを介してデバイスと対話していないため簡単に許可されます。[2].VFIO の使用時にグループが実行不可能であることを示すエラーが発生した場合、このグループ内のすべてのデバイスが適切なホストドライバーにバインドする必要があることを意味します。**virsh nodedev-dumpxml** を使用して IOMU グループの設定を調べ、**virsh nodedev-detach** を使用してデバイスを VFIO 互換ドライバーにバインドすると、このような問題を解決できます。

E.3. IOMMU グループの特定および割り当て方法

この例では、ターゲットシステムにある PCI デバイスを特定し、割り当てる方法を示しています。追加の例および情報については 「GPU デバイスの割り当て」 を参照してください。

手順E.1 IOMMU グループ

1. **デバイスのリスト表示**
   virsh nodev-list *device-type* コマンドを実行して、システムのデバイスを特定します。この例では、PCI デバイスを見つける方法を説明します。出力は簡潔にするために省略されています。

```
# virsh nodedev-list pci

pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
[...]
pci_0000_00_1c_0
pci_0000_00_1c_4
[...]
pci_0000_01_00_0
pci_0000_01_00_1
[...]
pci_0000_03_00_0
pci_0000_03_00_1
pci_0000_04_00_0
pci_0000_05_00_0
pci_0000_06_0d_0
```

2. **デバイスの IOMMU グループの特定**
   IOMMU グループなど、リスト表示された各デバイスの詳細は、**virsh nodedev-dumpxml *name-of-device*** コマンドを使用して確認できます。たとえば、pci_0000_04_00_0 という名前の PCI デバイス (PCI アドレス 0000:04:00.0) の IOMMU グループを確認するには、以下のコマンドを使用します。

```
# virsh nodedev-dumpxml pci_0000_04_00_0
```

   このコマンドは、以下に示すのと同様の XML ダンプを生成します。

図E.1 IOMMU グループ XML

```
<device>
  <name>pci_0000_04_00_0</name>
  <path>/sys/devices/pci0000:00/0000:00:1c.0/0000:04:00.0</path>
  <parent>pci_0000_00_1c_0</parent>
  <capability type='pci'>
    <domain>0</domain>
    <bus>4</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10d3'>82574L Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='8'>  <!--This is the element block you will need to use-->
      <address domain='0x0000' bus='0x00' slot='0x1c' function='0x0'/>
      <address domain='0x0000' bus='0x00' slot='0x1c' function='0x4'/>
      <address domain='0x0000' bus='0x04' slot='0x00' function='0x0'/>
      <address domain='0x0000' bus='0x05' slot='0x00' function='0x0'/>
    </iommuGroup>
    <pci-express>
      <link validity='cap' port='0' speed='2.5' width='1'/>
      <link validity='sta' speed='2.5' width='1'/>
    </pci-express>
  </capability>
</device>
```

3. **PCI データの表示**
   上記で取得した出力では、4 つのデバイスを持つ 1 つの IOMMU グループがあります。これは、ACS サポートのない多機能 PCIe ルートポートの例です。スロット 0x1c の 2 つの機能は、PCIe root ポートです。これは、(pciutils パッケージから) **lspci** コマンドを実行することで識別できます。

   ```
   # lspci -s 1c

   00:1c.0 PCI bridge: Intel Corporation 82801JI (ICH10 Family) PCI Express Root Port 1
   00:1c.4 PCI bridge: Intel Corporation 82801JI (ICH10 Family) PCI Express Root Port 5
   ```

   エンドデバイスであるバス 0x04 および 0x05 の 2 つの PCIe デバイスについてこの手順を繰り返します。

   ```
   # lspci -s 4
   04:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network Connection This is used in the next step and is called 04:00.0
   # lspci -s 5 This is used in the next step and is called 05:00.0
   05:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5755 Gigabit Ethernet PCI Express (rev 02)
   ```

4. **エンドポイントのゲスト仮想マシンへの割り当て**
   エンドポイントのいずれかを仮想マシンに割り当てるには、現時点で割り当てていないエンドポイントを VFIO 互換ドライバーにバインドして、IOMMU グループがユーザーおよびホストドライバー間で分離されないようにします。たとえば、上記で受信される出力では 04:00.0 のみ

を持つ仮想マシンを設定しようとしますが、マシンは 05:00.0 がホストドライバーから分離されない限り、起動に失敗します。05:00.0 の割り当てを解除するには、root で **virsh nodedev-detach** コマンドを実行します。

```
# virsh nodedev-detach pci_0000_05_00_0
Device pci_0000_05_00_0 detached
```

両方のエンドポイントを仮想マシンに割り当ててこの問題を解決することもできます。**<hostdev>** 要素の **managed** 属性に yes 値を使用すると、 libvirt は、接続されているデバイスに対して自動的にこのオペレーションを実行することに注意してください。例: **<hostdev mode='subsystem' type='pci' managed='yes'>**。詳細は、注記 を参照してください。

注記

libvirt には、PCI デバイスを処理する方法が 2 つあります。それらはマネージドにも非マネージドにもなります。これは、**<hostdev>** 要素の **managed** 属性に指定された値により決定します。デバイスがマネージドである場合、libvirt は既存のドライバーからデバイスの割り当てを自動的に解除し、起動時にこれを vfio-pci にバインドして仮想マシンに割り当てます (仮想マシンの場合)。仮想マシンがシャットダウンされるか、または削除される場合、または PCI デバイスの割り当てが仮想マシンから解除される場合、libvirt はデバイスを vfio-pci のバインドから解除し、これを元のドライバーに再びバインドします。デバイスが非マネージドの場合、libvirt はこのプロセスを自動化しません。説明したように、これらすべての管理面が、仮想マシンにデバイスを割り当てる前に完了しているようにしてください。また、デバイスが仮想マシンによって使用されなくなると、デバイスを再割り当てする必要もあります。デバイスを仮想マシンに割り当てる前に、ここで説明する管理内容すべてが完了していることを確認する必要があります。したがって、libvirt がデバイスを管理していることを確認する方が簡単な場合があります。

E.4. IOMMU ストラテジーおよびユースケース

必要以上のデバイスが含まれる IOMMU グループを処理する方法は多数あります。プラグインカードの場合、最初にカードを異なるスロットにインストールして必要なグループが生成されるかどうかを判別します。通常の Intel チップセットでは、PCIe ルートポートはプロセッサーと PCH (Platform Controller Hub) の両方で提供されます。これらのルートポートの各種機能はそれぞれ非常に異なる場合があります。PCH ルートポートの多くにネイティブの PCIe ACS サポートがなくても、Red Hat Enterprise Linux 7 は数多くの PCH ルートポート分離の公開をサポートします。そのため、これらのルートポートは小規模な IOMMU グループを作成する上での適切なターゲットになります。Intel® Xeon® class プロセッサー (E5 シリーズ以上) およびハイエンドのデスクトッププロセッサーの場合、通常プロセッサーベースの PCIe ルートポートは PCIe ACS のネイティブサポートを提供しますが、Core™ i3、i5、i7 および Xeon E3 プロセッサーなどのローエンドのクライアントプロセッサーはこれを提供しません。このようなシステムでは、通常、PCH の root ポートを使用すると、最も柔軟な分離設定が可能になります。

もう 1 つのオプションは、ハードウェアベンダーと連携して分離が存在するかどうかを確認し、この分離を認識するためにカーネルを問い合わせることです。一般的には、関数間の内部ピアツーピアが可能かどうかを判断することや、ダウンストリームポートの場合は、リダイレクトが可能かどうかを判断することが必要になります。Red Hat Enterprise Linux 7 カーネルには、このようなデバイスに関する多くの特異な動作が含まれており、Red Hat カスタマーサポートは、ハードウェアベンダーと協力して、ACS に相当する分離が利用可能であるかどうかについて、そして同様の特異な動作をカーネルに組み込んでこの分離を公開する最適な方法について判断するサポートをします。ハードウェアベンダーの場合、ピアツーピアに対応していないマルチファンクションのエンドポイントは、設定空間で 1 つの静的 ACS テーブルを使用してこれを公開でき、機能は公開しないことに注意してください。このような機能をハードウェアに追加すると、カーネルが分離された機能を自動的に検出し、ハードウェアのすべてのユーザーに対してこの問題を排除できます。

ユーザーが指定した特定のデバイスまたは特定のタイプのデバイスに対して、これらの分離チェックを無効にするオプションをカーネルが提供する必要があるというのが、上記の提案を利用できない場合の一般的な反応になります。多くの場合、以前の技術ではこの範囲内で分離が強制されず、すべてが正常に機能していると主張されます。残念ながら、このような分離機能を回避すると、移植できない環境が発生します。分離が存在するかどうかわからないということは、デバイスが実際に分離されているかどうかを知らないことを意味し、障害が発生する前に検出することが最善策となります。デバイスの分離機能のギャップは、トリガーするのが極めて難しく、その原因としてのデバイスの分離までさかのぼって追跡することは、さらに難しくなります。VFIO の仕事は、何よりもまず、ユーザーが所有するデバイスからホストカーネルを保護することであり、IOMMU グループは、VFIO が確実に分離するために使用するメカニズムです。

つまり、IOMMU グループの上に構築されることにより、VFIO は、従来の KVM デバイス割り当てを使用して可能だった場合よりも、デバイス間のセキュリティーと分離のレベルを強化して提供することができます。この分離は Linux カーネルレベルで適用され、カーネルが自身を保護し、ユーザーに危険な設定を防ぐことができます。さらに、ハードウェアベンダーには、マルチファンクションエンドポイントデバイスだけでなく、チップセットや相互接続デバイスにおいても、PCIe ACS をサポートすることが推奨されます。このサポートのない既存のデバイスについては、ハードウェアベンダーと協力して分離が利用可能かどうかを確認し、Linux カーネルサポートを追加してこの分離を公開することができます。

---

[1] この付録の元の内容はプリンシパルソフトウェアエンジニアの Alex Williamson によって提供されました。

[2] 例外は、レガシーの KVM デバイスの割り当てで、pci-stub ドライバーにバインドしている間デバイスと対話します。Red Hat Enterprise Linux 7 にはレガシー KVM デバイス割り当てが含まれないため、この対話と潜在的な競合の発生を防ぎます。Red Hat Enterprise Linux 7 にはレガシー KVM デバイス割り当てが含まれないため、この相互対話と潜在的な衝突が回避されます。

付録F 更新履歴

改訂 2-42 Thu August 9 2019 Jiri Herrmann
7.7 GA リリースのバージョン

改訂 2-40 Thu May 23 2019 Jiri Herrmann
7.7 ベータ版公開用バージョン

改訂 2-39 Thu Oct 25 2018 Jiri Herrmann
7.6 GA リリースのバージョン

改訂 2-37 Thu Aug 14 2018 Jiri Herrmann
7.6 ベータ版公開用バージョン

改訂 2-36 Thu Aug 14 2018 Jiri Herrmann
仮想ストレージの章の再作業が追加されました。

改訂 2-35 Thu Apr 5 2018 Jiri Herrmann
7.5 GA 公開用バージョン

改訂 2-32 Thu Jul 27 2017 Jiri Herrmann
7.4 GA 公開用バージョン

改訂 2-32 Thu Jul 27 2017 Jiri Herrmann
7.4 GA 公開用バージョン

改訂 2-29 Mon Oct 17 2016 Jiri Herrmann
7.3 GA 公開用バージョン

改訂 2-24 Thu Dec 17 2015 Laura Novich
ガイドを更新し、複数の問題を修正

改訂 2-23 Sun Nov 22 2015 Laura Novich
再公開されたガイド

改訂 2-21 Thu Nov 12 2015 Laura Novich
7.2 に対する複数のコンテンツの更新

改訂 2-19 Thu Oct 08 2015 Jiri Herrmann
改訂履歴の整理

改訂 2-17 Thu Aug 27 2015 Dayle Parker
7.2 ベータリリースの更新