Red Hat Enterprise Linux 6

가상화 관리 가이드

가상 환경 관리

Last Updated: 2022-07-03

Red Hat Enterprise Linux 6 가상화 관리 가이드

가상 환경 관리

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

법적 공지

초록

가상화 관리 가이드에서는 호스트 물리적 시스템, 네트워킹, 스토리지, 장치, 게스트 가상 시스템 관리 및 문제 해결에 대한 관리를 다룹니다. 참고: 이 문서는 개발 중이며 상당한 변경 사항이 적용되며, 미리 보기로만 제공됩니다. 포함 된 정보 및 지침은 완전 한 것으로 간주해서는 안되며 주의해서 사용해야 합니다. 전문 지식을 확장하려면 Red Hat Virtualization(RH318)교육 과정에 관심이 있을 수 있습니다.

차례

1장. 서버 모범 사례

다음 작업 및 팁은 에서 Red Hat Enterprise Linux 호스트의 성능을 높이는 데 도움이 될 수 있습니다. 자세한 내용은 *Red Hat Enterprise Linux Virtualization 튜닝 및 최적화 가이드에서* 확인할 수 있습니다.

- 강제 모드에서 SELinux를 실행합니다. **setenforce** 명령을 사용하여 SELinux를 강제 모드로 설정합니다.

```
# setenforce 1
```

- **AutoFS**, NFS ,**NFS**,**FTP**,**HTTP**,**NIS**,**telnetd** 와 같은 불필요한 서비스를 제거하거나 비활성화합니다.

- **서버의 플랫폼 관리에 필요한 최소 사용자 계정 수만 추가하고 불필요한 사용자 계정을 제거합니다.**

- **호스트에서 필수 애플리케이션이 실행되지 않도록 합니다. 호스트에서 애플리케이션을 실행하면 가상 시스템 성능에 영향을 줄 수 있으며 서버 안정성에 영향을 미칠 수 있습니다. 서버에 충돌이 발생할 수 있는 모든 애플리케이션도 서버의 모든 가상 시스템을 작동 중단시킵니다.**

- **가상 시스템 설치 및 이미지에 중앙 위치를 사용합니다. 가상 머신 이미지는 /var/lib/libvirt/images/에 저장해야 합니다. 가상 머신 이미지에 다른 디렉터리를 사용하는 경우 디렉토리를 SELinux 정책에 추가하고 설치를 시작하기 전에 레이블을 다시 지정해야 합니다. 공유 가능한 공유 가능, 중앙 위치에서 네트워크 스토리지를 사용하는 것이 좋습니다.**

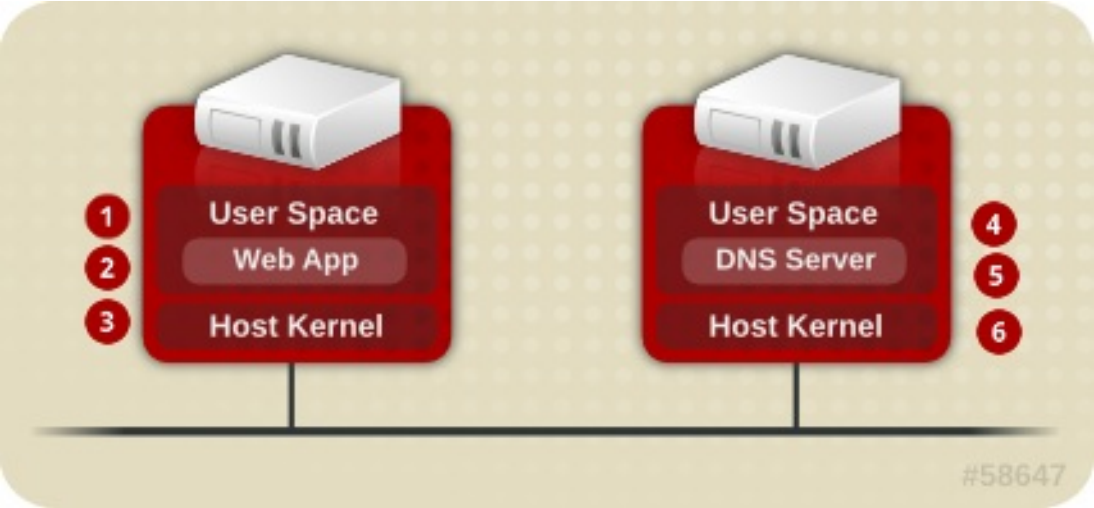
2장. SVIRT

sVirt는 Red Hat Enterprise Linux 6에 포함된 기술로, SELinux 및 가상화를 통합한 것입니다. sVirt는 게스트 가상 시스템을 사용할 때 보안을 강화하기 위해 MAC(Mandatory Access Control)를 적용합니다. 이 통합 기술은 보안이 향상되고 하이퍼바이저의 버그와 비교하여 시스템을 강화합니다. 호스트 물리적 시스템 또는 다른 게스트 가상 시스템에 대한 공격을 방지하는 데 특히 유용합니다.

이 장에서는 VMDK가 Red Hat Enterprise Linux 6의 가상화 기술과 통합되는 방법에 대해 설명합니다.

비가상화 환경

가상화되지 않은 환경에서는 호스트 물리적 시스템이 서로 분리되어 있으며 각 호스트 물리적 시스템에는 웹 서버 또는 DNS 서버와 같은 서비스로 구성된 자체 포함 환경이 있습니다. 이러한 서비스는 자체 사용자 공간과 물리적 시스템의 커널 및 물리적 하드웨어와 직접 통신하여 네트워크에 직접 서비스를 제공합니다. 다음 이미지는 가상화되지 않은 환경을 나타냅니다.



??????
: 사용자 공간 - 모든 사용자 모드 애플리케이션과 일부 드라이버가 실행되는 메모리 영역입니다.

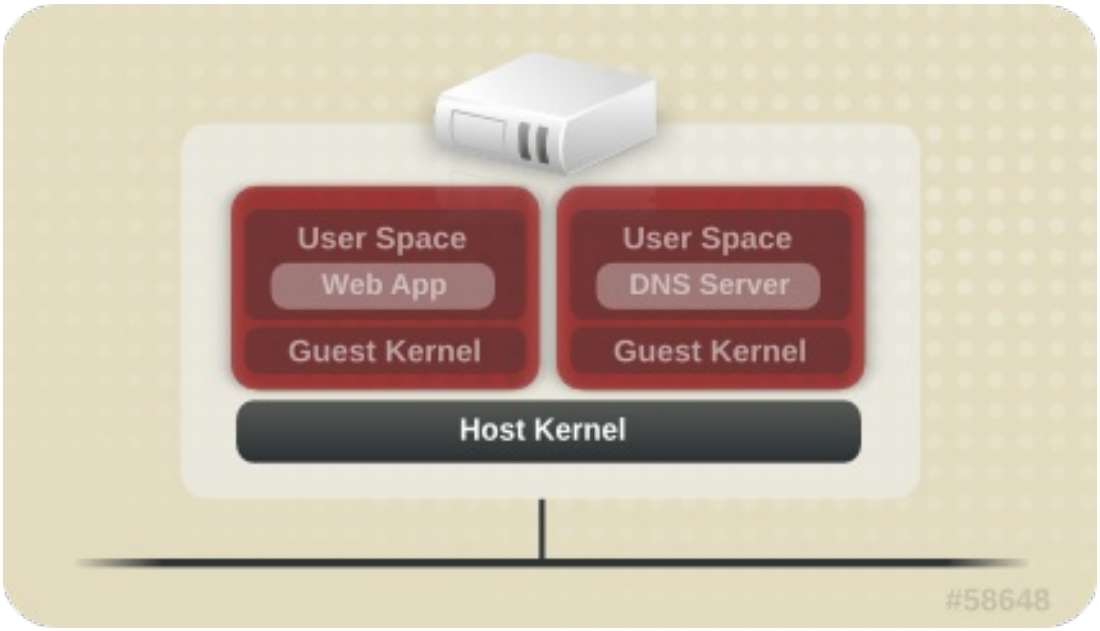
???
: 웹 앱(웹 애플리케이션 서버) - 브라우저를 통해 액세스할 수 있는 웹 콘텐츠를 제공합니다.

??????
: 호스트 커널 - 호스트 물리적 시스템의 권한 있는 커널, 커널 확장 및 대부분의 장치 드라이버를 실행하기 위해 엄격하게 예약되어 있습니다.

???
: DNS 서버 - 사용자가 IP 주소 대신 논리 이름을 사용하여 웹 페이지에 액세스할 수 있도록 허용하는 DNS 레코드를 저장합니다.

가상화된 환경

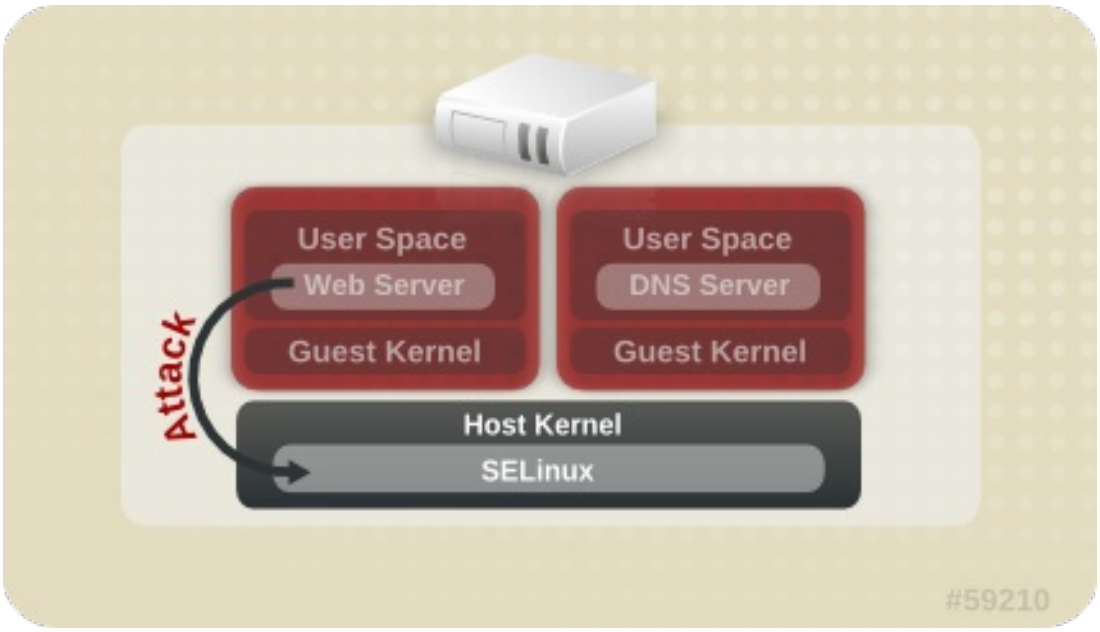
가상화 환경에서 호스트 물리적 시스템에 상주하는 단일 커널에서 여러 가상 운영 체제를 실행할 수 있습니다. 다음 이미지는 가상화된 환경을 나타냅니다.



2.1. 보안 및 가상화

서비스가 가상화되지 않으면 시스템은 물리적으로 분리됩니다. 모든 익스플로잇은 일반적으로 영향을 받는 시스템에 포함되며 네트워크 공격을 명백하게 제외합니다. 서비스가 가상화 환경에서 함께 그룹화되면 시스템에서 추가 취약점이 발생합니다. 게스트 가상 머신에서 악용할 수 있는 보안 취약점이 있는 경우 이 게스트 가상 머신은 호스트 물리적 시스템을 공격할 뿐만 아니라 해당 호스트 물리적 시스템에서 실행되는 다른 게스트 가상 머신도 공격할 수 있습니다. 이러한 공격은 게스트 가상 머신 이상으로 확장될 수 있으며 다른 게스트 가상 머신도 공격에 노출될 수 있습니다.

pkexec는 게스트 가상 머신을 분리하고 악용된 경우 추가 공격을 시작하는 기능을 제한하기 위한 것입니다. 이는 다음 이미지에서 설명됩니다. 공격이 게스트 가상 머신에서 벗어나 다른 게스트 가상 머신을 분리할 수 없는 경우 다음과 같습니다.



SELinux는 MAC(Mandatory Access Control)를 구현할 때 가상화된 인스턴스를 위한 플러그형 보안 프레임워크를 도입합니다. KnativeServing 프레임워크를 사용하면 게스트 가상 머신과 해당 리소스에 고유한 레이블이 지정될 수 있습니다. 레이블이 지정되면 다른 게스트 가상 머신 간 액세스를 거부할 수 있는 규칙을 적용할 수 있습니다.

2.2. KNATIVESERVING 라벨링

SELinux를 보호하는 다른 서비스와 마찬가지로, Mellanox는 프로세스 기반 메커니즘 및 제한 사항을 사용하여 게스트 가상 시스템에 대한 추가 보안 계층을 제공합니다. 일반적인 용도의 경우, Mellanox가 백그라운드에서 작동하고 있음을 알 수 없습니다. 이 섹션에서는 sVirt의 라벨링 기능에 대해 설명합니다.

다음 출력에 표시된 것처럼 trustedCA를 사용할 때 가상화된 게스트 가상 머신 프로세스의 레이블이 지정되고 동적으로 생성된 수준으로 실행됩니다. 각 프로세스는 다른 수준의 다른 VM과 격리됩니다.

```
# ps -eZ | grep qemu

system_u:system_r:svirt_t:s0:c87,c520 27950 ?  00:00:17 qemu-kvm
```

다음 출력에 표시된 대로 실제 디스크 이미지는 프로세스와 일치하도록 자동으로 레이블이 지정됩니다.

```
# ls -lZ /var/lib/libvirt/images/*

  system_u:object_r:svirt_image_t:s0:c87,c520   image1
```

다음 표에서는 4.7.1을 사용할 때 할당할 수 있는 다양한 컨텍스트 레이블을 간략하게 설명합니다.

표 2.1. sVirt context labels

SELinux Context	유형 / 설명
system_u:system_r:svirt_t:MCS1	게스트 가상 머신 프로세스. MCS1은 임의의 MCS 필드입니다. 대략 500,000개의 레이블이 지원됩니다.
system_u:object_r:svirt_image_t:MCS1	게스트 가상 머신 이미지. MCS 필드가 동일한*svirt_t* 프로세스만 이러한 이미지를 읽고 쓸 수 있습니다.
system_u:object_r:svirt_image_t:s0	게스트 가상 머신 공유 읽기/쓰기 콘텐츠. 모든*svirt_t* 프로세스는 svirt_image_t:s0 파일에 쓸 수 있습니다.

또한 sVirt를 사용할 때 정적 레이블링을 수행할 수도 있습니다. 정적 레이블을 사용하면 관리자가 게스트 가상 머신의 MCS/MLS 필드를 포함한 특정 레이블을 선택할 수 있습니다. 정적으로 레이블이 지정된 가상화된 게스트 가상 머신을 실행하는 관리자는 이미지 파일에 올바른 레이블을 설정해야 합니다. 게스트 가상 머신은 항상 해당 레이블로 시작되며, Restic 시스템은 정적으로 레이블이 지정된 가상 머신 콘텐츠의 레이블을 변경하지 않습니다. 이를 통해 KnativeServing 구성 요소를 MLS 환경에서 실행할 수 있습니다. 요구 사항에 따라 시스템의 민감도 수준이 다른 여러 게스트 가상 머신을 실행할 수도 있습니다.

3장. 가상 머신 복제

게스트 복사본을 생성하는 데 사용되는 게스트 가상 머신 인스턴스에는 다음 두 가지 유형이 있습니다.

- *복제본*은 단일 가상 시스템의 인스턴스입니다. 복제본은 동일한 가상 머신의 네트워크를 설정하는 데 사용할 수 있으며 다른 대상에 배포할 수도 있습니다.
- *템플릿*은 복제용 소스로 사용하도록 설계된 가상 머신의 인스턴스입니다. 템플릿에서 여러 복제본을 생성하고 각 복제를 약간 수정할 수 있습니다. 이는 시스템에서 이러한 변경의 영향을 확인하는 데 유용합니다.

복제본과 템플릿은 모두 가상 시스템 인스턴스입니다. 차이점은 어떻게 사용되는지에 있습니다.

생성된 복제가 제대로 작동하려면 복제 중인 가상 머신에 고유한 정보 및 구성을 복제하기 전에 제거해야 합니다. 복제 사용 방법에 따라 제거해야 하는 정보는 서로 다릅니다.

제거할 정보 및 구성은 다음 수준에 있을 수 있습니다.

- *플랫폼 수준*정보 및 구성에는 가상화 솔루션에서 가상 머신에 할당한 모든 사항이 포함됩니다. 예를 들면 NIC(네트워크 인터페이스 카드) 및 해당 MAC 주소가 있습니다.
- *게스트 운영 체제 수준*정보 및 구성에는 가상 머신 내에서 구성된 모든 항목이 포함됩니다. 예를 들면 SSH 키가 있습니다.
- *애플리케이션 수준*정보 및 구성에는 가상 머신에 설치된 애플리케이션에서 구성하는 모든 내용이 포함됩니다. 예를 들면 활성화 코드 및 등록 정보가 있습니다.

참고

이 장에서는 정보 및 접근 방식이 각 애플리케이션에 고유하므로 애플리케이션 수준 제거에 대한 정보는 포함되지 않습니다.

따라서 일부 정보와 구성은 가상 시스템 내에서 제거되어야 하는 반면, 다른 정보와 구성은 가상화 환경(예: 가상 시스템 관리자 또는 VMware)을 사용하여 가상 시스템에서 제거해야 합니다.

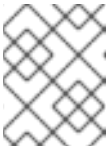
3.1. 클로닝을 위해 가상 머신 준비

가상 머신을 복제하기 전에 디스크 이미지에서 virt-sysprep 유틸리티를 실행하거나 다음 단계를 사용하여 준비해야 합니다.

절차 3.1. 복제를 위한 가상 머신 준비

1. 가상 머신 설정
   a. 복제본 또는 템플릿에 사용할 가상 머신을 빌드합니다.
      - 복제본에 필요한 소프트웨어를 설치합니다.
      - 운영 체제에 대한 고유하지 않은 설정을 구성합니다.
      - 고유하지 않은 애플리케이션 설정을 구성합니다.
2. 네트워크 구성 제거
   a. 다음 명령을 사용하여 영구 udev 규칙을 제거합니다.

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

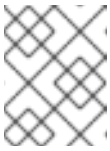


참고

udev 규칙이 제거되지 않으면 첫 번째 NIC 이름은 eth0 대신 eth1일 수 있습니다.

b. ifcfg 스크립트에서 **/etc/sysconfig/network-scripts/ifcfg-eth[x]** 를 다음과 같이 편집하여 고유한 네트워크 세부 정보를 제거합니다.

i. HWADDR 및 정적 행 제거



참고

HWADDR이 새 게스트의 MAC 주소와 일치하지 않으면 ifcfg가 무시됩니다. 따라서 파일에서 HWADDR을 제거하는 것이 중요합니다.

```
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. DHCP 구성이 HWADDR 또는 고유한 정보가 포함되지 않은 상태로 남아 있는지 확인합니다.

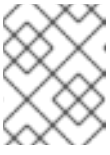
```
DEVICE=eth[x]
BOOTPROTO=dhcp
ONBOOT=yes
```

iii. 파일에 다음 행이 포함되어 있는지 확인합니다.

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

c. 다음 파일이 있는 경우 해당 파일에 동일한 콘텐츠가 포함되어 있는지 확인합니다.

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



참고

NetworkManager 또는 특수 설정을 가상 시스템과 함께 사용한 경우 추가 고유 정보가 ifcfg 스크립트에서 제거되었는지 확인합니다.

3. 등록 세부 정보 제거

a. 다음 중 하나를 사용하여 등록 세부 정보를 제거합니다.

- RHN(Red Hat Network)에 등록된 게스트 가상 머신의 경우 다음 명령을 실행합니다.

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

- Red Hat Subscription Manager (RHSM)가 등록된 게스트 가상 머신의 경우:
  - 원래 가상 머신을 사용하지 않는 경우 다음 명령을 실행합니다.

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

  - 원래 가상 시스템을 사용하는 경우 다음 명령만 실행합니다.

```
# subscription-manager clean
```

참고

원래 RHSM 프로파일은 포털에 남아 있습니다.

4. 기타 고유한 세부 정보 제거

a. 다음 명령을 사용하여 sshd 공개/개인 키 쌍을 제거합니다.

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

참고

ssh 키를 제거하면 ssh 클라이언트의 문제가 이러한 호스트를 신뢰하지 않습니다.

b. 여러 시스템에서 실행되는 경우 충돌을 일으킬 수 있는 다른 애플리케이션별 식별자 또는 구성을 제거합니다.

5. 다음 부팅 시 구성 마법사를 실행하도록 가상 머신 구성

a. 다음 중 하나를 수행하여 부팅될 때 관련 구성 마법사를 실행하도록 가상 머신을 구성합니다.

- Red Hat Enterprise Linux 6 이하의 경우 다음 명령을 사용하여 .unconfigured라는 루트 파일 시스템에 빈 파일을 생성합니다.

```
# touch /.unconfigured
```

- Red Hat Enterprise Linux 7의 경우 다음 명령을 실행하여 첫 번째 부팅 및 초기 설정 마법사를 활성화하십시오.

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

참고

다음 부팅 시 실행되는 마법사는 가상 머신에서 제거된 구성에 따라 다릅니다. 또한 복제본의 첫 번째 부팅 시 호스트 이름을 변경하는 것이 좋습니다.

3.2. 가상 머신 복제

복제를 진행하기 전에 가상 머신을 종료합니다. **virt-clone** 또는 **virt-manager** 를 사용하여 가상 머신을 복제할 수 있습니다.

3.2.1. virt-clone을 사용하여 게스트 복제

virt-clone 을 사용하여 명령줄에서 가상 머신을 복제할 수 있습니다.

virt-clone 을 성공적으로 완료하려면 root 권한이 필요합니다.

virt-clone 명령은 명령줄에서 전달할 수 있는 다양한 옵션을 제공합니다. 여기에는 일반 옵션, 스토리지 구성 옵션, 네트워킹 구성 옵션 및 기타 옵션이 포함됩니다. **--original** 만 필요합니다. 전체 옵션 목록을 보려면 다음 명령을 입력합니다.

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

virt-clone 도움말 페이지에는 각 명령 옵션, 중요한 변수 및 예제도 문서화되어 있습니다.

다음 예에서는 기본 연결에서 "demo"라는 게스트 가상 시스템을 복제하여 새 이름과 디스크 복제 경로를 자동으로 생성하는 방법을 보여줍니다.

예 3.1. **virt-clone** 을 사용하여 게스트 복제

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

다음 예는 여러 디스크가 있는 "demo"라는 QEMU 게스트 가상 머신을 복제하는 방법을 보여줍니다.

예 3.2. **virt-clone** 을 사용하여 게스트 복제

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

3.2.2. virt-manager를 사용하여 게스트 복제

다음 절차에서는 virt-manager 유틸리티를 사용하여 게스트 가상 머신 복제에 대해 설명합니다.

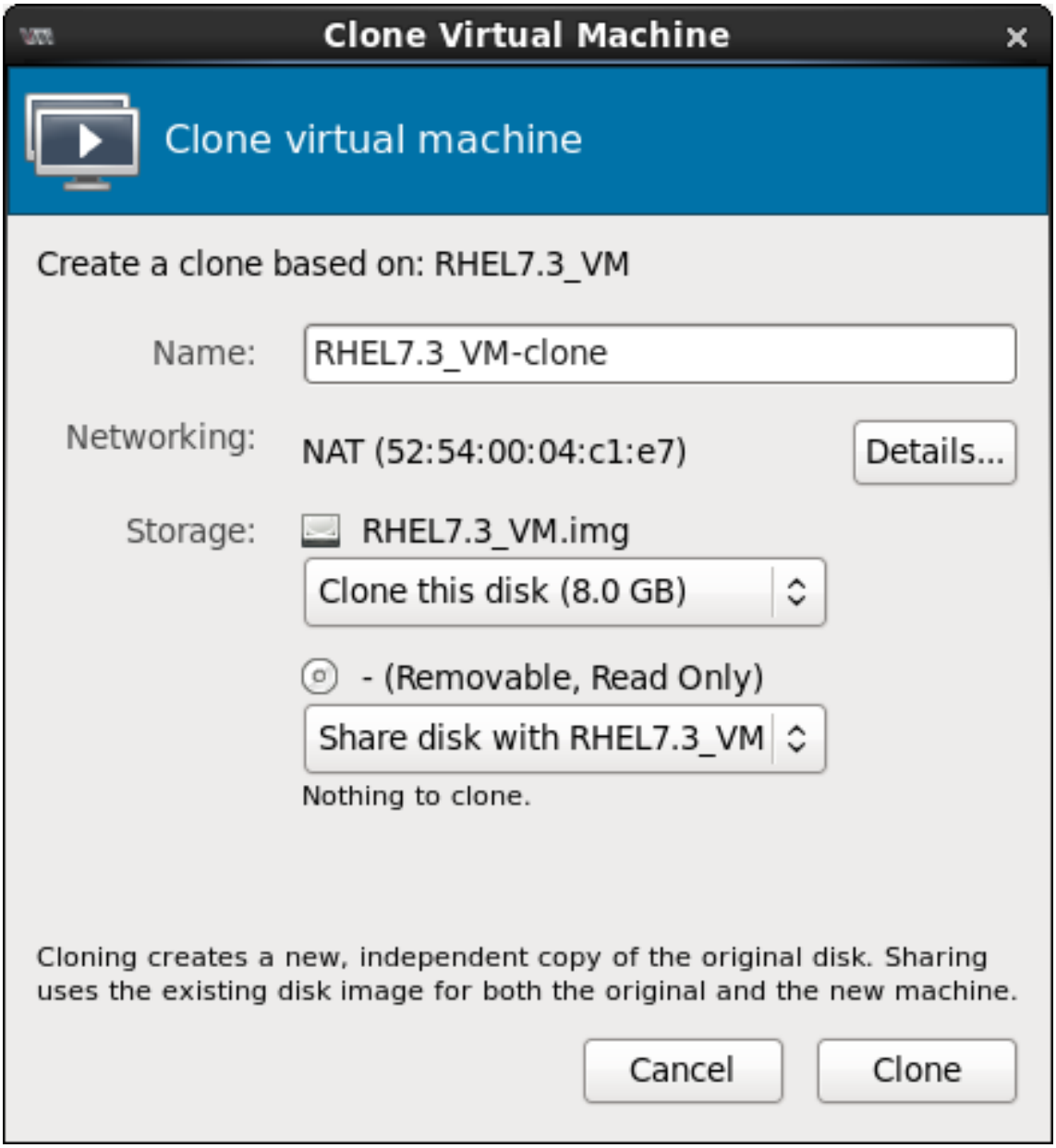
절차 3.2. virt-manager를 사용하여 가상 머신 복제

1. virt-manager를 엽니다.
   virt-manager 를 시작합니다. Applications (애플리케이션) 메뉴 및 System Tools (시스템 도구) 하위 메뉴에서 Virtual Machine Manager애플리케이션을 시작합니다. 또는 root로 **virt-manager** 명령을 실행합니다.

   가상 머신 관리자 의 게스트 가상 머신 목록에서 복제할 게스트 가상 머신을 선택합니다.

복제할 guest 가상 머신을 마우스 오른쪽 버튼으로 클릭하고 Clone 을 선택합니다. Clone Virtual Machine(가상 시스템 복제) 창이 열립니다.

그림 3.1. 가상 머신 창 복제



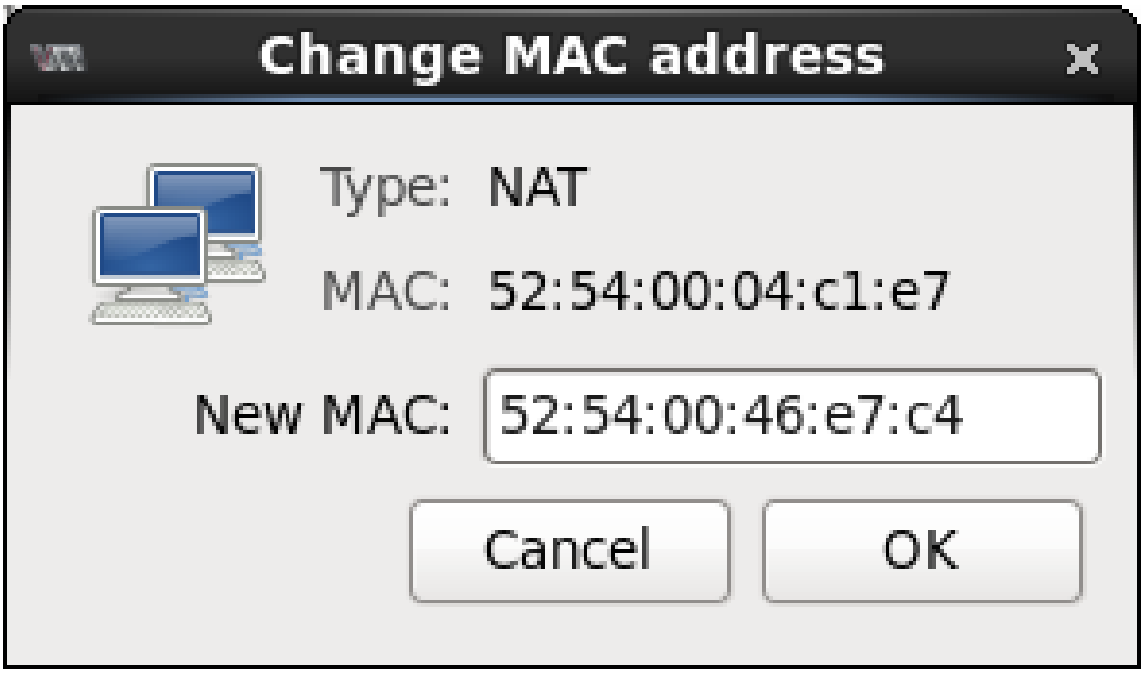
2. 복제 구성

- 복제본 이름을 변경하려면 복제본에 새 이름을 입력합니다.
- 네트워킹 구성을 변경하려면 세부 정보를 클릭합니다.

  복제본에 새 MAC 주소를 입력합니다.

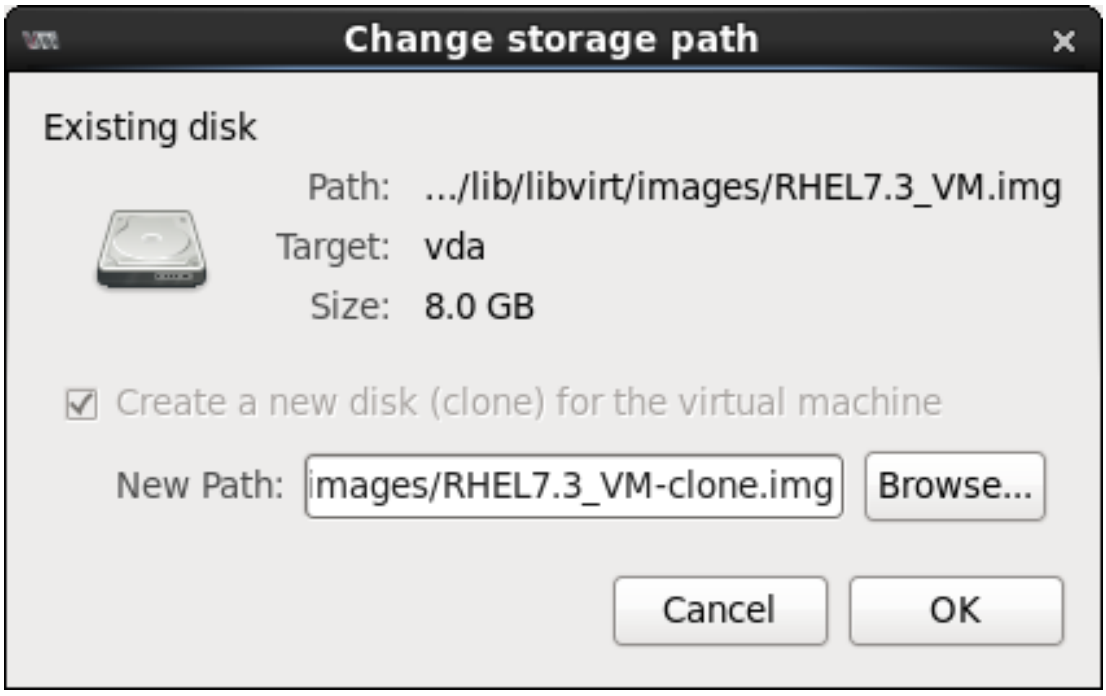
  OK를 클릭합니다.

그림 3.2. MAC 주소 창 변경



- 복제된 게스트 가상 머신의 각 디스크에 대해 다음 옵션 중 하나를 선택합니다.
  - 이 디스크 복제 - 복제된 게스트 가상 머신에 대해 디스크가 복제됩니다.
  - 디스크를 *게스트 가상 머신 이름*과공유 - 복제된 게스트 가상 머신에서 디스크를 공유합니다.
  - 세부 정보 - 디스크의 새 경로를 선택할 수 있는 변경 스토리지 경로 창을 엽니다.

    그림 3.3. 스토리지 경로 창 변경



3. 게스트 가상 머신 복제
   Clone (복제)을 클릭합니다.

4장. KVM 실시간 마이그레이션

이 장에서는 한 호스트 물리적 시스템에서 다른 호스트로 실행되는 게스트 가상 머신 마이그레이션에 대해 설명합니다. 두 인스턴스 모두에서 호스트 물리적 머신이 KVM 하이퍼바이저를 실행하고 있습니다.

마이그레이션은 한 호스트 물리적 시스템에서 다른 호스트로 게스트 가상 시스템을 이동하는 프로세스를 설명합니다. 게스트 가상 머신이 하드웨어에서 직접 실행되는 것이 아니라 가상화 환경에서 실행되기 때문에 가능합니다. 마이그레이션은 다음에 유용합니다.

- 부하 분산 - 호스트 물리적 시스템이 과부하되거나 다른 호스트 물리적 시스템이 활용도가 낮은 사용량을 사용하여 게스트 가상 시스템을 호스트할 수 있습니다.
- 하드웨어 독립성 - 호스트 물리적 시스템에서 하드웨어 장치를 업그레이드, 추가 또는 제거해야 하는 경우 게스트 가상 머신을 다른 호스트 물리적 시스템으로 안전하게 재배치할 수 있습니다. 즉, 게스트 가상 머신은 하드웨어 개선을 위해 다운타임이 발생하지 않습니다.
- 에너지 절약 - 게스트 가상 머신은 다른 호스트 물리적 시스템에 재배포될 수 있으므로 전력을 절약하고 사용 기간에 비용을 절감할 수 있습니다.
- 지리적 마이그레이션 - 대기 시간이 낮거나 심각한 상황에서 게스트 가상 머신을 다른 위치로 이동할 수 있습니다.

마이그레이션은 게스트 가상 시스템의 메모리와 가상화된 장치의 상태를 대상 호스트 물리적 시스템으로 전송하여 작동합니다. 마이그레이션할 게스트 가상 머신의 이미지를 저장하기 위해 공유, 네트워크로 연결된 스토리지를 사용하는 것이 좋습니다. 또한 가상 머신을 마이그레이션할 때 공유 스토리지에 libvirt 관리 스토리지 풀을 사용하는 것이 좋습니다.

마이그레이션은 실시간 또는 그렇지 않을 수 있습니다.

실시간 마이그레이션에서는 메모리 페이지가 순서대로 대상 호스트 실제 시스템으로 전송되는 동안 게스트 가상 시스템이 소스 호스트 물리적 시스템에서 계속 실행됩니다. 마이그레이션 중에 KVM은 이미 전송된 페이지의 변경 사항에 대해 소스를 모니터링하고 모든 초기 페이지가 전송되면 이러한 변경 사항을 전송하기 시작합니다. KVM은 마이그레이션 중에 전송 속도를 추정하므로 전송할 데이터의 나머지 양이 일정 시간(10 밀리초)이 필요한 경우 KVM은 원래 게스트 가상 머신을 중지하고 나머지 데이터를 전송하고, 대상 호스트 물리적 시스템에서 동일한 게스트 가상 머신을 재시작합니다.

실시간 수행되지 않고 게스트 가상 머신을 일시 중단한 다음 게스트 가상 머신 메모리의 이미지를 대상 호스트 물리적 시스템으로 이동하는 마이그레이션입니다. 그러면 게스트 가상 시스템이 대상 호스트 물리적 시스템에서 다시 시작되고 소스 호스트 실제 시스템에서 사용된 게스트 가상 시스템이 확보되는 메모리가 확보됩니다. 이러한 마이그레이션을 완료하는 데 걸리는 시간은 네트워크 대역폭 및 대기 시간에 따라 달라집니다. 네트워크에 사용량이 크거나 낮은 대역폭이 발생하는 경우 마이그레이션에 시간이 훨씬 오래 걸립니다.

원래 게스트 가상 머신이 KVM이 대상 호스트 물리적 머신으로 전송할 수 있는 것보다 더 빠르게 페이지를 수정하는 경우 실시간 마이그레이션이 완료되지 않기 때문에 오프라인 마이그레이션을 사용해야 합니다.

4.1. 실시간 마이그레이션 요구 사항

게스트 가상 머신을 마이그레이션하려면 다음이 필요합니다.

마이그레이션 요구 사항

- 다음 프로토콜 중 하나를 사용하여 공유 스토리지에 설치된 게스트 가상 머신:
    - 파이버 채널 기반 LUN

- iSCSI
- FCoE
- NFS
- GFS2
- SCSI RDMA 프로토콜(SCSI RCP): Infiniband 및 10GbE iWARP 어댑터에 사용되는 블록 내보내기 프로토콜

- 마이그레이션 플랫폼 및 버전은 표 4.1. "실시간 마이그레이션 호환성" 테이블에 대해 확인해야 합니다. 또한 Red Hat Enterprise Linux 6는 공유 스토리지에서 raw 및 qcow2 이미지를 사용하여 게스트 가상 시스템의 실시간 마이그레이션을 지원합니다.

- 두 시스템 모두 적절한 TCP/IP 포트가 열려 있어야 합니다. 방화벽을 사용하는 경우 자세한 포트 정보는 에서 확인할 https://access.redhat.com/site/documentation/ 수 있는 *Red Hat Enterprise Linux Virtualization Security 가이드* 를 참조하십시오.

- 공유 스토리지 매체를 내보내는 개별 시스템입니다. 마이그레이션에 사용되는 두 호스트 물리적 시스템 중 하나에 스토리지가 없어야 합니다.

- 공유 스토리지는 소스 및 대상 시스템의 동일한 위치에 마운트해야 합니다. 마운트된 디렉터리 이름은 동일해야 합니다. 이미지를 다른 경로를 사용하여 유지할 수 있지만 권장되지는 않습니다. virt-manager를 사용하여 마이그레이션을 수행하려는 경우 경로 이름이 동일해야 합니다. 그러나 virsh를 사용하여 마이그레이션을 수행하려는 경우 마이그레이션을 수행할 때 **--xml** 옵션 또는 pre-hooks와 함께 다른 네트워크 구성 및 마운트 디렉토리를 사용할 수 있습니다. 공유 스토리지가 없어도 **--copy-storage-all** (더 이상 사용되지 않음) 옵션을 사용하여 마이그레이션이 성공할 수 있습니다. **prehooks** 에 대한 자세한 내용은libvirt.org 를 참조하고 XML 옵션에 대한 자세한 내용은20장. *도메인 XML 조작* 을 참조하십시오.

- 공용 bridge+tap 네트워크의 기존 게스트 가상 시스템에서 마이그레이션을 시도하는 경우 소스 및 대상 호스트 물리적 시스템이 동일한 네트워크에 있어야 합니다. 그렇지 않으면 마이그레이션 후 게스트 가상 머신 네트워크가 작동하지 않습니다.

- Red Hat Enterprise Linux 5 및 6에서는 KVM 게스트 가상 머신의 기본 캐시 모드가 ***none*** 으로 설정되어 일치하지 않는 디스크 상태를 방지합니다. 캐시 옵션을 ***none*** (예: **virsh attach-disk** 캐시 사용)으로 설정하면 ***O_DIRECT*** 플래그(**Open syscall**을 호출할 때)를 사용하여 모든 게스트 가상 시스템의 파일이 열리 므로 호스트 물리적 시스템의 캐시를 우회하고 게스트 가상 시스템에서 캐싱만 제공합니다. 캐시 모드를 ***none*** 으로 설정하면 잠재적인 불일치 문제를 방지하고 가상 시스템을 실시간으로 마이그레이션할 수 있습니다. 캐시를 ***none*** 으로 설정하는 방법에 대한 자세한 내용은 13.3절. "게스트에 스토리지 장치 추가" 을 참조하십시오.

libvirtd 서비스가 활성화되어 있는지 확인합니다(**#service libvirtd** 시작). 또한 효과적으로 마이그레이션하는 기능은 **/etc/libvirt/libvirtd.conf** 설정 파일의 매개 변수 설정에 따라 달라집니다.

절차 4.1. libvirtd.conf 구성

1. **libvirtd.conf** 를 열려면 root로 명령을 실행해야 합니다.

```
# vim /etc/libvirt/libvirtd.conf
```

2. 필요에 따라 매개변수를 변경하고 파일을 저장합니다.

3. **libvirtd** 서비스를 다시 시작합니다.

```
# service libvirtd restart
```

4.2. 실시간 마이그레이션 및 RED HAT ENTERPRISE LINUX 버전 호환성

실시간 마이그레이션은 표 4.1. "실시간 마이그레이션 호환성" 표에 설명된 대로 지원됩니다.

표 4.1. 실시간 마이그레이션 호환성

마이그레이션 방법	릴리스 유형	예제	실시간 마이그레이션 지원	참고
forward	메이저 릴리스	5.x → 6.y	지원되지 않음	
forward	마이너 릴리스	5.x → 5.y (y>x, x>=4)	모두 지원됨	모든 문제가 보고되어야 합니다.
forward	마이너 릴리스	6.x → 6.y (y>x, x>=0)	모두 지원됨	모든 문제가 보고되어야 합니다.
이전	메이저 릴리스	6.x → 5.y	지원되지 않음	
이전	마이너 릴리스	5.x → 5.y (x>y,y>=4)	지원됨	알려진 문제는 마이그레이션 문제 해결에서 참조하십시오.
이전	마이너 릴리스	6.x → 6.y (x>y, y>=0)	지원됨	알려진 문제는 마이그레이션 문제 해결에서 참조하십시오.

마이그레이션 문제 해결

- SPICE 관련 문제 - Red Hat Enterprise Linux 6.0 → 6.1에서 마이그레이션할 때 SPICE에 호환되지 않는 변경 사항이 있음을 확인했습니다. 이러한 경우 클라이언트는 연결을 끊은 다음 다시 연결하여 오디오 및 비디오의 일시적인 손실을 초래할 수 있습니다. 이는 일시적인 것이며 모든 서비스가 다시 시작됩니다.

- USB 문제 - Red Hat Enterprise Linux 6.2는 마이그레이션 지원을 포함한 USB 기능을 추가했지만 USB 장치를 재설정하고 장치를 통해 실행되는 애플리케이션을 중단시키는 특정 경고가 발생하지 않았습니다. 이 문제는 Red Hat Enterprise Linux 6.4에서 해결되었으며 향후 버전에서는 발생하지 않아야 합니다. 6.4 이전 버전에서 이러한 문제가 발생하지 않도록하기 위해 USB 장치를 사용하는 동안 마이그레이션을 유도합니다.

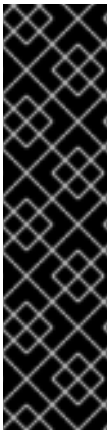
- 마이그레이션 프로토콜 문제 - 이전 마이그레이션이 "알리지 않은 섹션 오류"로 끝나는 경우 마이그레이션 프로세스를 반복하면 일시적인 오류일 수 있으므로 마이그레이션 프로세스를 반복할 수 있습니다. 그렇지 않은 경우 문제를 보고하십시오.

네트워크 스토리지 구성

공유 스토리지를 구성하고 공유 스토리지에 게스트 가상 머신을 설치합니다.

또는 에서 NFS 예제를 사용합니다. 4.3절. "공유 스토리지 예: 간단한 마이그레이션을 위한 NFS"

4.3. 공유 스토리지 예: 간단한 마이그레이션을 위한 NFS



중요

이 예에서는 NFS를 사용하여 게스트 가상 머신 이미지를 다른 KVM 호스트 물리적 시스템과 공유합니다. 대규모 설치에서는 실용적이지 않지만 마이그레이션 기술만 시연할 수 있습니다. 몇 가지 게스트 가상 머신을 마이그레이션하거나 실행하는 데 이 예제를 사용하지 마십시오. 또한 ***sync*** 매개 변수를 활성화해야 합니다. NFS 스토리지를 올바르게 내보내는 데 필요합니다. 또한 NFS는 소스 호스트 물리적 시스템에 마운트되어 있으며 소스 호스트 물리적 시스템에 있는 NFS 마운트 디렉터리에서 게스트 가상 시스템의 이미지를 생성해야 합니다. 또한 KVM에서 NFS 파일 잠금이 지원되지 않으므로 사용해서는 안 됩니다.

iSCSI 스토리지는 대규모 배포에 더 적합합니다. 구성 세부 정보는 12.5절. "iSCSI 기반 스토리지 풀" 를 참조하십시오.

또한 이 섹션에 제공된 지침은 *Red Hat Linux Storage 관리 가이드에*있는 자세한 지침을 대체하지 않습니다. NFS 구성, IP 테이블 열기 및 방화벽 구성에 대한 정보는 이 가이드를 참조하십시오.

1. 디스크 이미지 디렉터리 생성
   이 공유 디렉터리에는 게스트 가상 머신의 디스크 이미지가 포함됩니다. 이렇게 하려면 **/var/lib/libvirt/images** 와 다른 위치에 디렉터리를 생성합니다. 예를 들어 다음과 같습니다.

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

2. NFS 구성 파일에 새 디렉터리 경로 추가
   NFS 구성 파일은 **/etc/exports** 에 있는 텍스트 파일입니다. 파일을 열고 1단계에서 만든 새 파일의 경로를 추가합니다.

   ```
   # echo "/var/lib/libvirt-img/images" >> /etc/exports/[NFS-Config-FILENAME.txt]
   ```

3. NFS 시작

   a. **iptables** 에서 NFS 포트 (예: 2049)가 열려 있는지 확인하고**/etc/hosts.allow** 파일에 NFS를 추가합니다.

   b. NFS 서비스를 시작합니다.

   ```
   # service nfs start
   ```

4. 소스와 대상 모두에 공유 스토리지를 마운트합니다.
   소스 및 대상 시스템 모두에 **/var/lib/libvirt/images** 디렉터리를 마운트하여 다음 명령을 두 번 실행합니다. 소스 시스템에서 그리고 대상 시스템에서 다시 한 번.

   ```
   # mount source_host:/var/lib/libvirt-img/images /var/lib/libvirt/images
   ```

주의

4.1절. "실시간 마이그레이션 요구 사항" 에 설명된 대로 이 절차에서 생성하는 디렉터리가 요구 사항을 준수하는지 확인하십시오. 또한 디렉터리를 올바른 SELinux 레이블로 레이블을 지정해야 할 수도 있습니다. 자세한 내용은 Red Hat Enterprise Linux 스토리지 관리 가이드의 NFS 장을 참조하십시오.

4.4. VIRSH를 통한 실시간 KVM 마이그레이션

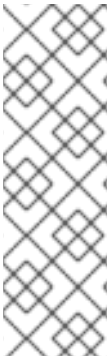
virsh 명령을 사용하여 게스트 가상 머신을 다른 호스트 물리적 시스템으로 마이그레이션할 수 있습니다. **migrate** 명령은 다음 형식의 매개 변수를 허용합니다.

```
# virsh migrate --live GuestName DestinationURL
```

실시간 마이그레이션이 필요하지 않은 경우 **--live** 옵션이 제거될 수 있습니다. 추가 옵션은4.4.2절. "virsh migrate 명령에 대한 추가 옵션" 에 나열됩니다.

GuestName 매개변수는 마이그레이션할 게스트 가상 머신의 이름을 나타냅니다.

DestinationURL 매개변수는 대상 호스트 물리적 시스템의 연결 URL입니다. 대상 시스템은 동일한 하이퍼바이저를 사용하고 **libvirt** 가 실행되고 있는 동일한 버전의 Red Hat Enterprise Linux를 실행해야 합니다.



참고

일반 마이그레이션 및 피어 투 피어 마이그레이션을 위한 ***DestinationURL*** 매개변수는 다른 의미가 있습니다.

- 일반 마이그레이션: ***DestinationURL*** 은 소스 게스트 가상 머신에 표시된 대로 대상 호스트 물리적 시스템의 URL입니다.
- 피어 투 피어 마이그레이션: ***DestinationURL*** 은 소스 호스트 물리적 시스템에서 표시된 대로 대상 호스트 물리적 시스템의 URL입니다.

명령을 입력하면 대상 시스템의 루트 암호를 입력하라는 메시지가 표시됩니다.

중요

마이그레이션이 성공하려면 소스 서버의 **/etc/hosts** 파일에 있는 대상 호스트 물리적 시스템에 대한 항목이 필요합니다. 다음 예에 표시된 대로 이 파일에서 대상 호스트 물리적 시스템의 IP 주소 및 호스트 이름을 입력합니다. 이 예에서는 대상 호스트 물리적 시스템의 IP 주소와 호스트 이름을 대체합니다.

```
10.0.0.20 host2.example.com
```

예: virsh를 사용한 실시간 마이그레이션

이 예에서는 **host1.example.com** 에서 **host2.example.com** 으로 마이그레이션합니다. 환경의 호스트 물리적 시스템 이름을 변경합니다. 이 예에서는 **guest1-rhel6-64** 라는 가상 머신을 마이그레이션합니다.

이 예제에서는 공유 스토리지를 완전히 구성하고 모든 사전 요구 사항을 충족한다고 가정합니다(여기에 나열됨: 마이그레이션 요구 사항).

1. 게스트 가상 머신이 실행 중인지 확인
   소스 시스템 **host1.example.com** 에서 **guest1-rhel6-64** 가 실행 중인지 확인합니다.

```
[root@host1 ~]# virsh list
Id Name                 State
----------------------------------
 10 guest1-rhel6-64     running
```

2. 게스트 가상 머신 마이그레이션
   다음 명령을 실행하여 게스트 가상 머신을 대상 **host2.example.com** 으로 실시간 마이그레이션합니다. libvirt에 전체 액세스 권한이 필요함을 확인하기 위해 대상 URL의 끝에 **/system** 을 추가합니다.

```
# virsh migrate --live guest1-rhel6-64 qemu+ssh://host2.example.com/system
```

   명령을 입력하면 대상 시스템의 루트 암호를 입력하라는 메시지가 표시됩니다.

3. wait
   마이그레이션은 게스트 가상 시스템의 로드 및 크기에 따라 다소 시간이 걸릴 수 있습니다. **virsh** 는 오류만 보고합니다. 게스트 가상 시스템은 완전히 마이그레이션될 때까지 소스 호스트 물리적 시스템에서 계속 실행됩니다.

참고

마이그레이션 중에 완료 백분율 표시 수는 프로세스가 완료되기 전에 여러 번 감소할 수 있습니다. 이는 마이그레이션을 시작한 후 변경된 소스 메모리 페이지를 다시 복사해야 하므로 전체 진행 상황을 다시 계산하기 때문입니다. 따라서 이 동작이 예상되며 마이그레이션에 문제가 표시되지 않습니다.

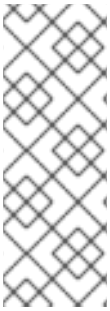
4. 게스트 가상 머신이 대상 호스트에 도착했는지 확인합니다.
   대상 시스템 **host2.example.com** 에서 **guest1-rhel6-64** 가 실행 중인지 확인합니다.

```
[root@host2 ~]# virsh list
Id Name                 State
----------------------------------
 10 guest1-rhel6-64     running
```

실시간 마이그레이션이 완료되었습니다.

참고

libvirt는 TLS/SSL, UNIX 소켓, SSH 및 암호화되지 않은 TCP를 포함한 다양한 네트워킹 방법을 지원합니다. 다른 방법을 사용하는 방법에 대한 자세한 내용은 5장. *원격 게스트 관리*를 참조하십시오.



참고

virsh migrate 명령을 사용하여 실행 중이 아닌 게스트 가상 머신을 마이그레이션할 수 없습니다. 실행 중인 게스트 가상 머신을 마이그레이션하려면 다음 스크립트를 사용해야 합니다.

```
virsh dumpxml Guest1 > Guest1.xml
virsh -c qemu+ssh://<target-system-FQDN>  define Guest1.xml
virsh undefine Guest1
```

4.4.1. virsh로 마이그레이션에 대한 추가 팁

각 마이그레이션이 별도의 명령 쉘에서 실행되는 여러 개의 동시 실시간 마이그레이션을 수행할 수 있습니다. 그러나 이 작업은 주의해서 수행해야 하며 각 마이그레이션 인스턴스가 각 측면(소스 및 대상)에서 하나의 MAX_CLIENT를 사용하므로 주의해야 합니다. 기본 설정은 20이므로 설정을 변경하지 않고 10개의 인스턴스를 실행하기에 충분합니다. 설정을 변경해야 하는 경우 절차 4.1. "libvirtd.conf 구성" 절차를 참조하십시오.

1. 절차 4.1. "libvirtd.conf 구성" 에 설명된 대로 libvirtd.conf 파일을 엽니다.
2. 처리 제어 섹션을 찾습니다.

```
#################################################################
#
# Processing controls
#

# The maximum number of concurrent client connections to allow
# over all sockets combined.
#max_clients = 20


# The minimum limit sets the number of workers to start up
# initially. If the number of active clients exceeds this,
# then more threads are spawned, upto max_workers limit.
# Typically you'd want max_workers to equal maximum number
# of clients allowed
#min_workers = 5
#max_workers = 20


# The number of priority workers. If all workers from above
# pool will stuck, some calls marked as high priority
# (notably domainDestroy) can be executed in this pool.
#prio_workers = 5

# Total global limit on concurrent RPC calls. Should be
# at least as large as max_workers. Beyond this, RPC requests
# will be read into memory and queued. This directly impact
# memory usage, currently each request requires 256 KB of
# memory. So by default upto 5 MB of memory is used
#
# XXX this isn't actually enforced yet, only the per-client
# limit is used so far
#max_requests = 20
```

```
# Limit on concurrent requests from a single client
# connection. To avoid one client monopolizing the server
# this should be a small fraction of the global max_requests
# and max_workers parameter
#max_client_requests = 5

#################################################################
```

3. ***max_clients*** 및 ***max_workers*** 매개 변수 설정을 변경합니다. 두 매개변수 모두에서 숫자가 동일해야 합니다. ***max_clients*** 는 마이그레이션당 2개의 클라이언트를 사용하고***max_workers*** 는 종료 단계에서 수행 단계에서 대상의 작업자 1명과 대상의 작업자 1개를 소스에 사용합니다.

중요

max_clients 및 ***max_workers*** 매개 변수 설정은 libvirtd 서비스에 대한 모든 게스트 가상 머신 연결에 적용됩니다. 즉, 동일한 게스트 가상 머신을 사용하고 동시에 마이그레이션을 수행하는 모든 사용자는 ***max_clients*** 및 ***max_workers*** 매개 변수 설정에 설정된 제한으로도 보류됩니다. 따라서 동시 실시간 마이그레이션을 수행하기 전에 최대값을 주의 깊게 고려해야 합니다.

4. 파일을 저장하고 서비스를 다시 시작합니다.

참고

시작되었지만 아직 인증되지 않은 ssh 세션이 너무 많기 때문에 마이그레이션 연결이 끊어지는 경우가 있을 수 있습니다. 기본적으로 **sshd** 는 언제든지 "사전 인증됨"에 있는 세션만 10개를 허용합니다. 이 설정은 sshd 구성 파일의 **MaxStartups** 매개변수(**/etc/ssh/sshd_config**)에 의해 제어되며 일부 조정이 필요할 수 있습니다. DoS 공격을 방지하기 위해 제한 사항이 배치되고 일반적으로 리소스를 과도하게 사용할 수 있으므로 이 매개변수를 조정하는 작업은 주의해야 합니다. 이 값을 너무 높게 설정하면 목적을 무효화합니다. 이 매개변수를 변경하려면 **/etc/ssh/sshd_config** 파일을 편집하고 **MaxStartups** 행의 시작 위치에서 **#** 을 제거하고, **10** (기본값)을 더 높은 숫자로 변경합니다. 파일을 저장하고**sshd** 서비스를 다시 시작하십시오. 자세한 내용은 **sshd_config** 매뉴얼 페이지를 참조하십시오.

4.4.2. virsh migrate 명령에 대한 추가 옵션

--live 외에도 virsh migrate는 다음 옵션을 허용합니다.

- **--direct** - 직접 마이그레이션에 사용
- **--p2p** - 피어 투 피어 마이그레이션에 사용
- **--tunnelled** - 터널링된 마이그레이션에 사용
- **--persistent** - 대상 호스트 물리적 시스템의 영구 상태로 도메인을 유지합니다.
- **--undefinesource** - 소스 호스트 물리적 시스템에서 게스트 가상 머신 제거
- **--suspend** - 대상 호스트 물리적 머신의 일시 정지 상태로 둡니다.
- **--change-protection** - 마이그레이션이 진행되는 동안 도메인에 호환되지 않는 구성 변경이 수행되지 않도록 강제 적용합니다. 하이퍼바이저에서 이 옵션을 암시적으로 활성화하지만 하이퍼바이저에 변경 보호 지원이 없는 경우 이 옵션을 명시적으로 사용하여 마이그레이션을 거부할 수 있습니다.

- **--unsafe** - 모든 안전 절차를 무시하고 마이그레이션을 강제 실행합니다.
- **--verbose** - 발생 시 마이그레이션 진행 상황을 표시합니다.
- **--abort-on-error** - 마이그레이션 프로세스 중에 소프트 오류(예: I/O 오류)가 발생하는 경우 마이그레이션을 취소합니다.
- **--migrateuri** - 일반적으로 생략된 마이그레이션 URI입니다.
- **--domain** *[string]*- 도메인 이름, ID 또는 uuid
- **--desturi** *[string]*- 클라이언트(정상 마이그레이션) 또는 source(p2p 마이그레이션)에서 표시된 대로 대상 호스트 물리적 시스템의 연결 URI
- **--migrateuri** - 마이그레이션 URI, 일반적으로 생략할 수 있습니다.
- **--timeout** *[seconds]*- 실시간 마이그레이션 카운터가 N초를 초과하면 게스트 가상 머신이 강제로 일시 중지됩니다. 실시간 마이그레이션에서만 사용할 수 있습니다. 제한 시간이 시작되면 일시 중단된 게스트 가상 머신에서 마이그레이션이 계속됩니다.
- **--D NAME** *[string]* - 마이그레이션 중에 게스트 가상 머신의 이름을 새 이름으로 변경합니다(지원되는 경우)
- **--** XML - 표시된 파일 이름은 기본 스토리지에 액세스하는 소스와 대상 간의 이름 지정 차이점과 같이 도메인 XML의 호스트별 부분에 대한 더 큰 변경 집합을 제공하기 위해 대상에 사용할 대체 XML 파일을 제공하는 데 사용할 수 있습니다. 이 옵션은 일반적으로 생략됩니다.

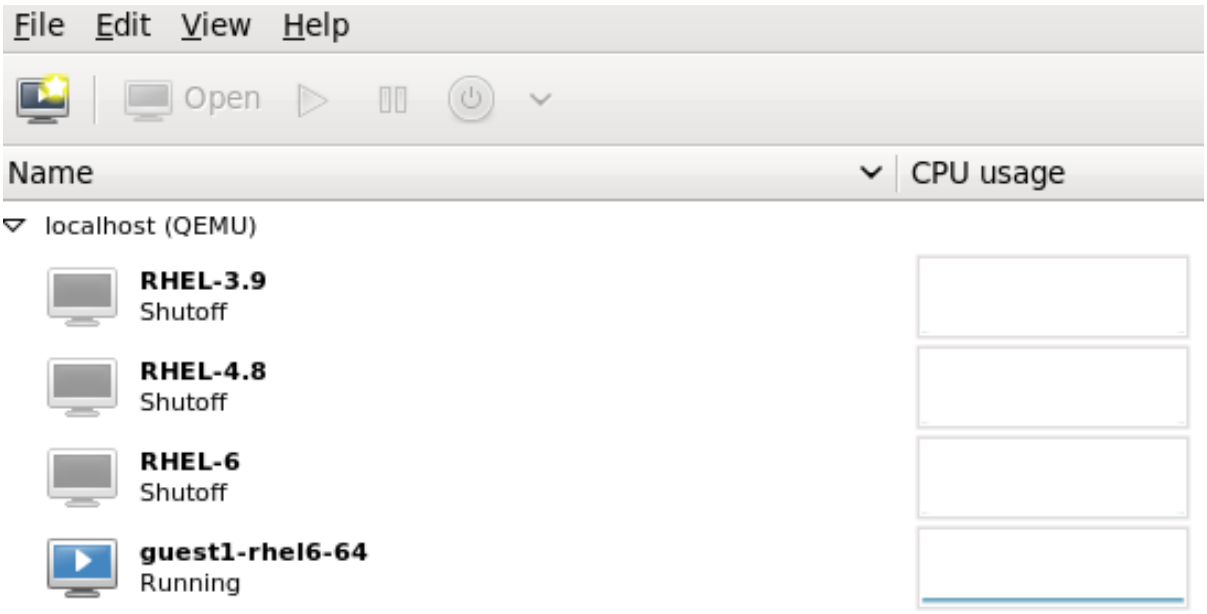
자세한 내용은 virsh man 페이지를 참조하십시오.

4.5. VIRT-MANAGER로 마이그레이션

이 섹션에서는 호스트 물리적 시스템에서 다른 호스트로 **virt-manager** 를 사용하는 KVM 게스트 가상 머신을 마이그레이션하는 방법을 설명합니다.

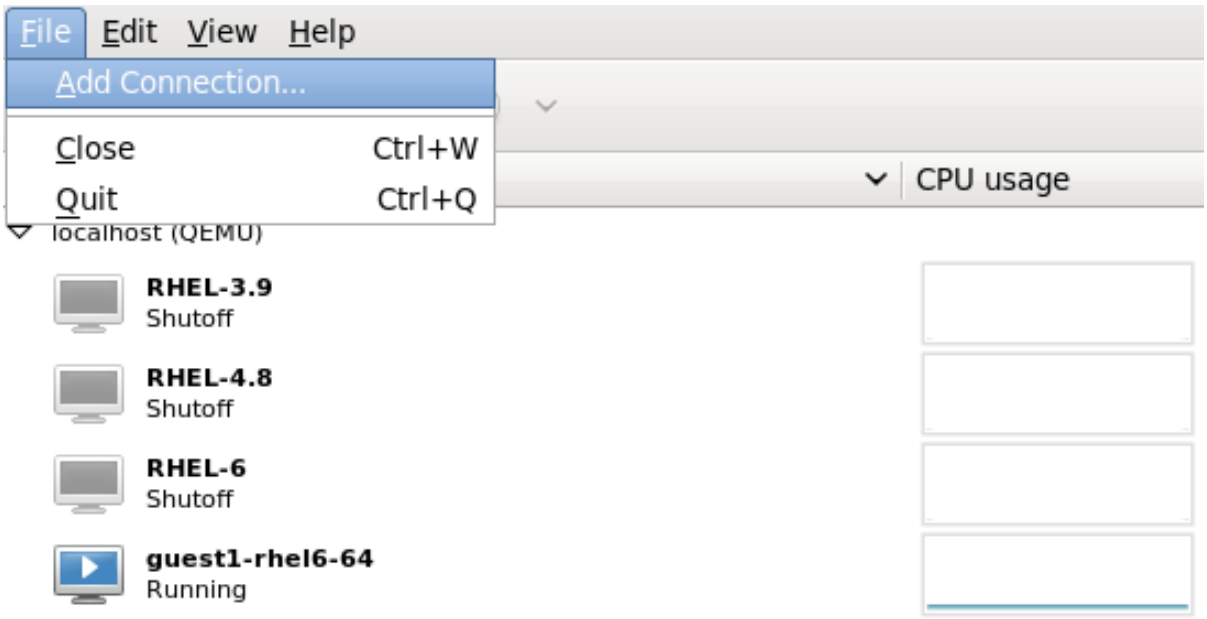
1. virt-manager를 엽니다.
   virt-manager 를 엽니다. 메인 메뉴 모음에서Applications → System Tools → Virtual Machine Manager 를 선택하여 **virt-manager** 를 시작합니다.

그림 4.1. virt-Manager 메인 메뉴



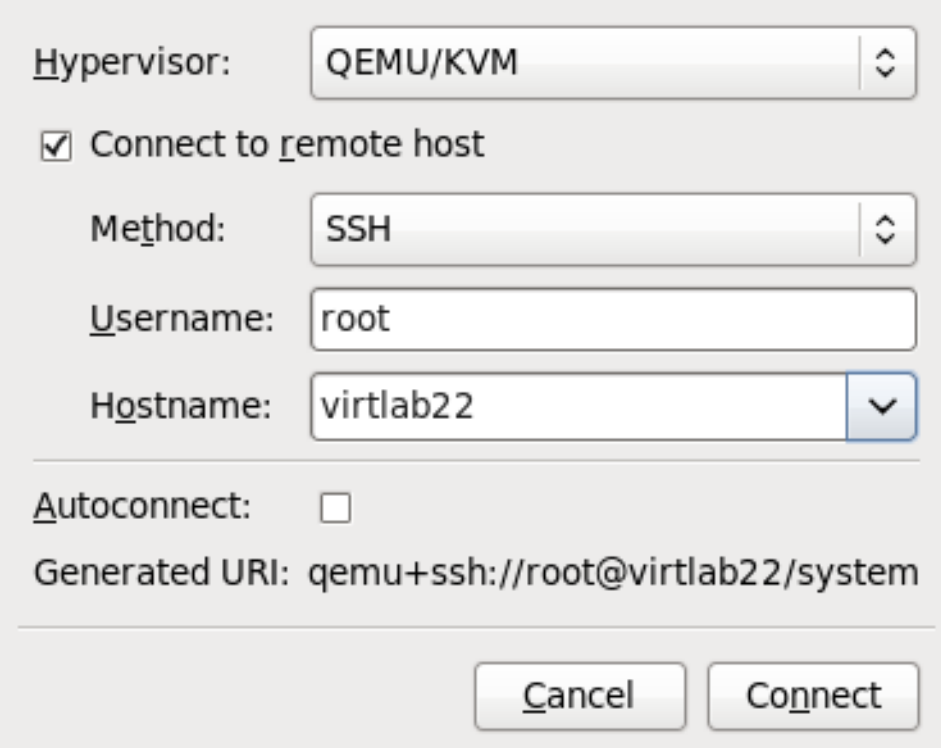
2. 대상 호스트 물리적 머신에 연결
파일 메뉴를 클릭하여 대상 호스트 물리적 시스템에 연결한 다음연결 추가 를 클릭합니다.

그림 4.2. 연결 추가 창을 엽니다.



3. 연결 추가
연결 추가 창이 나타납니다.

그림 4.3. 대상 호스트 물리적 머신에 연결 추가

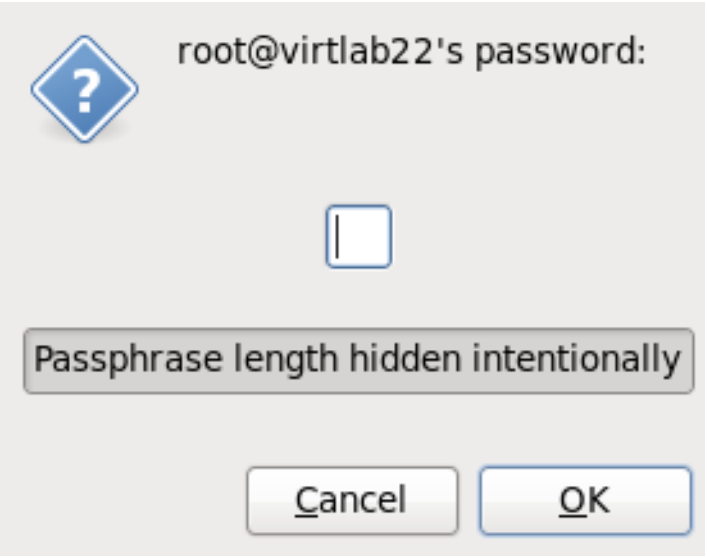


다음 세부 정보를 입력합니다.

- 하이퍼바이저: QEMU/KVM 을 선택합니다.
- 방법: 연결 방법을 선택합니다.
- **username**: 원격 호스트 물리적 머신의 사용자 이름을 입력합니다.
- **hostname**: 원격 호스트 물리적 머신의 호스트 이름을 입력합니다.

연결 버튼을 클릭합니다. 이 예에서는 SSH 연결이 사용되므로 다음 단계에서 지정된 사용자의 암호를 입력해야 합니다.

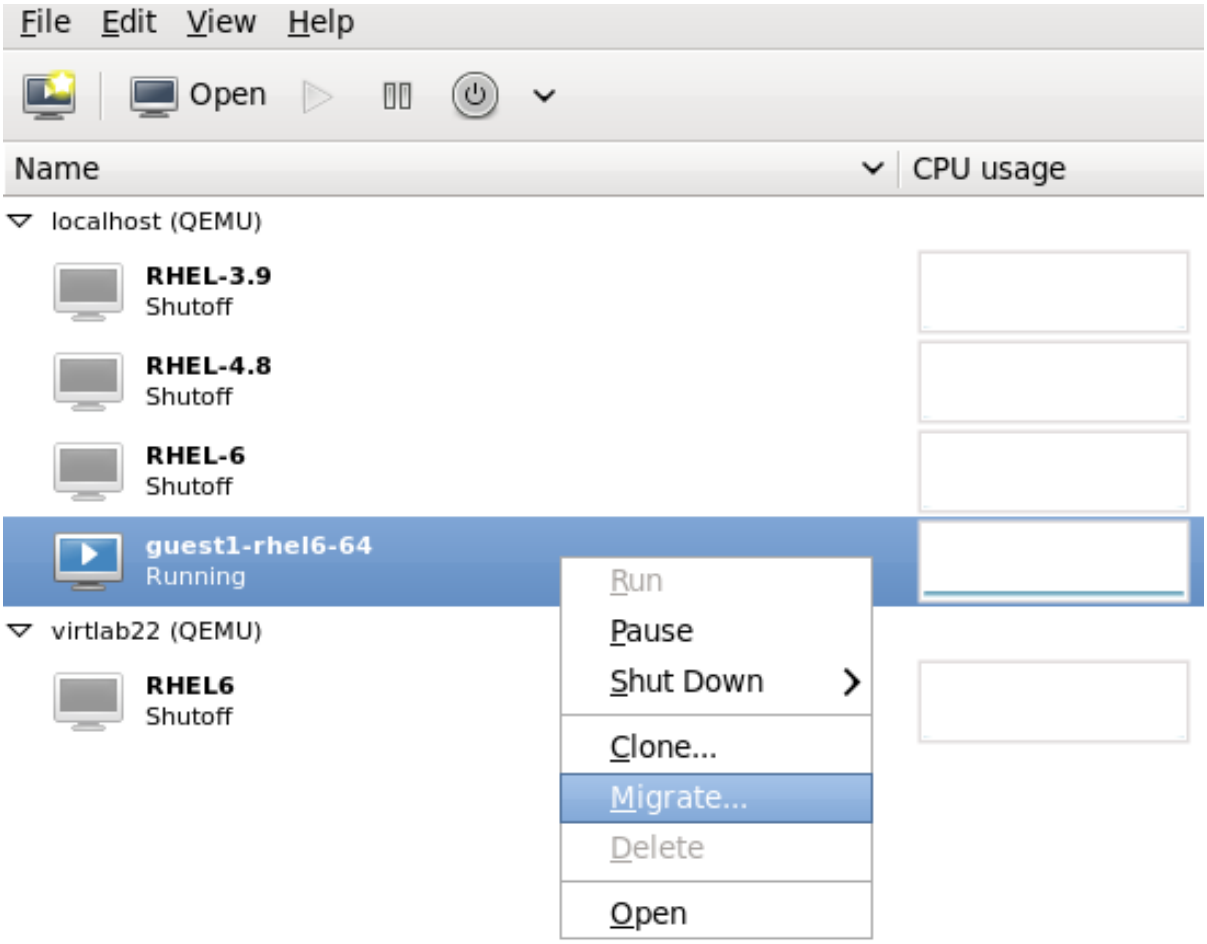
그림 4.4. 암호 입력



4. 게스트 가상 머신 마이그레이션

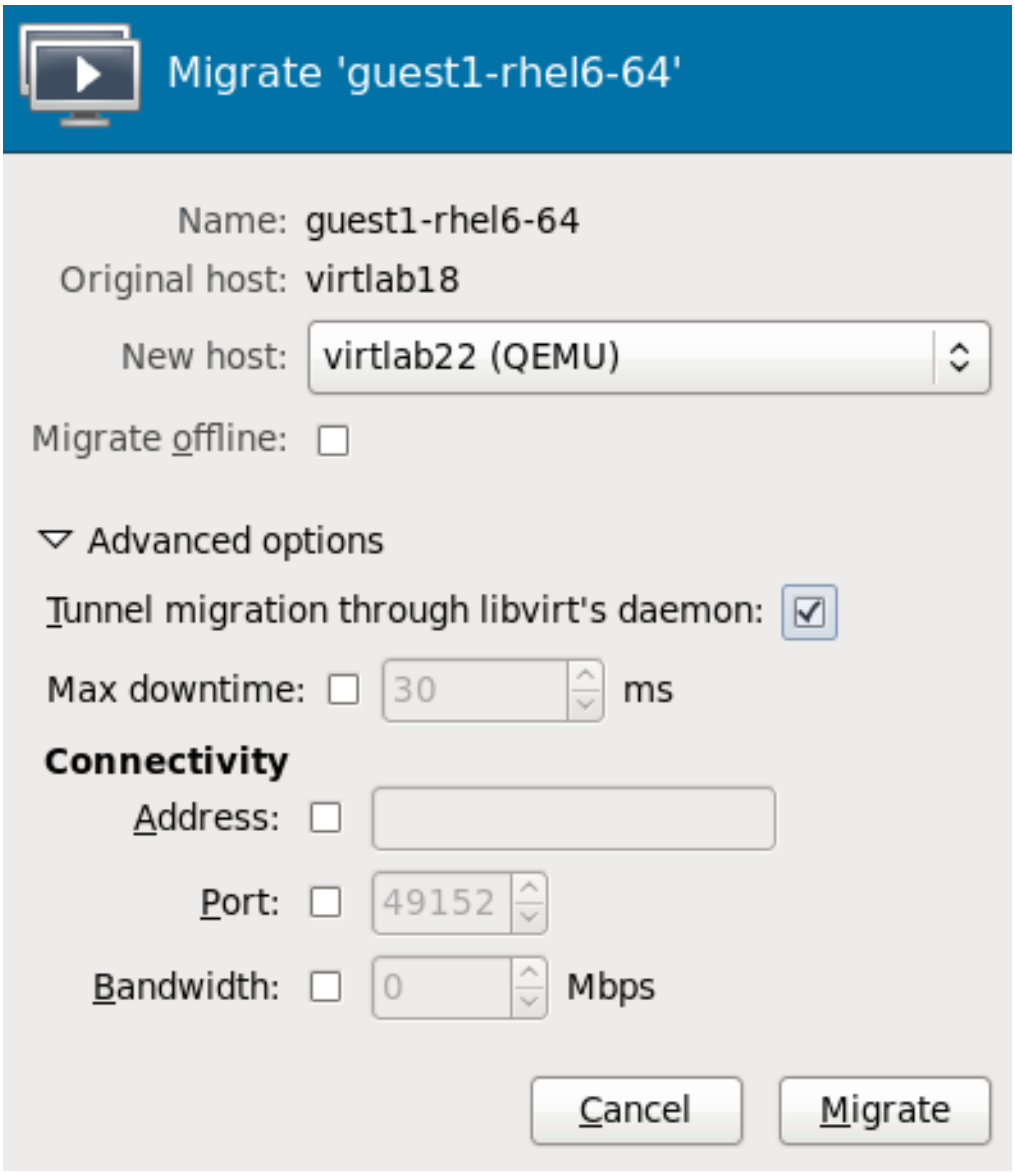
소스 호스트 물리적 시스템 내의 게스트 목록을 열고(호스트 이름의 왼쪽에 있는 작은 삼각형을 클릭) 마이그레이션할 게스트(이 예제에서는guest1-rhel 6-64)를 마우스 오른쪽 버튼으로 클릭하고 마이그레이션 을 클릭합니다.

그림 4.5. 마이그레이션할 게스트 선택



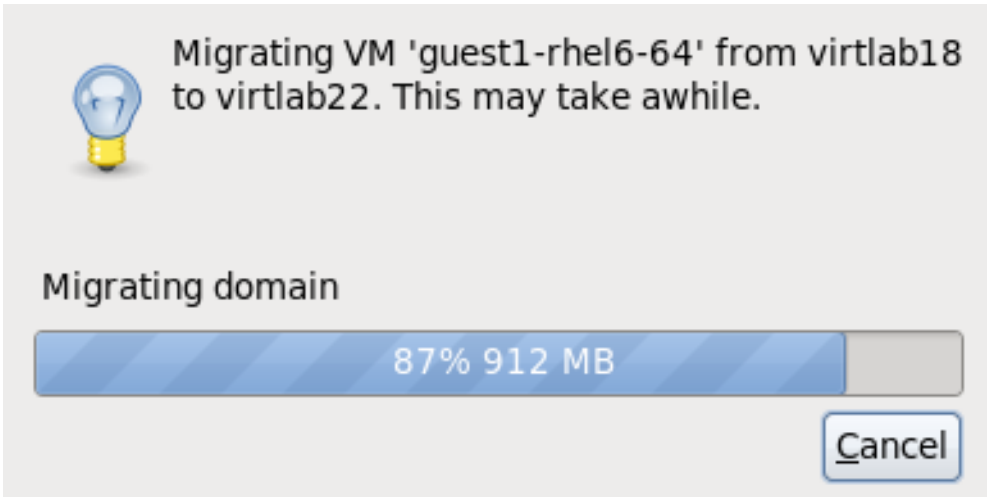
New Host (새 호스트) 필드에서 드롭다운 목록을 사용하여 guest 가상 머신을 마이그레이션할 호스트 물리적 시스템을 선택하고 마이그레이션을 클릭합니다.

그림 4.6. 대상 호스트 물리적 시스템 선택 및 마이그레이션 프로세스 시작



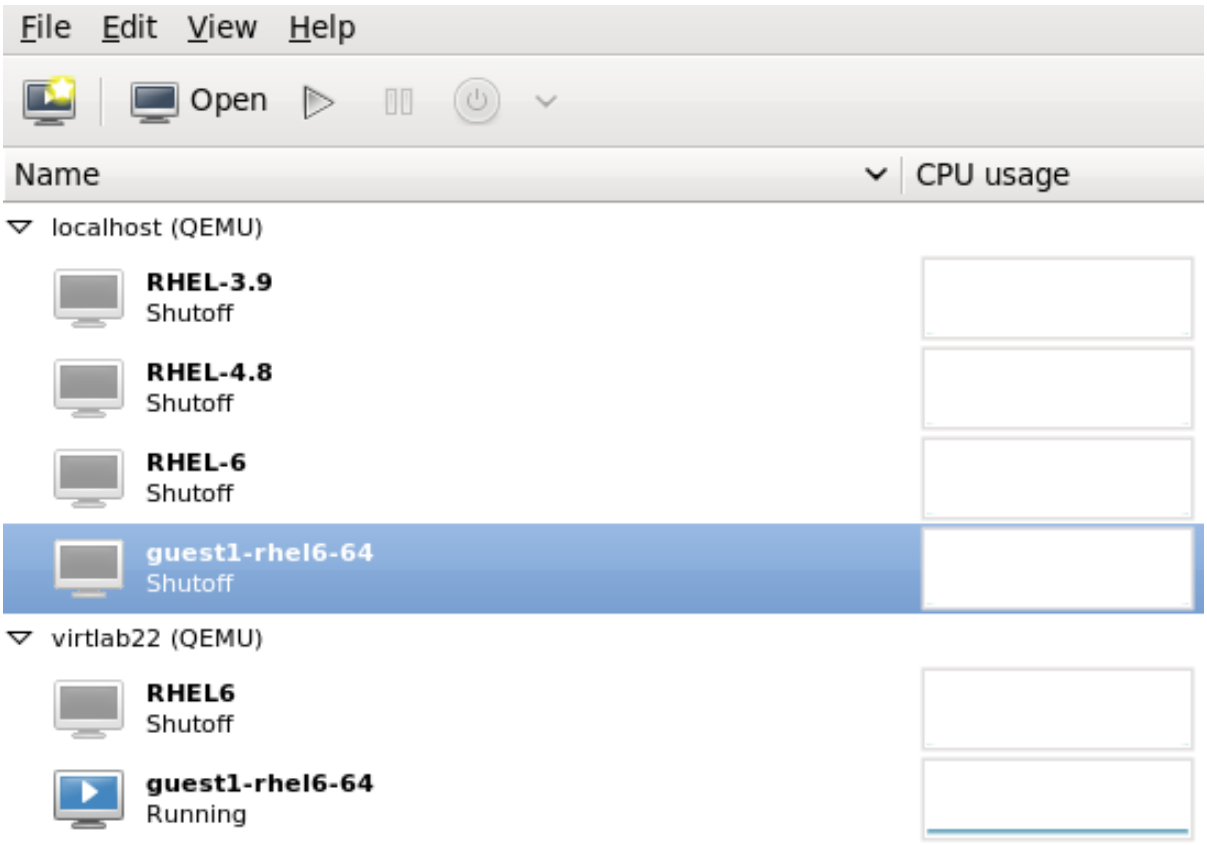
진행률 창이 나타납니다.

그림 4.7. 진행률 창



virt-manager 가 대상 호스트에서 실행 중인 새로 마이그레이션된 게스트 가상 머신을 표시합니다. 이제 소스 호스트 물리적 시스템에서 실행 중인 게스트 가상 머신이 Shutoff 상태에 나열됩니다.

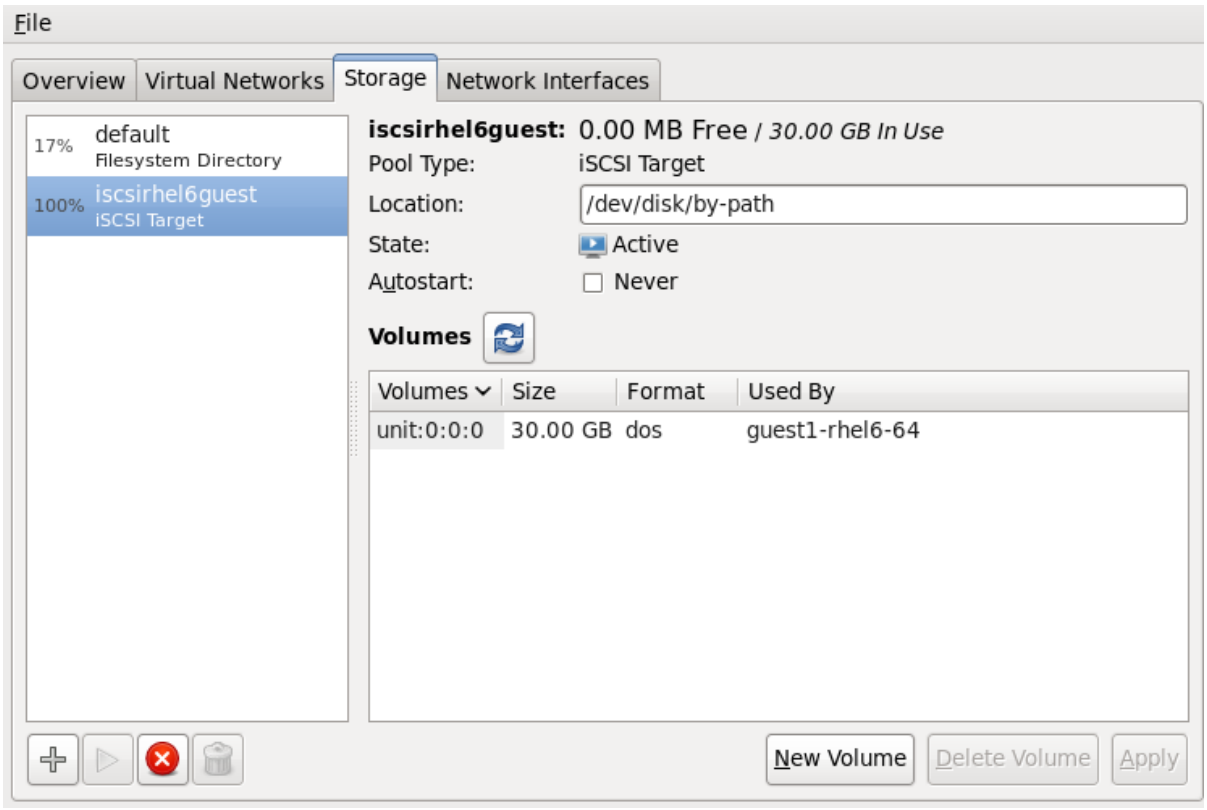
그림 4.8. 대상 호스트 물리적 머신에서 실행 중인 마이그레이션된 게스트 가상 머신



5. 선택 사항 - 호스트 물리적 머신의 스토리지 세부 정보 보기
편집 메뉴에서 연결 세부 정보를 클릭하면연결 세부 정보 창이 표시됩니다.

스토리지 탭을 클릭합니다. 대상 호스트 물리적 시스템의 iSCSI 대상 세부 정보가 표시됩니다. 마이그레이션된 게스트 가상 머신은 스토리지를 사용하여 로 나열됩니다.

그림 4.9. 스토리지 세부 정보



이 호스트는 다음 XML 구성으로 정의되었습니다.

그림 4.10. 대상 호스트 물리적 머신의 XML 구성

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

5장. 원격 게스트 관리

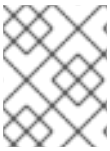
이 섹션에서는 **ssh** 또는 TLS 및 SSL을 사용하여 게스트를 원격으로 관리하는 방법을 설명합니다. SSH에 대한 자세한 내용은 *Red Hat Enterprise Linux 배포 가이드를 참조하십시오*.

5.1. SSH를 사용한 원격 관리

ssh 패키지는 원격 가상화 서버에 관리 기능을 안전하게 보낼 수 있는 암호화된 네트워크 프로토콜을 제공합니다. 설명된 방법은 SSH 연결을 통해 안전하게 터널링된**libvirt** 관리 연결을 사용하여 원격 시스템을 관리합니다. 모든 인증은 로컬 SSH 에이전트에서 수집한SSH 공개 키 암호화 및 암호 또는 암호를 사용하여 수행됩니다. 또한 각 게스트의 VNC 콘솔은SSH 를 통해 터널링됩니다.

다음을 포함하여 가상 머신을 원격으로 관리하기 위해 SSH 를 사용하는 문제에 유의하십시오.

- 가상 머신 관리를 위해 원격 시스템에 root가 로그인해야 합니다.
- 초기 연결 설정 프로세스가 느릴 수 있습니다.
- 모든 호스트 또는 게스트에서 사용자의 키를 취소하는 표준 또는 간단한 방법은 없습니다.
- SSH는 원격 머신 수가 많으면 제대로 확장되지 않습니다.



참고

Red Hat Virtualization은 많은 수의 가상 머신을 원격으로 관리할 수 있도록 지원합니다. 자세한 내용은 Red Hat Virtualization 설명서를 참조하십시오.

ssh 액세스에 다음 패키지가 필요합니다.

- openssh
- openssh-askpass
- openssh-clients
- openssh-server

virt-manager에 대한 암호 없는 또는 암호 관리 SSH 액세스 구성

다음 지침은 처음부터 시작 중이며 SSH 키가 아직 설정되어 있지 않다고 가정합니다. SSH 키를 설정하고 다른 시스템에 복사한 경우 이 절차를 건너뛸 수 있습니다.

중요

SSH 키는 사용자에 따라 다르며 소유자만 사용할 수 있습니다. 키의 소유자는 생성 된 사람입니다. 키는 공유할 수 없습니다.

원격 호스트에 연결할 키를 보유한 사용자가 **virt-manager** 를 실행해야 합니다. 즉, 루트가 아닌 사용자가 원격 시스템을 관리하는 경우 권한 없는 모드에서 **virt-manager** 를 실행해야 합니다. 원격 시스템이 로컬 root 사용자가 관리하는 경우 SSH 키는 root가 소유하고 생성해야 합니다.

virt-manager 를 사용하여 권한이 없는 사용자로 로컬 호스트를 관리할 수 없습니다.

1. 선택 사항: 사용자 변경

필요한 경우 사용자를 변경합니다. 이 예에서는 로컬 root 사용자를 사용하여 다른 호스트 및 로컬 호스트를 원격으로 관리합니다.

```
$ su -
```

2. SSH 키 쌍 생성
   virt-manager 가 사용되는 시스템에서 공개 키 쌍을 생성합니다. 이 예에서는~/.ssh/ 디렉터리의 기본 키 위치를 사용합니다.

   ```
   # ssh-keygen -t rsa
   ```

3. 원격 호스트에 키 복사
   암호 없이 원격 로그인하거나 암호를 사용하는 경우 SSH 키를 관리 중인 시스템에 배포해야 합니다. **ssh-copy-id** 명령을 사용하여 제공된 시스템 주소(예:***root@host2.example.com***)에서 root 사용자에게 키를 복사합니다.

   ```
   # ssh-copy-id -i ~/.ssh/id_rsa.pub root@host2.example.com
   root@host2.example.com's password:
   ```

   이제 **ssh root@host2.example.com** 명령을 사용하여 머신에 로그인하고**.ssh/authorized_keys** 파일을 확인하여 예기치 않은 키가 추가되지 않았는지 확인합니다.

   필요에 따라 다른 시스템에 대해 반복합니다.

4. 선택사항: ssh-agent에 암호를 추가합니다.
   아래 지침은 기존 ssh-agent에 암호를 추가하는 방법을 설명합니다. ssh-agent가 실행되고 있지 않으면 실행되지 않습니다. 오류 또는 충돌을 방지하려면 SSH 매개 변수가 올바르게 설정되어 있는지 확인합니다. 자세한 내용은 *Red Hat Enterprise Linux 배포 가이드*를 참조하십시오.

   필요한 경우 SSH 키의 암호를 **ssh-agent** 에 추가합니다. 로컬 호스트에서 다음 명령을 사용하여 암호 없이 로그인할 수 있도록 암호를 추가합니다.

   ```
   # ssh-add ~/.ssh/id_rsa
   ```

   SSH 키가 원격 시스템에 추가됩니다.

libvirt 데몬**(libvirtd)**

libvirt 데몬은 가상 머신을 관리하기 위한 인터페이스를 제공합니다. 관리를 필요로 하는 모든 원격 호스트에 **libvirtd** 데몬이 설치되어 실행되고 있어야 합니다.

```
$ ssh root@somehost
# chkconfig libvirtd on
# service libvirtd start
```

libvirtd 및 SSH 가 구성된 후 가상 시스템에 원격으로 액세스하고 관리할 수 있어야 합니다. 이 시점에서 **VNC** 를 사용하여 게스트에 액세스할 수도 있습니다.

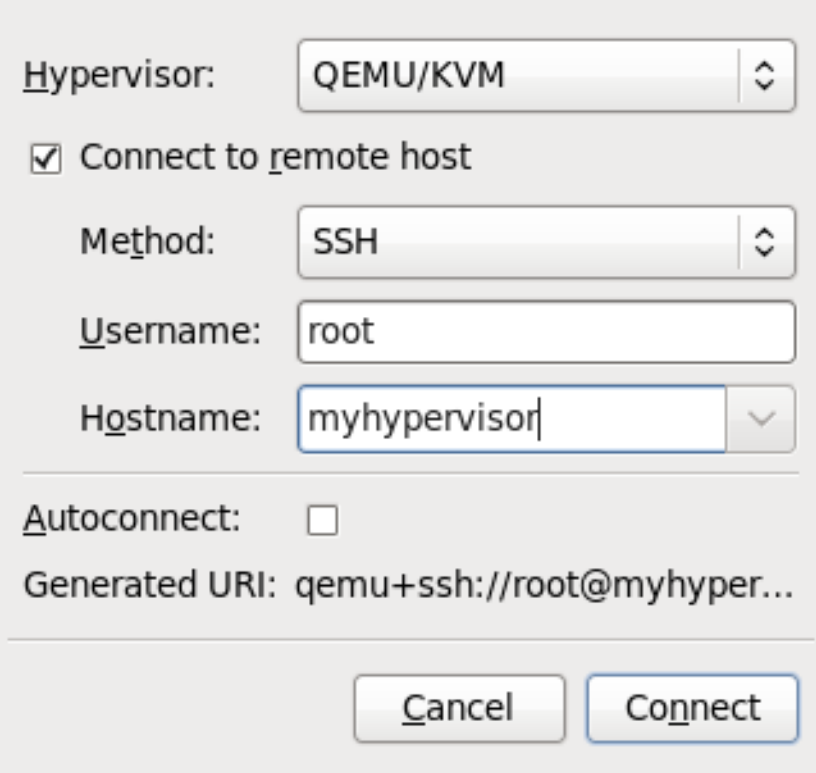
virt-manager를 사용하여 원격 호스트에 액세스

virt-manager GUI 도구를 사용하여 원격 호스트를 관리할 수 있습니다. 암호 없이 로그인할 수 있으려면 SSH 키는 virt-manager를 실행하는 사용자에 속해야 합니다.

1. virt-manager를 시작합니다.

2. 파일->연결 추가 메뉴를 엽니다.

그림 5.1. 연결 메뉴 추가



3. 드롭다운 메뉴를 사용하여 하이퍼바이저 유형을 선택하고, 원격 호스트에 연결 확인란을 클릭하여 Connection Method (이 경우 SSH를 통한 원격 터널)를 열고 원하는사용자 이름과 호스트 이름을 입력한 다음 연결을 클릭합니다.

5.2. TLS 및 SSL을 통한 원격 관리

TLS 및 SSL을 사용하여 가상 머신을 관리할 수 있습니다. TLS 및 SSL은 향상된 확장성을 제공하지만 ssh보다 더 복잡할 수 있습니다(5.1절. "SSH를 사용한 원격 관리"참조). TLS 및 SSL은 웹 브라우저에서 보안 연결을 위해 사용하는 것과 동일한 기술입니다. libvirt 관리 연결은 x509 인증서를 기반으로 안전하게 암호화되고 인증된 들어오는 연결에 대한 TCP 포트를 엽니다. 다음 절차에서는 TLS 및 SSL 관리를 위한 인증 인증서 생성 및 배포에 대한 지침을 제공합니다.

절차 5.1. TLS 관리를 위한 인증 기관(CA) 키 생성

1. 시작하기 전에 **certtool** 유틸리티가 설치되었는지 확인합니다. 그렇지 않은 경우:

```
# yum install gnutls-utils
```

2. 다음 명령을 사용하여 개인 키를 생성합니다.

```
# certtool --generate-privkey > cakey.pem
```

3. 키가 생성되면 다음 단계는 서명 파일을 작성하여 키를 자체 서명할 수 있도록 하는 것입니다. 이렇게 하려면 서명 세부 정보가 있는 파일을 생성하고 이름을 **ca.info** 로 지정합니다. 이 파일에는 다음이 포함되어야 합니다.

```
# vim ca.info
```

```
cn = Name of your organization
ca
cert_signing_key
```

4. 다음 명령을 사용하여 자체 서명 키를 생성합니다.

```
# certtool --generate-self-signed --load-privkey cakey.pem --template ca.info --outfile
cacert.pem
```

파일이 생성되면 **rm** 명령을 사용하여 **ca.info** 파일을 삭제할 수 있습니다. 생성 프로세스에서 발생하는 파일의 이름은 **cacert.pem** 입니다. 이 파일은 공개 키(certificate)입니다. 로드된 파일 **cakey.pem** 은 개인 키입니다. 이 파일은 공유 공간에 보관해서는 안 됩니다. 이 키를 비공개로 유지합니다.

5. **/etc/pki/CA/cacert.pem** 디렉터리의 모든 클라이언트 및 서버에**cacert.pem** 인증 인증 인증서 파일을 설치하여 CA에서 발급한 인증서를 신뢰할 수 있도록 합니다. 이 파일의 내용을 보려면 다음을 실행합니다.

```
# certtool -i --infile cacert.pem
```

이는 CA를 설정하는 데 필요한 모든 것입니다. 클라이언트 및 서버의 인증서를 발급하려면 CA의 개인 키를 안전하게 보관해야 합니다.

절차 5.2. 서버 인증서 발급

다음 절차에서는 서버 호스트 이름으로 X.509 CN(CommonName) 필드를 설정하여 인증서를 발급하는 방법을 설명합니다. CN은 클라이언트가 서버에 연결하는 데 사용할 호스트 이름과 일치해야 합니다. 이 예에서 클라이언트는 URI: **qemu://mycommonname/system** 을 사용하여 서버에 연결하므로 CN 필드가 동일해야 합니다.

1. 서버에 대한 개인 키를 만듭니다.

```
# certtool --generate-privkey > serverkey.pem
```

2. **server.info** .info라는 템플릿 파일을 먼저 생성하여 CA의 개인 키 서명을 생성합니다. CN이 서버의 호스트 이름과 동일하게 설정되어 있는지 확인합니다.

```
organization = Name of your organization
cn = mycommonname
tls_www_server
encryption_key
signing_key
```

3. 다음 명령을 사용하여 인증서를 생성합니다.

```
# certtool --generate-certificate --load-privkey serverkey.pem --load-ca-certificate cacert.pem
--load-ca-privkey cakey.pem \ --template server.info --outfile servercert.pem
```

4. 그러면 두 개의 파일이 생성됩니다.

- **serverkey.pem** - 서버의 개인 키
- **servercert.pem** - 서버의 공개 키

개인 키의 위치를 비밀로 유지합니다. 파일의 내용을 보려면 다음 명령을 수행합니다.

```
# certtool -i --inifile servercert.pem
```

이 파일을 열 때 **CN=** 매개변수는 이전에 설정한 CN과 동일해야 합니다. 예를 들면 **mycommonname** 입니다.

5. 다음 위치에 두 파일을 설치합니다.

   - **serverkey.pem** - 서버의 개인 키입니다. 이 파일을 다음 위치에 배치합니다. **/etc/pki/libvirt/private/serverkey.pem**
   - **servercert.pem** - 서버의 인증서입니다. 서버의 다음 위치에 설치합니다. **/etc/pki/libvirt/servercert.pem**

절차 5.3. 클라이언트 인증서 발급

1. 모든 클라이언트(예: virt-manager와 같은 libvirt에 연결된 모든 프로그램)의 경우 X.509 Distinguished Name(DN)을 적절한 이름으로 설정해야 합니다. 이는 기업 차원에서 결정해야 합니다.

   예를 들어 다음 정보가 사용됩니다.

   ```
   C=USA,ST=North Carolina,L=Raleigh,O=Red Hat,CN=name_of_client
   ```

   이 프로세스는 절차 5.2. "서버 인증서 발급" 와 매우 유사하지만 다음과 같은 예외가 명시되어 있습니다.

2. 다음 명령을 사용하여 개인 키를 만듭니다.

   ```
   # certtool --generate-privkey > clientkey.pem
   ```

3. 먼저 **client.info** .info라는 템플릿 파일을 생성하여 CA의 개인 키 서명을 생성합니다. 파일에는 다음 내용이 포함되어야 합니다(지역/위치를 반영하도록 필드를 사용자 지정해야 함).

   ```
   country = USA
   state = North Carolina
   locality = Raleigh
   organization = Red Hat
   cn = client1
   tls_www_client
   encryption_key
   signing_key
   ```

4. 다음 명령을 사용하여 인증서에 서명합니다.

   ```
   # certtool --generate-certificate --load-privkey clientkey.pem --load-ca-certificate cacert.pem \
   --load-ca-privkey cakey.pem --template client.info --outfile clientcert.pem
   ```

5. 클라이언트 머신에 인증서를 설치합니다.

   ```
   # cp clientkey.pem /etc/pki/libvirt/private/clientkey.pem
   # cp clientcert.pem /etc/pki/libvirt/clientcert.pem
   ```

5.3. 전송 모드

원격 관리를 위해 **libvirt** 는 다음 전송 모드를 지원합니다.

TLS(Transport Layer Security)

SSL 3.1(Transport Layer Security TLS 1.0)이 인증되고 암호화된 TCP/IP 소켓은 일반적으로 공용 포트 번호에서 수신 대기합니다. 이를 사용하려면 클라이언트 및 서버 인증서를 생성해야 합니다. 표준 포트는 16514입니다.

UNIX 소켓

UNIX 도메인 소켓은 로컬 시스템에서만 액세스할 수 있습니다. 소켓은 암호화되지 않으며 인증을 위해 UNIX 권한 또는 SELinux를 사용합니다. 표준 소켓 이름은 **/var/run/libvirt/libvirt-sock** 및 **/var/run/libvirt/libvirt-sock-ro** 입니다(읽기 전용 연결의 경우).

SSH

SSH(Secure Shell 프로토콜) 연결을 통해 전송. Netcat(nc 패키지)이 설치되어 있어야 합니다. libvirt 데몬(**libvirtd**)이 원격 시스템에서 실행되고 있어야 합니다. SSH 액세스를 위해 포트 22를 열어야 합니다. 일종의 SSH 키 관리(예: **ssh-agent** 유틸리티)를 사용해야 합니다. 그렇지 않으면 암호를 입력하라는 메시지가 표시됩니다.

ext

ext 매개 변수는 libvirt 범위를 벗어나는 방법으로 원격 시스템에 연결할 수 있는 모든 외부 프로그램에 사용됩니다. 이 매개변수는 지원되지 않습니다.

TCP

암호화되지 않은 TCP/IP 소켓. 프로덕션 용도는 권장되지 않지만 일반적으로 비활성화되어 있지만 신뢰할 수 있는 네트워크를 통해 테스트하거나 사용할 수 있습니다. 기본 포트는 16509입니다.

다른 값이 지정되지 않은 경우 기본 전송은 TLS입니다.

원격 URI

URI(Uniform Resource Identifier)는 **virsh** 및 libvirt 에서 원격 호스트에 연결하는 데 사용됩니다. URI를 **virsh** 명령의 **--connect** 매개 변수와 함께 사용하여 원격 호스트에서 단일 명령 또는 마이그레이션을 실행할 수도 있습니다. 원격 URI는 일반 로컬 URI를 사용하고 호스트 이름 또는 전송 이름을 추가하여 구성됩니다. 'remote'의 URI 체계를 사용하는 특수한 경우 는 원격 libvirtd 서버에 최적의 하이퍼바이저 드라이버를 검색하도록 지시합니다. 로컬 연결에 대해 NULL URI를 전달하는 것과 같습니다.

libvirt URI는 일반적인 형식을 사용합니다(대괄호 괄호의 내용, "[]", 선택적 함수를 나타냅니다).

```
driver[+transport]://[username@][hostname][:port]/path[?extraparameters]
```

하이퍼바이저(driver)가 QEMU인 경우 해당 경로는 필수입니다. XEN인 경우 선택 사항입니다.

다음은 유효한 원격 URI의 예입니다.

- qemu://hostname/
- xen://hostname/
- xen+ssh://hostname/

전송 방법 또는 호스트 이름은 외부 위치를 대상으로 지정되어야 합니다. 자세한 내용은 를 http://libvirt.org/guide/html/Application_Development_Guide-Architecture-Remote_URIs.html 참조하십시오.

원격 관리 매개변수의 예

- SSH 전송 및 SSH 사용자 이름 **virtuser** 를 사용하여 **host2** 라는 원격 KVM 호스트에 연결합니다. 각각에 대한 연결 명령은 [< **name**>] **[-readonly]** 입니다. 여기서 <**name** >은 여기에 설명된 대로 유효한 URI입니다. **virsh connect** 명령에 대한 자세한 내용은 다음을 참조하십시오.14.1.5절. “연결”

```
qemu+ssh://virtuser@hot2/
```

- TLS를 사용하여 **host2** 라는 호스트의 원격 KVM 하이퍼바이저에 연결합니다.

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

테스트 예

- 비표준 UNIX 소켓을 사용하여 로컬 KVM 하이퍼바이저에 연결합니다. 이 경우 UNIX 소켓의 전체 경로는 명시적으로 제공됩니다.

```
qemu+unix:///system?socket=/opt/libvirt/run/libvirt/libvirt-sock
```

- 포트 5000에서 IP 주소가 10.1.1.10인 서버에 암호화되지 않은 TCP/IP 연결을 사용하여 libvirt 데몬에 연결합니다. 기본 설정으로 테스트 드라이버를 사용합니다.

```
test+tcp://10.1.1.10:5000/default
```

추가 **URI** 매개변수

추가 매개 변수를 원격 URI에 추가할 수 있습니다. 표 5.1. “추가 URI 매개변수”아래 표는 인식된 매개변수를 다룹니다. 다른 모든 매개변수는 무시됩니다. 매개 변수 값은 URI로 이스케이프되어야 합니다(즉, 매개 변수 및 특수 문자가 URI 형식으로 변환되기 전에 물음표(?)를 추가합니다.

표 5.1. 추가 URI 매개변수

이름	전송 모드	설명	사용 예
name	모든 모드	이름이 원격 virConnectOpen 기능에 전달되었습니다. 이름은 일반적으로 원격 URI에서 전송, 호스트 이름, 포트 번호, 사용자 이름 및 추가 매개 변수를 제거하여 구성되지만, 매우 복잡한 경우 이름을 명시적으로 제공하는 것이 더 좋습니다.	name=qemu:///system

이름	전송 모드	설명	사용 예
command	SSH 및 ext	외부 명령 ext 전송에는 이 작업이 필요합니다. ssh의 경우 기본값은 ssh 입니다. PATH가 명령에 대해 검색됩니다.	command=/opt/openssh/bin/ssh
socket	Unix 및 ssh	기본값을 재정의하는 UNIX 도메인 소켓의 경로입니다. ssh 전송의 경우 원격 netcat 명령에 전달됩니다(넷cat 참조).	socket=/opt/libvirt/run/libvirt/libvirt-sock
netcat	ssh	**netcat** 명령을 사용하여 원격 시스템에 연결할 수 있습니다. 기본 netcat 매개변수는 **nc** 명령을 사용합니다. SSH 전송의 경우 libvirt는 아래 양식을 사용하여 SSH 명령을 구성합니다. ***command*** -p ***port*** [-l ***username***] ***hostname*** ***Netcat*** -U socket ***포트*,*username*** 및 ***hostname*** 매개 변수는 원격 URI의 일부로 지정할 수 있습니다. ***명령*,*netcat*** 및 ***socket*** 은 다른 추가 매개 변수에서 제공됩니다.	netcat=/opt/netcat/bin/nc
no_verify	tls	0이 아닌 값으로 설정하면 클라이언트 인증서가 비활성화됩니다. 클라이언트의 인증서 또는 IP 주소를 서버 확인을 비활성화하려면 libvirtd 구성을 변경해야 합니다.	no_verify=1
no_tty	ssh	0이 아닌 값으로 설정하면 원격 머신에 자동으로 로그인할 수 없는 경우 암호를 요청하는 ssh가 중지됩니다. . 터미널 에 액세스할 수 없는 경우 이 값을 사용합니다.	no_tty=1

6장. KVM으로 과다 할당

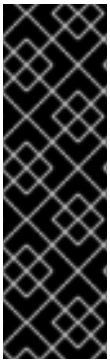
KVM 하이퍼바이저는 CPU 및 메모리를 자동으로 오버 커밋합니다. 즉, 시스템에 물리적 리소스가 있는 것보다 가상 머신에 더 많은 가상화 CPU와 메모리를 할당할 수 있습니다. 대부분의 프로세스는 할당된 리소스의 100 %에 항상 액세스하지 않기 때문에 가능합니다.

결과적으로 활용도가 낮은 가상화 서버 또는 데스크탑은 적은 수의 호스트에서 실행될 수 있으므로 서버 하드웨어에 대한 전력, 처리 및 투자를 줄일 수 있을 뿐만 아니라 시스템 리소스의 수를 줄일 수 있습니다.

6.1. 메모리 과다 할당

KVM 하이퍼바이저에서 실행되는 게스트 가상 머신에는 물리적 RAM의 전용 블록이 할당되지 않습니다. 대신 각 게스트 가상 머신은 호스트의 물리적 시스템의 Linux 커널이 요청된 경우에만 메모리를 할당하는 Linux 프로세스로 작동합니다. 또한 호스트의 메모리 관리자는 게스트 가상 시스템의 메모리를 실제 메모리와 스왑 공간으로 이동할 수 있습니다.

오버 커밋을 수행하려면 모든 게스트 가상 머신을 수용할 수 있도록 호스트 물리적 머신에 충분한 스왑 공간을 확보해야 하며 호스트 물리적 머신의 프로세스에 충분한 메모리를 할당해야 합니다. 기본적으로 호스트 물리적 시스템의 운영 체제는 최소 4GB의 스왑 공간과 함께 최대 4GB의 메모리가 필요합니다. 스왑 파티션에 적절한 크기를 결정하는 고급 지침은 Red Hat KnowledgeBase를 참조하십시오.



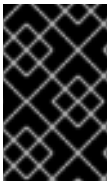
중요

오버 커밋은 일반적인 메모리 문제에 대한 이상적인 솔루션이 아닙니다. 메모리 부족을 처리하기 위해 권장되는 방법은 게스트당 메모리를 더 적게 할당하거나 호스트에 실제 메모리를 추가하거나 스왑 공간을 활용하는 것입니다.

가상 머신은 자주 교체되는 경우 느리게 실행됩니다. 또한 과다 할당으로 인해 시스템이 OOM(메모리 부족)이 발생하여 Linux 커널이 중요한 시스템 프로세스를 종료할 수 있습니다. 메모리 과다 할당을 결정하면 충분한 테스트가 수행되어야 합니다. 오버 커밋에 대한 지원은 Red Hat 지원에 문의하십시오.

과다 할당은 모든 게스트 가상 머신에서 작동하지 않지만 최소한의 집약적으로 데스크탑 가상화 설정에서 작동하거나 커널 동일 페이지 병합(KSM)을 사용하여 동일한 게스트 가상 머신을 여러 개 실행하는 것으로 나타났습니다.

KSM 및 오버 커밋에 대한 자세한 내용은 7장. *KSM* 에서 참조하십시오.



중요

장치 할당이 사용 중인 경우 할당된 장치를 사용하여 직접 메모리 액세스(DMA)를 활성화하려면 모든 가상 머신 메모리를 정적으로 사전 할당해야 합니다. 따라서 메모리 과다 할당은 장치 할당에서 지원되지 않습니다.

6.2. 가상화된 CPU 오버 커밋

KVM 하이퍼바이저는 가상화된 CPU 오버 커밋을 지원합니다. 게스트 가상 머신의 로드 제한에 허용하는 한 가상화된 CPU를 오버 커밋할 수 있습니다. VCPU 수를 100% 근처에 오버 커밋할 때 주의하십시오. 이로 인해 요청이 감소되거나 응답 시간을 사용할 수 없습니다.

단일 호스트 물리적 머신에 동일한 vCPU를 공유하지 않는 여러 게스트 가상 머신이 있는 경우 가상화된 CPU(vCPU)가 가장 적합합니다. KVM은 하나의 단일 호스트 물리적 시스템에서 하나의 물리적 CPU에 대해 5개의 VCPU(가상 시스템 5개)에서 100 % 미만의 부하가 있는 게스트 가상 머신을 안전하게 지원해야 합니다. KVM은 모든 시스템 간에 전환하여 부하가 균형을 유지합니다.

물리적 처리 코어 수보다 많은 게스트 가상 머신을 과다 할당하지 마십시오. 예를 들어 vCPU가 4개인 게스트 가상 시스템은 듀얼 코어 프로세서가 있지만 쿼드 코어 호스트가 있는 호스트 물리적 시스템에서 실행되어야 합니다. 또한 물리적 프로세서 코어당 총 할당된 vCPU 10개를 초과하지 않는 것이 좋습니다.

중요

광범위한 테스트 없이 프로덕션 환경에서 CPU를 과다 할당하지 마십시오. 오버 커밋된 환경에서 처리 리소스의 100%를 사용하는 애플리케이션은 불안정해질 수 있습니다. 배포 전에 테스트합니다.

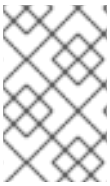
가상 머신에서 최상의 성능을 얻는 방법에 대한 자세한 내용은 Red Hat Enterprise Linux 6 Virtualization 조정 및 최적화 가이드 를 참조하십시오.

7장. KSM

공유 메모리의 개념은 최신 운영 체제에서 일반적입니다. 예를 들어 프로그램이 처음 시작되면 모든 메모리를 부모 프로그램과 공유합니다. 자식 또는 부모 프로그램이 이 메모리를 수정하려고 하면 커널은 새 메모리 영역을 할당하고, 원래 내용을 복사하고 프로그램이 이 새 영역을 수정할 수 있도록 합니다. 이 작업을 쓰기 시 복사라고 합니다.

KSM은 이 개념을 역순으로 사용하는 새로운 Linux 기능입니다. KSM을 사용하면 커널이 두 개 이상의 이미 실행 중인 프로그램을 검사하고 메모리를 비교할 수 있습니다. 메모리 영역 또는 페이지가 동일한 경우 KSM은 여러 동일한 메모리 페이지를 단일 페이지로 줄입니다. 그런 다음 이 페이지는 쓰기 시 복사로 표시됩니다. 페이지 콘텐츠를 게스트 가상 시스템에서 수정하면 해당 게스트 가상 머신에 대한 새 페이지가 생성됩니다.

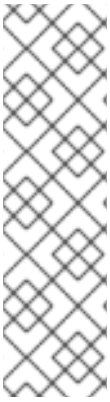
이는 KVM을 사용한 가상화에 유용합니다. 게스트 가상 머신이 시작되면 상위 **qemu-kvm** 프로세스의 메모리만 상속합니다. 게스트 가상 머신이 실행되면 게스트가 동일한 운영 체제 또는 애플리케이션을 실행할 때 게스트 가상 머신 운영 체제 이미지의 콘텐츠를 공유할 수 있습니다.



참고

페이지 중복 제거 기술(KSM 구현에서도 사용됨)은 여러 게스트에서 정보를 유출하는 데 사용할 수 있는 사이드 채널을 도입할 수 있습니다. 우려되는 경우 게스트당 KSM을 비활성화할 수 있습니다.

KSM은 향상된 메모리 속도 및 사용률을 제공합니다. KSM을 사용하면 일반적인 프로세스 데이터가 캐시 또는 주 메모리에 저장됩니다. 이를 통해 일부 애플리케이션 및 운영 체제의 성능을 향상시킬 수 있는 KVM 게스트의 캐시 누락이 줄어듭니다. 둘째, 메모리를 공유하면 더 높은 밀도와 리소스 사용률을 높일 수 있는 게스트의 전체 메모리 사용량이 줄어듭니다.



참고

Red Hat Enterprise Linux 6.5부터 KSM은 NUMA 인식입니다. 이를 통해 페이지를 병합하는 동안 NUMA 현지성을 고려할 수 있으므로 페이지와 관련된 성능 저하가 원격 노드로 이동되지 않습니다. KSM을 사용하는 경우 노드 간 메모리 병합을 방지하는 것이 좋습니다. KSM이 사용 중인 경우 NUMA 노드 간에 페이지를 병합하지 않도록 **/sys/kernel/mm/ksm/merge_across_nodes** 튜닝 가능 항목을 ***0*** 으로 변경합니다. 커널 메모리 회계 통계는 대량의 교차 노드 병합 후 결국 서로 모순될 수 있습니다. 따라서 KSM 데몬에서 대량의 메모리를 병합한 후 numad가 혼동될 수 있습니다. 시스템에 사용 가능한 메모리가 많은 경우 KSM 데몬을 끄고 비활성화하여 더 높은 성능을 얻을 수 있습니다. NUMA에 대한 자세한 내용은 *Red Hat Enterprise Linux 성능 튜닝 가이드*를 참조하십시오.

Red Hat Enterprise Linux는 KSM을 제어하기 위한 두 가지 다른 방법을 사용합니다.

- **ksm** 서비스는 KSM 커널 스레드를 시작하고 중지합니다.
- **ksmtuned** 서비스는 동일한 페이지 병합을 동적으로 관리하면서**ksm** 을 제어하고 조정합니다. **ksmtuned** 서비스는 **ksm** 을 시작하고 메모리 공유가 필요하지 않은 경우**ksm** 서비스를 중지합니다. **ksmtuned** 서비스에 새 게스트가 생성 또는 삭제될 때 실행할***retune*** 매개변수가 있어야 합니다.

이러한 두 서비스는 모두 표준 서비스 관리 툴로 제어됩니다.

KSM 서비스

ksm 서비스는 qemu-kvm 패키지에 포함되어 있습니다. KSM은 Red Hat Enterprise Linux 6에서 기본적으로 꺼져 있습니다. 그러나 Red Hat Enterprise Linux 6를 KVM 호스트 물리적 시스템으로 사용하는 경우 **ksm/ksmtuned** 서비스를 통해 켜질 수 있습니다.

ksm 서비스가 시작되지 않으면 KSM은 2000 페이지만 공유합니다. 이 기본값은 낮으며 제한된 메모리 저장 이점을 제공합니다.

ksm 서비스가 시작되면 KSM은 호스트 물리적 시스템 기본 메모리의 절반까지 공유합니다. KSM이 더 많은 메모리를 공유할 수 있도록 **ksm** 서비스를 시작합니다.

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

ksm 서비스는 기본 시작 시퀀스에 추가할 수 있습니다. chkconfig 명령을 사용하여**ksm** 서비스를 영구적으로 만듭니다.

```
# chkconfig ksm on
```

KSM Tuning Service

ksmtuned 서비스에는 옵션이 없습니다.**ksmtuned** 서비스 루프는**ksm** 을 조정합니다. 게스트 가상 머신을 만들거나 삭제할 때 **ksmtuned** 서비스는 libvirt에서 알림을 받습니다.

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

retune 매개변수를 사용하여 **ksmtuned** 서비스를 조정할 수 있습니다.***retune*** 매개변수는 **ksmtuned** 에 튜닝 기능을 수동으로 실행하도록 지시합니다.

파일에서 매개변수를 변경하기 전에 다음과 같이 명확히 해야 하는 몇 가지 용어가 있습니다.

- 제한 - 활성화 임계값(KB)입니다. 모든***qemu-kvm*** 프로세스의 합계에 추가된 증가 값이 총 시스템 메모리를 초과하는 경우 KSM 주기가 트리거됩니다. 이 매개변수는 ***KSM_THRES_COEF*** 에 정의된 백분율의 **kbytes**와 동일합니다.

/etc/ksmtuned.conf 파일은 **ksmtuned** 서비스의 구성 파일입니다. 아래의 파일 출력은 기본 **ksmtuned.conf** 파일입니다.

```
# Configuration file for ksmtuned.

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

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

# KSM_NPAGES_BOOST is added to the npages value, when free memory is less than thres.
# KSM_NPAGES_BOOST=300

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

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

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

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

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

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

KSM 변수 및 모니터링

KSM은 모니터링 데이터를 /sys/kernel/mm/ksm/ 디렉토리에 저장합니다. 이 디렉터리의 파일은 커널에 의해 업데이트되며 KSM 사용량 및 통계에 대한 정확한 기록입니다.

아래 목록의 변수는 아래에 언급된 /etc/ksmtuned.conf 파일의 설정 가능한 변수이기도 합니다.

/sys/kernel/mm/ksm/ 파일

full_scans

전체 검사 실행.

pages_shared

총 페이지 공유.

pages_sharing

현재 공유된 페이지입니다.

pages_to_scan

페이지를 스캔하지 않습니다.

pages_unshared

페이지가 더 이상 공유되지 않습니다.

pages_volatile

휘발성 페이지 수입니다.

run

KSM 프로세스가 실행 중인지 여부

sleep_millisecs

수면 밀리초입니다.

DEBUG=1 행이 /etc/ksmtuned.conf 파일에 추가되는 경우 KSM 튜닝 활동은 /var/log/ksmtuned 로그 파일에 저장됩니다. 로그 파일 위치는 *LOGFILE* 매개 변수를 사용하여 변경할 수 있습니다. 로그 파일 위치 변경은 권장되지 않으며 SELinux 설정을 특별한 설정이 필요할 수 있습니다.

KSM 비활성화

KSM에는 특정 환경 또는 물리적 머신 시스템에 비해 너무 커질 수 있는 성능 오버헤드가 있습니다.

KSM은 ksmtuned 및 ksm 서비스를 중지하여 비활성화할 수 있습니다. 서비스를 중지하면 KSM이 비활성화되지만 다시 시작한 후에는 유지되지 않습니다.

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

chkconfig 명령을 사용하여 KSM을 영구적으로 비활성화합니다. 서비스를 종료하려면 다음 명령을 실행합니다.

```
# chkconfig ksm off
# chkconfig ksmtuned off
```

중요

KSM을 사용하는 경우에도 스왑 크기가 커밋된 **RAM**에 충분한지 확인합니다. **KSM**은 동일하거나 유사한 게스트의 **RAM** 사용량을 줄입니다. 충분한 스왑 공간없이 게스트를 오버커밋할 수 있지만 게스트 가상 머신 메모리 사용을 사용하면 페이지가 공유되지 않을 수 있기 때문에 권장되지 않습니다.

8장. 고급 게스트 가상 시스템 관리

이 장에서는 게스트 가상 머신에서 시스템 리소스를 미세 조정 및 제어하는 고급 관리 툴에 대해 설명합니다.

8.1. 제어 그룹(CGROUPS)

Red Hat Enterprise Linux 6은 새로운 커널 기능을 제공합니다. 즉, 흔히 *cgroup* 이라고 하는 *제어 그룹*. CGroup을 사용하면 CPU 시간, 시스템 메모리, 네트워크 대역폭 또는 시스템에서 실행되는 사용자 정의 작업 그룹(processes) 간에 이러한 리소스의 조합을 할당할 수 있습니다. 구성 중인 cgroup을 모니터링하고 특정 리소스에 대한 cgroups 액세스를 거부하며 실행 중인 시스템에서 cgroup을 동적으로 재구성할 수 있습니다.

cgroup 기능은 libvirt 에서 완전히 지원됩니다. 기본적으로 libvirt 는 각 게스트를 다양한 컨트롤러(예: memory, cpu, blkio, 장치)에 대해 별도의 제어 그룹에 둡니다.

게스트가 시작되면 이미 cgroup에 있습니다. 필요할 수 있는 유일한 구성은 cgroup에 대한 정책 설정입니다. cgroups에 대한 자세한 내용은 ***Red Hat Enterprise Linux** 리소스 관리 가이드* 를 참조하십시오.

8.2. 대규모 페이지 지원

이 섹션에서는 대규모 페이지 지원에 대한 정보를 제공합니다.

소개

x86 CPU는 일반적으로 4kB 페이지의 메모리를 처리하지만 대규모 페이지라는 더 큰 페이지를 사용할 수 있습니다. ***TLB(Transaction Lookaside Buffer)** 에 대해 CPU 캐시 적중을 늘려 성능을 개선하기 위해 KVM 게스트를 대규모 페이지 메모리 지원과 함께 배포할 수 있습니다. 대규모 페이지는 특히 대규모 메모리 및 메모리 집약적 워크로드의 경우 성능을 크게 향상시킬 수 있습니다. Red Hat Enterprise Linux 6은 대규모 페이지 사용을 통해 페이지 크기를 늘려 많은 양의 메모리를 보다 효과적으로 관리할 수 있습니다.*

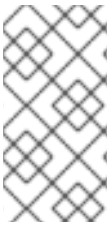
KVM 게스트에 대규모 페이지를 사용하면 페이지 테이블 및 TLB 누락에 더 적은 메모리가 사용되므로 특히 메모리 집약적인 상황에서는 성능이 크게 향상됩니다.

투명한 대규모 페이지

***THP(투명한 대규모 페이지)**는 애플리케이션에 필요한 TLB 항목을 줄이는 커널 기능입니다. 모든 여유 메모리를 캐시로 사용할 수도 있으므로 성능이 향상됩니다.*

투명한 대규모 페이지를 사용하려면 **qemu.conf** 파일의 특별한 구성이 필요하지 않습니다. 대규모 페이지는 **/sys/kernel/mm/redhat_transparent_hugepage/enabled** 가 항상 로 설정된 경우 기본적으로 사용됩니다.

투명한 대규모 페이지는 **hugetlbfs** 기능을 사용하지 않습니다. 그러나 **hugetlbfs**를 사용하지 않는 경우 **KVM**은 일반 **4kB** 페이지 크기 대신 투명한 대규모 페이지를 사용합니다.



참고

대규모 페이지로 메모리 성능 조정에 대한 지침은 **Red Hat Enterprise Linux 7 가상화 튜닝 및 최적화 가이드** 를 참조하십시오.

8.3. HYPER-V HYPERVISOR에서 RED HAT ENTERPRISE LINUX를 게스트 가상 머신으로 실행

Microsoft Windows Hyper-V 하이퍼바이저를 실행하는 **Microsoft Windows** 호스트 물리적 컴퓨터에서 **Red Hat Enterprise Linux** 게스트 가상 머신을 실행할 수 있습니다. 특히 **Red Hat Enterprise Linux** 게스트 가상 머신을 보다 쉽게 배포하고 관리할 수 있도록 다음과 같은 향상된 기능이 추가되었습니다.

- 업그레이드된 **VMBUS** 프로토콜 **- VMBUS** 프로토콜이 **Windows 8** 수준으로 업그레이드되었습니다. 이 작업의 일환으로 이제 게스트에서 사용 가능한 모든 가상 **CPU**에서 **VMBUS** 인터럽트를 처리할 수 있습니다. 또한 **Red Hat Enterprise Linux** 게스트 가상 머신과 **Windows** 호스트 물리적 머신 간의 신호 지정 프로토콜이 최적화되었습니다.

- 합성 프레임 버퍼 드라이버 **- Red Hat Enterprise Linux** 데스크탑 사용자를 위해 향상된 그래픽 성능과 뛰어난 해상도를 제공합니다.

- 실시간 가상 머신 백업 지원 **-** 라이브 **Red Hat Enterprise Linux** 게스트 가상 머신에 대한 중단없는 백업 지원

- 고정된 크기의 **Linux VHD**를 동적 확장 **-** 라이브 마운트 크기 **Red Hat Enterprise Linux VHDs**를 확장할 수 있습니다.

자세한 내용은 **Windows Server 2012 R2 Hyper-V에서 Linux 지원** 활성화를 참조하십시오.

참고

Hyper-V 하이퍼바이저는 마지막 파티션 뒤에 여유 공간이 있는 경우 **Red Hat Enterprise Linux** 게스트에서 **GPT** 파티션 디스크 축소를 지원하므로 사용자가 디스크의 사용되지 않은 마지막 부분을 삭제할 수 있습니다. 그러나 이 작업은 디스크의 보조 **GPT** 헤더를 자동으로 삭제합니다. 게스트에서 파티션 테이블을 검사할 때 오류 메시지가 생성될 수 있습니다(예: 파티션 테이블을 **parted**로 인쇄할 때). 이는 **Hyper-V**의 알려진 제한 사항입니다. 해결 방법으로 **gdisk** 및 **e** 명령의 전문가 메뉴를 사용하여 **GPT** 디스크를 축소한 후 보조 **GPT** 헤더를 수동으로 복원할 수 있습니다. 또한 **Hyper-V** 관리자의 **"expand"** 옵션을 사용하면 **GPT** 보조 헤더를 디스크가 아닌 다른 위치에 배치하지만 **parted** 로 이동할 수 있습니다. 이러한 명령에 대한 자세한 내용은 **gdisk** 및 **parted man** 페이지를 참조하십시오.

8.4. 게스트 가상 머신 메모리 할당

다음 절차에서는 게스트 가상 시스템에 메모리를 할당하는 방법을 보여줍니다. 이 할당 및 할당은 부팅 시에만 작동하며 메모리 값의 변경 사항은 다음 재부팅 시까지 적용되지 않습니다. 게스트당 할당할 수 있는 최대 메모리는 **4TiB**로, 이 메모리 할당은 호스트 물리적 시스템 리소스가 제공할 수 있는 호스트 실제 시스템 리소스보다 크지 않습니다.

유효한 메모리 단위는 다음과 같습니다.

- 바이트 의 **b** 또는 바이트
- **KB for kilobytes (10** 또는 **1,000** 바이트 블록**)**
- **k ibibytes**의 경우 **kibibytes 1024**바이트의 **K** 또는 **KiB**
- **MB for megabytes (10^6** 또는 **1,000,000** 바이트 블록**)**
- 메비바이트의 경우 **m** 또는 **MiB (2^{20}** 또는 **blocks 1,048,576** 바이트**)**
- **GB for gigabytes (10^9** 또는 **1,000,000,000** 바이트 블록**)**

- 기비바이트의 경우 **G** 또는 **GiB** (2^{30} 또는 **1,073,741,824** 바이트)
- **TB for terabytes** (10^{12} 또는 **1,000,000** 바이트 블록)
- **tebibytes**의 경우 **t or TiB** (2^{40} 또는 **1,099,511,627,776** 바이트)

모든 값은 **libvirt**에서 가장 가까운 **kibibyte**로 반올림되며 하이퍼바이저에서 지원하는 단위로 더 반올림될 수 있습니다. 일부 하이퍼바이저는 또한 **4000KiB**(또는 **4000 x** 2^{10} 또는 **4,096,000**바이트)와 같은 최소 용량을 적용합니다. 이 값의 단위는 선택적 특성 메모리 단위 에 따라 결정됩니다. 기본값은 **kibibytes(KiB)** 단위로 지정된 값이 **1024**바이트의 **2**개 또는 블록으로 곱한 단위입니다.

게스트 가상 머신이 선택적 특성 **dumpCore** 를 사용하여 게스트 가상 머신의 메모리를 생성된 코어 **dump(dumpCore='on')**에 포함해야 하는지 여부를 제어하는 데 사용할 수 있습니다**(dumpCore='off').** 기본 설정은 에 있으므로 매개변수가 **off** 로 설정되지 않은 경우 게스트 가상 머신 메모리가 **coredump** 파일에 포함됩니다.

currentMemory 속성은 게스트 가상 머신에 대한 실제 메모리 할당을 결정합니다. 이 값은 최대 할당보다 작을 수 있으며, 이를 통해 게스트 가상 머신 메모리를 즉시 늘릴 수 있습니다. 이 값을 생략하면 기본값은 **memory** 요소와 동일한 값입니다. **unit** 속성은 메모리와 동일하게 작동합니다.

이 섹션의 모든 경우 도메인 **XML**은 다음과 같이 변경해야 합니다.

```
<domain>

  <memory unit='KiB' dumpCore='off'>524288</memory>
  <!-- changes the memory unit to KiB and does not allow the guest virtual machine's memory to be
included in the generated coredump file -->
  <currentMemory unit='KiB'>524288</currentMemory>
  <!-- makes the current memory unit 524288 KiB -->
  ...
</domain>
```

8.5. 게스트 가상 머신 자동 시작

이 섹션에서는 호스트 물리적 시스템 부팅 단계에서 게스트 가상 머신을 자동으로 시작하는 방법에 대해 설명합니다.

이 예에서는 **virsh** 를 사용하여 게스트 가상 시스템 **TestServer** 를 설정하여 호스트 물리적 시스템이 부팅될 때 자동으로 시작됩니다.

```
# virsh autostart TestServer
Domain TestServer marked as autostarted
```

이제 게스트 가상 머신이 호스트 실제 시스템에서 자동으로 시작됩니다.

게스트 가상 머신이 자동으로 부팅되지 않도록 하려면 **--disable** 매개 변수를 사용합니다.

```
# virsh autostart --disable TestServer
Domain TestServer unmarked as autostarted
```

게스트 가상 시스템은 더 이상 호스트 물리적 시스템에서 자동으로 시작되지 않습니다.

8.6. 게스트 가상 머신에 대한 SMART 디스크 모니터링 비활성화

가상 디스크와 물리적 스토리지 장치는 호스트 물리적 시스템에서 관리되므로 **SMART** 디스크 모니터링을 안전하게 비활성화할 수 있습니다.

```
# service smartd stop
# chkconfig --del smartd
```

8.7. VNC 서버 구성

VNC 서버를 구성하려면 시스템에서 **Remote Desktop** 애플리케이션 을 사용합니다 . 또는 **vino-preferences** 명령을 실행할 수 있습니다.

다음 단계를 사용하여 전용 **VNC** 서버 세션을 설정합니다.

필요한 경우 **~/.vnc/xstartup** 파일을 만든 다음 편집하여 **vncserver** 가 시작될 때마다 **GNOME** 세션을 시작합니다. **vncserver** 스크립트를 처음 실행하는 경우 **VNC** 세션에 사용하려는 암호를 요청합니다. **vnc** 서버 파일에 대한 자세한 내용은 **Red Hat Enterprise Linux** 설치 가이드 를 참조하십시오.

8.8. 새로운 고유 MAC 주소 생성

게스트 가상 머신에 대한 새로운 고유 **MAC** 주소를 생성해야 하는 경우도 있습니다. 작성 시 새 **MAC** 주

*소를 생성하는 데 사용할 수 있는 명령행 툴이 없습니다. 아래에 제공된 스크립트는 게스트 가상 머신의 새 **MAC** 주소를 생성할 수 있습니다. 게스트 가상 머신의 스크립트를 **macgen.py** 로 저장합니다. 이제 해당 디렉토리에서 **./macgen.py** 를 사용하여 스크립트를 실행할 수 있으며 새 **MAC** 주소를 생성합니다. 샘플 출력은 다음과 같습니다.*

```
$ ./macgen.py
00:16:3e:20:b0:11
```

```
#!/usr/bin/python
# macgen.py script to generate a MAC address for guest virtual machines
#
import random
#
def randomMAC():
 mac = [ 0x00, 0x16, 0x3e,
  random.randint(0x00, 0x7f),
  random.randint(0x00, 0xff),
  random.randint(0x00, 0xff) ]
 return ':'.join(map(lambda x: "%02x" % x, mac))
#
print randomMAC()
```

*8.8.1. 게스트 가상 머신용 새 **MAC**을 생성하는 또 다른 방법*

*   **python-virtinst** *의 내장 모듈을 사용하여 게스트 가상 머신 구성 파일에 사용할 새 **MAC** 주소와 **UUID** 를 생성할 수도 있습니다.*

```
# echo  'import virtinst.util ; print\
 virtinst.util.uuidToString(virtinst.util.randomUUID())' | python
# echo  'import virtinst.util ; print virtinst.util.randomMAC()' | python
```

위의 스크립트를 아래에 표시된 대로 스크립트 파일로 구현할 수도 있습니다.

```
#!/usr/bin/env python
#  -*- mode: python; -*-
print ""
print "New UUID:"
import virtinst.util ; print virtinst.util.uuidToString(virtinst.util.randomUUID())
print "New MAC:"
import virtinst.util ; print virtinst.util.randomMAC()
print ""
```

8.9. 게스트 가상 머신 응답 시간 개선

게스트 가상 머신은 특정 워크로드 및 사용 패턴으로 응답하는 속도가 느릴 수 있습니다. 게스트 가상 머신이 느려지거나 응답하지 않는 상황을 예로 들 수 있습니다.

- *메모리 과다 할당.*
- *프로세서 사용량이 높은 오버 커밋된 메모리*
- *다른 (**qemu-kvm** 프로세스가 아님) 호스트 물리적 시스템에서 프로세스를 사용 중이거나 정지했습니다.*

***KVM** 게스트 가상 머신은 **Linux** 프로세스로 작동합니다. **Linux** 프로세스는 주 메모리(실제 **RAM**)에 영구적으로 보관되지 않으며 특히 사용되지 않는 경우 스왑 공간(가상 메모리)에 배치됩니다. 게스트 가상 머신이 장기간 비활성 상태이면 호스트 물리적 시스템 커널이 게스트 가상 머신을 스왑으로 이동할 수 있습니다. 스왑이 실제 메모리보다 느리면 게스트가 응답하지 않는 것처럼 보일 수 있습니다. 게스트가 주 메모리로 로드되면 변경됩니다. 스왑 메모리에서 메인 메모리로 게스트 가상 머신을 로드하는 프로세스는 스왑 및 구성 요소의 성능에 따라 게스트 가상 머신에 할당된 기가바이트의 **RAM**당 몇 초가 걸릴 수 있습니다.*

***KVM** 게스트 가상 머신 프로세스는 메모리 과다 할당 또는 전체 메모리 사용량에 관계없이 스왑으로 이동할 수 있습니다.*

안전하지 않은 과다 할당 수준을 사용하거나 스왑으로 과다 할당하면 게스트 가상 머신 프로세스 또는 기타 중요한 프로세스를 사용하지 않는 것이 좋습니다. 메모리를 과다 할당할 때 호스트 물리적 시스템에 충분한 스왑 공간이 있는지 항상 확인하십시오.

***KVM**을 통한 오버 커밋에 대한 자세한 내용은 **6장. KVM으로 과다 할당** 에서 참조하십시오.*

주의

가상 메모리를 사용하면 Linux 시스템에서 시스템에 실제 RAM보다 많은 메모리를 사용할 수 있습니다. 사용되지 않는 프로세스는 활성 프로세스가 메모리를 사용하여 메모리를 사용할 수 있도록 하여 메모리 사용률을 개선할 수 있도록 합니다. 스왑을 비활성화하면 모든 프로세스가 물리적 RAM에 저장되므로 메모리 사용률이 줄어듭니다.

스왑이 해제되면 게스트 가상 머신을 과다 할당하지 마십시오. 스왑없이 게스트 가상 머신을 과다 할당하면 게스트 가상 머신 또는 호스트 물리적 머신 시스템이 충돌할 수 있습니다.

8.10. LIBVIRT를 사용한 가상 머신 타이머 관리

게스트 가상 머신을 정확하게 유지하는 것은 가상화 플랫폼의 핵심 과제입니다. 다른 하이퍼바이저는 다양한 방법으로 유지하는 시간을 처리하려고 합니다. libvirt는 도메인 XML의 <clock> 및 <timer> 요소를 사용하여 시간 관리를 위해 하이퍼바이저 독립 구성 설정을 제공합니다. domain XML은 virsh edit 명령을 사용하여 편집할 수 있습니다. 자세한 내용은 14.6절. "게스트 가상 머신의 구성 파일 편집" 를 참조하십시오.

<clock> 요소는 게스트 가상 시스템 시계가 호스트 물리적 시스템 클럭과 어떻게 동기화되는지 결정하는 데 사용됩니다. clock 요소에는 다음과 같은 속성이 있습니다.

- *오프셋 은 게스트 가상 머신 클럭이 호스트 실제 머신 클럭에서 오프셋되는 방식을 결정합니다. offset 속성에는 다음과 같은 가능한 값이 있습니다.*

표 8.1. 특성 값 오프셋

값	설명
utc	부팅 시 게스트 가상 머신 클럭이 UTC에 동기화됩니다.
현지 시간	게스트 가상 머신 클럭은 부팅될 때 호스트 물리적 시스템의 구성된 시간대(있는 경우)에 동기화됩니다.
timezone	게스트 가상 머신 클럭은 ***timezone*** 특성으로 지정된 지정된 시간대에 동기화됩니다.

값	설명
변수	게스트 가상 머신 클럭은 UTC의 임의의 오프셋에 동기화됩니다. UTC를 기준으로 하는 delta는 ***조정*** 특성을 사용하여 초 단위로 지정됩니다. 게스트 가상 머신은 시간 경과에 따라 RTC(Real Time Clock)를 자유롭게 조정할 수 있으며 다음 재부팅 시에도 적용됩니다. 이는 ***utc*** 모드와 달리, 각 재부팅 시 RTC 조정이 손실됩니다.

참고

값 utc 는 기본적으로 가상 머신에서 클럭 오프셋으로 설정됩니다. 그러나 게스트 가상 머신 클럭이 localtime 값으로 실행되는 경우 게스트 가상 머신 클럭을 호스트 물리적 시스템 클럭과 동기화하려면 클럭 오프셋을 다른 값으로 변경해야 합니다.

- ***timezone** 속성은 게스트 가상 시스템 클럭에 사용되는 시간대를 결정합니다.*

- *조정 속성은 게스트 가상 머신 클럭 동기화를 위해 **delta**를 제공합니다. **UTC**를 기준으로 하는 시간(초)입니다.*

예 8.1. 항상 UTC와 동기화

```
<clock offset="utc" />
```

예 8.2. 항상 호스트 실제 시스템 시간대와 동기화

```
<clock offset="localtime" />
```

예 8.3. 임의의 시간대에 동기화

```
<clock offset="timezone" timezone="Europe/Paris" />
```

예 8.4. UTC + 임의의 오프셋과 동기화

```
<clock offset="variable" adjustment="123456" />
```

8.10.1. 클럭을 위한 타이머 하위 요소

시계 요소에는 0개 이상의 타이머 요소가 자식으로 있을 수 있습니다. timer 요소는 게스트 가상 머신 클럭 동기화에 사용되는 시간 소스를 지정합니다. timer 요소에는 다음과 같은 속성이 있습니다. 이름만 필요하며 다른 모든 속성은 선택 사항입니다.

name 속성은 사용할 시간 소스의 유형을 지시하며 다음 중 하나일 수 있습니다.

표 8.2. 이름 특성 값

값	설명
pit	Custom Interval timer - 주기적인 인터럽트가 있는 타이머입니다.
rtc	실시간 시계 - 주기적인 인터럽트를 사용하여 지속적으로 실행되는 타이머입니다.
tsc	Time Stamp counter - 재설정 이후의 틱 수를 계산하고, 인터럽트가 없습니다.
kvmclock	KVM 클럭 - KVM 게스트 가상 머신에 권장되는 클럭 소스입니다. KVM pvclock 또는 kvm-clock을 사용하면 게스트 가상 머신이 호스트 물리적 시스템의 월 시계 시간을 읽을 수 있습니다.

8.10.2. track

track 속성은 타이머를 통해 추적되는 항목을 지정합니다. 이름 값 rtc 에만 유효합니다.

표 8.3. 특성 값 추적

값	설명
boot	이전 *호스트 물리적 머신* 옵션에 해당하며 지원되지 않는 추적 옵션입니다.
게스트	RTC는 항상 게스트 가상 머신 시간을 추적합니다.
wall	RTC는 항상 호스트 시간을 추적합니다.

8.10.3. tickpolicy

tickpolicy 특성은 게스트 가상 머신에 눈금을 전달하는 데 사용되는 정책을 할당합니다. 다음 값이 허용됩니다.

표 8.4. tickpolicy 특성 값

값	설명
delay	계속 정상적으로 전달하십시오(추적이 지연됨).
catchup	더 높은 비율로 가져와 차지할 수 있습니다.
병합	하나의 진드기는 하나의 진드로 병합되었습니다.
삭제	누락된 모든 틱은 모두 삭제됩니다.

8.10.4. 빈도, 모드 및 현재

*빈도 속성은 고정 빈도를 설정하는 데 사용되며 **Hz**로 측정됩니다. 이 속성은 **name** 요소에 **tsc** 값이 있는 경우에만 관련이 있습니다. 다른 모든 타이머는 고정 빈도(**pit,rtc**) 에서 작동합니다.*

***mode** 는 시간 소스가 게스트 가상 머신에 노출되는 방식을 결정합니다. 이 속성은 이름 값 **tsc** 와만 관련이 있습니다. 다른 모든 타이머는 항상 에뮬레이션됩니다. 명령은 **< timer name='tsc' frequency='NNN' mode='auto|native|emulate|smpsafe'/ >** 와 같습니다. 모드 정의는 표에 제공됩니다.*

표 8.5. 모드 특성 값

값	설명
auto	TSC가 불안정한 경우 네이티브 TSC 액세스를 허용하십시오.
실제 하드웨어에 적용	항상 네이티브 TSC 액세스를 허용합니다.
에뮬레이션	항상 에뮬레이트해야 합니다.
smpsafe	항상 에뮬레이션 TSC 및 상호 잠금 가속화

***present** 는 게스트 가상 머신에 표시되는 기본 타이머 세트를 재정의하는 데 사용됩니다.*

표 8.6. 현재 특성 값

값	설명
제공됨	게스트 가상 머신에 표시되는 이 타이머를 강제 적용합니다.

값	설명
제공되지 않음	게스트 가상 머신에 이 타이머를 강제로 표시하지 않도록 합니다.

8.10.5. Clock Synchronization 사용 예

예 8.5. RTC 및 PIT 타이머를 사용하여 로컬 시간에 클럭 동기화

이 예에서 시계는 **RTC** 및 **PIT** 타이머와 현지 시간에 동기화됩니다.

```
<clock offset="localtime">
 <timer name="rtc" tickpolicy="catchup" track="guest virtual machine" />
 <timer name="pit" tickpolicy="delay" />

</clock>
```

참고

PIT 클럭 소스는 다음과 같은 조건에서 **64**비트 호스트(**PIT**를 사용할 수 없음)에서 실행되는 **32**비트 게스트와 함께 사용할 수 있습니다.

- 게스트 가상 머신에는 **CPU**가 하나만 있을 수 있습니다.
- **APIC** 타이머를 비활성화해야 합니다 (**noapictimer"** 명령줄 옵션 사용)
- **NoHZ** 모드는 게스트에서 비활성화해야 합니다 (**nohz=off"** 명령줄 옵션 사용)
- 높은 해상도 타이머 모드를 게스트에서 비활성화해야 합니다 (**'highres=off"** 명령줄 옵션 사용)
- **PIT** 클럭 소스는 높은 해상도의 타이머 모드 또는 **NoHz** 모드와 호환되지 않습니다.

8.11. PMU를 사용하여 게스트 가상 머신 성능 모니터링

Red Hat Enterprise Linux 6.4에서 vPMU(가상 PMU)는 기술 프리뷰로 도입되었습니다. vPMU는 Intel PMU(Performance Monitoring Unit)를 기반으로 하며 Intel 시스템에서만 사용할 수 있습니다. PMU는 게스트 가상 머신이 작동하는 방식을 나타내는 통계 추적을 허용합니다.

성능 모니터링을 사용하면 개발자가 프로파일링을 위해 성능 툴을 사용하는 동안 CPU의 PMU 카운터를 사용할 수 있습니다. 가상 머신 사용자는 가상 머신 사용자가 게스트 가상 머신에서 가능한 성능 문제의 원인을 파악할 수 있으므로 KVM 게스트 가상 머신을 프로파일링하는 기능이 향상됩니다.

기능을 활성화하려면 -cpu 호스트 플래그를 설정해야 합니다.

이 기능은 Red Hat Enterprise Linux 6를 실행하는 게스트 가상 시스템에서만 지원되며 기본적으로 비활성화되어 있습니다. 이 기능은 Linux perf 툴에서만 작동합니다. 명령을 사용하여 perf 패키지가 설치되었는지 확인합니다.

```
# yum install perf.
```

perf 명령에 대한 자세한 내용은 perf 의 man 페이지를 참조하십시오.

8.12. 게스트 가상 머신 전원 관리

Libvirt용 도메인 XML에서 다음 매개 변수를 변경하여 게스트 가상 머신의 운영 체제에 BIOS 알림을 활성화 또는 비활성화할 수 있습니다.

```
...
  <pm>
    <suspend-to-disk enabled='no'/>
    <suspend-to-mem enabled='yes'/>
  </pm>
  ...
```

pm 은 S3(suspend-to-disk) 및 S4(suspend-to-mem) ACPI sleep 상태에 대한 BIOS 지원을 비활성화('yes') 또는 비활성화합니다. 아무것도 지정하지 않으면 하이퍼바이저는 기본값으로 유지됩니다.

9장. 게스트 가상 머신 장치 구성

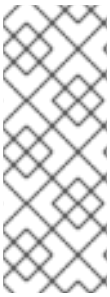
Red Hat Enterprise Linux 6는 게스트 가상 머신을 위한 세 가지 장치 클래스를 지원합니다.

- 에뮬레이션된 장치는 실제 하드웨어를 모방하는 순수한 가상 장치이므로 수정되지 않은 게스트 운영 체제가 표준 인 박스 드라이버를 사용하여 작업할 수 있습니다. **Red Hat Enterprise Linux 6**는 최대 **216 virtio** 장치를 지원합니다.

- **virtio** 장치는 가상 머신에서 최적으로 작동하도록 설계된 순전히 가상 장치입니다. **virtio** 장치는 에뮬레이션된 장치와 유사하지만 **Linux** 이외의 가상 시스템에는 기본적으로 필요한 드라이버가 포함되어 있지 않습니다. **Virtual Machine Manager(virt-manager)** 및 **RHV-H(Red Hat Virtualization Hypervisor)**와 같은 가상화 관리 소프트웨어는 지원되는 비 **Linux** 게스트 운영 체제에 이러한 드라이버를 자동으로 설치합니다. **Red Hat Enterprise Linux 6**는 최대700개의**scsi** 디스크를 지원합니다.

- 할당된 장치는 가상 머신에 노출되는 물리적 장치입니다. 이 방법은 **'passthrough'**라고도 합니다. 장치 할당을 통해 가상 머신은 다양한 작업을 위해 **PCI** 장치에 독점적으로 액세스할 수 있으며 **PCI** 장치는 게스트 운영 체제에 물리적으로 연결된 것처럼 표시 및 작동할 수 있습니다. **Red Hat Enterprise Linux 6**는 가상 머신당 최대 **32**개의 할당된 장치를 지원합니다.

장치 할당은 일부 그래픽 장치를 포함하여 **PCIe** 장치에서 지원됩니다. **Red Hat Enterprise Linux 6**의 장치 할당으로 **NVIDIA K-series Quadro, GRID** 및 **Tesla** 그래픽 카드 **GPU** 기능이 지원됩니다. 병렬 **PCI** 장치는 할당된 장치로 지원할 수 있지만 보안 및 시스템 구성 충돌로 인해 심각한 제한 사항이 있습니다.



참고

가상 머신에 연결할 수 있는 장치 수는 여러 요인 에 따라 다릅니다. 한 가지 요소는 **QEMU** 프로세스에서 여는 파일 수입니다(**/etc/libvirt/qemu.conf**에서 재정의할 수 있는 **/etc/security/limits.conf** 에서 구성됨). 다른 제한 요소에는 가상 버스에서 사용할 수 있는 슬롯 수와 **sysctl**이 설정한 오픈 파일에 대한 시스템 전체 제한이 포함됩니다.

특정 장치 및 제한 사항에 대한 자세한 내용은 **20.16**절. “장치” 을 참조하십시오.

Red Hat Enterprise Linux 6는 가상 머신에 단일 기능 슬롯으로 노출되는 장치의 **PCI** 핫 플러그를 지원합니다. 단일 기능 호스트 장치 및 다중 기능 호스트 장치의 개별 기능과 이를 사용하도록 구성할 수 있습니다. 가상 머신에 다중 기능 **PCI** 슬롯으로 장치를 노출하는 구성은 핫플러그 애플리케이션에만 권장됩니다.

참고

인터럽트 재조정을 위한 플랫폼 지원은 호스트에서 할당된 장치가 있는 게스트를 완전히 격리하는 데 필요합니다. 이러한 지원이 없으면 호스트가 악성 게스트의 삽입 공격을 방해하기 위해 취약해질 수 있습니다. 게스트가 신뢰할 수 있는 환경에서 관리자는 **vfio_iommu_type1** 모듈에 **allow_unsafe_interrupts** 옵션을 사용하여 **PCI** 장치 할당을 허용하도록 선택할 수 있습니다. 이 작업은 다음을 포함하는 **/etc/modprobe.d** 에 **.conf** 파일(예: **local.conf**)을 추가하여 영구적으로 수행할 수 있습니다.

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

또는 **sysfs** 항목을 사용하여 동일한 작업을 수행합니다.

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

9.1. PCI 장치

PCI 장치 할당은 **Intel VT-d** 또는 **AMD IOMMU**를 지원하는 하드웨어 플랫폼에서만 사용할 수 있습니다. **PCI** 장치 할당이 작동하려면 **BIOS**에서 이러한 **Intel VT-d** 또는 **AMD IOMMU** 사양을 활성화해야 합니다.

절차 **9.1. PCI** 장치 할당을 위한 **Intel** 시스템 준비

1. **Intel VT-d** 사양 활성화

   Intel VT-d 사양에서는 가상 머신에 물리적 장치를 직접 할당하는 하드웨어 지원을 제공합니다. 이러한 사양은 **Red Hat Enterprise Linux**에서 **PCI** 장치 할당을 사용하는 데 필요합니다.

   BIOS에서 **Intel VT-d** 사양을 활성화해야 합니다. 일부 시스템 제조업체에서는 기본적으로 이러한 사양을 사용하지 않도록 설정되어 있습니다. 이러한 사양을 참조하는 데 사용되는 조건은 제조업체마다 다를 수 있습니다. 적절한 약관에 대해서는 시스템 제조업체 설명서를 참조하십시오.

2. 커널에서 **Intel VT-d** 활성화

   /etc/sysconfig/grub 파일에 있는 **GRUB_CMDLINX_LINUX** 행 끝에 **intel_iommu=on** 매개변수를 추가하여 커널에서 **Intel VT-d**를 활성화합니다.

   아래 예제는 **Intel VT-d**가 활성화된 수정된 **grub** 파일입니다.

   ```
   GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
   ```

```
vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ] && /usr/sbin/
rhcrashkernel-param || :) rhgb quiet intel_iommu=on"
```

3. *구성 파일 다시 생성*

   *다음을 실행하여 **/etc/grub2.cfg**를 다시 생성합니다.*

```
grub2-mkconfig -o /etc/grub2.cfg
```

   ***UEFI** 기반 호스트를 사용하는 경우 대상 파일은 **/etc/grub2-efi.cfg** 여야 합니다.*

4. *사용할 준비가*

   *시스템을 재부팅하여 변경 사항을 활성화합니다. 이제 시스템이 **PCI** 장치 할당을 수행할 수 있습니다.*

*절차 **9.2. PCI** 장치 할당을 위한 **AMD** 시스템 준비*

1. ***AMD IOMMU** 사양 활성화*

   ***Red Hat Enterprise Linux**에서 **PCI** 장치 할당을 사용하려면 **AMD IOMMU** 사양이 필요합니다. **BIOS**에서 이러한 사양을 활성화해야 합니다. 일부 시스템 제조업체에서는 기본적으로 이러한 사양을 사용하지 않도록 설정되어 있습니다.*

2. ***IOMMU** 커널 지원 활성화*

   ***/etc/sysconfig/grub** 에 있는 **GRUB_CMDLINX_LINUX** 행의 끝에 **amd_iommu=on** 을 추가하여 부팅 시 **AMD IOMMU** 사양을 활성화합니다.*

3. *구성 파일 다시 생성*

   *다음을 실행하여 **/etc/grub2.cfg**를 다시 생성합니다.*

```
grub2-mkconfig -o /etc/grub2.cfg
```

   ***UEFI** 기반 호스트를 사용하는 경우 대상 파일은 **/etc/grub2-efi.cfg** 여야 합니다.*

4. *사용할 준비가*

   *시스템을 재부팅하여 변경 사항을 활성화합니다. 이제 시스템이 **PCI** 장치 할당을 수행할 수 있습니다.*

*9.1.1. virsh**를 사용하여 PCI** 장치 할당

*이러한 단계는 **PCI** 장치를 **KVM** 하이퍼바이저의 가상 머신에 할당하는 방법을 다룹니다.*

*이 예에서는 **PCI** 식별자 코드 **pci_0000_01_00_0** 과 함께 **PCIe** 네트워크 컨트롤러를 사용하고 **guest1-rhel6-64** 라는 완전히 가상화된 게스트 시스템을 사용합니다.*

*절차 **9.3. virsh**를 사용하여 게스트 가상 머신에 **PCI** 장치 할당*

1. *장치 확인*

   *먼저 가상 머신에 장치 할당을 위해 지정된 **PCI** 장치를 식별합니다. **lspci** 명령을 사용하여 사용 가능한 **PCI** 장치를 나열합니다. **grep** 을 사용하여 **lspci** 의 출력을 구체화할 수 있습니다.*

   이 예에서는 다음 출력에서 강조 표시된 이더넷 컨트롤러를 사용합니다.

```
# 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)
```

   *이 이더넷 컨트롤러는 짧은 식별자 **00:19.0** 과 함께 표시됩니다. 이 **PCI** 장치를 가상 시스템에 할당하려면 **virsh** 에서 사용하는 전체 식별자를 찾아야 합니다.*

   *이를 위해 **virsh nodedev-list** 명령을 사용하여 호스트 시스템에 연결된 특정 유형**(pci)**의 모든 장치를 나열합니다. 그런 다음 사용하려는 장치의 짧은 식별자에 매핑되는 문자열의 출력을 확인합니다.*

   *이 예제에서는 짧은 식별자가 **00:19.0** 인 이더넷 컨트롤러에 매핑되는 문자열을 강조 표시합니다. 이 예제에서는 **:** 및 **.** 문자는 전체 식별자에서 밑줄로 교체됩니다.*

```
# 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
```

*사용하려는 장치에 매핑되는 **PCI** 장치 번호를 기록합니다. 이는 다른 단계에서 필요합니다.*

2. *장치 정보 검토*

***domain, bus** 및 **function**에 대한 정보는 **virsh nodedev-dumpxml** 명령 출력에서 확인할 수 있습니다.*

```
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>
```

참고

IOMMU 그룹은 **IOMMU**의 관점에서 장치의 가시성 및 분리에 따라 결정됩니다. 각 **IOMMU** 그룹은 하나 이상의 장치를 포함할 수 있습니다. 여러 장치가 있는 경우 **IOMMU** 그룹 내의 모든 엔드 포인트를 게스트에 할당하려면 그룹 내의 모든 장치에 대해 클레임해야 합니다. 이 작업은 게스트에 추가 엔드포인트를 할당하거나 **virsh nodedev-detach** 를 사용하여 호스트 드라이버에서 분리하여 수행할 수 있습니다. 단일 그룹 내에 포함된 장치는 여러 게스트 간에 분할하거나 호스트와 게스트 간에 분할할 수 없습니다. **PCIe** 루트 포트, 스위치 포트 및 브리지와 같은 비 엔드 포인트 장치는 호스트 드라이버에서 분리되지 않아야 하며 끝점 할당을 방해하지 않습니다.

IOMMU 그룹 내의 장치는 **virsh nodedev-dumpxml** 출력의 **iommuGroup** 섹션을 사용하여 결정할 수 있습니다. 그룹의 각 멤버는 별도의 **"address"** 필드를 통해 제공됩니다. 이 정보는 다음을 사용하여 **sysfs**에서 찾을 수도 있습니다.

```
$ ls /sys/bus/pci/devices/0000:01:00.0/iommu_group/devices/
```

출력 예는 다음과 같습니다.

```
0000:01:00.0  0000:01:00.1
```

그룹에 **0000.01.00.0**만 할당하려면 게스트를 시작하기 전에 사용하지 않은 끝점을 호스트에서 분리해야 합니다.

```
$ virsh nodedev-detach pci_0000_01_00_1
```

3. 필수 구성 세부 정보 확인

구성 파일에 필요한 값에 대해서는 **virsh nodedev-dumpxml pci_0000_00_19_0** 명령의 출력을 참조하십시오.

예제 장치의 값은 **bus = 0, slot = 25**이고 **function = 0**입니다. **10**진수 구성은 다음 세 가지 값을 사용합니다.

```
bus='0'
slot='25'
function='0'
```

4. 구성 세부 정보 추가

virsh edit 를 실행하여 가상 시스템 이름을 지정하고 **< source >** 섹션에 장치 항목을 추가하

여 **PCI** 장치를 게스트 가상 머신에 할당합니다.

```
# virsh edit guest1-rhel6-64
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0' bus='0' slot='25' function='0'/>
  </source>
</hostdev>
```

또는 **virsh attach-device** 를 실행하여 가상 시스템 이름과 게스트의 **XML** 파일을 지정합니다.

```
virsh attach-device guest1-rhel6-64 file.xml
```

5. 가상 머신 시작

```
# virsh start guest1-rhel6-64
```

이제 **PCI** 장치를 가상 시스템에 성공적으로 할당하고 게스트 운영 체제에서 액세스할 수 있습니다.

9.1.2. virt-manager를 사용하여 PCI 장치 할당

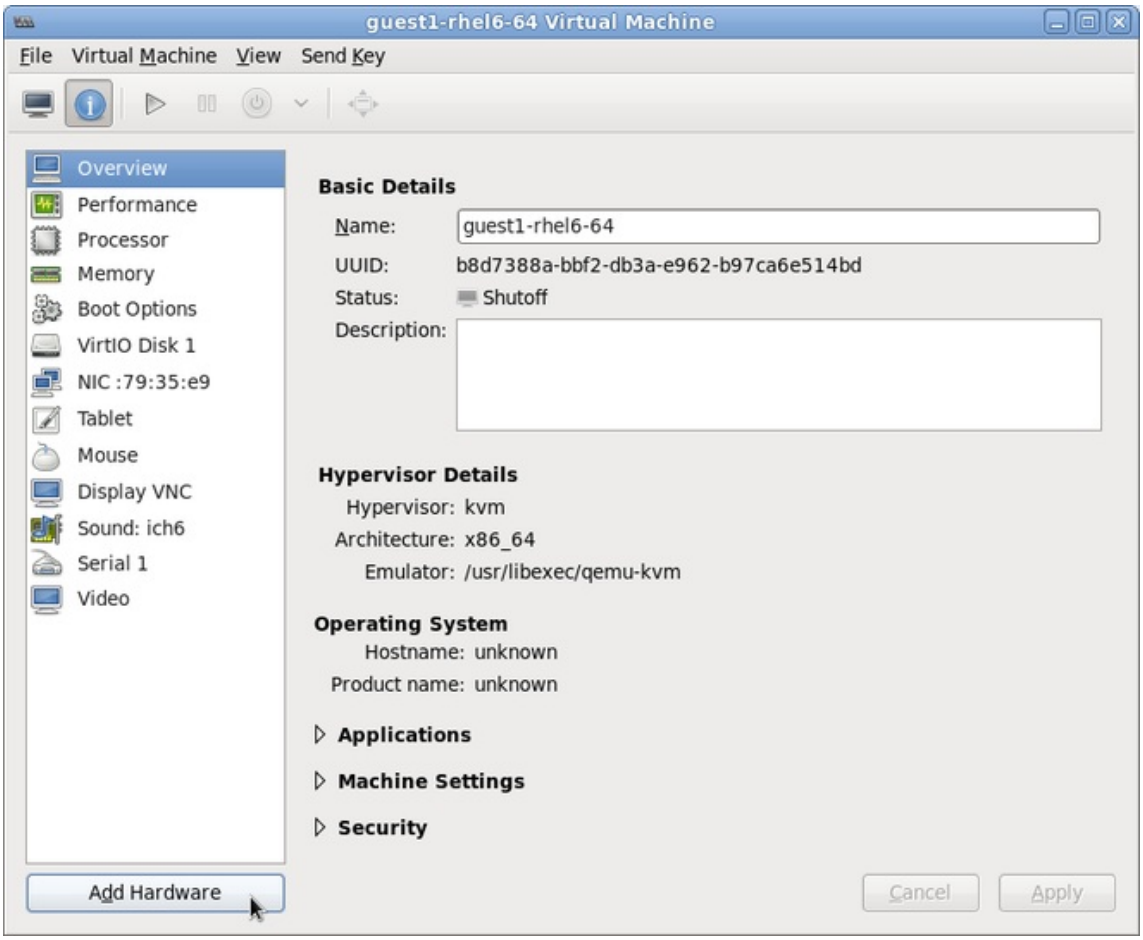
그래픽 **virt-manager** 도구를 사용하여 **PCI** 장치를 게스트 가상 머신에 추가할 수 있습니다. 다음 절차에서는 게스트 가상 머신에 기가비트 이더넷 컨트롤러를 추가합니다.

절차 **9.4. virt-manager**를 사용하여 게스트 가상 머신에 **PCI** 장치 할당

1. 하드웨어 설정 열기

   게스트 가상 머신을 열고 하드웨어 추가 버튼 을 클릭하여 가상 머신에 새 장치를 추가합니다.

*그림 **9.1.** 가상 머신 하드웨어 정보 창*

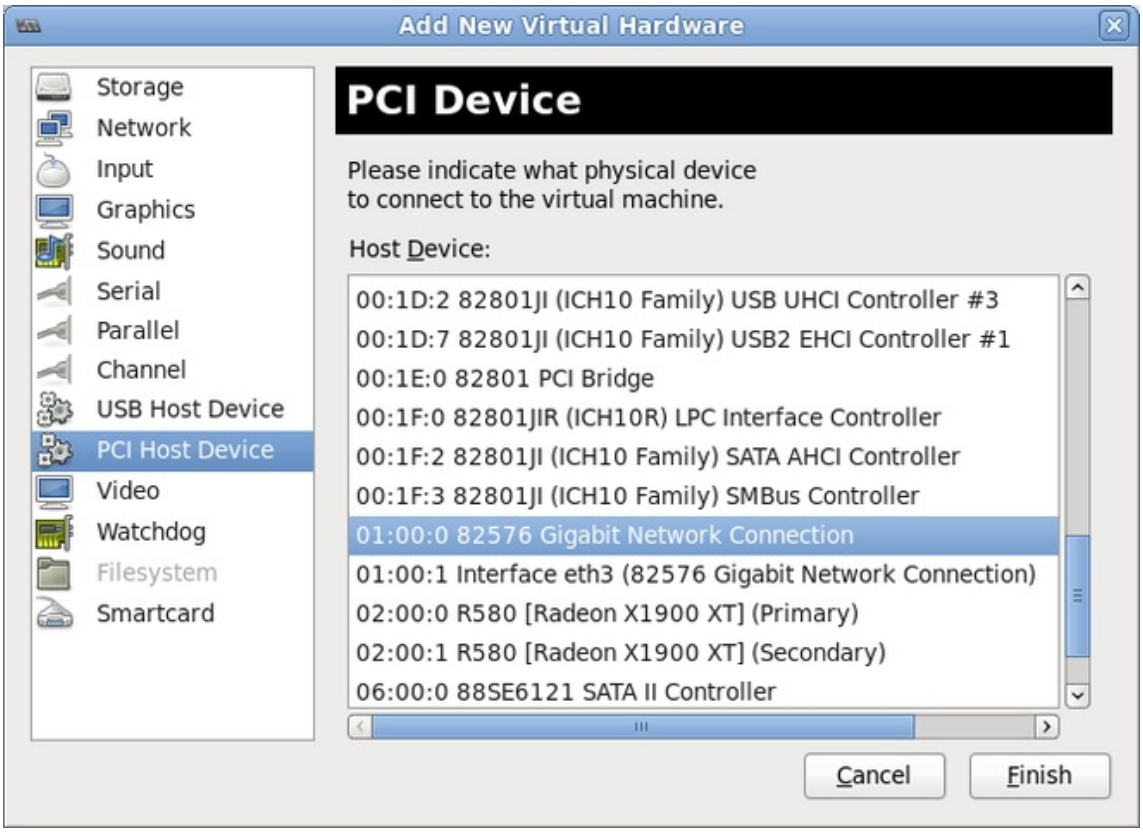


2. ***PCI** 장치 선택*

   *왼쪽의 하드웨어 목록에서 **PCI Host Device** 를 선택합니다.*

   *사용되지 않는 **PCI** 장치를 선택합니다. 다른 게스트에서 사용 중인 **PCI** 장치를 선택하면 오류가 발생할 수 있습니다. 이 예에서는 스페어 **82576** 네트워크 장치가 사용됩니다. 완료 를 클릭하여 설정을 완료합니다.*

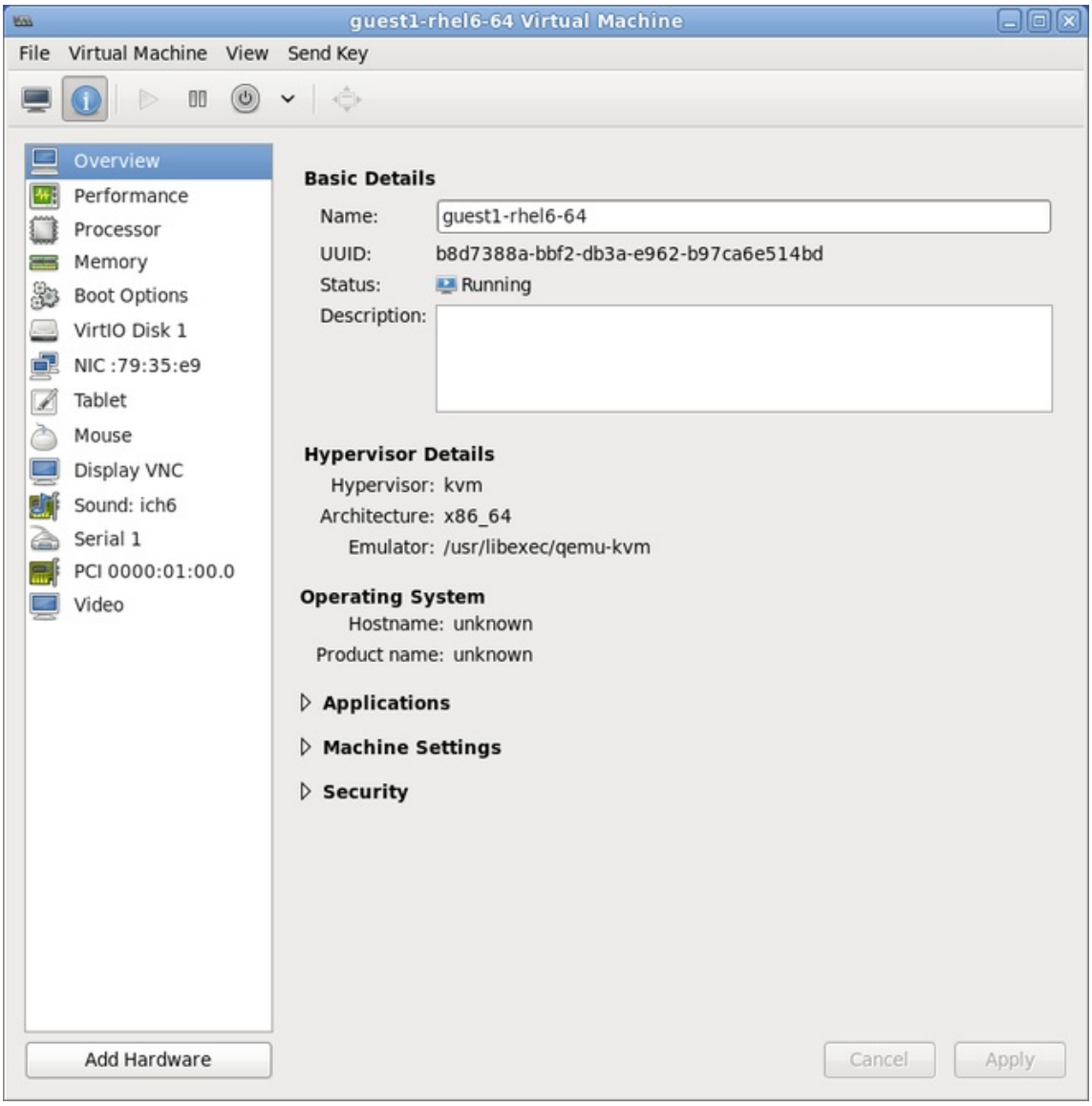
그림 9.2. 가상 하드웨어 추가 마법사

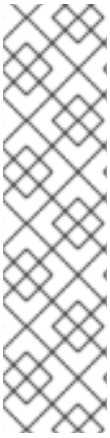


3. *새 장치 추가*

   *설정이 완료되었으며 게스트 가상 머신이 이제 **PCI** 장치에 직접 액세스할 수 있습니다.*

그림 9.3. 가상 머신 하드웨어 정보 창





참고

장치 할당이 실패하면 동일한 IOMMU 그룹에 여전히 호스트에 연결된 다른 엔드포인트가 있을 수 있습니다. virt-manager를 사용하여 그룹 정보를 검색할 수는 없지만 virsh 명령을 사용하여 IOMMU 그룹의 경계와 필요한 sequester 장치를 분석할 수 있습니다.

IOMMU 그룹 및 virsh를 사용하여 엔드포인트 장치를 분리하는 방법에 대한 자세한 내용은 참고 의 9.1.1절. “virsh를 사용하여 PCI 장치 할당” 를 참조하십시오.

9.1.3. virt-install을 사용한 PCI 장치 할당

virt-install 을 사용하여 PCI 장치를 할당하려면 --host-device 매개 변수를 사용합니다.

절차 9.5. virt-install을 사용하여 가상 머신에 PCI 장치 할당

1. *장치 확인*

   *게스트 가상 머신에 장치 할당을 위해 지정된 **PCI** 장치를 식별합니다.*

```
# 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** 명령은 시스템에 연결된 모든 장치를 나열하고 각 **PCI** 장치를 문자열로 식별합니다. 출력을 **PCI** 장치로만 제한하려면 다음 명령을 실행합니다.*

```
# 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** 장치 번호를 기록합니다. 다른 단계에는 숫자가 필요합니다.*

도메인, 버스 및 기능에 대한 정보는 **virsh nodedev-dumpxml** 명령 출력에서 확인할 수 있습니다.

```
# virsh nodedev-dumpxml pci_0000_01_00_0
<device>
  <name>pci_0000_01_00_0</name>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>1</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='7'>
      <address domain='0x0000' bus='0x00' slot='0x19' function='0x0'/>
    </iommuGroup>
  </capability>
</device>
```

참고

IOMMU 그룹에 여러 끝점이 있고 모든 엔드포인트가 게스트에 할당되지 않은 경우 게스트를 시작하기 전에 다음 명령을 실행하여 호스트에서 다른 끝점을 수동으로 분리해야 합니다.

```
$ virsh nodedev-detach pci_0000_00_19_1
```

IOMMU 그룹에 대한 자세한 내용은 참고 의 **9.1.1절. “virsh를 사용하여 PCI 장치 할당”** 를 참조하십시오.

2. 장치 추가

virsh nodedev 명령의 **PCI** 식별자 출력을 **--host-device** 매개 변수의 값으로 사용합니다.

```
virt-install \
--name=guest1-rhel6-64 \
--disk path=/var/lib/libvirt/images/guest1-rhel6-64.img,size=8 \
--nonsparse --graphics spice \
--vcpus=2 --ram=2048 \
--location=http://example1.com/installation_tree/RHEL6.0-Server-x86_64/os \
```

```
--nonetworks \
--os-type=linux \
--os-variant=rhel6
--host-device=pci_0000_01_00_0
```

3. *설치 완료*

   *게스트 설치를 완료합니다. **PCI** 장치를 게스트에 연결해야 합니다.*

9.1.4. 할당된 PCI 장치 분리

*호스트 **PCI** 장치가 게스트 시스템에 할당된 경우 호스트는 더 이상 장치를 사용할 수 없습니다. 이 섹션에서는 호스트 사용을 위해 사용할 수 있도록 **virsh** 또는 **virt-manager** 를 사용하여 게스트에서 장치를 분리하는 방법을 알아봅니다.*

*절차 **9.6. virsh**를 사용하여 게스트에서 **PCI** 장치 분리*

1. *장치 분리*

   *다음 명령을 사용하여 게스트의 **XML** 파일에서 **PCI** 장치를 제거하여 게스트에서 **PCI** 장치를 분리합니다.*

```
# virsh detach-device name_of_guest file.xml
```

2. *장치를 호스트에 다시 연결합니다(선택 사항)*

   장치가 관리 모드에 있는 경우 이 단계를 건너뜁니다. 장치가 자동으로 호스트로 반환됩니다.

   *장치가 관리 모드를 사용하지 않는 경우 다음 명령을 사용하여 **PCI** 장치를 호스트 머신에 다시 연결합니다.*

```
# virsh nodedev-reattach device
```

   *예를 들어 **pci_0000_01_00_0** 장치를 호스트에 다시 연결하려면 다음을 수행합니다.*

```
virsh nodedev-reattach pci_0000_01_00_0
```

   이제 호스트용으로 장치를 사용할 수 있습니다.

*절차 **9.7. virt-manager**를 사용하여 게스트에서 **PCI** 장치 분리*

1. *가상 하드웨어 세부 정보 화면을 엽니다.*

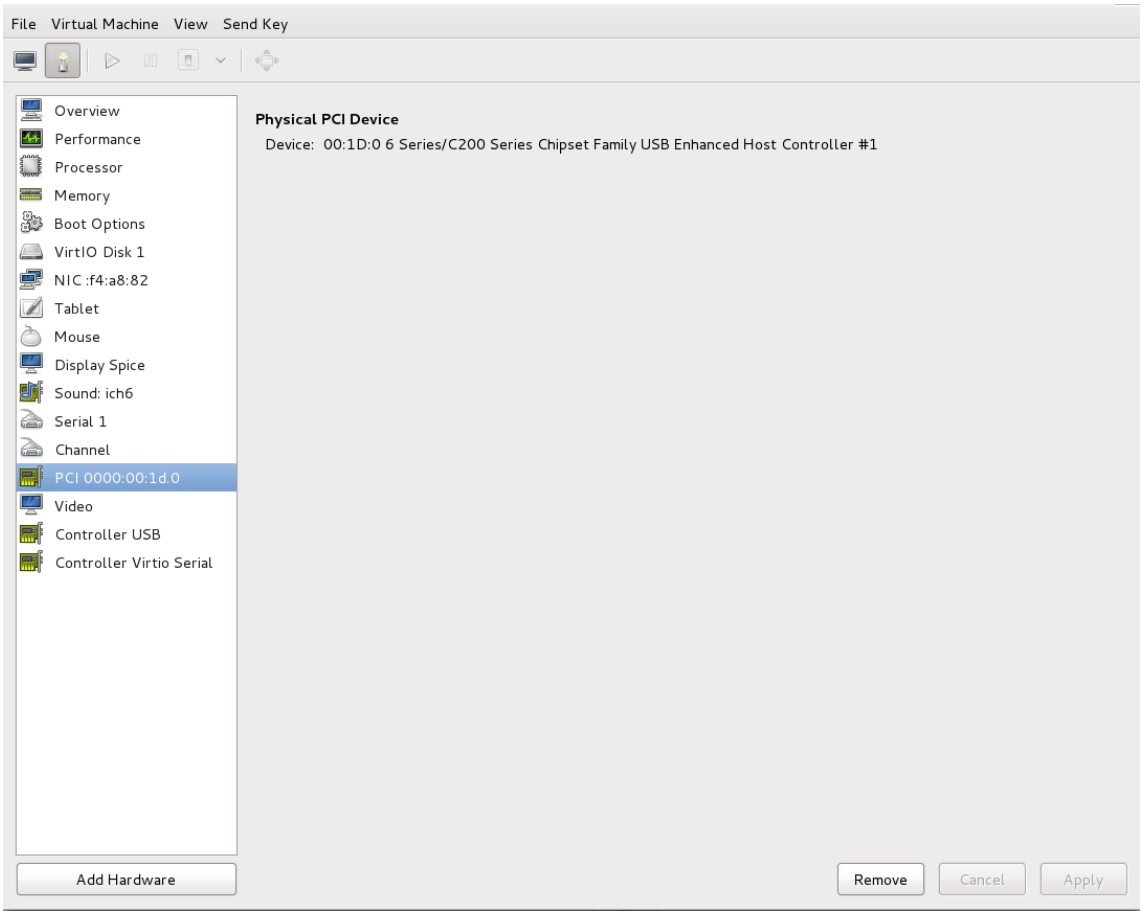
virt-manager 에서 장치가 포함된 가상 머신을 두 번 클릭합니다. 가상 하드웨어 세부 정보 표시 버튼을 선택하여 가상 하드웨어 목록을 표시합니다.

그림 9.4. 가상 하드웨어 세부 정보 버튼

2. *장치 선택 및 제거*

왼쪽 패널의 가상 장치 목록에서 분리할 PCI 장치를 선택합니다.

그림 9.5. 분리할 PCI 장치 선택



Remove (제거) 버튼을 클릭하여 확인합니다. 이제 호스트용으로 장치를 사용할 수 있습니다.

9.1.5. PCI 브리지 생성

PCI(Deviceer Component Interconnects) 브리지는 네트워크 카드, 모뎀 및 사운드 카드와 같은 장치에 연결하는 데 사용됩니다. 물리적 대응과 마찬가지로 가상 장치를 PCI 브리지에도 연결할 수 있습니다.

이전에는 31개의 PCI 장치만 게스트 가상 머신에 추가할 수 있었습니다. 이제 31st PCI 장치를 추가하면 추가 PCI 장치를 PCI 브리지로 이동하는 31번째 슬롯에 PCI 브리지가 자동으로 배치됩니다. 각 PCI 브릿지에는 31 개의 추가 장치를 위한 31개의 슬롯이 있으며, 모두 브리지일 수 있습니다. 이러한 방식으로 게스트 가상 머신에 900개 이상의 장치를 사용할 수 있습니다.

참고

게스트 가상 머신이 실행 중인 경우에는 이 작업을 수행할 수 없습니다. 종료 중인 게스트 가상 머신에 PCI 장치를 추가해야 합니다.

9.1.6. PCI 패스스루

PCI 네트워크 장치(<소스> 요소에 의해 지정됨)는 먼저 장치의 MAC 주소를 구성된 값으로 설정하고 선택적으로 지정된 <가상 포트 요소를 사용하여 장치를 802.1Qbhable 스위치와 연결한 후 type='direct' 네트워크 장치에 대해 지정한 가상> 포트의 예를 참조하십시오. 표준 단일 포트 PCI 이더넷 카드 드라이버 설계의 제한으로 인해 - SR-IOV(Single Root I/O Virtualization) 가상 기능(VF) 장치만 이러한 방식으로 할당할 수 있습니다. 표준 단일 포트 PCI 또는 PCIe 이더넷 카드를 게스트에 할당하려면 기존 <hostdev> 장치 정의를 사용합니다.

traditional/legacy KVM 장치 할당(VFIO)이 아닌 VFIO 장치 할당(VFIO)은 UEFI Secure Boot와 호환되는 새로운 장치 할당 방법입니다. <type='hostdev'> 인터페이스에는 name 속성이 "vfio"로 설정된 선택적 드라이버 하위 요소가 있을 수 있습니다. 레거시 KVM 장치 할당을 사용하려면 name을 "kvm"으로 설정할 수 있습니다(또는 < <driver> ='kvm이 현재 기본값이므로 드라이버> 요소를 생략하기만 하면 됩니다.

참고

네트워크 장치의 지능형 패스스루는 표준 <hostdev> 장치의 기능과 매우 유사합니다. 이 방법은 통과된 장치에 대한 MAC 주소와 <가상 포트를> 지정할 수 있다는 점입니다. 이러한 기능이 필요하지 않은 경우, SR-IOV를 지원하지 않는 표준 단일 포트 PCI, PCIe 또는 USB 네트워크 카드가 있는 경우(따라서 게스트 도메인에 할당된 후 재설정 중에 구성된 MAC 주소가 손실됨) 또는 0.9.11 이전 버전의 libvirt를 사용하는 경우 표준 <hostdev> 를 사용하여 장치를 <인터페이스 유형/dev 유형> 대신 게스트에 할당해야 합니다.

그림 9.6. PCI 장치 할당의 경우 XML

```
 <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>
```

9.1.7. SR-IOV 장치를 사용하여 PCI 할당(Passthrough) 구성

이 섹션은 SR-IOV 장치 전용입니다. SR-IOV 네트워크 카드는 각각 PCI 장치 할당을 사용하여 게스트 가상 머신에 개별적으로 할당할 수 있는 여러 VF(가상 기능)를 제공합니다. 할당된 후에는 각각 전체 물리적 네트워크 장치로 작동합니다. 이를 통해 많은 게스트 가상 머신에서 직접 PCI 장치 할당의 성능 이점을 얻을 수 있으며 호스트 물리적 시스템에서 단일 슬롯만 사용할 수 있습니다.

이러한 VF는 <hostdev> 요소를 사용하여 기존 방식으로 게스트 가상 머신에 할당할 수 있지만 SR-IOV VF 네트워크 장치에는 영구적인 MAC 주소가 없으므로 호스트 물리적 머신이 재부팅될 때마다 게스트 가상 머신의 네트워크 설정을 다시 구성해야 하는 문제가 발생합니다. 이 문제를 해결하려면 VF를 호스트 물리적 머신에 할당하기 전에 MAC 주소를 설정해야 하며 게스트 가상 머신이 부팅될 때마다 이 주소를 설정해야 합니다. 이 MAC 주소와 다른 옵션을 할당하려면 절차 9.8. “SR-IOV에서 PCI 장치를 할당하기 위해 MAC 주소, vLAN 및 가상 포트 구성” 에 설명된 절차를 참조하십시오.

절차 9.8. SR-IOV에서 PCI 장치를 할당하기 위해 MAC 주소, vLAN 및 가상 포트 구성

<mac>,<vlan>, <virtualport> 요소가 <hostdev> 용으로 유효하지 않으므로 MAC 주소 할당, vLAN 태그 ID 할당 또는 가상 포트 할당과 같은 기능별 항목에는 <hostdev> 요소를 사용할 수 없습니다. <인터페이스 용으로 유효하므로 새 인터페이스> 유형에 대한 지원이 추가되었습니다(<인터페이스 type='hostdev'>). 이 새 인터페이스 장치 유형은 <인터페이스> 및 <hostdev> 의 하이브리드 역할을 합니다. 따라서 PCI 장치를 게스트 가상 머신에 할당하기 전에 libvirt 는 표시된 네트워크별 하드웨어/슬래버스(예: MAC 주소 설정, vLAN 태그 설정, 802.1Qbh 스위치)를 게스트 가상 머신의 XML 구성 파일에서 초기화합니다. vLAN 태그 설정에 대한 자세한 내용은 18.14절. “vLAN 태그 설정” 을 참조하십시오.

1. 게스트 가상 머신 종료

   virsh shutdown 명령(14.9.1절. “게스트 가상 머신 종료”참조)을 사용하여 guestVM 이라는 게스트 가상 머신을 종료합니다.

```
# virsh shutdown guestVM
```

2. *정보 수집*

   <인터페이스 type='hostdev'> 를 사용하려면 SR-IOV 가능 네트워크 카드, Intel VT-d 또는 AMD IOMMU 확장 기능을 지원하는 물리적 머신 하드웨어를 호스트해야 하며, 할당할 VF의 PCI 주소를 알아야 합니다.

3. *편집을 위해 XML 파일을 엽니다.*

   # virsh save-image-edit 명령을 실행하여 편집할 XML 파일을 엽니다(자세한 내용은 14.8.10절. "도메인 XML 구성 파일 편집" 참조). 게스트 가상 머신을 이전 실행 상태로 복원하려는 경우 --running 이 사용됩니다. 이 예제의 구성 파일의 이름은 게스트 가상 머신의 이름이 guestVM 이므로 guestVM.xml 입니다.

```
# virsh save-image-edit guestVM.xml --running
```

   guestVM.xml 이 기본 편집기에서 열립니다.

4. *XML 파일 편집*

   구성 파일(guestVM.xml)을 다음과 유사한 <장치> 항목을 갖도록 업데이트합니다.

그림 9.7. hostdev 인터페이스 유형의 샘플 도메인 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>
```

   MAC 주소를 제공하지 않으면 다른 유형의 인터페이스 장치와 마찬가지로 MAC 주소가 자동으로 생성됩니다. 또한 <virtualport> 요소는 802.11Qgh 하드웨어 스위치 (802.11Qbg (a.k.a))에 연결하는 경우에만 사용됩니다. "VEPA") 스위치는 현재 지원되지 않습니다.

5. 게스트 가상 머신 다시 시작

   virsh start 명령을 실행하여 첫 번째 단계에서 종료한 게스트 가상 머신을 재시작합니다(예: **guestVM**을 게스트 가상 시스템의 도메인 이름으로 사용). 자세한 내용은 14.8.1절. “정의된 도메인 시작”를 참조하십시오.

   ```
   # virsh start guestVM
   ```

   게스트 가상 머신이 시작되면 실제 호스트 시스템의 어댑터에 의해 제공된 네트워크 장치가 구성된 **MAC** 주소와 함께 표시됩니다. 이 **MAC** 주소는 게스트 가상 시스템 및 호스트 물리적 시스템 재부팅 시 변경되지 않은 상태로 유지됩니다.

9.1.8. SR-IOV 가상 함수 풀에서 PCI 장치 할당 설정

특정 가상 기능 (VF)의 **PCI** 주소를 게스트 구성에 하드 코딩하면 다음과 같은 두 가지 심각한 제한 사항이 있습니다.

- 지정된 **VF**는 게스트 가상 머신이 시작될 때마다 사용할 수 있어야 합니다. 즉, 관리자가 각 **VF**를 단일 게스트 가상 머신에 영구적으로 할당해야 합니다(또는 게스트 가상 머신이 시작될 때마다 사용되지 않는 **VF**의 **PCI** 주소를 지정하도록 모든 게스트 가상 머신의 구성 파일 수정).

- 게스트 가상 시스템이 다른 호스트 물리적 시스템으로 이동하는 경우 호스트 물리적 시스템은 **PCI** 버스의 동일한 위치에 동일한 하드웨어가 있어야 합니다(또는 다시 시작 전에 게스트 가상 머신 구성을 수정해야 함).

SR-IOV 장치의 모든 **VF**가 포함된 장치 풀로 **libvirt** 네트워크를 생성하여 이러한 문제를 모두 방지할 수 있습니다. 이 작업이 완료되면 이 네트워크를 참조하도록 게스트 가상 머신을 구성합니다. 게스트를 시작할 때마다 단일 **VF**가 풀에서 할당되어 게스트 가상 머신에 할당됩니다. 게스트 가상 머신이 중지되면 **VF**가 다른 게스트 가상 머신에서 사용할 풀로 반환됩니다.

절차 **9.9.** 장치 풀 생성

1. 게스트 가상 머신 종료

   virsh shutdown 명령(14.9절. “게스트 가상 시스템의 종료, 재부팅 및 종료”참조)을 사용하여 **guestVM** 이라는 게스트 가상 머신을 종료합니다.

   ```
   # virsh shutdown guestVM
   ```

2. 구성 파일 생성

   선택한 편집기를 사용하면 **/tmp** 디렉터리에 **XML** 파일(예: **passthrough.xml**)이 생성됩니

다. **pf dev='eth3'** 을 자체 **SR-IOV** 장치의 **netdev** 이름으로 교체해야 합니다.

다음은 호스트 물리적 머신의 **"eth3'**에 있는 **PF(**물리 기능**)** 를 사용하여 **SR-IOV** 어댑터에 대한 모든 **VF** 풀을 사용할 수 있도록 하는 네트워크 정의 예제입니다.

그림 **9.8.** 네트워크 정의 도메인 **XML** 샘플

```
<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. 새 **XML** 파일 로드

다음 명령을 실행하여 **/tmp/passthrough.xml** 을 이전 단계에서 생성한 **XML** 파일의 이름 및 위치로 바꿉니다.

```
# virsh net-define /tmp/passthrough.xml
```

4. 게스트 다시 시작

passthrough.xml 을 이전 단계에서 생성한 **XML** 파일의 이름으로 교체하여 다음을 실행합니다.

```
# virsh net-autostart passthrough # virsh net-start passthrough
```

5. 게스트 가상 머신 다시 시작

virsh start 명령을 실행하여 첫 번째 단계에서 종료한 게스트 가상 머신을 재시작합니다**(**예**: guestVM**을 게스트 가상 시스템의 도메인 이름으로 사용**).** 자세한 내용은 **14.8.1**절**. "**정의된 도메인 시작**"** 를 참조하십시오.

```
# virsh start guestVM
```

6. 장치에 대한 패스스루 시작

단일 장치만 표시되지만 **libvirt**는 해당 **PF**와 연결된 모든 **VF** 목록을 자동으로 파생시킵니다. 그러면 다음과 같이 게스트 가상 머신이 도메인 **XML**의 인터페이스 정의로 처음 시작될 때 **libvirt** 가 자동으로 파생됩니다.

그림 9.9. 인터페이스 네트워크 정의를 위한 샘플 도메인 XML

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

7. 검증

네트워크를 사용하는 첫 번째 게스트를 시작한 후 **virsh net-dumpxml passthrough** 명령을 실행하여 확인할 수 있습니다. 다음과 유사한 출력을 얻을 수 있습니다.

그림 9.10. XML 덤프 파일 패스스루 콘텐츠

```
<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>
```

9.2. USB 장치

이 섹션에서는 USB 장치를 처리하는 데 필요한 명령을 제공합니다.

9.2.1. 게스트 가상 머신에 USB 장치 할당

웹 카메라, 카드 리더, 키보드 또는 마우스와 같은 대부분의 장치는 USB 포트 및 케이블을 사용하여 컴퓨터에 연결됩니다. 이러한 장치를 게스트 가상 머신에 전달하는 방법은 다음 두 가지가 있습니다.

- USB 패스스루 사용 - 이 경우 장치는 게스트 가상 머신을 호스팅하는 호스트 물리적 머신에 물리적으로 연결되어 있어야 합니다. 이 경우에는 SPICE가 필요하지 않습니다. 명령줄 또는 virt-manager 를 사용하여 호스트의 USB 장치를 게스트로 전달할 수 있습니다. 가상화 관리자 지침은 15.3.1절. “게스트 가상 머신에 USB 장치 연결” 를 참조하십시오.

참고

virt-manager 를 핫플러그 또는 연결 해제 장치에 사용해서는 안 됩니다. **USB** 장치를 핫 플러그/또는 핫 플러그 해제하려면 절차 14.1. “게스트 가상 머신에서 사용할 USB 장치 핫플러그” 을 참조하십시오.

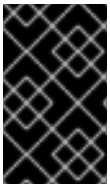
- **USB** 리디렉션 사용 - 데이터 센터에서 실행 중인 호스트 물리적 시스템이 있는 경우 **USB** 리디렉션이 가장 적합합니다. 사용자는 로컬 머신 또는 씬 클라이언트에서 게스트 가상 머신에 연결합니다. 이 로컬 머신에는 **SPICE** 클라이언트가 있습니다. 사용자는 모든 **USB** 장치를 씬 클라이언트에 연결할 수 있으며 **SPICE** 클라이언트는 씬 클라이언트에서 실행되는 게스트 가상 시스템에서 사용할 수 있도록 장치를 데이터 센터의 호스트 물리적 시스템으로 리디렉션합니다. **virt-manager** 를 사용하여 **USB** 리디렉션에 대한 지침은 15.3.1절. “게스트 가상 머신에 USB 장치 연결” 를 참조하십시오. **USB** 리디렉션은 **TCP** 프로토콜을 사용할 수 없습니다(**BZ#1085318**참조).

9.2.2. USB 장치 리디렉션에서 제한 설정

리디렉션에서 특정 장치를 필터링하려면 필터 속성을 **-device usb-redir** 에 전달합니다. **filter** 속성은 필터 규칙으로 구성된 문자열을 사용하며, 규칙의 형식은 다음과 같습니다.

```
<class>:<vendor>:<product>:<version>:<allow>
```

값 **-1** 을 사용하여 특정 필드에 대한 값을 수락하도록 지정합니다. |를 구분 기호로 사용하여 동일한 명령줄에서 여러 규칙을 사용할 수 있습니다.



중요

장치가 규칙 필터 중 어느 것과도 일치하지 않으면 리디렉션할 수 없습니다!

예 9.1. Windows 게스트 가상 시스템으로 리디렉션 제한의 예

1. **Windows 7** 게스트 가상 머신을 준비합니다.
2. 게스트 가상 머신의 **domain xml** 파일에 다음 코드 발췌를 추가합니다.

```
<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. *guest 가상 머신을 시작하고 다음을 실행하여 설정 변경 사항을 확인합니다.*

```
#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 장치를 호스트 물리적 머신에 연결하고 virt-manager 를 사용하여 게스트 가상 머신에 연결합니다.*

5. *메뉴에서 리디렉션 USB 서비스를 클릭합니다. "호스트 정책에서 일부 USB 장치를 차단함"라는 메시지가 표시됩니다. 확인 을 클릭하여 확인하고 계속합니다.Click OK to confirm and continue.*

   필터가 적용됩니다.

6. *필터가 USB 장치 공급 업체 및 제품을 올바르게 확인하려면 USB 리디렉션을 허용하도록 게스트 가상 시스템의 도메인 XML에서 다음과 같이 변경합니다.*

```
<redirfilter>
  <usbdev class='0x08' vendor='0x0951' product='0x1625' version='2.0' allow='yes'/>
  <usbdev allow='no'/>
 </redirfilter>
```

7. *게스트 가상 머신을 재시작한 다음 virt-viewer 를 사용하여 게스트 가상 머신에 연결합니다. 이제 USB 장치가 트래픽을 게스트 가상 머신으로 리디렉션합니다.*

9.3. 장치 컨트롤러 구성

게스트 가상 머신 아키텍처에 따라 일부 장치 버스는 가상 컨트롤러에 연결된 가상 장치 그룹과 함께 두 번 이상 표시될 수 있습니다. 일반적으로 libvirt는 명시적 XML 태그 없이 이러한 컨트롤러를 자동으로 유추할 수 있지만 경우에 따라 가상 컨트롤러 요소를 명시적으로 설정하는 것이 좋습니다.

그림 9.11. 가상 컨트롤러의 도메인 XML 예

```
...
<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>
...
```

각 컨트롤러에는 필수 특성 **<컨트롤러 유형>** 이 있으며 다음 중 하나여야 합니다.

- **IDE**
- **fdc**
- **scsi**
- **SATA**
- **usb**
- **CCID**
- **virtio-serial**
- **pci**

<컨트롤러> 요소에는 필수 특성 **<컨트롤러 인덱스>** 가 있으며, 이는 버스 컨트롤러가 발생하는 순서를 설명하는 **10**진수 정수입니다(**<address> elements**의 컨트롤러 특성에 사용). **<controller type ='virtio-serial'>** 에 두 개의 추가 선택적 속성(명명 포트 및 벡터)이 있는 경우 컨트롤러를 통해 연결할 수 있는 장

치 수를 제어합니다. Red Hat Enterprise Linux 6는 장치당 32 벡터를 초과하는 사용을 지원하지 않습니다. 벡터를 더 많이 사용하면 게스트 가상 머신을 마이그레이션할 때 오류가 발생합니다.

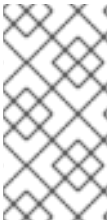
<controller type ='scsi'> 의 경우 다음과 같은 값을 가질 수 있는 선택적 특성 모델 모델이 있습니다.

- *auto*
- *buslogic*
- *ibmvscsi*
- *lsilogic*
- *lsisas1068*
- *lsisas1078*
- *virtio-scsi*
- *vmpvscsi*

<controller type ='usb'> 의 경우 다음과 같은 값을 가질 수 있는 선택적 특성 모델 모델이 있습니다.

- *piix3-uhci*
- *piix4-uhci*
- *ehci*

- *ich9-ehci1*
- *ich9-uhci1*
- *ich9-uhci2*
- *ich9-uhci3*
- *vt82c686b-uhci*
- *pci-ohci*
- *nec-xhci*



참고

*게스트 가상 머신에 대해 **USB** 버스를 명시적으로 비활성화해야 하는 경우 **<model='none'>** 을 사용할 수 있습니다. .*

PCI 또는 USB 버스의 장치 자체인 컨트롤러의 경우 선택적 하위 요소 <주소는> 컨트롤러의 마스터 버스와의 정확한 관계를 9.4절. "장치 주소 설정" 에 표시된 대로 지정할 수 있습니다.

선택적인 하위 요소 <드라이버는 드라이버> 특정 옵션을 지정할 수 있습니다. 현재는 컨트롤러의 대기열 수를 지정하는 특성 대기열만 지원합니다. 최상의 성능을 위해 vCPU 수와 일치하는 값을 지정하는 것이 좋습니다.

USB 도우미 컨트롤러는 해당 <마스터> 컨트롤러와의 정확한 관계를 지정하기 위한 선택적 하위 요소 마스터가 있습니다. 파트너 컨트롤러는 마스터와 동일한 버스에 있으므로 companion index 값이 동일해야 합니다.

사용할 수 있는 XML 예제는 다음과 같습니다.

그림 9.12. USB 컨트롤러의 도메인 XML 예

```
  ...
<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 컨트롤러에는 다음과 같은 가능한 값이 있는 선택적 모델 속성이 있습니다.

- pci-root
- pcie-root
- pci-bridge
- dmi-to-pci-bridge

루트 컨트롤러(pci-root 및 pcs ie-root)에는 64비트 PCI 홀트가 있어야 하는 pcihole64 요소 또는 pcihole64 의단위 속성(KB)을 지정하는 pcihole64 요소가 있습니다. 일부 게스트 가상 머신 (예: Windows Server 2003)은 단위가 비활성화 되지 않는 한 (unit ='0'으로 설정) 충돌을 일으킬 수 있습니다.

암시적 PCI 버스를 제공하는 시스템 유형의 경우 index='0' 인 pci-root 컨트롤러가 자동으로 추가되어 PCI 장치를 사용해야 합니다. pci-root에는 주소가 없습니다. PCI 브리지는 model='pci-root' 에서 제공하는 하나의 버스에 배치할 장치가 너무 많거나 0보다 큰 PCI 버스 번호가 지정된 경우 자동으로 추가됩니다. PCI 브리지를 수동으로 지정할 수도 있지만 주소는 이미 지정된 PCI 컨트롤러에서 제공하는 PCI 버스만 참조해야 합니다. PCI 컨트롤러 인덱스에서 간격을 남겨 두면 구성이 유효하지 않을 수 있습니다. 다음 XML 예제를 <devices> 섹션에 추가할 수 있습니다.

그림 9.13. PCI 브리지의 도메인 XML 예

```
...
<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>
...
```

암시적 PCI Express(PCI Express) 버스(예: Q35 칩셋 기반의 시스템 유형)를 제공하는 머신 유형의 경우 index='0' 을 사용하는pcie-root 컨트롤러가 도메인 구성에 자동 추가됩니다.pcie-root는 주소가 없지만 31 슬롯(수적으로 1-31)을 제공하는 데도 사용할 수 있습니다. PCie-root 컨트롤러가 있는 시스템에서 표준 PCI 장치를 연결하기 위해 model='dmi-to-pci-bridge' 가 있는 pci 컨트롤러가 자동으로 추가됩니다. dmi-to-pci-bridge 컨트롤러 플러그 인은 PCIe 슬롯에 플러그인하고(pcie-root에서 제공하는 대로) 31 표준 PCI 슬롯을 제공합니다(hot-pluggable). 게스트 시스템에 핫플러그 가능한 PCI 슬롯을 사용하기 위해 pci-bridge 컨트롤러도 자동으로 생성되고 자동 생성된 dmi-to-pci-bridge 컨트롤러 슬롯 중 하나에 연결됩니다. PCI 주소가 있는 모든 게스트 장치도 이 pci-bridge 장치에 배치됩니다.

그림 9.14. PCIe(PCI express)의 도메인 XML 예

```
  ...
<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>
...
```

9.4. 장치 주소 설정

많은 장치에는 게스트 가상 머신에 제공되는 가상 버스에 장치가 배치되는 위치를 설명하는 데 사용되는 선택적 <주소> 하위 요소가 있습니다. 입력 시 주소(또는 주소 내의 선택적 속성)를 생략하면 libvirt 에서 적절한 주소를 생성합니다. 그러나 레이아웃을 추가로 제어해야 하는 경우 명시적 주소가 필요합니다. <address> 요소를 포함한 도메인 XML 장치 예제는 그림 9.6. “PCI 장치 할당의 경우 XML” 를 참조하십시오.

모든 주소에는 장치가 있는 버스를 설명하는 필수 속성 유형이 있습니다. 지정된 장치에 사용할 주소 중 일부는 장치 및 게스트 가상 머신의 아키텍처가 제한됩니다. 예를 들어, <디스크> 장치는 type='drive' 를

사용하지만 <콘솔> 장치는 i686 또는 x86_64 게스트 가상 머신 아키텍처 에서 type='pci' 을 사용합니다. 각 주소 유형에는 장치가 표에 설명된 대로 배치되는 버스 위치를 제어하는 추가 선택적 속성이 있습니다.

표 9.1. 지원되는 장치 주소 유형

주소 유형	설명
type='pci'	PCI 주소에는 다음과 같은 추가 속성이 있습니다. ● 도메인 (현재 qemu에서 사용되지 않는 2바이트 16x 정수) ● 버스 (0xff와 0xff 사이의 16진수 값) ● 슬롯 (0x0과 0x1f 사이의 16진수 값, 포함) ● 기능 (0)에서 7 사이의 값 (포함) ● Multifunction 컨트롤은 PCI 제어 레지스터의 특정 슬롯/기능에 대해 다중 함수 비트를 켜면 기본적으로 'off'로 설정되지만, 여러 기능이 사용되는 슬롯의 함수 0에 대해 'on'으로 설정해야 합니다.
type='drive'	드라이브 주소에는 다음과 같은 추가 속성이 있습니다. ● 컨트롤러(두 자리 컨트롤러 번호) ● 버스(두 자리 버스 번호 ● 목표 (두 자리 버스 번호) ● (버스에 있는 2자리 단위 번호)
type='virtio-serial'	각 virtio-serial 주소에는 다음과 같은 추가 속성이 있습니다. ● 컨트롤러(두 자리 컨트롤러 번호) ● 버스(두 자리 버스 번호) ● 슬롯 (버스 내의 두 자리 슬롯)
type='ccid'	스마트 카드의 경우 CCID 주소에는 다음과 같은 추가 속성이 있습니다. ● 버스(두 자리 버스 번호) ● 슬롯 속성(버스 내의 두 자리 슬롯)

주소 유형	설명
type='usb'	USB 주소에는 다음과 같은 추가 속성이 있습니다. ● 버스 (가 0에서 0xfff 사이의 16진수 값, 포함) ● 포트(1.2 또는 2.1.3.1과 같이 최대 4개의 8진수로 구분된 표기법)
type='isa'	ISA 주소에는 다음과 같은 추가 속성이 있습니다. ● iobase ● IRQ

9.5. 게스트 가상 머신의 스토리지 컨트롤러 관리

***Red Hat Enterprise Linux 6.4**부터 **Red Hat Enterprise Linux 6.4** 이상을 실행하는 게스트 가상 머신에 **SCSI** 및 **virtio-SCSI** 장치를 추가할 수 있습니다. **virtio** 디스크와 달리 **SCSI** 장치에는 게스트 가상 시스템에 컨트롤러가 있어야 합니다. **virtio-SCSI**는 **SCSI LUN**에 직접 연결하고 **virtio-blk**에 비해 확장성을 크게 향상시킬 수 있는 기능을 제공합니다. **virtio-SCSI**의 장점은 **28**개의 장치 및 소진 **PCI** 슬롯만 처리할 수 있는 **virtio-blk**에 비해 수백 개의 장치를 처리할 수 있다는 점입니다. **virtio-SCSI**는 이제 다음과 같은 기능을 사용하여 대상 장치의 기능 세트를 상속할 수 있습니다.*

- ***virtio-scsi** 컨트롤러를 통해 가상 하드 드라이브 또는 **CD**를 연결합니다.*
- ***QEMU scsi-block** 장치를 통해 호스트에서 게스트로 물리적 **SCSI** 장치를 전달합니다.*
- *게스트당 수백 개의 장치를 사용할 수 있습니다. **virtio-blk**의 **28** 장치 제한을 개선할 수 있습니다.*

*이 섹션에서는 가상 **SCSI** 컨트롤러("호스트 버스 어댑터"라고도 함)를 생성하고 게스트 가상 시스템에 **SCSI** 스토리지를 추가하는 데 필요한 단계를 자세히 설명합니다.*

*절차 **9.10.** 가상 **SCSI** 컨트롤러 생성*

1. *게스트 가상 머신(**Guest1**)의 구성을 표시하고 기존 **SCSI** 컨트롤러를 찾습니다.*

```
# virsh dumpxml Guest1 | grep controller.*scsi
```

장치 컨트롤러가 있는 경우 명령은 다음과 유사한 하나 이상의 행을 출력합니다.

```
<controller type='scsi' model='virtio-scsi' index='0'/>
```

2. 이전 단계에서 장치 컨트롤러가 표시되지 않은 경우 다음 단계를 사용하여 새 파일에서 하나의 설명에 생성하고 가상 머신에 추가합니다.

   a. 새 파일에 **< controller>** 요소를 작성하여 장치 컨트롤러를 생성하고 **XML** 확장자로 이 파일을 저장합니다. **virtio-scsi-controller.xml (예:**

   ```
   <controller type='scsi' model='virtio-scsi'/>
   ```

   b. **virtio-scsi-controller.xml** 에서 방금 생성한 장치 컨트롤러를 게스트 가상 머신**(예: 1)** 과 연결합니다.

   ```
   # virsh attach-device --config Guest1 ~/virtio-scsi-controller.xml
   ```

   이 예에서 **--config** 옵션은 디스크에 대해 수행하는 것과 동일하게 작동합니다. 자세한 내용은 절차 **13.2.** “게스트에 물리적 블록 장치 추가” 를 참조하십시오.

3. 새 **SCSI** 디스크 또는 **CD-ROM**을 추가합니다. 새 디스크는 **13.3.1**절. “ 게스트에 파일 기반 스토리지 추가” 및 **13.3.2**절. “게스트에 하드 드라이브 및 기타 블록 장치 추가” 섹션의 방법을 사용하여 추가할 수 있습니다. **SCSI** 디스크를 만들려면 **sd** 로 시작하는 대상 장치 이름을 지정합니다.

   ```
   # virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.img sdb --cache none
   ```

   게스트 가상 머신의 드라이버 버전에 따라 실행 중인 게스트 가상 머신에 의해 새 디스크가 즉시 감지되지 않을 수 있습니다. **Red Hat Enterprise Linux** 스토리지 관리 가이드 의 단계를 따르십시오.

9.6. RNG(RANDOM NUMBER GENERATOR) 장치

virtio-rng 는 게스트 가상 머신의 운영 체제에 **RNG** 데이터를 제공하는 가상 **RNG(random number generator)** 장치로 요청 시 게스트 가상 머신에 새로운 엔트로피를 제공합니다.

RNG를 사용하면 키보드, 마우스 및 기타 입력과 같은 장치가 게스트 가상 머신에서 충분한 엔트로피를 생성하는 데 충분하지 않은 경우 특히 유용합니다. **virtio-rng** 장치는 **Red Hat Enterprise Linux** 및

Windows 게스트 가상 머신 모두에서 사용할 수 있습니다. Windows 요구 사항 설치에 대한 지침은 참고 를 참조하십시오. 별도로 명시하지 않는 한, Red Hat Enterprise Linux 및 Windows 게스트 가상 머신 모두에 대한 설명은 다음과 같습니다.

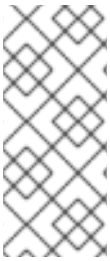
Linux 게스트 가상 머신에서 virtio-rng 가 활성화된 경우 /dev/hwrng/.의 게스트 가상 시스템에 chardev가 생성됩니다. 그러면 이 chardev를 열고 호스트 물리적 시스템에서 엔트로피를 가져올 수 있습니다. 게스트 가상 머신의 애플리케이션이 투명하게 virtio-rng 장치의 임의성을 사용할 수 있도록 하려면 게스트 가상 머신의 커널 엔트로피 풀로 /dev/hwrng/ 의 입력을 릴레이해야 합니다. 이 위치의 정보가 rgnd 데몬(rng-tools 내에 포함된 포함)과 연결된 경우 이 작업을 수행할 수 있습니다.

이러한 결합으로 인해 엔트로피가 게스트 가상 시스템의 /dev/random 파일로 라우팅됩니다. 이 프로세스는 Red Hat Enterprise Linux 6 게스트 가상 시스템에서 수동으로 수행됩니다.

Red Hat Enterprise Linux 6 게스트 가상 머신은 다음 명령을 실행하여 결합됩니다.

```
# rngd -b -r /dev/hwrng/ -o /dev/random/
```

더 많은 지원을 받으려면 man rngd 명령을 실행하여 여기에 표시된 명령 옵션에 대한 설명을 입력합니다. 자세한 내용은 virtio-rng 장치 구성은 절차 9.11. "명령줄 툴을 사용하여 virtio-rng 구현" 에서 참조하십시오.



참고

Windows 게스트 가상 머신은 viorng 드라이버를 설치해야 합니다. 가상 RNG 장치가 설치되면 Microsoft에서 제공하는 CNG(crypto 차세대) API를 사용하여 작동합니다. 드라이버가 설치되면 virtrng 장치가 RNG 공급자 목록에 나타납니다.

절차 9.11. 명령줄 툴을 사용하여 virtio-rng 구현

1. *게스트 가상 머신을 종료합니다.*

2. *터미널 창에서 virsh edit domain-name 명령을 사용하여 원하는 게스트 가상 시스템에 대한 XML 파일을 엽니다.*

3. *다음을 포함하도록 <devices> 요소를 편집합니다.*

```
...
```

```
<devices>
  <rng model='virtio'>
    <rate period="2000" bytes="1234"/>
    <backend model='random'>/dev/random</backend>
        <source mode='bind' service='1234'>
        <source mode='connect' host='192.0.2.1' service='1234'>
    </backend>
  </rng>
</devices>
...
```

10장. QEMU-IMG 및 QEMU 게스트 에이전트

이 장에는 게스트 가상 시스템과 **qemu-img** 패키지를 사용하는 데 필요한 유용한 팁과 팁이 포함되어 있습니다. **QEMU** 추적 이벤트 및 인수에 대한 정보를 찾고 있는 경우 여기에 있는 **README** 파일 **/usr/share/doc/qemu-*/README.systemtap** 을 참조하십시오.

10.1. QEMU-IMG 사용

qemu-img 명령줄 툴은 **KVM**에서 사용하는 다양한 파일 시스템을 포맷, 수정 및 확인하는 데 사용됩니다. **QEMU-img** 옵션 및 사용법은 아래에 나열되어 있습니다.

확인

디스크 이미지 파일 이름에 대해 일관성 검사를 수행합니다.

```
#  qemu-img check -f qcow2 --output=qcow2 -r all filename-img.qcow2
```

참고

qcow2 및 **vdi** 형식만 일관성 검사를 지원합니다.

r을 사용하여 검사 중에 발견된 불일치를 복구하려고 하지만 **-r** 누수 **s** 클러스터 누수와 함께 사용하면 모든 종류의 오류가 수정됩니다. 이 경우 잘못된 수정 사항을 선택하거나 이미 발생한 손상 문제를 숨길 위험이 있습니다.

커밋

지정된 파일(파일이름)에 기록된 변경 사항을 **qemu-img commit** 명령을 사용하여 파일의 기본 이미지에 커밋합니다. 필요한 경우 파일의 형식 유형(형식)을 지정합니다.

```
 # qemu-img commit [-f format] [-t cache] filename
```

convert

convert 옵션은 하나의 인식된 이미지 형식을 다른 이미지 형식으로 변환하는 데 사용됩니다.

명령 형식:

```
# qemu-img convert [-c] [-p] [-f format] [-t cache] [-O output_format] [-o options] [-S sparse_size]
filename output_filename
```

p 매개 변수는 명령의 진행 상황(선택 사항 및 모든 명령의 경우 아님)을 표시하며, **-S** 옵션을 사용하면 디스크 이미지에 포함된 스파스 파일을 생성할 수 있습니다. 스파스 파일은 모든 용도의 표준 파일처럼 작동합니다. 단, **0**만 포함된 물리 블록(**nothing**)을 제외하면 됩니다. 운영 체제에서 이 파일을 볼 때 실제로는 아무것도 사용하지 않더라도 실제 디스크 공간이 존재하는 것으로 취급되며 실제 디스크 공간을 차지합니다. 이는 디스크가 디스크 공간보다 많은 디스크 공간을 차지하는 것처럼 게스트 가상 머신에 대한 디스크를 만들 때 특히 유용합니다. 예를 들어 **10Gb**인 디스크 이미지에서 **-S**를 **50Gb**로 설정하면 **10Gb**의 디스크 공간만 실제로 사용되고 있지만 **10Gb**의 디스크 공간 크기는 **60Gb**로 나타납니다.

output_format 형식을 사용하여 디스크 이미지 파일 이름을 디스크 이미지 **output_ filename** 으로 변환합니다. 디스크 이미지는 선택적으로 **-c** 옵션으로 압축하거나, **-o encryption** 를 설정하여 **-o** 옵션으로 암호화할 수 있습니다. **o** 매개 변수와 함께 사용할 수 있는 옵션은 선택한 형식과 다릅니다.

qcow2 형식만 암호화 또는 압축을 지원합니다. **qcow2** 암호화는 보안 **128**비트 키와 함께 **AES** 형식을 사용합니다. **qcow2** 압축은 읽기 전용이므로 압축된 섹터를 **qcow2** 형식에서 변환하면 압축되지 않은 데이터로 새 형식으로 작성됩니다.

이미지 변환은 **qcow** 또는 **cow** 와 같이 커질 수 있는 형식을 사용할 때 작은 이미지를 가져오는 데 유용합니다. 빈 섹터는 감지되어 대상 이미지에서 비활성화됩니다.

create

크기 및 포맷 의 새 디스크 이미지 파일 이름을 생성합니다.

```
# qemu-img create [-f format] [-o options] filename [size][preallocation]
```

기본 이미지를 **-o backing_file=filename** 로 지정하면 이미지는 자체와 기본 이미지 간의 차이점만 기록합니다. **commit** 명령을 사용하지 않는 한 백업 파일은 수정되지 않습니다. 이 경우 크기를 지정할 필요가 없습니다.

사전 할당은 **qcow2** 이미지 생성에서만 사용할 수 있는 옵션입니다. 허용되는 값은 **-o preallocation=off|meta|full|falloc** 입니다. 사전 할당된 메타데이터가 있는 이미지는 이미지보다 큽니다. 그러나 이미지 크기가 증가하는 경우 이미지가 증가함에 따라 성능이 향상됩니다.

전체 할당을 사용하면 이미지가 큰 데 시간이 오래 걸릴 수 있습니다. 전체 할당과 시간이 필요한 경우, **falloc** 를 사용하면 시간을 절약할 수 있습니다.

정보

info 매개 변수는 디스크 이미지 파일 이름에 대한 정보를 표시합니다. **info** 옵션의 형식은 다음과 같습니다.

```
# qemu-img info [-f format] filename
```

이 명령은 종종 표시된 크기와 다를 수 있는 디스크에 예약된 크기를 검색하는 데 사용됩니다. 디스크 이미지에 스냅샷을 저장하면 해당 스냅샷도 표시됩니다. 예를 들어 이 명령은 블록 장치의 **qcow2** 이미지에서 가져온 공간 수를 표시합니다. 이 작업은 **qemu-img** 를 실행하여 수행됩니다. 사용 중인 이미지가 **qemu-img check** 명령과 함께 **qemu-img info** 명령의 출력과 일치하는지 확인할 수 있습니다. 10.1절. “qemu-img 사용” 에서 참조하십시오.

```
# 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
```

map

 # qemu-img 맵 [-f format] [--output=output_format] 파일 이름 명령은 이미지 파일 이름의 메타데이터와 백업 파일 체인을 덤프합니다. 특히 이 명령은 백업 파일 체인에 할당하는 최상위 파일과 함께 지정된 파일의 모든 섹터의 할당 상태를 덤프합니다. 예를 들어 **c.qcow2** → **b.qcow2** → **a.qcow2**와 같은 체인이 있는 경우 **a.qcow2**가 원래 파일인 경우 **b.qcow2** 및 **c.qcow2**는 **b.qcow2**의 **delta** 파일입니다. 이 체인이 생성되면 일반 이미지 데이터를 저장하고 파일에 어떤 파일에 있는지와 파일 내에 있는 위치에 대한 정보를 저장합니다. 이러한 정보를 이미지 메타데이터라고 합니다. **f** 형식 옵션은 지정된 이미지 파일의 형식입니다. **raw, qcow2, vhdx** 및 **vmdk**와 같은 형식을 사용할 수 있습니다. 사용 가능한 출력 옵션은 **human** 및 **json** 입니다.

- 사용자가 기본 설정입니다. 이 형식은 사람의 눈에 더 쉽게 읽을 수 있도록 설계되어 있으므로 이 형식을 구문 분석해서는 안 됩니다. 명확성과 단순성을 위해 기본 사람 형식은 파일의 **0**이 아닌 영역만 덤프합니다. 파일의 알려진 **0** 부분은 완전히 생략되며 체인 전체에 할당되지 않은 부분에도 적용됩니다. 명령을 실행하면 **qemu-img** 출력이 데이터를 읽을 수 있는 파일과 파일의 오프셋을 식별합니다. 출력은 네 개의 열이 있는 테이블로 표시됩니다. 처음 **3** 개는 **16**진수입니다.

  ```
  # qemu-img map -f qcow2 --output=human /tmp/test.qcow2
  Offset          Length          Mapped to       File
  0               0x20000         0x50000         /tmp/test.qcow2
  0x100000        0x80000         0x70000         /tmp/test.qcow2
  0x200000        0x1f0000        0xf0000         /tmp/test.qcow2
  0x3c00000       0x20000         0x2e0000        /tmp/test.qcow2
  0x3fd0000       0x10000         0x300000        /tmp/test.qcow2
  ```

- **JSON** 또는 **JSON(JavaScript Object Notation)**은 사람이 읽을 수 있지만 프로그래밍 언어이므로 구문 분석하도록 설계되었습니다. 예를 들어 구문 분석기에서 **"qemu-img 맵"**의 출력을 구

문 분석하려면 **--output=json** 옵션을 사용해야 합니다.

```
# qemu-img map -f qcow2 --output=json /tmp/test.qcow2
[{ "start": 0, "length": 131072, "depth": 0, "zero": false, "data": true, "offset": 327680},
{ "start": 131072, "length": 917504, "depth": 0, "zero": true, "data": false},
```

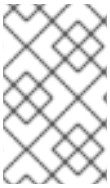
JSON 형식에 대한 자세한 내용은 **qemu-img(1)** 매뉴얼 페이지를 참조하십시오.

Rebase

이미지의 백업 파일을 변경합니다.

```
# qemu-img rebase [-f format] [-t cache] [-p] [-u] -b backing_file [-F backing_format] filename
```

백업 파일이 **backing_file** 으로 변경되고 (파일 이름의 형식을 지원하는 경우 백업 파일 형식이 **backing_format**)로 변경됩니다.



참고

qcow2 형식만 백업 파일 변경(리베이스)을 지원합니다.

리베이스 는 안전 및 안전하지 않음이라는 두 가지 모드를 사용할 수 있습니다.

안전 모드는 기본적으로 사용되며 실제 재베이스 작업을 수행합니다. 새로운 백업 파일은 이전 버전과 다를 수 있으며 **qemu-img rebase** 명령은 **guest** 가상 시스템의 파일 이름의 내용을 변경하지 않고 유지합니다. 이를 위해 **backing_file** 과 파일 이름의 이전 백업 파일 간에 다른 모든 클러스터는 백업 파일을 변경하기 전에 파일 이름으로 병합됩니다.

안전 모드는 이미지 변환과 비교했을 때 비용이 많이 드는 작업입니다. 이전 백업 파일이 성공적으로 작성되려면 필요합니다.

안전하지 않은 모드는 **-u** 옵션이 **qemu-img** 리베이스 에 전달되는 경우 사용됩니다. 이 모드에서는 파일 콘텐츠에 대한 검사가 수행되지 않고 백업 파일 이름과 파일 형식만 변경됩니다. 새 백업 파일이 올바르게 지정되었는지 또는 이미지의 게스트 눈에 띄는 콘텐츠가 손상되는지 확인합니다.

이 모드는 백업 파일의 이름을 변경하거나 이동하는 데 유용합니다. 액세스 가능한 이전 백업 파일없이 사용할 수 있습니다. 예를 들어 백업 파일이 이미 이동되었거나 이름이 변경된 이미지를 수정하는 데 사용

할 수 있습니다.

크기 조정

*크기 크기로 생성된 것처럼 디스크 이미지 파일 이름을 변경합니다. 버전에 관계없이 원시 형식의 이미지만 크기를 조정할 수 있습니다. **Red Hat Enterprise Linux 6.1** 이상에서는 **qcow2** 형식의 이미지를 확장(하지만 축소할 수 없음)할 수 있는 기능이 추가되었습니다.*

다음을 사용하여 디스크 이미지 파일 이름의 크기를 바이트 크기로 설정합니다.

```
# qemu-img resize filename size
```

*디스크 이미지의 현재 크기를 기준으로 크기를 조정할 수도 있습니다. 현재 크기를 기준으로 크기를 지정하려면 증가 하려는 바이트 수를 접두사로 추가하거나 **-** 는 해당 바이트 수만큼 디스크 이미지 크기를 줄입니다. 단위 접미사를 추가하면 킬로바이트(**K**), 메가바이트(**M**), 기가바이트(**G**) 또는 테라바이트(**T**)로 이미지 크기를 설정할 수 있습니다.*

```
# qemu-img resize filename [+|-]size[K|M|G|T]
```

주의

*디스크 이미지를 축소하려면 먼저 할당된 파일 시스템과 파티션 크기를 줄이기 위해 **VM** 자체 내에서 파일 시스템 및 파티션 툴을 사용해야 합니다. 그렇게 하지 않으면 데이터 손실이 발생합니다.*

*이 명령을 사용하여 디스크 이미지를 확장한 후에는 **VM** 내부에서 파일 시스템 및 파티션 툴을 사용하여 실제로 장치의 새 공간을 사용해야 합니다.*

스냅샷

이미지의 기존스냅샷(파일이름)을 나열, 적용, 생성 또는 삭제합니다.

```
# qemu-img snapshot [ -l | -a snapshot | -c snapshot | -d snapshot ] filename
```

***-L**은 지정된 디스크 이미지와 연결된 모든 스냅샷을 나열합니다. **apply** 옵션인 **-a** 은 디스크 이미지(파일 이름)를 이전에 저장한 스냅샷 의 상태로 되돌립니다. **-C**는 이미지(파일이름)의스냅샷(스냅샷)을 생*

성합니다. -D는 지정된 스냅샷을 삭제합니다.

지원되는 형식

QEMU-img 는 다음 형식 중 하나로 파일을 변환하도록 설계되었습니다.

raw

원시 디스크 이미지 형식(기본값). 가장 빠른 파일 기반 형식일 수 있습니다. 파일 시스템이 허점이 있는 경우(예: Windows의 Linux 또는 NTFS에서 ext2 또는 ext3의 ext3에서) 기록된 섹터만 공간을 예약합니다. qemu-img 정보를 사용하여 Unix/Linux에서 이미지 또는 ls -ls 에서 사용하는 실제 크기를 가져옵니다. 원시 이미지는 최적의 성능을 제공하지만 원시 이미지에서 매우 기본적인 기능만 사용할 수 있습니다(예: 스냅샷을 사용할 수 없음).

qcow2

QEMU 이미지 형식: 최상의 기능을 갖춘 가장 유연한 형식인 QEMU 이미지 형식입니다. 이 도구를 사용하면 선택적 AES 암호화, zlib 기반 압축, 여러 VM 스냅샷 지원 및 소형 이미지(Windows의 비 NTFS 파일 시스템)를 지원하지 않는 파일 시스템에서 유용합니다. 이 확장 기능 세트는 성능 비용으로 제공됩니다.

게스트 가상 머신 또는 호스트 물리적 머신 머신에서 실행되는 데 위의 형식만 사용할 수 있지만 qemu-img 는 원시 또는 qcow2 형식으로 변환하기 위해 다음 형식을 인식하고 지원합니다. 이미지 형식은 일반적으로 자동으로 탐지됩니다. 이러한 형식을 raw 또는 qcow2 로 변환하는 것 외에도 raw 또는 qcow2 에서 원래 형식으로 변환할 수 있습니다.

Bochs

디스크 이미지 형식입니다.

cloop

Linux Compressed loop 이미지, 예를 들어 Knoppix CD-ROM과 같이 직접 압축된 CD-ROM 이미지를 재사용할 수 있는 경우에만 유용합니다.

cow

User Mode Linux Copy On Write 이미지 형식입니다. cow 형식은 이전 버전과의 호환성을 위해서만 포함됩니다. Windows에서는 작동하지 않습니다.

dmg

Mac 디스크 이미지 형식입니다.

nbd

네트워크 블록 장치.

parallels

가상화 디스크 이미지 형식을 병렬로 지원합니다.

QCOW

이전 QEMU 이미지 형식. 이전 버전과의 호환성을 위해서만 포함되어 있습니다.

vdi

Oracle VM350 하드 디스크 이미지 형식입니다.

vmdk

VMware 호환 이미지 형식(버전 1 및 2에 대한 읽기-쓰기 지원 및 버전 3에 대한 읽기 전용 지원).

vpc

Windows 가상 PC 디스크 이미지 형식입니다. 또한 vhd 또는 Microsoft 가상 하드 디스크 이미지 형식이라고 합니다.

vvfat

가상 VFAT 디스크 이미지 형식입니다.

10.2. QEMU 게스트 에이전트

QEMU 게스트 에이전트는 게스트 내부에서 실행되며 호스트 머신에서 libvirt를 사용하여 게스트 운영 체제에 명령을 실행할 수 있습니다. 그러면 게스트 운영 체제가 해당 명령에 비동기적으로 응답합니다. 이 장에서는 게스트 에이전트에서 사용할 수 있는 libvirt 명령과 옵션에 대해 설명합니다.

중요

신뢰할 수 있는 게스트가 실행되는 경우에만 게스트 에이전트를 사용하는 것이 안전합니다. 신뢰할 수 없는 게스트는 게스트 에이전트 프로토콜을 악의적으로 무시하거나 악용할 수 있으며, 호스트의 서비스 거부 공격을 방지하기 위해 기본 제공 보호 장치가 존재하더라도 호스트에서 운영에 대한 게스트 공동 작업이 예상대로 실행되어야 합니다.

QEMU 게스트 에이전트는 게스트가 실행되는 동안 가상 **CPU(vCPU)**를 활성화 및 비활성화하는 데 사용할 수 있으므로 핫 플러그 및 핫 플러그 해제 기능을 사용하지 않고 **vCPU** 수를 조정할 수 있습니다. 자세한 내용은 **14.13.6절. "가상 CPU 수 구성"**를 참조하십시오.

10.2.1. 게스트 에이전트 설치 및 활성화

yum install qemu-guest-agent 명령을 사용하여 게스트 가상 머신에 **qemu-guest-agent**를 설치하고 서비스(**qemu-guest-agent.service**)로 모든 부팅 시 자동으로 실행되도록 합니다.

10.2.2. 게스트 에이전트와 호스트 간의 통신 설정

호스트 시스템은 호스트와 게스트 시스템 간의 **VirtIO** 직렬 연결을 통해 게스트 에이전트와 통신합니다. **VirtIO** 직렬 채널은 문자 장치 드라이버(일반적으로 **Unix** 소켓)를 통해 호스트에 연결되고 게스트는 이 직렬 채널에서 수신 대기합니다. 다음 절차에서는 게스트 에이전트용으로 호스트 및 게스트 시스템을 설정하는 방법을 보여줍니다.

참고

Windows 게스트에서 **QEMU** 게스트 에이전트를 설정하는 방법에 대한 자세한 내용은 의 지침을 참조하십시오. **http://msdn.microsoft.com/en-us/library/windows/desktop/bb968832%28v=vs.85%29.aspx**

절차 10.1. 게스트 에이전트와 호스트 간 통신 설정

1. **게스트 XML**을 엽니다.

   QEMU 게스트 에이전트 구성으로 게스트 **XML**을 엽니다. 파일을 열려면 게스트 이름이 필요합니다. 호스트 시스템에서 **# virsh list** 명령을 사용하여 인식할 수 있는 게스트를 나열합니다. 이 예에서 게스트 이름은 **rhel6** 입니다.

   ```
   # virsh edit rhel6
   ```

2. **게스트 XML** 파일 편집

XML 파일에 다음 요소를 추가하고 변경 사항을 저장합니다.

그림 10.1. 게스트 XML을 편집하여 QEMU 게스트 에이전트 구성

```
<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/rhel6.agent'/>
  <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

3. *게스트에서 QEMU 게스트 에이전트 시작*

   아직 수행하지 않은 경우 yum install qemu-guest-agent 를 사용하여 게스트 가상 머신에서 게스트 에이전트를 다운로드하여 설치합니다. 설치가 완료되면 다음과 같이 서비스를 시작합니다.

```
# service start qemu-guest-agent
```

이제 설정된 문자 장치 드라이버를 통해 유효한 libvirt 명령을 보내 게스트와 통신할 수 있습니다.

10.2.3. QEMU 게스트 에이전트 사용

QEMU 게스트 에이전트 프로토콜 (QEMU GA) 패키지 qemu-guest-agent 는 Red Hat Enterprise Linux 6.5 이상에서 완전하게 지원됩니다. 그러나 isa-serial/virtio-serial 전송과 관련하여 다음과 같은 제한 사항이 있습니다.

- *qemu-guest-agent 는 클라이언트가 채널에 연결되어 있는지 여부를 탐지할 수 없습니다.*
- *클라이언트에서 qemu-guest-agent 의 연결이 끊어졌는지 또는 백엔드에 다시 연결되었는지 여부를 감지할 수 있는 방법은 없습니다.*
- *virtio-serial 장치가 재설정되고 qemu-guest-agent 가 채널에 연결되지 않은 경우(일반적으로 재부팅 또는 핫 플러그로 인해) 클라이언트의 데이터가 삭제됩니다.*
- *qemu-guest-agent 가 virtio-serial 장치 재설정에 따라 채널에 연결된 경우 클라이언트의 데이터가 대기열에 추가되면(및 사용 가능한 버퍼가 소진되면 제한됨) qemu-guest-agent 가 계속 실행 중인지 여부와 관계없이 제한될 수 있습니다.*

10.2.4. libvirt와 함께 QEMU 게스트 에이전트 사용

QEMU 게스트 에이전트를 설치하면 다양한 다른 libvirt 명령이 보다 강력해질 수 있습니다. 게스트 에이전트는 다음 virsh 명령을 향상시킵니다.

- *virsh shutdown --mode=agent - 이 종료 방법은 QEMU 게스트 에이전트와 함께 사용되는 virsh shutdown --mode=acpi 보다 더 안정적입니다. 에이전트가 없으면 libvirt에서 ACPI 종료 이벤트 삽입에 의존하지만 일부 게스트에서는 해당 이벤트를 무시하므로 종료되지 않습니다.*

  virsh reboot 에 대해 동일한 구문과 함께 사용할 수 있습니다.

- *virsh snapshot-create --quiesce - 게스트가 스냅샷이 생성되기 전에 I/O를 안정적인 상태로 플러시할 수 있으므로 fsck를 수행하거나 부분 데이터베이스 트랜잭션을 손실하지 않고도 스냅샷을 사용할 수 있습니다. 게스트 에이전트는 게스트 공동 작업을 제공하여 높은 수준의 디스크 콘텐츠 안정성을 허용합니다.*

- *virsh setvcpus --guest - 게스트를 종료하여 CPU를 오프라인 상태로 전환합니다.*

- *virsh dompmsuspend - 게스트 운영 체제의 전원 관리 기능을 사용하여 실행 중인 게스트를 정상적으로 종료합니다.*

10.2.5. 게스트 가상 머신 디스크 백업 생성

libvirt 는 qemu-ga 와 통신하여 게스트 가상 머신 파일 시스템의 스냅샷이 내부적으로 일관되고 필요에 따라 사용할 준비가 되어 있는지 확인할 수 있습니다. Red Hat Enterprise Linux 6의 개선 사항은 파일 및 애플리케이션 수준 동기화(flushing)를 모두 완료할 수 있도록 개발되었습니다. 게스트 시스템 관리자는 애플리케이션별 freeze/thaw 후크 스크립트를 작성하고 설치할 수 있습니다. 파일 시스템을 해제하기 전에 qemu-ga 는 기본 후크 스크립트(qemu-ga 패키지에 포함됨)를 호출합니다. 정지 프로세스는 모든 게스트 가상 머신 애플리케이션을 일시적으로 비활성화합니다.

파일 시스템이 정지되기 직전에 다음과 같은 작업이 발생합니다.

- *파일 시스템 애플리케이션/데이터베이스에서 작업 버퍼를 가상 디스크에 플러시하고 클라이언트 연결 수락을 중지합니다.*

- *애플리케이션은 데이터 파일을 일관된 상태로 전환*

- *기본 후크 스크립트 반환*
- *QEMU-ga 파일 시스템 및 관리 스택이 정지하는 경우 스냅샷 사용*
- *스냅샷이 확인됨*
- *파일 시스템 기능 재개*

충돌은 반대 순서로 발생합니다.

snapshot-create-as 명령을 사용하여 게스트 디스크의 스냅샷을 생성합니다. 이 명령에 대한 자세한 내용은 14.15.2.2절. "현재 도메인의 스냅샷 생성" 를 참조하십시오.

참고

스크립트가 데이터베이스와 통신해야 하는 경우 애플리케이션별 후크 스크립트에는 올바르게 실행하려면 다양한 SELinux 권한이 필요할 수 있습니다. 일반적으로 로컬 SELinux 정책은 이러한 목적으로 개발 및 설치해야 합니다. 파일 시스템 노드에 액세스한 후 /etc/qemu-ga/fsfreeze-hook.d/.에서 표 10.1. "QEMU 게스트 에이전트 패키지 콘텐츠" 에 나열된 restorecon -FvvR 명령을 실행할 수 있습니다.

qemu-guest-agent 바이너리 RPM에는 다음 파일이 포함되어 있습니다.

표 10.1. QEMU 게스트 에이전트 패키지 콘텐츠

파일 이름	설명
/etc/rc.d/init.d/qemu-ga	QEMU 게스트 에이전트의 서비스 제어 스크립트 (start/stop)입니다.
/etc/sysconfig/qemu-ga	QEMU 게스트 에이전트의 구성 파일은 **/etc/rc.d/init.d/qemu-ga** 제어 스크립트에서 읽습니다. 설정은 쉘 스크립트 주석이 있는 파일에 설명되어 있습니다.
/usr/bin/qemu-ga	QEMU 게스트 에이전트 바이너리 파일.
/usr/libexec/qemu-ga/	후크 스크립트의 루트 디렉터리입니다.

파일 이름	설명
/usr/libexec/qemu-ga/fsfreeze-hook	기본 후크 스크립트. 여기서는 수정할 필요가 없습니다.
/usr/libexec/qemu-ga/fsfreeze-hook.d/	개별 애플리케이션별 후크 스크립트의 디렉터리입니다. 게스트 시스템 관리자는 후크 스크립트를 이 디렉터리에 수동으로 복사하고, 적절한 파일 모드 비트를 확인한 다음, 이 디렉터리에서 **restorecon -FvvR** 을 실행해야 합니다.
/usr/share/qemu-kvm/qemu-ga/	샘플 스크립트가 있는 디렉터리(예: 용도만 해당). 여기에 포함된 스크립트는 실행되지 않습니다.

*기본 후크 스크립트인 **/usr/libexec/qemu-ga/fsfreeze-hook** 는 자체 메시지와 애플리케이션별 스크립트의 표준 출력 및 오류 메시지를 다음 로그 파일에서 기록합니다. **/var/log/qemu-ga/fsfreeze-hook.log**. 자세한 내용은 **wiki.qemu.org** 또는 **libvirt.org** 의 **qemu-guest-agent wiki** 페이지를 참조하십시오.*

10.3. WINDOWS 게스트에서 QEMU 게스트 에이전트 실행

***Red Hat Enterprise Linux** 호스트 머신은 게스트에서 **QEMU** 게스트 에이전트를 실행하여 **Windows** 게스트에 명령을 실행할 수 있습니다. 이는 **Red Hat Enterprise Linux 6.5** 이상 및 다음 **Windows** 게스트 운영 체제에서 실행되는 호스트에서 지원됩니다.*

- ***Windows XP** 서비스 팩 3 (VSS는 지원되지 않음)*
- ***Windows Server 2003 R2 - x86** 및 **AMD64** (VSS는 지원되지 않음)*
- ***Windows Server 2008***
- ***Windows Server 2008 R2***
- ***Windows 7 - x86** 및 **AMD64***
- ***Windows Server 2012***

- ***Windows Server 2012 R2***
- ***Windows 8 - x86 및 AMD64***
- ***Windows 8.1 - x86 및 AMD64***

참고

***Windows** 게스트 가상 머신에는 **Windows, qemu-guest-agent-win** 용으로 **QEMU** 게스트 에이전트 패키지가 필요합니다. 이 에이전트는 **Red Hat Enterprise Linux**에서 실행되는 **Windows** 게스트 가상 머신에 대한 **VSS(Volumeshadow Copy Service)** 지원에 필요합니다. 더 많은 정보는 여기에서 찾을 수 있습니다.*

*절차 **10.2. Windows** 게스트에서 **QEMU** 게스트 에이전트 구성*

***Red Hat Enterprise Linux** 호스트 머신에서 실행 중인 **Windows** 게스트에 대해 다음 단계를 따르십시오.*

1. ***Red Hat Enterprise Linux** 호스트 머신 준비*

   ***Red Hat Enterprise Linux** 호스트 물리적 머신에 다음 패키지가 설치되어 있는지 확인합니다.*

   - ***virtio-win (/usr/share/virtio-win/에 있음)***

   ***Windows** 게스트에서 드라이버를 복사하려면 다음 명령을 사용하여 **qxl** 드라이버에 대한 ***.iso** 파일을 만듭니다.*

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

2. ***Windows** 게스트 준비*

   *드라이버를 업데이트하려면 ***.iso** 를 **Windows** 게스트에 마운트하여 게스트에 **virtio-serial driver**를 설치합니다. 게스트를 시작한 다음, 표시된 대로 드라이버 **.iso** 파일을 게스트에 연결합니다(**hdb**라는 디스크를 사용).*

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

Windows 제어판 을 사용하여 드라이버를 설치하려면 다음 메뉴로 이동합니다.

- *virtio-win 드라이버를 설치하려면 하드웨어 및 sound > 장치 관리자 > virtio-serial 드라이버를 선택합니다.*

3. *Windows 게스트 XML 구성 파일 업데이트*

Windows 게스트의 게스트 XML 파일은 Red Hat Enterprise Linux 호스트 머신에 있습니다. 이 파일에 액세스하려면 Windows 게스트 이름이 필요합니다. 호스트 시스템에서 # virsh list 명령을 사용하여 인식할 수 있는 게스트를 나열합니다. 이 예에서 게스트 이름은 win7x86입니다.

 # virsh edit win7x86 명령을 사용하여 XML 파일에 다음 요소를 추가하고 변경 사항을 저장합니다. 이 예에서 win7x86.agent 라는 호스트에서 소스 소켓 이름은 고유해야 합니다.

그림 10.2. Windows 게스트 XML을 편집하여 QEMU 게스트 에이전트 구성

```
...
<channel type='unix'>
   <source mode='bind' path='/var/lib/libvirt/qemu/win7x86.agent'/>
   <target type='virtio' name='org.qemu.guest_agent.0'/>
   <address type='virtio-serial' controller='0' bus='0' port='1'/>
</channel>
<channel type='spicevmc'>
   <target type='virtio' name='com.redhat.spice.0'/>
   <address type='virtio-serial' controller='0' bus='0' port='2'/>
</channel>
...
```

4. *Windows 게스트 재부팅*

Windows 게스트를 재부팅하여 변경 사항을 적용합니다.

```
# virsh reboot win7x86
```

5. *Windows 게스트에서 QEMU 게스트 에이전트 준비*

Windows 게스트에서 게스트 에이전트를 준비하려면 다음을 수행합니다.

a. *최신 virtio-win 패키지 설치*

Red Hat Enterprise Linux 호스트 물리적 시스템 터미널 창에서 다음 명령을 실행하여 설치할 파일을 찾습니다. 아래에 표시된 파일은 시스템이 찾은 것과 정확히 동일하지 않을 수 있지만 최신 공식 버전이어야 합니다.

```
# rpm -qa|grep virtio-win
virtio-win-1.6.8-5.el6.noarch

# rpm -iv virtio-win-1.6.8-5.el6.noarch
```

b. *설치가 완료되었는지 확인합니다.*

***virtio-win** 패키지 설치가 완료되면 **/usr/share/virtio-win/guest-agent/** 폴더를 확인하고 **qemu-ga-x64.msi** 또는 **qemu-ga-x86.msi**라는 파일을 찾을 수 있습니다.*

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

total 1544

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

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

c. ***.msi** 파일 설치*

***Windows** 게스트(예:**win7x86**)에서 **qemu-ga-x64.msi**를 두 번 클릭하여 **qemu-ga-x86.msi**를 설치합니다. 설치가 완료되면 시스템 관리자 내에서 **Windows** 게스트에 **qemu-ga** 서비스로 표시됩니다. 이 동일한 관리자를 사용하여 서비스 상태를 모니터링할 수 있습니다.*

10.3.1. Windows 게스트의 QEMU 게스트 에이전트와 함께 libvirt 명령 사용

***QEMU** 게스트 에이전트는 **Windows** 게스트에서 다음 **virsh** 명령을 사용할 수 있습니다.*

- ***virsh shutdown --mode=agent -** 이 종료 방법은 **QEMU** 게스트 에이전트와 함께 사용되는 **virsh shutdown --mode=acpi** 보다 더 안정적입니다. 에이전트가 없으면 **libvirt**에서 **ACPI** 종료 이벤트 삽입에 의존하지만 일부 게스트에서는 해당 이벤트를 무시하므로 종료되지 않습니다.*

  ***virsh reboot** 에 대해 동일한 구문과 함께 사용할 수 있습니다.*

- ***virsh snapshot-create --quiesce -** 게스트가 스냅샷이 생성되기 전에 **I/O**를 안정적인 상태로 플러시할 수 있으므로 **fsck**를 수행하거나 부분 데이터베이스 트랜잭션을 손실하지 않고도 스냅샷을 사용할 수 있습니다. 게스트 에이전트는 게스트 공동 작업을 제공하여 높은 수준의 디스크 콘텐츠 안정성을 허용합니다.*

- ***virsh dompmsuspend -** 게스트 운영 체제의 전원 관리 기능을 사용하여 실행 중인 게스트를 정상적으로 종료합니다.*

10.4. 장치 리디렉션에서 제한 설정

리디렉션에서 특정 장치를 필터링하려면 필터 속성을 **-device usb-redir** 에 전달합니다. **filter** 속성은 필터 규칙으로 구성된 문자열을 사용합니다. 규칙 형식은 다음과 같습니다.

```
<class>:<vendor>:<product>:<version>:<allow>
```

값 **-1** 을 사용하여 특정 필드에 대한 값을 수락하도록 지정합니다. |를 구분 기호로 사용하여 동일한 명령줄에서 여러 규칙을 사용할 수 있습니다. 장치가 필터 규칙과 일치하지 않으면 리디렉션이 허용되지 않습니다.

예 10.1. Windows 게스트 가상 머신으로 리디렉션 제한

1. **Windows 7** 게스트 가상 머신을 준비합니다.

2. 게스트 가상 머신의 **XML** 파일에 다음 코드 발췌를 추가합니다.

```
<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. **guest** 가상 머신을 시작하고 다음을 실행하여 설정 변경 사항을 확인합니다.

```
# 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** 장치를 호스트 물리적 시스템에 연결하고 **virt-viewer** 를 사용하여 게스트 가상 머신에 연결합니다.

5. 메뉴에서 **USB** 장치 선택을 클릭합니다. **"호스트 정책에서 일부 USB 장치를 차단함"**라는 메시지가 표시됩니다. 확인 을 클릭하여 확인하고 계속합니다.**Click OK to confirm and continue.**

필터가 적용됩니다.

6. *필터가 **USB** 장치 공급 업체 및 제품을 올바르게 확인하려면 **USB** 리디렉션을 허용하도록 호스트 물리적 시스템의 도메인 **XML**에서 다음과 같이 변경합니다.*

```
<redirfilter>
  <usbdev class='0x08' vendor='0x0951' product='0x1625' version='2.0' allow='yes'/>
  <usbdev allow='no'/>
 </redirfilter>
```

7. *게스트 가상 머신을 재시작한 다음 **virt-viewer** 를 사용하여 게스트 가상 머신에 연결합니다. 이제 **USB** 장치가 트래픽을 게스트 가상 머신으로 리디렉션합니다.*

10.5. 가상 NIC에 연결된 호스트 물리적 머신 또는 네트워크 브리지 변경

*이 섹션에서는 게스트 가상 머신을 손상시키지 않고 게스트 가상 머신이 실행되는 동안 게스트 가상 시스템의 **vNIC**를 한 브리지에서 다른 브리지로 이동하는 방법을 보여줍니다.*

1. *다음과 유사한 구성으로 게스트 가상 머신을 준비합니다.*

```
<interface type='bridge'>
    <mac address='52:54:00:4a:c9:5e'/>
    <source bridge='virbr0'/>
    <model type='virtio'/>
</interface>
```

2. *인터페이스 업데이트를 위해 **XML** 파일을 준비합니다.*

```
# cat br1.xml
```

```
<interface type='bridge'>
    <mac address='52:54:00:4a:c9:5e'/>
    <source bridge='virbr1'/>
    <model type='virtio'/>
</interface>
```

3. *게스트 가상 머신을 시작하고 게스트 가상 머신의 네트워크 기능을 확인하고 게스트 가상 머신의 **vnetX**가 표시된 브리지에 연결되어 있는지 확인합니다.*

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
virbr0          8000.5254007da9f2       yes             virbr0-nic

vnet0
virbr1          8000.525400682996       yes             virbr1-nic
```

4. *다음 명령을 사용하여 새 인터페이스 매개변수로 게스트 가상 머신의 네트워크를 업데이트합니다.*

```
# virsh update-device test1 br1.xml

Device updated successfully
```

5. *게스트 가상 머신에서 서비스 네트워크를 다시 시작합니다. 게스트 가상 머신은 **virbr1**의 새 **IP** 주소를 가져옵니다. 게스트 가상 시스템의 **vnet0**이 새 브릿지(**virbr1**)에 연결되어 있는지 확인합니다.*

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
virbr0          8000.5254007da9f2       yes             virbr0-nic
virbr1          8000.525400682996       yes             virbr1-nic      vnet0
```

11장. 스토리지 개념

이 장에서는 스토리지 장치를 설명하고 관리하는 데 사용되는 개념을 소개합니다. 스토리지 풀 및 볼륨과 같은 용어는 다음 섹션에 설명되어 있습니다.

11.1. 스토리지 풀

스토리지 풀 은 게스트 가상 시스템에 스토리지를 제공하기 위해 **libvirt**에서 관리하는 파일, 디렉터리 또는 스토리지 장치입니다. 스토리지 풀은 로컬이거나 네트워크를 통해 공유할 수 있습니다. 스토리지 풀은 관리자 측의 스토리지 세트 양이며, 게스트 가상 머신에서 사용할 수 있는 전용 스토리지 관리자입니다. 스토리지 풀은 스토리지 관리자 또는 시스템 관리자가 스토리지 볼륨으로 분할하며 볼륨은 블록 장치로 게스트 가상 머신에 할당됩니다. 짧은 스토리지 볼륨은 디스크에 스토리지 풀을 분할하는 것입니다. 스토리지 풀은 가상 컨테이너이지만 **qemu-kvm**에서 허용되는 최대 크기 및 호스트 물리적 시스템의 디스크 크기 등 두 가지 요소로 제한됩니다. 스토리지 풀은 호스트 물리적 시스템의 디스크 크기를 초과할 수 없습니다. 최대 크기는 다음과 같습니다.

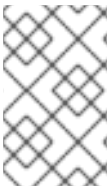
- **virtio-blk = 2^63** 바이트 또는 **8 Exabytes(raw** 파일 또는 디스크를 사용)
- **Ext4 = ~ 16TB (4KB** 블록 크기 사용)
- **XFS = ~8 Exabytes**
- **qcow2** 및 호스트 파일 시스템은 매우 큰 이미지 크기를 시도할 때 자체 메타데이터 및 확장성을 평가/**tuned**해야 합니다. 원시 디스크를 사용하면 확장성 또는 최대 크기에 영향을 줄 수 있는 레이어가 줄어듭니다.

libvirt는 디렉터리 기반 스토리지 풀인 **/var/lib/libvirt/images/** 디렉터리를 기본 스토리지 풀로 사용합니다. 기본 스토리지 풀을 다른 스토리지 풀로 변경할 수 있습니다.

- 로컬 스토리지 풀 **-** 로컬 스토리지 풀이 호스트 물리적 시스템 서버에 직접 연결됩니다. 로컬 스토리지 풀에는 로컬 디렉터리, 직접 연결된 디스크, 물리적 파티션 및 **LVM** 볼륨 그룹이 포함됩니다. 이러한 스토리지 볼륨은 게스트 가상 머신 이미지를 저장하거나 게스트 가상 머신에 추가 스토리지로 연결됩니다. 로컬 스토리지 풀이 호스트 물리적 머신 서버에 직접 연결되므로 마이그레이션 또는 게스트 가상 머신 수가 필요하지 않은 개발, 테스트 및 소규모 배포에 유용합니다. 로컬 스토리지 풀은 실시간 마이그레이션을 지원하지 않으므로 로컬 스토리지 풀은 여러 프로덕션 환경에 적합하지 않습니다.
- **Networked(**공유) 스토리지 풀 **-** 네트워크로 연결된 스토리지 풀에는 표준 프로토콜을 사용

하여 네트워크를 통해 공유하는 스토리지 장치가 포함됩니다. 네트워크 스토리지는 **virt-manager**를 사용하여 호스트 물리적 시스템 간에 가상 머신을 마이그레이션할 때 필요하지만 **virsh**로 마이그레이션할 때 선택 사항입니다. 네트워크 스토리지 풀은 **libvirt**에서 관리합니다. 네트워크 스토리지 풀에 지원되는 프로토콜은 다음과 같습니다.

- 파이버 채널 기반 **LUN**
- **iSCSI**
- **NFS**
- **GFS2**
- **InfiniBand** 및 **10GbE iWARP** 어댑터에 사용되는 블록 내보내기 프로토콜인 **SCSI RCP(SCSI RCP)**입니다.



참고

다중 경로 스토리지 풀은 완전히 지원되지 않으므로 만들거나 사용해서는 안 됩니다.

11.2. VOLUMES

스토리지 풀은 스토리지 볼륨으로 나뉩니다. 스토리지 볼륨은 물리 파티션, **LVM** 논리 볼륨, 파일 기반 디스크 이미지 및 **libvirt**에서 처리하는 기타 스토리지 유형에 대한 추상화입니다. 기본 하드웨어에 관계없이 스토리지 볼륨이 로컬 스토리지 장치로 게스트 가상 머신에 제공됩니다.

볼륨 참조

특정 볼륨을 참조하려면 다음 세 가지 접근 방법을 사용할 수 있습니다.

볼륨 및 스토리지 풀의 이름

볼륨은 해당 볼륨이 속한 스토리지 풀의 식별자와 함께 이름으로 참조할 수 있습니다. **virsh** 명령줄에서 **--pool storage_pool volume_name** 형식을 사용합니다.

*예를 들어 **guest_images** 풀의 **firstimage** 볼륨입니다.*

```
# virsh vol-info --pool guest_images firstimage
Name:           firstimage
Type:           block
Capacity:       20.00 GB
Allocation:     20.00 GB

virsh #
```

호스트 물리적 머신 시스템의 스토리지 전체 경로

볼륨은 파일 시스템의 전체 경로에 의해 참조될 수도 있습니다. 이 방법을 사용하는 경우 풀 식별자를 포함할 필요가 없습니다.

*예를 들어, **secondimage.img** 라는 볼륨이 호스트 물리적 머신 시스템에 **/images/secondimage.img** 로 표시됩니다. 이미지를 **/images/secondimage.img** 라고 합니다.*

```
# virsh vol-info /images/secondimage.img
Name:           secondimage.img
Type:           file
Capacity:       20.00 GB
Allocation:     136.00 kB
```

고유한 볼륨 키

가상화 시스템에서 볼륨이 처음 생성되면 고유 식별자가 생성되어 할당됩니다. 고유 식별자는 볼륨 키 라고 합니다. 이 볼륨 키 형식은 사용된 스토리지에 따라 다릅니다.

***LVM**과 같은 블록 기반 스토리지와 함께 사용하는 경우 볼륨 키는 다음 형식을 따를 수 있습니다.*

```
c3pKz4-qPVc-Xf7M-7WNM-WJc8-qSiz-mtvpGn
```

파일 기반 스토리지와 함께 사용하는 경우 볼륨 키가 볼륨 스토리지의 전체 경로 사본일 수 있습니다.

```
/images/secondimage.img
```

*예를 들어, **Wlvnf7-a4a3-Tlje-lje-lJDa-9eak-PZBv-LoZuUr** 의 볼륨 키가 있는 볼륨:*

```
# virsh vol-info Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
```

```
Name:           firstimage
Type:           block
Capacity:       20.00 GB
Allocation:     20.00 GB
```

virsh 는 볼륨 이름, 볼륨 경로 또는 볼륨 키 간 변환을 위한 명령을 제공합니다.

Vol-name

볼륨 경로 또는 볼륨 키와 함께 제공되는 경우 볼륨 이름을 반환합니다.

```
# virsh vol-name /dev/guest_images/firstimage
firstimage
# virsh vol-name Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
```

Vol-path

볼륨 키와 함께 제공된 경우 볼륨 경로를 반환하거나 스토리지 풀 식별자와 볼륨 이름을 반환합니다.

```
# virsh vol-path Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
/dev/guest_images/firstimage
# virsh vol-path --pool guest_images firstimage
/dev/guest_images/firstimage
```

vol-key 명령

볼륨 경로가 제공된 경우 볼륨 키를 반환하거나 스토리지 풀 식별자와 볼륨 이름을 반환합니다.

```
# virsh vol-key /dev/guest_images/firstimage
Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
# virsh vol-key --pool guest_images firstimage
Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
```

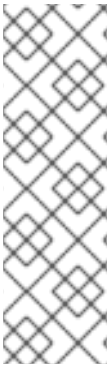
12장. 스토리지 풀

이 장에서는 다양한 유형의 스토리지 풀을 생성하는 방법을 설명합니다. 스토리지 풀 은 관리자 측의 스토리지 세트 양이며, 종종 가상 머신에서 사용할 전용 스토리지 관리자입니다. 스토리지 풀은 스토리지 관리자 또는 시스템 관리자가 스토리지 볼륨으로 구분하며 볼륨은 블록 장치로 게스트 가상 머신에 할당됩니다.

예 12.1. NFS 스토리지 풀

NFS 서버를 담당하는 스토리지 관리자가 게스트 가상 시스템의 데이터를 저장하는 공유를 생성한다고 가정합니다. 시스템 관리자는 공유 세부 정보를 사용하여 호스트 물리적 시스템의 풀을 정의합니다(nfs.example.com:/path/to/share 는 /vm_data에 마운트되어야 함). 풀이 시작되면 libvirt는 시스템 관리자가 로그인한 것처럼 NFS .example.com:/path/to/share /vmdata 와 마찬가지로 지정된 디렉터리에 공유를 마운트합니다. 풀이 autostart로 구성된 경우 libvirt는 libvirt를 시작할 때 지정된 디렉터리에 NFS 공유가 마운트되도록 합니다.

풀이 시작되면 NFS 공유 파일이 볼륨으로 보고되고 스토리지 볼륨의 경로는 libvirt API를 사용하여 쿼리됩니다. 그러면 게스트 가상 머신의 블록 장치에 대한 소스 스토리지를 설명하는 게스트 가상 머신의 XML 정의 파일의 섹션에 볼륨 경로를 복사할 수 있습니다. NFS를 사용하면 libvirt API를 사용하는 애플리케이션에서 풀 크기(공유의 최대 스토리지 용량)까지 풀 크기(NFS 공유 내의 파일)를 만들고 삭제할 수 있습니다. 일부 풀 유형이 볼륨 생성 및 삭제를 지원하는 것은 아닙니다. 풀을 중지하면 시작 작업이 무효화되어 NFS 공유를 마운트 해제합니다. 공유의 데이터는 이름에도 불구하고 destroy 작업에 의해 수정되지 않습니다. 자세한 내용은 man virsh를 참조하십시오.



참고

게스트 가상 머신을 제대로 작동하는 데는 스토리지 풀과 볼륨이 필요하지 않습니다. 풀과 볼륨은 libvirt에서 게스트 가상 머신에 특정 스토리지를 사용할 수 있도록 하는 방법을 제공하지만 일부 관리자는 자체 스토리지와 게스트 가상 머신을 관리하는 것이 좋습니다. 풀을 사용하지 않는 시스템에서 시스템 관리자는 원하는 툴을 사용하여 게스트 가상 시스템의 스토리지의 가용성을 확인해야 합니다(예: 호스트 물리적 시스템의 fstab에 NFS 공유를 추가하여 해당 공유를 부팅 시 마운트할 수 있도록 합니다.

주의

게스트에 스토리지 풀을 생성할 때 보안 고려 사항을 따르십시오. 이 정보는 에서 찾을 https://access.redhat.com/site/documentation/ 수 있는 Red Hat Enterprise Linux Virtualization 보안 가이드에서 자세히 설명합니다.

12.1. 디스크 기반 스토리지 풀

이 섹션에서는 게스트 가상 머신에 대한 디스크 기반 스토리지 장치를 생성하는 방법을 설명합니다.

주의

게스트에는 전체 디스크 또는 블록 장치(예: /dev/sdb)에 대한 쓰기 액세스 권한이 부여되지 않아야 합니다. 파티션(예: /dev/sdb1) 또는 LVM 볼륨 사용.

전체 블록 장치를 게스트에 전달하면 게스트에서 파티션을 지정하거나 자체 LVM 그룹을 만들 수 있습니다. 이로 인해 호스트 물리적 시스템에서 이러한 파티션 또는 LVM 그룹을 감지하고 오류가 발생할 수 있습니다.

12.1.1. virsh를 사용하여 디스크 기반 스토리지 풀 생성

이 절차에서는 virsh 명령으로 디스크 장치를 사용하여 새 스토리지 풀을 생성합니다.

주의

스토리지 풀에 디스크를 전용으로 사용하면 디스크 장치에 현재 저장된 모든 데이터를 다시 포맷하고 삭제합니다. 다음 절차를 시작하기 전에 스토리지 장치를 백업하는 것이 좋습니다.

1. 디스크에 GPT 디스크 레이블 만들기

   디스크는 GUID 파티션 테이블 (GPT) 디스크 레이블을 사용하여 레이블을 다시 지정해야 합니다. GPT 디스크 레이블을 사용하면 각 장치에 최대 128개의 파티션을 만들 수 있습니다. GPT 파티션 테이블은 MS-DOS 파티션 테이블보다 훨씬 더 많은 파티션에 대한 파티션 데이터를 저장할 수 있습니다.

```
# parted /dev/sdb
GNU Parted 2.1
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
```

```
(parted) mklabel
New disk label type? gpt
(parted) quit
Information: You may need to update /etc/fstab.
#
```

2. *스토리지 풀 구성 파일 생성*

*새 장치에 필요한 스토리지 풀 정보를 포함하는 임시 **XML** 텍스트 파일을 생성합니다.*

파일은 아래 표시된 형식이어야 하며 다음 필드를 포함해야 합니다.

<name>guest_images_disk</name>

***name** 매개변수는 스토리지 풀의 이름을 결정합니다. 이 예에서는 아래 예제에서 **guest_images_disk** 라는 이름을 사용합니다.*

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

***path** 속성이 있는 **device** 매개변수는 스토리지 장치의 장치 경로를 지정합니다. 이 예에서는 **/dev/sdb** 장치를 사용합니다.*

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

경로 하위 매개 변수가 있는 파일 시스템 대상 매개 변수는 이 스토리지 풀을 사용하여 생성된 볼륨을 연결할 호스트 물리적 머신 파일 시스템의 위치를 결정합니다.

*예를 들면 **sdb1, sdb2, sdb3**입니다. 아래 예제와 같이 **/dev/** 를 사용하면 이 스토리지 풀에서 생성된 볼륨에 **/dev /sdb1, /dev/sdb2, / dev/sdb3**로 액세스할 수 있음을 의미합니다.*

<format type='gpt'/>

***format** 매개 변수는 파티션 테이블 유형을 지정합니다. 이 예에서는 아래 예제의 **gpt** 를 사용하여 이전 단계에서 만든 **GPT** 디스크 레이블 유형과 일치합니다.*

*텍스트 편집기를 사용하여 스토리지 풀 장치의 **XML** 파일을 생성합니다.*

*예 **12.2.** 디스크 기반 스토리지 장치 스토리지 풀*

```
<pool type='disk'>
  <name>guest_images_disk</name>
```

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

3. *장치 연결*

   *이전 단계에서 만든 **XML** 구성 파일과 함께 **virsh pool-define** 명령을 사용하여 스토리지 풀 정의를 추가합니다.*

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

4. *스토리지 풀 시작*

   ***virsh pool-start** 명령을 사용하여 스토리지 풀을 시작합니다. 풀이 **virsh pool-list --all** 명령으로 시작되었는지 확인합니다.*

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

5. *자동 시작 켜기*

   *스토리지 풀에 대해 **autostart** 를 켭니다. **autostart**는 서비스가 시작될 때 스토리지 풀을 시작하도록 **libvirtd** 서비스를 구성합니다.*

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

6. *스토리지 풀 구성 확인*

스토리지 풀이 올바르게 생성되었고 크기가 올바르게 보고되었는지, 상태 보고서가 실행 중으로 생성되었는지 확인합니다.

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

7. **선택 사항:** 임시 구성 파일을 제거합니다.

   필요하지 않은 경우 임시 스토리지 풀 **XML** 구성 파일을 제거합니다.

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

디스크 기반 스토리지 풀을 사용할 수 있습니다.

12.1.2. virsh를 사용하여 스토리지 풀 삭제

다음은 **virsh**를 사용하여 스토리지 풀을 삭제하는 방법을 보여줍니다.

1. 동일한 풀을 사용하는 다른 게스트 가상 머신의 문제를 방지하려면 스토리지 풀을 중지하고 사용 중인 리소스를 모두 해제하는 것이 가장 좋습니다.

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

2. 스토리지 풀의 정의 제거

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

12.2. 파티션 기반 스토리지 풀

이 섹션에서는 미리 포맷된 블록 장치인 파티션을 스토리지 풀로 사용하는 방법을 설명합니다.

*다음 예제에서 호스트 물리적 시스템에는 **500GB** 하드 드라이브(**/dev/sdc**)가 **500GB, ext4** 포맷 파티션(**/dev/sdc1**)으로 분할되어 있습니다. 아래 절차를 사용하여 스토리지 풀을 설정합니다.*

12.2.1. virt-manager를 사용하여 파티션 기반 스토리지 풀 생성

다음 절차에서는 스토리지 장치의 파티션을 사용하여 새 스토리지 풀을 생성합니다.

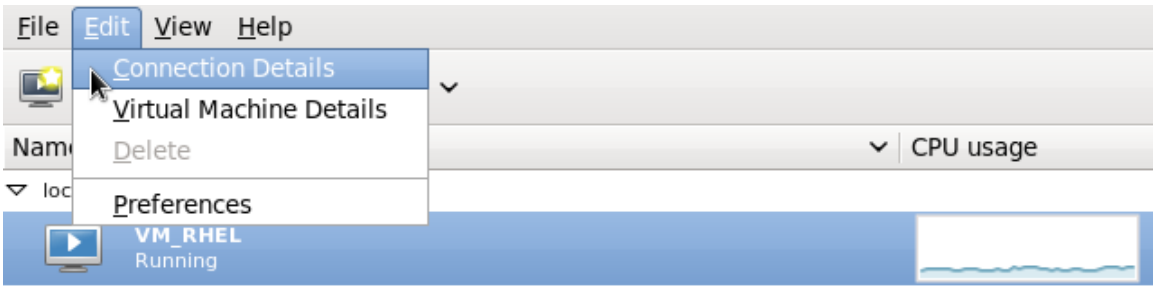
절차 12.1. virt-manager를 사용하여 파티션 기반 스토리지 풀 생성

1. *스토리지 풀 설정 열기*

   a. *virt-manager 그래픽 인터페이스의 기본 창에서 호스트 물리적 머신을 선택합니다.*

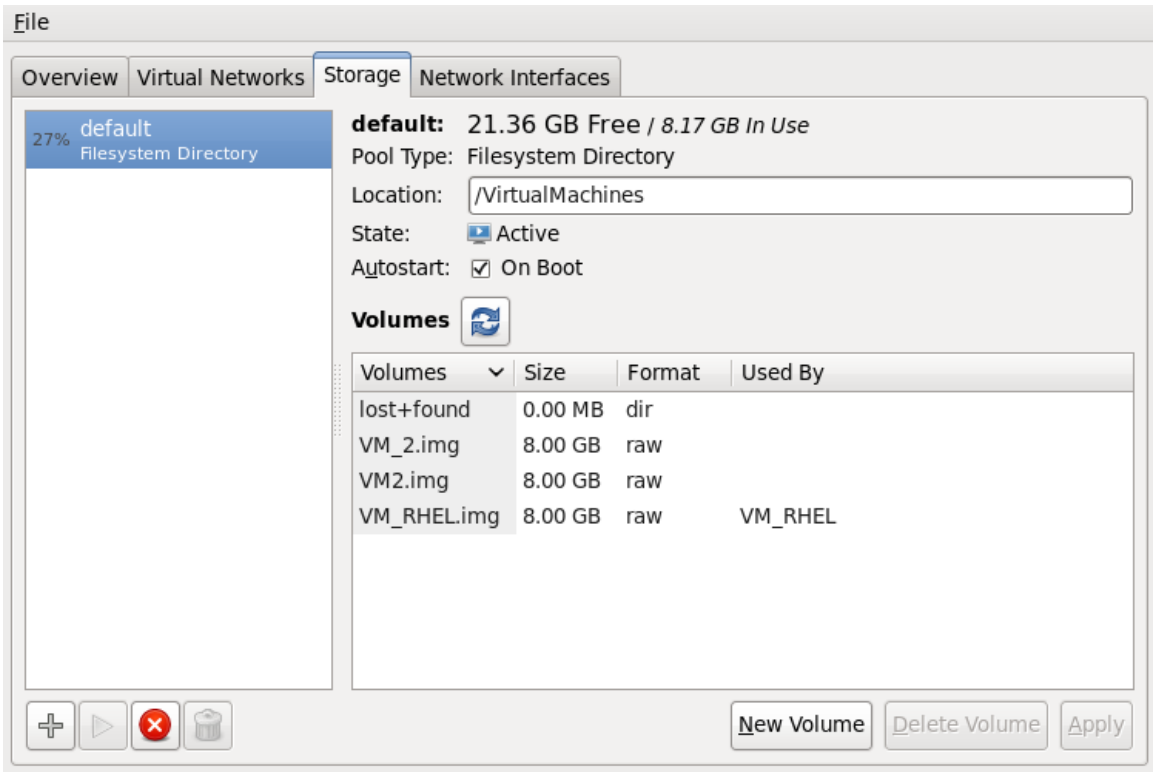
   편집 메뉴를 열고 연결 세부정보를 선택합니다.

   그림 12.1. 연결 세부 정보



   b. *연결 세부 정보 창의 스토리지 탭을 클릭합니다.*

그림 12.2. 스토리지 탭



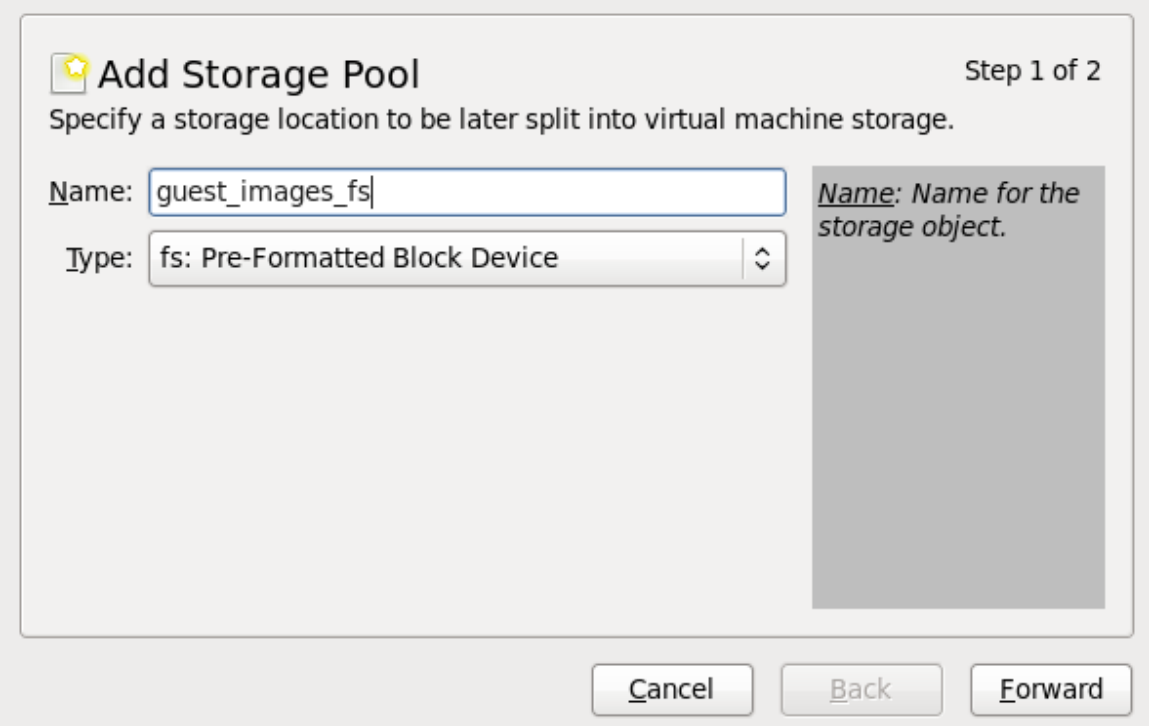
2. *새 스토리지 풀 생성*

   a. *새 풀 추가(1부)*

      *+ 버튼을 누릅니다(추가 풀 버튼). **Add a New Storage Pool** 마법사가 표시됩니다.*

      *스토리지 풀 의 이름을 선택합니다. 이 예에서는 이름이 **guest_images_fs** 를 사용합니다. 유형을 **fs**로 변경: 사전 포맷된 블록 장치.*

그림 12.3. 스토리지 풀 이름 및 유형

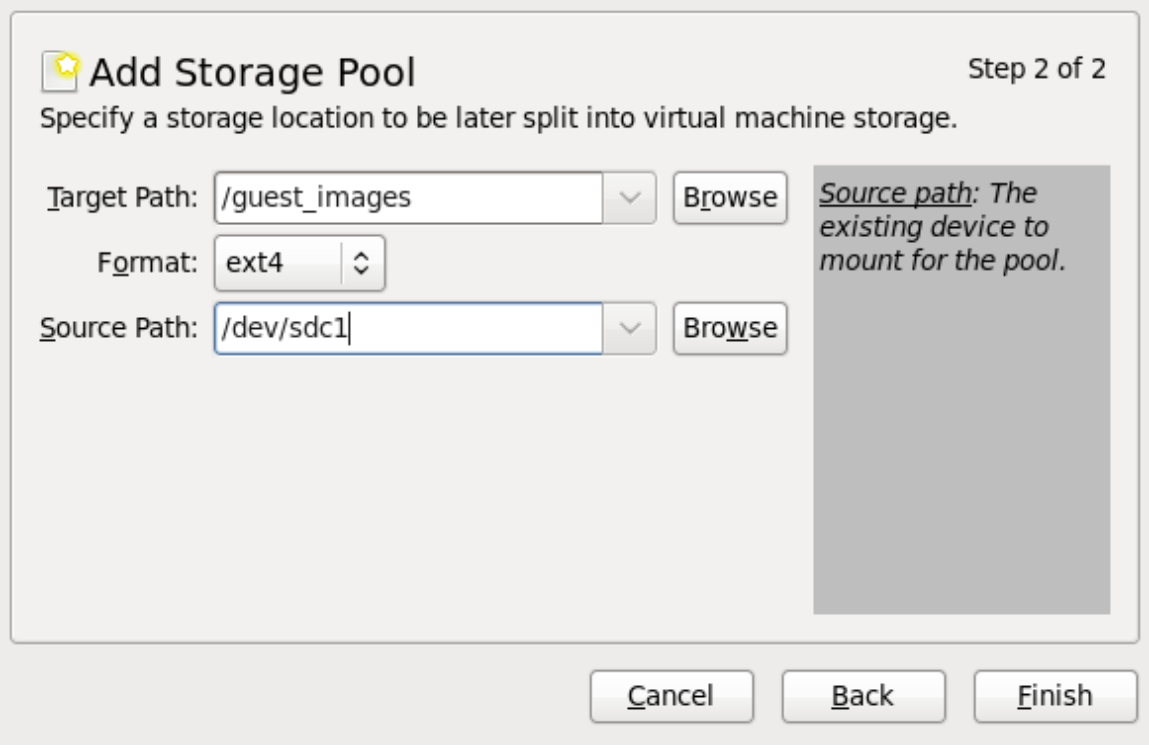


***Forward** 버튼을 눌러 계속합니다.*

b. *새 풀 추가(2부)*

대상 경로, 형식, 소스 경로 필드를 변경합니다.

그림 12.4. 스토리지 풀 경로 및 형식



대상 경로

*대상 경로 필드에 스토리지 풀의 소스 장치를 마운트할 위치를 입력합니다. 위치가 아직 없는 경우 **virt-manager** 는 디렉터리를 생성합니다.*

형식

*형식 목록에서 형식을 선택합니다.**Select a format from the Format list.** 선택한 형식으로 장치가 포맷됩니다.*

*이 예에서는 기본 **Red Hat Enterprise Linux** 파일 시스템인 **ext4** 파일 시스템을 사용합니다.*

소스 경로

소스 경로 필드에 장치를 입력합니다.

*이 예에서는 **/dev/sdc1** 장치를 사용합니다.*

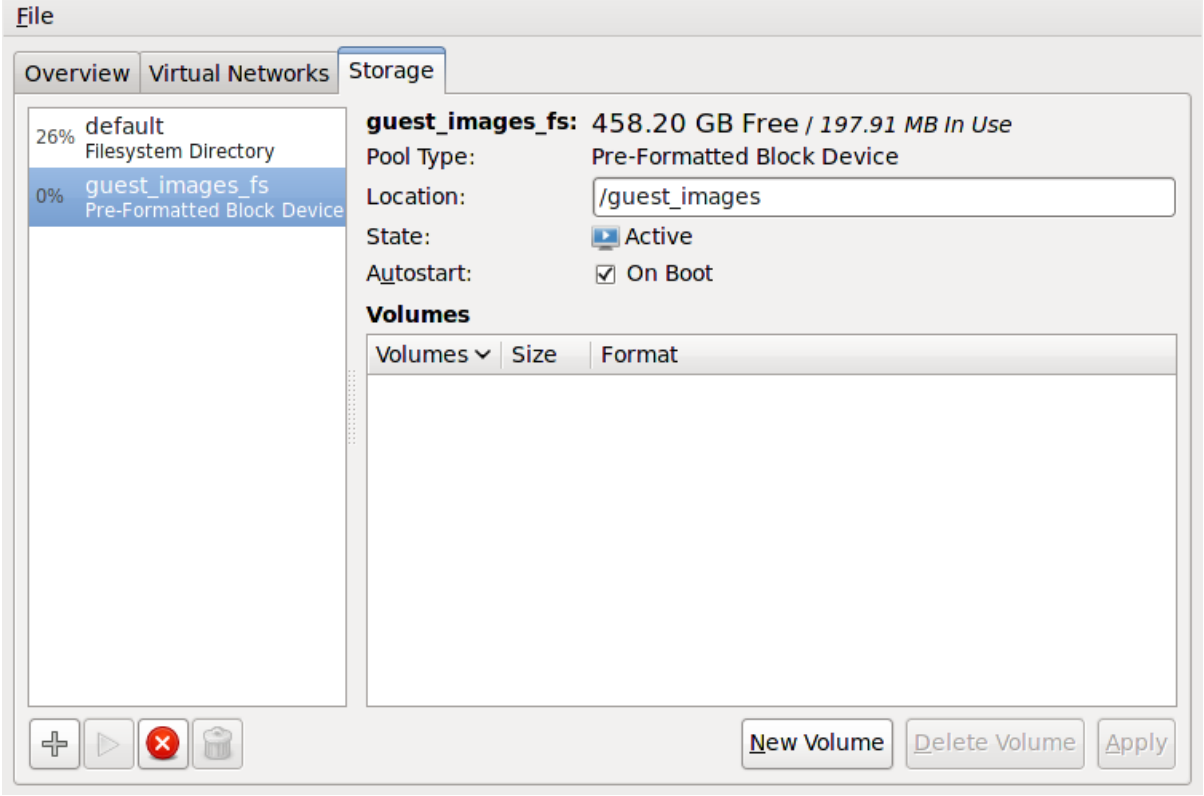
*세부 사항을 확인하고 **Finish** 버튼을 눌러 스토리지 풀을 생성합니다.*

3. *새 스토리지 풀 확인*

*새 스토리지 풀이 몇 초 후에 왼쪽의 스토리지 목록에 나타납니다. 이 예제에서는 크기가 예상대로 **458.20GB Free** 로 보고되었는지 확인합니다. **State** (상태) 필드가 새 스토리지 풀을 **Active** 로 보고하는지 확인합니다.*

*스토리지 풀을 선택합니다. **Autostart** (자동 시작) 필드에서 **On Boot** (부팅 시) 확인란을 클릭합니다. 이렇게 하면 **libvirtd** 서비스가 시작될 때마다 스토리지 장치가 시작됩니다.*

그림 12.5. 스토리지 목록 확인



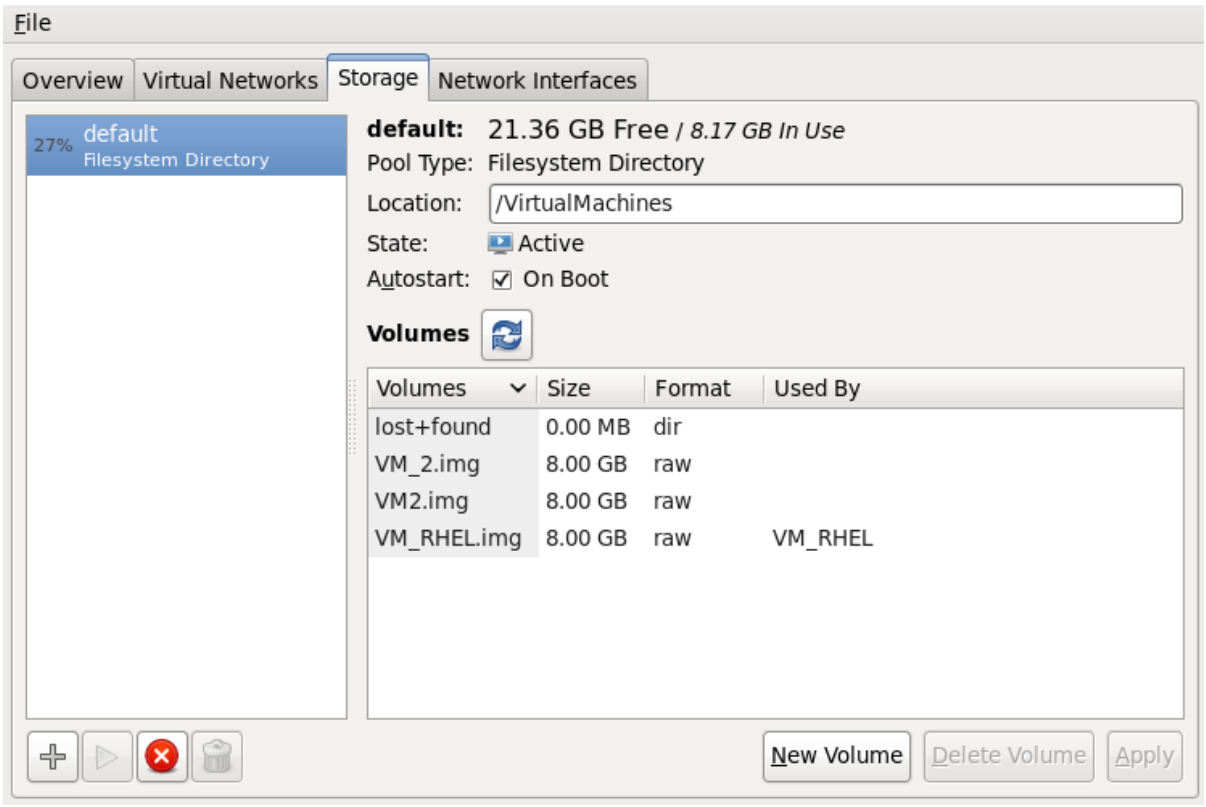
이제 스토리지 풀이 생성되고 연결 세부 정보 창을 닫습니다.

12.2.2. virt-manager를 사용하여 스토리지 풀 삭제

다음 절차에서는 스토리지 풀을 삭제하는 방법을 설명합니다.

1. *동일한 풀을 사용하는 다른 게스트 가상 머신의 문제를 방지하려면 스토리지 풀을 중지하고 사용 중인 리소스를 모두 해제하는 것이 가장 좋습니다. 이를 수행하려면 중지할 스토리지 풀을 선택하고 스토리지 창 하단에 있는 빨간색 **X** 아이콘을 클릭합니다.*

그림 12.6. 아이콘 중지



2. ***bin can** 아이콘을 클릭하여 스토리지 풀을 삭제합니다. 이 아이콘은 스토리지 풀을 먼저 중지한 경우에만 활성화됩니다.*

12.2.3. virsh를 사용하여 파티션 기반 스토리지 풀 생성

*이 섹션에서는 **virsh** 명령을 사용하여 파티션 기반 스토리지 풀을 생성하는 방법을 설명합니다.*

주의

*이 절차를 사용하여 전체 디스크를 스토리지 풀(예: **/dev/sdb**)으로 할당하지 마십시오. 게스트에 전체 디스크 또는 블록 장치에 대한 쓰기 액세스 권한이 부여되지 않아야 합니다. 파티션(예: **/dev/sdb1**)을 스토리지 풀에 할당하는 데 이 방법을 사용합니다.*

절차 12.2. virsh를 사용하여 사전 포맷된 블록 장치 스토리지 풀 생성

1. *스토리지 풀 정의 생성*

   ***virsh pool-define-as** 명령을 사용하여 새 스토리지 풀 정의를 생성합니다. 사전 포맷된 디스크를 스토리지 풀로 정의하려면 세 가지 옵션을 제공해야 합니다.*

파티션 이름

name 매개변수는 스토리지 풀의 이름을 결정합니다. 이 예에서는 아래 예제에서 **guest_images_fs** 라는 이름을 사용합니다.

device

path 속성이 있는 **device** 매개변수는 스토리지 장치의 장치 경로를 지정합니다. 이 예에서는 **/dev/sdc1** 파티션을 사용합니다.

mountpoint

포맷된 장치가 마운트 될 로컬 파일 시스템의 마운트 지점. 마운트 지점 디렉터리가 없는 경우 **virsh** 명령은 디렉터리를 생성할 수 있습니다.

이 예에서는 **/guest_images** 디렉터리가 사용됩니다.

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

이제 새 풀과 마운트 지점이 생성됩니다.

2. *새 풀 확인*

현재 스토리지 풀을 나열합니다.

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

3. *마운트 지점 만들기*

virsh pool-build 명령을 사용하여 사전 포맷된 파일 시스템 스토리지 풀에 대한 마운트 지점을 생성합니다.

```
# virsh pool-build guest_images_fs
Pool guest_images_fs built
# ls -la /guest_images
total 8
```

```
drwx------.  2 root root 4096 May 31 19:38 .
dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_fs      inactive   no
```

4. *스토리지 풀 시작*

   ***virsh pool-start** 명령을 사용하여 파일 시스템을 마운트 지점에 마운트하고 풀을 사용할 수 있도록 합니다.*

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

5. *자동 시작 켜기*

   *기본적으로 **virsh** 로 정의된 스토리지 풀은 **libvirtd** 가 시작될 때마다 자동으로 시작하도록 설정되지 않습니다. 이 문제를 해결하려면 **virsh pool-autostart** 명령으로 자동으로 시작합니다. 이제 **libvirtd** 가 시작될 때마다 스토리지 풀이 자동으로 시작됩니다.*

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

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

6. *스토리지 풀 확인*

   *스토리지 풀이 올바르게 생성되고 보고된 크기가 예상대로 있고 상태가 **running** 로 보고되었는지 확인합니다. 파일 시스템의 마운트 지점에 장치가 마운트되었음을 나타내는 **"lost+found"** 디렉터리가 있는지 확인합니다.*

```
# 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
```

12.2.4. virsh를 사용하여 스토리지 풀 삭제

1. 동일한 풀을 사용하는 다른 게스트 가상 머신의 문제를 방지하려면 스토리지 풀을 중지하고 사용 중인 리소스를 모두 해제하는 것이 가장 좋습니다.

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

2. 필요한 경우 스토리지 풀이 있는 디렉터리를 제거하려면 다음 명령을 사용합니다.

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

3. 스토리지 풀의 정의 제거

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

12.3. 디렉터리 기반 스토리지 풀

이 섹션에서는 호스트 물리적 시스템의 디렉터리에 게스트 가상 시스템을 저장하는 방법을 설명합니다.

디렉터리 기반 스토리지 풀은 **virt-manager** 또는 **virsh** 명령행 툴을 사용하여 생성할 수 있습니다.

12.3.1. virt-manager를 사용하여 디렉터리 기반 스토리지 풀 생성

1. 로컬 디렉터리 만들기

   a. 선택 사항: 스토리지 풀에 사용할 새 디렉터리 생성

      스토리지 풀의 호스트 물리적 시스템에 디렉터리를 생성합니다. 이 예에서는 **/guest** 가상 **machine_images** 라는 디렉터리를 사용합니다.

```
# mkdir /guest_images
```

b. *디렉터리 소유권 설정*

*디렉터리의 사용자 및 그룹 소유권을 변경합니다. 디렉터리는 **root** 사용자가 소유해야 합니다.*

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

c. *디렉터리 권한 설정*

디렉터리의 파일 권한을 변경합니다.

```
# chmod 700 /guest_images
```

d. *변경 사항 확인*

권한이 수정되었는지 확인합니다. 출력에 올바르게 구성된 빈 디렉터리가 표시됩니다.

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

2. ***SELinux** 파일 컨텍스트 구성*

*새 디렉터리에 올바른 **SELinux** 컨텍스트를 구성합니다. 풀 이름과 디렉터리의 이름은 일치할 필요가 없습니다. 그러나 게스트 가상 머신을 종료하는 경우 **libvirt**는 컨텍스트를 기본값으로 다시 설정해야 합니다. 디렉터리의 컨텍스트에 따라 이 기본값이 결정됩니다. 게스트 가상 시스템이 종료될 때 이미지에 **'virt_image_t'**라는 레이블이 지정되고 호스트 물리적 시스템에서 실행되는 다른 프로세스와 격리되도록 **virt_image_t** 디렉터리에 레이블을 지정할 수 있습니다.*

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

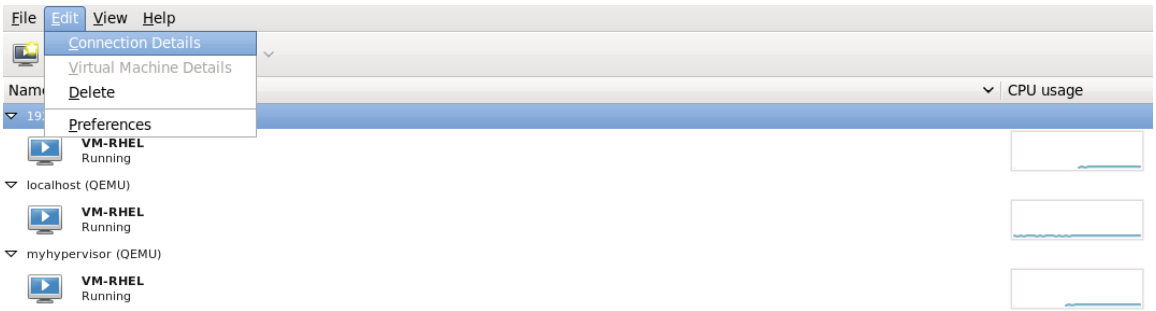
3. *스토리지 풀 설정 열기*

a.

***virt-manager** 그래픽 인터페이스의 기본 창에서 호스트 물리적 머신을 선택합니다.*

편집 메뉴를 열고 연결 세부정보를 선택합니다.

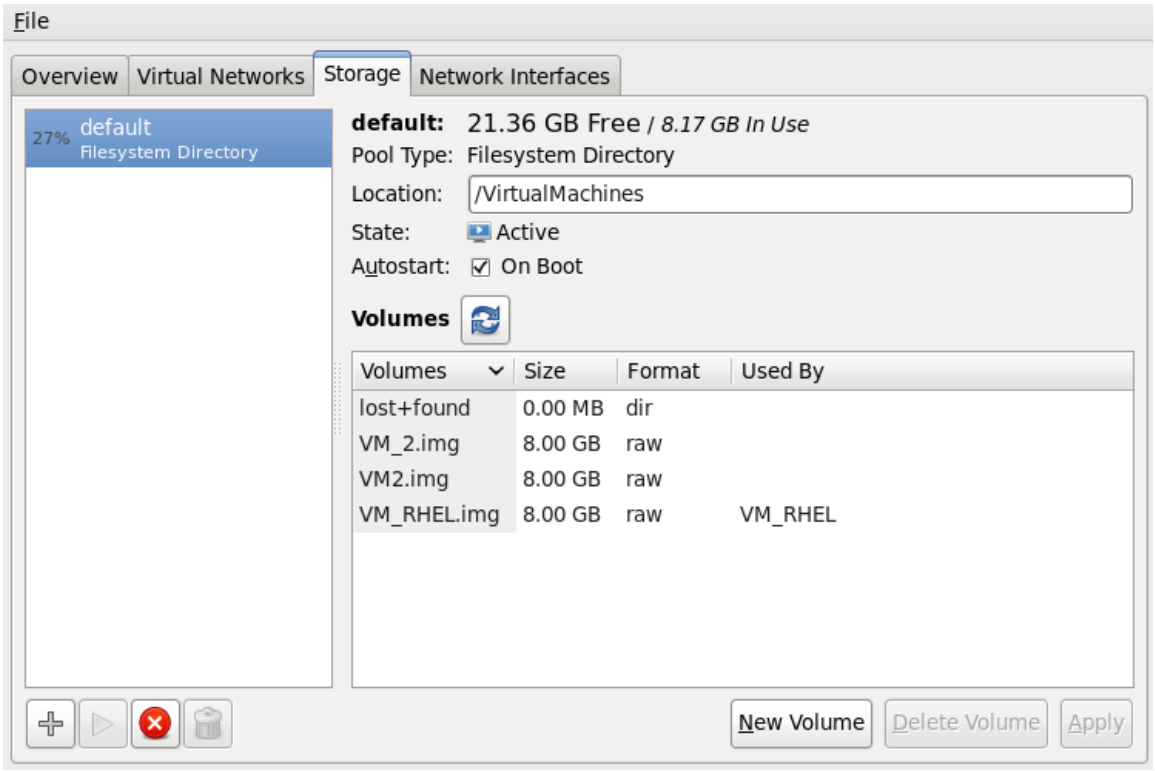
그림 12.7. 연결 세부 정보 창



b.

연결 세부 정보 창의 스토리지 탭을 클릭합니다.

그림 12.8. 스토리지 탭



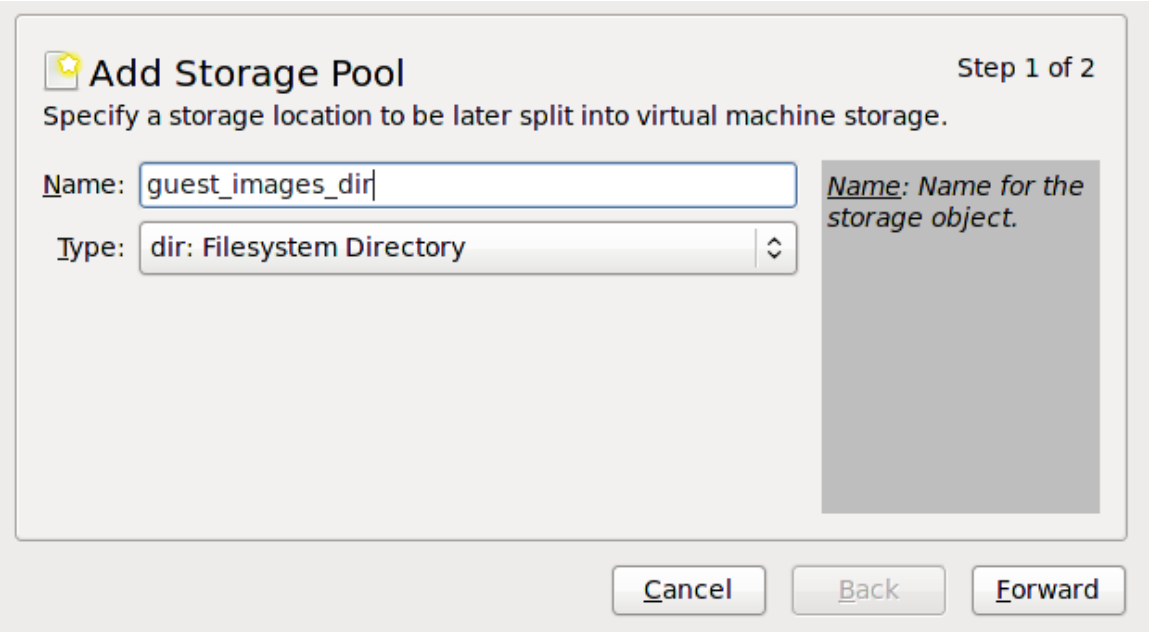
4. *새 스토리지 풀 생성*

a. *새 풀 추가(1부)*

*+ 버튼을 누릅니다(추가 풀 버튼). **Add a New Storage Pool** 마법사가 표시됩니다.*

*스토리지 풀 의 이름을 선택합니다. 이 예에서는 **guest_images** 라는 이름을 사용합니다. 유형을 **dir: Filesystem Directory** 로 변경합니다.*

그림 12.9. 스토리지 풀의 이름



***Forward** 버튼을 눌러 계속합니다.*

b. *새 풀 추가(2부)*

*대상 경로 필드를 변경합니다. 예를 들면 **/guest_images** 입니다.*

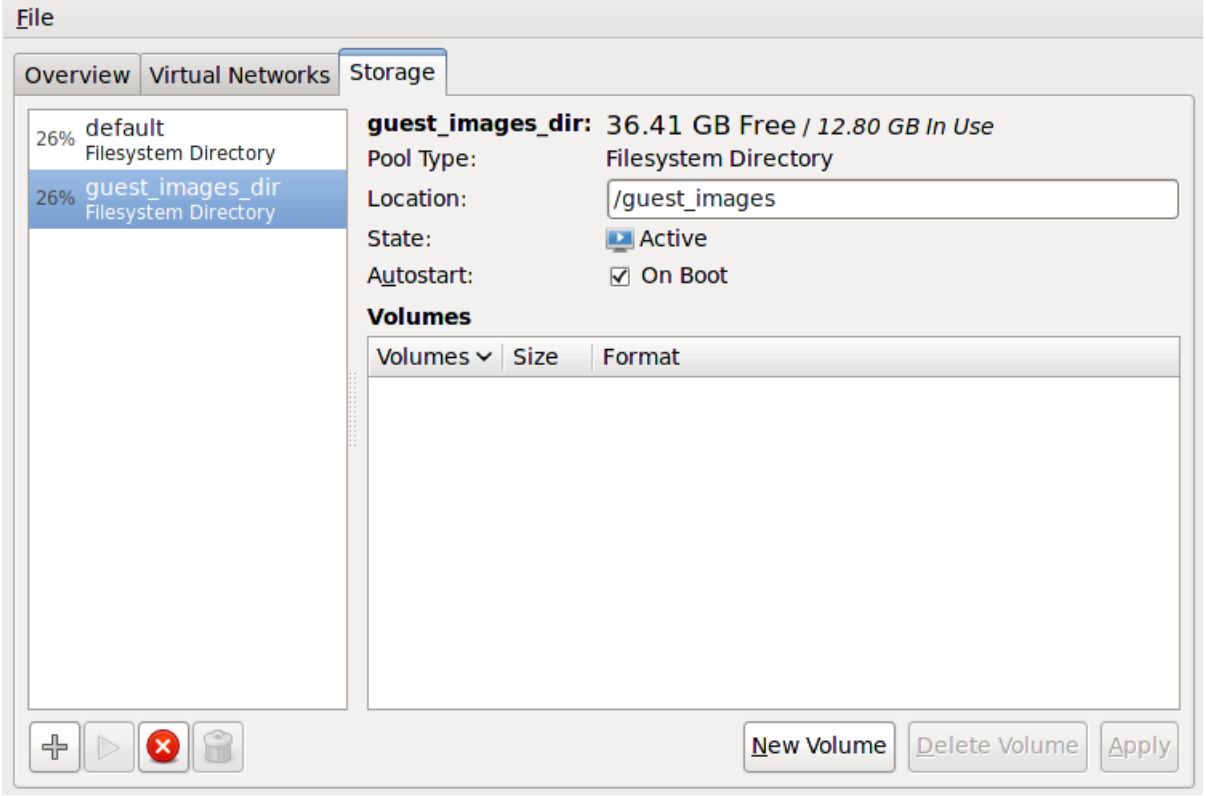
*세부 사항을 확인하고 **Finish** 버튼을 눌러 스토리지 풀을 생성합니다.*

5. *새 스토리지 풀 확인*

*새 스토리지 풀이 몇 초 후에 왼쪽의 스토리지 목록에 나타납니다. 이 예제에서는 크기가 예상대로 **36.41GB Free** 로 보고되었는지 확인합니다. **State** (상태) 필드가 새 스토리지 풀을 **Active** 로 보고하는지 확인합니다.*

*스토리지 풀을 선택합니다. **Autostart** (자동 시작) 필드에서 **On Boot** (부팅 시) 확인란이 선택되었는지 확인합니다. 이렇게 하면 **libvirtd** 서비스가 시작될 때마다 스토리지 풀이 시작됩니다.*

그림 12.10. 스토리지 풀 정보 확인



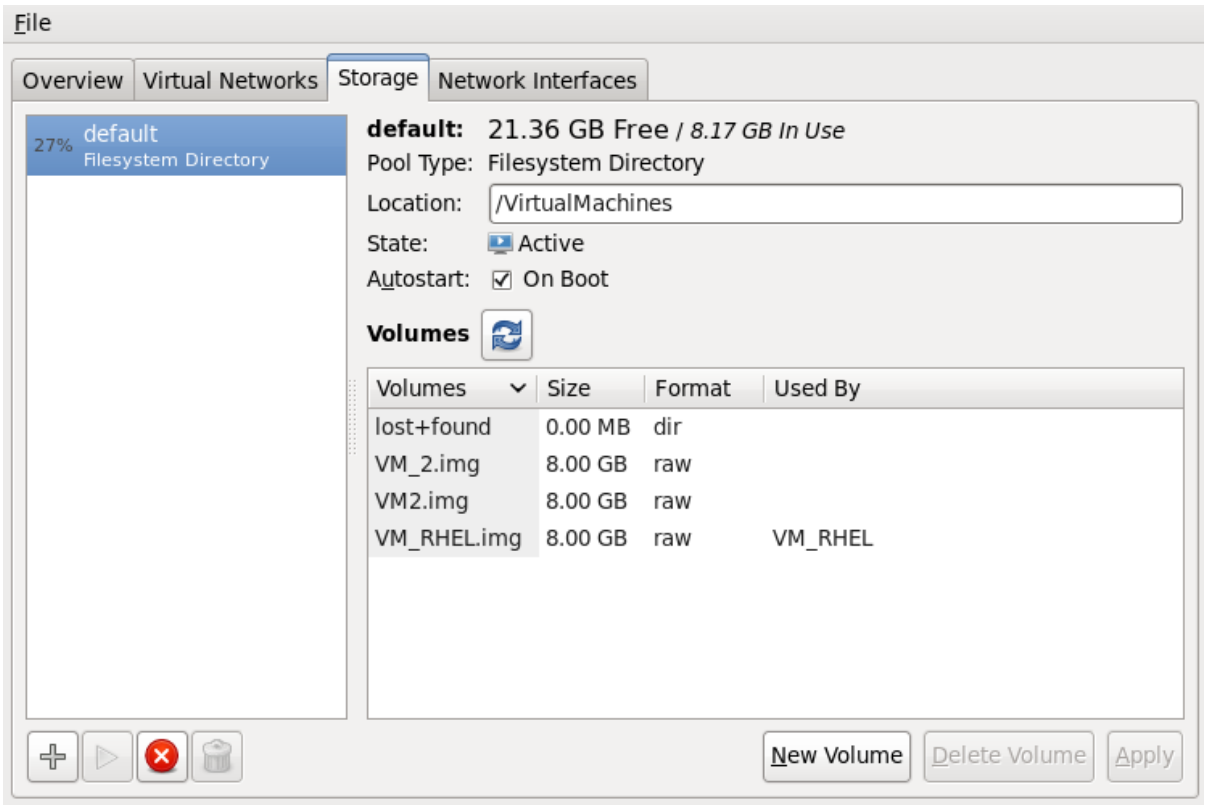
이제 스토리지 풀이 생성되고 연결 세부 정보 창을 닫습니다.

12.3.2. virt-manager를 사용하여 스토리지 풀 삭제

다음 절차에서는 스토리지 풀을 삭제하는 방법을 설명합니다.

1. *동일한 풀을 사용하는 다른 게스트 가상 머신의 문제를 방지하려면 스토리지 풀을 중지하고 사용 중인 리소스를 모두 해제하는 것이 가장 좋습니다. 이를 수행하려면 중지할 스토리지 풀을 선택하고 스토리지 창 하단에 있는 빨간색 **X** 아이콘을 클릭합니다.*

그림 12.11. 아이콘 중지



2.

***bin can** 아이콘을 클릭하여 스토리지 풀을 삭제합니다. 이 아이콘은 스토리지 풀을 먼저 중지한 경우에만 활성화됩니다.*

12.3.3. virsh를 사용하여 디렉터리 기반 스토리지 풀 생성

1. *스토리지 풀 정의 생성*

***virsh pool-define-as** 명령을 사용하여 새 스토리지 풀을 정의합니다. 디렉터리 기반 스토리지 풀을 생성하는 데 필요한 두 가지 옵션이 있습니다.*

- *스토리지 풀의 이름입니다.*

*이 예에서는 **guest_images** 라는 이름을 사용합니다. 이 예제에서 사용된 추가 **virsh** 명령은 모두 이 이름을 사용합니다.*

- *게스트 이미지 파일을 저장하기 위한 파일 시스템 디렉터리 경로 입니다. 이 디렉터리가 없으면 **virsh** 를 생성합니다.*

*이 예에서는 **/guest_images** 디렉터리를 사용합니다.*

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

2. 스토리지 풀이 나열되었는지 확인합니다.

   스토리지 풀 오브젝트가 올바르게 생성되고 상태가 비활성 으로 보고되는지 확인합니다.

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

3. 로컬 디렉터리 만들기

   다음과 같이 **virsh pool-build** 명령을 사용하여 **guest_images** 디렉터리의 디렉터리 기반 스토리지 풀을 빌드합니다.

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

4. 스토리지 풀 시작

   virsh 명령 **pool-start** 를 사용하여 디렉토리 스토리지 풀을 활성화하여 풀 볼륨을 게스트 디스크 이미지로 사용할 수 있도록 합니다.

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

5. 자동 시작 켜기

   스토리지 풀에 대해 **autostart** 를 켭니다. **autostart**는 서비스가 시작될 때 스토리지 풀을 시작하도록 **libvirtd** 서비스를 구성합니다.

```
# virsh pool-autostart guest_images
```

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

6. *스토리지 풀 구성 확인*

   *스토리지 풀이 올바르게 생성되고 크기가 올바르게 보고되었으며 상태가 **running** 으로 보고되었는지 확인합니다. 게스트 가상 시스템이 실행 중이 아닌 경우에도 풀에 액세스할 수 있도록 하려면 **Persistent** 가 **yes** 로 보고되는지 확인합니다. 서비스가 시작될 때 풀이 자동으로 시작되도록 하려면 **Autostart** 가 **yes** 로 보고되는지 확인하십시오.*

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

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

디렉터리 기반 스토리지 풀을 사용할 수 있습니다.

12.3.4. virsh를 사용하여 스토리지 풀 삭제

*다음은 **virsh**를 사용하여 스토리지 풀을 삭제하는 방법을 보여줍니다.*

1. *동일한 풀을 사용하는 다른 게스트 가상 머신의 문제를 방지하려면 스토리지 풀을 중지하고 사용 중인 리소스를 모두 해제하는 것이 가장 좋습니다.*

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

2. *필요한 경우 스토리지 풀이 있는 디렉터리를 제거하려면 다음 명령을 사용합니다.*

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

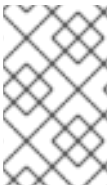
3. 스토리지 풀의 정의 제거

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

12.4. LVM 기반 스토리지 풀

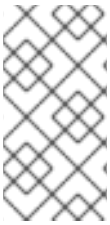
이 장에서는 LVM 볼륨 그룹을 스토리지 풀로 사용하는 방법을 설명합니다.

LVM 기반 스토리지 그룹은 LVM의 모든 유연성을 제공합니다.



참고

현재 LVM 기반 스토리지 풀에서는 씬 프로비저닝을 사용할 수 없습니다.



참고

LVM에 대한 자세한 내용은 Red Hat Enterprise Linux Storage 관리 가이드 를 참조하십시오.

주의

LVM 기반 스토리지 풀에는 전체 디스크 파티션이 필요합니다. 이러한 절차를 통해 새 파티션/장치를 활성화하면 파티션이 포맷되고 모든 데이터가 삭제됩니다. 호스트의 기존 볼륨 그룹(VG)을 사용하는 경우 아무것도 삭제되지 않습니다. 다음 절차를 시작하기 전에 스토리지 장치를 백업하는 것이 좋습니다.

12.4.1. virt-manager를 사용하여 LVM 기반 스토리지 풀 생성

LVM 기반 스토리지 풀은 기존 LVM 볼륨 그룹을 사용하거나 빈 파티션에 새 LVM 볼륨 그룹을 생성할 수 있습니다.

1. 선택 사항: LVM 볼륨의 새 파티션 만들기

   다음 단계에서는 새 하드 디스크 드라이브에 새 파티션 및 LVM 볼륨 그룹을 만드는 방법을 설명합니다.

주의

이 절차에서는 선택한 저장 장치에서 모든 데이터를 제거합니다.

a. *새 파티션 만들기*

fdisk 명령을 사용하여 명령줄에서 새 디스크 파티션을 만듭니다. 다음 예제에서는 저장 장치 /dev/sdb 에서 전체 디스크를 사용하는 새 파티션을 생성합니다.

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

새 파티션은 n 을 누릅니다.

b.

기본 파티션은 p 를 누릅니다.

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

c.

사용 가능한 파티션 번호를 선택합니다. 이 예에서는 1 을 입력하여 첫 번째 파티션이 선택됩니다.

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

d.

Enter 키를 눌러 기본 첫 번째 실린더를 입력합니다.

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

e.

파티션의 크기를 선택합니다. 이 예에서는 Enter 를 눌러 전체 디스크가 할당됩니다.

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

f.

t 를 눌러 파티션 유형을 설정합니다.

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

g.

이전 단계에서 만든 파티션을 선택합니다. 이 예에서 파티션 번호는 **1** 입니다.

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

h.

Linux LVM 파티션에 **8e** 를 입력합니다.

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

i.

디스크에 변경 사항을 쓰고 종료합니다.

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

j. 새 **LVM** 볼륨 그룹 만들기

vgcreate 명령을 사용하여 새 **LVM** 볼륨 그룹을 만듭니다. 이 예제에서는 **guest_images_lvm** 이라는 볼륨 그룹을 생성합니다.

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

이제 새 **LVM** 볼륨 그룹 **guest_images_lvm** 을 **LVM** 기반 스토리지 풀에 사용할 수 있습니다.

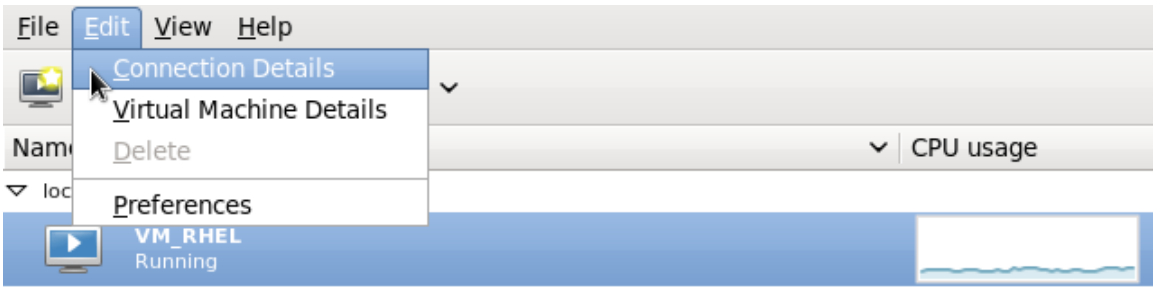
2. 스토리지 풀 설정 열기

a.

virt-manager 그래픽 인터페이스의 기본 창에서 호스트를 선택합니다.

편집 메뉴를 열고 연결 세부정보를 선택합니다.

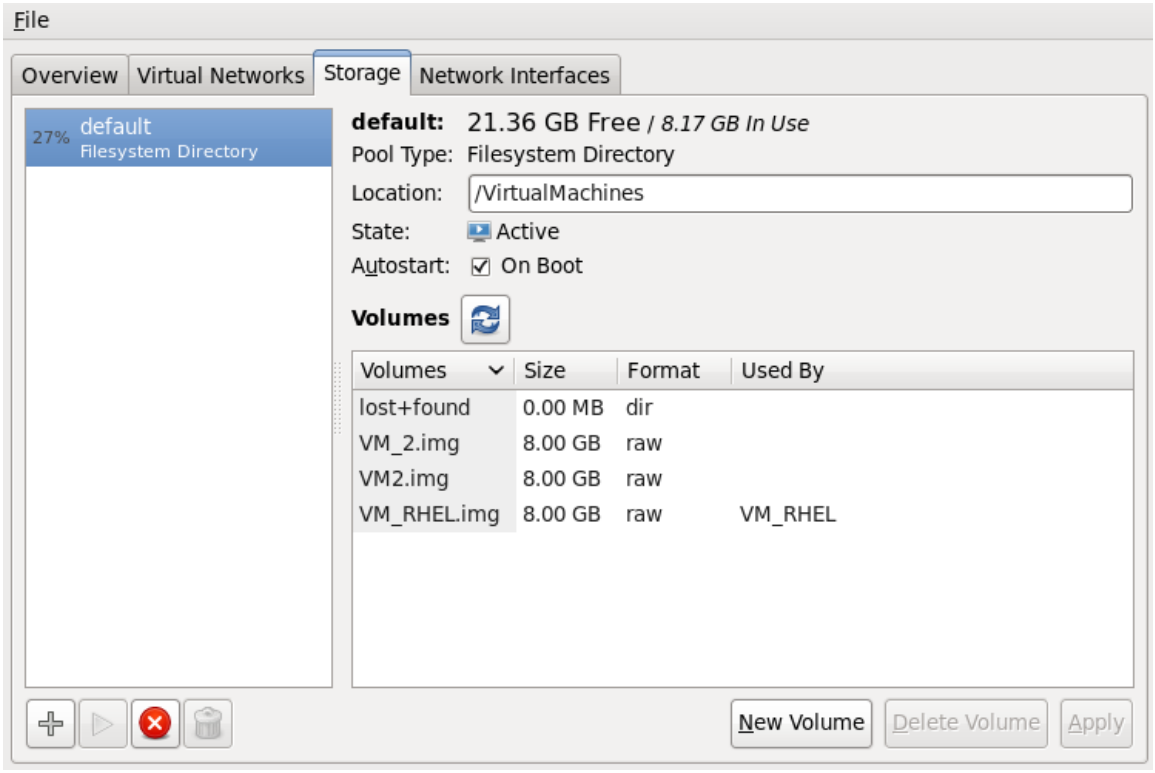
그림 12.12. 연결 세부 정보



b.

스토리지 탭을 클릭합니다.

그림 12.13. 스토리지 탭



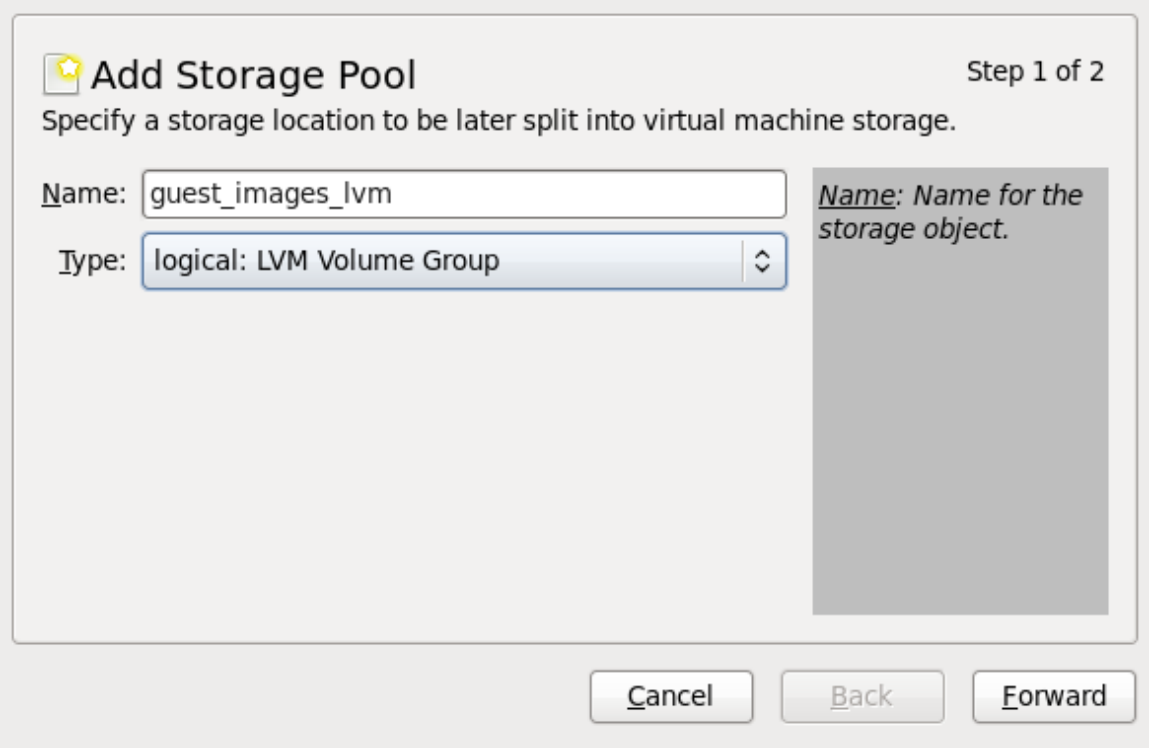
3. *새 스토리지 풀 생성*

a. *마법사 시작*

*+ 버튼을 누릅니다(추가 풀 버튼). **Add a New Storage Pool** 마법사가 표시됩니다.*

*스토리지 풀 의 이름을 선택합니다. 예에서는 **guest_images_lvm** 을 사용합니다. 그런 다음 유형을 논리로 변경합니다 . **LVM** 볼륨 그룹, 및*

그림 12.14. LVM 스토리지 풀 추가



***Forward** 버튼을 눌러 계속합니다.*

b. *새 풀 추가(2부)*

*대상 경로 필드를 변경합니다. 이 예에서는 **/guest_images** 를 사용합니다.*

대상 경로 및 소스 경로 필드를 입력한 다음 빌드 풀 확인란을 선택합니다.

- *대상 경로 필드를 사용하여 기존 **LVM** 볼륨 그룹을 선택하거나 새 볼륨 그룹의 이름으로 선택합니다. 기본 형식은 **/dev/storage_pool_name** 입니다.*

  *이 예에서는 **/dev/guest_images_lvm** 이라는 새 볼륨 그룹을 사용합니다.*

- *기존 **LVM** 볼륨 그룹이 대상 경로에 사용되는 경우 소스 경로 필드는 선택 사항입니다.*

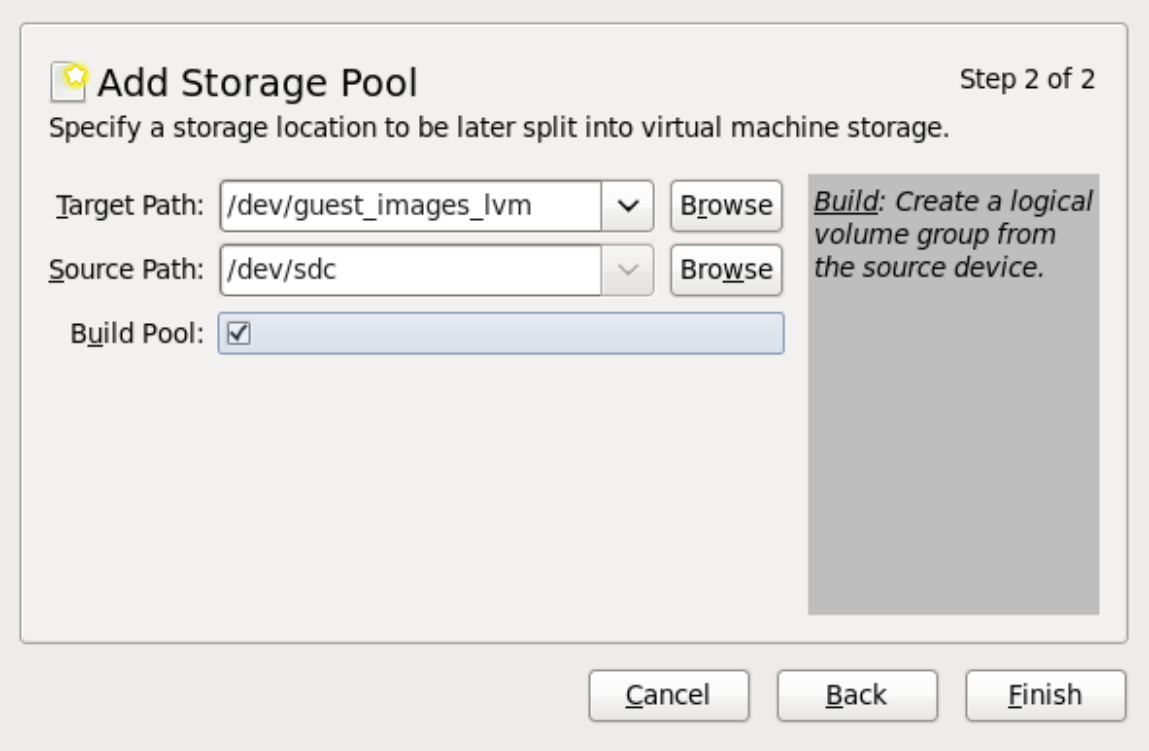
  *새 **LVM** 볼륨 그룹의 경우 소스 경로 필드에 스토리지 장치의 위치를 입력합니다. 이 예제에서는 빈 파티션 **/dev/sdc** 를 사용합니다.*

-

Build Pool (빌드 풀) 확인란을 통해 virt-manager 에 새 LVM 볼륨 그룹을 만들도록 지시합니다. 기존 볼륨 그룹을 사용하는 경우 빌드 풀 확인란을 선택하지 않아야 합니다.

이 예에서는 빈 파티션을 사용하여 새 볼륨 그룹을 생성하므로 Build Pool 확인란을 선택해야 합니다.

그림 12.15. 대상 및 소스 추가

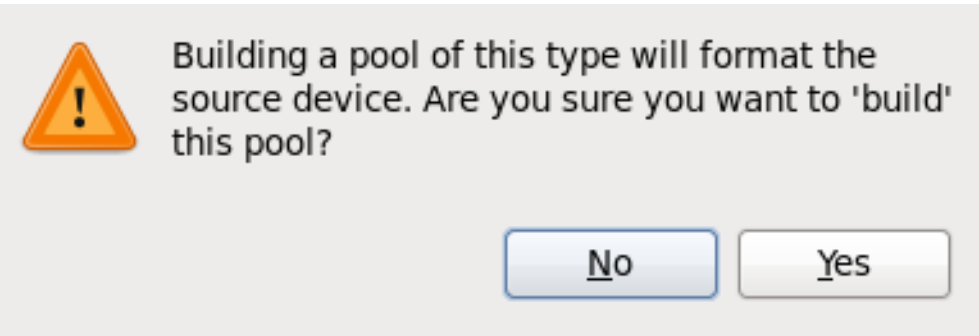


세부 사항을 확인하고 Finish 버튼 포맷을 LVM 볼륨 그룹을 선택하고 스토리지 풀을 만듭니다.

c. *포맷할 장치 확인*

경고 메시지가 나타납니다.

그림 12.16. 경고 메시지



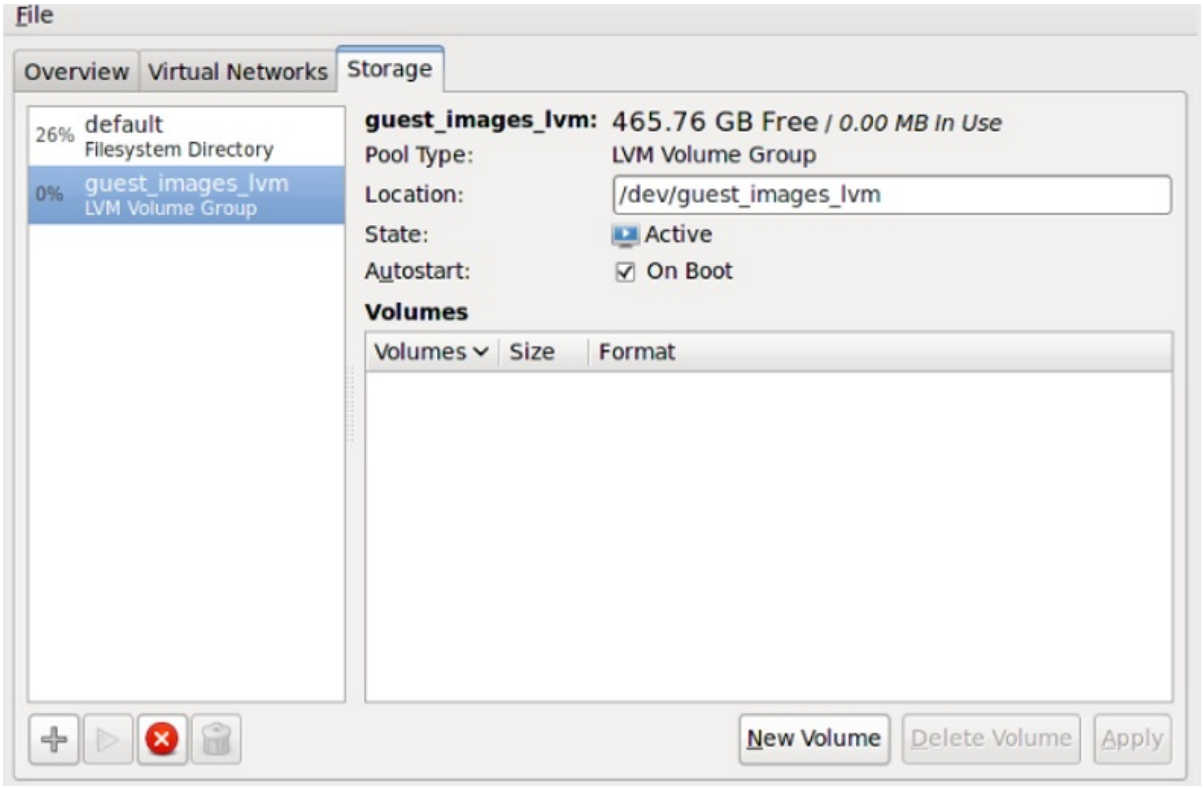
예 버튼을 눌러 저장 장치의 모든 데이터를 지우고 스토리지 풀을 생성합니다.

4. *새 스토리지 풀 확인*

*새 스토리지 풀이 몇 초 후에 왼쪽 목록에 나타납니다. 예제에서 세부 정보, **465.76GB**의 여유가 있는지 확인하십시오. 또한 **State** (상태) 필드가 새 스토리지 풀을 **Active** 로 보고하는지 확인합니다.*

*일반적으로 **Autostart** 확인란을 활성화하여 **libvirtd**에서 스토리지 풀이 자동으로 시작되도록 하는 것이 좋습니다.*

*그림 **12.17. LVM** 스토리지 풀 세부 정보 확인*



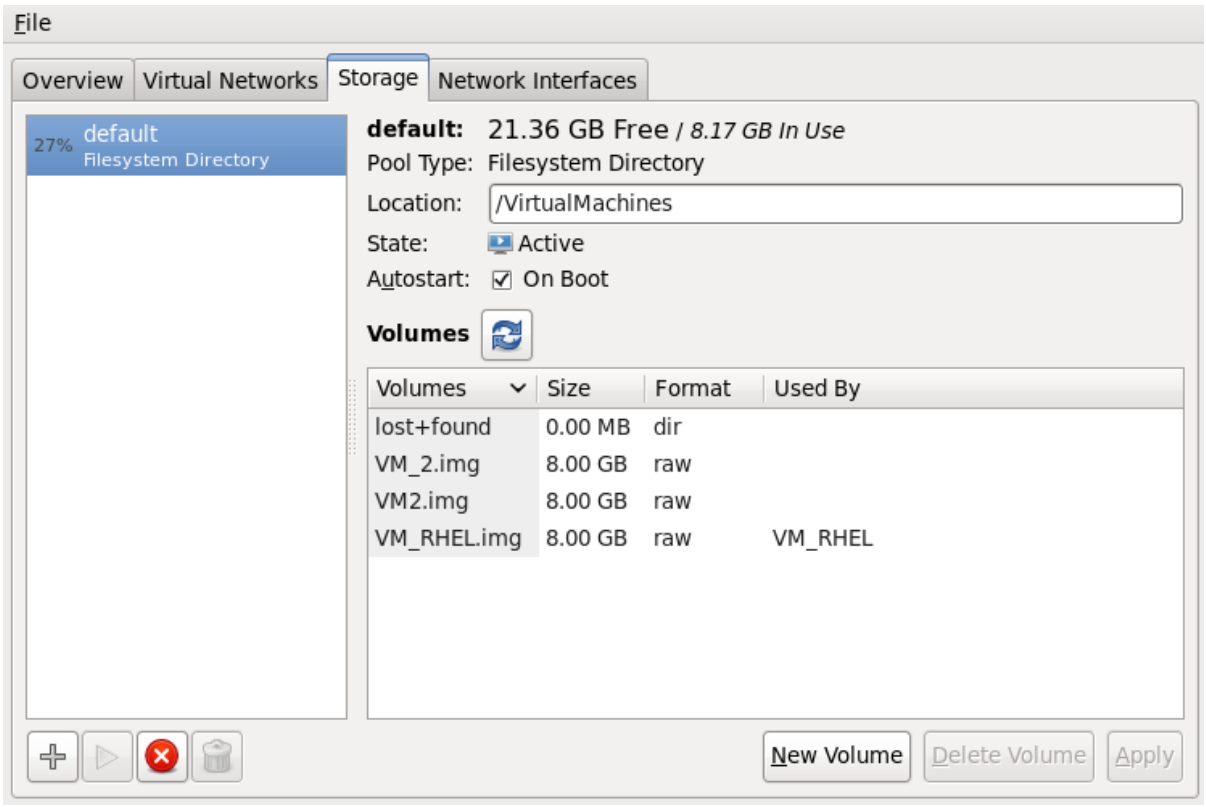
작업이 완료되면 호스트 세부 정보 대화 상자를 닫습니다.

12.4.2. virt-manager를 사용하여 스토리지 풀 삭제

다음 절차에서는 스토리지 풀을 삭제하는 방법을 설명합니다.

1. *동일한 풀을 사용하는 다른 게스트 가상 머신의 문제를 방지하려면 스토리지 풀을 중지하고 사용 중인 리소스를 모두 해제하는 것이 가장 좋습니다. 이를 수행하려면 중지할 스토리지 풀을 선택하고 스토리지 창 하단에 있는 빨간색 **X** 아이콘을 클릭합니다.*

그림 12.18. 아이콘 중지



2. ***bin can** 아이콘을 클릭하여 스토리지 풀을 삭제합니다. 이 아이콘은 스토리지 풀을 먼저 중지한 경우에만 활성화됩니다.*

12.4.3. virsh를 사용하여 LVM 기반 스토리지 풀 생성

*이 섹션에서는 **virsh** 명령을 사용하여 **LVM** 기반 스토리지 풀을 생성하는 데 필요한 단계에 대해 간단히 설명합니다. 단일 드라이브(**/dev/sdc**)에서 **guest_images_lvm** 이라는 풀의 예를 사용합니다. 이는 예제일 뿐이며 설정이 적절하게 대체되어야 합니다.*

절차 12.3. virsh를 사용하여 LVM 기반 스토리지 풀 생성

1. *풀 이름 **guest_images_lvm** 을 정의합니다.*

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

2. *지정된 이름에 따라 풀을 빌드합니다. 이미 기존 볼륨 그룹을 사용하는 경우 이 단계를 건너뜁니다.*

```
# virsh pool-build guest_images_lvm

Pool guest_images_lvm built
```

3. *새 풀을 초기화합니다.*

```
# virsh pool-start guest_images_lvm

Pool guest_images_lvm started
```

4. ***vgs** 명령을 사용하여 볼륨 그룹 정보를 표시합니다.*

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

5. *자동으로 시작하도록 풀을 설정합니다.*

```
# virsh pool-autostart guest_images_lvm
Pool guest_images_lvm marked as autostarted
```

6. ***virsh** 명령을 사용하여 사용 가능한 풀을 나열합니다.*

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

7. *다음 명령은 이 풀 내에서 **3**개의 볼륨(**volume1**,**volume2** 및 **volume3**)을 생성하는 방법을 보여줍니다.*

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

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

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

8. *이 풀에서 사용 가능한 볼륨을 **virsh** 명령과 함께 나열합니다.*

```
# virsh vol-list guest_images_lvm
```

```
Name                 Path
-----------------------------------------
volume1              /dev/libvirt_lvm/volume1
volume2              /dev/libvirt_lvm/volume2
volume3              /dev/libvirt_lvm/volume3
```

9. 다음 두 명령(**lvscan and lvs**)은 새로 생성된 볼륨에 대한 추가 정보를 표시합니다.

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

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

12.4.4. virsh를 사용하여 스토리지 풀 삭제

다음은 **virsh**를 사용하여 스토리지 풀을 삭제하는 방법을 보여줍니다.

1. 동일한 풀을 사용하는 다른 게스트의 문제를 방지하려면 스토리지 풀을 중지하고 사용하는 리소스를 모두 해제하는 것이 가장 좋습니다.

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

2. 필요한 경우 스토리지 풀이 있는 디렉터리를 제거하려면 다음 명령을 사용합니다.

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

3. 스토리지 풀의 정의 제거

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

12.5. ISCSI 기반 스토리지 풀

이 섹션에서는 게스트 가상 머신을 저장하기 위해 **iSCSI** 기반 장치를 사용하는 방법을 설명합니다.

iSCSI(Internet Small Computer System Interface)는 스토리지 장치를 공유하는 네트워크 프로토콜입니다. **iSCSI**는 **IP** 계층에 대한 **SCSI** 지침을 사용하여 이니시에이터(스토리지 클라이언트)를 대상으로 지정합니다.

12.5.1. 소프트웨어 iSCSI 대상 구성

scsi-target-utils 패키지는 소프트웨어 지원 **iSCSI** 대상을 생성하는 툴을 제공합니다.

절차 **12.4. iSCSI** 대상 생성

1. 필수 패키지 설치

   scsi-target-utils 패키지 및 모든 종속 항목 설치

```
# yum install scsi-target-utils
```

2. **tgtd** 서비스 시작

   tgtd 서비스는 물리적 시스템 **SCSI** 대상을 호스팅하고 **iSCSI** 프로토콜을 사용하여 물리적 머신 대상을 호스팅합니다. **tgtd** 서비스를 시작하고 **chkconfig** 명령을 사용하여 다시 시작한 후 서비스를 영구적으로 만듭니다.

```
# service tgtd start
# chkconfig tgtd on
```

3. 선택 사항: **LVM** 볼륨 생성

   LVM 볼륨은 **iSCSI** 백업 이미지에 유용합니다. **LVM** 스냅샷과 크기 조정은 게스트 가상 머신에 유용할 수 있습니다. 이 예제에서는 **iSCSI**를 사용하여 게스트 가상 머신을 호스트하기 위해 **RAID5** 어레이의 **virtstore** 라는 새 볼륨 그룹에 **virtimage1** 이라는 **LVM** 이미지를 생성합니다.

   a. **RAID** 배열 생성

      소프트웨어 **RAID5** 어레이 생성은 **Red Hat Enterprise Linux** 배포 가이드에서 다룹니다.

   b. **LVM** 볼륨 그룹 만들기

      vgcreate 명령을 사용하여 **virtstore** 라는 볼륨 그룹을 만듭니다.

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

   c. **LVM** 논리 볼륨 생성

lvcreate 명령을 사용하여 크기가 20GB인 virtstore 볼륨 그룹에 virtimage1 이라는 논리 볼륨 그룹을 생성합니다.

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

새로운 논리 볼륨 virtimage1 은 iSCSI에 사용할 준비가 되었습니다.

4. *선택 사항: 파일 기반 이미지 생성*

파일 기반 스토리지는 테스트용으로 충분하지만 프로덕션 환경 또는 중요한 I/O 활동에는 권장되지 않습니다. 이 선택적 절차에서는 iSCSI 대상에 대해 virtimage2.img 라는 파일 기반 이미지를 생성합니다.

a. *이미지에 대한 새 디렉터리 만들기*

이미지를 저장할 새 디렉터리를 만듭니다. 디렉터리에 올바른 SELinux 컨텍스트가 있어야 합니다.

```
# mkdir -p /var/lib/tgtd/virtualization
```

b. *이미지 파일 생성*

크기가 10GB인 virtimage2.img 라는 이미지를 만듭니다.

```
# dd if=/dev/zero of=/var/lib/tgtd/virtualization/virtimage2.img bs=1M seek=10000 count=0
```

c. *SELinux 파일 컨텍스트 구성*

새 이미지 및 디렉터리에 대해 올바른 SELinux 컨텍스트를 구성합니다.

```
# restorecon -R /var/lib/tgtd
```

새로운 파일 기반 이미지인 virtimage2.img 는 iSCSI에 사용할 수 있습니다.

5. *대상 생성*

/etc/tgt/targets.conf 파일에 XML 항목을 추가하여 대상을 생성할 수 있습니다. target 속성에는 IQN(iSCSI Qualified Name)이 필요합니다. IQN의 형식은 다음과 같습니다.

```
iqn.yyyy-mm.reversed domain name:optional identifier text
```

다음과 같습니다.

- ***YYYY-mm** 는 장치가 시작된 연도 및 월 (예: **2010-05**)을 나타냅니다.*

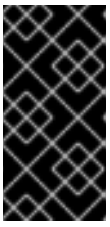
- *역방향 도메인 이름은 반대로 호스트 물리적 시스템 도메인 이름입니다(예: **IQN**의 **server1.example.com** 은 **com.example.server1**).*

- *선택적 식별자 텍스트는 관리자가 장치 또는 하드웨어를 식별하는 데 도움이 되는 공백이 없는 텍스트 문자열입니다.*

*이 예제에서는 선택적 식별자 평가판 으로 **server1.example.com** 의 선택적 단계에서 생성된 두 가지 유형의 이미지에 대해 **iSCSI** 대상을 생성합니다. 다음을 **/etc/tgt/targets.conf** 파일에 추가합니다.*

```
<target iqn.2010-05.com.example.server1:iscsirhel6guest>
  backing-store /dev/virtstore/virtimage1  #LUN 1
  backing-store /var/lib/tgtd/virtualization/virtimage2.img  #LUN 2
  write-cache off
</target>
```

*드라이버 유형을 **iSCSI**로 설정하는 기본**-driver iscsi** 줄이 **/etc/tgt/targets.conf** 파일에 포함되어 있는지 확인합니다. 드라이버는 기본적으로 **iSCSI**를 사용합니다.*



중요

*이 예제에서는 액세스 제어 없이 전역에서 액세스할 수 있는 대상을 생성합니다. 보안 액세스 구현에 대한 정보는 **scsi-target-utils**를 참조하십시오.*

6. ***tgtd** 서비스 다시 시작*

   ***tgtd** 서비스를 다시 시작하여 구성 변경 사항을 다시 로드합니다.*

```
# service tgtd restart
```

7. ***iptables** 구성*

   ***iptables** 를 사용하여 **iSCSI** 액세스를 위해 포트 **3260**을 엽니다.*

```
# iptables -I INPUT -p tcp -m tcp --dport 3260 -j ACCEPT
# service iptables save
# service iptables restart
```

8. *새 대상 확인*

   ***tgt-admin --show** 명령을 사용하여 설정이 성공했는지 확인하려면 새 대상을 확인합니다.*

```
# tgt-admin --show
Target 1: iqn.2010-05.com.example.server1:iscsirhel6guest
System information:
Driver: iscsi
State: ready
I_T nexus information:
LUN information:
LUN: 0
    Type: controller
    SCSI ID: IET     00010000
    SCSI SN: beaf10
    Size: 0 MB
    Online: Yes
    Removable media: No
    Backing store type: rdwr
    Backing store path: None
LUN: 1
    Type: disk
    SCSI ID: IET     00010001
    SCSI SN: beaf11
    Size: 20000 MB
    Online: Yes
    Removable media: No
    Backing store type: rdwr
    Backing store path: /dev/virtstore/virtimage1
LUN: 2
    Type: disk
    SCSI ID: IET     00010002
    SCSI SN: beaf12
    Size: 10000 MB
    Online: Yes
    Removable media: No
    Backing store type: rdwr
    Backing store path: /var/lib/tgtd/virtualization/virtimage2.img
Account information:
ACL information:
ALL
```

주의

***ACL** 목록은 **all**로 설정됩니다. 이렇게 하면 로컬 네트워크의 모든 시스템이 이 장치에 액세스할 수 있습니다. 프로덕션 환경에 대해 호스트 물리적 시스템 액세스 **ACL**을 설정하는 것이 좋습니다.*

9. *선택 사항: 테스트 검색*

   *새 **iSCSI** 장치를 검색할 수 있는지 테스트합니다.*

```
# iscsiadm --mode discovery --type sendtargets --portal server1.example.com
127.0.0.1:3260,1 iqn.2010-05.com.example.server1:iscsirhel6guest
```

10. *선택 사항: 장치 연결 테스트*

    *새 장치(**iqn.2010-05.com.example.server1:iscsirhel6guest**)를 연결하여 장치를 연결할 수 있는지 확인합니다.*

```
# iscsiadm -d2 -m node --login
scsiadm: Max file limits 1024 1024

Logging in to [iface: default, target: iqn.2010-05.com.example.server1:iscsirhel6guest, portal:
10.0.0.1,3260]
Login to [iface: default, target: iqn.2010-05.com.example.server1:iscsirhel6guest, portal:
10.0.0.1,3260] successful.
```

   장치를 분리합니다.

```
# iscsiadm -d2 -m node --logout
scsiadm: Max file limits 1024 1024

Logging out of session [sid: 2, target: iqn.2010-05.com.example.server1:iscsirhel6guest,
portal: 10.0.0.1,3260
Logout of [sid: 2, target: iqn.2010-05.com.example.server1:iscsirhel6guest, portal:
10.0.0.1,3260] successful.
```

*이제 **iSCSI** 장치를 가상화에 사용할 준비가 되었습니다.*

12.5.2. virt-manager에 iSCSI 대상 추가

*다음 절차에서는 **virt-manager** 에서 **iSCSI** 대상을 사용하여 스토리지 풀을 생성하는 방법을 설명합니다.*

*절차 **12.5. virt-manager에 iSCSI 장치 추가***

1. *호스트 물리적 시스템의 스토리지 탭을 엽니다.*

   *연결 세부 정보 창에서 **Storage** 탭을 엽니다.*

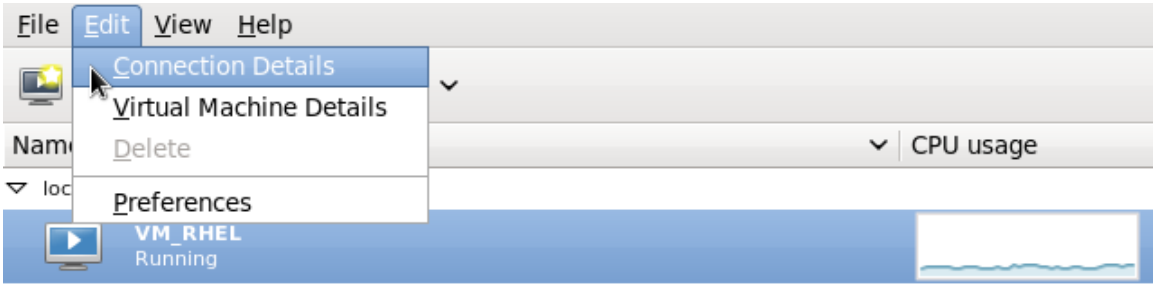
a.

virt-manager 를 엽니다.

b.

기본 virt-manager 창에서 호스트 물리적 시스템을 선택합니다. 편집 메뉴를 클릭하고 연결 세부 정보를 선택합니다.

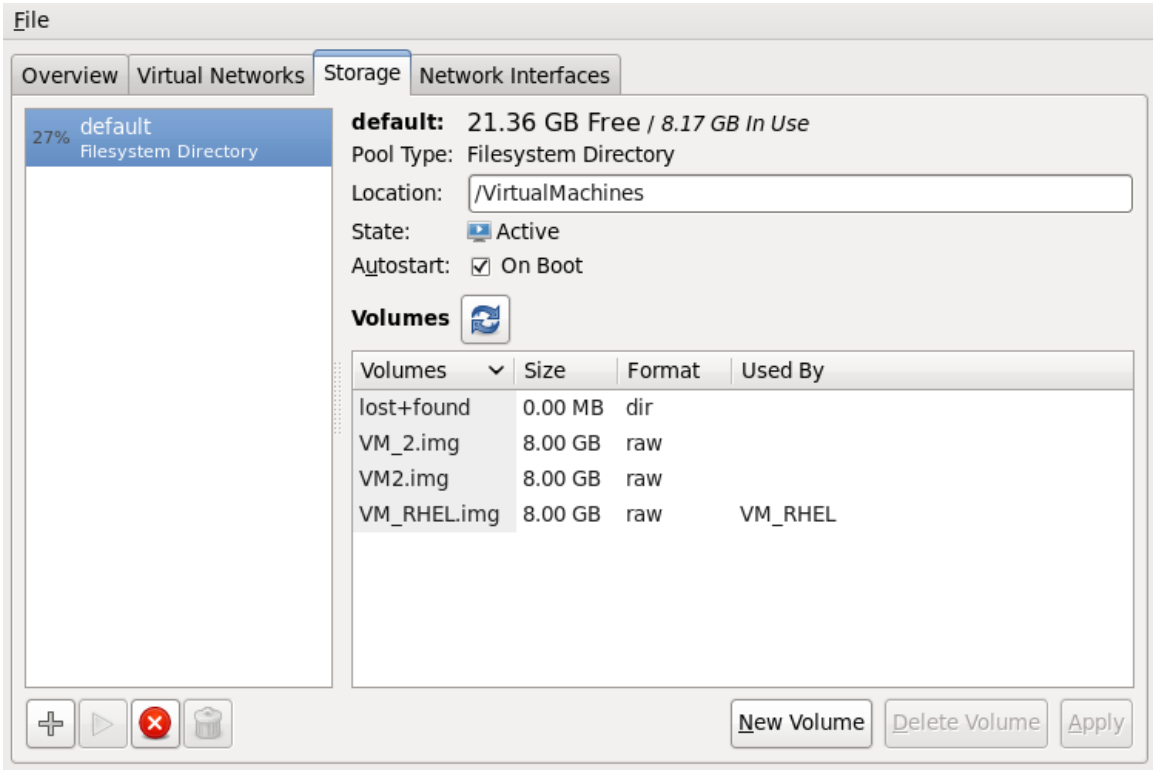
그림 12.19. 연결 세부 정보



c.

스토리지 탭을 클릭합니다.

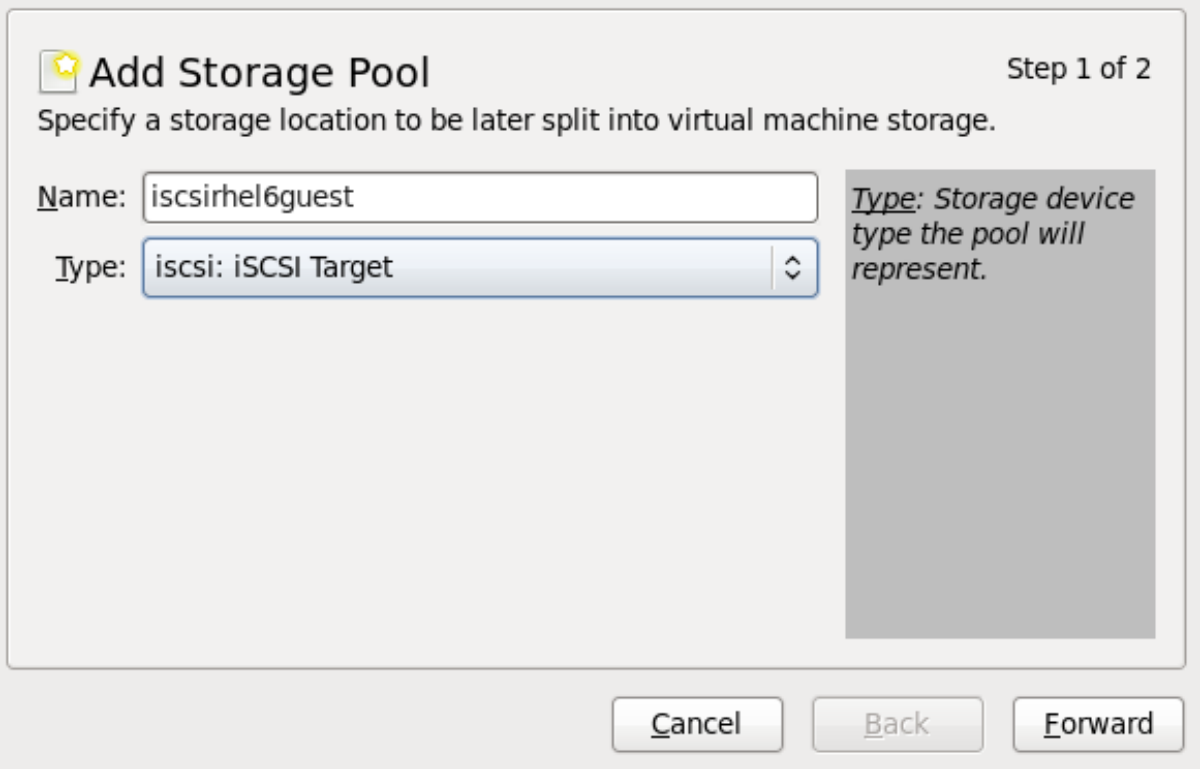
그림 12.20. 스토리지 메뉴



2. *새 풀 추가(1부)*

+ 버튼을 누릅니다(추가 풀 버튼). Add a New Storage Pool 마법사가 표시됩니다.

그림 12.21. iscsi 스토리지 풀 이름 및 유형 추가



*스토리지 풀의 이름을 선택하고 **Type**을 **iscsi**로 변경한 다음 **Forward** 를 눌러 계속합니다.*

3. *새 풀 추가(2부)*

이 메뉴의 필드를 완료하려면 12.5절. "iSCSI 기반 스토리지 풀" 및 절차 12.4. "iSCSI 대상 생성" 에서 사용한 정보가 필요합니다.

a. *iSCSI 소스 및 대상을 입력합니다. **Format** 옵션은 게스트 가상 머신에서 포맷을 처리할 수 없습니다. 대상 경로 를 편집하지 않는 것이 좋습니다. 기본 대상 경로 값인 **/dev/disk/by-path/** 은 해당 디렉터리에 드라이브 경로를 추가합니다. 대상 경로는 마이그레이션을 위한 모든 호스트 물리적 시스템에서 동일해야 합니다.*

b. *iSCSI 대상의 호스트 이름 또는 IP 주소를 입력합니다. 이 예에서는 **host1.example.com** 을 사용합니다.*

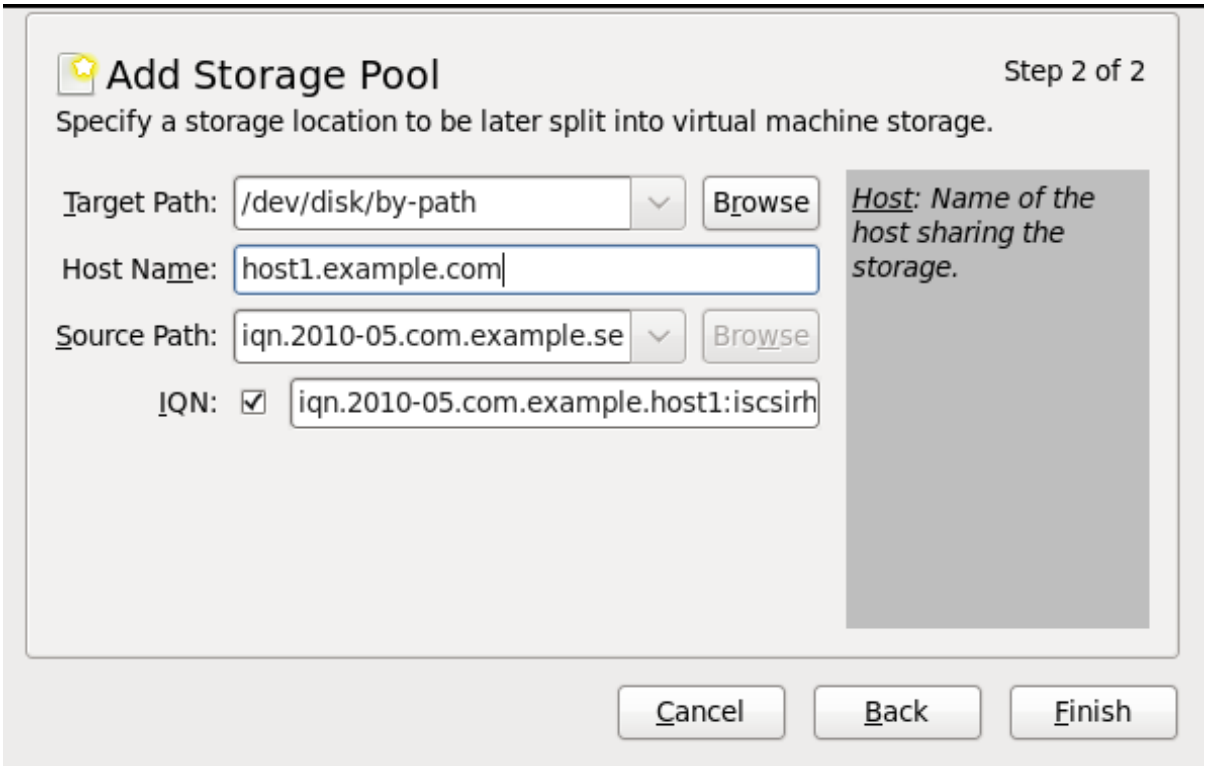
c. *소스 경로필드에 iSCSI 대상 IQN을 입력합니다. 절차 12.4. "iSCSI 대상 생성" 에서 12.5절. "iSCSI 기반 스토리지 풀" 을 살펴보면 **/etc/tgt/targets.conf** 파일에 추가한 정보입니다. 이 예에서는 **iqn.2010-05.com.example.server1:iscsirhel6guest** 를 사용합니다.*

d. *IQN 확인란을 선택하여 이니시에이터의 IQN을 입력합니다. 이 예에서는 **iqn.2010-05.com.example.host1:iscsirhel6** 을 사용합니다.*

e.

Finish 를 클릭하여 새 스토리지 풀을 생성합니다.

그림 12.22. iscsi 스토리지 풀 생성



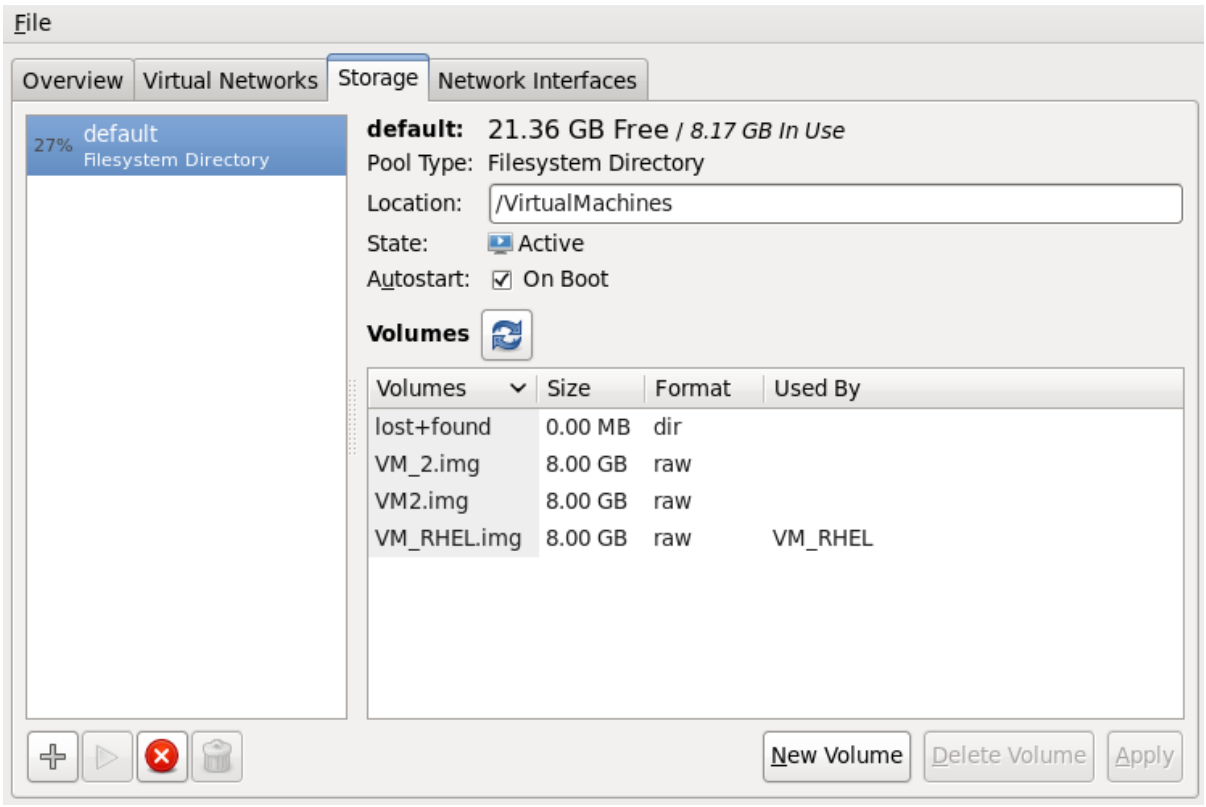
12.5.3. virt-manager를 사용하여 스토리지 풀 삭제

다음 절차에서는 스토리지 풀을 삭제하는 방법을 설명합니다.

1.

동일한 풀을 사용하는 다른 게스트 가상 머신의 문제를 방지하려면 스토리지 풀을 중지하고 사용 중인 리소스를 모두 해제하는 것이 가장 좋습니다. 이를 수행하려면 중지할 스토리지 풀을 선택하고 스토리지 창 하단에 있는 빨간색 **X** 아이콘을 클릭합니다.

그림 12.23. 아이콘 중지



2. ***bin can** 아이콘을 클릭하여 스토리지 풀을 삭제합니다. 이 아이콘은 스토리지 풀을 먼저 중지한 경우에만 활성화됩니다.*

12.5.4. virsh를 사용하여 iSCSI 기반 스토리지 풀 생성

1. ***pool-define-as**를 사용하여 명령줄에서 풀을 정의합니다.*

   ***virsh** 명령줄 도구를 사용하여 스토리지 풀 정의를 생성할 수 있습니다. **virsh** 를 사용하여 스토리지 풀을 생성하는 것은 시스템 관리자가 스크립트를 사용하여 여러 스토리지 풀을 생성하는 데 유용합니다.*

   ***virsh pool-define-as** 명령에는 다음 형식으로 허용되는 몇 가지 매개 변수가 있습니다.*

```
virsh pool-define-as name type source-host source-path source-dev source-name target
```

   매개변수는 다음과 같이 설명됩니다.

type

   *이 풀을 특정 유형으로 정의합니다. **iscsi for example***

name

고유해야 하며 스토리지 풀의 이름을 설정해야 합니다.

source-host* 및 *source-path

*호스트 이름과 **iSCSI IQN***

source-dev* 및 *source-name

***iSCSI** 기반 풀에는 이러한 매개 변수가 필요하지 않으며 **-** 문자를 사용하여 필드를 비워 둡니다.*

대상

*호스트 물리적 시스템에서 **iSCSI** 장치를 마운트할 위치를 정의합니다.*

*아래 예제에서는 이전 단계와 동일한 **iSCSI** 기반 스토리지 풀을 생성합니다.*

```
#   virsh pool-define-as --name scsirhel6guest --type iscsi \
    --source-host server1.example.com \
    --source-dev iqn.2010-05.com.example.server1:iscsirhel6guest
    --target /dev/disk/by-path
Pool iscsirhel6guest defined
```

2. *스토리지 풀이 나열되었는지 확인합니다.*

스토리지 풀 오브젝트가 올바르게 생성되고 상태가 비활성 으로 보고되는지 확인합니다.

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
iscsirhel6guest      inactive   no
```

3. *스토리지 풀 시작*

*이를 위해 **virsh** 명령 **pool-start** 를 사용합니다. **pool-start** 를 사용하면 디렉터리 스토리지 풀을 통해 볼륨 및 게스트 가상 머신에 사용할 수 있습니다.*

```
# virsh pool-start guest_images_disk
Pool guest_images_disk started
# virsh pool-list --all
Name                 State      Autostart
```

```
----------------------------------------
default              active     yes
iscsirhel6guest      active     no
```

4. *자동 시작 켜기*

   *스토리지 풀에 대해 **autostart** 를 켭니다. **autostart**는 서비스가 시작될 때 스토리지 풀을 시작하도록 **libvirtd** 서비스를 구성합니다.*

```
# virsh pool-autostart iscsirhel6guest
Pool iscsirhel6guest marked as autostarted
```

   ***iscsirhel6guest** 풀에 **autostart**가 설정되어 있는지 확인합니다.*

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
iscsirhel6guest      active     yes
```

5. *스토리지 풀 구성 확인*

   스토리지 풀이 올바르게 생성되었고 크기가 올바르게 보고되었는지, 상태 보고서가 실행 중으로 생성되었는지 확인합니다.

```
# virsh pool-info iscsirhel6guest
Name:           iscsirhel6guest
UUID:           afcc5367-6770-e151-bcb3-847bc36c5e28
State:          running
Persistent:     unknown
Autostart:      yes
Capacity:       100.31 GB
Allocation:     0.00
Available:      100.31 GB
```

*이제 **iSCSI** 기반 스토리지 풀을 사용할 수 있습니다.*

12.5.5. virsh를 사용하여 스토리지 풀 삭제

*다음은 **virsh**를 사용하여 스토리지 풀을 삭제하는 방법을 보여줍니다.*

1.

   동일한 풀을 사용하는 다른 게스트 가상 머신의 문제를 방지하려면 스토리지 풀을 중지하고 사용 중인 리소스를 모두 해제하는 것이 가장 좋습니다.

-

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

2. *스토리지 풀의 정의 제거*

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

12.6. NFS 기반 스토리지 풀

*다음 절차에서는 **virt-manager** 에서 **NFS** 마운트 지점을 사용하여 스토리지 풀을 생성하는 방법을 설명합니다.*

12.6.1. virt-manager를 사용하여 NFS 기반 스토리지 풀 생성

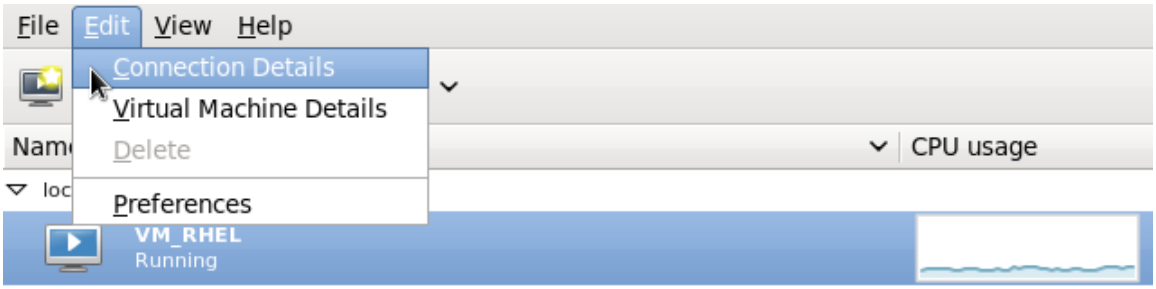
1. *호스트 물리적 시스템의 스토리지 탭을 엽니다.*

   *호스트 세부 정보 창에서 **Storage** 탭을 엽니다.*

   a. *virt-manager 를 엽니다.*

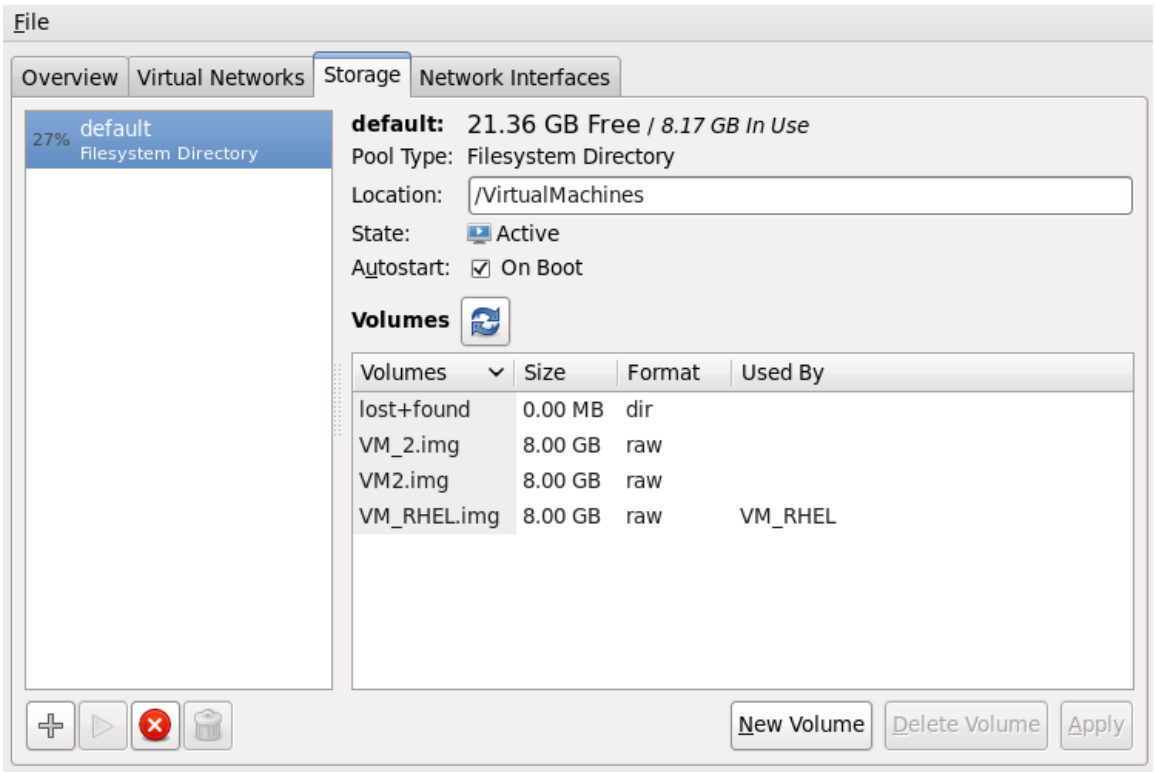
   b. *기본 **virt-manager** 창에서 호스트 물리적 시스템을 선택합니다. 편집 메뉴를 클릭하고 연결 세부 정보를 선택합니다.*

   *그림 **12.24.** 연결 세부 정보*



   c. *스토리지 탭을 클릭합니다.*

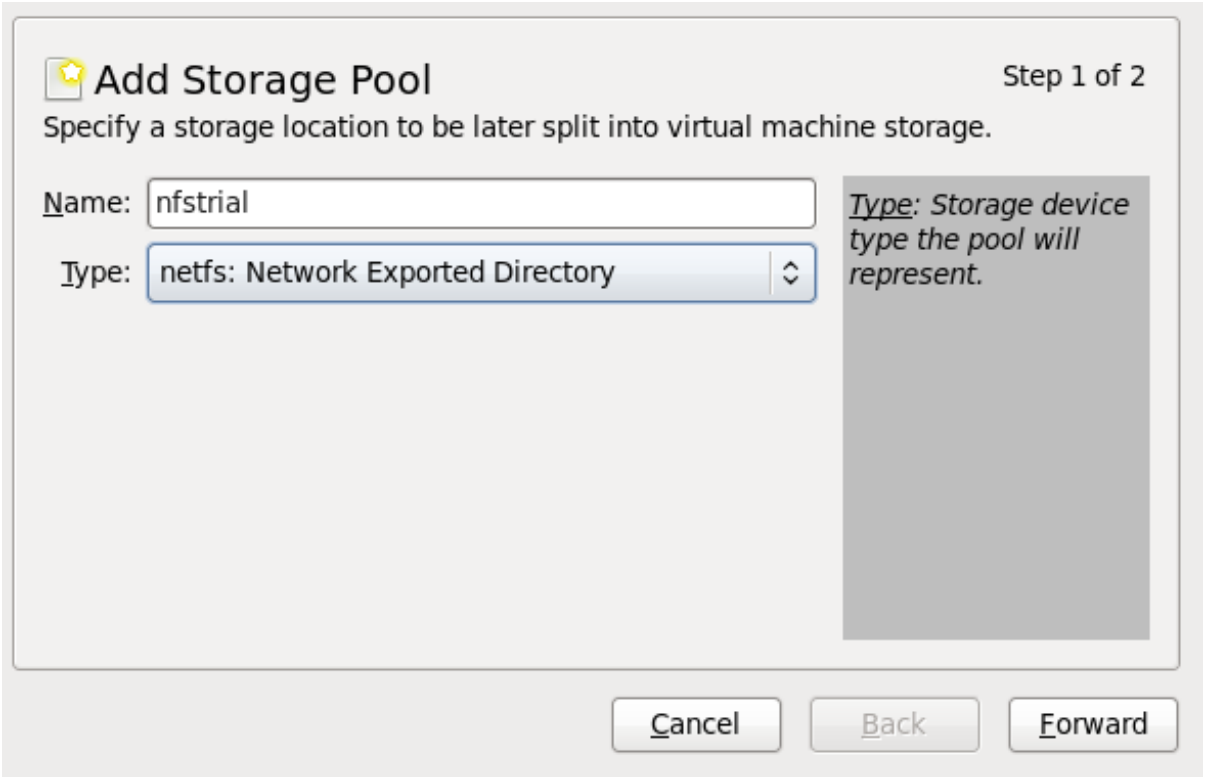
그림 12.25. 스토리지 탭



2. *새 풀 생성(1부)*

   *+ 버튼을 누릅니다(추가 풀 버튼). **Add a New Storage Pool** 마법사가 표시됩니다.*

그림 12.26. NFS 이름 및 유형 추가



*스토리지 풀의 이름을 선택하고 **Forward** 를 눌러 계속합니다.*

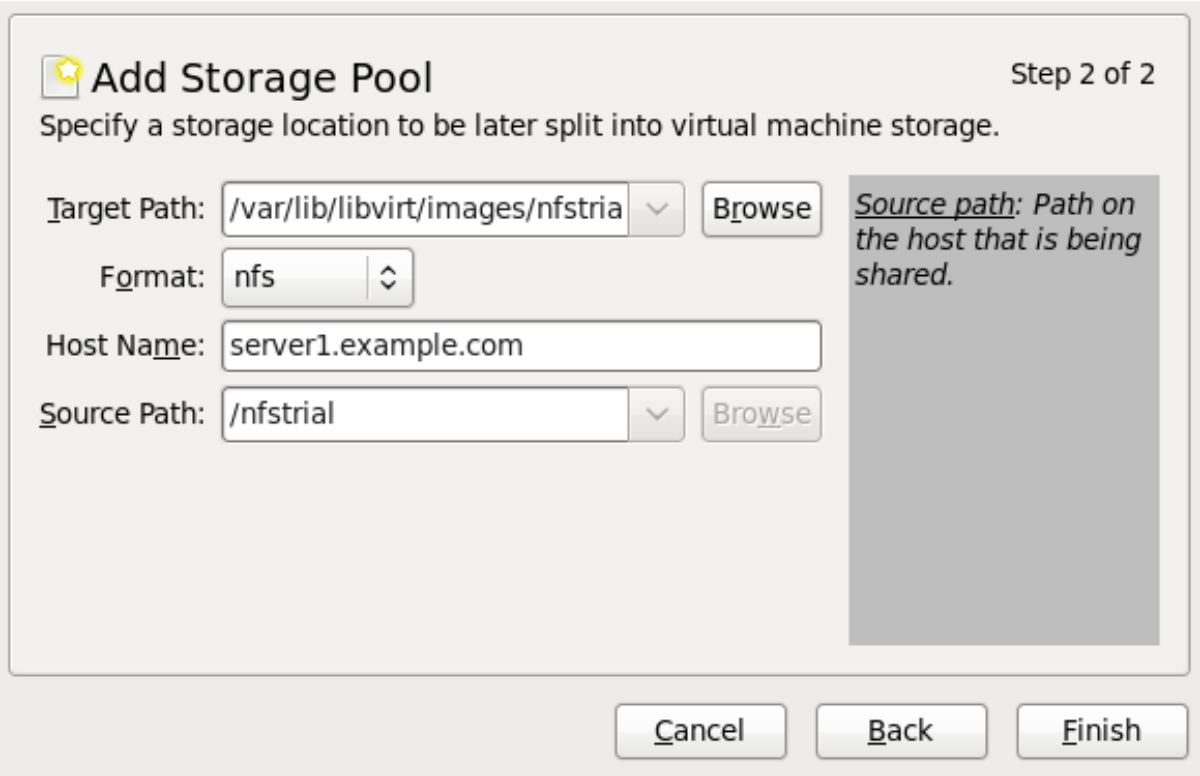
3. *새 풀 생성(2부)*

*장치의 대상 경로, 호스트 이름 및 **NFS** 공유 경로를 입력합니다. **Format** 옵션을 **NFS** 로 설정하거나 **auto** (유형을 감지)로 설정합니다. 마이그레이션을 위해 모든 호스트 물리적 시스템에서 대상 경로가 동일해야 합니다.*

***NFS** 서버의 호스트 이름 또는 **IP** 주소를 입력합니다. 이 예에서는 **server1.example.com** 을 사용합니다.*

***NFS** 경로를 입력합니다. 이 예에서는 **/nfstrial** 을 사용합니다.*

*그림 **12.27. NFS** 스토리지 풀 생성*



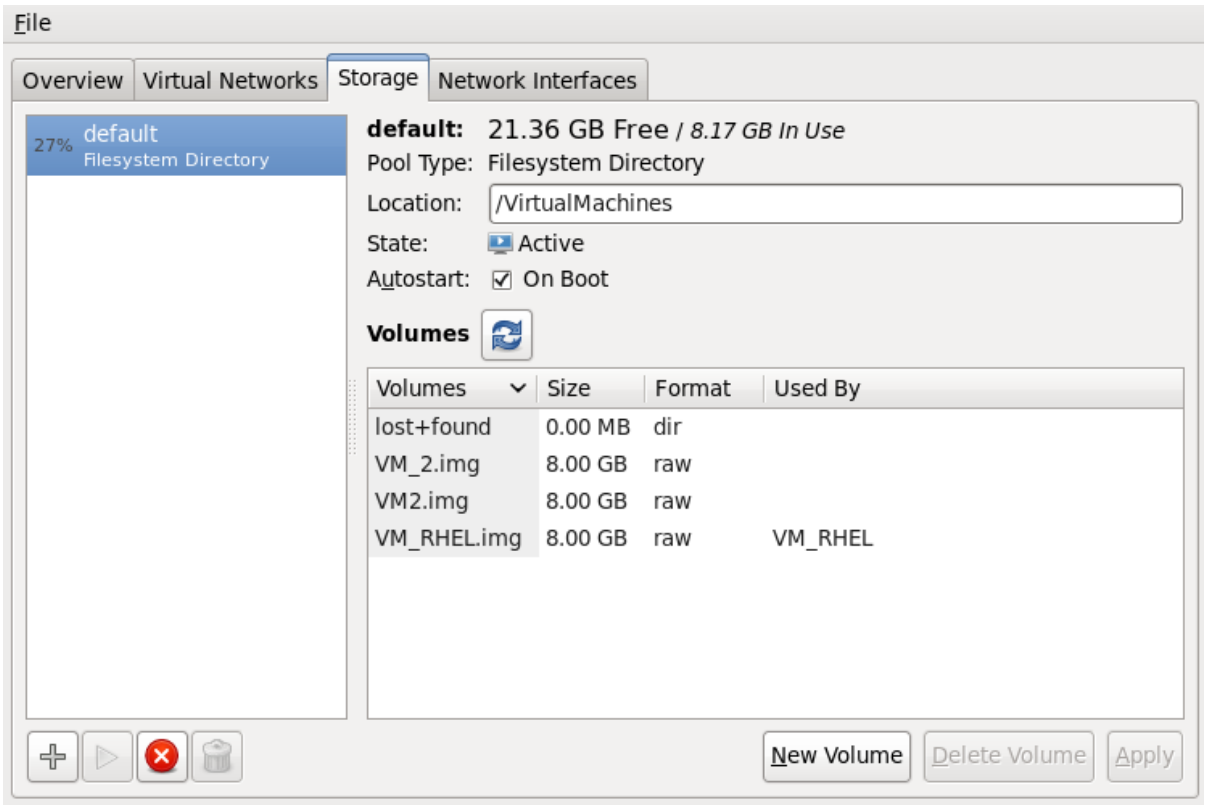
***Finish** 를 눌러 새 스토리지 풀을 생성합니다.*

12.6.2. virt-manager를 사용하여 스토리지 풀 삭제

다음 절차에서는 스토리지 풀을 삭제하는 방법을 설명합니다.

1. *동일한 풀을 사용하는 다른 게스트의 문제를 방지하려면 스토리지 풀을 중지하고 사용하는 리소스를 모두 해제하는 것이 가장 좋습니다. 이를 수행하려면 중지할 스토리지 풀을 선택하고 스토리지 창 하단에 있는 빨간색 **X** 아이콘을 클릭합니다.*

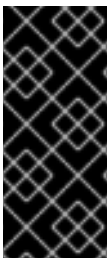
그림 12.28. 아이콘 중지



2. **bin can** 아이콘을 클릭하여 스토리지 풀을 삭제합니다. 이 아이콘은 스토리지 풀을 먼저 중지한 경우에만 활성화됩니다.

12.7. GLUSTERFS 스토리지 풀

GlusterFS는 **FUSE**를 사용하는 사용자 공간 파일 시스템입니다. 게스트 가상 머신에서 활성화하면 **KVM** 호스트 물리적 머신이 하나 이상의 **GlusterFS** 스토리지 볼륨에서 게스트 가상 머신 이미지를 부팅하고 **GlusterFS** 스토리지 볼륨의 이미지를 게스트 가상 머신의 데이터 디스크로 사용할 수 있습니다.



중요

Red Hat Enterprise Linux 6는 스토리지 풀에서 **GlusterFS** 사용을 지원하지 않습니다. 그러나 **Red Hat Enterprise Linux 6.5** 이상에는 **libgfapi** 라이브러리를 사용하여 **GlusterFS**로 가상 시스템을 생성하기 위한 기본 지원이 포함되어 있습니다.

12.8. SCSI 장치가 있는 NPIV VIRTUAL ADAPTER(VHBA) 사용

NPIV(N_Port ID Virtualization)는 하나의 물리적 파이버 채널 호스트 버스 어댑터**(HBA)**를 공유할 수 있는 소프트웨어 기술입니다.

이를 통해 여러 게스트가 여러 물리적 호스트의 동일한 스토리지를 볼 수 있으므로 스토리지 마이그레이션 경로가 쉬워집니다. 따라서 올바른 스토리지 경로가 지정된 한 마이그레이션이 스토리지를 생성하거나 복사하기 위해 마이그레이션이 필요하지 않습니다.

*가상화에서 가상 호스트 버스 어댑터 또는 **vHBA** 는 가상 시스템의 **LUN**을 제어합니다. 각 **vHBA**는 자체 **WWNN(World Wide Node Name)**과 **WWPN(Wide Wide Port Name)**으로 식별됩니다. 스토리지의 경로는 **WWNN** 및 **WWPN** 값에 의해 결정됩니다.*

*이 섹션에서는 가상 머신에서 **vHBA** 를 구성하는 데 필요한 지침을 제공합니다. **Red Hat Enterprise Linux 6**는 호스트 재부팅 시 영구 **vHBA** 구성을 지원하지 않습니다. 호스트 재부팅 후에도 **vHBA** 관련 설정을 확인합니다.*

12.8.1. vHBA 생성

절차 12.6. vHBA 생성

1. *호스트 시스템에서 **HBA** 찾기*

   *호스트 시스템에서 **HBA**를 찾으려면 호스트 시스템의 **SCSI** 장치를 검사하여 **vport** 기능이 있는 **scsi_host** 를 찾습니다.*

   *다음 명령을 실행하여 **scsi_host** 목록을 검색합니다.*

```
# virsh nodedev-list --cap scsi_host
scsi_host0
scsi_host1
scsi_host2
scsi_host3
scsi_host4
```

   ***scsi_host** 에 대해 다음 명령을 실행하여 **vport** 기능이 있는 **scsi_host** 을 나타내는 **<capability type='vport_ops'>** 행의 장치 **XML**을 검사합니다.*

```
# virsh nodedev-dumpxml scsi_hostN
```

2. ***HBA**의 세부 정보 확인*

   ***virsh nodedev-dumpxml HBA_device** 명령을 사용하여 **HBA**의 세부 정보를 확인합니다.*

   ***virsh nodedev-dumpxml** 명령의 **XML** 출력에는 **vHBA**를 만드는 데 사용되는 **<name>**,*

<wwnn> 및 <wwpn> 필드가 나열됩니다. <max_vports> 값은 지원되는 최대 vHBAs 수를 표시합니다.

```
# virsh nodedev-dumpxml scsi_host3
<device>
 <name>scsi_host3</name>
 <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3</path>
 <parent>pci_0000_10_00_0</parent>
 <capability type='scsi_host'>
  <host>3</host>
  <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>
```

이 예에서 <max_vports> 값은 HBA 구성에서 사용할 수 있는 총 127개의 가상 포트가 있음을 보여줍니다. <vports> 값은 현재 사용 중인 가상 포트 수를 보여줍니다. 이러한 값은 vHBA를 생성한 후 업데이트됩니다.

3. *vHBA 호스트 장치 만들기*

   vHBA 호스트에 대해 다음과 유사한 XML 파일을 만듭니다(이 예제에서는 vhba_host3.xml).

```
# cat vhba_host3.xml
  <device>
   <parent>scsi_host3</parent>
   <capability type='scsi_host'>
    <capability type='fc_host'>
    </capability>
   </capability>
  </device>
```

<parent> 필드는 이 vHBA 장치와 연결할 HBA 장치를 지정합니다. <device> 태그의 세부 사항은 다음 단계에서 호스트에 대한 새 vHBA 장치를 생성하는 데 사용됩니다. nodedev XML 형식에 대한 자세한 내용은 을 참조하십시오 http://libvirt.org/formatnode.html.

4. *vHBA 호스트 장치에 새 vHBA를 생성*

   vhba_host3 에 vHBA를 생성하려면 virsh nodedev-create 명령을 사용합니다.

```
# virsh nodedev-create vhba_host3.xml
Node device scsi_host5 created from vhba_host3.xml
```

5. **vHBA 확인**

   virsh nodedev-dumpxml 명령을 사용하여 새 **vHBA**의 세부 정보 **(scsi_host5)**를 확인합니다.

```
# 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>
    <capability type='fc_host'>
      <wwnn>5001a4a93526d0a1</wwnn>
      <wwpn>5001a4ace3ee047d</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
  </capability>
</device>
```

12.8.2. vHBA를 사용하여 스토리지 풀 생성

vHBA 구성을 유지하기 위해 **vHBA**를 기반으로 **libvirt** 스토리지 풀을 정의하는 것이 좋습니다.

스토리지 풀을 사용하면 다음과 같은 두 가지 주요 이점이 있습니다.

- **libvirt** 코드는 **virsh** 명령 출력을 사용하여 **LUN** 경로를 쉽게 찾을 수 있습니다.
- 가상 머신 마이그레이션은 대상 시스템에서 동일한 **vHBA** 이름으로 스토리지 풀을 정의하고 시작해야 합니다. 이렇게 하려면 가상 머신의 **XML** 구성에서 **vHBA LUN, libvirt** 스토리지 풀 및 볼륨 이름을 지정해야 합니다. 예제는 **12.8.3**절. “**vHBA LUN**을 사용하도록 가상 머신 구성” 를 참조하십시오.

1. **SCSI** 스토리지 풀 생성

   vHBA 구성을 만들려면 먼저 아래 형식을 사용하여 **vHBA** 를 기반으로 **libvirt 'scsi'** 스토리지 풀 **XML** 파일을 만듭니다.

참고

절차 12.6. “vHBA 생성” 에 생성된 **vHBA**를 호스트 이름으로 사용하고 스토리지 풀 구성의 경우 **vHBA** 이름 **scsi_hostN** 을 **hostN** 으로 수정합니다. 이 예에서 **vHBA**의 이름은 **scsi_host5** 이며, **Red Hat Enterprise Linux 6 libvirt** 스토리지 풀에서 **< adapter name='host5'/ >**로 지정됩니다.

시스템의 **/dev/disk/by-{path|id|uuid|label}** 위치 중 하나와 같이 **<path>** 값에 안정적인 위치를 사용하는 것이 좋습니다. **<path>** 및 **<target>** 의 요소에 대한 자세한 내용은 에서 **http://libvirt.org/formatstorage.html** 확인할 수 있습니다.

이 예에서 **'scsi'** 스토리지 풀 이름은 **vhbapool_host3.xml:**

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <uuid>e9392370-2917-565e-692b-d057f46512d6</uuid>
  <capacity unit='bytes'>0</capacity>
  <allocation unit='bytes'>0</allocation>
  <available unit='bytes'>0</available>
  <source>
    <adapter name='host5'/>
  </source>
   <target>
     <path>/dev/disk/by-path</path>
     <permissions>
       <mode>0700</mode>
       <owner>0</owner>
       <group>0</group>
     </permissions>
   </target>
 </pool>
```

2. 풀 정의

스토리지 풀(이 예제에서는 **vhbapool_host3** 이라는 이름) 을 정의하려면 **virsh pool-define** 명령을 사용합니다.

```
# virsh pool-define vhbapool_host3.xml
Pool vhbapool_host3 defined from vhbapool_host3.xml
```

3. 풀 시작

다음 명령을 사용하여 스토리지 풀을 시작합니다.

```
# virsh pool-start vhbapool_host3
Pool vhbapool_host3 started
```

4. *자동 시작 활성화*

   *마지막으로 후속 호스트 재부팅이 가상 머신에서 사용할 **vHBAs**를 자동으로 정의하도록 하려면 스토리지 풀 **autostart** 기능을 설정합니다(이 예에서는 **vhbapool_host3**풀의 경우).*

```
# virsh pool-autostart vhbapool_host3
```

12.8.3. vHBA LUN을 사용하도록 가상 머신 구성

vHBA용 스토리지 풀을 생성한 후 가상 머신 구성에 vHBA LUN을 추가합니다.

1. *사용 가능한 LUN 찾기*

   먼저 vHBA에서 사용 가능한 LUN 목록을 생성하려면 virsh vol-list 명령을 사용합니다. 예를 들어 다음과 같습니다.

```
# virsh vol-list vhbapool_host3
 Name                 Path
-----------------------------------------------------------------------------
 unit:0:4:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016844602198-lun-0
 unit:0:5:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0
```

   표시되는 LUN 이름 목록은 가상 머신 구성에서 디스크 볼륨으로 사용할 수 있습니다.

2. *가상 머신에 vHBA LUN 추가*

   가상 머신의 XML에 을 지정하여 가상 머신에 vHBA LUN을 추가합니다.

   - *lun 매개변수에서 장치 유형 disk 또는 <disk> 로,*
   - *<source> 매개변수의 소스 장치입니다. 이를 /dev/sdaN 로 입력하거나 /dev/disk/by-path|by-id|by-uuid|by-label 에서 udev에 의해 생성된 심볼릭 링크로 입력할 수 있습니다. virsh vol-list pool 명령을 실행하여 찾을 수 있습니다.*

   예를 들어 다음과 같습니다.

```
<disk type='block' device='lun'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/disk/by-path/pci-0000\:04\:00.1-fc-0x203400a0b85ad1d7-lun-0'/>
```

```
  <target dev='sda' bus='scsi'/>
</disk>
```

12.8.4. vHBA 스토리지 풀 삭제

vHBA 스토리지 풀은 **virsh pool-destroy** 명령을 통해 삭제할 수 있습니다.

```
# virsh pool-destroy vhbapool_host3
```

다음 명령을 사용하여 **vHBA**를 삭제합니다.

```
# virsh nodedev-destroy scsi_host5
```

풀과 **vHBA**가 제거되었는지 확인하려면 다음을 실행합니다.

```
# virsh nodedev-list --cap scsi_host
```

scsi_host5 더 이상 결과 목록에 표시되지 않습니다.

13장. VOLUMES

13.1. 볼륨 생성

이 섹션에서는 블록 기반 스토리지 풀 내에 디스크 볼륨을 생성하는 방법을 보여줍니다. 아래 예제에서 **virsh vol-create-as** 명령은 **guest_images_disk** 스토리지 풀 내에 특정 크기가 **GB**인 스토리지 볼륨을 생성합니다. 이 명령이 필요한 볼륨당 반복되므로 예제에 표시된 대로 **3**개의 볼륨이 생성됩니다.

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

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

# virsh vol-create-as guest_images_disk volume3 8G
Vol volume3 created

# virsh vol-list guest_images_disk
Name                 Path
-----------------------------------------
volume1              /dev/sdb1
volume2              /dev/sdb2
volume3              /dev/sdb3

# parted -s /dev/sdb print
Model: ATA ST3500418AS (scsi)
Disk /dev/sdb: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt

Number  Start   End     Size    File system  Name     Flags
2      17.4kB  8590MB  8590MB               primary
3      8590MB  17.2GB  8590MB               primary
1      21.5GB  30.1GB  8590MB               primary
```

13.2. 볼륨 복제

새 볼륨은 복제 중인 볼륨과 동일한 스토리지 풀의 스토리지에서 할당됩니다. **virsh vol-clone** 에는 복제할 볼륨이 포함된 스토리지 풀의 이름을 지정하는 **--pool** 인수가 있어야 합니다. 나머지 명령은 복제할 볼륨의 이름과 복제된 새 볼륨의 이름**(clone1)**입니다. **virsh vol-list** 명령은 스토리지 풀 **(guest_images_disk)**에 있는 볼륨을 나열합니다.

```
# virsh vol-clone --pool guest_images_disk volume3 clone1
Vol clone1 cloned from volume3

# virsh vol-list guest_images_disk
Name                 Path
-----------------------------------------
```

```
volume1             /dev/sdb1
volume2             /dev/sdb2
volume3             /dev/sdb3
clone1             /dev/sdb4


# parted -s /dev/sdb print
Model: ATA ST3500418AS (scsi)
Disk /dev/sdb: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    File system  Name     Flags
1      4211MB  12.8GB  8595MB  primary
2      12.8GB  21.4GB  8595MB  primary
3      21.4GB  30.0GB  8595MB  primary
4      30.0GB  38.6GB  8595MB  primary
```

13.3. 게스트에 스토리지 장치 추가

이 섹션에서는 게스트에 스토리지 장치를 추가하는 방법을 설명합니다. 추가 스토리지는 필요한 경우에만 추가할 수 있습니다.

13.3.1. 게스트에 파일 기반 스토리지 추가

파일 기반 스토리지는 게스트의 가상화된 하드 드라이브 역할을 하는 호스트 물리적 시스템 파일 시스템에 저장된 파일 컬렉션입니다. 파일 기반 스토리지를 추가하려면 다음 단계를 수행합니다.

절차 13.1. 파일 기반 스토리지 추가

1. 스토리지 파일을 생성하거나 기존 파일(예: **IMG** 파일)을 사용합니다. 다음 두 명령 모두 게스트의 추가 스토리지로 사용할 수 있는 **4GB** 파일을 생성합니다.

   - 파일 기반 스토리지 이미지에는 사전 할당된 파일을 사용하는 것이 좋습니다. 다음과 같이 다음 **dd** 명령을 사용하여 사전 할당된 파일을 생성합니다.

     ```
     # dd if=/dev/zero of=/var/lib/libvirt/images/FileName.img bs=1M count=4096
     ```

   - 또는 미리 할당된 파일 대신 스파스 파일을 만듭니다. 스파스 파일은 훨씬 더 빠르며 테스트에 사용할 수 있지만 데이터 무결성 및 성능 문제로 인해 프로덕션 환경에서는 사용하지 않는 것이 좋습니다.

     ```
     # dd if=/dev/zero of=/var/lib/libvirt/images/FileName.img bs=1M seek=4096 count=0
     ```

2. *새 파일에 **< disk>** 요소를 작성하여 추가 스토리지를 생성합니다. 이 예제에서는 이 파일을 **NewStorage.xml** 이라고 합니다.*

   & lt;disk > 요소는 디스크 소스와 가상 블록 장치의 장치 이름을 설명합니다. 장치 이름은 게스트의 모든 장치에서 고유해야 하며 게스트가 가상 블록 장치를 찾을 버스를 식별합니다. 다음 예제에서는 소스가 **FileName.img** 인 파일 기반 스토리지 컨테이너인 **virtio** 블록 장치를 정의합니다.

```
<disk type='file' device='disk'>
  <driver name='qemu' type='raw' cache='none'/>
  <source file='/var/lib/libvirt/images/FileName.img'/>
  <target dev='vdb'/>
</disk>
```

   장치 이름은 **"hd"** 또는 **"sd"**로 시작하여 각각 **IDE**와 **SCSI** 디스크를 확인할 수 있습니다. 구성 파일에는 새 장치의 버스 위치를 지정하는 **< address >** 하위 요소도 포함될 수 있습니다. **virtio** 블록 장치의 경우 **PCI** 주소여야 합니다. **< address>** 하위 요소를 생략하면 **libvirt**에서 사용 가능한 다음 **PCI** 슬롯을 찾아 할당할 수 있습니다.

3. ***CD-ROM**을 다음과 같이 연결합니다.*

```
<disk type='file' device='cdrom'>
  <driver name='qemu' type='raw' cache='none'/>
  <source file='/var/lib/libvirt/images/FileName.img'/>
  <readonly/>
  <target dev='hdc'/>
</disk >
```

4. ***NewStorage.xml** 에 정의된 장치를 게스트**(Guest1)**로 추가합니다.*

```
# virsh attach-device --config Guest1 ~/NewStorage.xml
```

참고

*이러한 변경 사항은 게스트를 삭제한 후 재시작한 후에만 적용됩니다. 또한 영구 장치는 **virsh define** 명령을 사용하여 구성을 저장한 도메인인 영구 도메인에만 추가할 수 있습니다.*

*게스트가 실행되고 있고 게스트가 삭제될 때까지 새 장치를 일시적으로 추가하려는 경우 **--config** 옵션을 생략합니다.*

```
# virsh attach-device Guest1 ~/NewStorage.xml
```

참고

virsh 명령을 사용하면 더 간단한 구문을 사용하고 **XML** 파일을 생성할 필요 없이 제한된 수의 매개 변수를 설정할 수 있는 **attach-disk** 명령을 허용합니다. **attach-disk** 명령은 다음과 같이 이전에 언급한 **attach-device** 명령과 유사한 방식으로 사용됩니다.

```
# virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.img vdb --cache
none --driver qemu --subdriver raw
```

virsh attach-disk 명령은 **--config** 옵션도 허용합니다.

5. 게스트 머신을 시작합니다(현재 실행 중인 경우).

```
# virsh start Guest1
```

참고

다음은 **Linux** 게스트 전용입니다. 다른 운영 체제는 다른 방식으로 새 스토리지 장치를 처리합니다. 다른 시스템의 경우 해당 운영 체제 설명서를 참조하십시오.

6. 디스크 드라이브 파티셔닝

게스트에는 이제 **/dev/vdb** 라는 하드 디스크 장치가 있습니다. 필요한 경우 이 디스크 드라이브를 분할하고 파티션을 포맷합니다. 추가한 장치가 표시되지 않으면 게스트 운영 체제에서 디스크 핫플러그에 문제가 있음을 나타냅니다.

a. 새 장치에 대해 **fdisk** 를 시작합니다.

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

b. 새 파티션에 대해 **n** 을 입력합니다.

c.

다음이 나타납니다.

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

*기본 파티션에 대해 **p** 를 입력합니다.*

d.

*사용 가능한 파티션 번호를 선택합니다. 이 예에서 첫 번째 파티션은 **1** 을 입력하여 선택합니다.*

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

e.

***Enter** 키를 눌러 기본 첫 번째 실린더를 입력합니다.*

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

f.

*파티션의 크기를 선택합니다. 이 예에서는 **Enter** 를 눌러 전체 디스크가 할당됩니다.*

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

g.

*파티션 유형을 구성하려면 **t** 를 입력합니다.*

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

h.

*이전 단계에서 만든 파티션을 선택합니다. 이 예에서는 파티션 번호가 **1**개뿐이므로 파티션 번호가 **1** 개뿐이므로 파티션 번호는 **1**개뿐이고, **metadata**에서 자동으로 파티션 **1**을 선택했습니다.*

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

i.

***Linux** 파티션에 **83** 을 입력합니다.*

```
Hex code (type L to list codes): 83
```

.

j.
w 를 입력하여 변경 사항을 작성하고 종료합니다.

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

k.
새 파티션을 ext3 파일 시스템으로 포맷합니다.

```
# mke2fs -j /dev/vdb1
```

7.
마운트 디렉터리를 생성하고 게스트에 디스크를 마운트합니다. 이 예에서는 디렉터리가 myfiles 에 있습니다.

```
# mkdir /myfiles
# mount /dev/vdb1 /myfiles
```

게스트에는 이제 가상화된 파일 기반 스토리지 장치가 추가로 있습니다. 그러나 이 스토리지는 게스트의 /etc/fstab 파일에 정의되어 있지 않으면 재부팅 후에도 영구적으로 마운트되지 않습니다.

```
/dev/vdb1    /myfiles    ext3    defaults    0 0
```

13.3.2. 게스트에 하드 드라이브 및 기타 블록 장치 추가

시스템 관리자는 추가 하드 드라이브를 사용하여 게스트의 스토리지 공간을 늘리거나 사용자 데이터와 시스템 데이터를 분리하는 옵션이 있습니다.

절차 13.2. 게스트에 물리적 블록 장치 추가

1.
다음 절차에서는 호스트 물리적 시스템의 하드 드라이브를 게스트에 추가하는 방법을 설명합니다. CD-ROM, DVD 및 플로피 장치를 포함한 모든 물리적 블록 장치에 적용됩니다.

하드 디스크 장치를 호스트 물리적 시스템에 물리적으로 연결합니다. 기본적으로 드라이브에 액세스할 수 없는 경우 호스트 물리적 시스템을 구성합니다.

2.
다음 중 하나를 수행합니다.

a.
새 파일에 disk 요소를 작성하여 추가 스토리지를 생성합니다. 이 예제에서는 이 파일을 NewStorage.xml 이라고 합니다. 다음 예제는 호스트 물리적 시스템 파티션 /dev/sr0에 대한

추가 장치 기반 스토리지 컨테이너를 포함하는 구성 파일 섹션입니다.

```
<disk type='block' device='disk'>
    <driver name='qemu' type='raw' cache='none'/>
    <source dev='/dev/sr0'/>
    <target dev='vdc' bus='virtio'/>
</disk>
```

b.

이전 섹션의 지침에 따라 장치를 게스트 가상 시스템에 연결합니다. 또는 다음과 같이 **virsh attach-disk** 명령을 사용할 수 있습니다.

```
# virsh attach-disk Guest1 /dev/sr0 vdc
```

다음 옵션을 사용할 수 있습니다.

- **virsh attach-disk** 명령에서는 다음과 같이 **--config,--type, --mode** 옵션도 허용합니다.

  ```
  # virsh attach-disk Guest1 /dev/sr0 vdc --config --type cdrom --mode readonly
  ```

- 또한 **--type** 은 장치가 하드 드라이브인 경우 **--type** 디스크 도 허용합니다.

3.

게스트 가상 머신에는 이제 **Linux**에서 **/dev/vdc** 라는 새 하드 디스크 장치(또는 게스트 가상 머신 **OS**가 선택한 항목) 또는 **Windows**의 **D:** 드라이브(예:)에 따라 비슷한 새로운 하드 디스크 장치가 있습니다. 이제 게스트 가상 머신의 운영 체제에 대한 표준 절차에 따라 게스트 가상 머신에서 디스크를 초기화할 수 있습니다. 예제는 *절차 13.1. "파일 기반 스토리지 추가"*를 참조하십시오.

주의

블록 장치를 게스트에 추가할 때는 보안 고려 사항을 따르십시오. 이 정보는 에서 찾을 **https://access.redhat.com/site/documentation/** 수 있는 **Red Hat Enterprise Linux Virtualization** 보안 가이드에서 자세히 설명합니다.

중요

게스트 가상 시스템에는 전체 디스크 또는 블록 장치에 대한 쓰기 액세스 권한이 부여되지 않아야 합니다(예: /dev/sdb). 전체 블록 장치에 액세스할 수 있는 게스트 가상 머신은 호스트 물리적 시스템 시스템을 손상시키는 데 사용할 수 있는 볼륨 레이블을 수정할 수 있습니다. 파티션(예: /dev/sdb1) 또는 LVM 볼륨을 사용하여 이 문제를 방지합니다.

13.4. 볼륨 삭제 및 제거

*이 섹션에서는 **virsh vol-delete** 명령을 사용하여 블록 기반 스토리지 풀에서 디스크 볼륨을 삭제하는 방법을 보여줍니다. 이 예에서 볼륨은 볼륨 **1** 이며 스토리지 풀은 **guest_images** 입니다.*

```
# virsh vol-delete --pool guest_images volume1
Vol volume1 deleted
```

14장. VIRSH로 게스트 가상 머신 관리

virsh 는 게스트 가상 머신과 하이퍼바이저를 관리하기 위한 명령줄 인터페이스 툴입니다. **virsh** 명령줄 툴은 **libvirt** 관리 **API**를 기반으로 하며 **qemu-kvm** 명령 및 그래픽 **virt-manager** 애플리케이션의 대안으로 작동합니다. **virsh** 명령은 권한이 없는 사용자가 읽기 전용 모드로 사용하거나 루트 액세스 권한, 전체 관리 기능을 통해 사용할 수 있습니다. **virsh** 명령은 가상화 관리 스크립팅에 이상적입니다.

14.1. 일반 명령

이 섹션의 명령은 도메인에 국한되지 않기 때문에 일반적입니다.

14.1.1. help

$ virsh help [command|group] help 명령은 옵션과 함께 또는 사용하지 않고 사용할 수 있습니다. 옵션 없이 사용하면 한 줄에 하나씩 모든 명령이 나열됩니다. 옵션과 함께 사용하면 카테고리로 그룹화되어 각 그룹의 키워드를 표시합니다.

특정 옵션에만 해당하는 명령을 표시하려면 해당 그룹에 대한 키워드를 옵션으로 제공해야 합니다. 예를 들어 다음과 같습니다.

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

동일한 명령을 명령 옵션과 함께 사용하면 해당 하나의 특정 명령에 대한 도움말 정보를 제공합니다. 예를 들어 다음과 같습니다.

```
$ virsh help vol-path
  NAME
    vol-path - returns the volume path for a given volume name or key

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

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

14.1.2. 종료 및 종료

quit 명령과 **exit** 명령은 터미널을 종료합니다. 예를 들어 다음과 같습니다.

```
$ virsh exit
```

```
$ virsh quit
```

14.1.3. version

version 명령은 현재 **libvirt** 버전을 표시하고 빌드의 위치에 대한 정보를 표시합니다. 예를 들어 다음과 같습니다.

```
$ virsh version
Compiled against library: libvirt 1.1.1
Using library: libvirt 1.1.1
Using API: QEMU 1.1.1
Running hypervisor: QEMU 1.5.3
```

14.1.4. 인수 표시

virsh echo [--shell][--xml][--xml] 명령을 에코 또는 지정된 인수를 표시합니다. **echoed** 인수는 공백으로 구분됩니다. **--shell** 옵션을 사용하면 출력이 필요한 경우 단일 인용되어 쉘 명령에서 재사용에 적합합니다. **--xml** 옵션을 사용하면 **XML** 파일에서 사용할 수 있는 출력이 적합합니다. 예를 들어 **virsh echo --shell "hello world"** 명령은 출력 **'hello world'** 을 보냅니다.

14.1.5. 연결

하이퍼바이저 세션에 연결합니다. 쉘이 처음 시작되면 **-c** 명령에서 **URI** 매개 변수를 요청하면 이 명령이 자동으로 실행됩니다. **URI**는 하이퍼바이저에 연결하는 방법을 지정합니다. 가장 일반적으로 사용되는 **URI**는 다음과 같습니다.

- **Xen :/// -** 로컬 **Xen** 하이퍼바이저에 연결합니다.
- **QEMU:///system - QEMU** 및 **KVM** 도메인을 감독하는 데몬에 **root**로 로컬로 연결합니다.
- **kvm:///session -** 사용자로 로컬로 사용자 **QEMU** 및 **KVM** 도메인 집합에 연결합니다.
- **LXC:/// -** 로컬 **Linux** 컨테이너에 연결됩니다.

추가 값은 **libvirt**의 웹 사이트 **http://libvirt.org/uri.html** 에서 사용할 수 있습니다.

명령은 다음과 같이 실행할 수 있습니다.

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

여기서 **{name}** 은 하이퍼바이저의 시스템 이름(호스트 이름) 또는 **URL(virsh uri** 명령의 출력)입니다. 읽기 전용 연결을 시작하려면 위 명령을 **--readonly** 로 추가합니다. **URI**에 대한 자세한 내용은 *원격 URI* 을 참조하십시오. **URI**를 잘 모를 경우 **virsh uri** 명령은 이를 표시합니다.

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

14.1.6. 기본 정보 표시

다음 명령을 사용하여 기본 정보를 표시할 수 있습니다.

- **$ hostname -** 하이퍼바이저의 호스트 이름을 표시
- **$ SysInfo -** 하이퍼바이저 시스템 정보 의 **XML** 표현(사용 가능한 경우)을 표시

14.1.7. NMI 삽입

$ virsh inject-nmi [domain] 은 게스트 가상 시스템에 **NMI (non-maskable interrupt)** 메시지를 삽입합니다. 이는 복구할 수 없는 하드웨어 오류와 같이 응답 시간이 중요할 때 사용됩니다. 이 명령을 실행하

려면 다음을 수행합니다.

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

14.2. VIRSH를 사용하여 장치 연결 및 업데이트

스토리지 장치 연결에 대한 자세한 내용은 다음을 참조하십시오. 13.3.1절. “ 게스트에 파일 기반 스토리지 추가”

절차 14.1. 게스트 가상 머신에서 사용할 USB 장치 핫플러그

다음 절차에서는 USB 장치를 게스트 가상 머신에 연결하는 방법을 설명합니다. 게스트 가상 머신이 핫플러그 프로세스로 실행 중이거나 게스트를 종료하는 동안 수행할 수 있습니다. 에뮬레이션할 장치를 호스트 물리적 시스템에 연결해야 합니다.

1. 다음 명령을 사용하여 연결할 USB 장치를 찾습니다.

```
# lsusb -v

idVendor           0x17ef Lenovo
idProduct          0x480f Integrated Webcam [R5U877]
```

2. XML 파일을 만들고 논리 이름(예:usb_device.xml)을 지정합니다. 검색에 표시된 것과 동일하게 공급 업체 및 제품 ID를 복사해야 합니다.

그림 14.1. USB 장치 XML 조각

```
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x17ef'/>
    <product id='0x480f'/>
  </source>
</hostdev>
...
```

3. 다음 명령을 사용하여 장치를 연결합니다.

```
# virsh attach-device rhel6 --file usb_device.xml --config
```

*이 예제에서 **[rhel6]**은 게스트 가상 머신의 이름이고 **[usb_device.xml]**은 이전 단계에서 만든 파일입니다. 다음 재부팅에 변경 사항이 적용되도록 하려면 **--config** 옵션을 사용합니다. 이 변경 사항을 영구적으로 사용하려면 **--persistent** 옵션을 사용합니다. 현재 도메인에 변경 사항을 적용하려면 **--current** 옵션을 사용합니다. 자세한 내용은 **Virsh man** 페이지를 참조하십시오.*

4.

*장치(**hot unplug**)를 분리하려면 다음 명령을 수행합니다.*

```
# virsh detach-device rhel6 --file usb_device.xml
```

*이 예제에서 **[rhel6]**은 게스트 가상 머신의 이름이고 **[usb_device.xml]**은 이전 단계에서 연결된 파일입니다.*

14.3. 인터페이스 장치 연결

***virsh attach-interface**도메인 유형 **source** 명령은 다음 옵션을 사용할 수 있습니다.*

- *--live - 실행 도메인에서 값 가져오기*
- *--config - 다음 부팅에서 사용할 값 가져오기*
- *--current - 현재 도메인 상태에 따라 값을 가져옵니다.*
- *--persistent - 오프라인 도메인의 **--config** 와 동일하게 작동하며, 실행 중인 도메인의 경우 **--live** 와 유사합니다.*
- *--target - 게스트 가상 머신의 대상 장치를 나타냅니다.*
- *--Mac - 이를 사용하여 네트워크 인터페이스의 **MAC** 주소를 지정합니다.*
- *--script - 이 값을 사용하여 기본 브리지 대신 브리지를 처리하는 스크립트 파일의 경로를 지정합니다.*

- ***--model -** 이를 사용하여 모델 유형을 지정합니다.*

- ***--inbound -** 인터페이스의 인바운드 대역폭을 제어합니다. 허용되는 값은 평균,최대, 버스트 입니다.*

- ***--outbound -** 인터페이스의 아웃바운드 대역폭을 제어합니다. 허용되는 값은 평균,최대, 버스트 입니다.*

*유형은 네트워크 중 하나이거나 물리적 네트워크 장치를 나타내는 브릿지 또는 장치에 브리지 를 나타낼 수 있습니다. **Source**는 장치의 소스입니다. 연결된 장치를 제거하려면 **virsh detach-device** 를 사용합니다.*

14.4. CDROM 미디어 변경

***CDROM**의 미디어를 다른 소스 또는 형식으로 변경*

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

- ***--path -** 디스크 장치의 정규화된 경로 또는 대상을 포함하는 문자열*

- ***--source -** 미디어의 소스를 포함하는 문자열*

- ***--eject -** 미디어를 제거*

- ***--insert -** 미디어를 삽입*

- ***--update -** 미디어를 업데이트*

- ***--current -** 하이퍼바이저 드라이버 구현에 따라 **--live** 및 **--config** 중 하나 또는 둘 다일 수 있습니다.*

- ***--live -** 실행 도메인의 실시간 구성 변경*

- ***--config -** 다음 부팅 시 관찰되는 영구 구성 변경*

- ***--force -** 미디어 변경을 강제 적용*

14.5. 도메인 명령

도메인 이름은 지정된 도메인을 직접 조작하기 때문에 대부분의 명령에 필요합니다. 도메인은 짧은 정수(0,1,2.), 이름 또는 전체 UUID로 지정할 수 있습니다.

14.5.1. 부팅 시 자동으로 시작하도록 도메인 구성

***$ virsh autostart [-disable]** 도메인 은 부팅 시 지정된 도메인이 자동으로 시작됩니다. **--disable** 옵션을 사용하면 **autostart**가 비활성화됩니다.*

```
# virsh autostart rhel6
```

*위의 예에서 호스트 물리적 시스템이 부팅될 때 **rhel6** 게스트 가상 시스템이 자동으로 시작됩니다.*

```
# virsh autostart rhel6 --disable
```

*위의 예에서 **autostart** 기능이 비활성화되고 호스트 물리적 시스템이 부팅될 때 **guest** 가상 머신이 자동으로 시작되지 않습니다.*

14.5.2. 게스트 가상 머신의 직렬 콘솔 연결

***$ virsh console <domain> [-devname <string>] [-force] [-safe]** 명령은 게스트 가상 머신의 가상 직렬 콘솔을 연결합니다. 선택적 **--devname <string>** 매개변수는 게스트 가상 머신에 대해 구성된 대체 콘솔, 직렬 또는 병렬 장치의 장치 별칭을 나타냅니다. 이 매개변수를 생략하면 기본 콘솔이 열립니다. **--force** 옵션은 콘솔 연결을 강제 적용하거나 연결이 끊긴 경우 연결이 끊어집니다. **--safe** 옵션을 사용하면 안전한 콘솔 처리가 지원되는 경우에만 게스트가 연결할 수 있습니다.*

```
$ virsh console virtual_machine --safe
```

14.5.3. XML 파일을 사용하여 도메인 정의

***define <FILE& gt;** 명령은 **XML** 파일의 도메인을 정의합니다. 이 경우 도메인 정의는 등록되었지만 시작되지 않습니다. 도메인이 이미 실행 중인 경우 다음 부팅에 변경 사항이 적용됩니다.*

14.5.4. 도메인의 설명 및 제목 편집 및 표시

다음 명령은 도메인의 설명 및 제목을 표시하거나 수정하는 데 사용되지만 구성하지는 않습니다.

```
# virsh desc [domain-name] [[--live] [--config] | [--current]] [--title] [--edit] [--new-desc New description or title message]
```

이러한 값은 임의의 텍스트 데이터의 저장을 허용하여 도메인을 쉽게 식별할 수 있는 사용자 필드입니다. **libvirt**에서 강제 적용하지는 않지만 제목은 짧아야 합니다.

--live 또는 **--config** 옵션은 이 명령이 도메인의 실시간 또는 영구 정의에서 작동하는지 여부를 선택합니다. **--live** 및 **--config** 를 둘 다 지정하면 명령에 입력한 설명이 라이브 구성과 영구 구성 설정에 적용되는 새 구성 설정이 되는 **--config** 옵션이 먼저 구현됩니다. **--current** 옵션은 현재 상태 구성을 수정하거나 가져오며 영구적이지 않습니다. **--live** 또는 **--config, --current** 가 지정되지 않은 경우 **--current** 옵션이 사용됩니다. **edit** 옵션은 현재 설명 또는 제목의 내용이 있는 편집기를 열고 나중에 다시 저장된 내용을 열도록 지정합니다. **title** 옵션을 사용하면 도메인의 제목 필드만 표시하거나 수정할 수 있으며 설명이 포함되지는 않습니다. 또한 명령에 **--edit** 또는 **--new-desc** 를 모두 사용하지 않으면 설명만 표시되고 수정할 수 없습니다.

예를 들어 다음 명령은 게스트 가상 머신의 제목을 **testvm** 에서 **TestVM-4F** 로 변경하고 설명을 **4**층의 **Guest VM** 으로 변경합니다.

```
$ virsh desc testvm --current --title TestVM-4F --new-desc Guest VM on fourth floor
```

14.5.5. 장치 블록 통계 표시

이 명령은 실행 중인 도메인에 대한 블록 통계를 표시합니다. 도메인 이름과 장치 이름(**virsh domblklist** 를 사용하여 장치를 나열해야 합니다.) 이 경우 블록 장치는 고유한 대상 이름(**<target dev='name'/>)** 또는 소스 파일(**< source file ='name'/>)**입니다. 모든 하이퍼바이저가 모든 필드를 표시할 수 있는 것은 아닙니다. 출력이 가장 적합한 형식으로 표시되도록 하려면 다음과 같이 **--human** 옵션을 사용합니다.

```
# virsh domblklist rhel6
Target     Source
------------------------------------------------
vda        /VirtualMachines/rhel6.img
hdc        -

# virsh domblkstat --human rhel6 vda
Device: vda
 number of read operations:      174670
 number of bytes read:           3219440128
```

```
number of write operations:     23897
number of bytes written:        164849664
number of flush operations:     11577
total duration of reads (ns):   1005410244506
total duration of writes (ns):  1085306686457
total duration of flushes (ns): 340645193294
```

14.5.6. 네트워크 통계 검색

domnetstat [domain][interface-device] 명령은 지정된 도메인에서 실행되는 지정된 장치에 대한 네트워크 인터페이스 통계를 표시합니다.

```
# domifstat rhel6 eth0
```

14.5.7. 도메인 가상 인터페이스의 링크 상태 수정

다음 명령은 지정된 인터페이스를 **up** 또는 **down**으로 구성할 수 있습니다.

```
# domif-setlink [domain][interface-device][state]{--config}
```

이를 사용하면 지정된 도메인의 지정된 인터페이스의 상태가 변경됩니다. 도메인의 영구 구성만 수정하려면 **--config**옵션을 사용해야 합니다. 또한 호환성을 이유로 **--persistent** 는 **--config** 의 별칭입니다. "인터페이스 장치"는 인터페이스의 대상 이름 또는 **MAC** 주소일 수 있습니다.

```
# domif-setlink rhel6 eth0 up
```

14.5.8. 도메인 가상 인터페이스의 링크 상태 나열

이 명령은 지정된 도메인에서 지정된 인터페이스의 상태를 쿼리하는 데 사용할 수 있습니다. 도메인의 영구 구성만 수정하려면 **--config**옵션을 사용해야 합니다. 또한 호환성을 이유로 **--persistent** 는 **--config** 의 별칭입니다. "인터페이스 장치"는 인터페이스의 대상 이름 또는 **MAC** 주소일 수 있습니다.

```
# domif-getlink rhel6 eth0 up
```

14.5.9. 네트워크 인터페이스 대역폭 매개 변수 설정

domiftune 은 게스트 가상 머신의 네트워크 인터페이스 대역폭 매개변수를 설정합니다. 다음 형식을 사용해야 합니다.

```
#virsh domiftune domain interface-device [[--config] [--live] | [--current]] [--inbound
average,peak,burst] [--outbound average,peak,burst]
```

필수 매개변수는 게스트 가상 머신의 도메인 이름 및 인터페이스 장치, **--config,--live, --current** 기능은 **14.19절. “일정 매개변수 설정”** 와 동일합니다. 제한을 지정하지 않으면 현재 네트워크 인터페이스 설정을 쿼리합니다. 그렇지 않으면 다음 옵션을 사용하여 제한을 변경합니다.

- **<interface-device>**는 필수이며 도메인의 네트워크 인터페이스의 대역폭 매개변수를 설정하거나 쿼리합니다. **interface-device** 는 인터페이스의 대상 이름(**<target dev='name'/>**) 또는 **MAC** 주소일 수 있습니다.

- **--inbound** 또는 **--outbound** 가 지정되지 않은 경우 이 명령은 대역폭 설정을 쿼리하고 표시합니다. 그렇지 않으면 인바운드 또는 아웃바운드 대역폭을 설정합니다. **average,peak,burst**는 **attach-interface** 명령과 동일합니다. 참조 **14.3절. “인터페이스 장치 연결”**

14.5.10. 실행 중인 도메인에 대한 메모리 통계 검색

이 명령은 사용하는 하이퍼바이저에 따라 다양한 결과를 반환할 수 있습니다.

dommemstat [domain] [--period (sec)][[--config][--live]|[--current]] 에는 실행 중인 도메인에 대한 메모리 통계가 표시됩니다. **--period** 옵션을 사용하려면 시간(초)이 필요합니다. 이 옵션을 **0**보다 큰 값으로 설정하면 **balloon** 드라이버에서 후속 **domemstat** 명령에서 표시할 추가 통계를 반환할 수 있습니다. **--period** 옵션을 **0**으로 설정하면 **balloon** 드라이버 컬렉션을 중지하지만 **balloon** 드라이버에서 통계를 지우지 않습니다. **balloon** 드라이버의 컬렉션 기간도 설정하기 위해 **--live,--config** 또는 **--current** 옵션도 **--period** 옵션을 설정하지 않고 사용할 수 없습니다. **--live** 옵션을 지정하면 실행 중인 게스트의 컬렉션 기간만 영향을 받습니다. **--config** 옵션을 사용하면 영구 게스트의 다음 부팅에 영향을 미칩니다. **--current** 옵션을 사용하면 현재 게스트 상태에 영향을 미칩니다.

--live 및 **--config** 옵션 둘 다 사용할 수 있지만 **--current** 는 독점적입니다. 옵션을 지정하지 않으면 게스트 상태에 따라 동작이 달라집니다.

```
#virsh domemstat rhel6 --current
```

14.5.11. 블록 장치에 오류 표시

이 명령은 **I/O** 오류로 인해 도메인이 일시 중지됨을 보고하는 **domstate** 다음에 가장 적합합니다. **domblkerror domain** 명령은 지정된 도메인에서 오류 상태에 있는 모든 블록 장치를 표시하고 장치에서 보고하는 오류 메시지를 표시합니다.

```
# virsh domblkerror rhel6
```

14.5.12. 블록 장치 크기 표시

이 경우 블록 장치는 고유한 대상 이름(<target dev='name'/>) 또는 소스 파일 (< source file ='name'/>)입니다. 목록을 검색하려면 domblklist 를 실행할 수 있습니다. 이 domblkinfo 에는 도메인 이름이 필요합니다.

```
# virsh domblkinfo rhel6
```

14.5.13. 도메인과 연결된 블록 장치 표시

domblklist domain --inactive --details 는 지정된 도메인과 연결된 모든 블록 장치의 테이블을 표시합니다.

--inactive 가 지정된 경우 결과는 다음 부팅 시 사용할 장치를 표시하고 실행 중인 도메인에서 현재 실행 중인 실행 중인 도메인에서 사용 중인 장치를 표시하지 않습니다. --details 가 지정되면 디스크 유형 및 장치 값이 테이블에 포함됩니다. 이 표에 표시된 정보는 domblkinfo 및 snapshot-create 와 함께 사용할 수 있습니다.

```
#domblklist rhel6 --details
```

14.5.14. 도메인과 연결된 가상 인터페이스 표시

domiflist 명령을 실행하면 테이블이 지정된 도메인과 연결된 모든 가상 인터페이스의 정보를 표시합니다. domiflist 에는 도메인 이름이 필요하며 선택적으로 --inactive 옵션을 사용할 수 있습니다.

--inactive 가 지정된 경우 결과는 다음 부팅 시 사용할 장치를 표시하고 실행 중인 도메인에서 현재 실행 중인 실행 중인 도메인에서 사용 중인 장치를 표시하지 않습니다.

가상 인터페이스의 MAC 주소가 필요한 명령(예: detach-interface 또는 domif-setlink)은 이 명령에 의해 표시되는 출력을 허용합니다.

14.5.15. blockcommit을 사용하여 백 엔드 체인 생성

이 섹션에서는 virsh blockcommit 을 사용하여 백업 체인을 단축하는 방법을 보여줍니다. 백업 체인에 대한 자세한 내용은 14.5.18절. “Live Block Copy를 통한 디스크 이미지 관리” 을 참조하십시오.

Blockcommit는 체인의 한 부분에서 데이터를 백업 파일로 복사하여 커밋 된 부분을 우회하기 위해 체인의 나머지 부분을 피벗할 수 있습니다. 예를 들어 현재 상태라고 가정합니다.For example, suppose this is the current state:

-

```
base ← snap1 ← snap2 ← active.
```

blockcommit 를 사용하면 **snap2**의 콘텐츠를 **snap1**로 이동하여 체인에서 **snap2**를 삭제하여 백업을 훨씬 더 빠르게 수행할 수 있습니다.

절차 **14.2. virsh blockcommit**

- 다음 명령을 실행합니다.

```
# virsh blockcommit $dom $disk -base snap1 -top snap2 -wait -verbose
```

snap2의 내용은 **snap1**로 이동하여 다음을 실행합니다.

기본 스냅**1** 활성. **snap2**는 더 이상 유효하지 않으며 삭제할 수 있습니다.

주의

blockcommit 는 **-base** 옵션에 따라 달라지는 모든 파일 (**-top** 옵션에 종속된 파일 제외)이 손상됩니다. 이를 방지하려면 둘 이상의 게스트가 공유하는 파일에 변경 사항을 커밋하지 마십시오. **-verbose** 옵션을 사용하면 화면에 진행률을 인쇄할 수 있습니다.

14.5.16. 블록pull를 사용하여 체인 백딩 단축

blockpull 는 다음 응용 프로그램에서 사용할 수 있습니다.

- 백업 이미지 체인의 데이터로 이미지를 채워서 이미지를 병합합니다. 이렇게 하면 이미지 파일이 자체 포함되므로 더 이상 백업 이미지에 의존하지 않고 다음과 같이 표시됩니다.
  - 이전: **base.img Active**
  - **after: base.img**는 더 이상 게스트에서 사용되지 않으며 **Active**는 모든 데이터를 포함합니다.

- *백업 이미지 체인의 일부를 병합합니다. 이는 스냅샷을 최상위 이미지로 병합하는 데 사용할 수 있으며 다음과 같습니다.*

  - *이전: 기본 **sn1 prep active***

  - *다음으로 **base.img** 활성 상태입니다. 이제 **active**에 **sn1** 및 **sn2**의 모든 데이터가 포함되어 있으며 게스트가 **sn1**이나 **sn2**를 사용하지 않습니다.*

- *디스크 이미지를 호스트의 새 파일 시스템으로 이동합니다. 이렇게 하면 게스트가 실행되고 있는 동안 이미지 파일을 이동할 수 있습니다.*

  - *이전 (원래 이미지 파일): **/fs1/base.vm.img***

  - *이후: **/fs2/active.vm.qcow2** 는 이제 새 파일 시스템이며 **/fs1/base.vm.img** 가 더 이상 사용되지 않습니다.*

- *사후 복사 스토리지 마이그레이션을 통한 실시간 마이그레이션에 유용합니다. 실시간 마이그레이션이 완료된 후 디스크 이미지가 소스 호스트에서 대상 호스트로 복사됩니다.*

  *즉,**/source-host/base.vm.img after:/destination-host/active.vm.qcow2./source-host/base.vm.img** 가 더 이상 사용되지 않습니다.*

*절차 **14.3.** 블록**pull**를 사용하여 체인 백딩 단축*

1. ***blockpull** 를 실행하기 전에 이 명령을 실행하는 것이 유용할 수 있습니다.*

```
# virsh snapshot-create-as $dom $name - disk-only
```

2. *체인이 다음과 같이 보이는 경우: 기본 **snap1 snap2 active active run the following:***

```
# virsh blockpull $dom $disk snap1
```

*이 명령은 **snap2**에서 활성 상태로 데이터를 가져와 **'snap1'**의 백업 파일을 활성 상태로 만듭니다.*

3.

***blockpull** 가 완료되면 체인에서 추가 이미지를 생성한 스냅샷의 **libvirt** 추적은 더 이상 유용하지 않습니다. 이 명령을 사용하여 오래된 스냅샷에서 추적을 삭제합니다.*

```
# virsh snapshot-delete $dom $name - metadata
```

***blockpull** 의 추가 애플리케이션은 다음과 같이 수행할 수 있습니다.*

- *단일 이미지를 병합하고 백업 이미지 체인의 데이터로 채우려면**# virsh blockpull example-domain vda - wait***
- *백업 이미지 체인의 일부를 병합하려면**# virsh blockpull example-domain vda -** 기본 **/path/to/base.img***
- *디스크 이미지를 호스트의 새 파일 시스템으로 이동하려면**:# virsh snapshot-create example-domaine - xmlfile /path/to/new.xml - disk-only followed by # virsh blockpull example-domain vda - wait***
- *사후 복사 스토리지 마이그레이션과 함께 실시간 마이그레이션을 사용하려면 다음을 수행합니다.*
  - *대상 실행에서 다음을 수행합니다.*

    ```
    # qemu-img create -f qcow2 -o backing_file=/source-host/vm.img /destination-host/vm.qcow2
    ```

  - *소스 실행에서 다음을 수행합니다.*

    ```
    # virsh migrate example-domain
    ```

  - *대상 실행에서 다음을 수행합니다.*

    ```
    # virsh blockpull example-domain vda - wait
    ```

14.5.17. 블록 크기를 사용하여 도메인 경로의 크기 변경

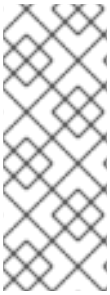
Block resize 는 도메인이 실행되는 동안 도메인의 블록 장치를 다시 조정하는 데 사용할 수 있습니다. 이 블록 장치는 고유한 대상 이름(**<target dev="name"/>**) 또는 소스 파일(**<source file="name"/>**)에 해당하는 블록 장치의 절대 경로를 사용합니다. 이는 도메인에 연결된 디스크 장치 중 하나에 적용할 수 있습니다(**domblklist** 명령을 사용하여 지정된 도메인과 연결된 모든 블록 장치의 간단한 정보를 표시하는 테이블을 인쇄할 수 있습니다).

참고

실시간 이미지 다시 지정은 항상 이미지 크기를 조정하지만 게스트에 의해 즉시 선택되지는 않을 수 있습니다. 최근 게스트 커널에서는 **virtio-blk** 장치의 크기가 자동으로 업데이트됩니다(이전 커널에는 게스트 재부팅이 필요함). **SCSI** 장치를 사용하는 경우 명령을 사용하여 게스트에서 수동으로 재 검사를 트리거해야 합니다. **echo > /sys/class/scsi_device/0:0:0:0/device/rescan.** 또한 **IDE**를 사용하면 새 크기를 선택하기 전에 게스트를 재부팅해야 합니다.

- 다음과 같은 명령을 실행합니다. **blockresize [domain] [path size] where:**
  - **domain**은 변경하려는 도메인의 고유한 대상 이름 또는 소스 파일입니다.
  - 경로 크기는 접미사가 없는 경우 기본값은 **KiB(24**바이트의 블록)입니다. 바이트를 위해 **"B"** 접미사를 사용해야 합니다.

14.5.18. Live Block Copy를 통한 디스크 이미지 관리



참고

라이브 블록 사본은 **Red Hat Enterprise Linux**에서 제공하는 **KVM** 버전에서 지원되지 않는 기능입니다. 라이브 블록 사본은 **Red Hat Virtualization**과 함께 제공되는 **KVM** 버전과 함께 사용할 수 있습니다. 기능이 지원되려면 이 버전의 **KVM**이 실제 호스트 머신에서 실행되어야 합니다. 자세한 내용은 **Red Hat** 담당자에게 문의하십시오.

라이브 블록 사본을 사용하면 게스트 디스크 이미지를 대상 이미지에 복사하고 게스트가 실행되는 동안 게스트 디스크 이미지를 대상 게스트 이미지로 전환할 수 있습니다. 실시간 마이그레이션에서는 호스트의 메모리 및 레지스트리 상태를 이동하는 동안 게스트는 공유 스토리지에 유지됩니다. 라이브 블록 복사본을 사용하면 게스트가 실행되는 동안 전체 게스트 콘텐츠를 다른 호스트로 이동할 수 있습니다. 실시간 블록 사본은 영구 공유 스토리지 없이도 실시간 마이그레이션에 사용할 수 있습니다. 이 방법에서는 게스트가 실행되는 동안 디스크 이미지가 마이그레이션 후 대상 호스트에 복사됩니다.

실시간 블록 복사는 다음 애플리케이션에 특히 유용합니다.

- *게스트 이미지를 로컬 스토리지에서 중앙 위치로 이동*
- *유지 관리가 필요한 경우 성능 손실 없이 게스트를 다른 위치로 전송할 수 있습니다.*
- *속도와 효율성을 위해 게스트 이미지 관리 가능*
- *게스트를 종료하지 않고도 이미지 형식 변환을 수행할 수 있습니다.*

예 14.1. 라이브 블록 복사 사용 예

*이 예는 라이브 블록 복사를 수행할 때 발생하는 상황을 보여줍니다. 예제에는 소스와 대상 간에 공유되는 백업 파일(**base**)이 있습니다. 또한 소스에만 존재하며 복사해야 하는 두 개의 오버레이(**sn1** 및 **sn2**)가 있습니다.*

1. *처음에 백업 파일 체인은 다음과 같습니다.*

   base** ← **sn1** ← **sn2

   구성 요소는 다음과 같습니다.

   - ***base -** 원본 디스크 이미지*
   - ***sn1 -** 기본 디스크 이미지로 가져온 첫 번째 스냅샷*
   - ***sn2 -** 최신 스냅샷*
   - ***Active -** 디스크 복사본*

2. *sn2 상단에 이미지 복사본이 새 이미지로 생성되면 다음과 같습니다.*

   base ← sn1 ← sn2 ← active

3. *이 시점에서 읽기 권한이 모두 올바른 순서로 설정되며 자동으로 설정됩니다. 쓰기 권한이 올바르게 설정되어 있는지 확인하기 위해 미러 메커니즘은 모든 쓰기를 sn2 및 활성 상태로 리디렉션하므로 언제든지 sn2 및 활성 읽기를 리디렉션합니다 (이 미러 메커니즘은 라이브 블록 복사와 이미지 스트리밍의 필수 차이점입니다).*

4. *모든 디스크 클러스터에서 반복되는 백그라운드 작업이 실행됩니다. 각 클러스터에는 다음과 같은 가능한 사례 및 작업이 있습니다.*

   - *클러스터가 이미 활성에 할당되었으므로 수행할 작업이 없습니다.*
   - *bdrv_is_allocated() 를 사용하여 백업 파일 체인을 따릅니다. 클러스터를 기본(공유)에서 읽는 경우 수행할 작업이 없습니다.*
   - *bdrv_is_allocated() 변형을 실현할 수 없는 경우, 이미지를 재베이스하고 읽기 데이터를 기반과 비교하여 복사가 필요한지 결정합니다.*
   - *다른 모든 경우 클러스터를 활성 상태로복사합니다.*

5. *복사가 완료되면 활성의 백업 파일이 base로 전환됩니다(rebase와 유사)*

일련의 스냅샷 후 백업 체인의 길이를 줄이기 위해 다음 명령이 도움이 됩니다. blockcommit 및 blockpull. 자세한 내용은 14.5.15절. "blockcommit을 사용하여 백 엔드 체인 생성" 를 참조하십시오.

14.5.19. 그래픽 디스플레이에 대한 연결 URI 표시

virsh domdisplay 명령을 실행하면 VNC, SPICE 또는 RDP를 통해 도메인의 그래픽 디스플레이에 연결하는 데 사용할 수 있는 URI가 출력됩니다. --include-password 옵션을 사용하면 SPICE 채널 암호가 URI에 포함됩니다.

14.5.20. 도메인 검색 명령

다음 명령은 지정된 도메인에 대한 다른 정보를 표시합니다.

- **virsh domhostname** 도메인 은 하이퍼바이저가 게시할 수 있는 지정된 도메인의 호스트 이름을 표시합니다.
- **virsh dominfo domain** 은 지정된 도메인에 대한 기본 정보를 표시합니다.
- **virsh domuid** 도메인|**ID** 는 지정된 도메인 이름 또는 **ID**를 **UUID**로 변환합니다.
- **virsh domid** 도메인|**ID** 는 지정된 도메인 이름 또는 **UUID**를 **ID**로 변환합니다.
- **virsh domjobabort** 도메인 은 지정된 도메인에서 현재 실행 중인 작업을 중단합니다.
- **virsh domjobinfo domain** 은 마이그레이션 통계를 포함하여 지정된 도메인에서 실행 중인 작업에 대한 정보를 표시합니다.
- **virsh domname** 도메인 **ID|UUID** 는 지정된 도메인 **ID** 또는 **UUID**를 도메인 이름으로 변환합니다.
- **virsh domstate** 도메인 은 지정된 도메인의 상태를 표시합니다. **--reason** 옵션을 사용하면 표시된 상태의 이유도 표시됩니다.
- **virsh domcontrol** 도메인 은 도메인을 제어하는 데 사용된 **VMM**에 대한 인터페이스 상태를 표시합니다. **OK** 또는 **Error**가 아닌 상태에 대해 제어 인터페이스가 표시된 상태를 입력했기 때문에 경과한 초 수를 출력합니다.

예 14.2. 통계 피드백의 예

도메인에 대한 정보를 가져오려면 다음 명령을 실행합니다.

```
# virsh domjobinfo rhel6
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
```

14.5.21. QEMU 인수에서 도메인 XML로 변환

virsh domxml-from-native 는 **libvirt**에서 사용할 수 있는 **libvirt** 도메인 **XML**을 사용하여 기존 **QEMU** 인수를 게스트 설명으로 변환할 수 있는 방법을 제공합니다. 이 명령은 **libvirt**를 통해 관리할 수 있도록 이전에 명령줄에서 시작된 기존 **qemu** 게스트를 변환하는 데만 사용됩니다. 여기에 설명된 방법은 처음부터 새 게스트를 만드는 데 사용해서는 안 됩니다. **virsh** 또는 **virt-manager**를 사용하여 새 게스트를 생성해야 합니다. 추가 정보는 여기에서 찾을 수 있습니다.

다음 **args** 파일이 있는 **QEMU** 게스트가 있다고 가정합니다.

```
 $ 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
```

libvirt에서 게스트를 관리할 수 있도록 도메인 **XML** 파일로 변환하려면 다음을 실행합니다.

```
$ virsh domxml-from-native qemu-argv demo.args
```

이 명령은 위의 **args** 파일을 이 도메인 **XML** 파일로 설정합니다.

```
<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>
```

14.5.22. 도메인 코어의 덤프 파일 생성

*경우에 따라(특히 문제 해결 시) 도메인의 코어가 포함된 덤프 파일을 생성하여 분석할 수 있도록 해야 합니다. 이 경우 **virsh dump domain corefilepath --bypass-cache --live |---crash |--reset -- verbose --verbose --memory-only dumps** 도메인 코어를 **corefilepath**에 지정된 파일에 덤프 일부 하이퍼바이저는 이 작업에 대한 제한 사항을 부여하고 사용자가 **corefilepath** 매개변수에 지정된 파일 및 경로에 대한 적절한 권한을 수동으로 확인해야 할 수 있습니다. 이 명령은 **SR-IOV** 장치 및 기타 패스스루 장치에서 지원됩니다. 다음 옵션이 지원되며 다음과 같은 효과가 있습니다.*

- *__--bypass-cache__ 파일에 파일 시스템 캐시가 없습니다. 이 옵션을 선택하면 덤프 작업이 느려질 수 있습니다.*
- *__--live__ 는 도메인이 계속 실행될 때 파일을 저장하고 도메인을 일시 중지하거나 중지하지 않습니다.*
- *__--crash__ 는 덤프 파일이 저장되는 동안 일시 정지된 상태로 두는 대신 도메인을 크래시 상태로 둡니다.*
- *__--reset__ 이 덤프 파일이 성공적으로 저장되면 도메인이 재설정됩니다.*
- *__--verbose__ 에서 덤프 프로세스의 진행 상황을 표시*
- *__--memory-only__ 덤프 파일에 저장되는 유일한 정보는 도메인의 메모리와 **CPU** 공통 레지스터 파일입니다.*

domjobinfo 명령을 사용하여 전체 프로세스를 모니터링할 수 있으며 **domjobabort** 명령을 사용하여 취소할 수 있습니다.

14.5.23. 가상 머신 XML 덤프 생성 (구성 파일)

virsh 로 게스트 가상 머신의 **XML** 구성 파일을 출력합니다.

```
# virsh dumpxml {guest-id, guestname or uuid}
```

이 명령은 게스트 가상 머신의 **XML** 구성 파일을 표준 아웃**(stdout)**으로 출력합니다. 출력을 파일에 파이핑하여 데이터를 저장할 수 있습니다. 출력을 **guest.xml** 이라는 파일에 파이핑하는 예:

```
# virsh dumpxml GuestID > guest.xml
```

이 파일 **guest.xml** 은 게스트 가상 머신을 다시 생성할 수 있습니다(14.6절. “게스트 가상 머신의 구성 파일 편집” 참조). 이 **XML** 구성 파일을 편집하여 추가 장치를 구성하거나 추가 게스트 가상 머신을 배포할 수 있습니다.

virsh dumpxml 출력 예:

```
# virsh dumpxml guest1-rhel6-64
<domain type='kvm'>
  <name>guest1-rhel6-64</name>
  <uuid>b8d7388a-bbf2-db3a-e962-b97ca6e514bd</uuid>
  <memory>2097152</memory>
  <currentMemory>2097152</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
```

```
    <driver name='qemu' type='raw' cache='none' io='threads'/>
    <source file='/home/guest-images/guest1-rhel6-64.img'/>
    <target dev='vda' bus='virtio'/>
    <shareable/<
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
  </disk>
  <interface type='bridge'>
    <mac address='52:54:00:b9:35:a9'/>
    <source bridge='br0'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <serial type='pty'>
    <target port='0'/>
  </serial>
  <console type='pty'>
    <target type='serial' port='0'/>
  </console>
  <input type='tablet' bus='usb'/>
  <input type='mouse' bus='ps2'/>
  <graphics type='vnc' port='-1' autoport='yes'/>
  <sound model='ich6'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
  </sound>
  <video>
    <model type='cirrus' vram='9216' heads='1'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
  </video>
  <memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
  </memballoon>
 </devices>
</domain>
```

<shareable/> 플래그가 설정되어 있습니다. 이는 도메인 간에 장치가 공유될 것으로 예상되며(하이퍼바이저와 **OS** 지원의 경우) 해당 장치에 대해 캐싱을 비활성화해야 함을 의미합니다.

14.5.24. 구성 파일에서 게스트 가상 머신 생성

게스트 가상 머신은 **XML** 구성 파일에서 생성할 수 있습니다. 이전에 생성된 게스트 가상 머신에서 기존 **XML**을 복사하거나 **dumpxml** 옵션(14.5.23절. “가상 머신 XML 덤프 생성 (구성 파일)”참조)을 사용할 수 있습니다. **XML** 파일에서 **virsh** 를 사용하여 게스트 가상 머신을 생성하려면 다음을 수행합니다.

```
# virsh create configuration_file.xml
```

14.6. 게스트 가상 머신의 구성 파일 편집

dumpxml 옵션(14.5.23절. “가상 머신 XML 덤프 생성 (구성 파일)”참조)을 사용하는 대신 게스트 가상 머신을 실행 중이나 오프라인 상태에서 편집할 수 있습니다. **virsh edit** 명령은 이 기능을 제공합니다. 예

*를 들어 **rhel6** 이라는 게스트 가상 머신을 편집하려면 다음을 수행합니다.*

```
# virsh edit rhel6
```

*그러면 텍스트 편집기가 열립니다. 기본 텍스트 편집기는 **$EDITOR** 쉘 매개변수입니다(기본적으로 **vi** 로 설정).*

14.6.1. KVM 게스트 가상 머신에 Multifunction PCI 장치 추가

*이 섹션에서는 **KVM** 게스트 가상 머신에 다중 기능 **PCI** 장치를 추가하는 방법을 보여줍니다.*

1. ***virsh edit [guestname]** 명령을 실행하여 게스트 가상 시스템의 **XML** 구성 파일을 편집합니다.*

2. ***address** 유형 태그에서 **function='0x0'** 에 대해 **multifunction='on'** 항목을 추가합니다.*

   *이를 통해 게스트 가상 머신은 다기능 **PCI** 장치를 사용할 수 있습니다.*

```
<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>
```

*두 개의 기능이 있는 **PCI** 장치의 경우 첫 번째 장치와 동일한 슬롯 번호와 **function='0x1'** 과 같은 다른 함수 번호를 사용하여 두 번째 장치를 포함하도록 **XML** 구성 파일을 수정합니다.*

예를 들면 다음과 같습니다.

```
<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. **KVM** 게스트 가상 머신의 **lspci** 출력에는 다음이 표시됩니다.

```
$ 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
```

14.6.2. 실행 중인 도메인을 중지하여 다시 시작합니다.

virsh managedsave domain --bypass-cache --running | --verbose saves and destroys (stop) a running domain so that it can be restarted from the same state at a later time. virsh start 명령과 함께 사용하면 이 저장 지점에서 자동으로 시작됩니다. **--bypass-cache** 옵션과 함께 사용되는 경우 저장 시 파일 시스템 캐시가 발생하지 않습니다. 이 옵션은 저장 프로세스 속도가 느려질 수 있습니다.

--verbose 에서 덤프 프로세스의 진행 상황을 표시

정상적인 조건에서 관리 저장은 저장 작업이 완료될 때 도메인의 상태에 따라 결정되는 대로 실행 중이거나 일시 중지된 상태를 결정합니다. 그러나 **--running** 옵션을 사용하여 이 상태를 실행 중 상태로 유지해야 함을 나타내거나 일시 중지됨을 나타내는 **--paused** 옵션을 사용하여 재정의할 수 있습니다.

관리형 저장 상태를 제거하려면 다음에 시작할 때 도메인을 강제로 전체 부팅하도록 하는 **virsh managedsave-remove** 명령을 사용합니다.

domjobinfo 명령을 사용하여 전체 관리 저장 프로세스를 모니터링할 수 있으며 **domjobabort** 명령을 사용하여 취소할 수도 있습니다.

14.6.3. 지정된 도메인에 대한 CPU 통계 표시

virsh cpu-stats 도메인 **--total start count** 명령은 지정된 도메인에 대한 **CPU** 통계 정보를 제공합니다. 기본적으로 모든 **CPU** 및 총 **CPU**의 통계가 표시됩니다. **--total** 옵션은 총 통계만 표시합니다.

14.6.4. 스크린샷 저장

virsh screenshot 명령은 현재 도메인 콘솔의 스크린샷을 가져와서 파일에 저장합니다. 그러나 하이퍼바이저가 도메인에 대해 더 많은 디스플레이를 지원하는 경우 **--screen** 을 사용하여 화면 **ID**를 제공하

면 캡처할 화면을 지정합니다. 여러 개의 그래픽 카드가 있는 경우 헤드가 장치 전에 번호 매겨진 경우 화면 ID 5는 두 번째 카드의 두 번째 헤드를 처리합니다.

14.6.5. 지정된 도메인으로 Keystroke Combination 전송

virsh send-key domain --codeset --holdtime keycode 명령을 사용하면 시퀀스를 특정 도메인에 키 코드로 보낼 수 있습니다.

각 키 코드는 해당 코드 집합 의 숫자 값 또는 심볼릭 이름일 수 있습니다. 여러 키 코드가 지정되면 thay는 모두 게스트 가상 머신으로 동시에 전송되므로 임의의 순서로 수신될 수 있습니다. 고유한 키 코드가 필요한 경우 send-key 명령을 여러 번 보내야 합니다.

```
# virsh send-key rhel6 --holdtime 1000 0xf
```

--holdtime 을 제공하면 각 키 입력이 지정된 밀리초 단위로 유지됩니다. codeset 을 사용하면 코드 세트를 지정할 수 있으며 기본값은 Linux이지만 다음 옵션은 허용됩니다.

- Linux - 이 옵션을 선택하면 심볼릭 이름이 해당 Linux 키의 일관된 매크로 이름과 일치하도록 하며 숫자 값은 Linux 일반 입력 이벤트 하위 시스템에서 제공합니다.
- XT - XT 키보드 컨트롤러에서 정의한 값을 보냅니다. 심볼릭 이름은 제공되지 않습니다.
- atset1 - 숫자 값은 AT 키보드 컨트롤러, set1(XT 호환 세트)에서 정의한 값입니다. atset1에서 확장된 키 코드는 XT 코드 집합의 확장 키 코드와 다를 수 있습니다. 심볼릭 이름은 제공되지 않습니다.
- atset2 - 숫자 값은 AT 키보드 컨트롤러에서 정의한 값이며 2를 설정합니다. 심볼릭 이름은 제공되지 않습니다.
- atset3 - 숫자 값은 AT 키보드 컨트롤러에서 정의한 값이며 3(PS/2와 호환됨)을 설정합니다. 심볼릭 이름은 제공되지 않습니다.
- os_x - 숫자 값은 OS-X 키보드 입력 하위 시스템에서 정의합니다. 심볼릭 이름은 해당 OS-X 키의 상수 매크로 이름과 일치합니다.
- X t_kbd - 숫자 값은 Linux KBD 장치에서 정의하는 값입니다. 이는 원래 XT 코드 세트에서

변형이지만 종종 확장된 키 코드에 대한 다른 인코딩을 사용하는 경우가 많습니다. 심볼릭 이름은 제공되지 않습니다.

- **win32 -** 숫자 값은 **Win32** 키보드 입력 하위 시스템에서 정의한 값입니다. 심볼릭 이름은 해당 **Win32** 키 상수 매크로 이름과 일치합니다.

- **USB -** 숫자 값은 키보드 입력을 위해 **USB HID** 사양에 의해 정의된 값입니다. 심볼릭 이름은 제공되지 않습니다.

- **RF B -** 숫자 값은 원시 키 코드를 전송하기 위해 **RFB** 확장에 의해 정의된 값입니다. 이는 **XT** 코드 세트에서 변형이지만 확장 키코드는 첫 번째 바이트의 높은 비트 대신 두 번째 비트 세트의 낮은 비트를 갖습니다. 심볼릭 이름은 제공되지 않습니다.

14.6.6. 가상 프로세스에 프로세스 신호 이름 전송

virsh send-process-signal domain-ID PID signame 명령을 사용하면 지정된 신호(고정 기호로 식별됨)를 가상 도메인(도메인 **ID**에 의해 지정됨)으로 실행되고 해당 프로세스 **ID(PID)**로 식별됩니다.

정수 신호 상수 번호 또는 심볼릭 신호 이름은 이러한 방식으로 보낼 수 있습니다. 따라서 예를 들어 다음 두 명령은 **rhel6** 도메인의 **ID(187)**에 종료 신호를 보냅니다.

```
# virsh send-process-signal rhel6 187 kill
# virsh send-process-signal rhel6 187 9
```

사용 가능한 신호 및 용도의 전체 목록은 **virsh(1)** 및 **signal(7)** 매뉴얼 페이지를 참조하십시오.

14.6.7. VNC 디스플레이의 IP 주소 및 포트 번호 표시

virsh vncdisplay 는 지정된 도메인에 대해 **VNC** 디스플레이의 **IP** 주소와 포트 번호를 출력합니다. 정보를 사용할 수 없는 경우 종료 코드 **1**이 표시됩니다.

```
# virsh vncdisplay rhel6
127.0.0.1:0
```

14.7. NUMA 노드 관리

이 섹션에는 **NUMA** 노드 관리에 필요한 명령이 포함되어 있습니다.

14.7.1. 노드 정보 표시

nodeinfo 명령은 모델 번호, **CPU** 수, **CPU** 유형, 실제 메모리 크기를 포함하여 노드에 대한 기본 정보를 표시합니다. 출력은 **virNodeInfo** 구조에 해당합니다. 특히 **"CPU** 소켓**"** 필드는 **NUMA** 셀당 **CPU** 소켓 수를 나타냅니다.

```
$ 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
```

14.7.2. NUMA 매개변수 설정

virsh numatune 은 지정된 도메인의 **NUMA** 매개 변수를 설정하거나 검색할 수 있습니다. **Domain XML** 파일 내에서 이러한 매개변수는 **<numatune>** 요소 내에 중첩됩니다. 옵션을 사용하지 않으면 현재 설정만 표시됩니다. **numatune domain** 명령에는 지정된 도메인이 필요하며 다음 옵션을 사용할 수 있습니다.

- **--mode -** 모드를 **strict,interleave** 또는 **preferred** 로 설정할 수 있습니다. 도메인이 엄격한 모드 내에서 시작되지 않는 한, 실행 중인 도메인은 실시간 중에 모드를 변경할 수 없습니다.
- **--nodeset** 에는 호스트 물리적 시스템에서 도메인을 실행하는 데 사용하는 **NUMA** 노드 목록이 포함되어 있습니다. 목록에는 각 노드가 쉼표로 구분되어 있으며 노드 범위에 대시를 사용하고 노드를 제외하는 데 사용되는 **caret ^** 이 포함됩니다.
- 다음 세 가지 옵션 중 하나만 인스턴스당 사용할 수 있습니다.
  - **--config** 는 영구 게스트 가상 머신의 다음 부팅에 적용됩니다.
  - **--live** 는 실행 중인 게스트 가상 머신의 스케줄러 정보를 설정합니다.
  - **--current** 는 게스트 가상 머신의 현재 상태에 영향을 미칩니다.

14.7.3. NUMA 셀에 사용 가능한 메모리 마운트 표시

virsh freecell 은 지정된 **NUMA** 셀 내에 시스템에서 사용 가능한 메모리 양을 표시합니다. 이 명령은 지정된 옵션에 따라 시스템에서 사용 가능한 메모리 **3**개 중 하나를 제공할 수 있습니다. 옵션을 사용하지 않으면 시스템의 사용 가능한 총 메모리가 표시됩니다. **all** 옵션을 사용하면 각 셀에 사용 가능한 메모리와 시스템의 사용 가능한 총 메모리가 표시됩니다. 숫자 인수를 사용하거나 셀 번호와 함께 **--cellno** 옵션과 함께 사용하면 지정된 셀에 사용 가능한 메모리가 표시됩니다.

14.7.4. CPU 목록 표시

nodecpumap 명령은 노드에서 사용할 수 있는 **CPU** 수를 표시합니다(온라인이든 아님) 및 현재 온라인에 있는 수도 나열합니다.

```
$ virsh nodecpumap
  CPUs present: 4
  CPUs online: 1
  CPU map: y
```

14.7.5. CPU 통계 표시

CPU가 제공되는 경우 **nodecpustats** 명령은 지정된 **CPU**에 대한 통계 정보를 표시합니다. 그러지 않으면 노드의 **CPU** 상태를 표시합니다. 백분율을 지정하면 **1**초 간격으로 기록된 각 **CPU** 통계 유형의 백분율을 표시합니다.

이 예에서는 지정된 **CPU**를 표시하지 않습니다.

```
$ virsh nodecpustats
user:                1056442260000000
system:               401675280000000
idle:                7549613380000000
iowait:                94593570000000
```

이 예에서는 **CPU** 번호 **2**에 대한 통계 백분율을 보여줍니다.

```
$ virsh nodecpustats 2 --percent
usage:          2.0%
user:           1.0%
system:         1.0%
idle:          98.0%
iowait:         0.0%
```

게스트 가상 시스템의 구성 파일에서 **on_reboot** 요소를 수정하여 재부팅 게스트 가상 머신의 동작을 제어할 수 있습니다.

14.7.6. 호스트 물리적 시스템 일시 중단

nodesuspend 명령은 **Suspend-to-RAM(s3), Suspend-to-Disk(s4)** 또는 **Hybrid-Suspend**의 경우와 유사하게 호스트 물리적 머신을 시스템 전체의 절전 상태에 배치하고 이전에 설정된 기간이 지난 후 노드를 중단하도록 **Real-Time-Clock**을 설정합니다. **--target** 옵션은 **mem,disk** 또는 **hybrid** 로 설정할 수 있습니다. 이러한 옵션은 일시 중단할 메모리, 디스크 또는 둘의 조합을 설정하도록 나타냅니다. **--duration**를 설정하면 설정된 기간이 만료된 후 호스트 물리적 시스템이 중단되도록 지시합니다. 몇 초 내에 설정됩니다. 기간 시간은 **60**초를 초과할 것을 권장합니다.

```
$ virsh nodesuspend disk 60
```

14.7.7. 노드 메모리 매개변수 설정 및 표시

node-memory-tune [shm-pages-to-scan] [shm-sleep-milisecs] [shm-merge-across-nodes] 명령이 표시되고 노드 메모리 매개변수를 설정할 수 있습니다. 다음 명령으로 설정할 수 있는 세 가지 매개변수가 있습니다.

- **shm-pages-to-scan -** 공유 메모리 서비스가 잠자기 상태로 전환되기 전에 검사할 페이지 수를 설정합니다.
- **shm-sleep-milisecs -** 공유 메모리 서비스가 다음 검사 전에 절전하게 될 밀리초 수를 설정합니다.
- **shm-merge-across-nodes -** 여러 **NUMA** 노드의 페이지를 병합할 수 있는지 지정합니다. 허용되는 값은 **0** 및 **1** 입니다. **0** 으로 설정하면 병합할 수 있는 유일한 페이지는 동일한 **NUMA** 노드의 메모리 영역에 물리적으로 상주하는 페이지입니다. **1** 로 설정하면 모든 **NUMA** 노드의 페이지를 병합할 수 있습니다. 기본 설정은 **1** 입니다.

14.7.8. 호스트 노드에서 장치 생성

virsh nodedev-create file 명령을 사용하면 호스트 노드에서 장치를 생성한 다음 게스트 가상 머신에 할당할 수 있습니다. **libvirt** 는 일반적으로 자동으로 사용할 수 있는 호스트 노드를 감지하지만, 이 명령을 사용하면 **libvirt** 에서 감지하지 않은 호스트 하드웨어를 등록할 수 있습니다. 파일에 는 노드 **<장치의 최상위 장치>** 설명에 대한 **XML**이 포함되어야 합니다.

이 장치를 중지하려면 **nodedev-destroy device** 명령을 사용합니다.

14.7.9. 노드 장치 분리

virsh nodedev-detach 는 호스트에서 **nodedev**를 분리하므로 **<hostdev>** 패스스루를 통해 게스트가 안전하게 사용할 수 있습니다. 이 작업은 **nodedev-reattach** 명령을 사용하여 취소할 수 있지만 관리 서비스에 대해 자동으로 수행됩니다. 이 명령은 **nodedev-dettach** 도 허용합니다.

다른 드라이버는 장치가 다른 더미 장치에 바인딩될 것으로 예상합니다. 드라이버 옵션을 사용하면 원하는 백엔드 드라이버를 지정할 수 있습니다.

14.7.10. 장치의 구성 설정 검색

virsh nodedev-dumpxml [device] 명령은 지정된 노드 **<장치에>** 대한 **XML** 구성 파일을 덤프합니다. **XML** 구성에는 장치 이름(예: 장치, 공급업체, 제품 **ID**)과 같은 정보가 포함됩니다. 인수 장치는 **WWNN | WWPN** 형식(**HBA**만 해당)의 장치 이름 또는 **WWN** 쌍일 수 있습니다.

14.7.11. 노드에 장치 나열

virsh nodedev-list cap --tree 명령은 **libvirt** 에서 알려진 노드에서 사용 가능한 모든 장치를 나열합니다. **cap** 는 기능 유형별로 목록을 필터링하는 데 사용되며, 각각 쉼표로 구분되며 **--tree** 와 함께 사용할 수 없습니다. **--tree** 옵션을 사용하여 다음과 같이 출력을 트리 구조로 둡니다.

```
 #  virsh nodedev-list --tree
 computer
|
+- net_lo_00_00_00_00_00_00
+- net_macvtap0_52_54_00_12_fe_50
+- net_tun0
+- net_virbr0_nic_52_54_00_03_7d_cb
+- pci_0000_00_00_0
+- pci_0000_00_02_0
+- pci_0000_00_16_0
+- pci_0000_00_19_0
|   |
|   +- net_eth0_f0_de_f1_3a_35_4f

(this is a partial screen)
```

14.7.12. 노드의 재설정 트리거

nodedev-reset nodedev 명령은 지정된 **nodedev** 에 대한 장치 재설정을 트리거합니다. 이 명령을 실행하면 게스트 가상 머신 통과와 호스트 물리적 시스템 간에 노드 장치를 전송하기 전에 유용합니다. **libvirt** 는 필요한 경우 이 작업을 암시적으로 수행하지만, 필요한 경우 이 명령을 사용하면 명시적으로 재설정할 수 있습니다.

14.8. 게스트 가상 머신 시작, SUSPENDING, RESUMING, SAVING 및 RESTORING

이 섹션에서는 게스트 가상 머신 시작, 일시 중지, 재시작, 저장 및 복원하는 방법에 대한 정보를 제공합니다.

14.8.1. 정의된 도메인 시작

virsh start domain --console --console --paused --autodestroy --bypass-cache --force-boot --pass-fds 명령은 이미 정의되었지만 누가 마지막 관리 저장 상태 또는 새 부팅 이후 비활성 도메인을 시작합니다. 명령은 다음 옵션을 사용할 수 있습니다.

- **--console -** 콘솔에 도메인 연결을 부팅합니다.
- **--paused -** 드라이버에서 이를 지원하는 경우 도메인을 부팅한 다음 일시 정지 상태로 전환합니다.
- **--autodestroy - guest** 가상 머신은 **virsh** 세션이 닫히거나 **libvirt** 연결이 닫히거나 그렇지 않으면 종료될 때 자동으로 삭제됩니다.
- **--bypass-cache -** 도메인이 관리형**save** 상태인 경우 사용됩니다. 이 기능을 사용하면 게스트 가상 머신을 복원하여 시스템 캐시를 피할 수 있습니다. 이 경우 복원 프로세스가 느려집니다.
- **--force-boot -** 관리 저장 옵션을 폐기하고 새로운 부팅이 발생하는 경우
- **--pass-fds -**는 쉼표로 구분된 추가 옵션 목록으로, 게스트 가상 머신에 전달됩니다.

14.8.2. 게스트 가상 머신 일시 중단

virsh 를 사용하여 게스트 가상 머신 일시 중단:

```
# virsh suspend {domain-id, domain-name or domain-uuid}
```

게스트 가상 머신이 일시 중단된 상태인 경우 시스템 **RAM**을 사용하지만 프로세서 리소스는 사용하지 않습니다. 게스트 가상 머신이 일시 중지되는 동안 디스크 및 네트워크 **I/O**가 발생하지 않습니다. 이 작업은 즉각적이고 게스트 가상 머신을 재개 (**14.8.6**절. "게스트 가상 머신 재시작") 옵션으로 다시 시작할 수 있습니다.

14.8.3. 실행 중인 도메인 일시 중단

virsh dompmsuspend 도메인 **--duration --target** 명령은 실행 중인 도메인을 사용하고 일시 중지되므로 세 가지 상태(**S3, S4** 또는 하이브리드) 중 하나에 배치할 수 있습니다.

```
# virsh dompmsuspend rhel6 --duration 100 --target mem
```

이 명령은 다음 옵션을 사용할 수 있습니다.

- **--duration -** 상태 변경 시간(초)을 설정합니다.
- **--target - mem(S3)** 디스크(**suspend todisk (S4)**) 또는 하이브리드(**hybrid suspend**)일 수 있습니다.

14.8.4. pmsuspend 상태에서 도메인 시작

이 명령은 만료되도록 설정된 기간 동안 대기하지 않고 **pmsuspend** 상태에 있는 게스트에 경고를 삽입합니다. 도메인이 실행 중이면 이 작업이 실패하지 않습니다.

```
# dompmwakeup rhel6
```

이 명령에는 표시된 대로 도메인 이름이 필요합니다(예: **rhel 6**).

14.8.5. 도메인 정의 취소

```
# virsh undefine domain --managed-save --snapshots-metadata --storage --remove-all-storage --
wipe-storage
```

이 명령은 도메인 정의를 해제합니다. 실행 중인 도메인에서 작동할 수 있지만 실행 중인 도메인을 중지하지 않고 임시 도메인으로 변환합니다. 도메인이 비활성 상태이면 도메인 구성이 제거됩니다.

명령은 다음 옵션을 사용할 수 있습니다.

- **--managed-save -** 이 옵션을 사용하면 관리 저장소 이미지도 정리됩니다. 이 옵션을 사용하지 않으면 관리형 저장 이미지로 도메인을 정의 해제하려고 하면 실패합니다.

- **--snapshots-metadata -** 이 옵션을 사용하면 비활성 도메인을 정의할 때 스냅샷(**snapshot-list**와 같이)도 정리됩니다. 구성 파일에 스냅샷 메타데이터가 포함된 비활성 도메인을 정의 해제하려고 하면 실패합니다. 이 옵션이 사용되고 도메인이 활성 상태이면 무시됩니다.

- **--storage -** 이 옵션을 사용하려면 정의되지 않은 도메인과 함께 제거할 볼륨 대상 이름 또는 스토리지 볼륨의 소스 경로 목록이 쉼표로 구분되어 있어야 합니다. 이 작업은 스토리지 볼륨이 제거되기 전에 정의 해제됩니다. 이 작업은 비활성 도메인에서만 수행할 수 있습니다. 또한 **libvirt** 에서 관리하는 스토리지 볼륨에서만 작동합니다.

- **--remove-all-storage -** 도메인 정의 외에 연결된 모든 스토리지 볼륨이 삭제됩니다.

- **--wipe-storage -** 스토리지 볼륨을 삭제하는 것 외에도 콘텐츠가 지워집니다.

14.8.6. 게스트 가상 머신 재시작

resume 옵션을 사용하여 **virsh** 로 일시 중단된 게스트 가상 시스템을 복원합니다.

```
# virsh resume {domain-id, domain-name or domain-uuid}
```

이 작업은 즉시 수행되며 게스트 가상 머신 매개변수는 일시 중단 및 재개 작업을 위해 유지됩니다.

14.8.7. 게스트 가상 머신 저장

virsh 명령을 사용하여 **guest** 가상 머신의 현재 상태를 파일에 저장합니다.

```
# virsh save {domain-name|domain-id|domain-uuid} state-file --bypass-cache --xml --running --
paused --verbose
```

사용자가 지정한 게스트 가상 머신이 중지되고 파일에 데이터를 저장하면 게스트 가상 머신에서 사용 중인 메모리 양이 다소 걸릴 수 있습니다. 복원(**14.8.11절. "게스트 가상 머신 복원"**) 옵션을 사용하여 게스트 가상 머신의 상태를 복원할 수 있습니다. 저장은 게스트 가상 머신을 일시 중지하지 않고 게스트 가상 머신의 현재 상태를 저장하는 대신 일시 중지와 유사합니다.

virsh save 명령은 다음 옵션을 사용할 수 있습니다.

- **--bypass-cache -** 복원으로 인해 파일 시스템 캐시가 발생하지 않지만 이 옵션을 사용하면

복원 작업이 느려질 수 있습니다.

- **--XML -** 이 옵션을 **XML** 파일 이름과 함께 사용해야 합니다. 이 옵션은 일반적으로 생략되지만 복원된 게스트 가상 머신에서 도메인 **XML**의 호스트별 부분에만 변경 사항이 있는 대체 **XML** 파일을 제공하는 데 사용할 수 있습니다. 예를 들어 게스트를 저장한 후 가져온 디스크 스냅샷으로 인해 기본 스토리지의 파일 이름 차이를 고려할 수 있습니다.

- **--running -** 저장 이미지에 기록된 상태를 재정의하여 도메인을 실행 중으로 시작합니다.

- **--paused-** 저장 이미지에 기록된 상태를 재정의하여 도메인을 일시 중지한 상태로 시작합니다.

- **--verbose -** 저장 진행률을 표시합니다.

XML 파일에서 직접 게스트 가상 머신을 복원하려면 **virsh restore** 명령이 바로 이를 수행합니다. **domjobinfo** 를 사용하여 프로세스를 모니터링하고 **domjobabort** 를 사용하여 취소할 수 있습니다.

14.8.8. 게스트 복구에 사용할 도메인 XML 파일 업데이트

virsh save-image-define 파일 **xml --running|--paused** 명령은 지정된 파일이 나중에 **virsh restore** 명령 중에 사용될 때 사용되는 도메인 **XML** 파일을 업데이트합니다. **xml** 인수는 도메인 **XML**의 호스트 물리적 시스템 특정 부분에서만 변경 사항이 있는 대체 **XML**이 포함된 **XML** 파일 이름이어야 합니다. 예를 들어 게스트를 저장한 후 기본 스토리지의 디스크 스냅샷을 생성하여 파일 이름 지정 차이를 고려할 수 있습니다. 도메인을 실행 중이거나 일시 정지된 상태로 복원해야 하는 경우 저장 이미지 레코드입니다. **--running** 또는 **--paused** 옵션을 사용하면 사용할 상태가 됩니다.

14.8.9. 도메인 XML 파일 추출

save-image-dumpxml 파일 **--security-info** 명령은 저장된 상태 파일(**virsh save** 명령에서 사용)을 참조할 때 적용되는 도메인 **XML** 파일을 추출합니다. **security-info** 옵션을 사용하면 파일에 보안에 중요한 정보가 포함됩니다.

14.8.10. 도메인 XML 구성 파일 편집

save-image-edit file --running --paused 명령은 **virsh save** 명령으로 생성된 저장된 파일과 관련된 **XML** 구성 파일을 편집합니다.

*저장 이미지 레코드는 도메인을 **--running** 또는 **--paused** 상태로 복원해야 하는지 여부를 기록합니다. 이러한 옵션을 사용하지 않으면 파일 자체에 의해 상태가 결정됩니다. **--running** 또는 **--paused** 를 선택하면 **virsh restore** 에서 사용할 상태를 덮어쓸 수 있습니다.*

14.8.11. 게스트 가상 머신 복원

***virsh**를 사용하여 **virsh save** 명령(14.8.7절. "게스트 가상 머신 저장")로 이전에 저장된 게스트 가상 머신을 복원합니다.*

```
# virsh restore state-file
```

*이렇게 하면 저장된 게스트 가상 머신이 다시 시작되므로 다소 시간이 걸릴 수 있습니다. 게스트 가상 머신의 이름과 **UUID**는 보존되지만 새 **ID**에 할당됩니다.*

***virsh restore state-file** 명령은 다음 옵션을 사용할 수 있습니다.*

- ***--bypass-cache -** 복원으로 인해 파일 시스템 캐시가 발생하지 않지만 이 옵션을 사용하면 복원 작업이 느려질 수 있습니다.*
- ***--XML -** 이 옵션을 **XML** 파일 이름과 함께 사용해야 합니다. 이 옵션은 일반적으로 생략되지만 복원된 게스트 가상 머신에서 도메인 **XML**의 호스트별 부분에만 변경 사항이 있는 대체 **XML** 파일을 제공하는 데 사용할 수 있습니다. 예를 들어 게스트를 저장한 후 가져온 디스크 스냅샷으로 인해 기본 스토리지의 파일 이름 차이를 고려할 수 있습니다.*
- ***--running -** 저장 이미지에 기록된 상태를 재정의하여 도메인을 실행 중으로 시작합니다.*
- ***--paused-** 저장 이미지에 기록된 상태를 재정의하여 도메인을 일시 중지한 상태로 시작합니다.*

14.9. 게스트 가상 시스템의 종료, 재부팅 및 종료

이 섹션에서는 게스트 가상 머신 종료, 재부팅 및 강제 종료하는 방법에 대한 정보를 제공합니다.

14.9.1. 게스트 가상 머신 종료

virsh shutdown 명령을 사용하여 게스트 가상 머신을 종료합니다.

```
# virsh shutdown {domain-id, domain-name or domain-uuid} [--mode method]
```

게스트 가상 시스템의 구성 파일에서 **on_shutdown** 매개 변수를 수정하여 재부팅 게스트 가상 머신의 동작을 제어할 수 있습니다.

14.9.2. Red Hat Enterprise Linux 7 호스트에서 Red Hat Enterprise Linux 6 게스트 종료

최소 설치 옵션을 사용하여 **Red Hat Enterprise Linux 6** 게스트 가상 머신을 설치하면 **acpid** 패키지를 설치 하지 않습니다. **Red Hat Enterprise Linux 7**은 **systemd** 에 의해 수행되었기 때문에 더 이상 이 패키지가 필요하지 않습니다. 그러나 **Red Hat Enterprise Linux 7** 호스트에서 실행되는 **Red Hat Enterprise Linux 6** 게스트 가상 머신에는 여전히 필요합니다.

acpid 패키지가 없으면 **virsh shutdown** 명령을 실행할 때 **Red Hat Enterprise Linux 6** 게스트 가상 머신이 종료되지 않습니다. **virsh shutdown** 명령은 게스트 가상 머신을 정상적으로 종료하도록 설계되었습니다.

virsh 종료 를 사용하는 것은 시스템 관리에서 더 쉽고 안전합니다. **virsh shutdown** 명령을 사용하여 정상적으로 종료하지 않으면 시스템 관리자가 게스트 가상 시스템에 수동으로 로그인하거나 **Ctrl-Alt-Del** 키 조합을 각 게스트 가상 시스템에 보내야 합니다.

참고

다른 가상화된 운영 체제는 이 문제의 영향을 받을 수 있습니다. **virsh shutdown** 명령을 사용하려면 **guest** 가상 시스템 운영 체제가 **ACPI** 종료 요청을 처리하도록 구성되어 있어야 합니다. 대부분의 운영 체제에는 **ACPI** 종료 요청을 수락하려면 게스트 가상 시스템 운영 체제에 대한 추가 구성이 필요합니다.

절차 **14.4. Red Hat Enterprise Linux 6** 게스트의 해결 방법

1. **acpid** 패키지 설치

   acpid 서비스는 **ACPI** 요청을 수신하고 처리합니다.

   게스트 가상 머신에 로그인하고 게스트 가상 머신에 **acpid** 패키지를 설치합니다.

   ```
   # yum install acpid
   ```

2. **acpid** 서비스 활성화

   게스트 가상 머신 부팅 순서 중에 **acpid** 서비스를 설정하고 서비스를 시작합니다.

```
# chkconfig acpid on
# service acpid start
```

3. 게스트 도메인 **xml** 준비

   다음 요소를 포함하도록 도메인 **XML** 파일을 편집합니다. **virtio** 직렬 포트를 **org.qemu.guest_agent.0** 으로 교체하고 **$guestname**대신 게스트 이름을 사용합니다.

   그림 **14.2.** 게스트 **XML** 대체

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

4. **QEMU** 게스트 에이전트 설치

   QEMU 게스트 에이전트**(QEMU-GA)**를 설치하고 **10**장**. QEMU-img** 및 **QEMU** 게스트 에이전트 에서 지시한 대로 서비스를 시작합니다. **Windows** 게스트를 실행하는 경우 이 장에도 이에 대한 지침이 있습니다.

5. 게스트 종료

   a. 다음 명령을 실행하십시오.

```
# virsh list --all  - this command lists all of the known domains
 Id Name            State
----------------------------------
 rhel6              running
```

   b. 게스트 가상 머신 종료

```
# virsh shutdown rhel6

Domain rhel6 is being shutdown
```

   c. 게스트 가상 머신이 종료될 때까지 몇 초 동안 기다립니다.

```
# virsh list --all
 Id Name                 State
----------------------------------
  . rhel6                shut off
```

d.

*편집한 **XML** 파일을 사용하여 **rhel6** 이라는 도메인을 시작합니다.*

```
# virsh start rhel6
```

e.

***rhel6** 게스트 가상 시스템에서 **acpi**를 종료합니다.*

```
# virsh shutdown --mode acpi rhel6
```

f.

***rhel6**은 모든 도메인을 다시 나열합니다. **rhel6** 은 목록에 계속 있어야 하며, 종료됨으로 표시되어야 합니다.*

```
# virsh list --all
  Id Name                 State
----------------------------------
  rhel6                  shut off
```

g.

*편집한 **XML** 파일을 사용하여 **rhel6** 이라는 도메인을 시작합니다.*

```
# virsh start rhel6
```

h.

***rhel6** 게스트 가상 시스템 게스트 에이전트를 종료합니다.*

```
# virsh shutdown --mode agent rhel6
```

i.

*도메인을 나열합니다. **rhel6** 은 목록에 있어야 하며 종료되었음을 나타냅니다.*

```
# virsh list --all
  Id Name                 State
----------------------------------
  rhel6                  shut off
```

*게스트 가상 시스템은 위에 설명된 해결 방법을 사용하지 않고 연속 종료에 **virsh shutdown** 명령을 사용하여 종료됩니다.*

위에 설명된 방법 외에도 **libvirt-guest** 서비스를 중지하여 게스트를 자동으로 종료할 수 있습니다. 이 방법에 대한 자세한 내용은 14.9.3절. “libvirt-guests 구성 설정 조작” 를 참조하십시오.

14.9.3. libvirt-guests 구성 설정 조작

libvirt-guests 서비스에는 게스트가 올바르게 종료되도록 구성할 수 있는 매개 변수 설정이 있습니다. 이 패키지는 **libvirt** 설치의 일부이며 기본적으로 설치됩니다. 이 서비스는 호스트가 종료될 때 자동으로 게스트를 디스크에 저장하고 호스트가 재부팅될 때 이전 상태로 복원합니다. 기본적으로 이 설정은 게스트를 일시 중단하도록 설정되어 있습니다. 게스트를 종료하려면 **libvirt-guests** 구성 파일의 매개 변수 중 하나를 변경해야 합니다.

절차 14.5. 게스트의 정상 종료를 허용하도록 libvirt-guests 서비스 매개 변수 변경

여기에 설명된 절차를 통해 호스트 물리적 시스템이 꺼져 있거나 전원이 꺼지거나 다시 시작해야 하는 경우 게스트 가상 시스템의 정상 종료가 허용됩니다.

1. 구성 파일을 엽니다.

   구성 파일은 **/etc/sysconfig/libvirt-guests** 에 있습니다. 파일을 편집하고 주석 표시(**#**)를 제거하고 **ON_SHUTDOWN=suspend** 를 **ON_SHUTDOWN=shutdown** 으로 변경합니다. 변경 사항을 저장합니다.

```
$ 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. It should be noted 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 time out (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** 서비스 시작

서비스를 시작하지 않은 경우 **libvirt-guests** 서비스를 시작합니다. 서비스를 다시 시작하지 않도록 하면 실행 중인 모든 도메인이 종료됩니다.

14.9.4. 게스트 가상 머신 재부팅

virsh reboot 명령을 사용하여 **guest** 가상 시스템을 재부팅합니다. 재부팅이 실행되면 프롬프트가 반환됩니다. 게스트 가상 머신이 반환될 때까지 시간이 경과할 수 있습니다.

```
#virsh reboot {domain-id, domain-name or domain-uuid} [--mode method]
```

게스트 가상 시스템의 구성 파일에서 **<on_reboot>** 요소를 수정하여 재부팅 게스트 가상 머신의 동작을 제어할 수 있습니다. 자세한 내용은 **20.12**절. “이벤트 구성” 를 참조하십시오.

기본적으로 하이퍼바이저는 적절한 종료 방법을 선택합니다. 대체 방법을 지정하려면 **--mode** 옵션은 **initctl,acpi,agent, signal** 를 포함하는 쉼표로 구분된 목록을 지정할 수 있습니다. 드라이버에서 각 모드

를 시도하는 순서는 명령에 지정된 순서와 관련이 없습니다. 순서를 엄격하게 제어하려면 한 번에 단일 모드를 사용하고 명령을 반복합니다.

14.9.5. 게스트 가상 머신이 중지되도록 강제 적용

virsh destroy 명령을 사용하여 게스트 가상 머신을 강제로 중지합니다.

```
# virsh destroy {domain-id, domain-name or domain-uuid} [--graceful]
```

이 명령은 즉시 비정상적인 종료를 수행하고 지정된 게스트 가상 머신을 중지합니다. virsh destroy 를 사용하면 게스트 가상 머신 파일 시스템이 손상될 수 있습니다. 게스트 가상 머신이 응답하지 않는 경우에만 destroy 옵션을 사용합니다. 정상 종료를 시작하려면 virsh destroy --graceful 명령을 사용하십시오.

14.9.6. 가상 머신 재설정

virsh reset domain 은 게스트 종료 없이 즉시 도메인을 재설정합니다. 재설정을 통해 시스템의 전원 재설정 버튼을 에뮬레이션합니다. 여기서 모든 게스트 하드웨어는 RST 행을 보고 내부 상태를 다시 초기화합니다. 게스트 가상 시스템 OS를 종료하지 않으면 데이터 손실 위험이 있습니다.

14.10. 게스트 가상 머신 정보 검색

이 섹션에서는 게스트 가상 머신 정보를 검색하는 방법에 대한 정보를 제공합니다.

14.10.1. 게스트 가상 머신의 도메인 ID 가져오기

게스트 가상 머신의 도메인 ID를 가져오려면 다음을 수행합니다.

```
# virsh domid {domain-name or domain-uuid}
```

14.10.2. 게스트 가상 머신의 도메인 이름 가져오기

게스트 가상 머신의 도메인 이름을 가져오려면 다음을 수행합니다.

```
# virsh domname {domain-id or domain-uuid}
```

14.10.3. 게스트 가상 머신의 UUID 가져오기

게스트 가상 머신의 **UUID(Universally Unique Identifier)**를 가져오려면 다음을 수행합니다.

```
# virsh domuuid {domain-id or domain-name}
```

virsh domuuid 출력의 예는 다음과 같습니다.

```
# virsh domuuid r5b2-mySQL01
4a4c59a7-ee3f-c781-96e4-288f2862f011
```

14.10.4. 게스트 가상 머신 정보 표시

virsh 를 게스트 가상 머신의 도메인 **ID**, 도메인 이름 또는 **UUID**와 함께 사용하면 지정된 게스트 가상 머신에 정보를 표시할 수 있습니다.

```
# virsh dominfo {domain-id, domain-name or domain-uuid}
```

virsh dominfo 출력의 예입니다.

```
# virsh dominfo vr-rhel6u1-x86_64-kvm
Id:             9
Name:           vr-rhel6u1-x86_64-kvm
UUID:           a03093a1-5da6-a2a2-3baf-a845db2f10b9
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       21.6s
Max memory:     2097152 kB
Used memory:    1025000 kB
Persistent:     yes
Autostart:      disable
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c612,c921 (permissive)
```

14.11. 스토리지 풀 명령

다음 명령은 스토리지 풀을 조작합니다. **libvirt** 를 사용하면 가상 머신 내에서 장치로 표시되는 스토리지 볼륨을 제공하는 데 사용되는 파일, 원시 파티션 및 도메인별 형식을 비롯한 다양한 스토리지 솔루션을 관리할 수 있습니다. 이 기능에 대한 자세한 내용은 **libvirt.org** 에서 자세한 내용을 참조하십시오. 스토리지 풀에 대한 대부분의 명령은 도메인에 사용되는 명령과 유사합니다.

14.11.1. 스토리지 풀 XML 검색

find-storage-pool-sources type srcSpec 명령은 찾을 수 있는 지정된 유형의 모든 스토리지 풀을 설명하는 XML을 표시합니다. srcSpec 이 제공된 경우 풀에 대한 쿼리를 추가로 제한하는 XML이 포함된 파일입니다.

find-storage-pool-sources-as 유형 호스트 포트 이니시에이터 는 찾을 수 있는 지정된 유형의 모든 스토리지 풀을 설명하는 XML을 표시합니다. 호스트,포트 또는 이니시에이터 가 제공되는 경우 쿼리가 수행되는 위치를 제어합니다.

pool-info pool-or-uuid 명령은 지정된 스토리지 풀 오브젝트에 대한 기본 정보를 나열합니다. 이 명령에는 스토리지 풀의 이름 또는 UUID가 필요합니다. 이 정보를 검색하려면 다음 암호를 사용합니다.To retrieve this information, use the following coomand:

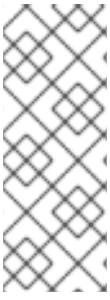
```
pool-list [--inactive] [--all] [--persistent] [--transient] [--autostart] [--no-autostart] [--details] type
```

libvirt 에 알려진 모든 스토리지 풀 오브젝트가 나열됩니다. 기본적으로 활성 풀만 나열되지만 --inactive 옵션을 사용하면 비활성 풀만 나열하고 --all 옵션을 사용하면 모든 스토리지 풀이 나열됩니다.

이러한 옵션 외에도 목록의 내용을 필터링하는 데 사용할 수 있는 몇 가지 필터링 옵션 세트가 있습니다. --persistent 에서는 목록을 영구 풀로 제한합니다. --transient 는 목록을 일시적인 풀로 제한합니다. --autostart 는 목록을 풀 자동 시작으로 제한하며, 마지막으로 --no-autostart 는 목록을 자동 시작 해제한 스토리지 풀로 제한합니다.

유형이 필요한 모든 스토리지 풀 명령의 경우 풀 유형을 쉼표로 구분해야 합니다. 유효한 풀 유형에는 dir,fs,netfs,logical,disk,iscsi,scsi,mpath,rbd, sheepdog 등이 있습니다.

--details 옵션은 virsh 에 사용 가능한 경우 풀 지속성 및 용량 관련 정보를 추가로 표시하도록 지시합니다.



참고

이 명령을 이전 서버와 함께 사용하면 고유의 경쟁과 함께 일련의 API 호출을 사용해야 합니다. 여기에서 풀이 나열되지 않을 수 있으며 목록이 수집되는 동안 호출 간 상태를 변경하는 경우 한 번 이상 나타날 수 있습니다. 그러나 최신 서버에는 이러한 문제가 발생하지 않습니다.

pool-refresh pool-or-uuid 는 풀에 포함된 볼륨 목록을 새로 고칩니다.

14.11.2. 스토리지 풀 생성, 삭제 및 시작

이 섹션에서는 스토리지 풀 생성, 정의 및 시작에 대한 정보를 제공합니다.

14.11.2.1. 스토리지 풀 빌드

pool-build pool-or-uuid --overwrite --no-overwrite 명령은 지정된 풀 이름 또는 **UUID**를 사용하여 풀 을 빌드합니다. **--overwrite** 및 **--no-overwrite** 옵션은 유형이 파일 시스템인 풀에서만 사용할 수 있습니다. 옵션을 둘 다 지정하지 않고 풀이 파일 시스템 유형 풀인 경우 결과 빌드에서는 디렉터리만 만듭니다.

--no-overwrite 가 지정된 경우 파일 시스템이 대상 장치에 이미 있는지 확인하거나, 존재하는 경우 오류를 반환하거나 **mkfs** 를 사용하여 대상 장치를 포맷합니다. **--overwrite** 가 지정된 경우 **mkfs** 명령이 실행되고 대상 장치의 기존 데이터를 덮어씁니다.

14.11.2.2. XML 파일에서 스토리지 풀 생성 및 정의

pool-create 파일은 연결된 **XML** 파일에서 스토리지 풀을 생성하고 시작합니다.

pool-define 파일은 생성하지만 시작하지는 않습니다. **XML** 파일에서 스토리지 풀 오브젝트를 생성합니다.

14.11.2.3. 원시 매개변수에서 스토리지 풀 생성 및 시작

```
# pool-create-as name --print-xml type source-host source-path source-dev source-name <target> --
source-format format
```

이 명령은 지정된 원시 매개변수에서 풀 오브젝트 이름을 생성하고 시작합니다.

--print-xml 을 지정하면 풀을 생성하지 않고 스토리지 풀 오브젝트의 **XML**을 출력합니다. 그렇지 않으면 풀을 구축해야 합니다. 유형이 필요한 모든 스토리지 풀 명령의 경우 풀 유형을 쉼표로 구분해야 합니다. 유효한 풀 유형에는 **dir,fs,netfs,logical,disk,iscsi,scsi,mpath,rbd, sheepdog** 등이 있습니다.

반대로 다음 명령은 지정된 원시 매개변수에서 풀 오브젝트 이름을 생성하지만 시작하지 않습니다.

```
# pool-define-as name --print-xml type source-host source-path source-dev source-name <target> --
source-format format
```

--print-xml 을 지정하면 풀을 정의하지 않고 풀 오브젝트의 **XML**을 출력합니다. 그렇지 않으면 풀에 지정된 유형이 있어야 합니다. 유형이 필요한 모든 스토리지 풀 명령의 경우 풀 유형을 쉼표로 구분해야 합니다. 유효한 풀 유형에는 **dir,fs,netfs,logical,disk,iscsi,scsi,mpath,rbd, sheepdog** 등이 있습니다.

pool-start pool-or-uuid 는 이전에 정의되었지만 비활성 상태인 지정된 스토리지 풀을 시작합니다.

14.11.2.4. 스토리지 풀 자동 시작

pool-autostart pool-or-uuid --disable 명령은 부팅 시 자동으로 시작되도록 스토리지 풀을 활성화하거나 비활성화합니다. 이 명령에는 풀 이름 또는 **UUID**가 필요합니다. **pool-autostart** 명령을 비활성화하려면 **--disable** 옵션을 사용합니다.

14.11.3. 스토리지 풀 중지 및 삭제

pool-destroy pool-or-uuid 는 스토리지 풀을 중지합니다. 중지되면 **libvirt** 에서 더 이상 풀을 관리하지 않지만 풀에 포함된 원시 데이터는 변경되지 않으며 나중에 **pool-create** 명령으로 복구할 수 있습니다.

pool-delete pool-or-uuid 는 지정된 스토리지 풀에서 사용하는 리소스를 제거합니다. 이 작업은 복구할 수 없고 되돌릴 수 없다는 점에 유의해야 합니다. 그러나 이 명령 후에도 풀 구조가 계속 존재하며 새 스토리지 볼륨 생성을 허용할 준비가 되었습니다.

pool-undefine pool-or-uuid 명령은 비활성 풀에 대한 구성을 정의하지 않습니다.

14.11.4. 스토리지 풀에 대한 XML 덤프 파일 생성

pool-dumpxml --inactive pool-or-uuid 명령은 지정된 스토리지 풀 오브젝트에 대한 **XML** 정보를 반환합니다. **--inactive** 를 사용하면 현재 풀 구성과 달리 풀의 다음 시작에 사용할 구성을 덤프합니다.

14.11.5. 스토리지 풀의 구성 파일 편집

pool-edit pool-or-uuid 는 편집하기 위해 지정된 스토리지 풀의 **XML** 구성 파일을 엽니다.

이 메서드는 적용 전에 오류 검사를 수행하기 때문에 **XML** 구성 파일을 편집하는 데 사용해야 하는 유일한 방법입니다.

14.11.6. 스토리지 풀 변환

pool-name uuid 명령은 지정된 **UUID**를 풀 이름으로 변환합니다.

pool-uuid pool 명령은 지정된 풀의 **UUID**를 반환합니다.

14.12. 스토리지 볼륨 명령

이 섹션에서는 스토리지 볼륨을 생성, 삭제 및 관리하기 위한 모든 명령에 대해 설명합니다. 스토리지 풀 이름 또는 **UUID**로 스토리지 풀을 생성하면 이 작업을 수행하는 것이 가장 좋습니다. 스토리지 풀에 대한 정보는 *12장. 스토리지 풀* 에서 참조하십시오. 스토리지 볼륨에 대한 자세한 내용은 *13장. volumes* 에서 참조하십시오.

14.12.1. 스토리지 볼륨 생성

vol-create-from pool-or-uuid file --inputpool pool-or-uuid vol-name-or-key-or-path 명령은 다른 스토리지 볼륨을 콘텐츠의 템플릿으로 사용하여 스토리지 볼륨을 생성합니다. 이 명령을 사용하려면 볼륨을 생성할 스토리지 풀의 이름 또는 **UUID**인 **pool-or-uuid** 가 필요합니다.

file 인수는 볼륨 정의가 포함된 **XML** 파일 및 경로를 지정합니다. **--inputpool pool-or-uuid** 옵션은 소스 볼륨이 있는 스토리지 풀의 이름 또는 **uuid**를 지정합니다. **vol-name-or-key-or-path** 인수는 소스 볼륨의 이름 또는 키 또는 경로를 지정합니다. 일부 예제는 *13.1절. “볼륨 생성”* 를 참조하십시오.

vol-create-as 명령은 일련의 인수에서 볼륨을 생성합니다. **pool-or-uuid** 인수에는 볼륨을 생성할 스토리지 풀의 이름 또는 **UUID**가 포함되어 있습니다.

```
vol-create-as pool-or-uuid name capacity --allocation <size> --format <string> --backing-vol <vol-name-or-key-or-path> --backing-vol-format <string>
```

name 은 새 볼륨의 이름입니다. **capacity** 는 접미사가 없는 경우 크기가 조정된 정수로 생성할 볼륨의 크기이며, 접미사가 없는 경우 기본값으로 바이트로 설정됩니다. **--allocation <크기는>** 스케일링된 정수 기본값을 바이트로 지정하여 볼륨에 할당할 초기 크기입니다. **--format <문자열>** 은 파일 기반 스토리지 풀에서 사용되어 쉼표로 구분된 허용 가능한 형식 문자열인 볼륨 파일 형식을 지정합니다. 허용 가능한 형식에는 **raw,bochs,qcow 2 ,qcow2,vmdk, --backing- vol-name-or-key-or-path** 가 기존 볼륨의 스냅샷을 찍는 경우 사용할 소스 백업 볼륨이 포함됩니다. **--backing-vol-format** 문자열 은 쉼표로 구분된 형식 문자열인 스냅샷 지원 볼륨의 형식입니다. 허용되는 값에는 **raw,bochs,qcow,qcow2,, vmdk, host_device** 가 포함됩니다. 그러나 이는 파일 기반 스토리지 풀에만 적용됩니다.

14.12.1.1. XML 파일에서 스토리지 볼륨 생성

vol-create pool-or-uuid 파일은 저장된 **XML** 파일에서 스토리지 볼륨을 생성합니다. 또한 이 명령에는 볼륨이 생성될 스토리지 풀의 이름 또는 **UUID**인 **pool-or-uuid** 도 필요합니다. **file** 인수에는 볼륨 정의의 **XML** 파일이 있는 경로가 포함되어 있습니다. **XML** 파일을 쉽게 생성하는 방법은 **vol-dumpxml** 명령을 사용하여 기존 볼륨의 정의를 가져온 다음 저장한 다음, **vol-create** 을 실행하는 것입니다.

```
virsh vol-dumpxml --pool storagepool1 appvolume1 > newvolume.xml
virsh edit newvolume.xml
virsh vol-create differentstoragepool newvolume.xml
```

사용 가능한 기타 옵션은 다음과 같습니다.

- **--inactive** 옵션은 비활성 게스트 가상 머신(즉, 정의되어 있지만 현재 활성화되어 있지 않은 게스트 가상 머신)을 나열합니다.
- **all** 옵션은 모든 게스트 가상 머신을 나열합니다.

14.12.1.2. 스토리지 볼륨 복제

vol-clone --pool pool-or-uuid vol-name-or-key-or-path name 명령은 기존 스토리지 볼륨을 복제합니다. **vol-create-from** 도 사용할 수 있지만 스토리지 볼륨을 복제하는 것이 권장되는 방법은 아닙니다. **--pool pool-or-uuid** 옵션은 볼륨을 생성할 스토리지 풀의 이름 또는 **UUID**입니다. **vol-name-or-key-or-path** 인수는 소스 볼륨의 이름 또는 키입니다. **name** 인수를 사용하면 새 볼륨의 이름을 나타냅니다.

14.12.2. 스토리지 볼륨 삭제

vol-delete --pool pool-or-uuid vol-name-or-key-path 명령은 지정된 볼륨을 삭제합니다. 명령에는 볼륨이 있는 스토리지 풀의 이름 또는 **UUID**인 특정 **--pool pool-or-uuid** 가 필요합니다. **vol-name-or-key-or-path** 옵션은 삭제할 볼륨의 이름 또는 키 또는 경로를 지정합니다.

vol-wipe --pool pool-or-uuid -- algorithm알고리즘 **vol-name-or-key-path** 명령은 볼륨을 지우고 이전에 볼륨의 데이터를 나중에 읽을 수 없도록 합니다. 명령에는 볼륨이 있는 스토리지 풀의 이름 또는 **UUID**인 **--pool pool-or-uuid** 가 필요합니다. **vol-name-or-key-or-path** 에는 초기화할 볼륨의 이름 또는 키 또는 경로가 포함됩니다. 기본값 대신 다른 삭제 알고리즘을 선택할 수 있습니다(스토리지 볼륨의 모든 섹터는 값 **"0"**)로 작성됩니다. 삭제 알고리즘을 지정하려면 다음 지원되는 알고리즘 유형 중 하나로 **--algorithm** 옵션을 사용합니다.

- **Zero - 1-pass all zeros**
-

NNSA - 4-pass NNSA 정책 Letter NAP-14.1-C(XVI-8)는 이동식 및 비독성 하드 디스크 (random x2, 0x00)를 삭제합니다.

- *DoD - 4-pass DoD 5220.22-M 섹션 8-306 절차: random, 0x00, 0xff와 같은 이동식 및 비모형 디스크를 정리합니다.*

- *HEALTH - Information Technologies의 독일어 센터 보안 센터에서 권장하는 방법 (http://www.bsi.bund.de): 0xff, 0xfe, 0xfd, 0xf7, 0xf7, 0xdf, 0xbf, 0x7f, 0x7f.*

- *Gutmann - Gutmann 문서에 설명된 정식 35-pass 시퀀스입니다.*

- *Schneier - 7-pass 방법: Bruce Schneier in "1996): 0x00, 0xff, random x5.*

- *pfitzner7 - Roy Pfitzner의 7-random-pass 방법: 임의의 x7*

- *pfitzner33 - Roy Pfitzner의 33-random-pass 방법: 임의의 x33.*

- *random - 1-pass 패턴: random.*

참고

호스트에 설치된 scrub 바이너리의 버전은 사용 가능한 알고리즘을 제한합니다.

14.12.3. XML 파일에 스토리지 볼륨 정보 덤프

Vol-dumpxml --pool pool-or -uuidvol-name-or-key-path 명령은 볼륨 정보를 지정된 파일에 XML 덤프로 사용합니다.

이 명령에는 볼륨이 있는 스토리지 풀의 이름 또는 UUID인 --pool pool-or-uuid 가 필요합니다. vol-name-or-key-path 는 결과 XML 파일을 배치할 볼륨의 이름 또는 키 또는 경로입니다.

14.12.4. 볼륨 정보 나열

vol-info --pool pool-or-uuid vol-name-or-key-or-path 명령은 지정된 스토리지 볼륨 --pool 에 대한 기본 정보를 나열합니다. 여기서 pool-or-uuid 는 볼륨이 있는 스토리지 풀의 이름 또는 UUID입니다. Vol-name-or-key-path 는 정보를 반환할 볼륨의 이름 또는 키 또는 경로입니다.

vol-list --pool pool-or-uuid --details 는 지정된 스토리지 풀의 모든 볼륨을 나열합니다. 이 명령에는 스토리지 풀의 이름 또는 UUID인 --pool pool-or-uuid 가 필요합니다. --details 옵션은 virsh 에 사용 가능한 볼륨 유형 및 용량 관련 정보를 추가로 표시하도록 지시합니다.

14.12.5. 스토리지 볼륨 정보 검색

vol-pool --uuid vol-key-or-path 명령은 지정된 볼륨의 풀 이름 또는 UUID를 반환합니다. 기본적으로 풀 이름은 반환됩니다. --uuid 옵션이 지정되면 풀 UUID가 대신 반환됩니다. 명령에는 요청된 정보를 반환하는 볼륨의 키 또는 경로인 vol-key-or-path 가 필요합니다.

vol-path --pool pool-or-uuid vol-name-or-key 명령은 지정된 볼륨의 경로를 반환합니다. 명령에는 볼륨이 있는 스토리지 풀의 이름 또는 UUID인 --pool pool-or-uuid 가 필요합니다. 또한 경로가 요청된 볼륨의 이름 또는 키인 vol-name-or-key 가 필요합니다.

vol-name vol-key-or-path 명령은 지정된 볼륨의 이름을 반환합니다. 여기서 vol-key-or-path 는 의 이름을 반환할 볼륨의 키 또는 경로입니다.

vol-key --pool pool-or-uuid vol-name-or-path 명령은 지정된 볼륨에 대한 볼륨 키를 반환합니다. 여기서 --pool pool-or-uuid 는 볼륨이 있고 vol-name-or-path 가 볼륨 키를 반환하는 볼륨의 이름 또는 경로입니다.

14.12.6. 스토리지 볼륨 업로드 및 다운로드

이 섹션에서는 스토리지 볼륨에 정보를 업로드하고 다운로드하는 방법을 지시합니다.

14.12.6.1. 스토리지 볼륨에 콘텐츠 업로드

vol-upload --pool pool-or-uuid --offset 바이트 --length 바이트 vol-name-or-key-or-path local-file 명령은 지정된 local-file 의 콘텐츠를 스토리지 볼륨에 업로드합니다. 명령에는 볼륨이 있는 스토리지 풀의 이름 또는 UUID인 --pool pool-or-uuid 가 필요합니다. 또한 초기화할 볼륨의 이름 또는 키 또는 경로인 vol- name-or-key-path가 필요합니다. --offset 옵션은 데이터 쓰기를 시작할 스토리지 볼륨의 위치입니다. -- lengthlength 는 업로드할 데이터의 양에 대한 상한을 지정합니다. local-file 이 지정된 --length 보다 큰 경우 오류가 발생합니다.

14.12.6.2. 스토리지 볼륨에서 콘텐츠 다운로드

```
# vol-download --pool pool-or-uuid --offset bytes --length bytes vol-name-or-key-or-path local-file
```

*이 명령 명령은 스토리지 볼륨에서 **local-file** 내용을 다운로드합니다. 볼륨이 있는 스토리지 풀의 이름 또는 **UUID**인 **--pool pool-or-uuid** 가 필요합니다. 또한 초기화할 볼륨의 이름 또는 키 또는 경로인 **vol-name-or-key-path**가 필요합니다. **--offset** 옵션을 사용하면 데이터 읽기를 시작할 스토리지 볼륨의 위치가 지정됩니다. **-- lengthlength** 는 다운로드할 데이터의 양에 대한 상한을 지정합니다.*

14.12.7. 스토리지 볼륨 다시 지정

```
# vol-resize --pool pool-or-uuid vol-name-or-path pool-or-uuid capacity --allocate --delta --shrink
```

*이 **command** 명령은 지정된 볼륨의 용량을 바이트 단위로 다시 조정합니다. 명령에는 볼륨이 있는 스토리지 풀의 이름 또는 **UUID**인 **--pool pool-or-uuid** 가 필요합니다. 또한 이 명령에는 **vol-name-or-key-or-path** 가 다시 크기 조정할 볼륨의 이름 또는 키 또는 경로가 있어야 합니다.*

*새 용량은 **--allocate** 옵션을 지정하지 않는 한 스파스 파일을 만들 수 있습니다. 일반적으로 용량은 새 크기이지만 **--delta** 가 있는 경우 기존 크기에 추가됩니다. **--shrink** 옵션이 없으면 볼륨 축소가 실패합니다.*

***--shrink** 옵션을 제공하지 않고 음수 기호가 필요하지 않은 경우 용량은 음수일 수 없습니다. **capacity** 는 접미사가 없는 경우 기본값은 바이트입니다. 이 명령은 활성 게스트에서 사용하지 않는 스토리지 볼륨에만 안전합니다. 실시간 재조정은 14.5.17절. “블록 크기를 사용하여 도메인 경로의 크기 변경” 를 참조하십시오.*

14.13. 게스트별 가상 머신 정보 표시

이 섹션에서는 각 게스트에 대한 가상 머신 정보를 표시하는 방법에 대한 정보를 제공합니다.

14.13.1. 게스트 가상 머신 표시

***virsh** 로 게스트 가상 머신 목록 및 현재 상태를 표시하려면 다음을 수행합니다.*

```
# virsh list
```

사용 가능한 기타 옵션은 다음과 같습니다.

- *__--inactive__ 옵션은 비활성 게스트 가상 머신(즉, 정의되어 있지만 현재 활성화되어 있지 않은 게스트 가상 머신)을 나열합니다.*

- ***--all** 옵션은 모든 게스트 가상 머신을 나열합니다. 예를 들어 다음과 같습니다.*

```
# virsh list --all
 Id Name                 State
----------------------------------
 0 Domain-0              running
 1 Domain202             paused
 2 Domain010             inactive
 3 Domain9600            crashed
```

*이 명령을 사용하여 볼 수 있는 **7** 가지 상태가 있습니다.*

  - *실행 중 **-** 실행 중 상태는 현재 **CPU**에서 활성 상태인 게스트 가상 머신을 나타냅니다.*

  - ***idle - idle** 상태는 도메인이 유휴 상태이며 실행 중이거나 실행되지 않을 수 있음을 나타냅니다. 이는 도메인이 **IO**(기존 대기 상태)에서 대기 중이거나 다른 작업을 수행할 수 없기 때문에 발생할 수 있습니다.*

  - ***paused -** 일시 정지된 도메인이 나열됩니다. 이는 관리자가 **virt-manager** 또는 **virsh** 일시 중지 에서 일시 정지 버튼을 사용하는 경우 발생합니다. 게스트 가상 머신이 일시 중지되면 메모리와 기타 리소스가 소모되지만 하이퍼바이저에서의 예약 및 **CPU** 리소스는 불가능합니다.*

  - ***shutdown -** 종료 상태가 종료 프로세스 중 게스트 가상 시스템용입니다. 게스트 가상 시스템은 종료 신호를 전송하므로 작업을 정상적으로 중지해야 합니다. 일부 운영 체제는 모든 게스트 가상 머신 운영 체제에서는 작동하지 않을 수 있습니다. 일부 운영 체제는 이러한 신호에 응답하지 않습니다.*

  - ***shut off -** 종료 상태는 도메인이 실행되지 않음을 나타냅니다. 이는 도메인이 완전히 종료되거나 시작되지 않은 경우 발생할 수 있습니다.*

  - *충돌 **-** 충돌 됨 상태는 도메인이 충돌했음을 나타내며 게스트 가상 머신이 충돌 시 재시작되지 않도록 구성된 경우에만 발생할 수 있습니다.*

  - ***dying** 상태의 도메인은 **dying** 프로세스이며, 이는 도메인이 완전히 종료되거나 충돌하지 않은 상태입니다.*

- *--managed-save A-save 이 옵션만으로는 도메인을 필터링하지 않지만, 관리되는 저장 상태가 활성화된 도메인이 나열됩니다. 실제로 도메인을 별도로 나열하려면 --inactive 옵션도 사용해야 합니다.*

- *--name 은 지정된 도메인 이름이 목록에 인쇄됩니다. --uuid 가 지정된 경우 도메인의 UUID가 대신 인쇄됩니다. --table 옵션을 사용하면 테이블 스타일 출력을 사용해야 합니다. 세 가지 명령은 모두 상호 배타적입니다.*

- *--title 이 명령은 --table 출력과 함께 사용해야 합니다. --title은 짧은 도메인 설명(title)을 사용하여 테이블에서 추가 열을 생성합니다.*

- *--persistent에는 목록에 영구 도메인이 포함되어 있습니다. --transient 옵션을 사용합니다.*

- *--with-managed-save 는 관리 저장으로 구성된 도메인이 나열됩니다. 없이 명령을 나열하려면 --without-managed-save명령을 사용합니다.*

- *--state-running 필터는 실행 중인 도메인에 대해 --state-paused, 일시 중지된 도메인의 --state-paused, 해제된 도메인의 --state-shutoff, --state-other 는 모든 상태를 폴백으로 나열합니다.*

- *--autostart 이 옵션을 사용하면 자동 시작 도메인이 나열됩니다. 이 기능이 비활성화된 도메인을 나열하려면 --no-autostart 옵션을 사용합니다.*

- *--with-snapshot 은 스냅샷 이미지를 나열할 수 있는 도메인이 나열됩니다. 스냅샷이 없는 도메인을 필터링하려면 --without-snapshot옵션을 사용합니다.*

```
$ virsh list --title --name

 Id    Name                           State    Title
 0     Domain-0                       running  Mailserver1
 2     rhelvm                         paused
```

virsh vcpuinfo 출력의 예는 다음을 참조하십시오. 14.13.2절. “가상 CPU 정보 표시”

14.13.2. 가상 CPU 정보 표시

virsh 로 게스트 가상 머신에서 가상 **CPU** 정보를 표시하려면 다음을 수행합니다.

```
# virsh vcpuinfo {domain-id, domain-name or domain-uuid}
```

virsh vcpuinfo 출력의 예는 다음과 같습니다.

```
# virsh vcpuinfo rhel6
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
```

14.13.3. 가상 CPU 선호도 구성

물리적 **CPU**를 사용하여 가상 **CPU**의 선호도를 구성하려면 예 14.3. "호스트 물리적 머신의 CPU에 vCPU 고정" 를 참조하십시오.

예 14.3. 호스트 물리적 머신의 **CPU**에 **vCPU** 고정

virsh vcpupin 은 가상 **CPU**를 실제 **CPU**에 할당합니다.

```
# virsh vcpupin rhel6
VCPU: CPU Affinity
----------------------------------
   0: 0-3
   1: 0-3
```

vcpupin 은 다음 옵션을 사용할 수 있습니다.

- **--vCPU의 vcpu** 번호 필요
- **[-cpulist] >string <** 설정할 호스트 물리적 시스템의 **CPU** 번호를 나열하거나 선택적 쿼리를 생략합니다.

- **--config** 는 다음 부팅에 영향을 미칩니다.
- **--live** 는 실행 중인 도메인에 영향을 미칩니다.
- **--current** 가 현재 도메인에 영향을 미칩니다.

14.13.4. 도메인의 가상 CPU 수에 대한 정보 표시

virsh vcpucount 에는 도메인 이름 또는 도메인 ID가 필요합니다. 예를 들어 다음과 같습니다.

```
# virsh vcpucount rhel6
maximum      config         2
maximum      live           2
current      config         2
current      live           2
```

vcpucount 는 다음 옵션을 사용할 수 있습니다.

- **--maximum** 사용 가능한 최대 **vCPU** 수 표시
- **--active** 는 현재 활성화된 **vCPU** 수를 표시
- **--live** 에서 실행 중인 도메인의 값을 표시
- **--config** 는 게스트 가상 머신의 다음 부팅에 구성할 값을 표시합니다.
- **--current** 가 현재 도메인 상태에 따라 값을 표시
- **--guest** 는 반환된 수를 게스트 관점에서 표시합니다.

14.13.5. 가상 CPU 선호도 구성

물리적 CPU를 사용하여 가상 CPU의 선호도를 구성하려면 다음을 수행합니다.

```
# virsh vcpupin domain-id vcpu cpulist
```

domain-id 매개변수는 게스트 가상 머신의 ID 번호 또는 이름입니다.

vcpu 매개변수는 게스트 가상 머신에 할당된 가상화된 CPU 수를 나타냅니다. vcpu 매개 변수를 제공해야 합니다.

cpulist 매개변수는 쉼표로 구분된 물리적 CPU 식별자 번호 목록입니다. cpulist 매개변수는 VCPU가 실행할 수 있는 물리적 CPU를 결정합니다.

--config 와 같은 추가 매개변수는 다음 부팅에 영향을 미치지만 --live 는 실행 중인 도메인에 영향을 미치며 --current 는 현재 도메인에 영향을 미칩니다.

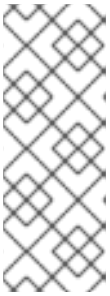
14.13.6. 가상 CPU 수 구성

게스트 가상 머신에 할당된 CPU 수를 수정하려면 virsh setvcpus 명령을 사용합니다.

```
# virsh setvcpus {domain-name, domain-id or domain-uuid} count [[--config] [--live] | [--current] [--guest]
```

virsh setvcpus 명령에 대해 다음 매개 변수를 설정할 수 있습니다.

- {domain-name, domain-id 또는 domain-uuid} - 가상 머신을 지정합니다.
- count - 설정할 가상 CPU 수를 지정합니다.



참고

개수 값은 게스트 가상 시스템을 생성할 때 게스트 가상 머신에 할당된 CPU 수를 초과할 수 없습니다. 호스트 또는 하이퍼바이저에 의해 제한될 수도 있습니다. Xen의 경우 도메인이 반가상화된 경우에만 실행 중인 도메인의 가상 CPU를 조정할 수 있습니다.

- ***--live -*** *지정하지 않은 경우 사용되는 기본 옵션입니다. 구성 변경 사항은 실행 중인 게스트 가상 머신에 적용됩니다.* ***vCPU*** *수가 늘어나면 핫 플러그 라고 하며 축소된 경우 핫 플러그 라고 합니다.*

중요

vCPU *핫 플러그 해제 기능은 기술 프리뷰입니다. 따라서 이 기능은 지원되지 않으며 높은 가치의 배포에는 사용하지 않는 것이 좋습니다.*

- ***--config -*** *구성 변경 사항이 게스트의 다음 재부팅에 적용됩니다. 하이퍼바이저에서 지원하는 경우* ***--config*** *및* ***--live*** *옵션 둘 다 함께 지정할 수 있습니다.*

- ***--current -*** *구성 변경 사항이 게스트 가상 머신의 현재 상태에 적용됩니다. 실행 중인 게스트에서 사용되는 경우 종료 게스트에서 사용되는 경우* ***--config*** *역할을 합니다.*

- ***--maximum -*** *다음 게스트 재부팅 시 핫플러그할 수 있는 최대* ***vCPU*** *제한을 설정합니다. 따라서* ***--live*** *옵션이 아닌* ***--config*** *옵션과 함께 사용해야 합니다.*

- ***--guest -*** *핫 플러그 또는 핫 플러그 해제 대신* ***vCPU****를 활성화하거나 비활성화하여 실행 중인 게스트에서* ***vCPU*** *수를 직접 수정합니다. 이 옵션은* ***gueet****의 현재* ***vCPU*** *수보다 높은 수의 개수 와 함께 사용할 수 없으며, 게스트를 재부팅할 때* ***--guest*** *로 설정된 구성은 재설정됩니다.*

예 14.4. vCPU *핫 플러그 및 핫 언플러그*

vCPU*를 핫 플러그하려면 단일* ***vCPU****가 있는 게스트에서 다음 명령을 실행합니다.*

```
virsh setvcpus guestVM1 2 --live
```

이로 인해 ***guestVM1****의* ***vCPU*** *수가* ***2****개로 증가합니다. 변경 사항은* ***--live*** *옵션에 표시된 대로* ***guestVM1****이 실행되는 동안 수행됩니다.*

동일한 실행 중인 게스트에서 하나의 ***vCPU****를 핫 플러그하려면 다음을 실행합니다.*

```
virsh setvcpus guestVM1 1 --live
```

그러나 현재 ***vCPU*** *핫 플러그를 사용하면* ***vCPU*** *수를 추가로 수정하는 데 문제가 발생할 수 있습니*

다.

14.13.7. 메모리 할당 구성

virsh 로 게스트 가상 머신의 메모리 할당을 수정하려면 다음을 수행합니다.

```
# virsh setmem {domain-id or domain-name} count
```

```
# virsh setmem vr-rhel6u1-x86_64-kvm --kilobytes 1025000
```

수를 킬로바이트 로 지정해야 합니다. 새 개수 값은 게스트 가상 머신을 생성할 때 지정한 양을 초과할 수 없습니다. **64MB**보다 낮은 값은 대부분의 게스트 가상 머신 운영 체제에서는 작동하지 않습니다. 최대 메모리 값이 활성 게스트 가상 머신에는 영향을 미치지 않습니다. 새 값이 사용 가능한 메모리보다 작으면 게스트 가상 머신이 충돌하게 될 수 있습니다.

이 명령에는 다음 옵션이 있습니다.

- **[--domain] <string>** 도메인 이름, **ID** 또는 **uuid**
- **[-size] <number>** 새 메모리 크기, 크기 조정된 정수(기본값 **KiB**)

  유효한 메모리 단위는 다음과 같습니다.

  - 바이트 의 **b** 또는 바이트
  - **KB for kilobytes (10** 또는 **1,000** 바이트 블록)
  - **k ibibytes**의 경우 **kibibytes 1024**바이트의 **K** 또는 **KiB**
  - **MB for megabytes (10^6** 또는 **1,000,000** 바이트 블록)

- 메비바이트의 경우 **m** 또는 **MiB** (**2^{20}** 또는 **blocks 1,048,576** 바이트)
- **GB for gigabytes** (**10^{9}** 또는 **1,000,000,000** 바이트 블록)
- 기비바이트의 경우 **G** 또는 **GiB** (**2^{30}** 또는 **1,073,741,824** 바이트)
- **TB for terabytes** (**10^{12}** 또는 **1,000,000** 바이트 블록)
- **tebibytes**의 경우 **t or TiB** (**2^{40}** 또는 **1,099,511,627,776** 바이트)

모든 값은 **libvirt**에서 가장 가까운 **kibibyte**로 반올림되며 하이퍼바이저에서 지원하는 단위로 더 반올림될 수 있습니다. 일부 하이퍼바이저는 또한 **4000KiB**(또는 **4000 x 2^{10}** 또는 **4,096,000**바이트)와 같은 최소 용량을 적용합니다. 이 값의 단위는 선택적 특성 메모리 단위 에 따라 결정됩니다. 기본값은 **kibibytes(KiB)** 단위로 지정된 값이 **1024**바이트의 **2**개 또는 블록으로 곱한 단위입니다.

- **--config**가 다음 부팅에 영향을 미칩니다.
- **--live**는 실행 중인 도메인의 메모리를 제어합니다.
- **--current**가 현재 도메인의 메모리를 제어합니다.

14.13.8. 도메인의 메모리 할당 변경

virsh setmaxmem 도메인 크기 **--config --live --current** 를 사용하면 다음과 같이 게스트 가상 머신에 대한 최대 메모리 할당을 설정할 수 있습니다.

```
virsh setmaxmem rhel6 1024 --current
```

최대 메모리에 지정할 수 있는 크기는 지원되는 접미사가 제공되지 않는 한 기본적으로 **kibibytes**로 표시되는 스케일링된 정수입니다. 다음 옵션은 이 명령과 함께 사용할 수 있습니다.

- **--config -** 다음 부팅에 영향을 미칩니다.

- **--live -** 실행 중인 도메인의 메모리를 제어하고, 하이퍼바이저에서 모든 하이퍼바이저가 최대 메모리 제한을 실시간으로 변경할 수 있는 것은 아니므로 이 작업을 지원합니다.

- **--current -** 현재 도메인에서 메모리를 제어합니다.

14.13.9. 게스트 가상 머신 블록 장치 정보 표시

virsh domblkstat 를 사용하여 실행 중인 게스트 가상 시스템에 대한 블록 장치 통계를 표시합니다.

```
# virsh domblkstat GuestName block-device
```

14.13.10. 게스트 가상 머신 네트워크 장치 정보 표시

virsh domifstat 를 사용하여 실행 중인 게스트 가상 시스템에 대한 네트워크 인터페이스 통계를 표시합니다.

```
# virsh domifstat GuestName interface-device
```

14.14. 가상 네트워크 관리

이 섹션에서는 **virsh** 명령을 사용하여 가상 네트워크 관리에 대해 설명합니다. 가상 네트워크를 나열하려면 다음을 수행합니다.

```
# virsh net-list
```

이 명령은 다음과 유사한 출력을 생성합니다.

```
# virsh net-list
Name                 State      Autostart
-----------------------------------------
default              active     yes
vnet1                active     yes
vnet2                active     yes
```

특정 가상 네트워크에 대한 네트워크 정보를 보려면 다음을 수행합니다.

```
# virsh net-dumpxml NetworkName
```

*이렇게 하면 **XML** 형식의 지정된 가상 네트워크에 대한 정보가 표시됩니다.*

```
# 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>
```

*가상 네트워크 관리에 사용되는 기타 **virsh** 명령은 다음과 같습니다.*

- ***virsh net-autostart network-name - network-name** 으로 지정된 네트워크를 자동 시작합니다.*
- ***virsh net-create XMLfile** - 기존 **XML** 파일을 사용하여 새 네트워크를 생성하고 시작합니다.*
- ***virsh net-define XMLfile** - 시작하지 않고 기존 **XML** 파일에서 새 네트워크 장치를 생성합니다.*
- ***virsh net-destroy network-name - network-name** 으로 지정된 네트워크를 삭제합니다.*
- ***virsh net-name networkUUID** - 지정된 네트워크**UUID** 를 네트워크 이름으로 변환합니다.*
- ***virsh net-uuid network-name** - 지정된 **network-name** 을 네트워크 **UUID**로 변환합니다.*
- ***virsh net-start nameOfInactiveNetwork** - 비활성 네트워크를 시작합니다.*

- **virsh net-undefine nameOfInactiveNetwork** - 비활성 네트워크의 정의를 제거합니다.

14.15. VIRSH를 사용하여 게스트 가상 머신 마이그레이션

virsh를 사용한 마이그레이션에 대한 정보는 **virsh**를 통한 라이브 **KVM** 마이그레이션 섹션에 있습니다. 4.4절. “virsh를 통한 실시간 KVM 마이그레이션”

14.15.1. 인터페이스 명령

다음 명령은 게스트 가상 시스템에서 호스트 인터페이스를 조작하지 않아야 합니다. 이러한 명령은 호스트 물리적 시스템의 터미널에서 실행해야 합니다.

주의

이 섹션의 명령은 시스템에 **NetworkManager** 서비스가 비활성화되고 대신 네트워크 서비스를 사용하는 경우에만 지원됩니다.

이러한 호스트 인터페이스는 도메인 인터페이스 요소(예: 시스템 생성 브리지 **<인터페이스>**) 내에서 이름별로 사용할 수 있지만 특정 게스트 구성 **XML**에 호스트 인터페이스를 연결해야 하는 요구 사항은 없습니다. 호스트 인터페이스에 대한 대부분의 명령은 도메인에 사용되는 명령과 유사하며 인터페이스의 이름을 지정하는 방법은 이름 또는 **MAC** 주소입니다. 그러나 **iface** 옵션에는 **MAC** 주소를 사용하는 것이 해당 주소가 고유한 경우에만 작동합니다(인터페이스와 브리지가 동일한 **MAC** 주소를 공유하는 경우 종종 해당 **MAC** 주소를 사용하면 모호한 오류로 인해 오류가 발생하고 대신 이름을 사용해야 합니다).

14.15.1.1. XML 파일을 통해 호스트 물리적 시스템 인터페이스 정의 및 시작

virsh iface-define file 명령은 **XML** 파일에서 호스트 인터페이스를 정의합니다. 이 명령은 인터페이스만 정의하고 시작할 수 없습니다.

```
virsh iface-define iface.xml
```

이미 정의된 인터페이스를 시작하려면 **iface-start** 인터페이스를 실행합니다. 여기서 **interface** 는 인터페이스 이름입니다.

14.15.1.2. 호스트 인터페이스에 대한 XML 구성 파일 편집

명령 **iface-edit** 인터페이스는 호스트 인터페이스에 대한 **XML** 구성 파일을 편집합니다. 이 방법은 **XML** 구성 파일을 편집하는 데 권장되는 유일한 방법입니다. (이러한 파일에 대한 자세한 내용은 20장. 도메인 XML 조작 를 참조하십시오.)

14.15.1.3. 활성 호스트 인터페이스 나열

iface-list --inactive --all 은 활성 호스트 인터페이스 목록을 표시합니다. **--all** 이 지정되면 이 목록에 정의되었지만 비활성 상태인 인터페이스도 포함됩니다. **--inactive** 가 지정된 경우 비활성 인터페이스만 나열됩니다.

14.15.1.4. MAC 주소를 인터페이스 이름으로 변환

MAC 주소가 호스트 인터페이스 인터페이스 간에 고유하면 **iface-name interface**명령은 호스트 인터페이스 **MAC**를 인터페이스 이름으로 변환합니다. 이 명령에는 인터페이스 의 **MAC** 주소가 필요합니다.

iface-mac interface 명령은 호스트의 인터페이스 이름을 **MAC** 주소로 변환합니다. 여기서 이 경우 인터페이스 이름은 인터페이스 이름입니다.

14.15.1.5. 특정 호스트 물리적 시스템 인터페이스 중지

virsh iface-destroy interface 명령은 지정된 호스트 인터페이스를 제거(중지)합니다. 이는 호스트에서 **if-down** 을 실행하는 것과 동일합니다. 이 명령은 해당 인터페이스가 활성 사용에서 비활성화되고 즉시 적용됩니다.

인터페이스를 정의 해제하려면 인터페이스 이름과 함께 **iface-undefine interface** 명령을 사용합니다.

14.15.1.6. 호스트 구성 파일 표시

virsh iface-dumpxml 인터페이스 **--inactive** 는 호스트 인터페이스 정보를 **stdout**에 **XML** 덤프로 표시합니다. **--inactive** 옵션을 지정하면 다음에 시작될 때 사용할 인터페이스의 영구 상태가 출력에 반영됩니다.

14.15.1.7. 브리지 장치 만들기

iface-bridge 는 **bridge**라는 브릿지 장치를 만들고 기존 네트워크 장치 인터페이스를 새 브릿지에 연결하고 **STP**가 활성화되고 지연이 **0**인 경우 즉시 작동을 시작합니다.

```
# virsh iface-bridge interface bridge --no-stp delay --no-start
```

이러한 설정은 **--no-stp, --no-start** 및 지연을 위한 정수 시간(초)을 사용하여 변경할 수 있습니다. 인터페이스의 모든 **IP** 주소 구성이 새 브리지 장치로 이동합니다. 브릿지 제거에 대한 정보는 **14.15.1.8**절. "브리지 장치 삭제" 를 참조하십시오.

14.15.1.8. 브리지 장치 삭제

iface-un bridge bridge --no-start 명령에서 **bridge** 라는 지정된 브릿지 장치를 종료하고 기본 인터페이스를 다시 일반 사용법으로 해제하고 모든 **IP** 주소 구성을 브리지 장치에서 기본 장치로 이동합니다. **--no-start** 옵션을 사용하지 않는 한 기본 인터페이스는 재시작되지만, 일반적으로 다시 시작하지 않는 것은 권장되지 않습니다. 브리지를 만드는 데 사용하는 명령은 **14.15.1.7**절. "브리지 장치 만들기" 를 참조하십시오.

14.15.1.9. 인터페이스 스냅샷 조작

iface-begin 명령은 현재 호스트 인터페이스 설정의 스냅샷을 생성하여 나중에 커밋**(sace -commit)** 또는 복원**(iface-rollback)**할 수 있습니다. 스냅샷이 이미 존재하는 경우 이전 스냅샷이 커밋되거나 복원될 때까지 이 명령이 실패합니다. 정의되지 않은 동작이 스냅샷과 최종 커밋 또는 롤백 시간 사이에 **libvirt API** 외부의 호스트 인터페이스에 대한 외부 변경 사항이 생성되는 경우 발생합니다.

iface-commit 명령을 사용하여 마지막 **iface-begin** 이후의 모든 변경 사항을 작동으로 선언한 다음 롤백 지점을 삭제합니다. **iface-begin** 을 통해 이미 인터페이스 스냅샷이 시작되지 않은 경우 이 명령은 실패합니다.

iface-begin 명령을 마지막으로 실행한 상태로 되돌리려면 **iface-rollback** 을 사용하여 모든 호스트 인터페이스 설정을 다시 되돌립니다. **iface-begin** 명령이 이전에 실행되지 않은 경우 **iface-rollback** 이 실패합니다. 호스트 물리적 시스템을 재부팅하는 것도 암시적 롤백 포인트로 사용됩니다.

14.15.2. 스냅샷 관리

다음 섹션에서는 도메인 스냅샷을 조작하기 위해 수행할 수 있는 작업을 설명합니다. 스냅샷 은 지정된 시점에서 도메인의 디스크, 메모리, 장치 상태를 가져와서 나중에 사용할 수 있도록 저장합니다. 스냅샷에는 많은 용도가 있습니다. **OS** 이미지의 **"clean"** 복사본을 저장하여 도메인 상태를 절약할 수 있습니다. 스냅샷은 고유한 이름으로 식별됩니다. 스냅샷의 속성을 나타내는 데 사용되는 **XML** 형식에 대한 문서는 **libvirt** 웹 사이트를 참조하십시오.

14.15.2.1. 스냅샷 생성

virsh snapshot-create 명령은 도메인 **XML** 파일(예: **<name>** 및 **<description>** 요소, **<disks>** 등)에 지정된 속성을 사용하여 도메인용 스냅샷을 생성합니다.

스냅샷을 생성하려면 다음을 실행합니다.

```
# snapshot-create <domain> <xmlfile> [--redefine] [--current] [--no-metadata] [--reuse-external]
```

도메인 이름, **ID** 또는 **UID**를 도메인 요구 사항으로 사용할 수 있습니다. **XML** 요구 사항은 **<name>**, **<description>** 및 **<disks>** 요소를 포함해야 하는 문자열입니다.

참고

Red Hat Enterprise Linux에서는 실시간 스냅샷이 지원되지 않습니다. **libvirt**에는 표시되지만 **Red Hat Enterprise Linux 6**에서는 지원되지 않는 실시간 스냅샷과 함께 사용할 수 있는 **virsh snapshot-create** 명령에 추가 옵션이 있습니다.

Red Hat Enterprise Linux에서 제공되는 옵션은 다음과 같습니다.

- **--redefine** 은 **snapshot-dumpxml** 에 의해 생성된 모든 **XML** 요소가 유효한 경우, 스냅샷 계층을 한 머신에서 다른 시스템으로 마이그레이션하는 데 사용할 수 있습니다. 임시 도메인의 경우 계층 구조를 다시 만들고 나중에 동일한 이름과 **UUID**로 다시 생성되거나 스냅샷 메타데이터(예: 스냅샷에 포함된 도메인 **XML**의 호스트별 측면)에서 약간의 변경을 수행하는 데 사용할 수 있습니다. 이 옵션을 제공하면 **xmlfile** 인수가 필수이며 **--current** 옵션도 제공되지 않는 한 도메인의 현재 스냅샷이 변경되지 않습니다.

- **--no-metadata** 가 스냅샷을 생성하지만 메타데이터는 즉시 취소되지만(즉, **libvirt**는 현재 상태로 취급하지 않으며 **--redefine** 이 나중에 메타데이터에 대해 **libvirt** 를 다시 지시하는 데 사용되지 않는 한) 스냅샷으로 되돌릴 수 없습니다.

- **--reuse-external**, 사용하는 경우 이 옵션은 사용할 기존 외부 **XML** 스냅샷의 위치를 지정합니다. 기존 외부 스냅샷이 아직 없는 경우 기존 파일의 콘텐츠가 손실되지 않도록 명령을 실행하면 스냅샷을 생성하지 못합니다.

14.15.2.2. 현재 도메인의 스냅샷 생성

virsh snapshot-create-as 도메인 명령은 도메인 **XML** 파일(예: **<name>** 및 **<description>** 요소)에 지정된 속성을 사용하여 도메인의 스냅샷을 생성합니다. 이러한 값이 **XML** 문자열에 포함되지 않은 경우 **libvirt** 에서 값을 선택합니다. 스냅샷 실행을 생성하려면 다음을 수행합니다.

```
# virsh snapshot-create-as domain {[--print-xml] | [--no-metadata] [--reuse-external]} [name]
[description] [--diskspec] diskspec]
```

나머지 옵션은 다음과 같습니다.

- **--print-xml** 은 실제로 스냅샷을 생성하지 않고 스냅샷 생성에 적절한 **XML**을 생성합니다.

- **--diskspec** 옵션을 사용하여 **--disk-only** 및 외부 체크포인트에서 외부 파일을 생성하는 방법을 제어할 수 있습니다. 이 옵션은 도메인 **XML**의 **<disk>** 요소 수에 따라 여러 번 발생할 수 있습니다. 각 **<diskspec>**은 디스크**[,snapshot=type][,driver=type][,file=name]**) 형식으로 되어 있습니다. 리터럴 쉼표를 디스크 또는 **file=name** 에 포함하려면 두 번째 쉼표로 이스케이프합니다. 리터럴 **--diskspec** 은 **<domain>, <name>, <description>** 중 세 개가 모두 존재하지 않는 한 각 **diskspec** 앞에 있어야 합니다. 예를 들어 **vda,snapshot=external,file=/path/to**의 **diskspec** 은 다음 **XML**로 생성 됩니다.

```
<disk name='vda' snapshot='external'>
  <source file='/path/to,new'/>
</disk>
```

- **--reuse-external** 은 기존 파일을 대상으로 재사용합니다(이 파일을 덮어쓰는 참조). 이 대상이 존재하지 않는 경우 기존 파일의 콘텐츠가 손실되지 않도록 스냅샷 요청이 거부됩니다.

- **--no-metadata** 는 스냅샷 데이터를 생성하지만 메타데이터는 즉시 삭제됩니다(즉, **libvirt** 는 현재 상태로 취급하지 않으며, 이후 메타데이터에 대해 **libvirt**에 대해 **libvirt**를 처리하는 데 사용 안 함)을 제외하고 스냅샷으로 되돌릴 수 없습니다. 이 옵션은 **--print-xml** 과 호환되지 않습니다.

14.15.2.3. 현재 도메인의 스냅샷 가져오기

이 명령은 현재 사용 중인 스냅샷을 쿼리하는 데 사용됩니다. 사용하려면 다음을 실행합니다.

```
# virsh snapshot-current domain {[--name] | [--security-info] | [snapshotname]}
```

snapshotname 을 사용하지 않으면 도메인의 현재 스냅샷(있는 경우)에 대한 스냅샷 **XML**이 출력으로 표시됩니다. **--name** 을 지정하면 전체 **XML** 대신 현재 스냅샷 이름만 출력으로 전송됩니다. **--security-info** 를 제공하면 민감한 보안 정보가 **XML**에 포함됩니다. **libvirt** 는 스냅샷 이름을 사용하여 기존 이름의 스냅샷이 현재 스냅샷이 되도록 하는 요청을 생성합니다.

14.15.2.4. snapshot-edit-domain

이 명령은 현재 사용 중인 스냅샷을 편집하는 데 사용됩니다. 사용하려면 다음을 실행합니다.

```
#virsh snapshot-edit domain [snapshotname] [--current] {[--rename] [--clone]}
```

snapshotname 및 **--current** 가 모두 지정된 경우 편집된 스냅샷이 현재 스냅샷이 되도록 강제 적용합니다. **snapshotname** 을 생략하면 현재 스냅샷을 편집하려면 **--current** 를 제공해야 합니다.

이는 아래의 다음 명령 순서와 동일하지만 몇 가지 오류 검사도 포함됩니다.

```
# 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 을 지정하면 결과 편집한 파일이 다른 파일 이름에 저장됩니다. **--clone** 을 지정하면 스냅샷 이름을 변경하면 스냅샷 메타데이터의 복제본이 생성됩니다. 둘 다 지정하지 않으면 편집에서 스냅샷 이름을 변경하지 않습니다. 단일 **qcow2** 파일 내의 내부 스냅샷과 같은 일부 스냅샷의 내용은 원본 스냅샷 파일 이름에서만 액세스할 수 있으므로 스냅샷 이름을 변경하는 작업은 주의해야 합니다.

14.15.2.5. snapshot-info-domain

snapshot-info-domain 은 스냅샷에 대한 정보를 표시합니다. 사용하려면 다음을 실행합니다.

```
# snapshot-info domain {snapshot | --current}
```

지정된 스냅샷 또는 **--current** 를 사용하여 현재 스냅샷에 대한 기본 정보를 출력합니다.

14.15.2.6. snapshot-list-domain

지정된 도메인에 대해 사용 가능한 모든 스냅샷을 나열하고, 기본적으로 스냅샷 이름, 생성 시간 및 도메인 상태에 대한 열을 표시합니다. 사용하려면 다음을 실행합니다.

```
#virsh snapshot-list domain [{--parent | --roots | --tree}] [{[--from] snapshot | --current} [--
descendants]] [--metadata] [--no-metadata] [--leaves] [--no-leaves] [--inactive] [--active] [--internal] [--
external]
```

나머지 선택적 옵션은 다음과 같습니다.

-

--parent 는 각 스냅샷의 상위 이름을 제공하는 열을 출력 테이블에 추가합니다. 이 옵션은 --roots 또는 --tree 와 함께 사용할 수 없습니다.

- *--roots 는 목록을 필터링하여 부모가 없는 스냅샷만 표시합니다. 이 옵션은 --parent 또는 --tree 와 함께 사용할 수 없습니다.*

- *--tree 는 스냅샷 이름만 나열하는 트리 형식으로 출력을 표시합니다. 이 세 가지 옵션은 함께 사용할 수 없습니다. 이 옵션은 --roots 또는 --parent 와 함께 사용할 수 없습니다.*

- *--from 은 지정된 스냅샷의 하위 스냅샷으로 목록을 필터링하거나 --current 가 제공되는 경우 현재 스냅샷에서 목록이 시작됩니다. 격리 또는 --parent 와 함께 사용하면 --descendants 도 없는 한 목록은 직접 자식으로 제한됩니다. --tree 와 함께 사용하면 --descendants 를 사용할 수 없습니다. 이 옵션은 --roots 와 호환되지 않습니다. --from 또는 --current 의 시작점은 --tree 옵션이 없으면 목록에 포함되지 않습니다.*

- *--leaves 가 지정되면 목록이 자식이 없는 스냅샷으로 필터링됩니다. 마찬가지로 --no-leaves 가 지정된 경우 목록이 하위 항목이 있는 스냅샷으로 필터링됩니다. (두 옵션을 모두 생략해도 필터링은 필터링되지 않지만 두 옵션을 제공하면 서버에서 옵션을 인식하는지 여부에 따라 동일한 목록 또는 오류가 생성됩니다.) 필터링 옵션은 --tree 와 호환되지 않습니다.*

- *--metadata 가 지정되고, 이 목록은 libvirt 메타데이터와 관련된 스냅샷으로 필터링되므로 영구 도메인을 정의하지 않거나 일시적인 도메인 삭제 시 손실되지 않습니다. 마찬가지로 --no-metadata 가 지정된 경우 libvirt 메타데이터없이 존재하는 스냅샷으로만 목록이 필터링됩니다.*

- *--inactive 가 지정되면 목록이 도메인이 종료될 때 수행된 스냅샷으로 필터링됩니다. --active 가 지정된 경우, 도메인 실행 시 수행된 스냅샷으로 목록이 필터링되고 스냅샷에 해당 실행 상태로 되돌리는 메모리 상태가 포함됩니다. --disk-only 가 지정된 경우 도메인을 실행할 때 가져온 스냅샷으로 목록이 필터링되지만 스냅샷에는 디스크 상태만 포함됩니다.*

- *--internal 이 지정되면 기존 디스크 이미지의 내부 스토리지를 사용하는 스냅샷으로 목록이 필터링됩니다. --external을 지정하면 디스크 이미지 또는 메모리 상태에 외부 파일을 사용하는 스냅샷으로 목록이 필터링됩니다.*

14.15.2.7. snapshot-dumpxml 도메인 스냅샷

virsh snapshot-dumpxml 도메인 스냅샷 은 snapshot라는 도메인의 스냅샷에 대한 스냅샷 XML을 출력합니다. 사용하려면 다음을 실행합니다.

```
# virsh snapshot-dumpxml domain snapshot [--security-info]
```

security-info 옵션에는 보안에 중요한 정보도 포함됩니다. **snapshot-current** 를 사용하여 현재 스냅샷의 **XML**에 쉽게 액세스할 수 있습니다.

14.15.2.8. snapshot-parent 도메인

지정된 스냅샷 또는 **--current** 을 사용하여 현재 스냅샷에 대해 상위 스냅샷의 이름을 출력합니다. 사용하려면 다음을 실행합니다.

```
#virsh snapshot-parent domain {snapshot | --current}
```

14.15.2.9. snapshot-revert 도메인

지정된 도메인을 스냅샷에 의해 지정된 스냅샷 또는 **--current** 을 사용하여 현재 스냅샷으로 되돌립니다.

주의

이는 파괴적인 동작입니다. 마지막 스냅샷이 수행된 이후 도메인의 모든 변경 사항이 손실됩니다. 또한 원래 스냅샷을 만든 시점의 도메인 상태가 **snapshot-revert** 가 완료된 후의 도메인 상태가 됩니다.

스냅샷을 되돌리려면 다음을 실행합니다.

```
# snapshot-revert domain {snapshot | --current} [{--running | --paused}] [--force]
```

일반적으로 스냅샷으로 되돌리면 스냅샷이 생성된 시점의 상태로 도메인을 남겨 둡니다. 단, 게스트 가상 시스템 상태가 없는 디스크 스냅샷은 도메인을 비활성 상태로 유지합니다. **--running** 또는 **--paused** 옵션을 전달하면 비활성 도메인 부팅 또는 실행 중인 도메인 일시 중지와 같은 추가 상태 변경 사항이 수행됩니다. 임시 도메인은 비활성화할 수 없으므로 임시 도메인의 디스크 스냅샷으로 되돌릴 때 이러한 옵션 중 하나를 사용해야 합니다.

스냅샷을 되돌리려면 **--force** 를 사용하여 진행해야 하는 추가 위험이 있는 두 가지 경우가 있습니다. 하나는 구성을 되돌리기 위한 전체 도메인 정보가 없는 스냅샷입니다. **libvirt** 는 현재 구성의 스냅샷에서 사용 중인 구성과 일치한다는 것을 증명할 수 없으므로 **--force** 는 스냅샷이 현재 구성과 호환되는지 확인(및 그렇지 않으면 도메인이 실행되지 않을 가능성이 큽니다). 다른 하나는 실행 중인 도메인에서 기존 **VNC** 또는 **Spice** 연결 중단과 같은 단점을 의미하는 대신 기존 하이퍼바이저를 재사용하는 대신 새 하이

퍼바이저를 생성해야 하는 활성 상태로 되돌리는 경우입니다. 이 조건은 매개 변수가 호환되지 않는 구성과 함께 결합된 비활성 스냅샷과 함께 **--start** 또는 **--pause** 옵션과 결합된 비활성 스냅샷과 함께 발생합니다.

14.15.2.10. snapshot-delete 도메인

snapshot-delete domain 은 지정된 도메인의 스냅샷을 삭제합니다. 이렇게 하려면 다음을 실행합니다.

```
# virsh snapshot-delete domain {snapshot | --current} [--metadata] [{--children | --children-only}]
```

이 명령은 스냅샷이라는 도메인의 스냅샷을 삭제하거나 **--current** 을 사용하여 현재 스냅샷을 삭제합니다. 이 스냅샷에 하위 스냅샷이 있는 경우 이 스냅샷의 변경 사항이 하위 스냅샷으로 병합됩니다. **--children** 옵션을 사용하면 이 스냅샷과 이 스냅샷의 하위 항목이 삭제됩니다. **--children-only** 를 사용하는 경우 이 스냅샷은 이 스냅샷의 하위 항목을 삭제하지만 이 스냅샷을 그대로 유지합니다. 이 두 옵션은 함께 사용할 수 없습니다.

--metadata 를 사용하면 **libvirt** 에서 유지 관리하는 스냅샷의 메타데이터가 삭제되고, 외부 툴에서 액세스할 수 있도록 스냅샷 내용은 그대로 유지됩니다. 그러지 않으면 스냅샷을 삭제하면 해당 시점에서 해당 데이터 콘텐츠도 제거됩니다.

14.16. 게스트 가상 머신 CPU 모델 구성

이 섹션에서는 게스트 가상 머신 **CPU** 모델 구성에 대한 정보를 제공합니다.

14.16.1. 소개

모든 하이퍼바이저에는 기본적으로 게스트 가상 머신이 해당 **CPU**에 대해 볼 수 있는 고유한 정책이 있습니다. 일부 하이퍼바이저는 게스트 가상 머신에서 사용할 수 있는 **CPU** 호스트 물리적 머신 기능을 결정하는 반면, **QEMU/KVM**은 게스트 가상 머신에 **qemu32** 또는 **qemu64** 라는 일반 모델을 제공합니다. 이러한 하이퍼바이저는 고급 필터링을 수행하여 모든 물리적 **CPU**를 몇 개의 그룹으로 분류하고 게스트 가상 머신에 제공되는 각 그룹에 대해 하나의 기본 **CPU** 모델을 갖습니다. 이러한 동작을 통해 호스트 물리적 시스템 간에 게스트 가상 시스템을 안전하게 마이그레이션할 수 있습니다. 이 경우 모두 동일한 그룹으로 분류되는 물리적 **CPU**가 있습니다. **libvirt**는 일반적으로 정책 자체를 적용하지 않고 상위 계층이 원하는 정책을 정의하는 메커니즘을 제공합니다. **CPU** 모델 정보를 얻고 적절한 게스트 가상 머신 **CPU** 모델을 정의하는 방법을 이해하는 것은 호스트 물리적 시스템 간에 게스트 가상 시스템 마이그레이션을 성공적으로 마이그레이션하는 데 중요합니다. 하이퍼바이저는 하이퍼바이저가 릴리스된 후 생성된 기능 및 기능만 에뮬레이션할 수 있습니다.

14.16.2. 호스트 물리적 시스템 CPU 모델에 대해 알아보기

virsh capabilities 명령은 하이퍼바이저 연결 및 호스트 물리적 시스템의 기능을 설명하는 **XML** 문서

를 표시합니다. 호스트 물리적 머신 CPU 모델에 대한 정보를 제공하기 위해 표시되는 XML 스키마가 확장되었습니다. CPU 모델을 설명하는 데 큰 문제 중 하나는 모든 아키텍처에서 기능을 노출하는 다른 접근 방식을 가지고 있다는 것입니다. x86에서 최신 CPU의 기능은 CPUID 명령을 통해 노출됩니다. 기본적으로 이 값은 각각 비트마다 특정 의미를 부여한 32비트 정수 집합에 추가됩니다. 다행히도 AMD와 Intel은 이러한 문제에 대한 일반적인 의미 체계에 동의합니다. 다른 하이퍼바이저는 게스트 가상 머신 구성 형식에 CPUID 마스크의 개념을 직접 노출합니다. 그러나 QEMU/KVM은 x86 아키텍처 이상의 기능을 지원하므로 CPUID는 정식 구성 형식으로 적합하지 않습니다. QEMU는 CPU 모델 이름 문자열과 이름이 지정된 옵션 집합을 결합하는 체계를 사용하여 종료되었습니다. x86에서 CPU 모델은 기준 CPUID 마스크에 매핑되고, 옵션을 사용하여 마스크의 비트를 설정하거나 해제하는 데 사용할 수 있습니다. libvirt는 이 리드를 따르고 모델 이름과 옵션의 조합을 사용하기로 결정했습니다.

알려진 모든 CPU 모델을 나열하는 데이터베이스가 실용적이지 않으므로 libvirt에 기본 CPU 모델 이름이 작은 목록이 있습니다. 실제 호스트 물리적 시스템 CPU와 CPUID의 최대 수를 공유하는 CPUID 비트를 선택한 다음 나머지 비트를 명명된 기능으로 나열합니다. libvirt는 기준 CPU에 포함된 기능을 표시하지 않습니다. 처음에는 결함이있는 것처럼 보일 수 있지만 이 섹션에서 설명하는 것처럼, 실제로이 정보를 알 필요가 없습니다.

14.16.3. 호스트 물리적 머신 풀을 지원할 수 있는 호환되는 CPU 모델 확인

이제 단일 호스트 물리적 시스템의 CPU 기능을 확인할 수 있으므로 다음 단계는 게스트 가상 머신에 가장 적합한 CPU 기능을 결정하는 것입니다. 게스트 가상 머신을 다른 호스트 물리적 시스템으로 마이그레이션할 필요가 없다는 사실을 알고 있는 경우 호스트 물리적 머신 CPU 모델을 수정하지 않은 상태로 직접 전달할 수 있습니다. 가상화된 데이터 센터에는 모든 서버의 CPU가 100% 동일하도록 보장할 수 있는 구성 세트가 있을 수 있습니다. 다시 호스트 물리적 머신 CPU 모델을 수정하지 않은 상태로 바로 전달할 수 있습니다. 그러나 일반적인 경우는 호스트 물리적 시스템 간에 CPU가 다릅니다. 이 혼합된 CPU 환경에서 가장 낮은 공통 분모 CPU를 결정해야 합니다. 이는 쉽지 않으므로 libvirt는 이 작업에 대한 API를 제공합니다. libvirt가 호스트 물리적 시스템의 CPU 모델을 설명하는 XML 문서 목록을 제공하는 경우 libvirt는 내부적으로 이 모델을 CPUID 마스크로 변환하고 교집합을 계산하고, CPUID 마스크 결과를 XML CPU 설명으로 다시 변환합니다.

다음은 virsh기능이 실행될 때 기본 워크스테이션에서 기능으로 어떤 libvirt를 보고하는지 보여줍니다.

*그림 **14.3.** 호스트 물리적 시스템의 **CPU** 모델 정보 가져오기*

```
<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>
```

*이제 동일한 **virsh capabilities** 명령을 사용하여 임의의 임의의 서버와 비교합니다.*

그림 14.4. 임의의 서버에서 CPU 설명 생성

```
<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...
```

이 CPU 설명이 이전 워크스테이션 CPU 설명과 호환되는지 확인하려면 virsh cpu-compare 명령을 사용합니다.

감소된 콘텐츠는 virsh-caps- workstation-cpu-only.xml 이라는 파일에 저장되었으며 virsh cpu-compare 명령은 이 파일에서 실행할 수 있습니다.

```
# 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
```

이 출력에서 볼 수 있듯이 libvirt 는 CPU가 엄격하게 호환되지 않음을 올바르게 보고합니다. 서버 CPU에 클라이언트 CPU가 없는 여러 기능이 있기 때문입니다. 클라이언트와 서버 간에 마이그레이션할 수 있으려면 XML 파일을 열고 일부 기능을 주석 처리해야 합니다. 제거해야 하는 기능을 확인하려면 두 시스템의 CPU 정보가 포함된 both-cpus.xml 에서 virsh cpu-baseline 명령을 실행합니다. # virsh cpu-baseline both-cpus.xml 을 실행하면 다음과 같은 결과가 나타납니다.

그림 14.5. 복합 CPU 기준

```
<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>
```

이 복합 파일은 어떤 요소가 공통인지 보여줍니다. 공통적이지 않은 모든 것은 주석 처리해야합니다.

14.17. 게스트 가상 머신 CPU 모델 구성

간단한 기본값의 경우 게스트 가상 머신 CPU 구성은 호스트 물리 머신 기능 XML 노출과 동일한 기본 XML 표현을 허용합니다. 즉, cpu-baseline virsh 명령의 XML을 <domain> 요소의 최상위 수준에 있는 게스트 가상 머신 XML로 직접 복사할 수 있습니다. 이전 XML 스니펫에는 게스트 가상 머신 XML에서 CPU를 설명할 때 사용할 수 있는 몇 가지 추가 속성이 있습니다. 대부분은 무시할 수 있지만, 여기서 호기심스러운 것은 무엇을 하는지에 대한 간략한 설명입니다. 최상위 수준 <cpu> 요소에는 가능한 값이 있는 match 속성이 있습니다.

- *match='minimum' - 호스트 물리적 머신 CPU에는 게스트 가상 머신 XML에 설명된 CPU 기능이 최소한 있어야 합니다. 호스트 물리적 시스템에 게스트 가상 머신 구성 이외의 추가 기능이 있는 경우 게스트 가상 머신에도 노출됩니다.*

- *match='exact' - 호스트 물리적 머신 CPU에는 게스트 가상 머신 XML에 설명된 CPU 기능이 있어야 합니다. 호스트 물리적 시스템에 게스트 가상 머신 구성 이외의 추가 기능이 있는 경우 게스트 가상 시스템에서 마스킹됩니다.*

- *match='strict' - 호스트 물리적 머신 CPU에는 게스트 가상 머신 XML에 설명된 것과 동일한 CPU 기능이 있어야 합니다.*

다음 개선 사항은 <feature> 요소가 각각 가능한 값이 있는 추가 'policy' 특성을 보유할 수 있다는 것입니다.

- *policy='force' - 호스트 물리적 시스템에 없는 경우에도 해당 기능을 게스트 가상 시스템에 노*

출합니다. 이는 일반적으로 소프트웨어 에뮬레이션의 경우에만 유용합니다.

- policy='require' - 이 기능을 게스트 가상 시스템에 노출하고 호스트 물리적 시스템에 없는 경우 실패합니다. 이는 민감한 기본값입니다.
- policy='optional' - 기능을 지원하는 경우 게스트 가상 머신에 기능을 노출합니다.
- policy='disable' - 호스트 물리적 시스템에 이 기능이 있는 경우 게스트 가상 시스템에서 숨겨집니다.
- policy='forbid' - 호스트 물리적 시스템에 이 기능이 있는 경우 실패하고 게스트 가상 머신 시작을 거부합니다.

'forbid' 정책은 응용 프로그램이 CPUID 마스크에 없는 경우에도 기능을 사용하려고 하는 틈새 시나리오를 위한 것이며 해당 기능이 있는 호스트 물리적 시스템에서 게스트 가상 시스템을 실수로 실행하지 않도록 하려는 경우입니다. '선택 사항' 정책에는 마이그레이션과 관련된 특수한 동작이 있습니다. 게스트 가상 머신이 처음 시작되는 경우 매개 변수가 선택 사항이지만 게스트 가상 머신이 실시간 마이그레이션 되는 경우 마이그레이션 전체에서 기능이 손실되지 않았기 때문에 이 정책은 '요청'으로 전환됩니다.

14.18. 게스트 가상 머신용 리소스 관리

virsh 를 사용하면 게스트 가상 시스템별로 리소스를 그룹화하고 할당할 수 있습니다. 이는 libvirt 데몬에서 관리하며, 이 데몬은 cgroup 을 생성하고 게스트 가상 머신을 대신하여 관리합니다. 시스템 관리자가 수행할 유일한 작업은 지정된 게스트 가상 머신에 대해 튜닝 가능 항목을 쿼리하거나 설정하는 것입니다. 다음과 같은 튜닝 가능 항목을 사용할 수 있습니다.

- 메모리 - 메모리 컨트롤러에서 RAM 및 스왑 사용에 대한 제한을 설정하고 그룹에 있는 모든 프로세스의 누적 사용량을 쿼리할 수 있습니다.
- cpuset - CPU 세트 컨트롤러는 그룹 내의 프로세스를 CPU 집합에 바인딩하고 CPU 간 마이그레이션을 제어합니다.
- cpuacct - CPU 계정 컨트롤러는 프로세스 그룹의 CPU 사용량에 대한 정보를 제공합니다.
- CPU - CPU 스케줄러 컨트롤러는 그룹에서 프로세스의 우선 순위를 제어합니다. 이는 nice 수준 권한을 부여하는 것과 유사합니다.

- *장치 - 장치 컨트롤러는 문자 및 블록 장치에 대한 액세스 제어 목록을 부여합니다.*

- *freezer - freezer 컨트롤러가 일시 중지되고 그룹의 프로세스 실행을 재개합니다. 이는 전체 그룹의 경우 SIGSTOP 와 유사합니다.*

- *net_cls - 프로세스를 tc 네트워크 클래스와 연결하여 네트워크 클래스를 관리합니다.*

그룹 계층 cgroup을 생성할 때 마운트 지점 및 디렉터리 설정은 관리자의 재량에 전적으로 남겨지며 일부 마운트 지점을 /etc/fstab 에 추가하는 것보다 더 복잡해집니다. 디렉터리 계층 구조를 설정하고 프로세스가 그 안에 배치되는 방식을 결정해야 합니다. 이 작업은 다음 virsh 명령으로 수행할 수 있습니다.

- *schedinfo - 설명 14.19절. “일정 매개변수 설정”*

- *blkiotune- 설명 14.20절. “블록 I/O 매개 변수 표시 또는 설정”*

- *domiftune- 설명 14.5.9절. “네트워크 인터페이스 대역폭 매개 변수 설정”*

- *memtune - 설명 14.21절. “메모리 튜닝 구성”*

14.19. 일정 매개변수 설정

schedinfo 를 사용하면 스케줄러 매개변수를 게스트 가상 머신에 전달할 수 있습니다. 다음 명령 형식을 사용해야 합니다.

```
#virsh schedinfo domain --set --weight --cap --current --config --live
```

각 매개변수는 다음과 같습니다.

- *domain - 게스트 가상 머신 도메인입니다.*

- *--set - 여기에 배치된 문자열은 호출할 컨트롤러 또는 작업입니다. 필요한 경우 추가 매개변*

수 또는 값도 추가해야 합니다.

- **--current - --set** 과 함께 사용하면 지정된 **set** 문자열을 현재 스케줄러 정보로 사용합니다. 없이 사용하면 현재 스케줄러 정보가 표시됩니다.

- **--config - - - set** 과 함께 사용하면 다음 재부팅 시 지정된 **set** 문자열이 사용됩니다. 없이 사용하면 구성 파일에 저장된 스케줄러 정보가 표시됩니다.

- **--live - --set** 과 함께 사용하면 현재 실행 중인 게스트 가상 머신에서 지정된 **set** 문자열을 사용합니다. **without**를 사용하면 실행 중인 가상 머신에서 현재 사용되는 구성 설정이 표시됩니다.

스케줄러는 **cpu_shares,vcpu_period** 및 **vcpu_quota** 매개 변수를 사용하여 설정할 수 있습니다.

예 14.5. schedinfo show

이 예에서는 쉘 게스트 가상 머신의 일정 정보를 보여줍니다.

```
# virsh schedinfo shell
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
```

예 14.6. schedinfo set

이 예에서는 **cpu_shares**가 **2046**로 변경됩니다. 이는 구성 파일이 아닌 현재 상태에 영향을 미칩니다.

```
# virsh schedinfo --set cpu_shares=2046 shell
Scheduler      : posix
cpu_shares     : 2046
vcpu_period    : 100000
vcpu_quota     : -1
```

14.20. 블록 I/O 매개 변수 표시 또는 설정

blkiotune 은 지정된 게스트 가상 머신에 대한 I/O 매개변수를 설정 및 표시하거나 표시합니다. 다음 형식을 사용해야 합니다.

```
# virsh blkiotune domain [--weight weight] [--device-weights device-weights] [[--config] [--live] | [--current]]
```

이 명령에 대한 자세한 내용은 가상화 튜닝 및 최적화 가이드에서확인할 수 있습니다.

14.21. 메모리 튜닝 구성

virsh memtune virtual_machine --parameter size 는 가상화 튜닝 및 최적화 가이드에서 다룹니다.

14.22. 가상 네트워킹 명령

다음 명령은 가상 네트워크를 조작합니다. **libvirt** 에는 가상 네트워크를 정의하는 기능이 있으며, 이 기능은 도메인에서 사용하고 실제 네트워크 장치에 연결할 수 있습니다. 이 기능에 대한 자세한 내용은 **libvirt** 웹 사이트의 설명서를 참조하십시오. 가상 네트워크에 대한 대부분의 명령은 도메인에 사용되는 명령과 유사하지만 가상 네트워크의 이름을 지정하는 방법은 이름 또는 **UUID**입니다.

14.22.1. 가상 네트워크 자동 시작

이 명령은 게스트 가상 시스템이 부팅될 때 자동으로 시작할 가상 네트워크를 구성합니다. 이 명령을 실행하려면 다음을 수행합니다.

```
# virsh net-autostart network [--disable]
```

이 명령은 **autostart** 명령을 비활성화하는 **--disable** 옵션을 허용합니다.

14.22.2. XML 파일에서 가상 네트워크 생성

이 명령은 **XML** 파일에서 가상 네트워크를 생성합니다. **libvirt** 의 웹 사이트를 참조하여 **libvirt** 에서 사용하는 **XML** 네트워크 형식에 대한 설명을 가져옵니다. 이 명령 파일은 **XML** 파일의 경로입니다. **XML** 파일에서 가상 네트워크를 생성하려면 다음을 실행합니다.

```
# virsh net-create file
```

14.22.3. XML 파일에서 가상 네트워크 정의

이 명령은 **XML** 파일에서 가상 네트워크를 정의하지만 인스턴스화할 수 없습니다. 가상 네트워크를 정의하려면 다음을 실행합니다.

```
# net-define file
```

14.22.4. 가상 네트워크 중지

이 명령은 이름 또는 **UUID**로 지정된 지정된 가상 네트워크를 제거합니다. 이것은 즉시 효과를 가져옵니다. 지정된 네트워크 네트워크를 중지하려면 다음을 수행해야 합니다.

```
# net-destroy network
```

14.22.5. 덤프 파일 생성

이 명령은 가상 네트워크 정보를 지정된 가상 네트워크에 대한 **stdout**에 **XML** 덤프로 출력합니다. **--inactive** 가 지정된 경우 물리적 기능은 연결된 가상 기능으로 확장되지 않습니다. 덤프 파일을 생성하려면 다음을 실행합니다.

```
# virsh net-dumpxml network [--inactive]
```

14.22.6. 가상 네트워크의 XML 구성 파일 편집

다음 명령은 네트워크의 **XML** 구성 파일을 편집합니다.

```
# virsh net-edit network
```

XML 파일을 편집하는 데 사용되는 편집기는 **$VISUAL** 또는 **$EDITOR** 환경 변수에서 제공할 수 있으며 기본값은 **vi** 입니다.

14.22.7. 가상 네트워크에 대한 정보 가져오기

이 명령은 **network** 오브젝트에 대한 기본 정보를 반환합니다. 네트워크 정보를 가져오려면 다음을 실행합니다.

```
# virsh net-info network
```

14.22.8. 가상 네트워크에 대한 정보 나열

활성 네트워크 목록을 반환합니다. **--all** 이 이 값을 지정하면 비활성 네트워크만 나열되는 경우 **--inactive** 가 지정되지 않은 네트워크도 포함됩니다. 또한 반환된 네트워크를 **--persistent** 에서 필터링하여 관련 네트워크를 나열하고, 임시 네트워크를 나열하는 **--transient** 를 사용하여 **autostart**가 활성화된

*것을 나열하는 **--autostart** 및 **--no-autostart** 를 사용하여 **autostart**가 비활성화된 네트워크를 나열할 수도 있습니다.*

*참고: 이전 서버와 통신할 때 이 명령은 고유한 경쟁과 함께 일련의 **API** 호출을 사용해야 합니다. 여기서 풀은 목록이 수집되는 동안 호출 간에 상태를 변경하는 경우 한 번 이상 표시될 수 있습니다. 최신 서버에는 이러한 문제가 없습니다.*

가상 네트워크를 나열하려면 다음을 실행합니다.

```
# net-list [--inactive | --all] [--persistent] [<--transient>] [--autostart] [<--no-autostart>]
```

14.22.9. 네트워크 UUID를 네트워크 이름으로 변환

*이 명령은 네트워크 **UUID**를 네트워크 이름으로 변환합니다. 이 실행을 수행하려면 다음을 수행합니다.*

```
# virsh net-name network-UUID
```

14.22.10. (Previous Defined Network) Inactive Network를 시작

이 명령은 이전에 정의된 비활성 네트워크를 시작합니다. 이렇게 하려면 다음을 실행합니다.

```
# virsh net-start network
```

14.22.11. Inactive Network 구성 정의 해제

이 명령은 비활성 네트워크의 구성을 정의 취소합니다. 이렇게 하려면 다음을 실행합니다.

```
# net-undefine network
```

14.22.12. 네트워크 이름을 네트워크 UUID로 변환

*이 명령은 네트워크 이름을 네트워크 **UUID**로 변환합니다. 이렇게 하려면 다음을 실행합니다.*

```
# virsh net-uuid network-name
```

14.22.13. 기존 네트워크 정의 파일 업데이트

*이 명령은 기존 네트워크 정의의 지정된 섹션을 업데이트하여 네트워크를 삭제하고 다시 시작하지 않고도 즉시 적용됩니다. 이 명령은 **"add-first", "add-last", "add-last", "delete"** 또는 **"modify"**. 섹션은 **""bridge", "ip-dhcp-host", "ip-dhcp-host", "ip-dhcp-range", "forward-dhcp-range", "forward-interface", "forward-interface", "forward-interface"** 중 하나입니다. **"forward-pf", "portgroup", "dns-host", "dns-txt"** 또는 **"dns-srv"**는 요소가 변경되는 **xml** 요소 계층의 연결로 이름이 지정됩니다. 예를 들어, **"ip-dhcp-host"**는 네트워크의 **<ip>** 요소 내에 **<dhcp>** 요소에 포함된 **<호스트>** 요소를 변경합니다. **xml**는 변경 중인 유형의 완전한 **xml** 요소의 텍스트입니다(예: **"<host mac="00:11:11:22:33:44:55"**의 **ip='192.0.2.1'/>** 또는 파일이 포함된 파일 이름. **disambiguation**은 제공된 텍스트의 첫 번째 문자를 보면 첫 번째 문자가 **"<"**인 경우 첫 문자가 **">"**가 아닌 경우 사용할 **xml** 텍스트가 포함된 파일의 이름입니다. **parent-index** 옵션은 요청된 요소가 (0부터 시작)되는 여러 개의 부모 요소 중 어느 것을 지정하는 데 사용됩니다. 예를 들어 **<dhcp> <host>** 요소는 네트워크의 여러 **< <ip> >** 요소 중 하나일 수 있습니다. **parent-index**가 제공되지 않으면 "가장 적절한" **IP** 요소가 선택되지만 **--parent-index**가 지정된 경우 **<ip>** 의 특정 인스턴스가 수정됩니다. **--live** 가 지정된 경우 실행 중인 네트워크에 영향을 미칩니다. **--config** 를 지정하면 영구 네트워크의 다음 시작에 영향을 미칩니다. **current** 를 지정하면 현재 네트워크 상태에 영향을 미칩니다. **--live** 및 **--config** 옵션 모두 제공될 수 있지만 **--current**는 독점적입니다. 옵션을 지정하지 않는 것은 **--current** 를 지정하는 것과 동일합니다.*

구성 파일을 업데이트하려면 다음을 실행합니다.

```
# virsh net-update network command section xml [--parent-index index] [[--live] [--config] | [--current]]
```

15장. VIRTUAL MACHINE MANAGER(VIRT-MANAGER)를 사용하여 게스트 관리

이 섹션에서는 가상 머신 관리자(virt-manager) 창, 대화 상자 및 다양한 GUI 컨트롤에 대해 설명합니다.

virt-manager 는 호스트 시스템 및 원격 호스트 시스템의 하이퍼바이저 및 게스트에 대한 그래픽 보기를 제공합니다. virt-manager 는 다음을 포함한 가상화 관리 작업을 수행할 수 있습니다.

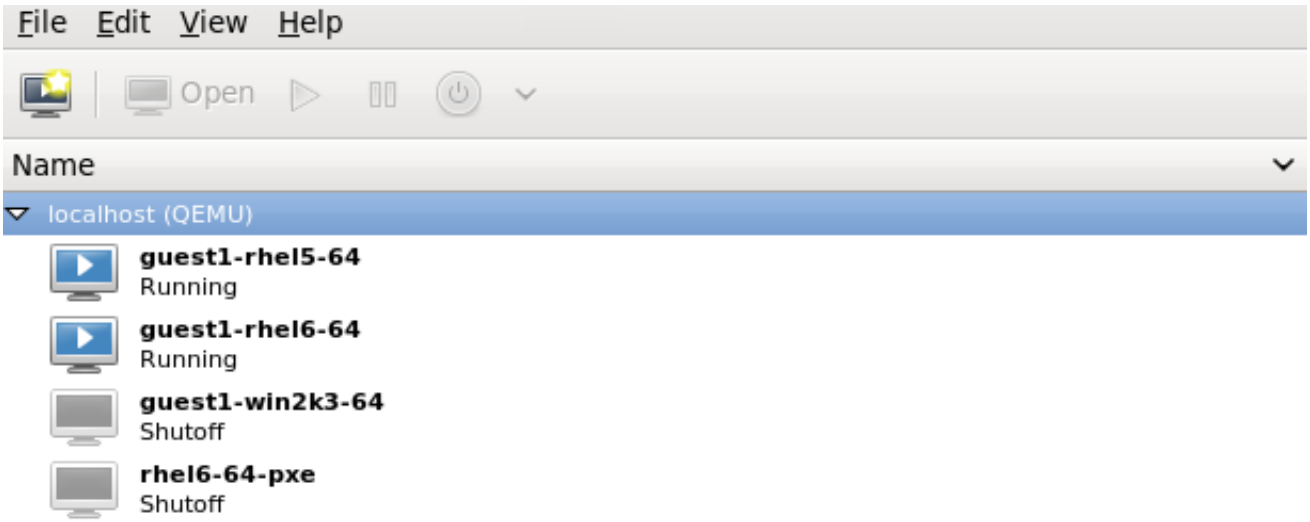
- 게스트 정의 및 생성,
- 메모리 할당,
- 가상 CPU 할당
- 운영 성능 모니터링
- 저장 및 복원, 일시 중지 및 다시 시작, 게스트 종료 및 시작,
- 텍스트 및 그래픽 콘솔에 대한 링크, 및
- 실시간 및 오프라인 마이그레이션.

15.1. VIRT-MANAGER 시작

virt-manager 세션을 시작하려면 Applications (애플리케이션) 메뉴를 연 다음 System Tools (시스템 도구) 메뉴를 열고 Virtual Machine Manager (virt-manager)를 선택합니다.

virt-manager 기본 창이 표시됩니다.

그림 15.1. virt-manager시작



*또는 다음 명령에서 시연한 대로 **ssh**를 사용하여 **virt-manager** 를 원격으로 시작할 수 있습니다.*

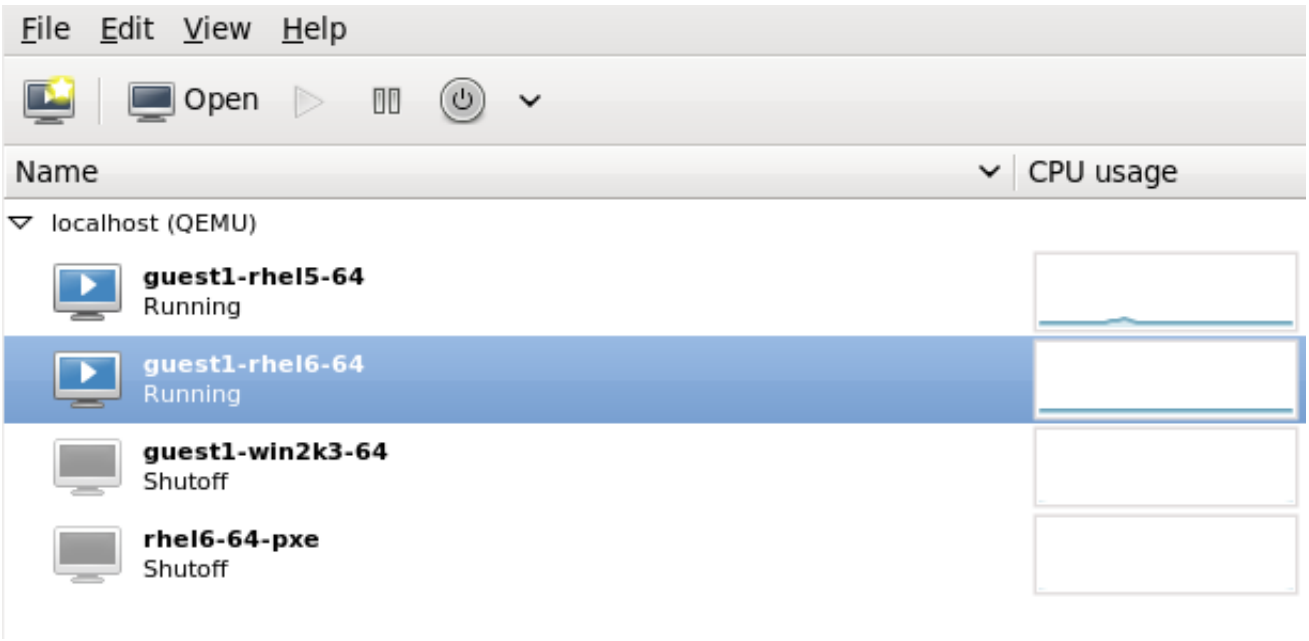
```
ssh -X host's address
[remotehost]# virt-manager
```

***ssh** 를 사용하여 가상 머신 및 호스트를 관리하는 방법은 5.1절. "SSH를 사용한 원격 관리" 에서 자세히 설명합니다.*

15.2. 가상 머신 관리자의 메인 창

이 기본 창에는 게스트가 사용하는 실행 중인 모든 게스트 및 리소스가 표시됩니다. 게스트 이름을 두 번 클릭하여 게스트를 선택합니다.

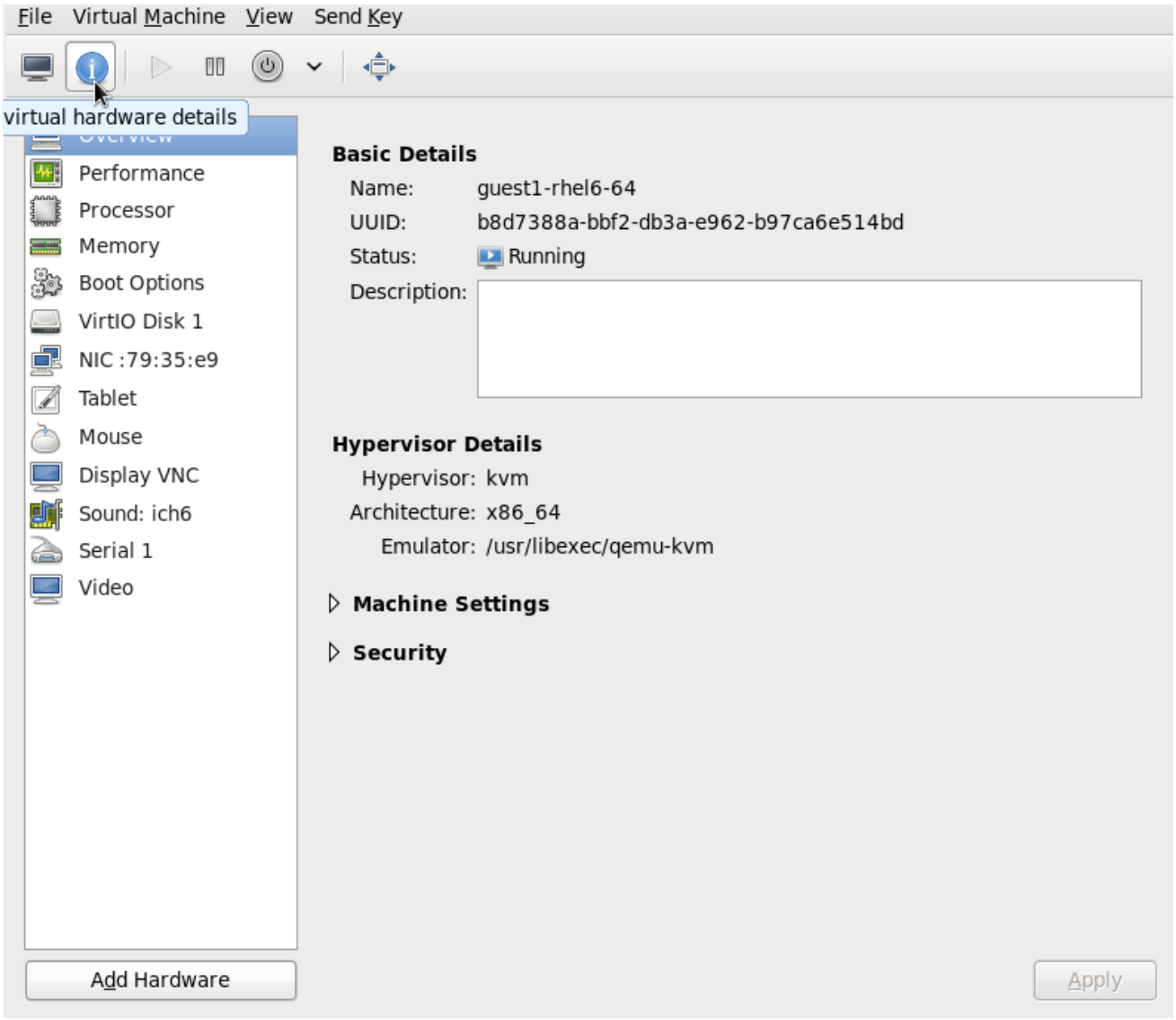
그림 15.2. 가상 머신 관리자 메인 창



15.3. 가상 하드웨어 세부 정보 창

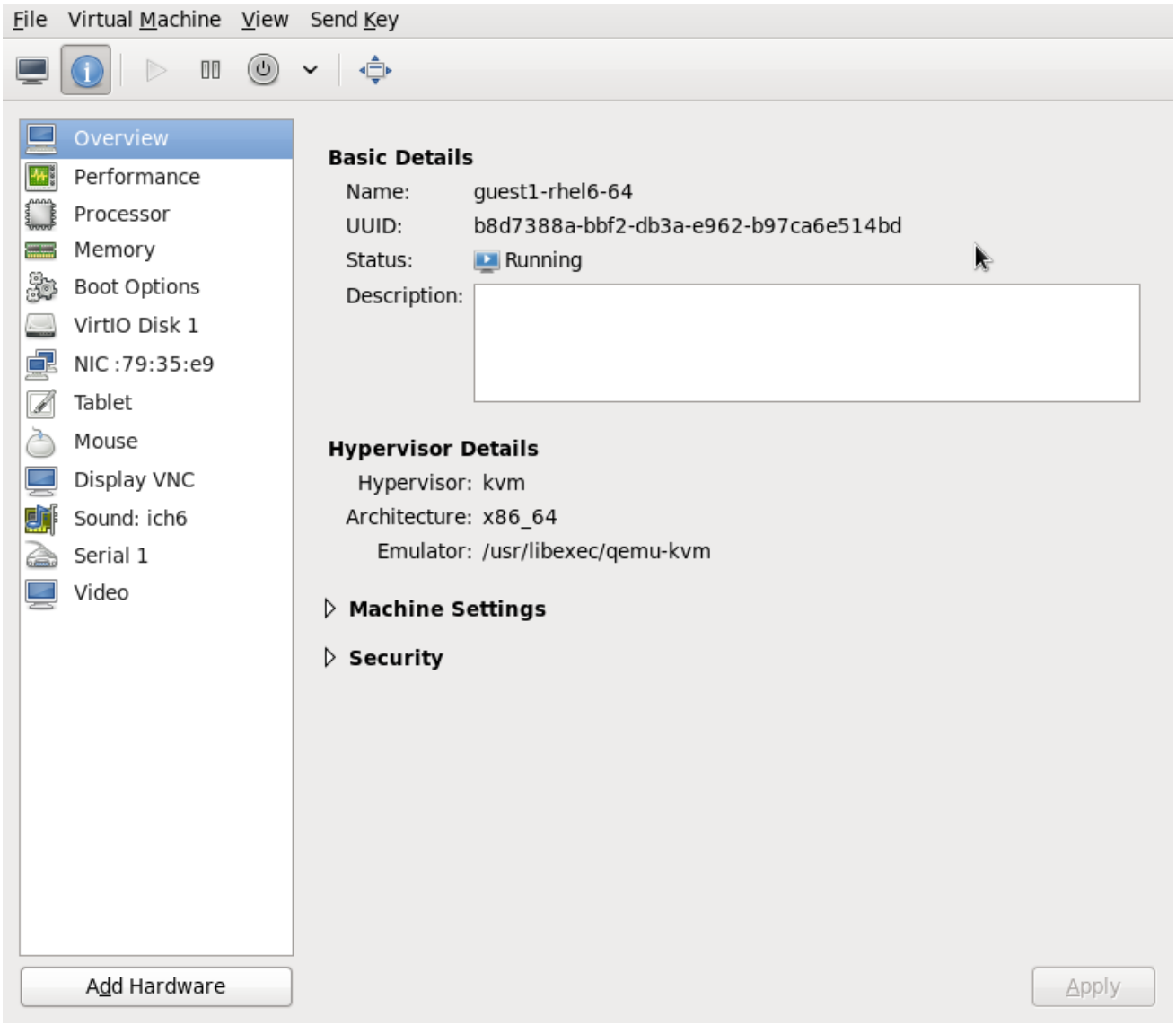
가상 하드웨어 세부 정보 창에는 게스트에 대해 구성된 가상 하드웨어에 대한 정보가 표시됩니다. 이 창에서 가상 하드웨어 리소스를 추가, 제거 및 수정할 수 있습니다. 가상 하드웨어 세부 정보 창에 액세스하려면 도구 모음의 아이콘을 클릭합니다.

그림 15.3. 가상 하드웨어 세부 정보 아이콘

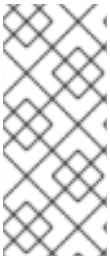


아이콘을 클릭하면 가상 하드웨어 세부 정보 창이 표시됩니다.

그림 15.4. 가상 하드웨어 세부 정보 창



15.3.1. 게스트 가상 머신에 USB 장치 연결



참고

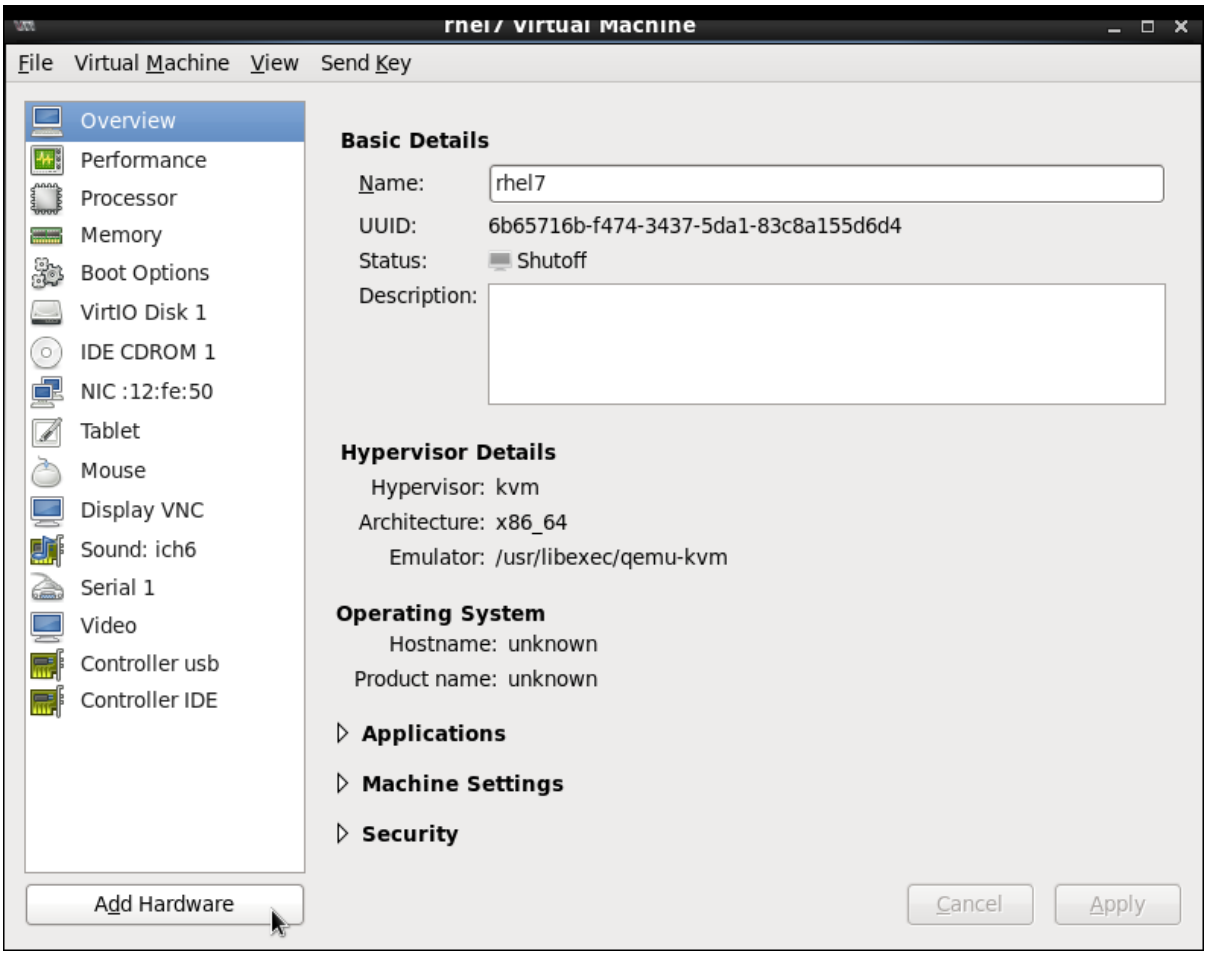
*게스트 가상 머신에 **USB** 장치를 연결하려면 먼저 호스트 물리적 시스템에 연결하고 장치가 작동하는지 확인해야 합니다. 게스트가 실행 중인 경우 계속하기 전에 종료해야 합니다.*

절차 15.1. Virt-Manager를 사용하여 USB 장치 연결

1. *게스트 가상 머신의 가상 머신 세부 정보 화면을 엽니다.*

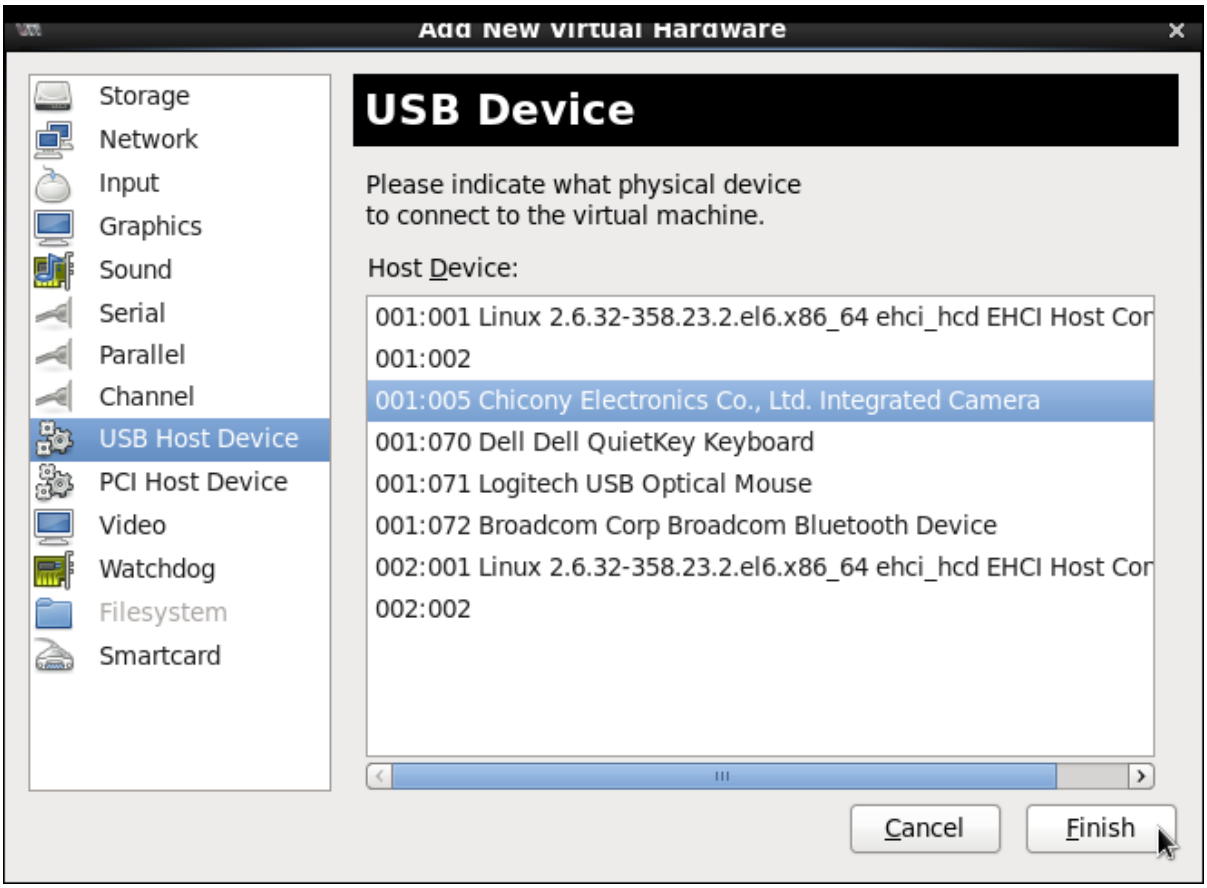
2. *하드웨어 추가를클릭합니다.*

그림 15.5. 하드웨어 버튼 추가



3. ***Add New Virtual Hardware** (새 가상 하드웨어 추가) 팝업에서 **USB Host Device** 를 선택하고 목록에서 연결할 장치를 선택하고 **Finish** 를 클릭합니다.*

그림 15.6. USB 장치 추가

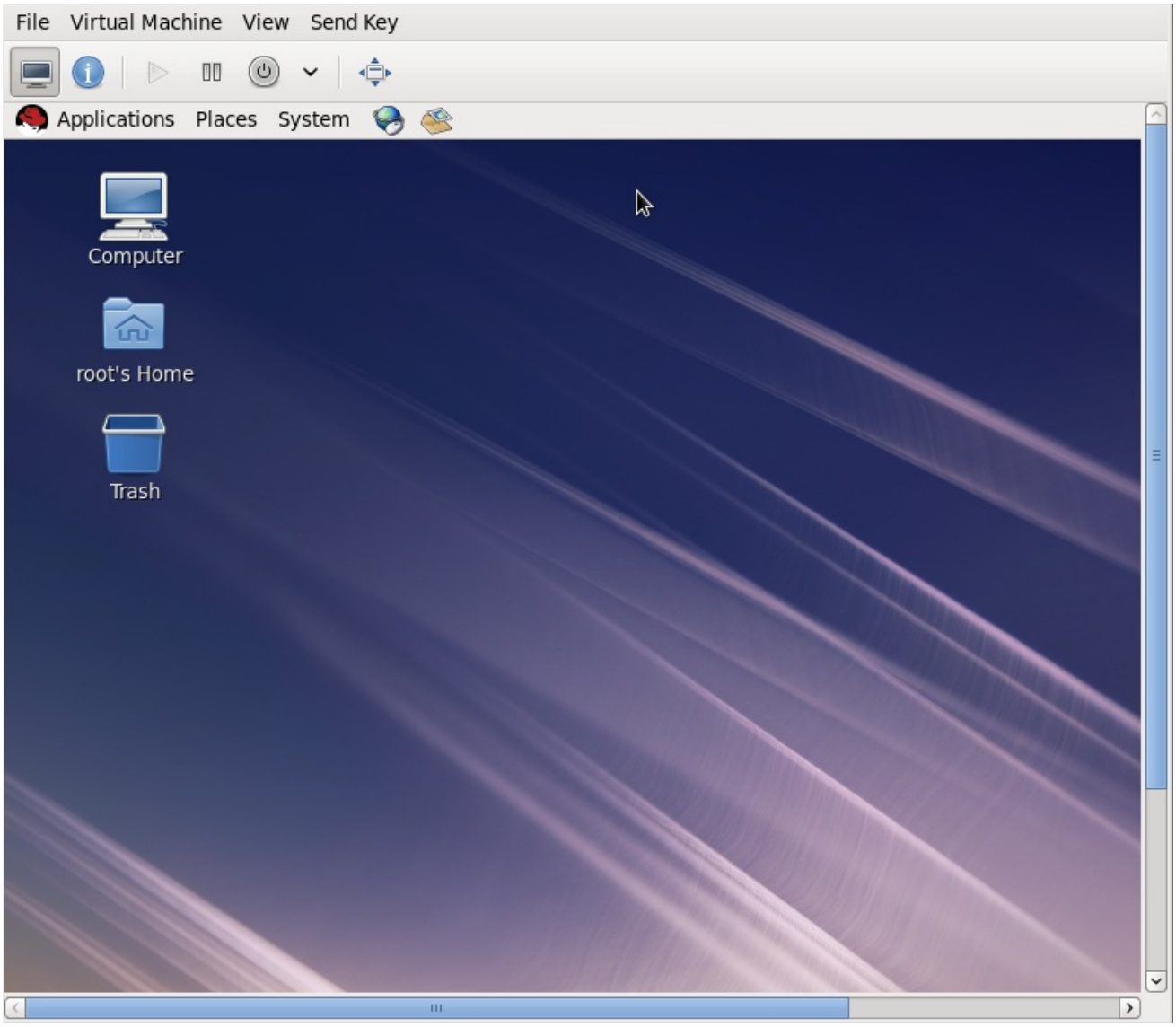


4. *게스트 가상 머신에서 **USB** 장치를 사용하려면 게스트 가상 머신을 시작합니다.*

15.4. 가상 머신 그래픽 콘솔

*이 창에는 게스트의 그래픽 콘솔이 표시됩니다. 게스트는 여러 다른 프로토콜을 사용하여 그래픽 프레임버를 내보낼 수 있습니다. **virt-manager** 는 **VNC** 및 **SPICE** 를 지원합니다. 가상 머신이 인증이 필요하도록 설정된 경우 가상 머신 그래픽 콘솔에서 디스플레이가 표시되기 전에 암호를 입력하라는 메시지를 표시합니다.*

그림 15.7. 그래픽 콘솔 창



참고

VNC는 많은 보안 전문가에 의해 안전하지 않은 것으로 간주되지만 Red Hat Enterprise Linux에서 가상화를 위해 VNC를 안전하게 사용할 수 있도록 몇 가지 변경 사항이 변경되었습니다. 게스트 시스템은 로컬 호스트의 루프백 주소(127.0.0.1)만 수신 대기합니다. 이렇게 하면 호스트의 쉘 권한이 있는 사용자만 VNC를 통해 virt-manager 및 가상 머신에 액세스할 수 있습니다. virt-manager가 다른 공용 네트워크 인터페이스와 대체 방법을 청취하도록 구성되어 있지만 구성하는 것은 권장되지 않습니다.

원격 관리는 트래픽을 암호화하는 SSH를 통한 터널링으로 수행할 수 있습니다. 보안상의 이유로 SSH를 통해 터널링하지 않고 원격으로 액세스하도록 VNC를 구성할 수 있습니다. 게스트를 원격으로 관리하려면 5장. 원격 게스트 관리 의 지침을 따릅니다. TLS는 게스트 및 호스트 시스템 관리를 위한 엔터프라이즈 수준 보안을 제공할 수 있습니다.

로컬 데스크탑에서 키 조합(예: Ctrl+Alt+F1)을 가로채어 게스트 시스템으로 전송되는 것을 방지할 수 있습니다. 전송 키 메뉴 옵션을 사용하여 이러한 시퀀스를 보낼 수 있습니다. 게스트 머신 창에서 전송 키

메뉴를 클릭하고 보낼 키 시퀀스를 선택합니다. 또한 이 메뉴에서는 화면 출력을 캡처할 수도 있습니다.

***SPICE**는 **Red Hat Enterprise Linux**에서 사용할 수 있는 **VNC**의 대안입니다.*

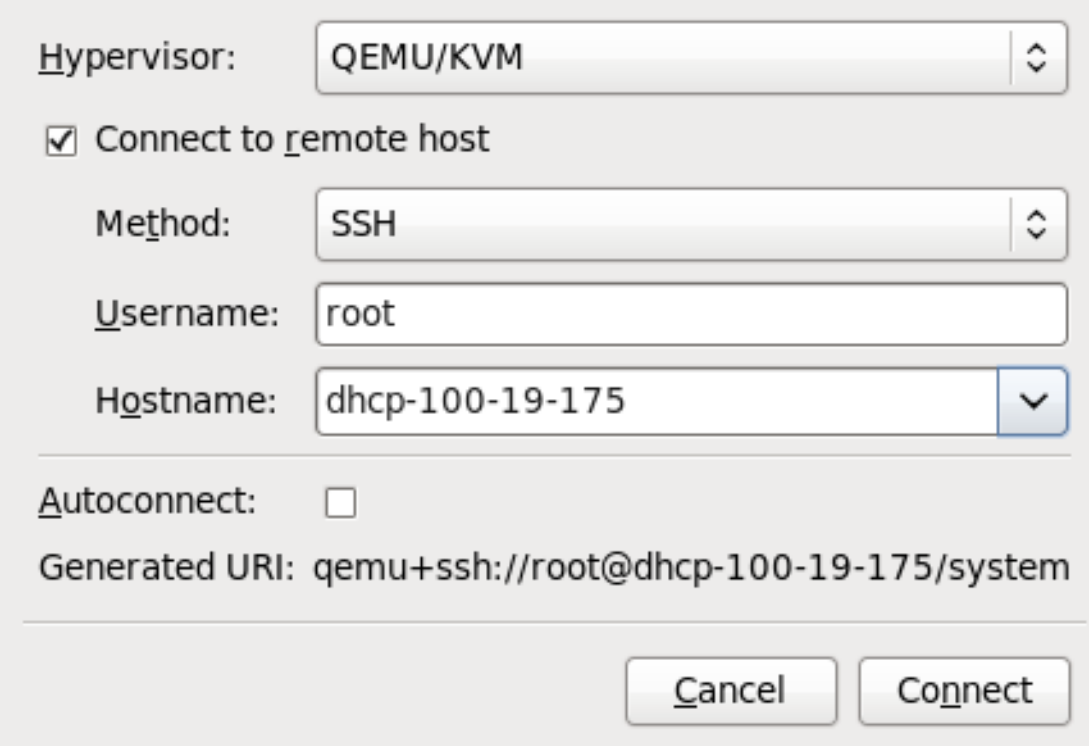
15.5. 원격 연결 추가

*다음 절차에서는 **virt-manager** 를 사용하여 원격 시스템에 대한 연결을 설정하는 방법을 설명합니다.*

1. *새 연결을 만들려면 파일 메뉴를 열고 연결 추가... 메뉴 항목을 선택합니다.*

2. *연결 추가 마법사가 나타납니다. 하이퍼바이저를 선택합니다. **Red Hat Enterprise Linux 6** 시스템의 경우 **QEMU/KVM** 을 선택합니다. 로컬 시스템에 대해 **Local**을 선택하거나 원격 연결 옵션 중 하나를 선택하고 연결을 클릭합니다. 이 예에서는 기본 설치에서 작동하는 **SSH**를 통한 원격 터널을 사용합니다. 원격 연결 구성에 대한 자세한 내용은 다음을 참조하십시오. **5장. 원격 게스트 관리***

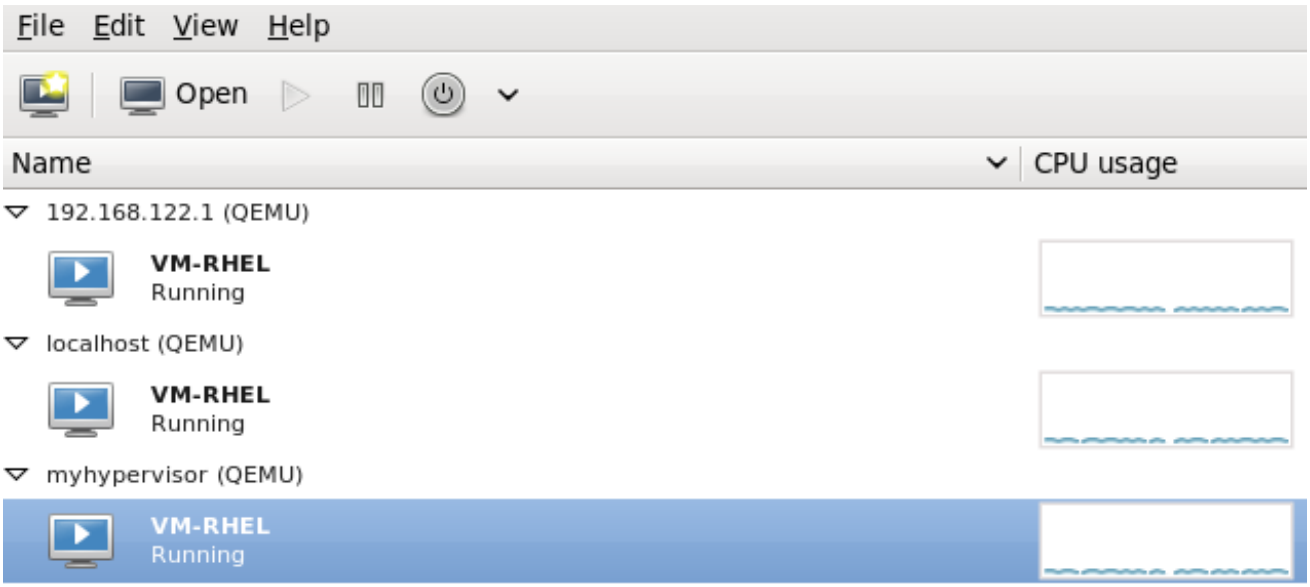
   *그림 **15.8.** 연결 추가*



3. *메시지가 표시되면 선택한 호스트의 루트 암호를 입력합니다.*

*원격 호스트가 이제 연결되었으며 기본 **virt-manager** 창에 표시됩니다.*

그림 15.9. 기본 virt-manager 창에 있는 원격 호스트



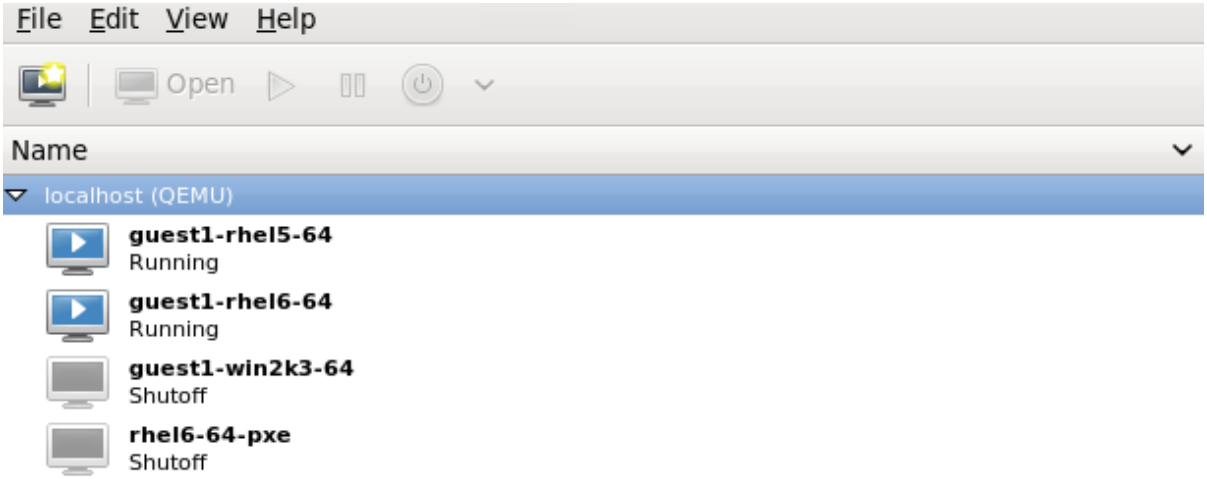
15.6. 게스트 세부 정보 표시

가상 머신 모니터를 사용하여 시스템의 모든 가상 머신에 대한 활동 정보를 볼 수 있습니다.

가상 시스템의 세부 정보를 보려면 다음을 수행합니다.

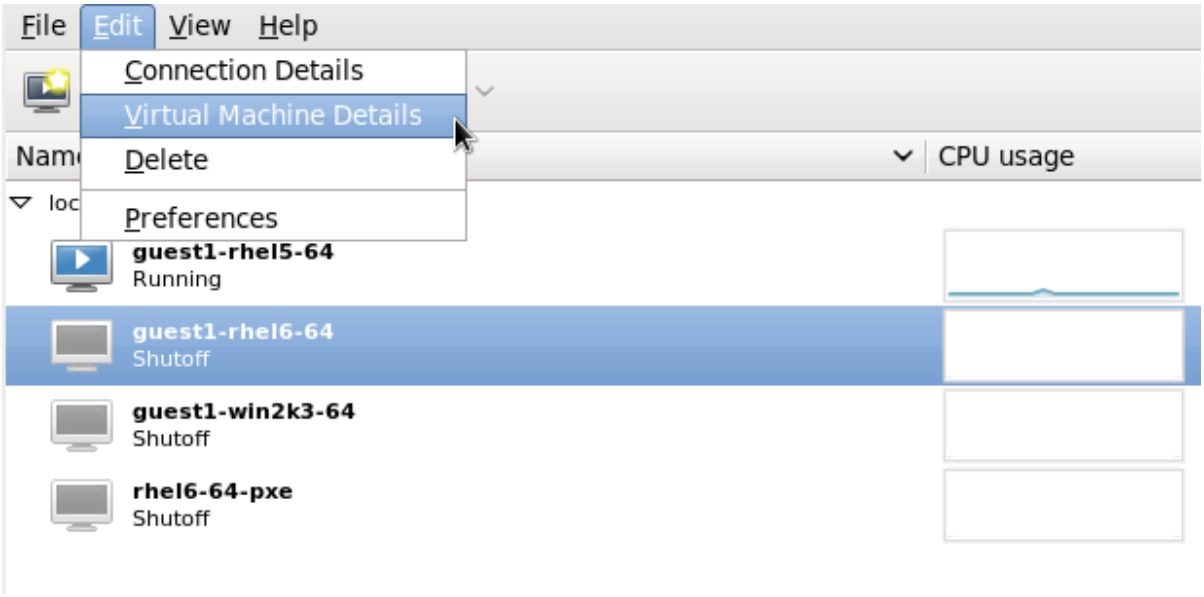
1. ***Virtual Machine Manager** 메인 창에서 보려는 가상 머신을 강조 표시합니다.*

   그림 15.10. 표시할 가상 머신 선택

2. *가상 머신 관리자 편집 메뉴에서 가상 머신 세부 정보를 선택합니다.*

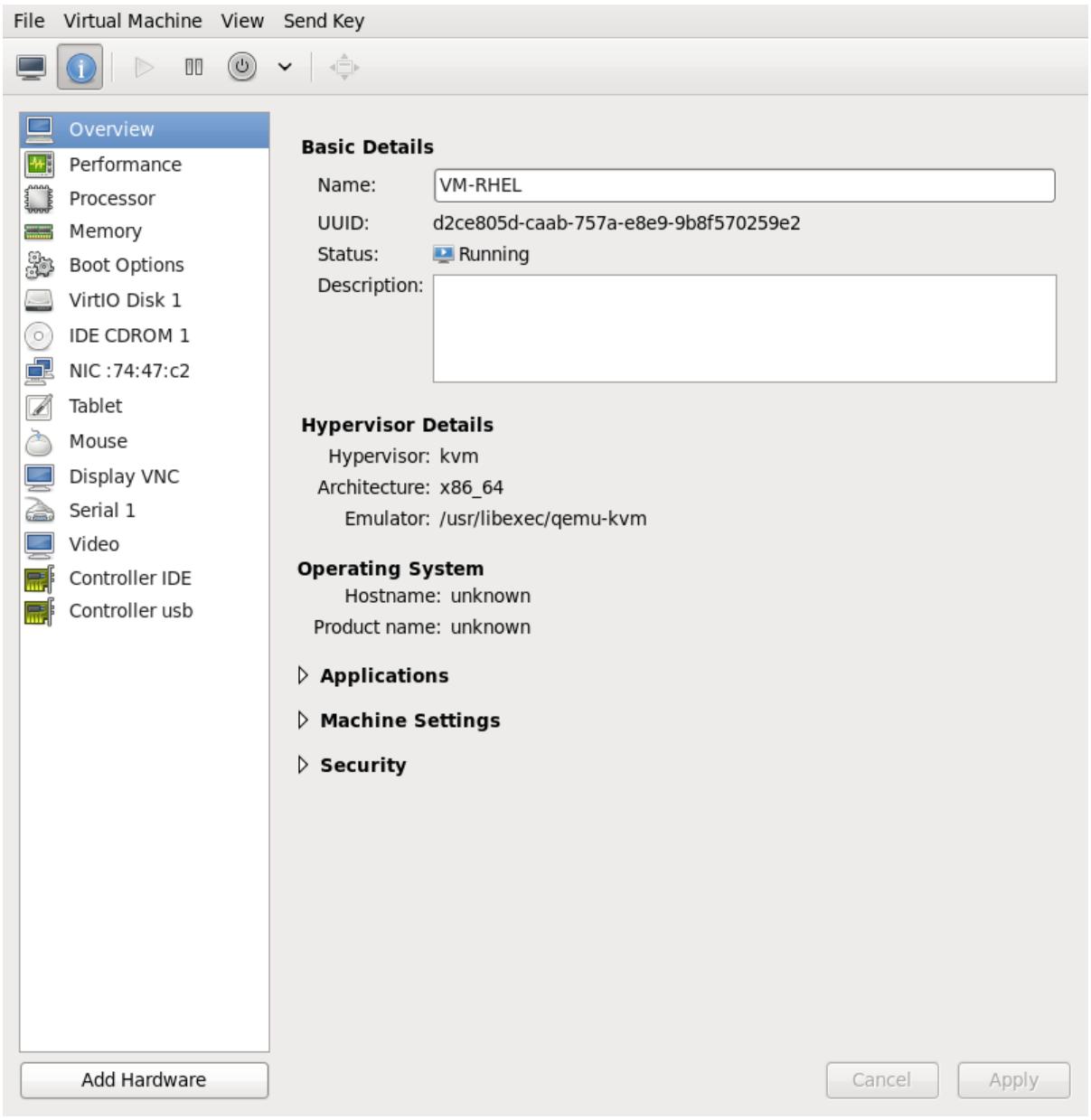
그림 15.11. 가상 머신 세부 정보 표시



가상 머신 세부 정보 창이 열리면 콘솔이 표시될 수 있습니다. 이러한 상황이 발생하면 보기를 클릭한 다음 세부 정보를 선택합니다. 기본적으로 개요 창이 먼저 열립니다. 이 창으로 돌아가려면 왼쪽 탐색 창에서 개요 를 선택합니다.

개요 보기에는 게스트에 대한 구성 세부 정보가 요약되어 있습니다.

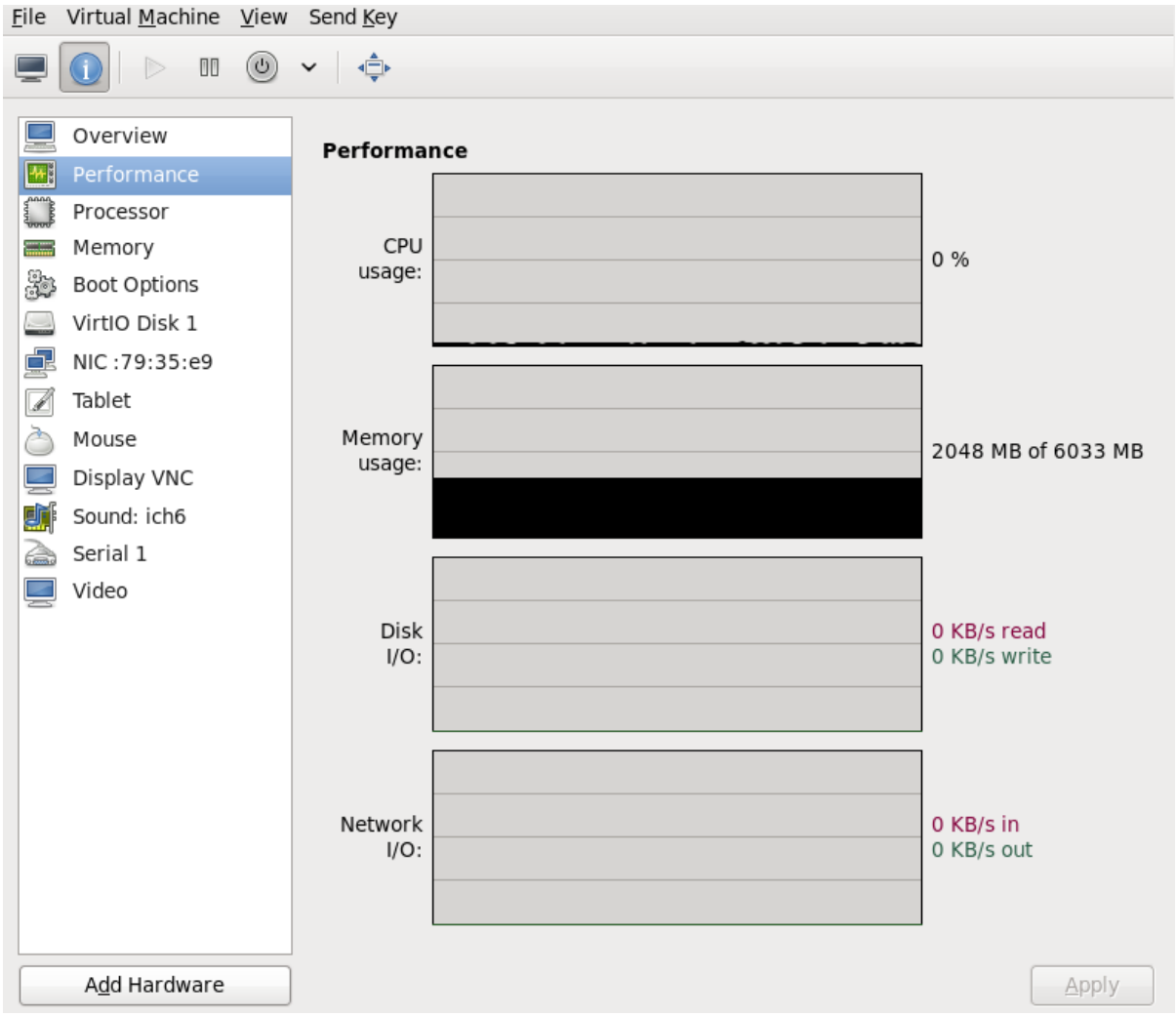
그림 15.12. 게스트 세부 정보 표시 개요



3. *왼쪽의 탐색 창에서 **Performance** 를 선택합니다.*

   *성능 보기에는 **CPU** 및 메모리 사용량을 포함하여 게스트 성능에 대한 요약이 표시됩니다.*

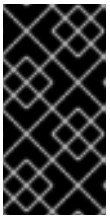
그림 15.13. 게스트 성능 세부 정보 표시



4.

*왼쪽의 탐색 창에서 **Processor** 를 선택합니다. 프로세서 뷰를 사용하면 현재 프로세서 할당을 보고 변경할 수 있습니다.*

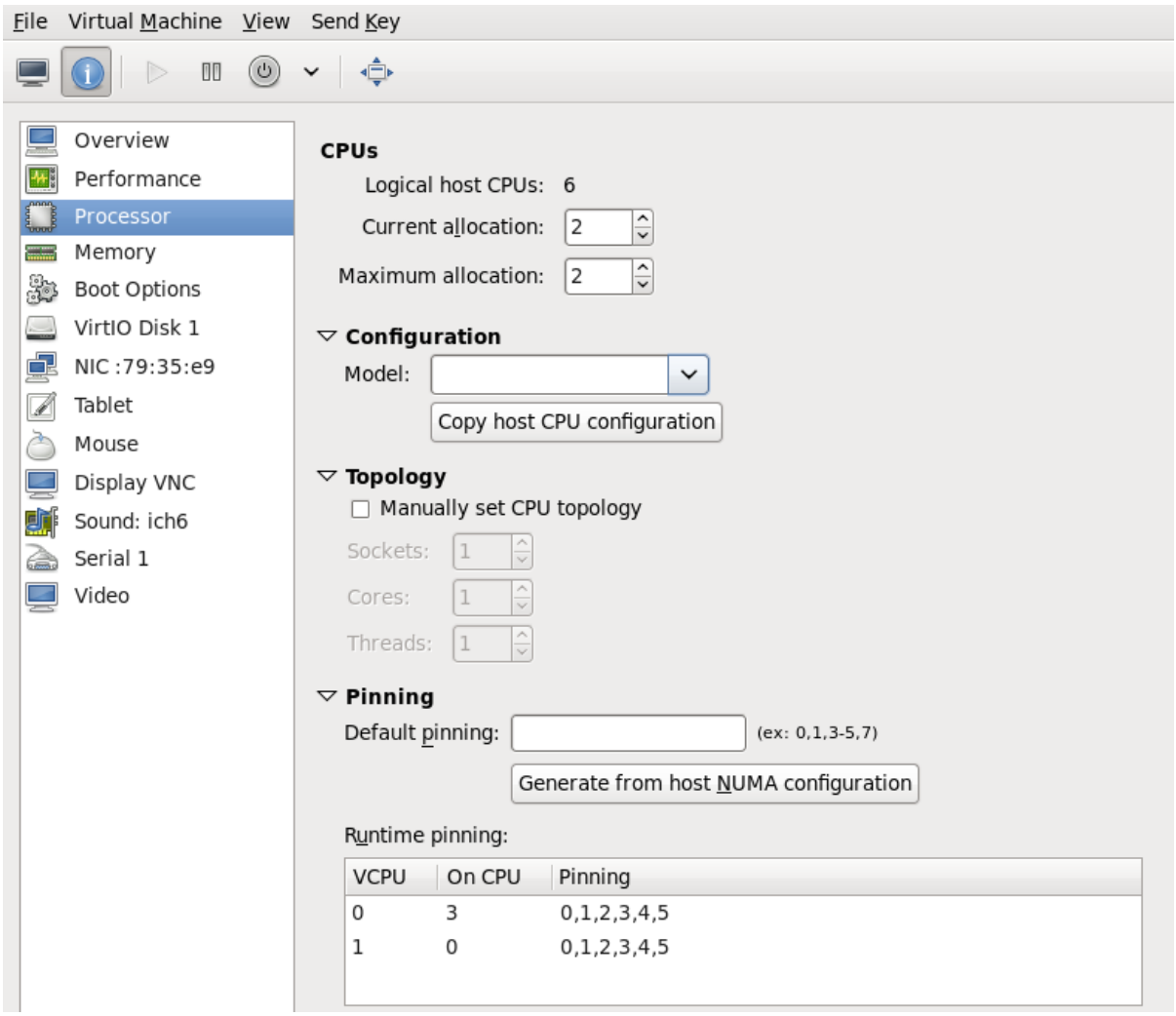
*또한 가상 머신이 실행되는 동안 **vCPU(가상 CPU)** 수를 변경할 수 있습니다. 이 수는 핫플러그 및 핫 플러그 연결 이라고 합니다.*



중요

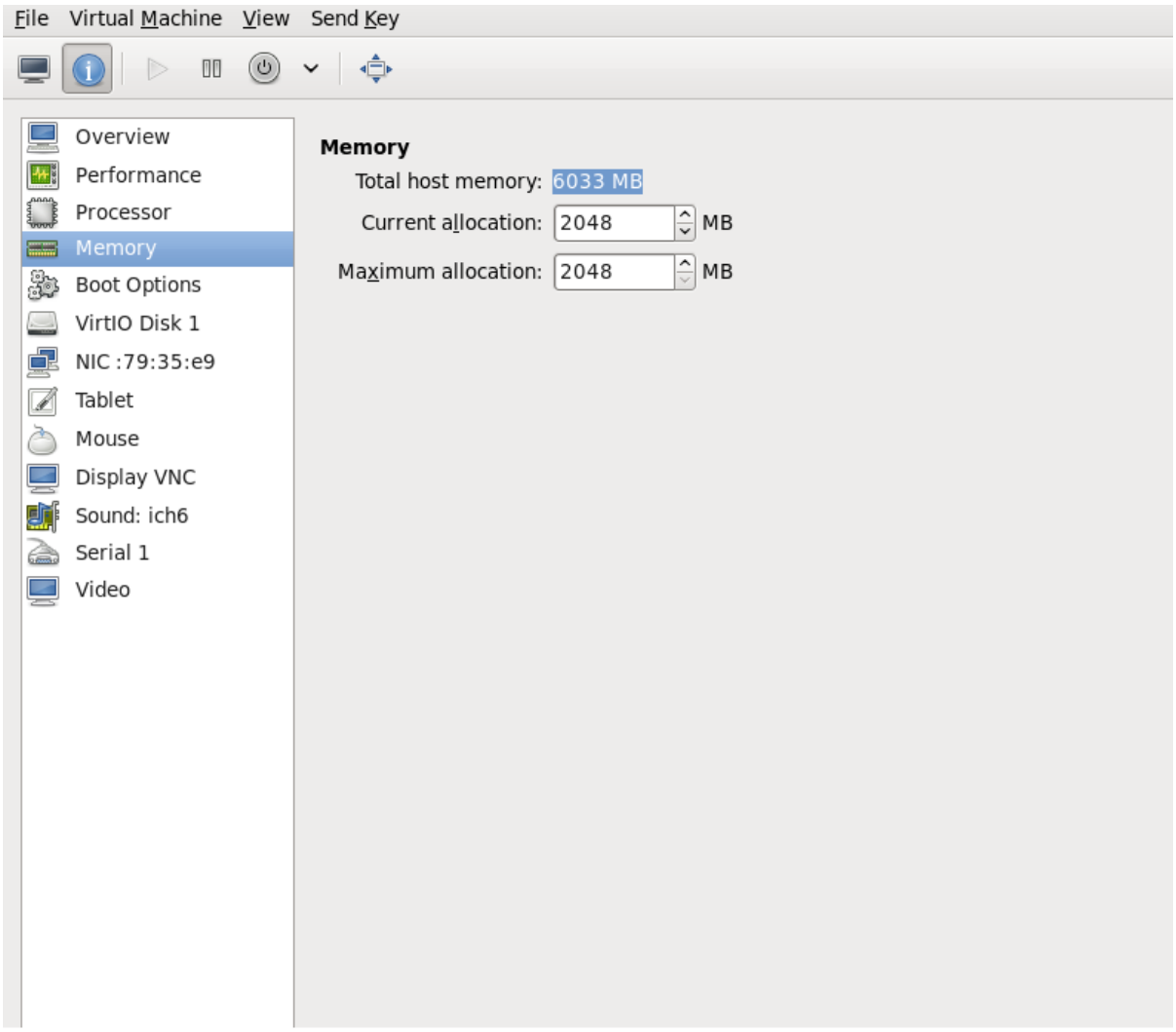
핫 플러그 해제 기능은 기술 프리뷰로만 사용할 수 있습니다. 따라서 이 기능은 지원되지 않으며 높은 가치의 배포에는 사용하지 않는 것이 좋습니다.

그림 15.14. 프로세서 할당 패널



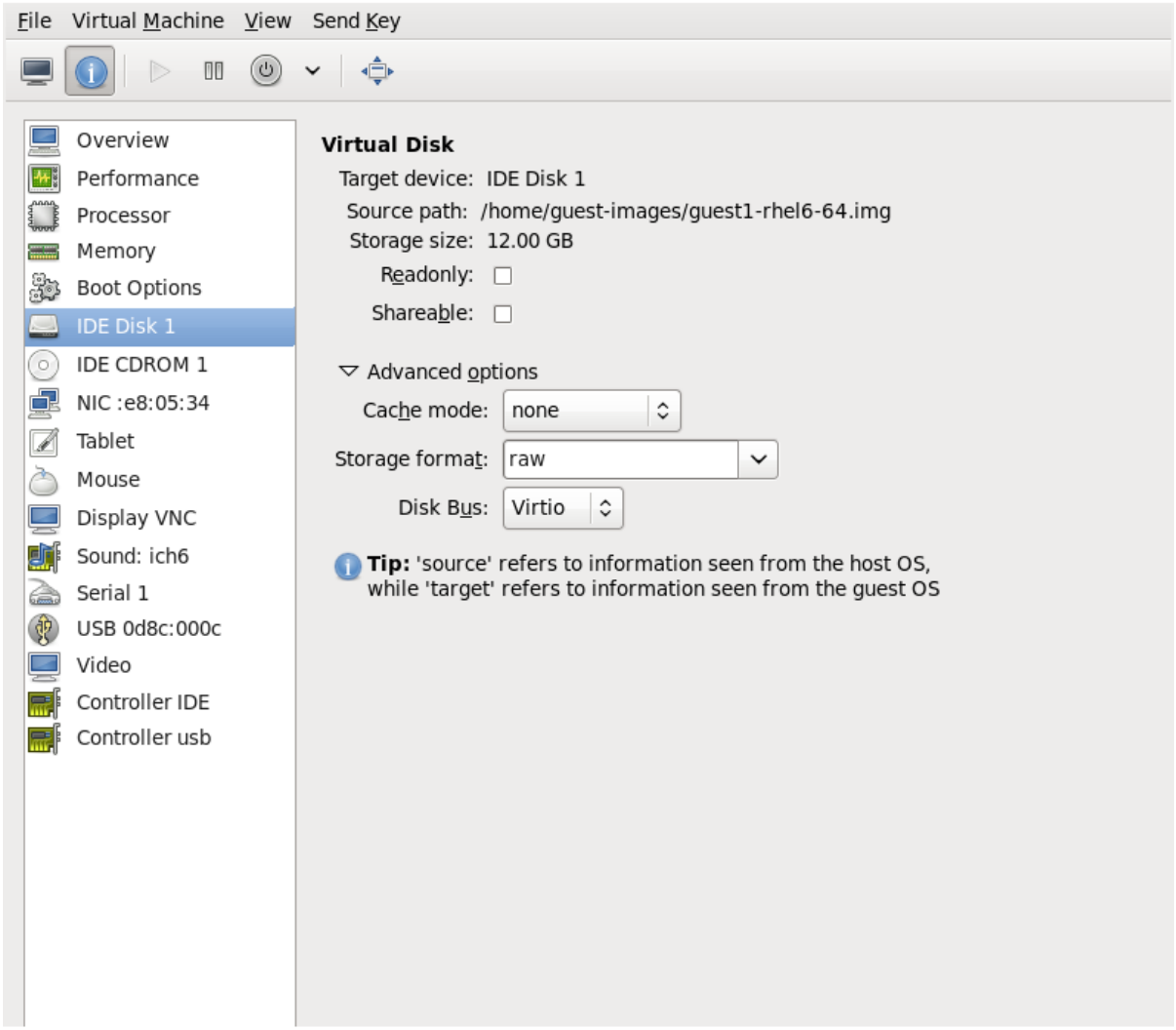
5. *왼쪽의 탐색 창에서 메모리 를 선택합니다. 메모리 보기를 사용하면 현재 메모리 할당을 보거나 변경할 수 있습니다.*

그림 15.15. 메모리 할당 표시



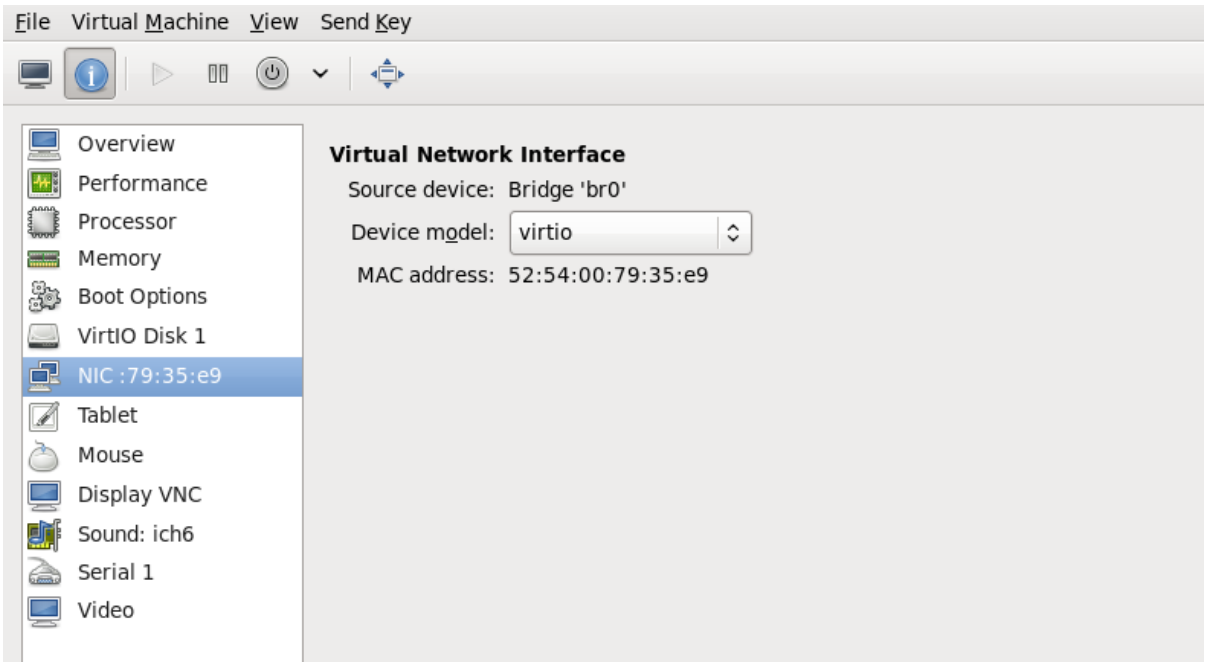
6. *가상 머신에 연결된 각 가상 디스크가 탐색 창에 표시됩니다. 가상 디스크를 클릭하여 수정하거나 제거합니다.*

그림 15.16. 디스크 구성 표시



7. *가상 머신에 연결된 각 가상 네트워크 인터페이스가 탐색 창에 표시됩니다. 가상 네트워크 인터페이스를 클릭하여 수정하거나 제거합니다.*

그림 15.17. 네트워크 구성 표시



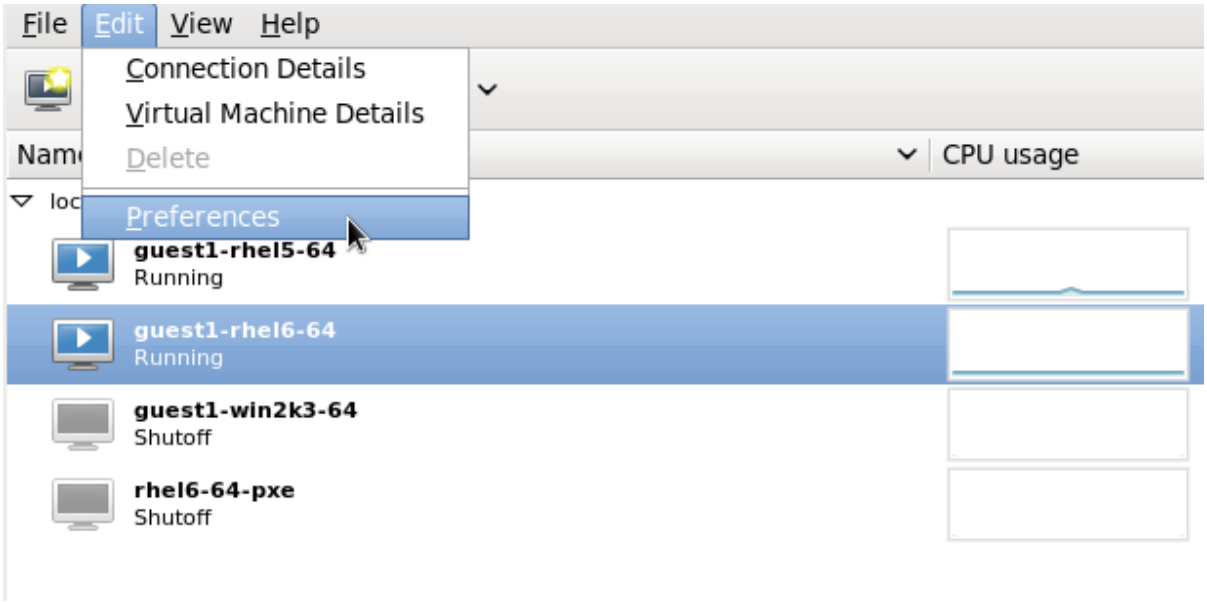
15.7. 성능 모니터링

***virt-manager** 의 기본 설정 창을 사용하여 성능 모니터링 기본 설정을 수정할 수 있습니다.*

성능 모니터링을 구성하려면 다음을 수행합니다.

1. *편집 메뉴에서 환경 설정을 선택합니다.*

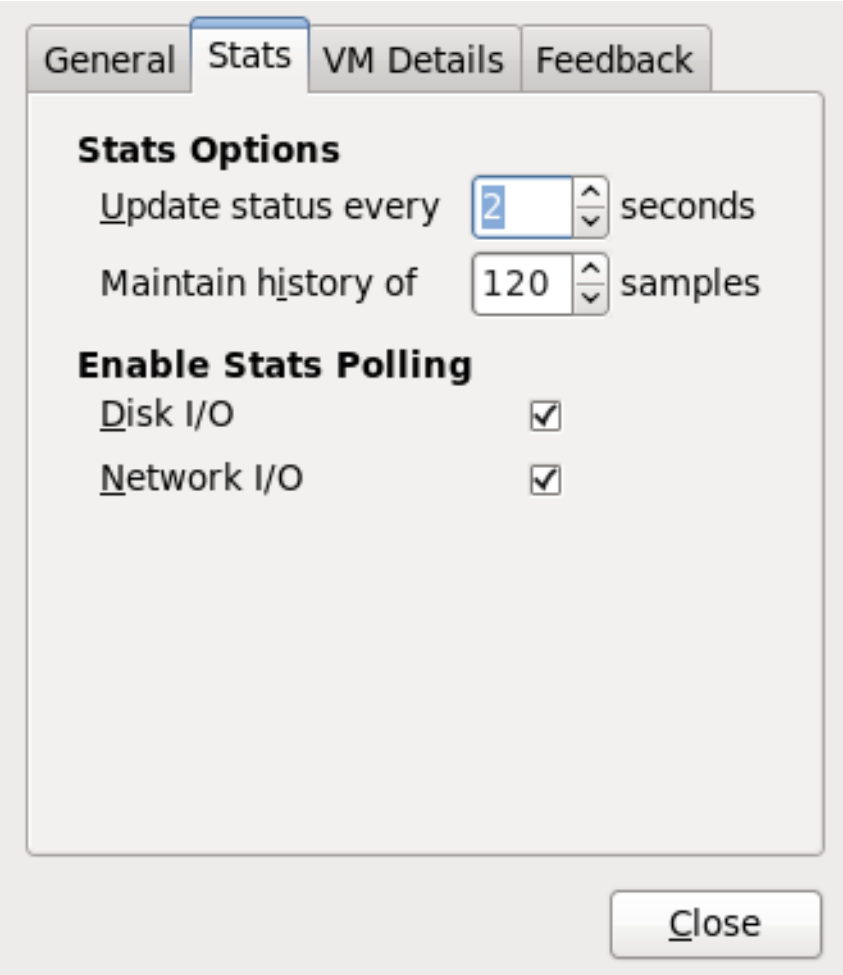
그림 15.18. 게스트 기본 설정 수정



Preference 창이 나타납니다.

2. *Stats 탭에서는 시간(초) 또는 통계 폴링 옵션을 지정합니다.*

그림 15.19. 성능 모니터링 구성

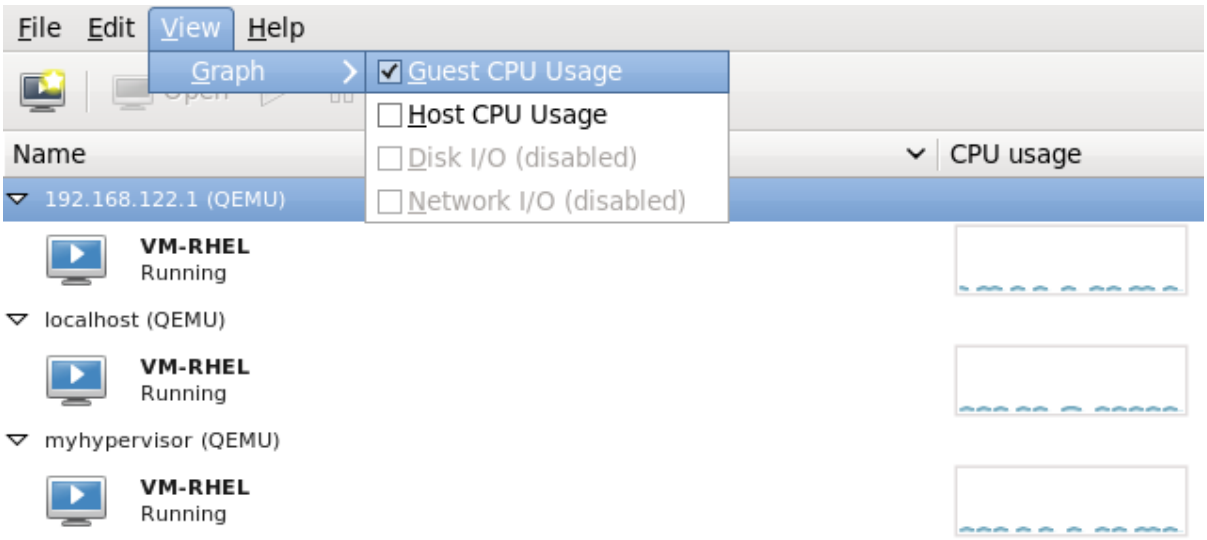


15.8. 게스트의 CPU 사용량 표시

시스템의 모든 게스트의 CPU 사용량을 보려면 다음을 수행합니다.

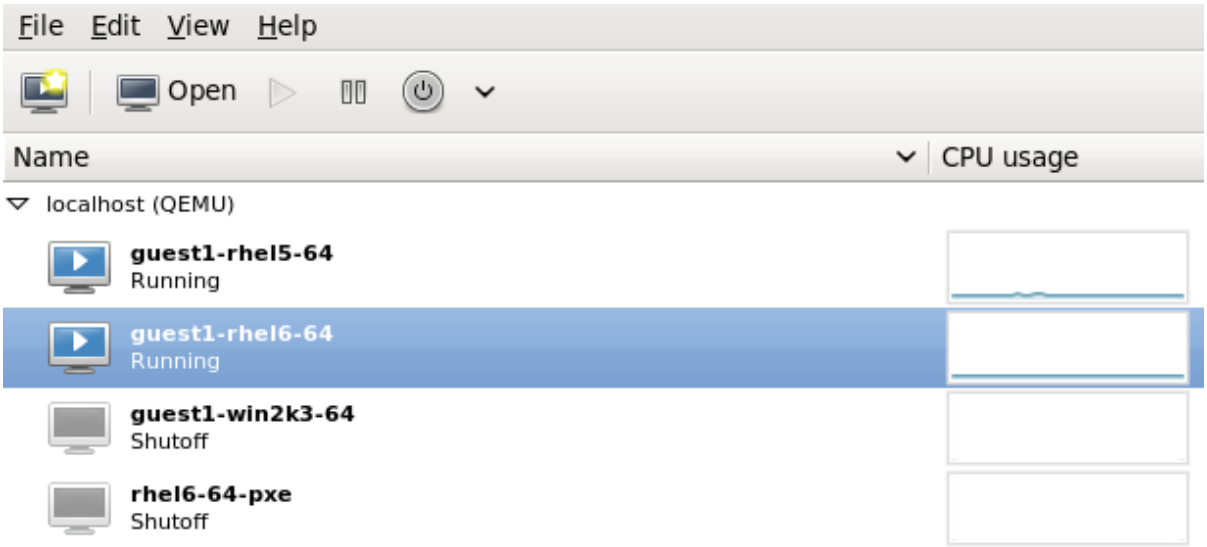
1. *보기 메뉴에서 그래프 를 선택하고 Guest CPU Usage (게스트 CPU 사용량) 확인란을 선택합니다.*

그림 15.20. 게스트 CPU 사용량 통계 그래프 활성화



2. *가상 머신 관리자는 시스템의 모든 가상 머신에 대한 CPU 사용량 그래프를 보여줍니다.*

그림 15.21. 게스트 CPU 사용량 그래프

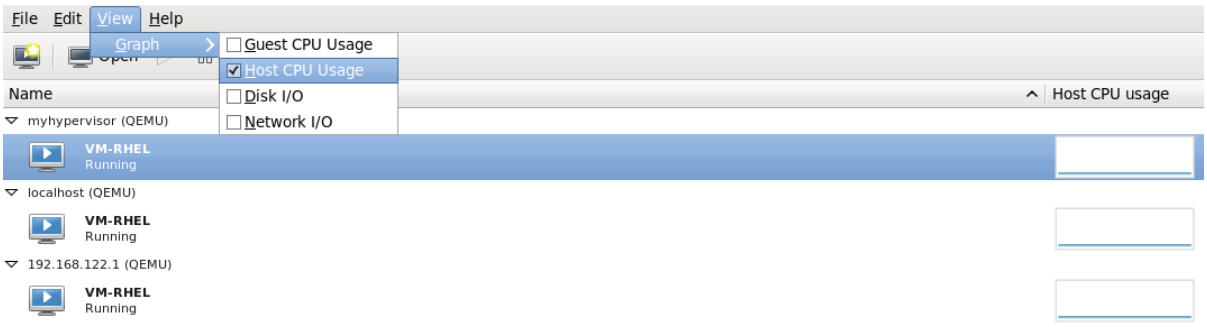


15.9. 호스트의 CPU 사용량 표시

시스템의 모든 호스트에 대한 CPU 사용량을 보려면 다음을 수행합니다.

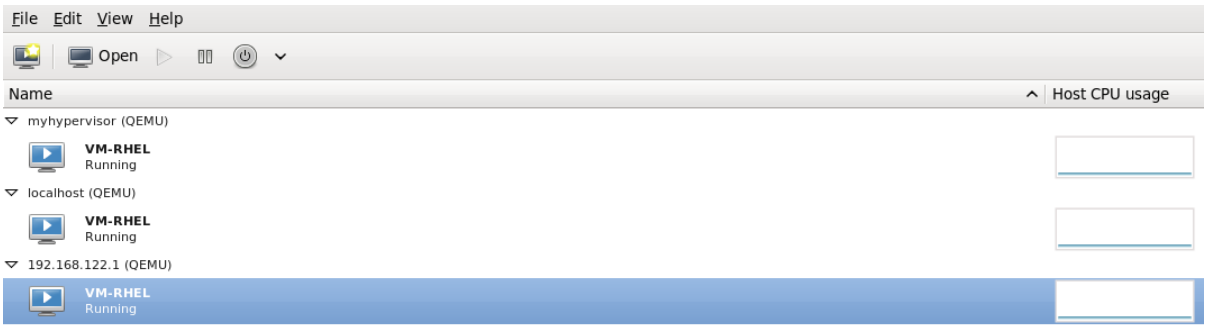
1. ***View** (보기) 메뉴에서 **Graph, Host CPU Usage** (호스트 CPU 사용량) 확인란을 선택합니다.*

그림 15.22. 호스트 CPU 사용량 통계 그래프 활성화



2. *가상 머신 관리자는 시스템의 호스트 CPU 사용량 그래프를 보여줍니다.*

그림 15.23. 호스트 CPU 사용량 그래프



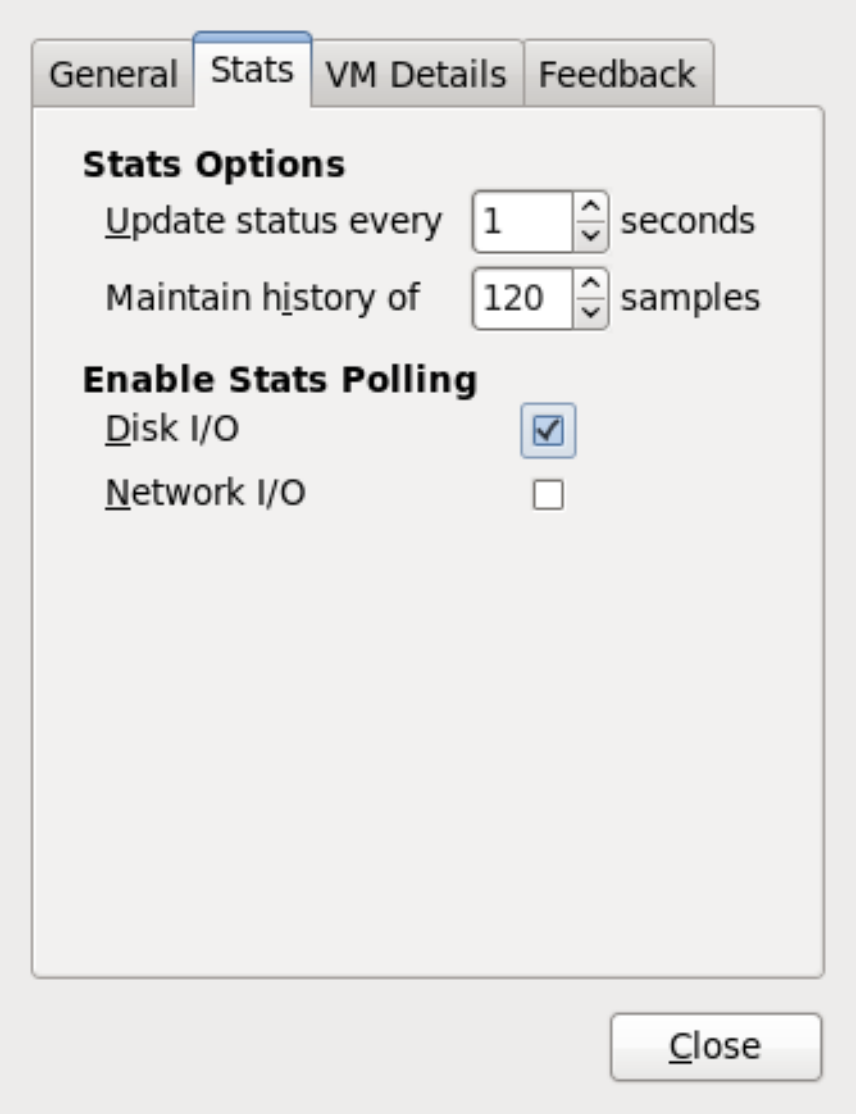
15.10. 디스크 I/O 표시

시스템의 모든 가상 머신의 디스크 I/O를 보려면 다음을 수행합니다.

1. *디스크 I/O 통계 컬렉션이 활성화되어 있는지 확인합니다. 이렇게 하려면 편집 메뉴에서 환경 설정을 선택하고 통계탭을 클릭합니다.*

2. *디스크 I/O 확인란을 선택합니다.*

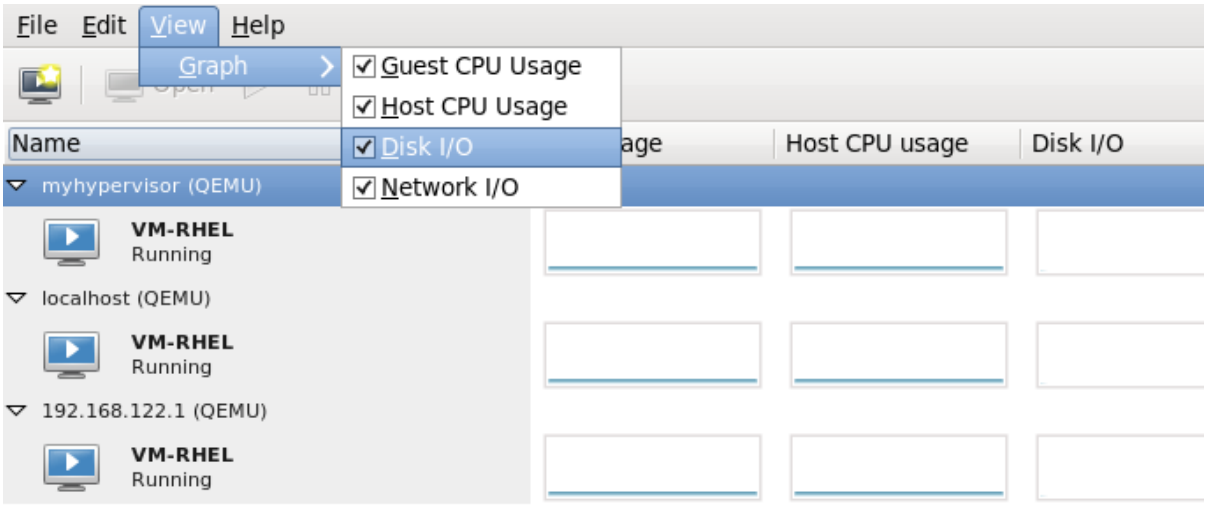
그림 15.24. 디스크 I/O 활성화



3.

*디스크 I.O 디스플레이를 활성화하려면 **View** 메뉴에서 **Graph, then Disk I/O** 확인란을 선택합니다.*

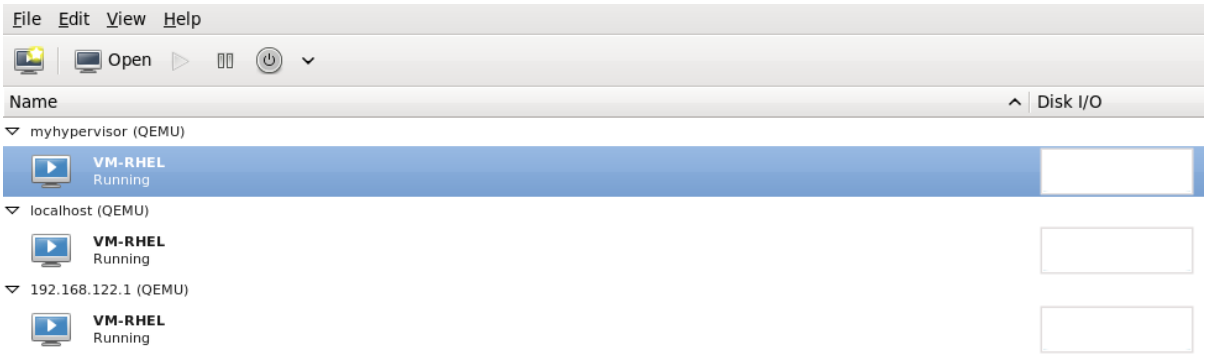
그림 15.25. 디스크 I/O 선택



4.

가상 머신 관리자는 시스템의 모든 가상 머신에 대한 디스크 I/O 그래프를 보여줍니다.

그림 15.26. 디스크 I/O 표시



15.11. 네트워크 I/O 표시

시스템에 있는 모든 가상 머신의 네트워크 I/O를 보려면 다음을 수행합니다.

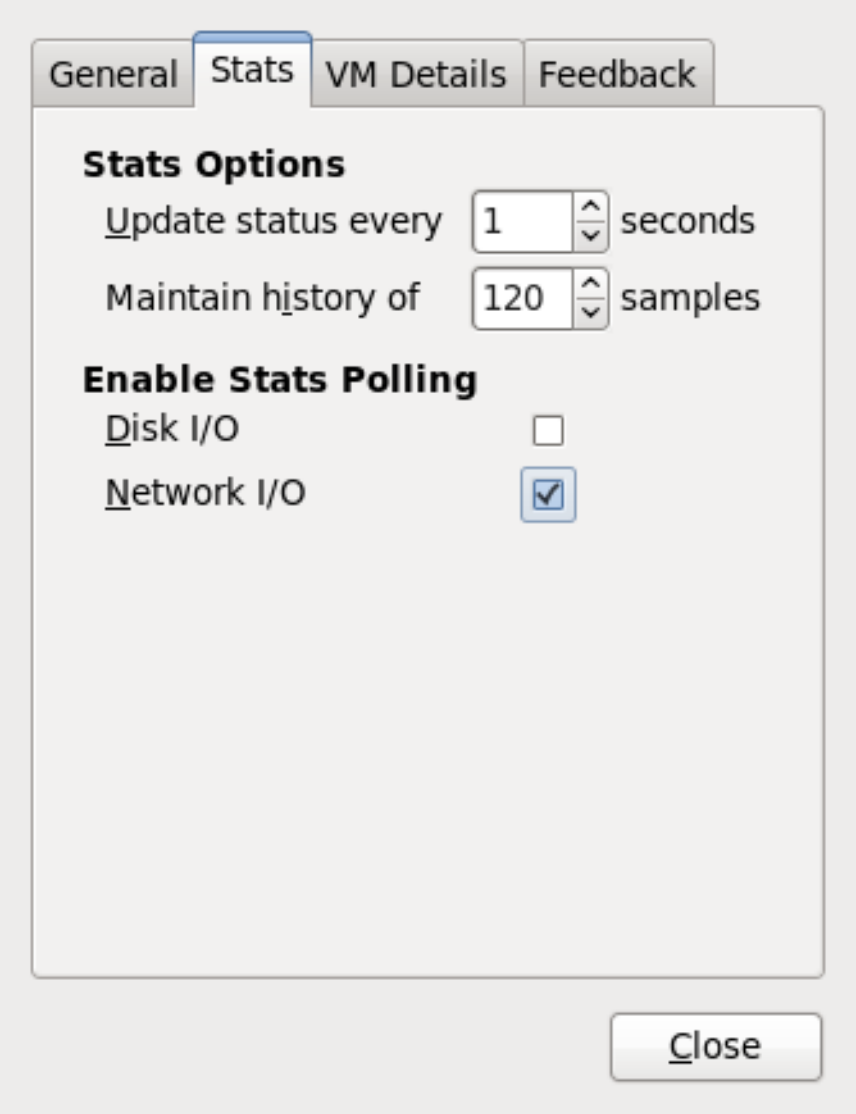
1.

네트워크 I/O 통계 컬렉션이 활성화되어 있는지 확인합니다. 이렇게 하려면 편집 메뉴에서 환경 설정을 선택하고 통계탭을 클릭합니다.

2.

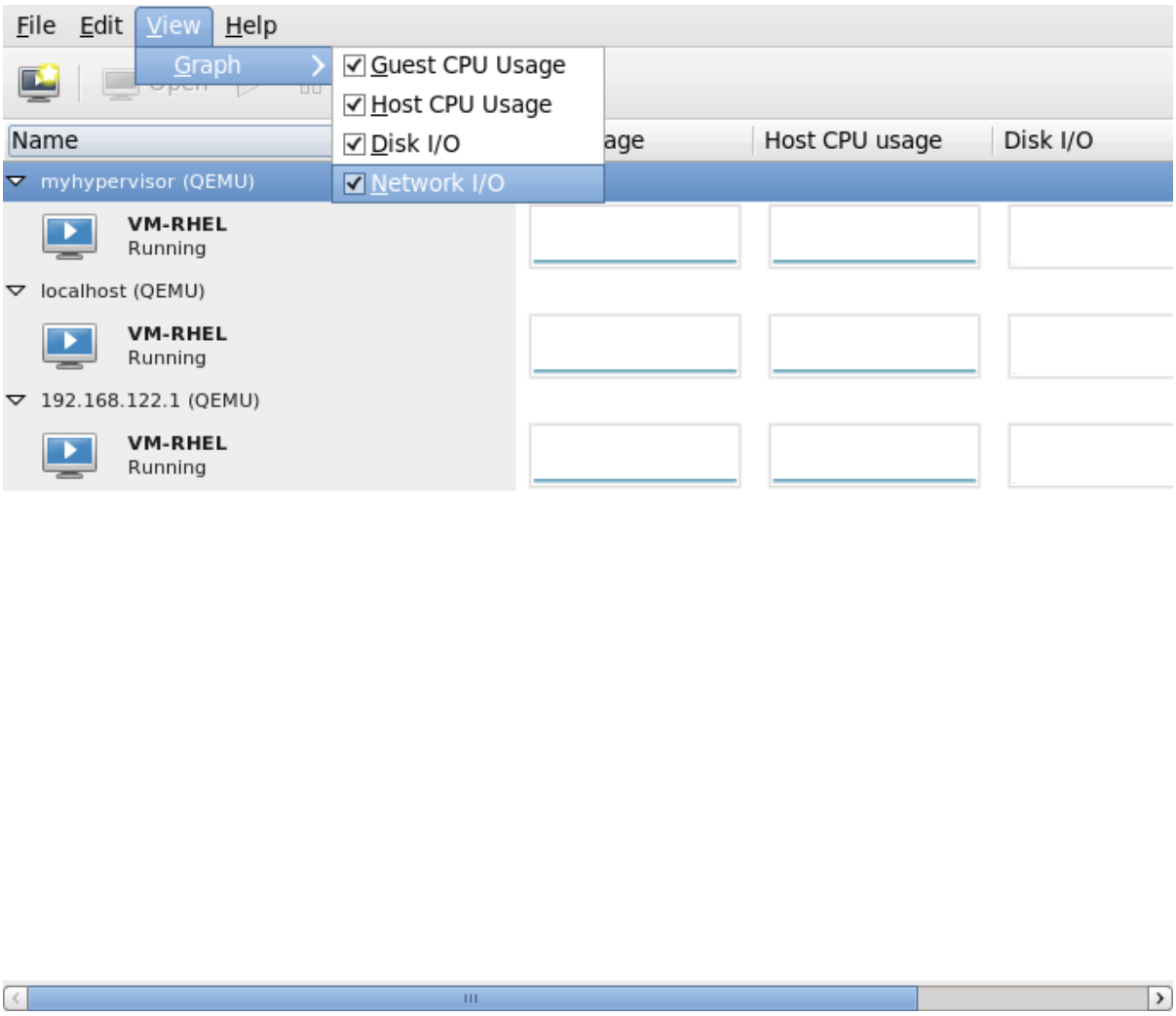
***Network I/O** 확인란을 선택합니다.*

그림 15.27. 네트워크 I/O 활성화



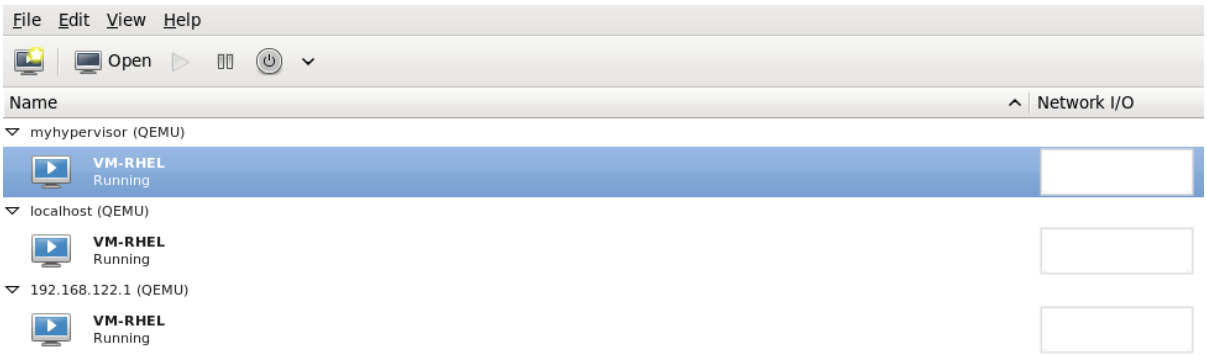
3. *네트워크 I/O 통계를 표시하려면 보기 메뉴에서 그래프, 네트워크 I/O 확인란을 선택합니다.*

그림 15.28. 네트워크 I/O 선택



4. *가상 머신 관리자는 시스템의 모든 가상 머신에 대한 네트워크 I/O 그래프를 보여줍니다.*

그림 15.29. 네트워크 I/O 표시



16장. 오프라인 툴을 사용한 게스트 가상 머신 디스크 액세스

16.1. 소개

Red Hat Enterprise Linux 6에는 호스트 물리적 머신 디스크 또는 기타 디스크 이미지에 액세스하고 편집 및 생성할 수 있는 툴이 포함되어 있습니다. 이러한 툴에는 다음과 같은 몇 가지 용도가 있습니다.

- 호스트 물리적 머신 디스크에 있는 파일 보기 또는 다운로드.
- 호스트 물리적 머신 디스크에 파일을 편집하거나 업로드합니다.
- 호스트 물리적 시스템 설정 읽기 또는 쓰기.
- **Windows** 호스트 물리적 시스템에서 **Windows** 레지스트리 읽기 또는 쓰기.
- 파일, 디렉토리, 파일 시스템, 파티션, 논리 볼륨 및 기타 옵션이 포함된 새 디스크 이미지 준비
- 부팅하지 못한 호스트 물리적 시스템 또는 부팅 구성을 변경해야 하는 호스트 물리적 시스템을 복구 및 복구합니다.
- 호스트 물리적 시스템의 디스크 사용량 모니터링.
- 조직의 보안 표준과 같이 호스트 물리적 시스템의 규정 준수 감사.
- 템플릿을 복제 및 수정하여 호스트 물리적 시스템 배포.
- **CD** 및 **DVD ISO** 및 플로피 디스크 이미지를 읽습니다.

주의

이러한 툴을 사용하여 쓰기 모드에서는 실행 중인 가상 시스템에 연결된 호스트 물리적 머신 또는 디스크 이미지에 쓰는 것이 아니라, 이러한 디스크 이미지를 쓰기 모드로 여는 것은 아닙니다. 이렇게 하면 게스트 가상 머신이 손상됩니다. 도구가 이 작업을 수행하지 못하도록 시도하지만 모든 사례를 파악하지는 않습니다. 게스트 가상 머신이 실행 중일 수 있는 일시 중단이 있는 경우 도구를 사용하지 않거나 적어도 읽기 전용 모드로 툴을 사용하는 것이 좋습니다.

참고

***Red Hat Enterprise Linux 6**의 일부 가상화 명령을 사용하면 원격 **libvirt** 연결을 지정할 수 있습니다. 예를 들어 다음과 같습니다.*

```
virt-df -c qemu://remote/system -d Guest
```

*그러나 **Red Hat Enterprise Linux 6**의 **libguestfs**는 원격 게스트에 액세스할 수 없으며 원격 **URL**을 사용하는 명령은 예상대로 작동하지 않습니다. 이는 다음 **Red Hat Enterprise Linux 6** 명령에 영향을 미칩니다.*

- ***guestfish***
- *게스트**mount***
- ***virt-alignment-scan***
- ***virt-cat***
- ***virt-copy-in***
- ***virt-copy-out***

- ***virt-df***
- ***virt-edit***
- ***virt-filesystems***
- ***virt-inspector***
- ***virt-inspector2***
- ***virt-list-filesystems***
- ***virt-list-partitions***
- ***virt-ls***
- ***virt-rescue***
- ***virt-sysprep***
- ***virt-tar***
- ***virt-tar-in***
- ***virt-tar-out***
- ***virt-win-reg***

16.2. 용어

이 섹션에서는 이 장의 용어에 대해 설명합니다.

- **libguestfs(Guest 파일 시스템 라이브러리) -** 디스크 이미지를 열고 파일을 읽고 쓰는 기본 기능을 제공하는 기본 **C** 라이브러리입니다. 이 **API**에 **C** 프로그램을 직접 작성할 수 있지만 수준이 매우 낮습니다.

- **Guestfish(Guest** 파일 시스템 대화형 쉘**)** 는 명령줄 또는 쉘 스크립트에서 사용할 수 있는 대화형 쉘입니다. **libguestfs API**의 모든 기능을 노출합니다.

- 다양한 **virt** 툴은 **libguestfs** 위에 구축되며, 이를 통해 명령줄에서 특정 단일 작업을 수행할 수 있습니다. 툴에는 **virt-df,virt-rescue,virt-resize** 및 **virt-edit** 가 포함됩니다.

- **Hivex** 및 **Augeas** 는 각각 **Windows** 레지스트리 및 **Linux** 구성 파일을 편집하기 위한 라이브러리입니다. **libguestfs**와 분리되어 있지만 대부분의 **libguestfs** 값은 이러한 도구의 조합에서 비롯됩니다.

- **guestmount** 는 **libguestfs**와 **FUSE** 간의 인터페이스입니다. 호스트 물리적 시스템의 디스크 이미지에서 파일 시스템을 마운트하는 데 주로 사용됩니다. 이 기능은 필요하지 않지만 유용할 수 있습니다.

16.3. 설치

libguestfs, Guestfs, libguestfs 툴, 게스트 마운트 및 **Windows** 게스트 가상 머신에 대한 지원을 설치하려면 **Red Hat Enterprise Linux V2WIN** 채널을 구독하고 **Red Hat** 웹 사이트로 이동하여 다음 명령을 실행합니다.

```
# yum install libguestfs guestfish libguestfs-tools libguestfs-winsupport
```

언어 바인딩을 포함한 모든 **libguestfs** 관련 패키지를 설치하려면 다음 명령을 실행합니다.

```
# yum install '*guestf*'
```

16.4. GUESTMET 쉘

Guest fish는 명령줄 또는 쉘 스크립트에서 게스트 가상 머신 파일 시스템에 액세스하는 데 사용할 수

있는 대화형 쉘입니다. **libguestfs API**의 모든 기능은 쉘에서 사용할 수 있습니다.

가상 머신 디스크 이미지 보기 또는 편집을 시작하려면 다음 명령을 실행하여 원하는 디스크 이미지의 경로를 대체합니다.

```
guestfish --ro -a /path/to/disk/image
```

--ro 는 디스크 이미지가 읽기 전용으로 열려 있음을 의미합니다. 이 모드는 항상 안전하지만 쓰기 액세스 허용은 허용하지 않습니다. 게스트 가상 머신이 실행되고 있지 않거나 디스크 이미지가 라이브 게스트 가상 머신에 연결되지 않은 경우에만 이 옵션을 생략합니다. **libguestfs** 를 사용하여 라이브 게스트 가상 머신을 편집할 수 없으며 시도하면 되돌릴 수 없는 디스크 손상이 발생합니다.

/path/to/disk/image 는 디스크의 경로입니다. 이 파일은 호스트 물리적 머신 논리 볼륨(예: **/dev/VG/LV**), 호스트 물리적 머신 장치(**/dev/cdrom**) 또는 **SAN LUN**(**/dev/sdf3**)일 수 있습니다.

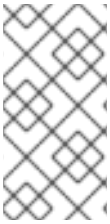
참고

libguestfs 및 **fish**는 루트 권한이 필요하지 않습니다. 액세스 중인 디스크 이미지에 **root**가 읽기 또는 쓰기 또는 둘 다 필요한 경우에만 **root**로 실행하면 됩니다.

interactivelylyian을 시작하면 다음과 같은 프롬프트가 표시됩니다.

```
guestfish --ro -a /path/to/disk/image

Welcome to guestfish, the libguestfs filesystem interactive shell for editing virtual machine
filesystems.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

><fs>
```

프롬프트에서 **run** 을 입력하여 라이브러리를 시작하고 디스크 이미지를 연결합니다. 처음 수행할 때 최대 **30**초가 걸릴 수 있습니다. 후속 시작은 훨씬 더 빨리 완료됩니다.



참고

libguestfs는 **KVM**(사용 가능한 경우)과 같은 하드웨어 가상화 가속을 사용하여 이 프로세스를 가속화합니다.

run 명령을 입력한 후 다음 섹션에서 설명하는 다른 명령을 사용할 수 있습니다.

16.4.1. fish로 파일 시스템 보기

이 섹션에서는**fish**로 파일을 보는 방법에 대해 설명합니다.

16.4.1.1. 수동 목록 및 보기

list-filesystems 명령은 **libguestfs**에서 찾은 파일 시스템을 나열합니다. 이 출력은 **Red Hat Enterprise Linux 4** 디스크 이미지를 보여줍니다.

```
><fs> run
><fs> list-filesystems
/dev/vda1: ext3
/dev/VolGroup00/LogVol00: ext3
/dev/VolGroup00/LogVol01: swap
```

이 출력에는 **Windows** 디스크 이미지가 표시됩니다.

```
><fs> run
><fs> list-filesystems
/dev/vda1: ntfs
/dev/vda2: ntfs
```

기타 유용한 명령은 목록 장치,**list-partitions,lvs,pvs,vfs-type** 및 파일 입니다. 다음 출력에 표시된 대로 **help** 명령을 입력하여 모든 명령에 대한 자세한 정보와 도움말 을 얻을 수 있습니다.

```
><fs> help vfs-type
 NAME
    vfs-type - get the Linux VFS type corresponding to a mounted device

 SYNOPSIS
     vfs-type device

 DESCRIPTION
    This command gets the file system type corresponding to the file system on
    "device".
```

```
For most file systems, the result is the name of the Linux VFS module
which would be used to mount this file system if you mounted it without
specifying the file system type. For example a string such as "ext3" or
"ntfs".
```

*파일 시스템의 실제 내용을 보려면 먼저 마운트해야 합니다. 이 예에서는 이전 출력(**/dev/vda2**)에 표시된 **Windows** 파티션 중 하나를 사용합니다. 이 경우 **C:** 드라이브에 해당하는 것으로 알려져 있습니다.*

```
><fs> mount-ro /dev/vda2 /
><fs> ll /
total 1834753
 drwxrwxrwx  1 root root       4096 Nov  1 11:40 .
 drwxr-xr-x 21 root root       4096 Nov 16 21:45 ..
 lrwxrwxrwx  2 root root         60 Jul 14  2009 Documents and Settings
 drwxrwxrwx  1 root root       4096 Nov 15 18:00 Program Files
 drwxrwxrwx  1 root root       4096 Sep 19 10:34 Users
 drwxrwxrwx  1 root root      16384 Sep 19 10:34 Windows
```

***ls,ll,cat,more,download** 및 **tar-out** 과 같은**fish** 명령을 사용하여 파일과 디렉토리를 보고 다운로드 할 수 있습니다.*

참고

*이 쉘에는 현재 작업 중인 디렉터리의 개념이 없습니다. 일반 쉘과 달리 **cd** 명령을 사용하여 디렉터리를 변경할 수 없습니다. 모든 경로는 슬래시(/) 문자로 시작하여 정상으로 정규화된이어야 합니다. 경로를 완료하려면 **Tab** 키를 사용합니다.*

***fish** 쉘을 종료하려면 **exit** 를 입력하거나 **Ctrl+d** 를 입력합니다.*

16.4.1.2. inspection 사용

*파일 시스템을 나열 및 마운트하는 대신,**fish** 자체에서 이미지를 검사하고 게스트 가상 머신에 있는 것처럼 파일 시스템을 마운트하도록 할 수 있습니다. 이렇게 하려면 명령줄에 **-i** 옵션을 추가합니다.*

```
guestfish --ro -a /path/to/disk/image -i

Welcome to guestfish, the libguestfs filesystem interactive shell for
 editing virtual machine filesystems.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell
```

```
Operating system: Red Hat Enterprise Linux AS release 4 (Nahant Update 8)
/dev/VolGroup00/LogVol00 mounted on /
/dev/vda1 mounted on /boot

><fs> ll /
total 210
drwxr-xr-x. 24 root root  4096 Oct 28 09:09 .
drwxr-xr-x  21 root root  4096 Nov 17 15:10 ..
drwxr-xr-x.  2 root root  4096 Oct 27 22:37 bin
drwxr-xr-x.  4 root root  1024 Oct 27 21:52 boot
drwxr-xr-x.  4 root root  4096 Oct 27 21:21 dev
drwxr-xr-x. 86 root root 12288 Oct 28 09:09 etc
[etc]
```

inspection 및 **mount**를 수행하려면 **libguestfs** 백엔드를 시작해야 하므로 **-i** 옵션을 사용할 때 **run** 명령이 필요하지 않습니다. **i** 옵션은 일반적인 **Linux** 및 **Windows** 게스트 가상 머신에서 작동합니다.

16.4.1.3. 이름으로 게스트 가상 머신에 액세스

libvirt에 알려진 이름을 지정할 때 명령줄에서 게스트 가상 시스템에 액세스할 수 있습니다(즉, **virsh list --all**). **d** 옵션을 사용하여 **- i** 옵션을 사용하거나 사용하지 않고 이름으로 게스트 가상 머신에 액세스합니다.

```
guestfish --ro -d GuestName -i
```

16.4.2. fish를 사용하여 파일 수정

파일을 수정하려면 디렉터리를 생성하거나 게스트 가상 머신을 추가로 변경하려면 먼저 이 섹션의 시작 부분에 경고를 처리했습니다. 게스트 가상 머신을 종료해야 합니다. **6.7**을 사용하여 실행 중인 디스크를 편집하거나 변경하면 디스크 손상이 발생합니다. 이 섹션에서는 **/boot/grub/grub.conf** 파일을 편집하는 예를 제공합니다. 게스트 가상 머신이 종료되었는지 확인하는 경우 다음과 같은 명령을 통해 쓰기 액세스 권한을 얻으려면 **--ro** 옵션을 생략할 수 있습니다.

```
guestfish -d RHEL3 -i

Welcome to guestfish, the libguestfs filesystem interactive shell for
 editing virtual machine filesystems.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

 Operating system: Red Hat Enterprise Linux AS release 3 (Taroon Update 9)
 /dev/vda2 mounted on /
 /dev/vda1 mounted on /boot

 ><fs> edit /boot/grub/grub.conf
```

*파일을 편집하는 명령은 ,**vi** 및 **emacs** 를 포함합니다. 파일 및 디렉토리 작성을 위한 많은 명령(예: 쓰기,**mkdir**,**upload**, **tar-in**)도 존재합니다.*

*16.4.3. **fish**를 사용한 기타 작업*

*또한 파일 시스템을 포맷하고, 파티션을 만들고, **LVM** 논리 볼륨을 만들고 크기를 조정하며, **mkfs**, **part-add** ,**part-add**, **lvresize** ,**lvresize**,**lvcreate** 및 **pvcreate** 과 같은 명령을 사용하여 훨씬 더 많은 작업을 수행할 수 있습니다.*

*16.4.4. **fish**를 사용한 쉘 스크립트*

*요구 사항에 따라 **schedule**를 대화형으로 사용하는 방법을 잘 알고 있으면 쉘 스크립트를 작성하는 것이 유용할 수 있습니다. 다음은 새로운 **MOTD**(요일 메시지)를 게스트에 추가하는 간단한 쉘 스크립트입니다.*

```
#!/bin/bash -
set -e
guestname="$1"

guestfish -d "$guestname" -i <<'EOF'
  write /etc/motd "Welcome to Acme Incorporated."
  chmod 0644 /etc/motd
EOF
```

*16.4.5. **Augeas** 및 **libguestfs** 스크립팅*

***libguestfs**와 **Augeas**를 결합하면 **Linux** 게스트 가상 머신 구성을 조작하는 스크립트를 작성할 때 도움이 될 수 있습니다. 예를 들어 다음 스크립트에서는 **Augeas**를 사용하여 게스트 가상 시스템의 키보드 구성을 구문 분석하고 레이아웃을 출력합니다. 이 예제는 **Red Hat Enterprise Linux**를 실행하는 게스트 가상 머신에서만 작동합니다.*

```
#!/bin/bash -
set -e
guestname="$1"

guestfish -d "$1" -i --ro <<'EOF'
  aug-init / 0
  aug-get /files/etc/sysconfig/keyboard/LAYOUT
EOF
```

***Augeas**는 또한 설정 파일을 수정하는 데 사용할 수 있습니다. 위의 스크립트를 수정하여 키보드 레이아웃을 변경할 수 있습니다.*

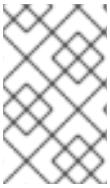
```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$1" -i <<'EOF'
   aug-init / 0
   aug-set /files/etc/sysconfig/keyboard/LAYOUT '"gb"'
   aug-save
 EOF
```

두 스크립트 간의 세 가지 변경 사항에 유의하십시오.

1. *두 번째 예제에서는 **--ro** 옵션이 제거되어 게스트 가상 머신에 쓸 수 있는 기능이 제공되었습니다.*

2. ***aug-get** 명령이 값을 가져오는 대신 수정하도록 **aug-set** 로 변경되었습니다. 새 값은 **"gb"(quotes** 포함)입니다.*

3. *여기에서 **aug-save** 명령이 사용되므로 **Augeas**는 변경 사항을 디스크에 씁니다.*



참고

***Augeas**에 대한 자세한 내용은 웹 사이트에서 **http://augeas.net** 찾을 수 있습니다.*

***fish**는 이 소개 문서에서 다룰 수 있는 것보다 훨씬 많은 작업을 수행할 수 있습니다. 예를 들어 처음부터 디스크 이미지 생성:*

```
guestfish -N fs
```

또는 디스크 이미지에서 전체 디렉터리를 복사하십시오.

```
><fs> copy-out /home /tmp/home
```

*자세한 내용은 **man** 페이지**fish(1)**를 참조하십시오.*

16.5. 기타 명령

이 섹션에서는**fish**를 사용하여 게스트 가상 머신 디스크 이미지를 보고 편집하는 것과 더 간단한 툴에 대해 설명합니다.

- **virt-cat** 은**fish** 다운로드 명령과 유사합니다. 단일 파일을 다운로드하여 게스트 가상 머신에 표시합니다. 예를 들어 다음과 같습니다.

```
# virt-cat RHEL3 /etc/ntp.conf | grep ^server
server     127.127.1.0       # local clock
```

- **virt-edit** 은**fish edit** 명령과 유사합니다. 게스트 가상 시스템에서 단일 파일을 대화형으로 편집하는 데 사용할 수 있습니다. 예를 들어 부팅하지 않는 **Linux** 기반 게스트 가상 머신에서 **grub.conf** 파일을 편집해야 할 수 있습니다.

```
# virt-edit LinuxGuest /boot/grub/grub.conf
```

virt-edit 에는 단일 파일에 대한 간단한 비대화형 변경을 수행하는 데 사용할 수 있는 다른 모드가 있습니다. 이를 위해 **-e** 옵션이 사용됩니다. 예를 들어 이 명령은 **Linux** 게스트 가상 머신의 **root** 암호를 암호가 없도록 변경합니다.

```
# virt-edit LinuxGuest /etc/passwd -e 's/^root:.*?:/root::/'
```

- **virt-ls** 는 **6.7 ls,ll** 및 **find** 명령과 유사합니다. 디렉토리 또는 디렉터리를 나열하는 데 사용됩니다(리브적으로). 예를 들어 다음 명령은 **Linux** 게스트 가상 머신에서 **/home** 아래에 파일 및 디렉토리를 재귀적으로 나열합니다.

```
# virt-ls -R LinuxGuest /home/ | less
```

16.6. VIRT-RESCUE: 구조 쉘

이 섹션에서는 **rescue** 쉘 사용에 대한 정보를 제공합니다.

16.6.1. 소개

이 섹션에서는 가상 시스템의 복구 **CD**와 유사한 것으로 간주될 수 있는 **virt-rescue** 에 대해 설명합니다. 유지 관리를 수행하여 오류를 수정하고 게스트 가상 머신을 복구할 수 있도록 게스트 가상 머신을 복구 쉘로 부팅합니다.

virt-rescue와**fish** 사이에 몇 가지 중복이 있습니다. 다양한 용도를 구별하는 것이 중요합니다. **virt-**

rescue는 일반 **Linux** 파일 시스템 도구를 사용하여 대화형 에드혹 변경을 수행하는 것입니다. 특히 실패한 게스트 가상 머신을 회수하는 데 적합합니다. **virt-rescue**는 스크립팅할 수 없습니다.

반면,**fish**는 대화식으로 사용할 수도 있지만 공식 명령 집합(**libguestfs API**)을 통해 스크립트화되고 구조화된 변경을 수행하는 데 특히 유용합니다.

16.6.2. virt-rescue 실행

게스트 가상 머신에서 **virt-rescue** 를 사용하기 전에 **guest** 가상 머신이 실행되고 있지 않은지 확인합니다. 그렇지 않으면 디스크 손상이 발생합니다. **guest** 가상 머신이 활성 상태가 되지 않은 경우 다음을 입력합니다.

```
virt-rescue GuestName
```

(여기서 **guestName**은 **libvirt**에 알려진 게스트 이름) 또는:

```
virt-rescue /path/to/disk/image
```

(여기서 경로는 게스트 가상 머신 디스크를 포함하는 모든 논리 볼륨, **LUN** 등)일 수 있습니다.

virt-rescue가 복구 **VM**이 부팅되므로 먼저 출력 스크롤이 표시됩니다. 결국 다음이 표시됩니다.

```
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>
```

여기에서 쉘 프롬프트는 일반 **bash** 쉘이며, 일반 **Red Hat Enterprise Linux** 명령 세트를 사용할 수 있습니다. 예를 들어 다음을 입력할 수 있습니다.

```
><rescue> fdisk -l /dev/vda
```

이전 명령은 디스크 파티션을 나열합니다. 파일 시스템을 마운트하려면 사용자가 원하는 항목을 마운트할 수 있도록 복구 시스템의 빈 디렉토리인 **/sysroot** 아래에 마운트하는 것이 좋습니다. / 아래의 파일은

복구 VM 자체의 파일입니다.

```
><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
 [...]
```

*게스트 가상 시스템을 다시 비우고 나면 **exit** 또는 **Ctrl+d** 를 입력하여 쉘을 종료합니다.*

***virt-rescue** 에는 많은 명령줄 옵션이 있습니다. 가장 자주 사용되는 옵션은 다음과 같습니다.*

- ***--ro:** 게스트 가상 머신의 읽기 전용 모드로 **Operate.** 변경 사항은 저장되지 않습니다. 이를 사용하여 게스트 가상 머신을 실험할 수 있습니다. 쉘을 종료하는 즉시 모든 변경 사항이 삭제됩니다.*

- ***--network: rescue** 쉘에서 네트워크 액세스를 활성화합니다. 필요한 경우 **RPM** 또는 기타 파일을 게스트 가상 시스템으로 다운로드합니다.*

16.7. VIRT-DF: 디스크 사용량 모니터링

*이 섹션에서는 **virt-df** 를 사용하여 디스크 사용량 모니터링에 대한 정보를 제공합니다.*

16.7.1. 소개

*이 섹션에서는 디스크 이미지 또는 게스트 가상 머신에서 파일 시스템 사용량을 표시하는 **virt-df** 에 대해 설명합니다. **Linux df** 명령과 유사하지만 가상 머신의 경우도 유사합니다.*

16.7.2. virt-df 실행

디스크 이미지에 있는 모든 파일 시스템에 대한 파일 시스템 사용량을 표시하려면 다음을 입력합니다.

```
# virt-df /dev/vg_guests/RHEL6
 Filesystem                1K-blocks       Used  Available  Use%
```

```
RHEL6:/dev/sda1              101086     10233     85634   11%
RHEL6:/dev/VolGroup00/LogVol00 7127864    2272744    4493036   32%
```

*(여기서 **/dev/vg_guests/RHEL6** 은 **Red Hat Enterprise Linux 6** 게스트 가상 머신 디스크 이미지입니다. 이 경우 경로는 이 디스크 이미지가 있는 호스트 물리적 시스템 논리 볼륨입니다.)*

*자체적으로 **virt-df** 를 사용하여 모든 게스트 가상 머신(예: **libvirt**에 알려진)에 대한 정보를 나열할 수도 있습니다. **virt-df** 명령은 **-h (human-readable)** 및 **-i (**블록 대신 **inode** 표시)와 같은 표준 **df** 와 동일한 옵션 중 일부를 인식합니다.*

***virt-df** 는 **Windows** 게스트 가상 머신에서도 작동합니다.*

```
# virt-df -h
 Filesystem                  Size       Used  Available  Use%
 F14x64:/dev/sda1            484.2M      66.3M     392.9M   14%
 F14x64:/dev/vg_f14x64/lv_root    7.4G       3.0G       4.4G   41%
 RHEL6brewx64:/dev/sda1        484.2M      52.6M     406.6M   11%
 RHEL6brewx64:/dev/vg_rhel6brewx64/lv_root
                           13.3G       3.4G       9.2G   26%
 Win7x32:/dev/sda1            100.0M      24.1M      75.9M   25%
 Win7x32:/dev/sda2      19.9G      7.4G      12.5G   38%
```

참고

*라이브 게스트 가상 머신에서 **virt-df** 를 안전하게 사용할 수 있습니다. 읽기 전용 권한만 있으면 되기 때문입니다. 하지만 게스트 가상 머신 내에서 실행 중인 **df** 명령의 숫자와 정확히 같을 것으로 예상해서는 안 됩니다. 이는 디스크에 있는 항목이 라이브 게스트 가상 머신 상태와 약간 동기화되지 않기 때문입니다. 그러나 분석 및 모니터링을 위해 충분한 접근법이 있어야합니다.*

***virt-df**는 통계를 모니터링 도구, 데이터베이스 등에 통합할 수 있도록 설계되었습니다. 이를 통해 시스템 관리자는 디스크 사용량의 추세에 대한 보고서를 생성하고 게스트 가상 머신이 디스크 공간이 부족해질 경우 경고를 생성할 수 있습니다. 이렇게 하려면 **--csv** 옵션을 사용하여 머신에서 읽을 수 있는 **Comma-Separated-Values (CSV)** 출력을 생성해야 합니다. **CSV** 출력은 대부분의 데이터베이스, 스프레드 시트 소프트웨어 및 다양한 다른 도구 및 프로그래밍 언어에서 읽을 수 있습니다. 원시 **CSV**는 다음과 같습니다.*

```
# virt-df --csv WindowsGuest
 Virtual Machine,Filesystem,1K-blocks,Used,Available,Use%
 Win7x32,/dev/sda1,102396,24712,77684,24.1%
 Win7x32,/dev/sda2,20866940,7786652,13080288,37.3%
```

추세 및 알림을 생성하기 위해 이 출력을 처리하는 방법에 대한 리소스와 아이디어를 보려면 다음

http://libguestfs.org/virt-df.1.html URL을 참조하십시오.

16.8. VIRT-RESIZE: 게스트 가상 머신 오프라인 복원

이 섹션에서는 오프라인 게스트 가상 머신 크기 조정에 대한 정보를 제공합니다.

16.8.1. 소개

이 섹션에서는 게스트 가상 머신을 확장하거나 줄이기 위한 툴인 **virt-resize** 에 대해 설명합니다. 오프라인 상태인 게스트 가상 머신에만 사용할 수 있습니다(대기 종료). 게스트 가상 머신 이미지를 복사하고 원래 디스크 이미지를 그대로 두는 방식으로 작동합니다. 원본 이미지를 백업으로 사용할 수 있기 때문에 이상적이지만 디스크 공간의 양 두 배에 필요한 장단점이 있습니다.

16.8.2. 디스크 이미지 확장

이 섹션에서는 디스크 이미지를 확장하는 간단한 사례를 보여줍니다.

1. 크기를 조정할 디스크 이미지를 찾습니다. **libvirt** 게스트 가상 머신에 **virsh dumpxml GuestName** 명령을 사용할 수 있습니다.

2. **guest** 가상 머신을 확장하는 방법을 결정합니다. 다음 출력에 표시된 대로 게스트 가상 머신 디스크에서 **virt-df -h** 및 **virt-list-partitions -lh** 를 실행합니다.

```
# virt-df -h /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-list-partitions -lh /dev/vg_guests/RHEL6
/dev/sda1 ext3 101.9M
/dev/sda2 pv 7.9G
```

이 예제에서는 다음을 수행하는 방법을 보여줍니다.

- 첫 번째 (부팅) 파티션의 크기를 약 **100MB**에서 **500MB**로 늘립니다.
- 총 디스크 크기를 **8GB**에서 **16GB**로 늘립니다.

- *두 번째 파티션을 확장하여 남은 공간을 채웁니다.*

- */dev/VolGroup00/LogVol00 을 확장하여 두 번째 파티션에 새 공간을 채웁니다.*

1. *guest 가상 머신이 종료되었는지 확인합니다.*

2. *원래 디스크의 이름을 백업으로 변경합니다. 이 작업을 수행하는 방법은 원래 디스크의 호스트 물리적 시스템 스토리지 환경에 따라 다릅니다. 파일로 저장된 경우 mv 명령을 사용합니다. 논리 볼륨(이 예에서 설명한 것처럼)의 경우 lvrename 을 사용하십시오.*

```
# lvrename /dev/vg_guests/RHEL6 /dev/vg_guests/RHEL6.backup
```

3. *새 디스크를 만듭니다. 이 예제의 요구 사항은 총 디스크 크기를 최대 16GB까지 확장하는 것입니다. 여기에서 논리 볼륨이 사용되므로 다음 명령이 사용됩니다.*

```
# lvcreate -L 16G -n RHEL6 /dev/vg_guests
Logical volume "RHEL6" created
```

4. *2단계의 요구 사항은 다음 명령으로 표시됩니다.*

```
# virt-resize \
    /dev/vg_guests/RHEL6.backup /dev/vg_guests/RHEL6 \
    --resize /dev/sda1=500M \
    --expand /dev/sda2 \
    --LV-expand /dev/VolGroup00/LogVol00
```

처음 두 개의 인수는 입력 디스크 및 출력 디스크입니다. --resize /dev/sda1=500M 첫 번째 파티션의 크기를 500MB까지 조정합니다. --expand /dev/sda2 는 두 번째 파티션을 확장하여 나머지 공간을 모두 채웁니다. --LV-expand /dev/VolGroup00/LogVol00 은 guest 가상 머신 논리 볼륨을 확장하여 두 번째 파티션에 추가 공간을 채웁니다.

virt-resize 는 출력에서 수행하는 작업을 설명합니다.

```
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. 가상 머신을 부팅합니다. 작동하는 경우 (및 철저하게 테스트한 후에) 백업 디스크를 삭제할 수 있습니다. 실패하면 가상 머신을 종료하고 새 디스크를 삭제한 다음 백업 디스크의 이름을 원래 이름으로 다시 변경합니다.

6. **virt-df** 또는 **virt-list-partitions** 를 사용하여 새 크기를 표시합니다.

```
# virt-df -h /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%
```

게스트 가상 머신의 크기를 조정하는 것은 정확한 과학이 아닙니다. **virt-resize** 에 실패하면 **virt-resize(1)** 매뉴얼 페이지를 검토하고 시도할 수 있는 여러 가지 팁이 있습니다. 이전의 **Red Hat Enterprise Linux** 게스트 가상 머신의 경우 **GRUB**에 관한 팁에 특별히 주의를 기울여야 할 수 있습니다.

16.9. VIRT-INSPECTOR: 게스트 가상 머신 검사

이 섹션에서는 **virt-inspector** 를 사용하여 게스트 가상 머신 검사에 대한 정보를 제공합니다.

16.9.1. 소개

virt-inspector 는 디스크 이미지를 검사하여 포함된 운영 체제를 확인하는 툴입니다.

참고

Red Hat Enterprise Linux 6.2는 이 프로그램의 두 가지 변형을 제공합니다. **virt-inspector** 는 **Red Hat Enterprise Linux 6.0**에서 발견된 원본 프로그램이며 현재는 더 이상 사용되지 않는 업스트림 프로그램입니다. **virt-inspector2** 는 새로운 업스트림 **virt-inspector** 프로그램과 동일합니다.

16.9.2. 설치

virt-inspector 및 문서를 설치하려면 다음 명령을 입력합니다.

```
# yum install libguestfs-tools libguestfs-devel
```

Windows 게스트 가상 머신을 처리하려면 **libguestfs-winsupport** 도 설치해야 합니다. 자세한 내용은 16.10.2절. “설치” 를 참조하십시오. **XML** 출력 및 출력에 대한 **Relax-NG** 스키마를 포함한 설명서는 **/usr/share/doc/libguestfs-devel-*/** 에 설치되어 있습니다. 여기서 "*"는 버전 번호 **libguestfs**로 대체됩니다.

16.9.3. virt-inspector 실행

다음 예와 같이 모든 디스크 이미지 또는 **libvirt** 게스트 가상 머신에 대해 **virt-inspector** 를 실행할 수 있습니다.

```
virt-inspector --xml disk.img > report.xml
```

또는 다음과 같이:

```
virt-inspector --xml GuestName > report.xml
```

결과는 **XML** 보고서(**report.xml**)입니다. **XML** 파일의 주요 구성 요소는 일반적으로 다음과 유사하게 단일 **<operatingsystem>** 요소를 포함하는 최상위 **<operatingsystems>** 요소입니다.

```
<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>
```

*이러한 보고서를 처리하는 것은 **W3C** 표준 **XPath** 쿼리를 사용하여 가장 효과적으로 수행됩니다. **Red Hat Enterprise Linux 6**에는 간단한 인스턴스에 사용할 수 있는 명령행 프로그램(**xpath**)이 함께 제공되지만 장기 및 고급 사용을 위해서는 원하는 프로그래밍 언어와 함께 **XPath** 라이브러리를 사용해야 합니다.*

*예를 들어 다음 **XPath** 쿼리를 사용하여 모든 파일 시스템 장치를 나열할 수 있습니다.*

```
virt-inspector --xml GuestName | xpath //filesystem/@dev
 Found 3 nodes:
 -- NODE --
 dev="/dev/sda1"
 -- NODE --
 dev="/dev/vg_f12x64/lv_root"
 -- NODE --
 dev="/dev/vg_f12x64/lv_swap"
```

다음을 입력하여 설치된 모든 애플리케이션의 이름을 나열할 수도 있습니다.

```
 virt-inspector --xml GuestName | xpath //application/name
 [...long list...]
```

16.10. VIRT-WIN-REG: 읽기 및 WINDOWS 레지스트리 편집

16.10.1. 소개

virt-win-reg 는 **Windows** 게스트 가상 머신에서 레지스트리를 조작하는 툴입니다. 레지스트리 키를 읽는 데 사용할 수 있습니다. 이를 사용하여 레지스트리를 변경할 수도 있지만 디스크 손상이 발생할 수 있으므로 실시간/실행 게스트 가상 머신에 대해 이 작업을 수행하지 않아야 합니다.

16.10.2. 설치

virt-win-reg 를 사용하려면 다음을 실행해야 합니다.

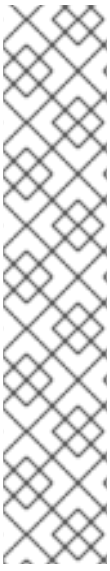
```
# yum install /usr/bin/virt-win-reg
```

16.10.3. virt-win-reg 사용

레지스트리 키를 읽으려면 게스트 가상 머신(또는 디스크 이미지)의 이름과 레지스트리 키 이름을 지정합니다. 원하는 키의 이름을 묶을 때 작은따옴표를 사용해야 합니다.

```
# virt-win-reg WindowsGuest \
  'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall' \
  | less
```

출력은 **Windows**의 **.REG** 파일에서 사용되는 표준 텍스트 기반 형식입니다.



참고

Hex-quoting은 문자열에 대해 이식 가능한 인코딩 방법을 제대로 정의하지 않기 때문에 문자열에 사용됩니다. 이것은 **.REG** 파일을 한 시스템에서 다른 시스템으로 전송할 때 **fidelity**를 보장할 수 있는 유일한 방법입니다.

이 간단한 **Perl** 스크립트를 통해 **virt-win-reg** 의 출력을 파이핑하여 **hex-quoted** 문자열을 인쇄 가능하게 만들 수 있습니다.

```
perl -MEncode -pe's?hex\((\d+)\):(\S+)?
$t=$1;$_=$2;s,\,,,g;"str($t):\"".decode(utf16le=>pack("H*",$_))."\""?eg'
```

오프라인 게스트 가상 머신의 **Windows** 레지스트리에 변경 사항을 병합하려면 먼저 **.REG** 파일을 준비해야 합니다. 여기에 이 작업을 수행하는 방법에 대한 많은 문서가 있습니다. **.REG** 파일을 준비했으면 다음을 입력합니다.

```
# virt-win-reg --merge WindowsGuest input.reg
```

그러면 게스트 가상 머신의 레지스트리가 업데이트됩니다.

16.11. 프로그래밍 언어에서 API 사용

libguestfs API는 Red Hat Enterprise Linux 6.2의 다음 언어에서 직접 사용할 수 있습니다. C, C++, Perl, Python, Java, Ruby 및 OCaml.

- **C 및 C++ 바인딩을 설치하려면 다음 명령을 입력합니다.**

```
# yum install libguestfs-devel
```

- **Perl 바인딩을 설치하려면 다음을 수행합니다.**

```
# yum install 'perl(Sys::Guestfs)'
```

- **Python 바인딩을 설치하려면 다음을 수행합니다.**

```
# yum install python-libguestfs
```

- **Java 바인딩을 설치하려면 다음을 수행합니다.**

```
# yum install libguestfs-java libguestfs-java-devel libguestfs-javadoc
```

- **Ruby 바인딩을 설치하려면 다음을 수행합니다.**

```
# yum install ruby-libguestfs
```

- **OCaml 바인딩을 설치하려면 다음을 수행합니다.**

```
# yum install ocaml-libguestfs ocaml-libguestfs-devel
```

각 언어의 바인딩은 기본적으로 동일하지만 약간의 구문 변경 사항이 있습니다. **C** 문의:

```
guestfs_launch (g);
```

-

Perl에서 다음과 같이 표시됩니다.

```
$g->launch ()
```

OCaml에서 다음과 같은 경우:

```
g#launch ()
```

이 섹션에서는 C API만 자세히 설명합니다.

C 및 C++ 바인딩에서 오류를 수동으로 확인해야 합니다. 다른 바인딩에서는 오류가 예외로 변환됩니다. 아래 예에 표시된 추가 오류 검사는 다른 언어에 필요하지 않지만 예외를 catch하는 코드를 추가하려는 경우도 있습니다. libguestfs API의 아키텍처에 대한 몇 가지 관심 사항은 다음 목록을 참조하십시오.

- *libguestfs API가 동기화됩니다. 각 호출 블록은 완료될 때까지 차단됩니다. 비동기적으로 호출하려면 스레드를 만들어야 합니다.If you want to make calls asynchronously, you have to create a thread.*
- *libguestfs API는 스레드로부터 안전하지 않습니다. 각 핸들은 단일 스레드에서만 사용해야 합니다. 또는 두 스레드가 동시에 하나의 처리에서 명령을 실행할 수 없도록 자체 뮤텍스 간에 핸들을 공유해야 합니다.*
- *동일한 디스크 이미지에서 여러 프로세스를 열 수 없습니다. 모든 핸들이 읽기 전용이지만 여전히 권장되지는 않는 경우 허용됩니다.*
- *해당 디스크 이미지(예: 라이브 VM)를 사용할 수 있는 경우 쓰기용 디스크 이미지를 추가하지 않아야 합니다. 이렇게 하면 디스크 손상이 발생합니다.*
- *현재 사용 중인 디스크 이미지에 읽기 전용 핸들을 열 수 있지만(예: 라이브 VM) 디스크 이미지를 읽는 시점에 디스크 이미지가 많이 기록되는 경우 결과를 예측할 수 없거나 특별히 일치하지 않을 수 있습니다.*

16.11.1. C 프로그램을 통한 API와 상호 작용

C 프로그램은 헤더 파일을 {s.> 포함하여 시작하며, 핸들을 생성합니다.

```
#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);
 }
```

이 프로그램을 파일에 저장합니다(test.c). 이 프로그램을 컴파일하고 다음 두 명령을 사용하여 실행합니다.

```
gcc -Wall test.c -o test -lguestfs
./test
```

이 단계에서는 출력을 출력하지 않아야 합니다. 이 섹션의 나머지 부분에서는 이 프로그램을 확장하여 새 디스크 이미지를 생성하고 파티션을 지정하고, ext4 파일 시스템으로 포맷한 다음 파일 시스템에 일부 파일을 생성하는 방법을 보여주는 예제를 보여줍니다. 디스크 이미지는 disk.img 라고 하며 현재 디렉터리에 생성됩니다.

이 프로그램의 개요는 다음과 같습니다.

- *핸들을 만듭니다.*
- *디스크에 디스크에 추가합니다.*
- *libguestfs 백엔드를 시작합니다.*

- *파티션, 파일 시스템 및 파일을 만듭니다.*

- *핸들을 닫고 종료합니다.*

다음은 수정된 프로그램입니다.

```
#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);
}
```

다음 두 명령을 사용하여 이 프로그램을 컴파일하고 실행합니다.

```
gcc -Wall test.c -o test -lguestfs
./test
```

*프로그램이 성공적으로 완료되는 경우 **disk.img** 라는 디스크 이미지로 남아 있어야 합니다. 이 이미지는**fish**로 검사할 수 있습니다.*

```
guestfish --ro -a disk.img -m /dev/sda1
><fs> ll /
><fs> cat /foo/resolv.conf
```

*기본적으로 **(C** 및 **C++** 바인딩의 경우에만) **libguestfs**는 **stderr**에 오류를 출력합니다. 오류 처리기를 설정하여 이 동작을 변경할 수 있습니다. **guestfs(3)** 도움말 페이지에서 이에 대해 자세히 설명합니다.*

16.12. VIRT-SYSPREP: 가상 머신 설정 재설정

***virt-sysprep** 명령줄 툴을 사용하여 게스트 가상 머신을 재설정하거나 구성 해제하여 복제를 수행할 수 있습니다. 이 프로세스에는 **SSH** 호스트 키, 영구 네트워크 **MAC** 구성 및 사용자 계정을 제거해야 합니다. **virt-sysprep** 는 예를 들어 **SSH** 키, 사용자 또는 로고를 추가하여 가상 머신을 사용자 지정할 수도 있습니다. 필요에 따라 각 단계를 활성화하거나 비활성화할 수 있습니다.*

*"**sysprep**"이라는 용어는 **Microsoft Windows** 시스템과 함께 사용되는 시스템 준비 도구*

(sysprep.exe)에서 파생됩니다. 이 경우에도 도구는 현재 Windows 게스트에서 작동하지 않습니다.

참고

libguestfs 및 fish 는 루트 권한이 필요하지 않습니다. 액세스 중인 디스크 이미지에 읽기 또는 쓰기에 대한 루트 액세스 권한이 필요한 경우에만 root로 실행하면 됩니다.

virt-sysprep 도구는 다음 명령을 사용하여 설치된 libguestfs-tools-c 패키지의 일부입니다.

```
$ yum install libguestfs-tools-c
```

또는 다음 명령을 사용하여 virt-sysprep 툴만 설치할 수 있습니다.

```
$ yum install /usr/bin/virt-sysprep
```

중요

virt-sysprep 은 게스트 또는 디스크 이미지를 즉시 수정합니다. virt-sysprep 을 사용하려면 게스트 가상 머신이 오프라인 상태여야 하므로 명령을 실행하기 전에 종료해야 합니다. 게스트 가상 시스템의 기존 콘텐츠를 보존하려면 먼저 디스크를 스냅샷, 복사 또는 복제해야 합니다. 디스크 복사 및 복제에 대한 자세한 내용은 libguestfs.org 를 참조하십시오.

다음 명령을 virt-sysprep 과 함께 사용할 수 있습니다.

표 16.1. virt-sysprep 명령

명령	설명	예제
--help	특정 명령 또는 전체 패키지에 대한 간략한 도움말 항목을 표시합니다. 추가 도움말은 virt-sysprep 매뉴얼 페이지를 참조하십시오.	$ **virt-sysprep --help**
-a [*file*] 또는 --add [*file*]	지정된 *파일* 을 게스트 가상 머신의 디스크 이미지여야 합니다. 디스크 이미지의 형식이 자동으로 탐지됩니다. 이 값을 재정의하고 특정 형식을 강제 적용하려면 **--format** 옵션을 사용합니다.	$ **virt-sysprep --add /dev/vms/disk.img**

명령	설명	예제
-c [*URI*] 또는 --connect [*URI*]	libvirt 를 사용하는 경우 지정된 URI에 연결합니다. 생략하면 KVM 하이퍼바이저를 통해 연결됩니다. 게스트 블록 장치를 직접 지정하는 경우 (**virt-sysprep -a**), libvirt 가 전혀 사용되지 않습니다.	$ **virt-sysprep -c qemu:///system**
-d [*guest*] 또는 --domain [*guest*]	지정된 게스트 가상 머신의 모든 디스크를 추가합니다. 도메인 UUID는 도메인 이름 대신 사용할 수 있습니다.	$ **virt-sysprep --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***
-n 또는 --dry-run 또는 --dryrun	게스트 가상 머신에서 읽기 전용 "dry run" sysprep 작업을 수행합니다. 이렇게 하면 sysprep 작업이 실행되지만 마지막에 디스크의 변경 사항이 취소됩니다.	$ **virt-sysprep -n**
--enable [*operations*]	지정된 *작업을* 활성화합니다. 사용 가능한 작업을 나열하려면 --list 명령을 사용합니다.	$ **virt-sysprep --enable ssh-hotkeys,udev-persistent-net**
--format [**raw**|**qcow2**|**auto**]	a 옵션의 기본값은 디스크 이미지의 형식을 자동으로 탐지하는 것입니다. 이 경우 명령줄에서 이어지는 -a 옵션에 대한 디스크 형식이 강제 적용됩니다. --format 자동 스위치를 후속 -a 옵션에 대한 자동 감지로 다시 사용합니다(위의 -a 명령 참조).	$ **virt-sysprep --format raw -a *disk.img*** 는 disk.img에 대해 원시 형식(자동 감지 없음)을 강제 적용하지만 **virt-sysprep --format raw -a *disk.img* --format auto -a *another.img*** *forces raw format (no auto-detection) 은 **다른.img** 의 자동 감지로 되돌아갑니다. 신뢰할 수 없는 원시 형식 게스트 디스크 이미지가 있는 경우 이 옵션을 사용하여 디스크 형식을 지정해야 합니다. 이렇게 하면 악의적인 게스트의 보안 문제가 발생할 수 있습니다.*
--list-operations	virt-sysprep 프로그램에서 지원하는 작업을 나열합니다. 해당 항목은 하나 이상의 공백으로 구분된 하나 이상의 필드가 포함된 행당 하나씩 나열됩니다. 출력의 첫 번째 필드는 **--enable** 플래그에 제공할 수 있는 작업 이름입니다. 두 번째 필드는 기본적으로 작업을 사용할 수 있는 경우 * 문자이거나 그렇지 않은 경우 비어 있습니다. 동일한 줄의 추가 필드에는 작업에 대한 설명이 포함되어 있습니다.	$ **virt-sysprep --list-operations**

명령	설명	예제
--mount-options	게스트 가상 시스템의 각 마운트 지점에 대한 마운트 옵션을 설정합니다. semi-separated mountpoint:options 쌍을 사용합니다. 쉘에서 보호하려면 이 목록을 따옴표로 설정해야 할 수도 있습니다.	$ **virt-sysprep --mount-options "/:notime"** 은 **notime** 작업을 사용하여 루트 디렉터리를 마운트합니다.
--SELinux-relabel 및 --no-selinux-relabel	virt-sysprep는 게스트의 첫 번째 부팅 시 SELinux 재레이블링을 항상 예약하는 것은 아닙니다. 경우에 따라 레이블 재지정이 수행됩니다(예: virt-sysprep가 파일을 수정한 경우). 그러나 모든 작업이 파일만 제거하는 경우 (예: **--enable delete --delete /some/file**을 사용하는 경우) 다시 레이블이 지정되지 않습니다. **--selinux-relabel** 옵션을 사용하면 항상 SELinux 재레이블링을 강제 적용하는 반면 **--no-selinux-relabel** 이 설정된 동안 재레이블이 예약되지 않습니다. 파일에 올바른 SELinux 레이블이 있는지 확인하려면 **--selinux-relabel** 을 사용하는 것이 좋습니다.	$ **virt-sysprep --selinux-relabel**
-q 또는 --quiet	로그 메시지의 인쇄를 방지합니다.	$ **virt-sysprep -q**
-V 또는 --verbose	디버깅 목적으로 상세 메시지를 활성화합니다.	$ **virt-sysprep -v**
-V 또는 --version	virt-sysprep 버전 번호를 표시하고 종료합니다.	$ **virt-sysprep -V**
--root-password	루트 암호를 설정합니다. 를 사용하여 새 암호를 명시적으로 지정하거나 선택한 파일의 첫 번째 줄에서 문자열을 사용할 수 있으므로 더 안전합니다.	$ **virt-sysprep --root-password password:***123456* **-a** *guest.img* 또는 $ **virt-sysprep --root-password 파일:***SOURCE_FILE_PATH* **-a** *guest.img*

*자세한 내용은 **libguestfs** 설명서를 참조하십시오.*

16.13. 문제 해결

테스트 도구는 **libguestfs**가 작동하는지 확인하는 데 사용할 수 있습니다. 정상적인 작업을 테스트하려면 **libguestfs(root** 액세스 없음)를 설치한 후 다음 명령을 실행합니다.

```
$ libguestfs-test-tool
```

이 툴은 **libguestfs**의 작업을 테스트하기 위해 많은 양의 텍스트를 출력합니다. 테스트에 성공하면 출력 끝에 다음 텍스트가 나타납니다.

```
===== TEST FINISHED OK =====
```

16.14. FURTHER 문서를 찾을 수 있는 위치

libguestfs 및 툴에 대한 설명서의 주요 소스는 **Unix** 도움말 페이지입니다. **API**에 대한 자세한 내용은 **guestfs 4)**에 설명되어 있습니다. **6.7**은 **6.7(1)**에 설명되어 있습니다. **virt** 툴은 자체 도움말 페이지(예: **virt-df(1))**로 문서화되어 있습니다.

17장. 게스트 가상 머신 관리를 위한 그래픽 사용자 인터페이스 도구

Red Hat Enterprise Linux 6는 **virt-manager** 외에도 게스트 가상 머신의 콘솔에 액세스할 수 있는 다음과 같은 도구를 제공합니다.

17.1. VIRT-VIEWER

virt-viewer 는 게스트 가상 시스템의 그래픽 콘솔을 표시하는 데 필요한 최소한의 명령줄 유틸리티입니다. 콘솔에 **VNC** 또는 **SPICE** 프로토콜을 사용하여 액세스할 수 있습니다. 게스트는 이름, **ID** 또는 **UUID** 로 참조할 수 있습니다. 게스트가 아직 실행되지 않은 경우 콘솔에 연결을 시도하기 전에 가 시작될 때까지 대기하도록 뷰어를 설정할 수 있습니다. 뷰어는 원격 호스트에 연결하여 콘솔 정보를 가져온 다음 동일한 네트워크 전송을 사용하여 원격 콘솔에 연결할 수도 있습니다.

virt-manager 와 비교하여**virt-viewer** 는 더 작은 기능 세트를 제공하지만 리소스 수요가 줄어듭니다. 또한 **virt-manager** 와 달리 대부분의 경우**virt-viewer** 는 **libvirt**에 대한 읽기-쓰기 권한이 필요하지 않습니다. 따라서 게스트에 연결하고 구성할 수는 없는 권한이 없는 사용자가 사용할 수 있습니다.

virt-viewer 유틸리티를 설치하려면 다음을 실행합니다.

```
# sudo yum install virt-viewer
```

구문

기본 **virt-viewer** 명령줄 구문은 다음과 같습니다.

```
# virt-viewer [OPTIONS] {guest-name|id|uuid}
```

기본 **virt-viewer** 명령줄 구문은 다음과 같습니다.

게스트 가상 머신에 연결

옵션 없이 사용하는 경우 **virt-viewer** 는 로컬 시스템의 기본 하이퍼바이저에서 연결할 수 있는 게스트를 나열합니다.

기본 하이퍼바이저를 사용하는 게스트 가상 머신에 연결하려면 다음을 수행합니다.

```
# virt-viewer guest-name-or-UUID
```

KVM-QEMU 하이퍼바이저를 사용하는 게스트 가상 머신에 연결하려면 다음을 수행합니다.

```
# virt-viewer --connect qemu:///system guest-name-or-UUID
```

TLS를 사용하여 원격 콘솔에 연결하려면 다음을 수행합니다.

```
# virt-viewer --connect xen://example.org/ guest-name-or-UUID
```

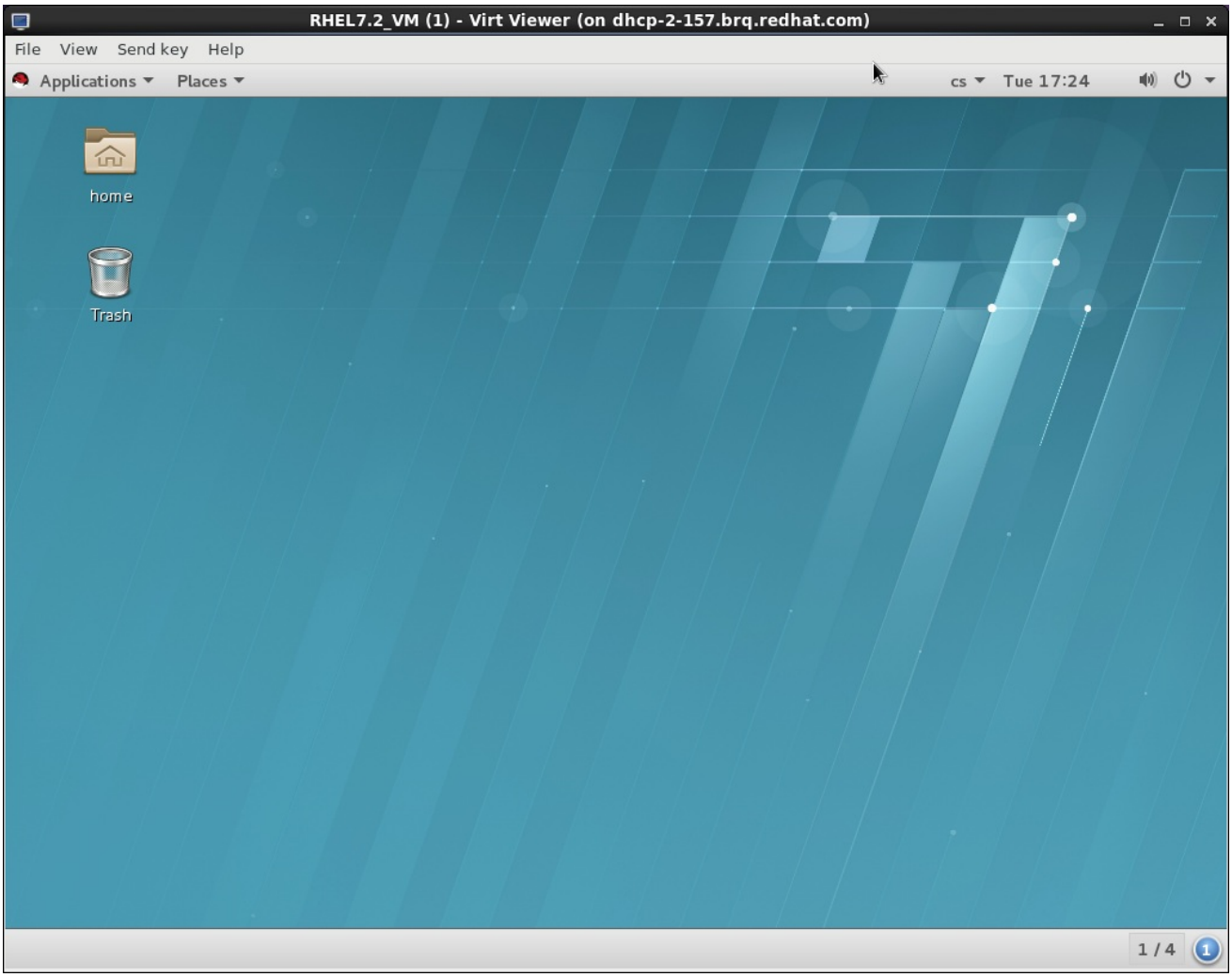
SSH를 사용하여 원격 호스트의 콘솔에 연결하려면 게스트 구성을 검색한 다음 콘솔에 직접 연결되지 않은 연결을 설정합니다.

```
# virt-viewer --direct --connect xen+ssh://root@example.org/ guest-name-or-UUID
```

인터페이스

기본적으로 **virt-viewer** 인터페이스는 게스트와 상호 작용하는 기본 툴만 제공합니다.

*그림 17.1. 샘플 **virt-viewer** 인터페이스*



핫키 설정

__virt-viewer__ 세션에 대한 사용자 정의 키보드 바로 가기(키라고도 함)를 생성하려면 __--hotkeys__ 옵션을 사용합니다.

```
# virt-viewer --hotkeys=action1=key-combination1[,action2=key-combination2] guest-name-or-UUID
```

다음 작업을 핫키에 할당할 수 있습니다.

- ***toggle-fullscreen***
- ***release-cursor***
- ***smartcard-insert***

- ***smartcard-remove***

*키 이름 조합 핫키는 대/소문자를 구분하지 않습니다. **hotkey** 설정은 향후 **virt-viewer** 세션으로 대체되지 않습니다.*

> ***예 17.1. virt-viewer 핫키 설정***
>
> ***testguest**라는 **KVM-QEMU** 게스트에 연결할 때 전체 화면 모드로 변경할 핫키를 추가하려면 다음을 수행합니다.*
>
> ```
> # virt-viewer --hotkeys=toggle-fullscreen=shift+f11 qemu:///system testguest
> ```

키오스크 모드

*키오스크 모드에서는 **virt-viewer** 사용자만 연결된 데스크탑과 상호 작용할 수 있으며, 게스트가 종료되지 않는 한 게스트 설정 또는 호스트 시스템과 상호 작용할 수 있는 옵션은 제공되지 않습니다. 예를 들어 관리자가 사용자의 작업을 지정된 게스트로 제한하려는 경우 유용합니다.*

*키오스크 모드를 사용하려면 **-k** 또는 **--kiosk** 옵션을 사용하여 게스트에 연결합니다.*

> ***예 17.2. 키오스크 모드에서 virt-viewer 사용***
>
> *시스템을 종료한 후 종료되는 키오스크 모드에서 **KVM-QEMU** 가상 머신에 연결하려면 다음 명령을 사용합니다.*
>
> ```
> # virt-viewer --connect qemu:///system guest-name-or-UUID --kiosk --kiosk-quit on-disconnect
> ```

그러나 키오스크 모드만으로는 게스트가 종료된 후 사용자가 호스트 시스템 또는 게스트 설정과 상호 작용하지 않도록 할 수 없습니다. 이를 위해서는 호스트에서 창 관리자 비활성화와 같은 추가 보안 조치가 필요합니다.

17.2. REMOTE-VIEWER

***remote-viewer** 는 **SPICE** 및 **VNC**를 지원하는 간단한 원격 데스크탑 디스플레이 클라이언트입니다. 대부분의 기능과 제한 사항을 **virt-viewer** 와 공유합니다.*

그러나 **virt-viewer**와 달리 **remote-viewer** 는 원격 게스트 디스플레이에 연결하기 위해 **libvirt**를 필요로 하지 않습니다. 따라서 **remote-viewer** 를 사용하여 **libvirt**와 상호 작용하거나 **SSH** 연결을 사용할 수 있는 권한을 제공하지 않는 원격 호스트의 가상 시스템에 연결할 수 있습니다.

remote-viewer 유틸리티를 설치하려면 다음을 실행합니다.

```
# sudo yum install virt-viewer
```

구문

기본 **remote-viewer** 명령줄 구문은 다음과 같습니다.

```
# remote-viewer [OPTIONS] {guest-name|id|uuid}
```

remote-viewer에서 사용할 수 있는 전체 옵션 목록을 보려면 **man remote-viewer** 를 사용합니다.

게스트 가상 머신에 연결

옵션 없이 사용하는 경우 **remote-viewer** 는 로컬 시스템의 기본 **URI**에서 연결할 수 있는 게스트를 나열합니다.

remote-viewer를 사용하여 특정 게스트에 연결하려면 **VNC/SPICE URI**를 사용합니다. **URI** 가져오기에 대한 자세한 내용은 **14.5.19**절. “그래픽 디스플레이에 대한 연결 **URI** 표시” 을 참조하십시오.

예 **17.3. SPICE**를 사용하여 게스트 디스플레이에 연결

SPICE 통신에 포트 **5900**을 사용하는 **"testguest"**라는 시스템의 **SPICE** 서버에 연결하려면 다음을 사용하십시오.

```
# remote-viewer spice://testguest:5900
```

예 **17.4. VNC**를 사용하여 게스트 디스플레이에 연결

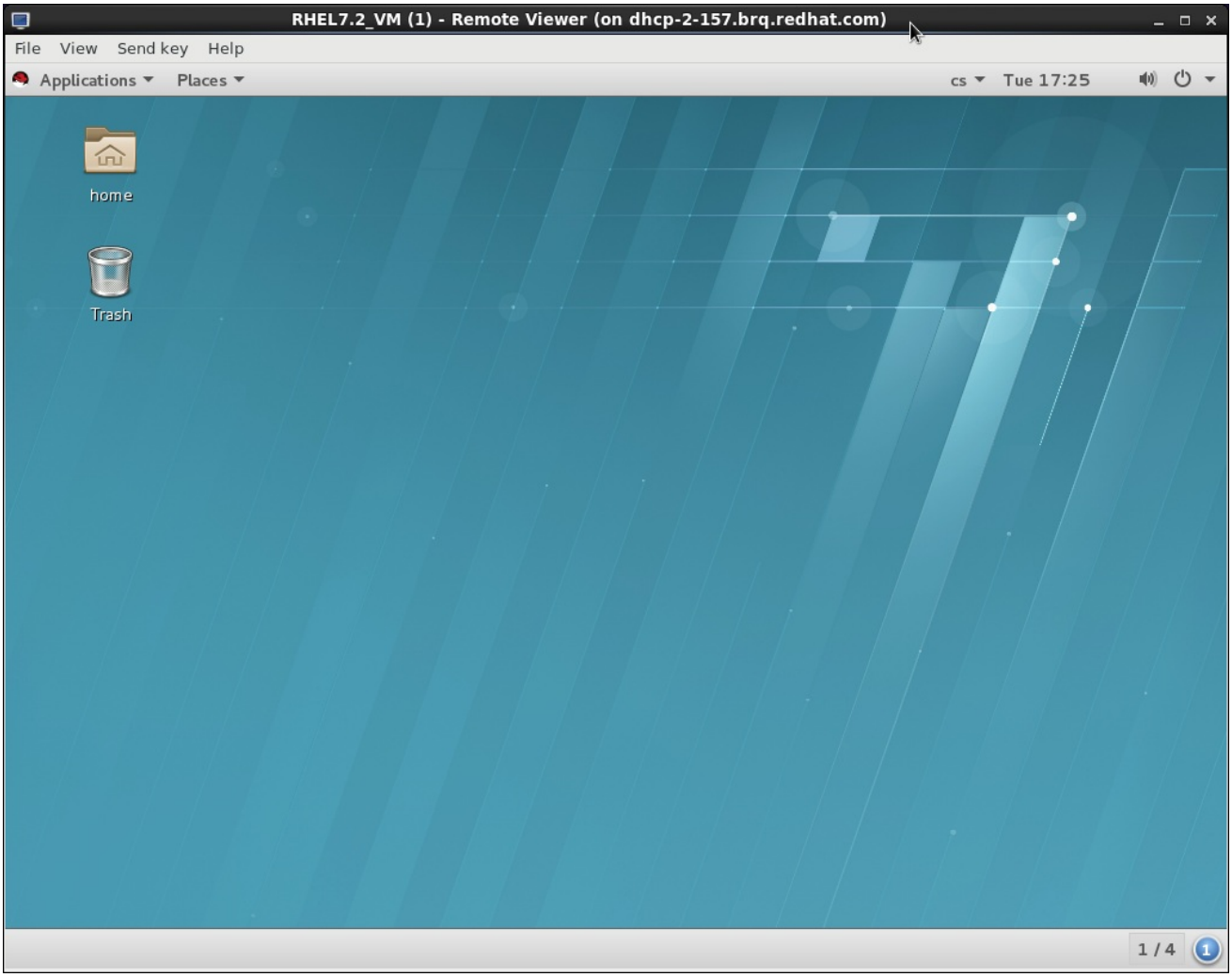
VNC 통신에 포트 **5900**을 사용하는 **testguest2** 라는 시스템의 **VNC** 서버에 연결하려면 다음을 사용합니다.

```
# remote-viewer vnc://testguest2:5900
```

인터페이스

*기본적으로 **remote-viewer** 인터페이스는 게스트와 상호 작용하는 기본 도구만 제공합니다.*

*그림 17.2. **remote-viewer** 인터페이스 샘플*



18장. 가상 네트워킹

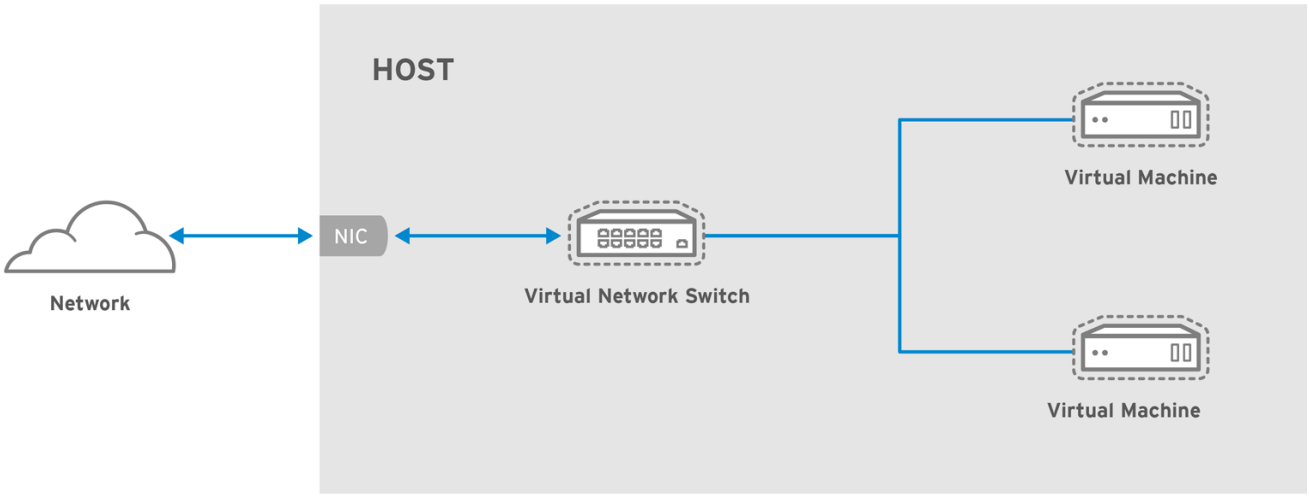
이 장에서는 **libvirt**를 사용하여 가상 네트워크를 생성, 시작, 중지, 제거 및 수정하는 데 필요한 개념을 소개합니다.

자세한 내용은 **libvirt** 참조 장을 참조하십시오.

18.1. 가상 네트워크 스위치

libvirt 가상 네트워킹은 가상 네트워크 스위치 의 개념을 사용합니다. 가상 네트워크 스위치는 호스트 물리적 시스템 서버에서 작동하고 가상 시스템(게스트)이 연결되는 소프트웨어 구조입니다. 게스트의 네트워크 트래픽은 다음 스위치를 통해 이동합니다.

그림 **18.1.** 두 게스트로 가상 네트워크 전환



Linux 호스트 물리적 시스템 서버는 가상 네트워크 스위치를 네트워크 인터페이스로 나타냅니다. **libvirtd** 데몬(**libvirtd**)이 처음 설치 및 시작되면 가상 네트워크 스위치를 나타내는 기본 네트워크 인터페이스는 **virbr0** 입니다.

이 **virbr0** 인터페이스는 다른 인터페이스와 마찬가지로 **ip** 명령으로 볼 수 있습니다.

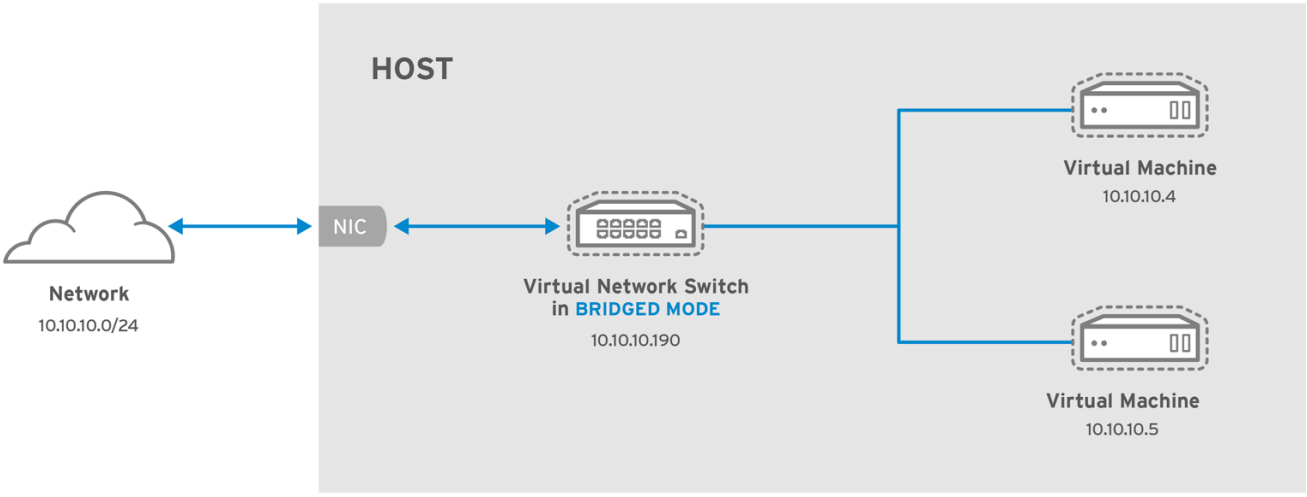
```
$ ip addr show virbr0
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 1b:c4:94:cf:fd:17 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
```

18.2. 브리지 모드

Bridged 모드를 사용하면 모든 게스트 가상 머신이 호스트 물리적 시스템과 동일한 서브넷에 나타납니다. 동일한 물리적 네트워크의 다른 모든 물리적 시스템은 가상 머신을 인식하고 가상 머신에 액세스할 수 있습니다. 브리징은 **OSI** 네트워킹 모델의 계층 **2**에서 작동합니다.

본딩과 함께 연결하여 하이퍼바이저에서 여러 물리적 인터페이스를 사용할 수 있습니다. 그런 다음 본딩을 브리지에 추가한 다음 게스트 가상 머신도 브리지에 추가됩니다. 그러나 본딩 드라이버에는 여러 가지 작동 모드가 있으며, 이러한 모드 중 일부만 가상 게스트 시스템이 사용 중인 브리지에서 작동합니다.

그림 **18.2.** 브리지 모드의 가상 네트워크 스위치



주의

게스트 가상 머신에서 사용해야 하는 유일한 본딩 모드는 모드 **1**, 모드 **2** 및 모드 **4** 입니다. 어떠한 경우에도 모드 **0**, **3**, **5** 또는 **6**을 사용해서는 안 됩니다. 또한, **arp-monitoring**가 작동하지 않기 때문에 **mii-monitoring**을 사용하여 본딩 모드를 모니터링해야 합니다.

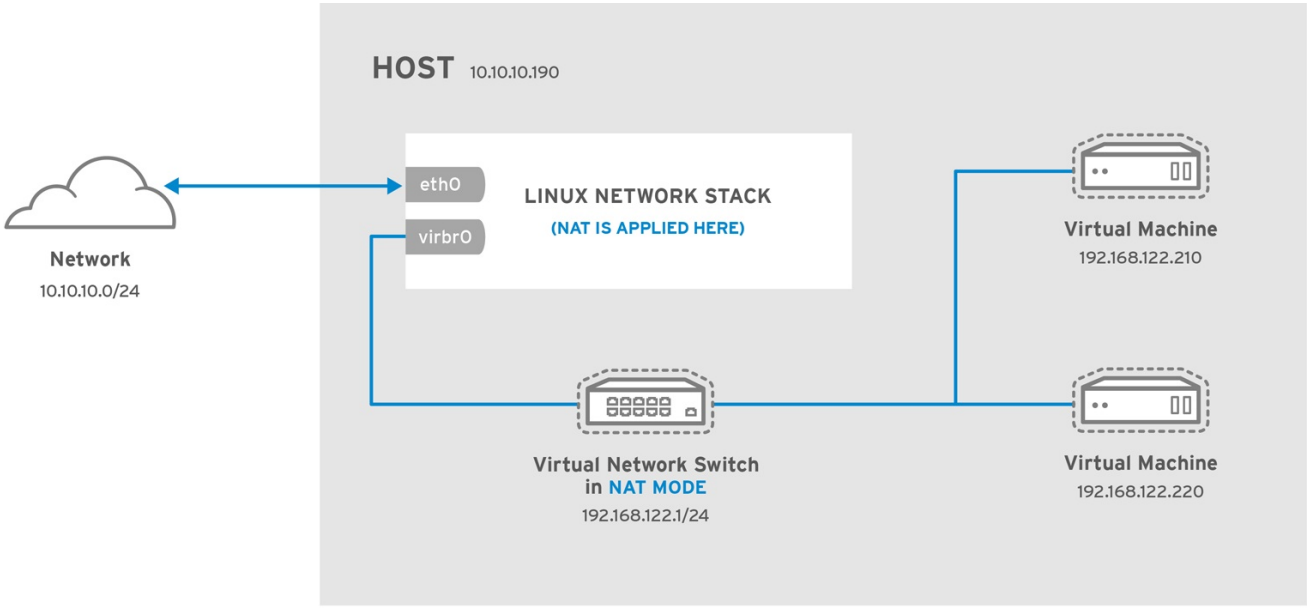
본딩 모드에 대한 자세한 내용은 본딩 모드에 대한 기술 자료 문서 또는 **Red Hat Enterprise Linux 6** 배포 가이드를 참조하십시오.

bridge_opts 매개변수에 대한 자세한 설명은 **Red Hat Virtualization** 관리 가이드 를 참조하십시오.

18.3. 네트워크 주소 변환 모드

기본적으로 가상 네트워크 스위치는 **NAT** 모드에서 작동합니다. **SNAT(Source-NAT)** 또는 **DNAT(Destination-NAT)** 대신 **IP** 마스커레이딩을 사용합니다. **IP** 마스커레이딩을 사용하면 연결된 게스트가 외부 네트워크와의 통신에 호스트 물리적 시스템 **IP** 주소를 사용할 수 있습니다. 기본적으로 호스트 물리적 시스템에 외부에 배치된 컴퓨터는 다음 다이어그램에 표시된 대로 가상 네트워크 스위치가 **NAT** 모드에서 작동하는 경우 내부의 게스트와 통신할 수 없습니다.

그림 **18.3.** 두 개의 게스트가 있는 **NAT**를 사용하는 가상 네트워크 스위치



주의

가상 네트워크 스위치는 **iptables** 규칙에 따라 구성된 **NAT**를 사용합니다. 스위치가 실행되는 동안 이러한 규칙을 편집하는 것은 권장되지 않습니다. 잘못된 규칙으로 인해 스위치가 통신할 수 없을 수 있습니다.

스위치가 실행되고 있지 않은 경우 다음을 실행하여 포트 마스커레이딩 범위를 생성하기 위해 전달 모드 **NAT**에 대해 **th** 공용 **IP** 범위를 설정할 수 있습니다.

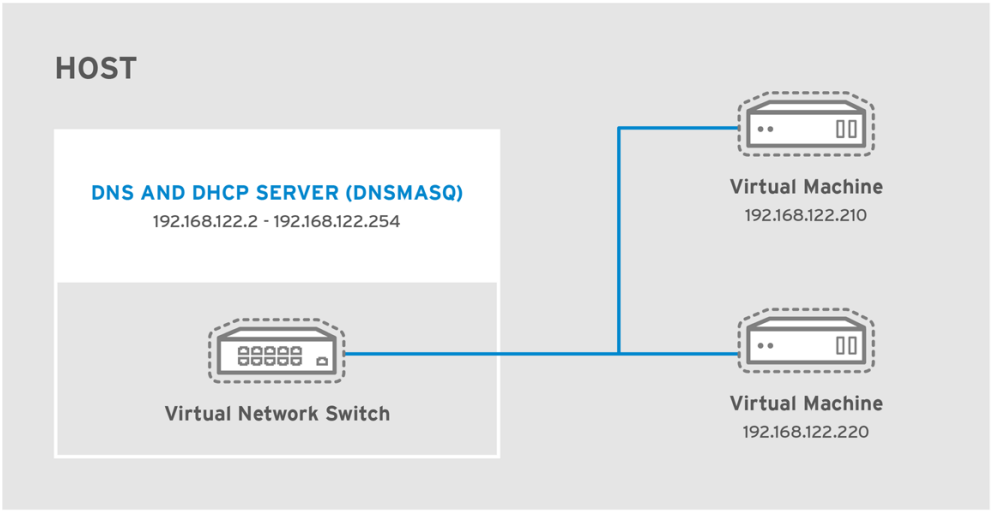
```
# iptables -j SNAT --to-source [start]-[end]
```

18.3.1. DNS 및 DHCP

IP 정보는 **DHCP**를 통해 게스트에 할당할 수 있습니다. 이러한 목적으로 가상 네트워크 스위치에 주소 풀을 할당할 수 있습니다. **libvirt**는 이를 위해 **dnsmasq** 프로그램을 사용합니다. **dnsmasq** 인스턴스는 필

*요한 각 가상 네트워크 스위치에 대해 **libvirt**에서 자동으로 구성 및 시작됩니다.*

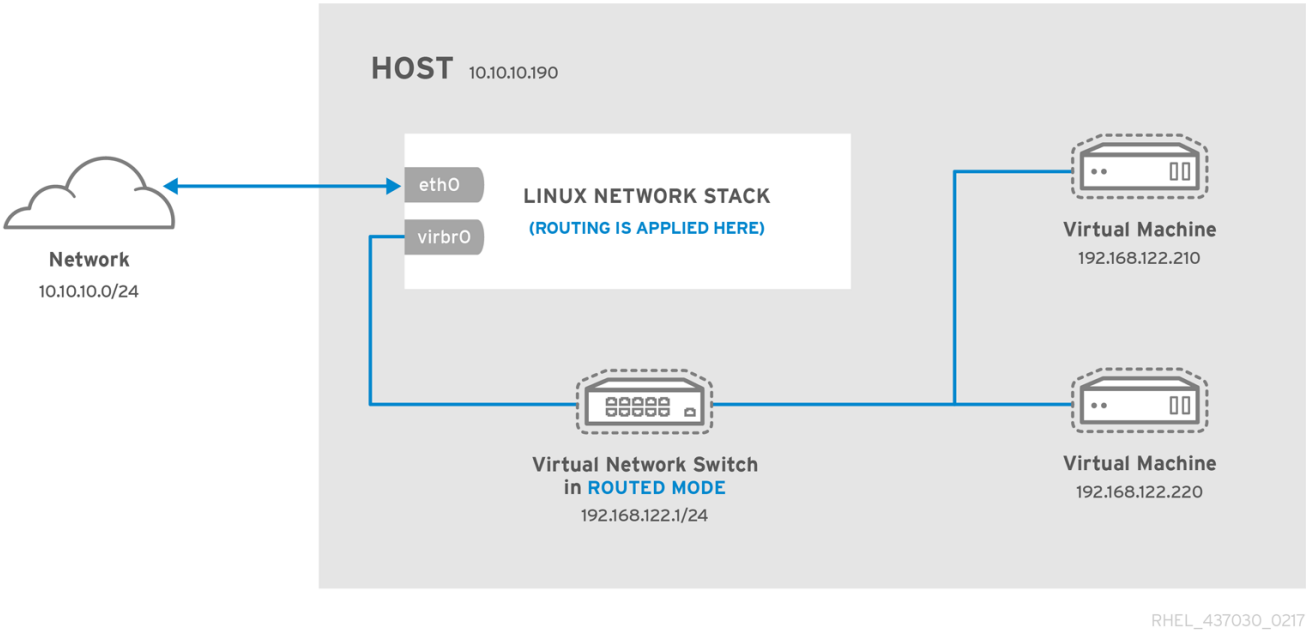
*그림 **18.4. dnsmasq**를 실행 중인 가상 네트워크 스위치*



18.4. 라우팅된 모드

***Routed** 모드를 사용하는 경우 가상 스위치는 호스트 물리적 시스템에 연결된 물리적 **LAN**에 연결하여 **NAT**를 사용하지 않고 트래픽을 다시 통과합니다. 가상 스위치는 모든 트래픽을 검사하고 네트워크 패킷 내에 포함된 정보를 사용하여 라우팅 결정을 내릴 수 있습니다. 이 모드를 사용하면 모든 가상 시스템이 가상 스위치를 통해 라우팅되는 고유한 서브넷에 있습니다. 이 상황은 물리적 네트워크의 다른 호스트 물리적 시스템이 수동 물리적 라우터 구성 없이 가상 시스템을 인식하지 못하기 때문에 항상 이상적인 것은 아니며 가상 머신에 액세스할 수 없습니다. 라우팅된 모드는 **OSI** 네트워킹 모델의 계층 **3**에서 작동합니다.*

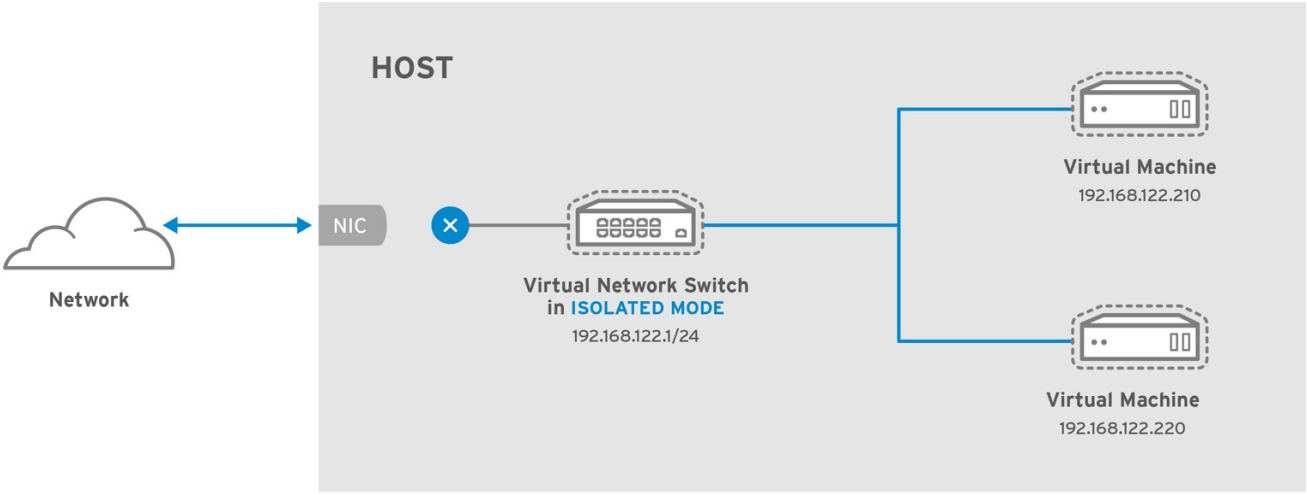
*그림 **18.5.** 라우팅 모드의 가상 네트워크 스위치*



18.5. 격리된 모드

*격리 모드를 사용하는 경우 가상 스위치에 연결된 게스트는 호스트 물리적 시스템과 통신할 수 있지만 해당 트래픽은 호스트 물리적 시스템 외부에서도 트래픽을 수신할 수 없습니다. 이 모드에서 **dnsmasq**를 사용하려면 **DHCP**와 같은 기본 기능에 필요합니다. 그러나 이 네트워크가 물리적 네트워크와 분리되어 있어도 **DNS** 이름은 계속 확인됩니다. 따라서 **DNS** 이름을 확인하지만 **ICMP** 에코 요청(**ping**) 명령이 실패할 때 상황이 발생할 수 있습니다.*

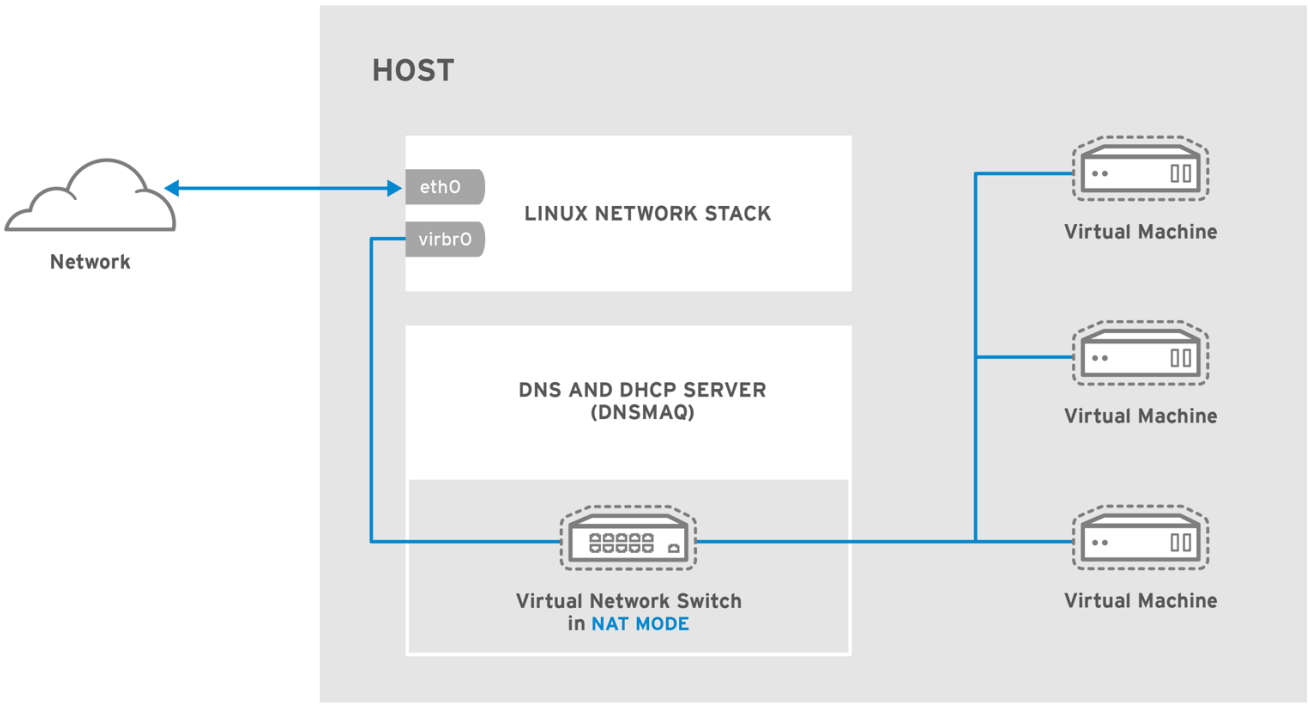
*그림 **18.6.** 격리된 모드에서 가상 네트워크 스위치*



18.6. 기본 설정

*libvirtd 데몬(libvirtd)을 처음 설치하면 **NAT** 모드에서 초기 가상 네트워크 스위치 구성이 포함됩니다. 이 구성은 설치된 게스트가 호스트 물리적 시스템을 통해 외부 네트워크와 통신할 수 있도록 하는 데 사용됩니다. 다음 이미지는 **libvirtd** 에 대한 이 기본 구성을 보여줍니다.*

그림 18.7. 기본 libvirt 네트워크 구성



참고

가상 네트워크는 특정 물리적 인터페이스로 제한할 수 있습니다. 이는 여러 인터페이스가 있는 물리적 시스템(예: eth0,eth1 및 eth2)에서 유용할 수 있습니다. 이는 라우팅 및 NAT 모드에서만 유용하며 dev=<interface > 옵션 또는 새 가상 네트워크를 생성할 때 virt-manager 에 정의할 수 있습니다.

18.7. COMMON SCENARIOS의 예

이 섹션에서는 다양한 가상 네트워킹 모드를 설명하고 몇 가지 예제 시나리오를 제공합니다.

18.7.1. 브리지 모드

브리지 모드는 OSI 모델의 계층 2에서 작동합니다. 를 사용하면 모든 게스트 가상 시스템이 호스트 물리적 시스템과 동일한 서브넷에 표시됩니다. bridged 모드의 가장 일반적인 사용 사례는 다음과 같습니다.

- 물리적 시스템과 함께 기존 네트워크에 게스트 가상 시스템을 배포하면 가상 시스템과 물리적 시스템의 차이를 최종 사용자에게 투명하게 만들 수 있습니다.
- 기존 물리적 네트워크 구성 설정을 변경하지 않고 게스트 가상 머신 배포.

- 기존 물리적 네트워크에서 쉽게 액세스할 수 있어야 하는 게스트 가상 머신 배포. 게스트 가상 머신을 물리적 네트워크에 배치하면 **DHCP**와 같은 기존 브로드캐스트 도메인 내의 서비스에 액세스해야 합니다.

- **VLAN**이 사용되는 기존 네트워크에 게스트 가상 머신을 연결합니다.

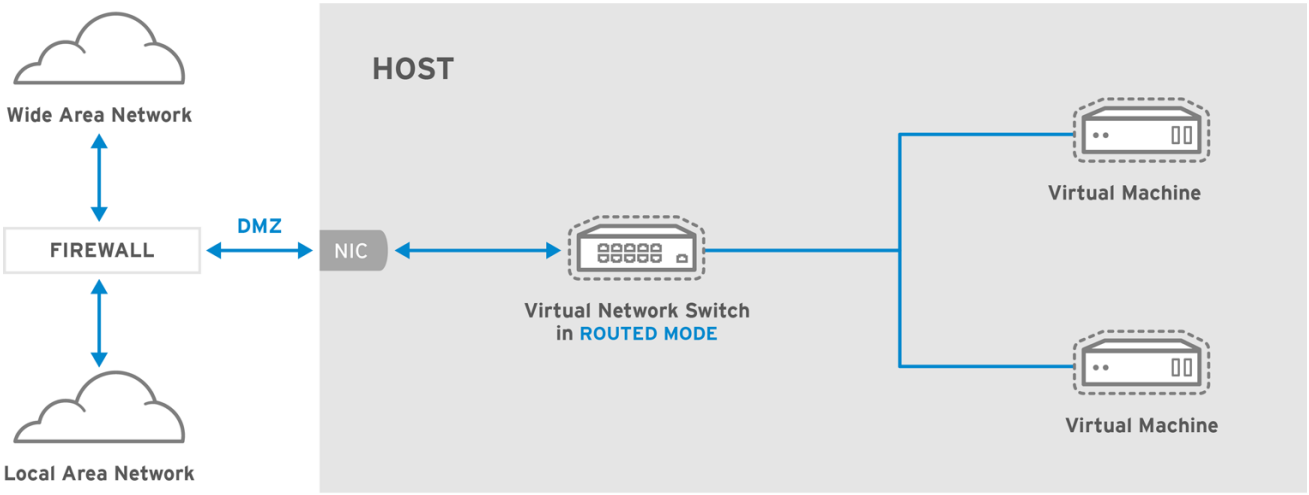
18.7.2. 라우팅된 모드

이 섹션에서는 라우팅 모드에 대한 정보를 제공합니다.

DMZ

보안상의 이유로 하나 이상의 노드가 제어된 서브네트워크에 배치되는 네트워크를 고려하십시오. 이와 같은 특수 서브네트워크를 배포하는 것은 일반적인 관행이며, 서브네트워크는**Period**으로 알려져 있습니다. 이 레이아웃에 대한 자세한 내용은 다음 다이어그램을 참조하십시오.

그림 18.8. 샘플 RedFish 구성

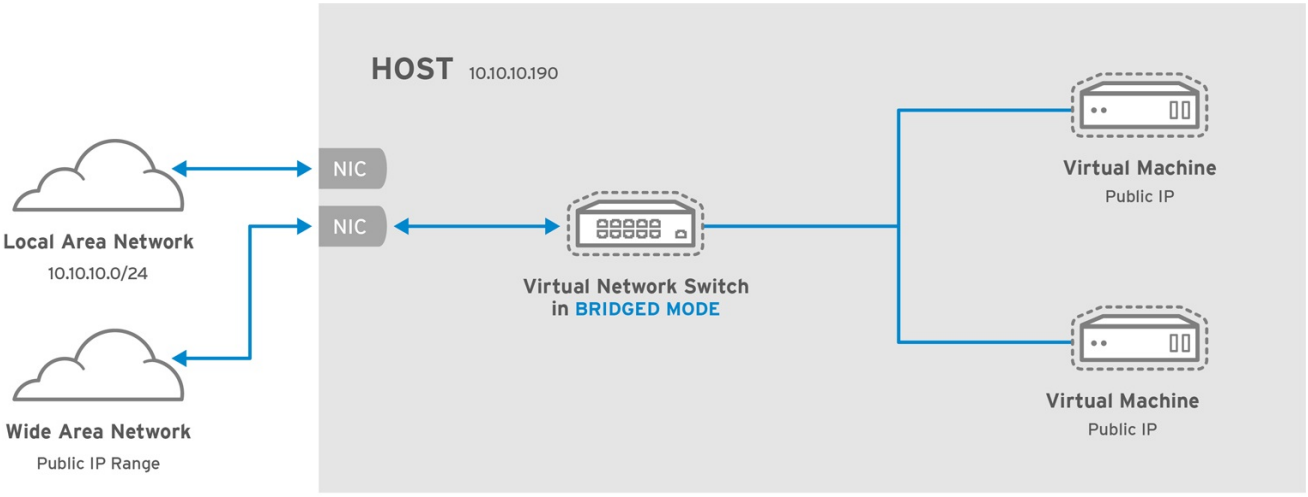


RedFish의 호스트 물리적 시스템은 일반적으로 **WAN**(외부) 호스트 물리적 시스템과 **LAN**(내부) 호스트 물리적 시스템에 대한 서비스를 제공합니다. 이를 위해서는 여러 위치에서 액세스할 수 있어야 하며 이러한 위치가 보안 및 신뢰 수준에 따라 다양한 방식으로 제어 및 운영된다는 점을 고려할 때 이 환경에 가장 적합한 구성입니다.

가상 서버 호스팅

각각 두 개의 물리적 네트워크 연결이 있는 여러 호스트 물리적 시스템이 있는 가상 서버 호스팅 회사를 고려해 보십시오. 한 인터페이스는 관리 및 회계에 사용되며, 다른 인터페이스는 가상 머신이 를 통해 연결하는 것입니다. 각 게스트에는 자체 공용 **IP** 주소가 있지만 호스트 물리적 시스템은 개인 **IP** 주소를 게스트의 관리로만 사용합니다. 이 시나리오를 이해하려면 다음 다이어그램을 참조하십시오.

그림 18.9. 가상 서버 호스팅 샘플 구성



18.7.3. NAT 모드

***NAT (Network Address Translation)** 모드는 기본 모드입니다. 직접 네트워크 가시성이 필요하지 않은 경우 테스트에 사용할 수 있습니다.*

18.7.4. 격리된 모드

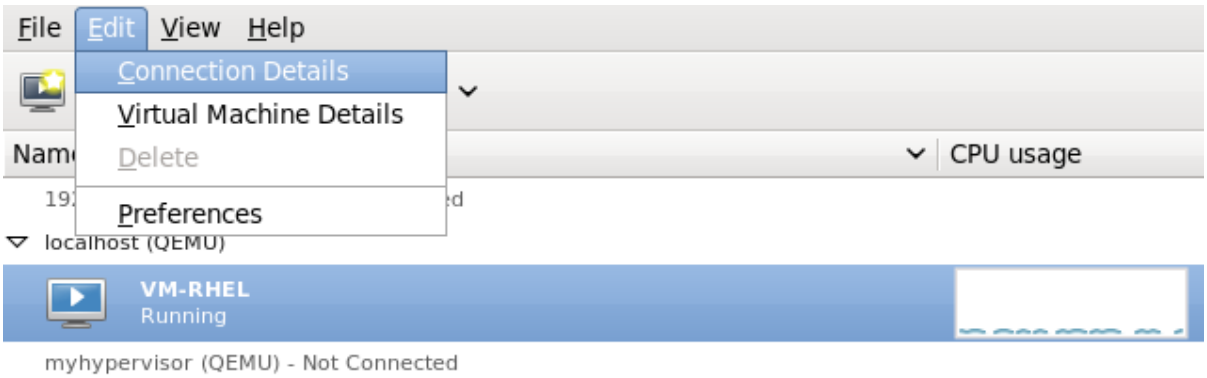
격리된 모드에서는 가상 시스템이 서로만 통신할 수 있습니다. 물리적 네트워크와 상호 작용할 수 없습니다.

18.8. 가상 네트워크 관리

시스템에서 가상 네트워크를 구성하려면 다음을 수행합니다.

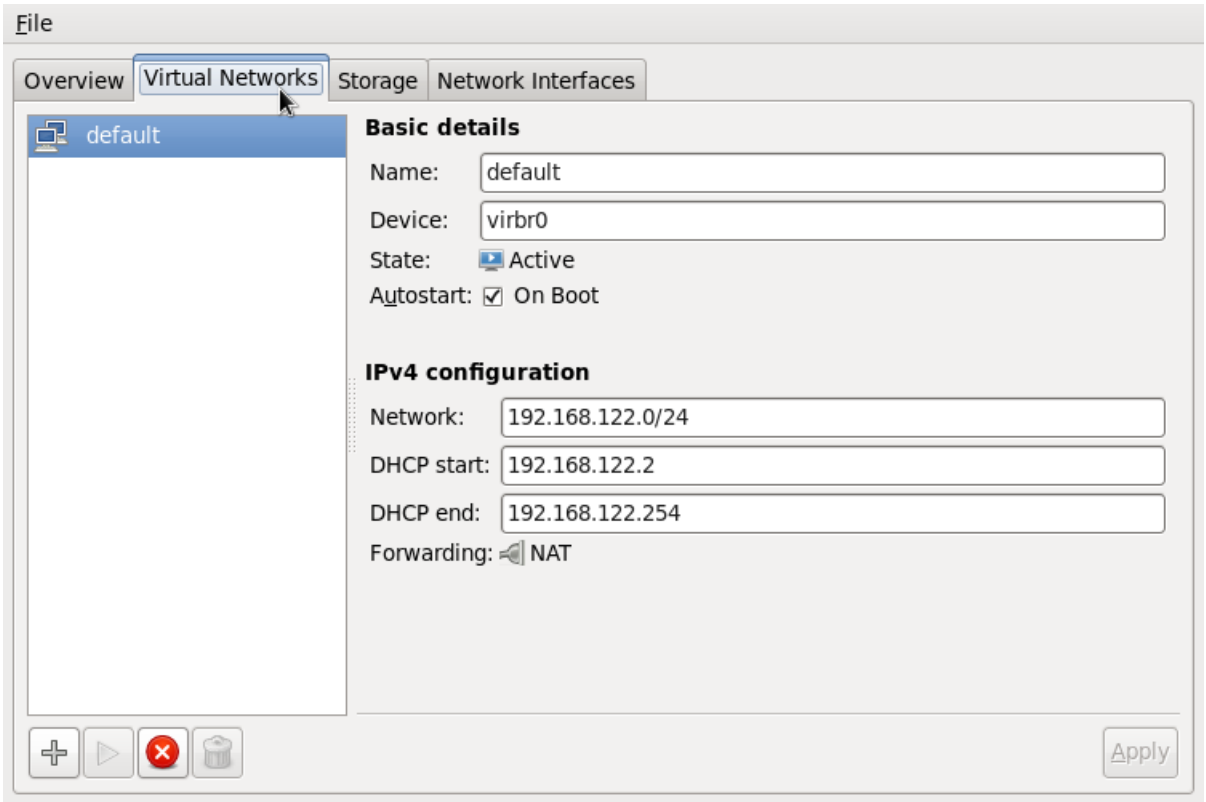
1. *편집 메뉴에서 연결 세부 정보를 선택합니다.*

그림 18.10. 호스트 물리적 시스템의 세부 정보 선택



2. *이렇게 하면 연결 세부 정보 메뉴가 열립니다. 가상 네트워크 탭을 클릭합니다.*

그림 18.11. 가상 네트워크 구성



3. *사용 가능한 모든 가상 네트워크는 메뉴의 왼쪽 상자에 나열됩니다. 이 상자에서 선택하고 필요에 따라 편집하면 가상 네트워크의 구성을 편집할 수 있습니다.*

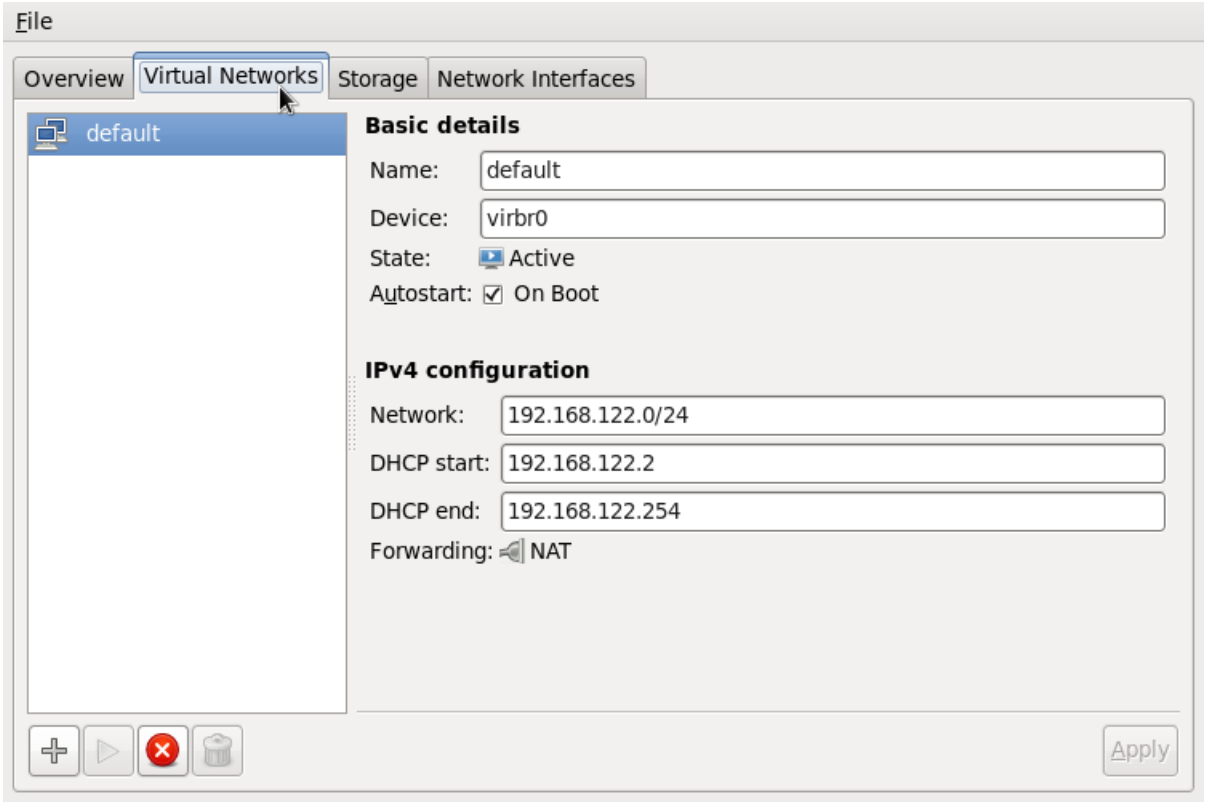
18.9. 가상 네트워크 생성

시스템에 가상 네트워크를 생성하려면 다음을 수행합니다.

1.

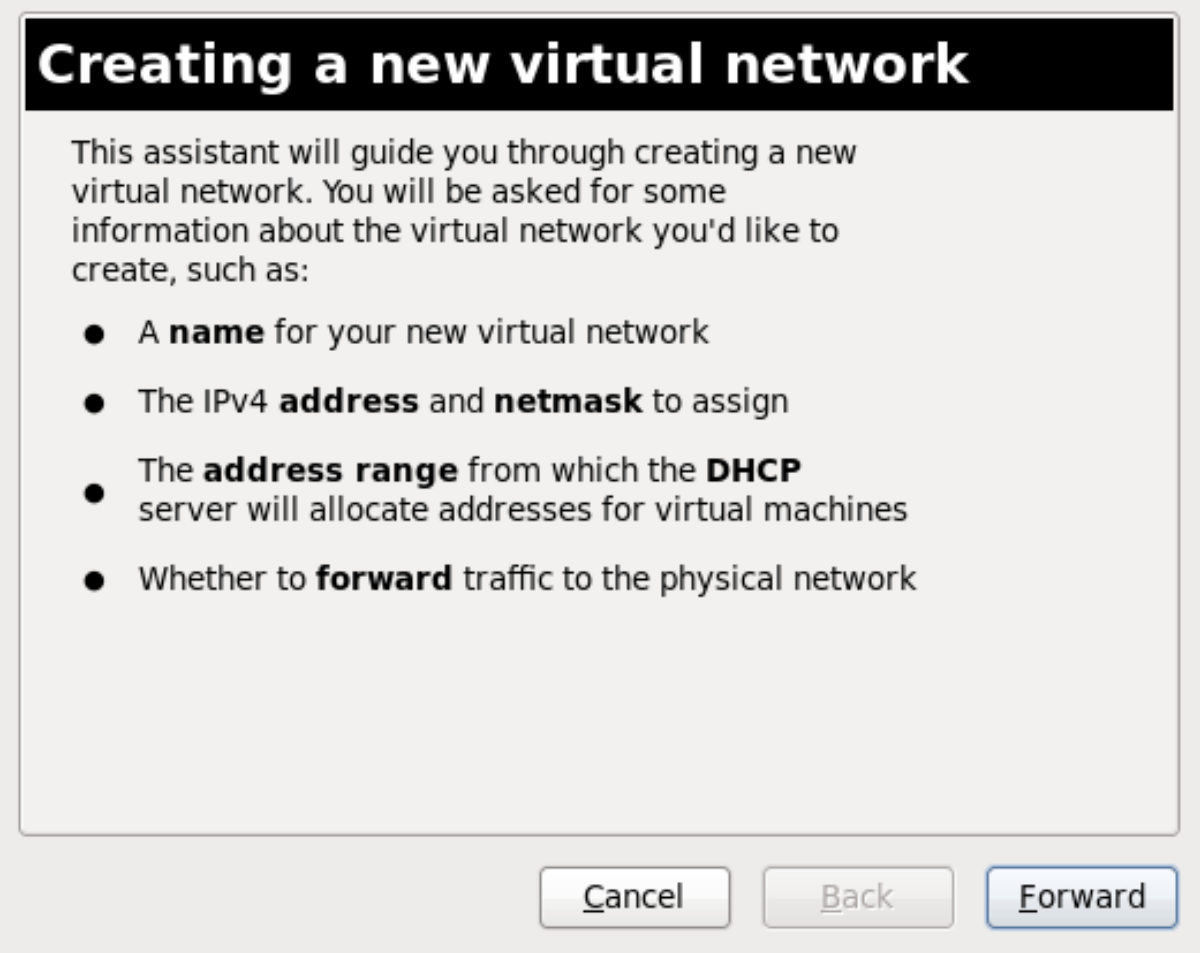
Connection Details *(연결 세부 정보) 메뉴 내에서 가상 네트워크 탭을 엽니다. 더하기 기호 (+) 아이콘으로 식별되는 네트워크 추가 버튼을 클릭합니다. 자세한 내용은* ***18.8절. "가상 네트워크 관리"*** *에서 참조하십시오.*

그림 ***18.12.*** *가상 네트워크 구성*



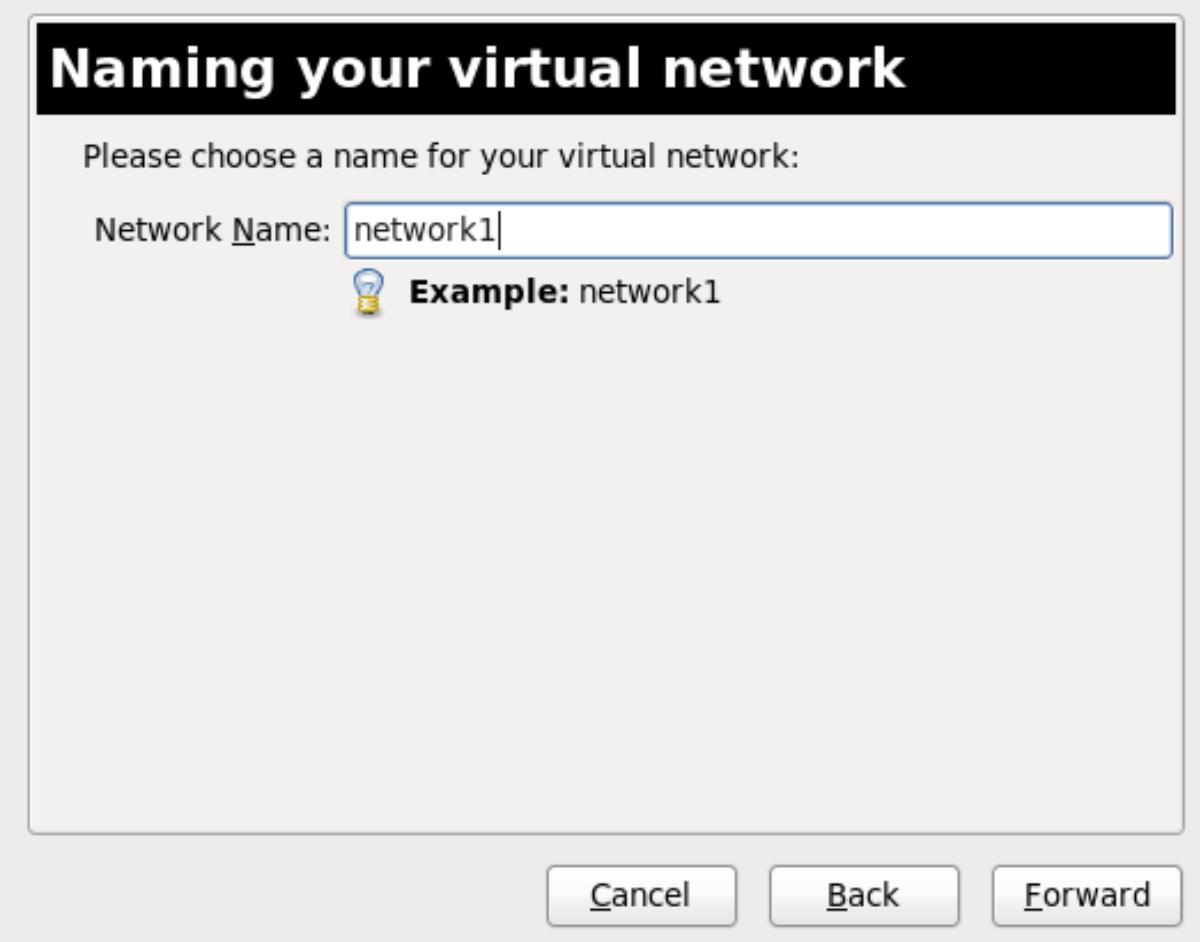
그러면 새 가상 네트워크 생성 창이 열립니다. 계속하려면 ***Forward*** *를 클릭합니다.*

그림 18.13. 새 가상 네트워크 생성



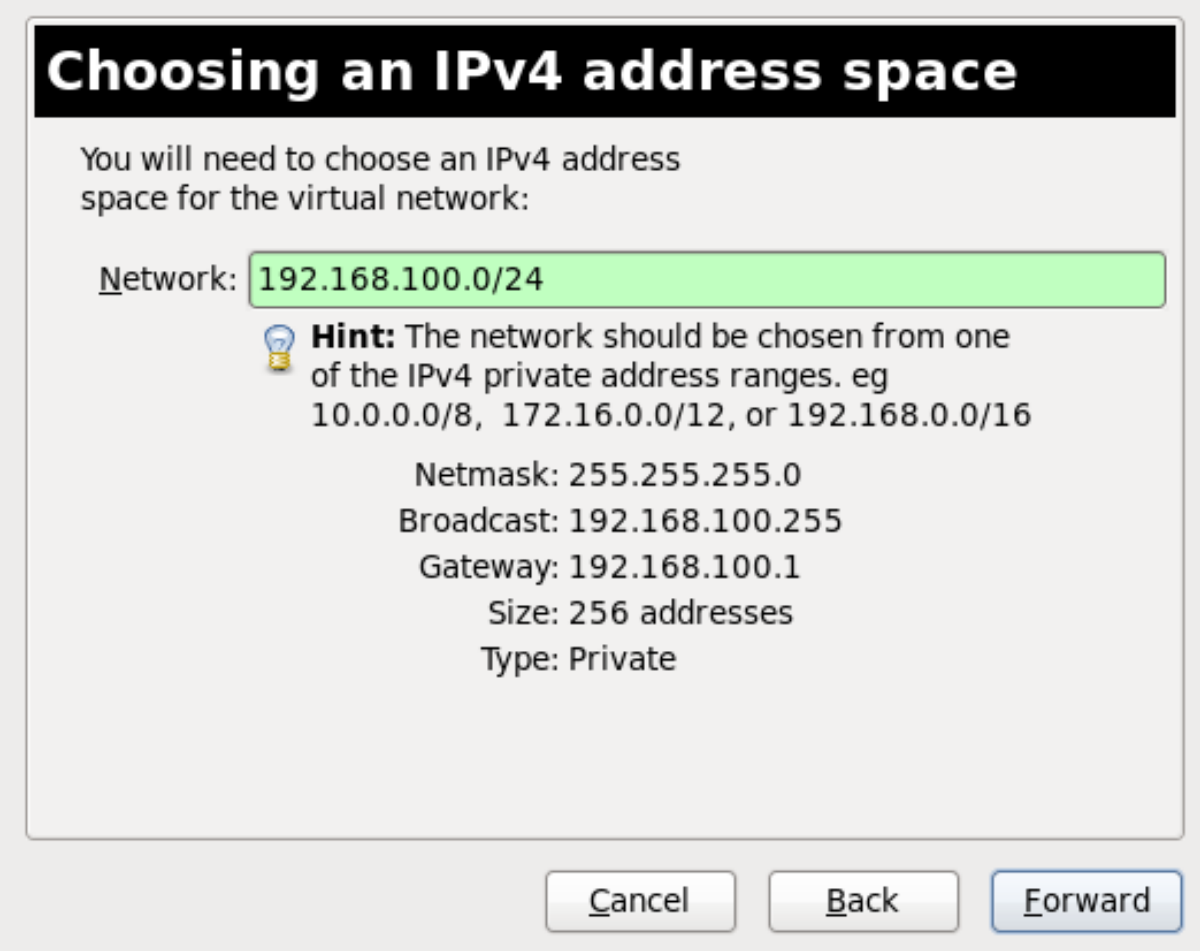
2. *가상 네트워크에 적절한 이름을 입력하고 **Forward** 를 클릭합니다.*

그림 18.14. 가상 네트워크 이름 지정



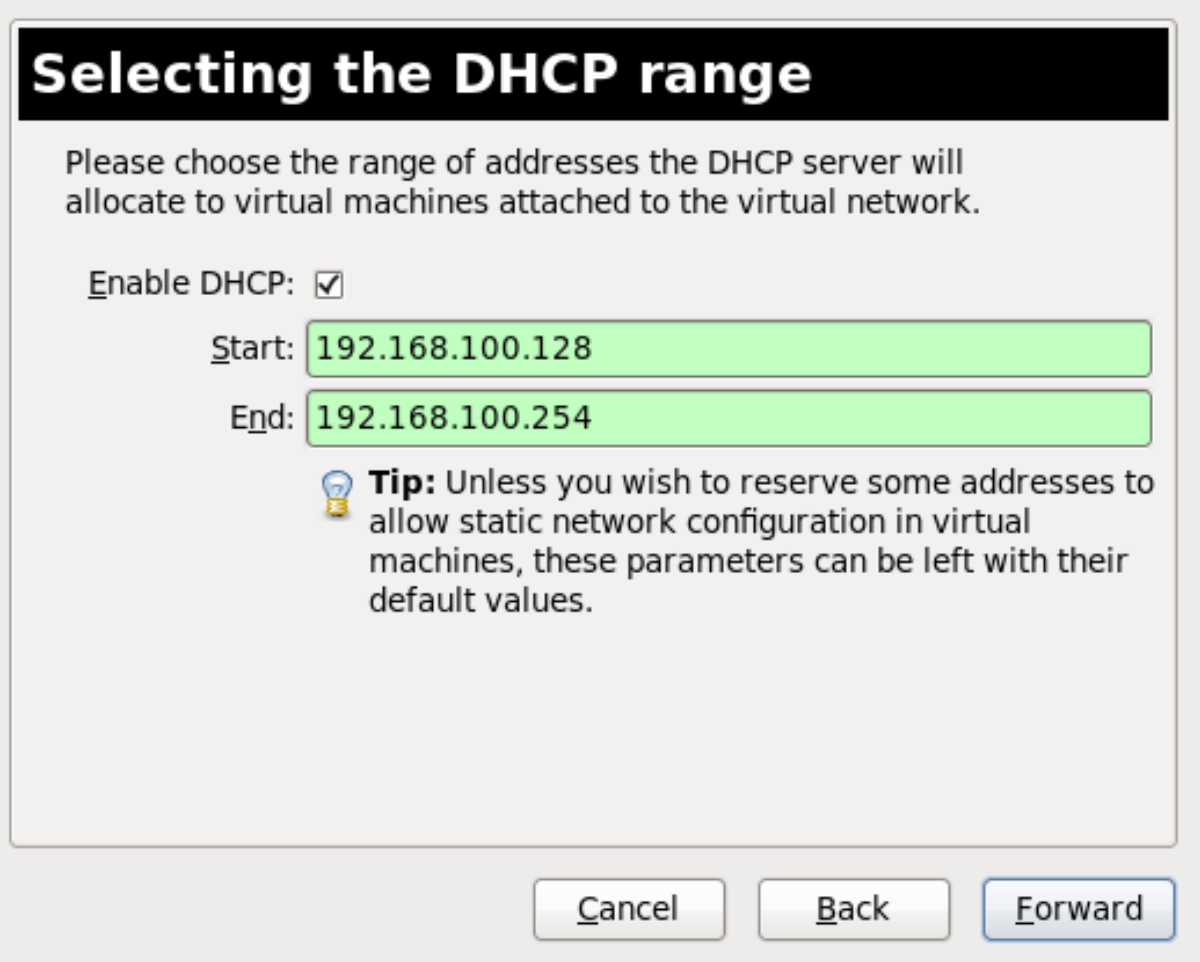
3. *가상 네트워크의 **IPv4** 주소 공간을 입력하고 **Forward** 를 클릭합니다.*

그림 18.15. IPv4 주소 공간 선택



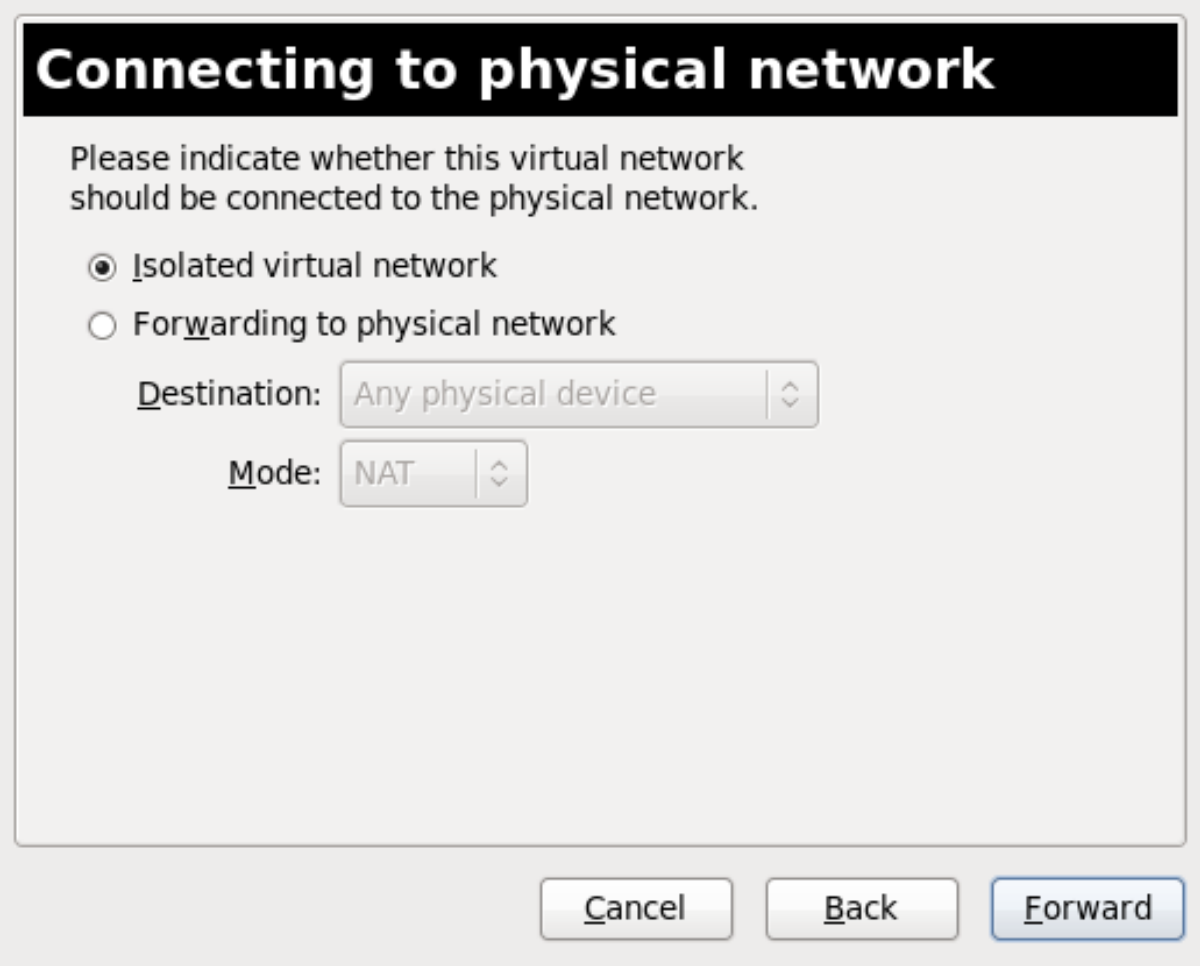
4. *시작 및 종료 **IP** 주소를 지정하여 가상 네트워크의 **DHCP** 범위를 정의합니다. 계속하려면 **Forward** 를 클릭합니다.*

그림 18.16. DHCP 범위 선택



5. *가상 네트워크가 물리적 네트워크에 연결하는 방법을 선택합니다.*

그림 18.17. 물리적 네트워크에 연결

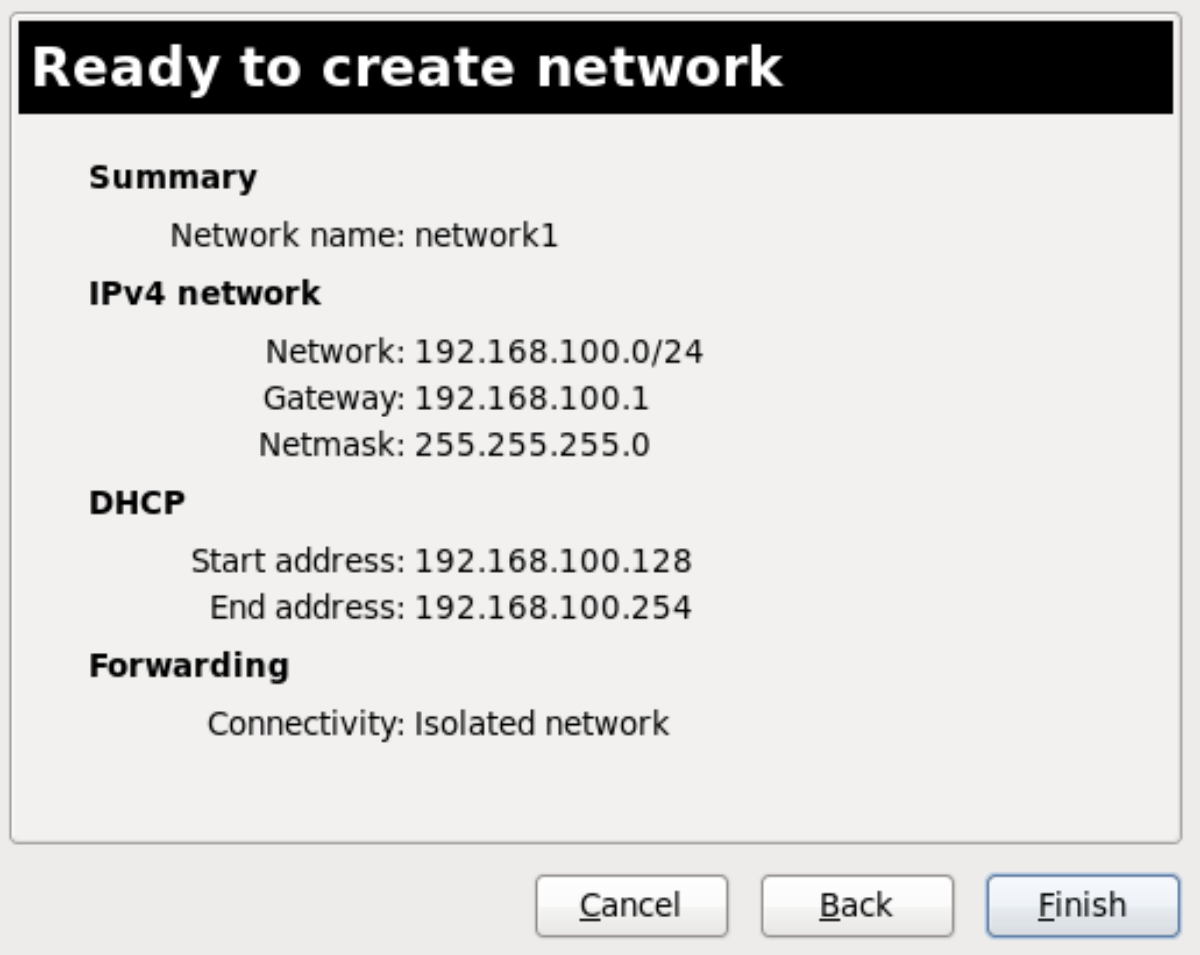


*물리적 네트워크로의 전달을 선택하는 경우 목적지 가 모든 물리적 장치인지 특정 물리적 장치 여야 하는지 선택합니다. 모드가 **NAT** 또는 라우트 여야 하는지 여부도 선택합니다.*

*계속하려면 **Forward** 를 클릭합니다.*

6. *이제 네트워크를 만들 준비가 되었습니다. 네트워크의 구성을 확인하고 **Finish** 를 클릭합니다.*

그림 18.18. 네트워크 생성 준비



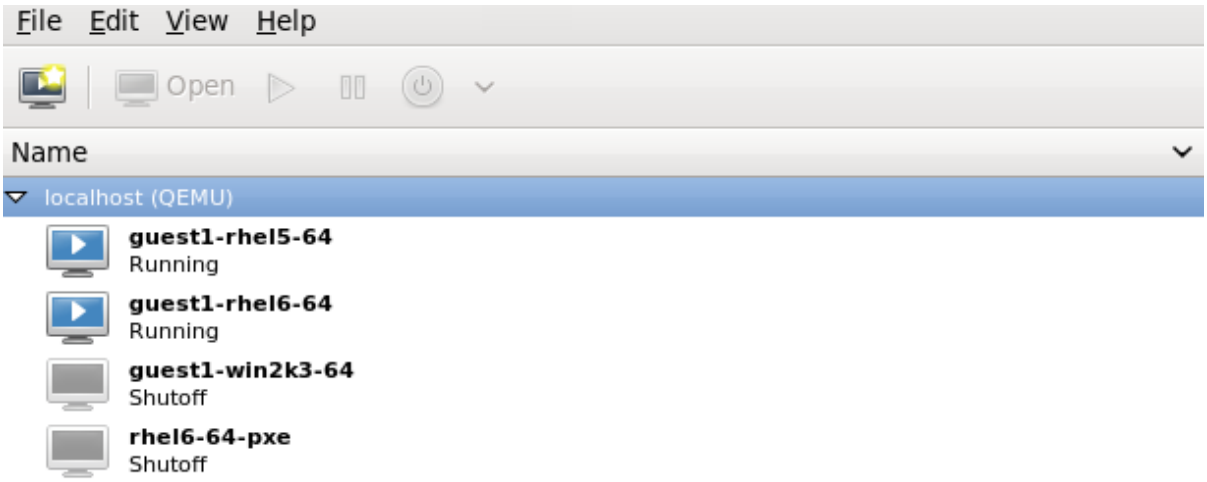
7. *이제 연결 세부 정보 창의 **Virtual Networks** (가상 네트워크) 탭에서 새 가상 네트워크를 사용할 수 있습니다.*

18.10. 가상 네트워크를 게스트에 연결

가상 네트워크를 게스트에 연결하려면 다음을 수행합니다.

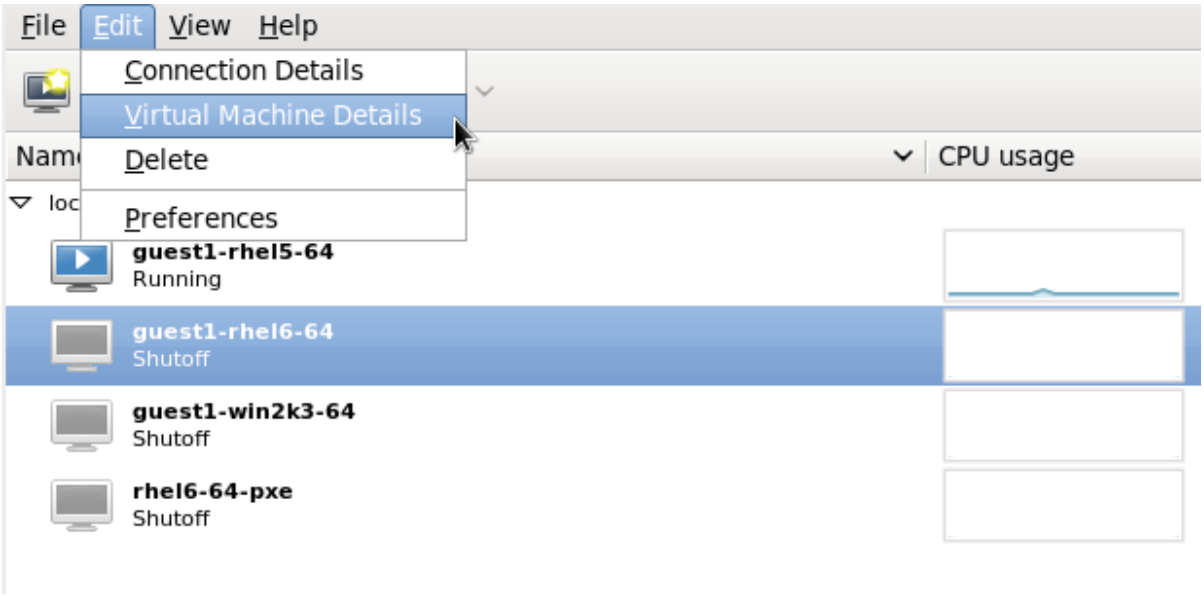
1. ***Virtual Machine Manager** 창에서 네트워크가 할당될 게스트를 강조 표시합니다.*

그림 18.19. 표시할 가상 머신 선택



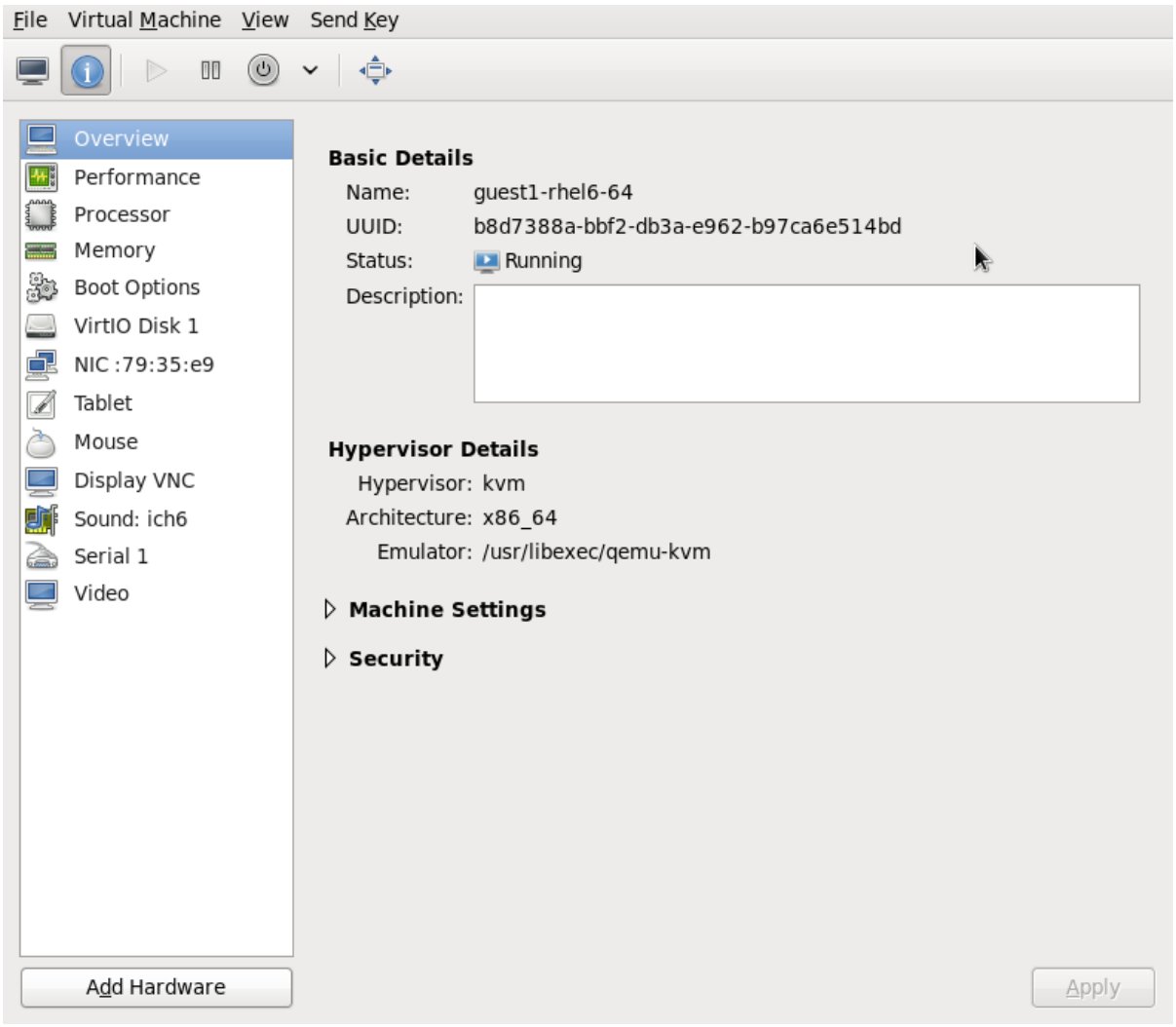
2. *가상 머신 관리자 편집 메뉴에서 가상 머신 세부 정보를 선택합니다.*

그림 18.20. 가상 머신 세부 정보 표시



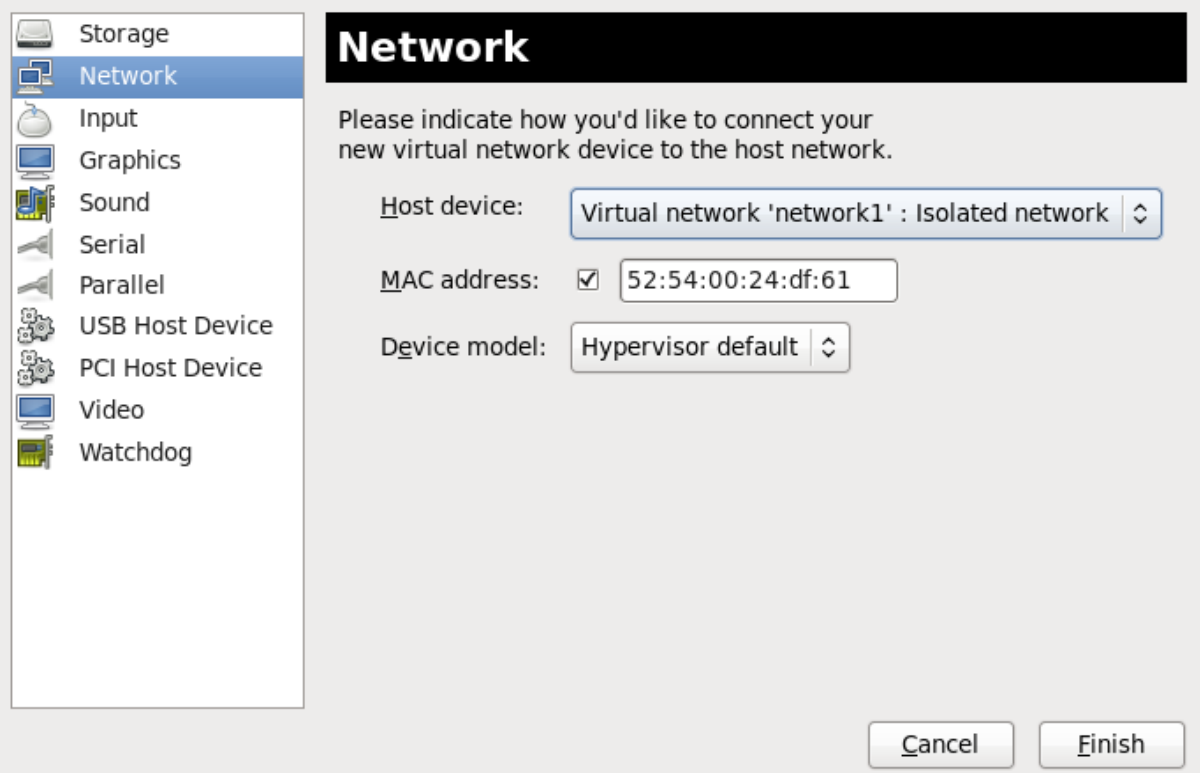
3. *가상 머신 세부 정보 창에서 하드웨어 추가 버튼을 클릭합니다.*

그림 18.21. 가상 머신 세부 정보 창



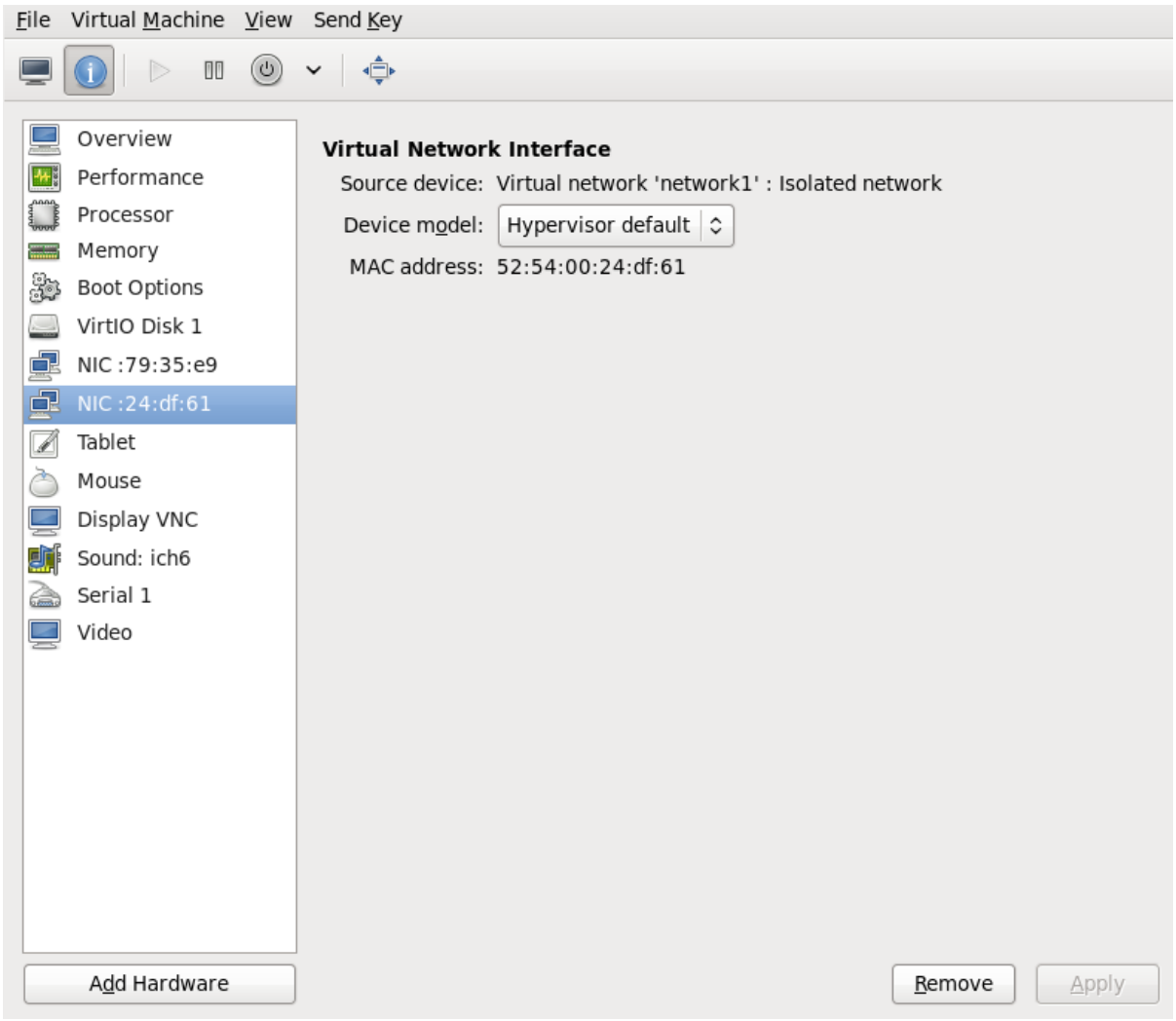
4. *새 가상 하드웨어 추가 창의 왼쪽 창에서 **Network** 를 선택하고 호스트 장치 메뉴에서 네트워크 이름(이 예에서는**network 1**)을 선택하고 **Finish** 를 클릭합니다.*

그림 18.22. 새 가상 하드웨어 추가 창에서 네트워크를 선택합니다.



5. *시작 시 새 네트워크가 게스트에 표시될 가상 네트워크 인터페이스로 표시됩니다.*

그림 18.23. 게스트 하드웨어 목록에 표시된 새 네트워크



18.11. 물리적 인터페이스에 가상 NIC 직접 연결

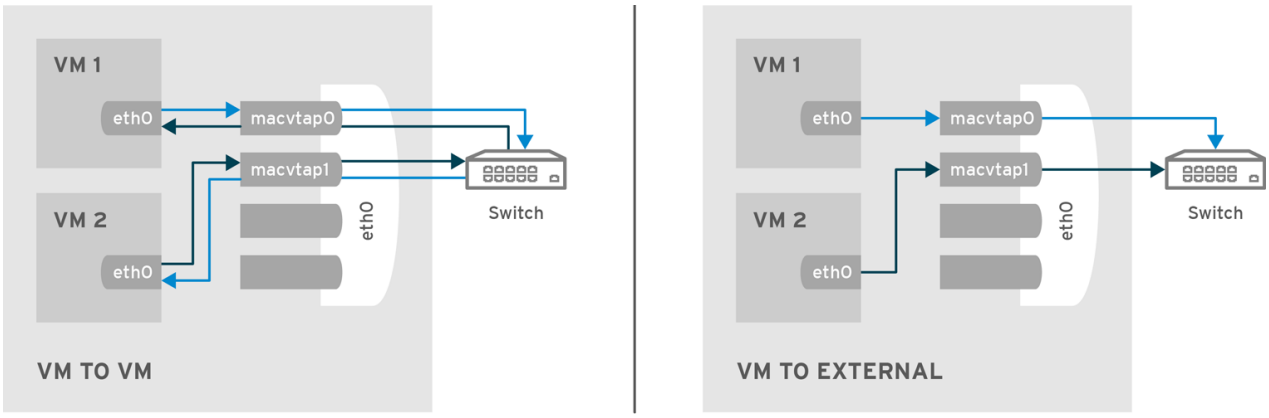
*기본 **NAT** 연결 대신 **macvtap** 드라이버를 사용하여 게스트 **NIC**를 호스트 시스템의 지정된 물리적 인터페이스에 직접 연결할 수 있습니다. 이는 장치 할당 **(passthrough**이라고도 함)과 혼동되지 않습니다. **MacVTap** 연결에는 다음과 같은 모드가 있으며, 각각 다른 이점과 사용 사례가 있습니다.*

물리적 인터페이스 제공 모드

VEPA

*가상 이더넷 포트 수집기(**VEPA**) 모드에서는 게스트의 모든 패킷이 외부 스위치로 전송됩니다. 이를 통해 사용자는 스위치를 통해 게스트 트래픽을 강제 적용할 수 있습니다. **VEPA** 모드가 제대로 작동하려면 외부 스위치가 **hairpin** 모드도 지원해야 합니다. 이 모드를 사용하면 대상이 대상인 패킷이 소스 게스트와 동일한 호스트 시스템의 게스트인 패킷이 외부 스위치에 의해 호스트로 다시 전송되도록 합니다.*

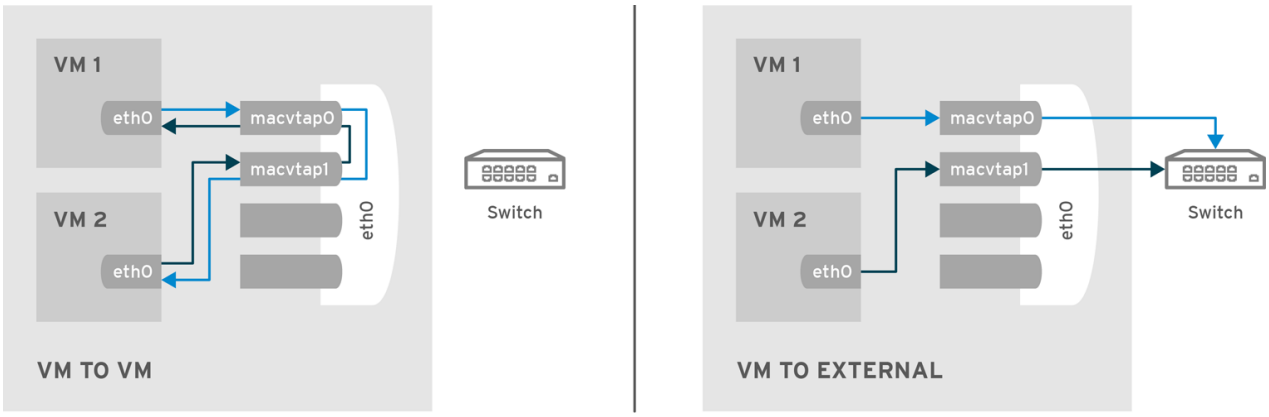
그림 18.24. VEPA 모드



브릿지

*소스 게스트와 대상이 동일한 호스트 시스템에 있는 패킷은 대상 **macvtap** 장치로 직접 전달됩니다. 직접 전달이 성공하려면 소스 장치와 대상 장치 둘 다 브리지 모드에 있어야 합니다. 장치 중 하나가 **VEPA** 모드에 있는 경우 헤어핀 외부 스위치가 필요합니다.*

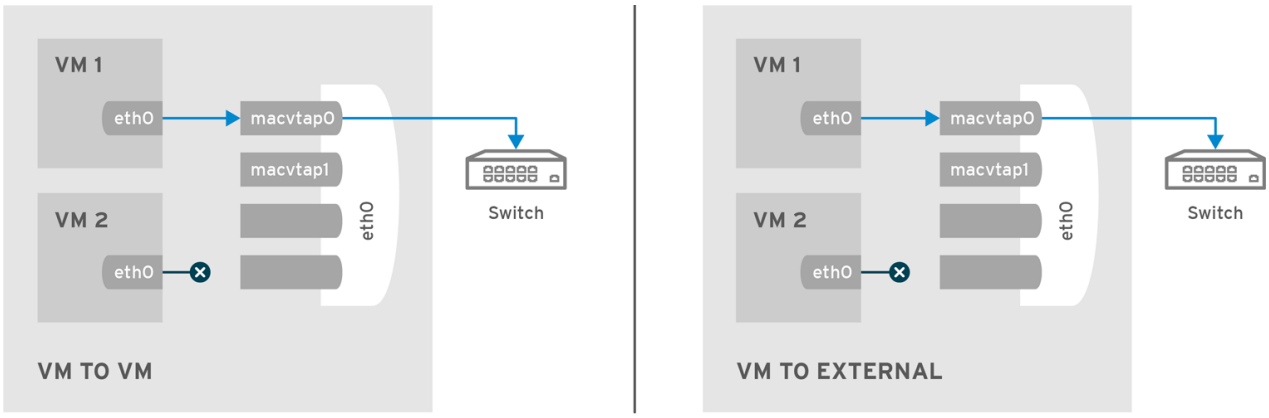
그림 18.25. 브릿지 모드



비공개

모든 패킷은 외부 스위치로 전송되며 외부 라우터 또는 게이트웨이를 통해 전송되는 경우에만 동일한 호스트 시스템의 대상 게스트로 전달되고 이를 호스트로 다시 보냅니다. 개인 모드는 단일 호스트의 개별 게스트가 서로 통신하지 못하도록 방지하는 데 사용할 수 있습니다. 소스 또는 대상 장치가 개인 모드인 경우 이 절차를 따릅니다.

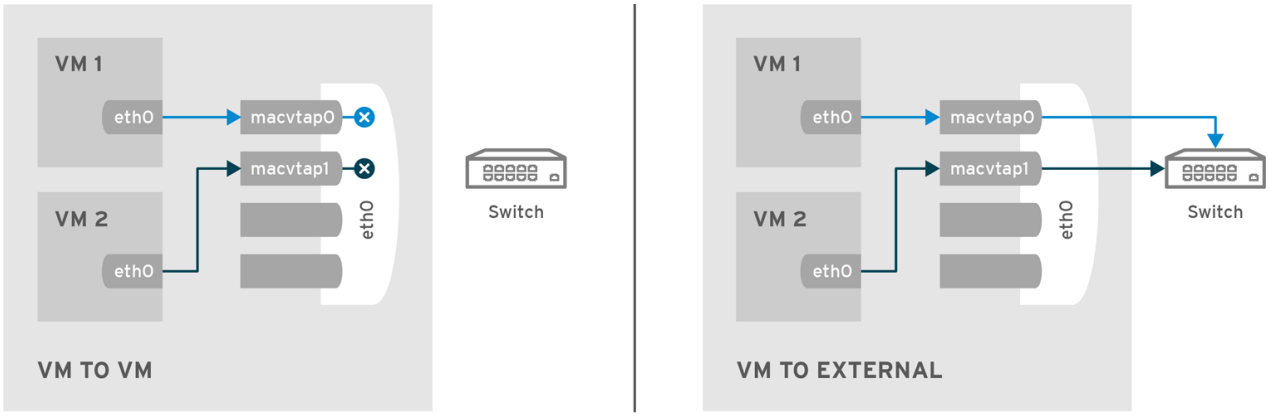
그림 18.26. 비공개 모드



passthrough

*이 기능은 마이그레이션 기능을 손실하지 않고 물리적 인터페이스 장치 또는 **SR-IOV** **VF**(가상 기능)를 게스트에 직접 연결합니다. 모든 패킷은 지정된 네트워크 장치로 직접 전송됩니다. 네트워크 장치는 통과 모드의 게스트 간에 공유할 수 없으므로 단일 네트워크 장치를 단일 게스트로만 전달할 수 있습니다.*

*그림 18.27. **Passthrough** 모드*



*네 가지 모드 각각 도메인 **xml** 파일을 변경하여 구성합니다. 이 파일이 열리면 다음과 같이 모드 설정을 변경합니다.*

```
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='vepa'/>
  </interface>
</devices>
```

직접 연결된 게스트 가상 시스템의 네트워크 액세스는 호스트 물리적 시스템의 물리적 인터페이스가 연결된 하드웨어 스위치로 관리할 수 있습니다.

*스위치가 **IEEE 802.1Qbg** 표준을 준수하는 경우 인터페이스에 다음과 같이 추가 매개 변수가 있을 수 있습니다. **virtualport** 요소의 매개변수는 **IEEE 802.1Qbg** 표준에 자세히 설명되어 있습니다. 값은 네트워크에 따라 다르며 네트워크 관리자가 제공해야 합니다. **802.1Qbg** 측면에서 **VSI**(가상 스테이션 인터페이스)는 가상 머신의 가상 인터페이스를 나타냅니다.*

***IEEE 802.1Qbg**에는 **VLAN ID**에 대해 **0**이 아닌 값이 필요합니다. 스위치가 **IEEE 802.1Qbh** 표준을 준수하는 경우에도 값은 네트워크에 고유하며 네트워크 관리자가 제공해야 합니다.*

가상 스테이션 인터페이스 유형

managerid

***VSI Manager ID**는 **VSI** 유형 및 인스턴스 정의가 포함된 데이터베이스를 식별합니다. 이는 정수 값이며 값 **0**이 예약됩니다.*

typeid

***VSI** 유형 **ID**는 네트워크 액세스를 나타내는 **VSI** 유형을 식별합니다. **VSI** 유형은 일반적으로 네트워크 관리자가 관리합니다. 정수 값입니다.*

typeidversion

***VSI** 유형 버전에서는 여러 버전의 **VSI** 유형을 사용할 수 있습니다. 정수 값입니다.*

InstanceID

***VSI** 인스턴스 **ID** 식별자는 **VSI** 인스턴스(가상 시스템의 가상 인터페이스)가 생성될 때 생성됩니다. 이는 전역적으로 고유한 식별자입니다.*

profileid

*프로필 **ID**에는 이 인터페이스에 적용할 포트 프로필의 이름이 포함되어 있습니다. 이 이름은 **port** 프로필 데이터베이스에서 포트 프로필에서 네트워크 매개 변수로 확인하며 해당 네트워크 매개 변수가 이 인터페이스에 적용됩니다.*

*네 가지 유형 각각은 도메인 **xml** 파일을 변경하여 구성합니다. 이 파일이 열리면 다음과 같이 모드 설정을 변경합니다.*

```
<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>
```

프로필 **ID**가 여기에 표시됩니다.

```
 <devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private'/>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance'/>
    </virtualport>
  </interface>
 </devices>
 ...
```

18.12. 네트워크 필터링 적용

이 섹션에서는 **libvirt**의 네트워크 필터, 해당 목표, 개념 및 **XML** 형식에 대한 소개를 제공합니다.

18.12.1. 소개

네트워크 필터링의 목표는 가상화 시스템의 관리자가 가상 시스템에서 네트워크 트래픽 필터링 규칙을 설정 및 적용하고 가상 머신이 보내거나 받을 수 있는 네트워크 트래픽의 매개 변수를 관리할 수 있도록 하는 것입니다. 가상 시스템이 시작될 때 네트워크 트래픽 필터링 규칙이 호스트 실제 시스템에 적용됩니다. 필터링 규칙을 가상 머신 내에서 우회할 수 없기 때문에 가상 머신 사용자의 관점에서 의무화됩니다.

게스트 가상 머신의 관점에서 네트워크 필터링 시스템을 사용하면 각 가상 시스템의 네트워크 트래픽 필터링 규칙을 인터페이스별로 개별적으로 구성할 수 있습니다. 이러한 규칙은 가상 시스템이 시작될 때 호스트 물리적 시스템에 적용되며 가상 시스템이 실행되는 동안 수정할 수 있습니다. 후자는 네트워크 필터의 **XML** 설명을 수정하여 수행할 수 있습니다.

여러 가상 머신은 동일한 일반 네트워크 필터를 사용할 수 있습니다. 이러한 필터를 수정하면 이 필터를 참조하는 실행 중인 모든 가상 시스템의 네트워크 트래픽 필터링 규칙이 업데이트됩니다. 실행 중이 아닌 머신이 시작 시 업데이트됩니다.

앞에서 언급했듯이 특정 유형의 네트워크 구성에 대해 구성된 개별 네트워크 인터페이스에서 네트워크 트래픽 필터링 규칙을 적용할 수 있습니다. 지원되는 네트워크 유형은 다음과 같습니다.

- **network**
- 이더넷 - 브리징 모드에서 사용해야 합니다.
- 브릿지

예 18.1. 네트워크 필터링의 예

인터페이스 **XML**은 최상위 필터를 참조하는 데 사용됩니다. 다음 예제에서 인터페이스 설명은 필터 정리를 참조합니다.

```
<devices>
 <interface type='bridge'>
  <mac address='00:16:3e:5d:c7:9e'/>
  <filterref filter='clean-traffic'/>
 </interface>
</devices>
```

네트워크 필터는 **XML**로 작성되며 다른 필터에 대한 참조, 트래픽 필터링 규칙 또는 둘 다의 조합을 포함할 수 있습니다. 위의 참조된 필터 정리**-traffic**는 다른 필터에 대한 참조만 포함하고 실제 필터링 규칙을 포함하지 않는 필터입니다. 다른 필터에 대한 참조를 사용할 수 있으므로 필터 트리를 빌드할 수 있습니다. **clean-traffic** 필터는 명령을 사용하여 볼 수 있습니다. **# virsh nwfilter-dumpxml clean-traffic.**

앞에서 언급했듯이 여러 가상 머신에서 단일 네트워크 필터를 참조할 수 있습니다. 일반적으로 인터페이스에는 해당 트래픽 필터링 규칙과 연관된 개별 매개 변수가 있으므로 필터 **XML**에 설명된 규칙을 변수를 사용하여 일반화할 수 있습니다. 이 경우 변수 이름은 필터 **XML**에서 사용되며 이름 및 값은 필터를 참조하는 위치에 제공됩니다.

예 18.2. 설명 연장

다음 예에서 인터페이스 설명은 매개 변수 **IP** 및 점으로 **IP** 주소가 값으로 확장되었습니다.

```
<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>
```

*이 특별한 예에서, **clean-traffic** 네트워크 트래픽 필터는 **IP** 주소 매개변수 **10.0.0.1**으로 표시되고 규칙에 따라 이 인터페이스의 모든 트래픽이 항상 소스 **IP** 주소로 **10.0.0.1**을 사용하고 있으며, 이 필터의 용도 중 하나인 소스 **IP** 주소로 **10.0.0.1**을 사용합니다.*

18.12.2. 필터링 체인

*필터링 규칙은 필터 체인에서 구성됩니다. 이러한 체인은 패킷 필터링 규칙을 개별 체인(**branches**)의 항목으로 사용하는 트리 구조로 간주할 수 있습니다.*

패킷은 루트 체인에서 필터 평가를 시작한 다음 다른 체인에서 평가를 계속하거나 해당 체인에서 다시 루트 체인으로 반환하거나 트래버스 체인 중 하나에서 필터링 규칙에 의해 삭제 또는 수락할 수 있습니다.

***libvirt**의 네트워크 필터링 시스템은 사용자가 트래픽 필터링을 활성화하도록 선택하는 모든 가상 시스템의 네트워크 인터페이스에 대해 개별 루트 체인을 자동으로 생성합니다. 사용자는 루트 체인에서 직접 인스턴스화되는 필터링 규칙을 작성하거나 프로토콜별 규칙의 효율적인 평가를 위해 프로토콜별 필터링 체인을 생성할 수 있습니다.*

다음과 같은 체인이 있습니다.

- *루트*
- ***mac***
- ***STP(범위 트리 프로토콜)***
- ***vlan***
- ***ARP 및 rarp***

- ***ipv4***

- ***ipv6***

mac, stp, vlan, arp, rarp, ipv4, *또는* ***ipv6*** *프로토콜을 평가하는 여러 체인은 체인 이름의 접두사로만 프로토콜 이름을 사용하여 생성할 수 있습니다.*

예 18.3. ARP 트래픽 필터링

이 예제에서는 이름이 ***arp-xyz*** *또는* ***arp-test****인 체인을 지정할 수 있으며 해당 체인에서* ***ARP*** *프로토콜 패킷을 평가하도록 합니다.*

다음 필터 ***XML****은* ***arp*** *체인에서* ***ARP*** *트래픽을 필터링하는 예를 보여줍니다.*

```
<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>
```

예를 들어 루트 체인에서 아닌 ***ARP*** *관련 규칙을* ***arp*** *체인에 배치한 결과* ***ARP*** *이외의 패킷 프로토콜을* ***ARP*** *프로토콜별 규칙으로 평가할 필요가 없습니다. 이렇게 하면 트래픽 필터링의 효율성이 향상됩니다. 그러나 다른 규칙이 평가되지 않으므로 지정된 프로토콜에 대한 필터링 규칙만 체인에 배치하*

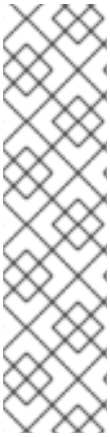
> 는 데 주의를 기울여야 합니다. 예를 들어 **IPv4** 프로토콜 패킷이 **ARP** 체인을 통과하지 않기 때문에 **IPv4** 규칙이 **ARP** 체인에서 평가되지 않습니다.

18.12.3. 체인 우선순위 필터링

앞서 언급한 것처럼 필터링 규칙을 생성할 때 모든 체인이 루트 체인에 연결됩니다. 이러한 체인에 액세스하는 순서는 체인의 우선 순위에 영향을 미칩니다. 다음 표는 우선 순위와 기본 우선순위를 할당할 수 있는 체인을 보여줍니다.

표 18.1. 기본 우선순위 값 필터링

체인(prefix)	기본 우선순위
stp	-810
mac	-800
vlan	-750
ipv4	-700
ipv6	-600
ARP	-500
RARP	-400



참고

우선순위 값이 더 낮은 체인에 액세스한 후 값이 더 높습니다.

표 18.1. "기본 우선순위 값 필터링"에 나열된 체인은 필터 노드의 우선 순위(**XML**) 속성에 **[-1000**에서 **1000]** 범위의 값을 작성하여 사용자 정의 우선 순위를 지정할 수도 있습니다. 18.12.2절. "필터링 체인"**filter**는 예를 들어 **arp** 체인의 경우 **-500**의 기본 우선 순위를 표시합니다.

18.12.4. 필터에서 변수 사용

네트워크 트래픽 필터링 하위 시스템에서 사용하기 위해 **MAC** 및 **IP**의 두 가지 변수가 있습니다.

MAC 은 네트워크 인터페이스의 MAC 주소로 지정됩니다. 이 변수를 참조하는 필터링 규칙은 인터페이스의 MAC 주소로 자동으로 교체됩니다. 사용자가 MAC 매개 변수를 명시적으로 제공하지 않아도 됩니다. 위의 IP 매개변수와 유사한 MAC 매개 변수를 지정할 수는 있지만 libvirt에서 인터페이스에서 사용할 MAC 주소를 알고 있으므로 권장되지 않습니다.

매개 변수 IP 는 지정된 인터페이스에서 가상 머신 내부의 운영 체제가 사용할 것으로 예상되는 IP 주소를 나타냅니다. libvirt 데몬이 매개 변수가 명시적으로 제공되지 않고 참조되는 경우 인터페이스에서 사용 중인 IP 주소(및 IP 매개변수 값)를 확인하려고 하면 IP 매개변수가 특수하게 됩니다. IP 주소 감지에 대한 현재 제한 사항은 이 기능을 사용하는 방법과 사용할 때 예상해야 하는 사항에 대한 제한 18.12.12절. "제한 사항" 섹션을 참조하십시오. 18.12.2절. "필터링 체인" 에 표시된 XML 파일에는 MAC 및 IP 변수를 참조하는 데 네트워크 필터 XML을 사용하는 예입니다.

참조된 변수 앞에는 항상 $ 문자 접두사가 지정됩니다. 변수 값 형식은 XML에서 식별되는 필터 특성에 따라 예상되는 유형이어야 합니다. 위의 예에서 IP 매개변수는 표준 형식으로 법률 IP 주소를 보유해야 합니다. 올바른 구조를 제공하지 않으면 필터 변수가 값으로 교체되지 않아 핫 플러그를 사용할 때 가상 머신이 시작되지 않거나 인터페이스가 연결되지 않습니다. 각 XML 특성에 예상되는 일부 유형은 예제 예 18.4. "샘플 변수 유형" 에 표시됩니다.

예 18.4. 샘플 변수 유형

변수에 요소 목록이 포함될 수 있으므로 (예: 변수 IP는 특정 인터페이스에서 유효한 여러 IP 주소를 포함할 수 있음) IP 변수에 대한 여러 요소를 제공하는 표기법은 다음과 같습니다.

```
<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>
```

이 XML 파일은 필터를 생성하여 인터페이스당 여러 IP 주소를 활성화합니다. 각 IP 주소에는 별도의 필터링 규칙이 생성됩니다. 따라서 위의 XML과 다음 규칙을 사용하면 세 개의 개별 필터링 규칙(각 IP 주소에 하나씩)이 생성됩니다.

```
<rule action='accept' direction='in' priority='500'>
  <tcp srpipaddr='$IP'/>
</rule>
```

요소 목록을 포함하는 변수의 개별 요소에 액세스할 수 있으므로 다음과 같은 필터링 규칙은 변수 DSTPORTS의 두 번째 요소에 액세스합니다.As it is possible to access individual elements of a

variable holding a list of elements, a filtering rule like the following accesses the 2nd element of the variable DSTPORTS.

```
<rule action='accept' direction='in' priority='500'>
  <udp dstportstart='$DSTPORTS[1]'/>
</rule>
```

*예 **18.5.** 다양한 변수 사용*

*표기법 **$VARIABLE[@<iterator id="x">]** 을 사용하여 다양한 목록에서 가능한 모든 규칙 조합을 나타내는 필터링 규칙을 생성할 수 있습니다. 다음 규칙을 사용하면 가상 머신이 **SRCIPADDRESSES** 에 지정된 소스 **IP** 주소 집합에서 **DSTPORTS** 에 지정된 포트 집합에 대한 트래픽을 수신할 수 있습니다. 이 규칙은 두 개의 독립 이터레이터를 사용하여 해당 요소에 액세스하여 변수 **DSTPORTS** 의 모든 조합을 **SRCIPADDRESSES** 와 함께 생성합니다.*

```
<rule action='accept' direction='in' priority='500'>
  <ip srcipaddr='$SRCIPADDRESSES[@1]' dstportstart='$DSTPORTS[@2]'/>
</rule>
```

*다음과 같이 **SRCIPADDRESSES** 및 **DSTPORTS** 에 구체적인 값을 할당합니다.*

```
SRCIPADDRESSES = [ 10.0.0.1, 11.1.2.3 ]
DSTPORTS = [ 80, 8080 ]
```

***$SRCIPADDRESSES[@1]** 및 **$DSTPORTS[@2]** 를 사용하여 변수에 값을 할당하면 다음과 같이 모든 주소와 포트 조합이 생성됩니다.*

- ***10.0.0.1, 80***
- ***10.0.0.1, 8080***
- ***11.1.2.3, 80***
- ***11.1.2.3, 8080***

*예를 들어 **$SRCIPADDRESSES[@1]** 및 **$DSTPORTS[@1]** 표기법을 사용하여 동일한 변수에 액세스하면 목록 모두에 대한 병렬 액세스가 발생하고 다음과 같은 조합이 발생합니다.*

- *10.0.0.1, 80*
- *11.1.2.3, 8080*

참고

$VARIABLE 은 $VARIABLE[@0] 의 약어입니다. 이전 표기법은 항상 이 섹션의 맨 위에 있는 여는 단락에 표시된 대로 반복기 id="0"을 사용하여 반복 자의 역할을 가정합니다.

18.12.5. 자동 IP 주소 탐지 및 DHCP 스누핑

이 섹션에서는 자동 IP 주소 탐지 및 DHCP 스누핑에 대한 정보를 제공합니다.

18.12.5.1. 소개

변수 IP가 참조되지만 값이 할당되지 않은 경우 가상 머신의 인터페이스에 사용되는 IP 주소 검색이 자동으로 활성화됩니다. 변수 CTRL_IP_LEARNING 을 사용하여 사용할 IP 주소 학습 방법을 지정할 수 있습니다. 유효한 값에는 모든,dhcp, 또는 none 이 있습니다.

값 any 는 libvirt에서 모든 패킷을 사용하여 가상 시스템에서 사용 중인 주소를 결정하도록 지시합니다. 이는 TRL_IP_LEARNING 변수가 설정되지 않은 경우 기본 설정입니다. 이 메서드는 인터페이스당 단일 IP 주소만 감지합니다. 게스트 가상 머신의 IP 주소가 감지되면 해당 IP 네트워크 트래픽이 해당 주소로 잠깁니다. 예를 들어 필터 중 하나로 IP 주소 스푸핑이 차단됩니다. 이 경우 VM의 사용자는 guest 가상 머신 내부의 인터페이스의 IP 주소를 변경할 수 없으므로 IP 주소 스푸핑으로 간주됩니다. 게스트 가상 머신이 다른 호스트 물리적 시스템으로 마이그레이션되거나 일시 중지 작업 후에 다시 시작되면 게스트 가상 시스템에서 보낸 첫 번째 패킷은 게스트 가상 머신이 특정 인터페이스에서 사용할 수 있는 IP 주소를 다시 결정합니다.

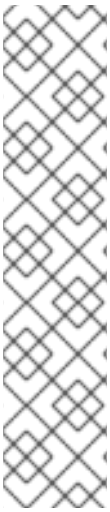
dhcp 값은 libvirt에 유효한 리스가 있는 DHCP 서버 할당 주소만 적용하도록 지시합니다. 이 방법은 인터페이스당 여러 IP 주소 감지 및 사용을 지원합니다. 게스트 가상 머신이 일시 중지 작업 후에 다시 시작되면 유효한 IP 주소 리스가 해당 필터에 적용됩니다. 그렇지 않으면 게스트 가상 머신에서 DHCP를 사용하여 새 IP 주소를 가져와야 합니다. 게스트 가상 머신이 다른 물리적 호스트 물리적 시스템으로 마이그레이션되면 DHCP 프로토콜을 다시 실행하려면 게스트 가상 머신이 필요합니다.

CTRL_IP_LEARNING을 none 으로 설정하면 libvirt에서 IP 주소 학습 및 참조하지 않고 명시적 값을 할당하지 않습니다.

18.12.5.2. DHCP 스누핑

Ctrl_IP_LEARNING=dhcp (DHCP 스nooping)은 특히 신뢰할 수 있는 DHCP 서버만 IP 주소를 할당할 수 있도록 하는 필터와 결합된 경우 추가 방지 방지 보안을 제공합니다. 이를 활성화하려면 변수 DHCPSERVER 를 유효한 DHCP 서버의 IP 주소로 설정하고 이 변수를 사용하여 수신되는 DHCP 응답을 필터링하는 필터를 제공합니다.

DHCP 스누핑이 활성화되고 DHCP 리스가 만료되면 게스트 가상 머신이 DHCP 서버에서 유효한 새 리스를 얻을 때까지 더 이상 IP 주소를 사용할 수 없습니다. 게스트 가상 머신이 마이그레이션되는 경우 IP 주소를 사용하는 데 유효한 새로운 DHCP 리스가 있어야 합니다(예: VM 인터페이스를 작동 중단하고 다시 작동함).



참고

자동 DHCP 탐지는 게스트 가상 머신이 인프라의 DHCP 서버와 교환하는 DHCP 트래픽을 수신 대기합니다. libvirt에 대한 서비스 거부 공격을 방지하기 위해 해당 패킷의 평가는 속도가 제한됩니다. 즉, 인터페이스에서 초당 과도한 수의 DHCP 패킷을 보내는 게스트 가상 머신에는 해당 패킷이 평가되지 않으므로 필터가 조정되지 않을 수 있습니다. 일반 DHCP 클라이언트 동작은 초당 DHCP 패킷 수가 낮은 수를 보내는 것으로 가정합니다. 또한 인프라의 모든 게스트 가상 머신에 적절한 필터를 설정하여 DHCP 패킷을 보낼 수 없도록 하는 것이 중요합니다. 따라서 게스트 가상 머신은 UDP와 TCP 트래픽을 포트 68에서 포트 68로 전송할 수 없도록 해야 하며, 모든 게스트 가상 머신에서 DHCP 서버 메시지를 신뢰할 수 있는 DHCP 서버에서만 허용하도록 제한해야 합니다. 동시에 서브넷의 모든 게스트 가상 시스템에서 거부 방지를 활성화해야 합니다.

예 18.6. DHCP 스누핑을 위한 IP 활성화

다음 XML은 DHCP 스누핑 방법을 사용하여 IP 주소 학습의 활성화에 대한 예를 제공합니다.

```
<interface type='bridge'>
  <source bridge='virbr0'/>
  <filterref filter='clean-traffic'>
    <parameter name='CTRL_IP_LEARNING' value='dhcp'/>
  </filterref>
</interface>
```

18.12.6. 예약된 변수

표 18.2. “예약 변수” libvirt에서 예약 및 사용되는 변수를 표시합니다.

표 18.2. 예약 변수

변수 이름	정의
MAC	인터페이스의 MAC 주소
IP	인터페이스에서 사용하는 IP 주소 목록
IPV6	현재 구현되지 않음: 인터페이스에서 사용 중인 IPV6 주소 목록
DHCPSERVER	신뢰할 수 있는 DHCP 서버의 IP 주소 목록
DHCPSERVERV6	현재 구현되지 않음: 신뢰할 수 있는 DHCP 서버의 IPv6 주소 목록
CTRL_IP_LEARNING	IP 주소 탐지 모드 선택

18.12.7. 요소 및 속성 개요

*모든 네트워크 필터에 필요한 루트 요소의 이름은 두 가지 가능한 속성이 있는 **<filter>** 입니다. **name** 특성은 지정된 필터의 고유 이름을 제공합니다. **chain** 속성은 선택 사항이지만 기본 호스트 물리적 시스템의 방화벽 하위 시스템에서 보다 효율적으로 처리하기 위해 특정 필터를 더 효과적으로 구성할 수 있습니다. 현재 시스템은 루트,**ipv4**,**ipv6**,**arp** 및 **rarp** 등 체인만 지원합니다.*

18.12.8. 기타 필터에 대한 참조

모든 필터는 다른 필터에 대한 참조를 포함할 수 있습니다. 개별 필터는 필터 트리에서 여러 번 참조될 수 있지만 필터 간 참조는 루프를 도입해서는 안 됩니다.

> ***예 18.7. 정리된 트래픽 필터의 예***
>
> *다음은 몇 가지 다른 필터를 참조하는 **clean-traffic** 네트워크 필터의 **XML**을 보여줍니다.*
>
> ```
> <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>
> ```
>
> *다른 필터를 참조하려면 필터 노드 내에서 **XML** 노드 **filterref**를 제공해야 합니다. 이 노드에는 참조할 필터의 이름이 포함된 속성 필터가 있어야 합니다.*

새 네트워크 필터는 언제든지 정의할 수 있으며, 아직 **libvirt**에 알려지지 않은 네트워크 필터에 대한 참조를 포함할 수 있습니다. 그러나 가상 머신이 시작되거나 필터를 참조하는 네트워크 인터페이스가 핫플러그되면 필터 트리의 모든 네트워크 필터를 사용할 수 있어야 합니다. 그렇지 않으면 가상 머신이 시작되지 않거나 네트워크 인터페이스를 연결할 수 없습니다.

18.12.9. 필터 규칙

다음 **XML**은 나가는 **IP** 패킷에서 **IP** 주소(변수 **IP**의 값을 통해 제공)가 예상되지 않아 **VM**이 **IP** 주소를 스푸핑하는 경우 트래픽을 삭제하는 규칙을 구현하는 네트워크 트래픽 필터의 간단한 예를 보여줍니다.

예 18.8. 네트워크 트래픽 필터링의 예

```
<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>
```

트래픽 필터링 규칙은 규칙 노드로 시작됩니다. 이 노드는 다음 특성 중 최대 **3**개를 포함할 수 있습니다.

- 작업은 다음 값을 가질 수 있습니다.
  - 삭제(규칙을 자동으로 일치시키면 추가 분석 없이 패킷을 자동으로 삭제)
  - 거부(규칙을 일치하면 추가 분석 없이 **ICMP** 거부 메시지가 생성됨)
  - 수락(규칙을 더 이상 분석 없이 패킷을 수락)
  - 반환(규칙이 이 필터를 통과하고 추가 분석을 위해 호출 필터에 제어를 반환)
  - 계속(조치 분석을 위한 다음 규칙으로 일치)

- 방향은 다음 값을 가질 수 있습니다.
  - 들어오는 트래픽의 경우
  - 나가는 트래픽의 경우
  - 들어오고 나가는 트래픽의 수신 수신

- **priority**는 선택 사항입니다. 규칙의 우선 순위는 다른 규칙에 따라 규칙을 인스턴스화할 순서를 제어합니다. 값이 낮은 규칙이 더 높은 값이 있는 규칙보다 먼저 인스턴스화됩니다. 유효한 값은 **-1000**에서 **1000**까지의 범위에 있습니다. 이 속성을 제공하지 않으면 우선순위 **500**이 기본적으로 할당됩니다. 루트 체인의 필터링 규칙은 우선 순위에 따라 루트 체인에 연결된 필터와 함께 정렬됩니다. 이를 통해 필터 체인에 대한 액세스 권한이 있는 필터링 규칙을 인터리브할 수 있습니다. 자세한 내용은 18.12.3절. "체인 우선순위 필터링" 를 참조하십시오.

- **statematch**는 선택 사항입니다. 가능한 값은 기본 연결 상태를 끄는 **'0'** 또는 **'false'**입니다. 기본 설정은 **'true'** 또는 **1**입니다.

자세한 내용은 18.12.11절. "고급 필터 구성 주제" 에서 참조하십시오.

위의 예제 예 18.7. "정리된 트래픽 필터의 예" 는 **ip** 유형의 트래픽이 체인 **ipv4** 와 연결되며 규칙에 우선순위**=500** 이 있음을 나타냅니다. 예를 들어 다른 필터에서 **ip** 유형의 트래픽도 체인 **ipv4** 와 연결되어 있는 경우 해당 필터의 규칙이 표시된 규칙의 **priority=500** 에 따라 정렬됩니다.

규칙에는 트래픽 필터링을 위한 단일 규칙이 포함될 수 있습니다. 위 예제에서는 **ip** 유형의 트래픽이 필터링되는 것을 보여줍니다.

18.12.10. 지원되는 프로토콜

다음 섹션에서는 네트워크 필터링 하위 시스템에서 지원하는 프로토콜에 대한 세부 정보를 나열하고 제공합니다. 이러한 트래픽 유형은 규칙 노드에 중첩된 노드로 제공됩니다. 규칙에서 필터링하는 트래픽 유형에 따라 속성은 다릅니다. 위의 예제에서는 **ip** 트래픽 필터링 노드 내부에서 유효한 단일 특성 **srcipaddr** 을 표시했습니다. 다음 섹션에서는 어떤 속성이 유효한지와 예상되는 데이터 유형을 보여줍니다. 다음 데이터 유형을 사용할 수 있습니다.

- *UINT8 : 8 비트 정수; 0-255 범위*
- *UINT16: 16비트 정수, 범위 0-65535*
- *MAC_ADDR: 00:11:22:33:44:55와 같이 점으로 10진수 형식의 MAC 주소*
- *MAC_MASK: FF:FF:FC:00과 같은 MAC 주소 형식의 MAC 주소 마스크*
- *IP_ADDR: 10.1.2.3과 같은 점으로 10.1.2.3 형식의 IP 주소*
- *IP_MASK: 점으로 지정된 10진수 형식(255.255.248.0) 또는 CIDR 마스크(0-32)의 IP 주소 마스크*
- *IPV6_ADDR: FFFF::1과 같은 번호 형식으로 IPv6 주소*
- *IPV6_MASK: IPv6 mask in numbers format (FFFF:FFFF:FC00::) 또는 CIDR 마스크(0-128)*
- *문자열: 문자열*
- *BOOLEAN: 'true', 'yes', '1' 또는 'false', 'no', '0'*
- *IPSETFLAGS: 최대 6개의 'src' 또는 'dst' 요소에서 패킷 헤더의 소스 또는 대상 부분(예: src,src,dst)에서 기능을 선택하는 ipset의 소스 및 대상 플래그입니다. 여기에서 제공할 'selectors' 수는 참조되는 ipset 유형에 따라 다릅니다.*

IP_MASK 또는 IPV6_MASK 유형의 속성을 제외한 모든 속성은 값이 없는 match 속성을 사용하여 부정할 수 있습니다. 여러 개의 부정 특성을 함께 그룹화할 수 있습니다. 다음 XML 조각은 추상 특성을 사용하는 이러한 예를 보여줍니다.

```
[...]
  <rule action='drop' direction='in'>
    <protocol match='no' attribute1='value1' attribute2='value2'/>
```

```
    <protocol attribute3='value3'/>
  </rule>
[...]
```

*규칙은 규칙과 지정된 프로토콜 속성의 경계 내에서 논리적으로 살펴봅니다. 따라서 단일 속성의 값이 규칙에서 지정된 값과 일치하지 않으면 평가 프로세스 중에 전체 규칙이 건너뜁니다. 따라서 위 예제에서 들어오는 트래픽은 **value 1** 과 **protocol property attribute 2** 가 **value2** 와 일치하지 않고 **protocol** 속성 속성**3**이 **value 3** 과 일치하는 경우에만 삭제됩니다.*

18.12.10.1. MAC (Ethernet)

*프로토콜 **ID: mac***

이 유형의 규칙은 루트 체인으로 이동해야합니다.

*표 **18.3. MAC** 프로토콜 유형*

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcmacmask	MAC_MASK	보낸 사람의 MAC 주소에 적용되는 마스크
dstmacaddr	MAC_ADDR	대상의 MAC 주소
dstmacmask	MAC_MASK	대상의 MAC 주소에 적용되는 마스크
protocolid	UINT16 (0x600-0xffff), STRING	계층 3 프로토콜 ID입니다. 유효한 문자열에는 [arp, rarp, ipv4, ipv6]
주석	문자열	256자까지 텍스트 문자열

필터는 다음과 같이 작성할 수 있습니다.

```
[...]
<mac match='no' srcmacaddr='$MAC'/>
[...]
```

18.12.10.2. VLAN (802.1Q)

프로토콜 ID: vlan

이 유형의 규칙은 root 또는 vlan 체인으로 이동해야 합니다.

표 18.4. VLAN 프로토콜 유형

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcmacmask	MAC_MASK	보낸 사람의 MAC 주소에 적용되는 마스크
dstmacaddr	MAC_ADDR	대상의 MAC 주소
dstmacmask	MAC_MASK	대상의 MAC 주소에 적용되는 마스크
vlan-id	UINT16 (0x0-0xfff, 0 - 4095)	VLAN ID
encap-protocol	UINT16 (0x03c-0xfff), 문자열	캡슐화된 계층 3 프로토콜 ID, 유효한 문자열은 arp, ipv4, ipv6입니다.
주석	문자열	256자까지 텍스트 문자열

18.12.10.3. STP(Spanning Tree Protocol)

프로토콜 ID: stp

이 유형의 규칙은 루트 또는 스트립 체인으로 이동해야합니다.

표 18.5. STP 프로토콜 유형

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcmacmask	MAC_MASK	보낸 사람의 MAC 주소에 적용되는 마스크
type	UINT8	브릿지 프로토콜 데이터 단위 (BICE) 유형

속성 이름	datatype	정의
플래그	UINT8	BECDHE 플래그dstmacmask
root-priority	UINT16	루트 우선 순위 범위 시작
root-priority-hi	UINT16 (0x0-0xfff, 0 - 4095)	루트 우선 순위 범위 종료
root-address	MAC _ADDRESS	루트 MAC 주소
root-address-mask	MAC _MASK	루트 MAC 주소 마스크
로터 비용	UINT32	루트 경로 비용(시작 기준)
root-cost-hi	UINT32	루트 경로 비용 범위 종료
sender-priority-hi	UINT16	보낸 사람 우선 순위 범위 종료
sender-address	MAC_ADDRESS	B92) 발신자 MAC 주소
sender-address-mask	MAC_MASK	B typical sender MAC 주소 마스크
port	UINT16	포트 식별자(범위 시작)
port_hi	UINT16	포트 식별자 범위 종료
msg-age	UINT16	메시지 기간 타이머 (시작 범위)
msg-age-hi	UINT16	메시지 기간 타이머 범위 종료
max-age-hi	UINT16	최대 수명 범위 종료
hello-time	UINT16	hello 시간 타이머(시작 기준)
hello-time-hi	UINT16	hello 시간 타이머 범위 종료
forward-delay	UINT16	전달 지연 (시작 범위)
forward-delay-hi	UINT16	전달 지연 범위 종료
주석	문자열	256자까지 텍스트 문자열

18.12.10.4. ARP/RARP

프로토콜 ID: arp 또는 rarp

*이러한 유형의 규칙은 **root** 또는 **arp/rarp** 체인으로 이동해야 합니다.*

표 18.6. ARP 및 RARP 프로토콜 유형

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcmacmask	MAC_MASK	보낸 사람의 MAC 주소에 적용되는 마스크
dstmacaddr	MAC_ADDR	대상의 MAC 주소
dstmacmask	MAC_MASK	대상의 MAC 주소에 적용되는 마스크
hwtype	UINT16	하드웨어 유형
protocoltype	UINT16	프로토콜 유형
opcode	UINT16, STRING	opcode 유효한 문자열: Request, Reply, Request_Reverse, Reply_Reverse, DRARP_Request, DRARP_Reply, DRARP_Error, InARP_Request, ARP_NAK
arpsrcmacaddr	MAC_ADDR	ARP/RARP 패킷의 소스 MAC 주소
arpdstmacaddr	MAC _ADDR	ARP/RARP 패킷의 대상 MAC 주소
arpsrcipaddr	IP_ADDR	ARP/RARP 패킷의 소스 IP 주소
arpdstipaddr	IP_ADDR	ARP/RARP 패킷의 대상 IP 주소
gratuitous	부울	ELS Indiating whether to check for a gratuitous ARP 패킷
주석	문자열	256자까지 텍스트 문자열

18.12.10.5. IPv4

*프로토콜 **ID: ip***

*이 유형의 규칙은 **root** 또는 **ipv4** 체인으로 이동해야 합니다.*

표 18.7. IPv4 프로토콜 유형

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcmacmask	MAC_MASK	보낸 사람의 MAC 주소에 적용되는 마스크
dstmacaddr	MAC_ADDR	대상의 MAC 주소
dstmacmask	MAC_MASK	대상의 MAC 주소에 적용되는 마스크
srcipaddr	IP_ADDR	소스 IP 주소
srcipmask	IP_MASK	소스 IP 주소에 적용되는 마스크
dstipaddr	IP_ADDR	대상 IP 주소
dstipmask	IP_MASK	대상 IP 주소에 적용되는 마스크
프로토콜	UINT8, 문자열	계층 4 프로토콜 식별자 프로토콜의 유효한 문자열: tcp, udp, udplite, esp, ah, icmp, igmp, sctp
srcportstart	UINT16	유효한 소스 포트 범위 시작; 필요한 프로토콜
srcportend	UINT16	유효한 소스 포트의 끝 부분; 필요한 프로토콜
dstportstart	UNIT16	유효한 대상 포트 범위 시작; 필요한 프로토콜
dstportend	UNIT16	유효한 대상 포트의 끝, 필요한 프로토콜
주석	문자열	256자까지 텍스트 문자열

18.12.10.6. IPv6

Protocol ID: ipv6

*이 유형의 규칙은 **root** 또는 **ipv6** 체인으로 이동해야 합니다.*

표 18.8. IPv6 프로토콜 유형

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcmacmask	MAC_MASK	보낸 사람의 MAC 주소에 적용되는 마스크
dstmacaddr	MAC_ADDR	대상의 MAC 주소
dstmacmask	MAC_MASK	대상의 MAC 주소에 적용되는 마스크
srcipaddr	IP_ADDR	소스 IP 주소
srcipmask	IP_MASK	소스 IP 주소에 적용되는 마스크
dstipaddr	IP_ADDR	대상 IP 주소
dstipmask	IP_MASK	대상 IP 주소에 적용되는 마스크
프로토콜	UINT8, 문자열	계층 4 프로토콜 식별자 프로토콜의 유효한 문자열: tcp, udp, udplite, esp, ah, icmpv6, sctp
scrportstart	UNIT16	유효한 소스 포트 범위 시작; 필요한 프로토콜
srcportend	UINT16	유효한 소스 포트의 끝 부분; 필요한 프로토콜
dstportstart	UNIT16	유효한 대상 포트 범위 시작; 필요한 프로토콜
dstportend	UNIT16	유효한 대상 포트의 끝, 필요한 프로토콜
주석	문자열	256자까지 텍스트 문자열

18.12.10.7. TCP/UDP/SCTP

프로토콜 ID: tcp, udp, sctp

chain 매개변수는 이러한 유형의 트래픽에 대해 무시되며 생략하거나 root로 설정해야 합니다.

표 18.9. TCP/UDP/SCTP 프로토콜 유형

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcipaddr	IP_ADDR	소스 IP 주소
srcipmask	IP_MASK	소스 IP 주소에 적용되는 마스크
dstipaddr	IP_ADDR	대상 IP 주소
dstipmask	IP_MASK	대상 IP 주소에 적용되는 마스크
scripto	IP_ADDR	소스 IP 주소 범위 시작
srcipfrom	IP_ADDR	소스 IP 주소 범위의 끝
dstipfrom	IP_ADDR	대상 IP 주소 범위 시작
dstipto	IP_ADDR	대상 IP 주소 범위의 끝
scrportstart	UNIT16	유효한 소스 포트 범위 시작; 필요한 프로토콜
srcportend	UINT16	유효한 소스 포트의 끝 부분; 필요한 프로토콜
dstportstart	UNIT16	유효한 대상 포트 범위 시작; 필요한 프로토콜
dstportend	UNIT16	유효한 대상 포트의 끝, 필요한 프로토콜
주석	문자열	256자까지 텍스트 문자열
상태	문자열	콤마로 구분된 NEW,ESTABLISHED,RELATED,INVALID 또는 NONE
플래그	문자열	TCP 전용: 마스크/플래그와 함께 마스크/플래그의 포맷은 각각 SYN,ACK,URG,PSH,FIN,RST 또는 NONE 또는 ALL의 쉼표로 구분된 목록일 수 있습니다.
ipset	문자열	libvirt 외부에서 관리되는 IPSet의 이름입니다.

속성 이름	datatype	정의
ipsetflags	IPSETFLAGS	IPSet의 플래그입니다. ipset 속성이 필요합니다.

18.12.10.8. ICMP

*프로토콜 **ID: icmp***

*참고: **chain** 매개변수는 이러한 유형의 트래픽에 대해 무시되며 생략하거나 **root**로 설정해야 합니다.*

표 18.10. ICMP 프로토콜 유형

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcmacmask	MAC_MASK	보낸 사람의 MAC 주소에 적용되는 마스크
dstmacaddr	MAD_ADDR	대상의 MAC 주소
dstmacmask	MAC_MASK	대상의 MAC 주소에 적용되는 마스크
srcipaddr	IP_ADDR	소스 IP 주소
srcipmask	IP_MASK	소스 IP 주소에 적용되는 마스크
dstipaddr	IP_ADDR	대상 IP 주소
dstipmask	IP_MASK	대상 IP 주소에 적용되는 마스크
srcipfrom	IP_ADDR	소스 IP 주소 범위 시작
scripto	IP_ADDR	소스 IP 주소 범위의 끝
dstipfrom	IP_ADDR	대상 IP 주소 범위 시작
dstipto	IP_ADDR	대상 IP 주소 범위의 끝
type	UNIT16	ICMP 유형
코드	UNIT16	ICMP 코드

속성 이름	datatype	정의
주석	문자열	256자까지 텍스트 문자열
상태	문자열	콤마로 구분된 NEW,ESTABLISHED,RELATED,INVALID 또는 NONE
ipset	문자열	libvirt 외부에서 관리되는 IPSet의 이름입니다.
ipsetflags	IPSETFLAGS	IPSet의 플래그입니다. ipset 속성이 필요합니다.

18.12.10.9. IGMP, ESP, AH, UDPLITE, 'ALL'

프로토콜 ID: igmp, esp, ah, udplite, all

***chain** 매개변수는 이러한 유형의 트래픽에 대해 무시되며 생략하거나 **root**로 설정해야 합니다.*

표 18.11. IGMP, ESP, AH, UDPLITE, 'ALL'

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcmacmask	MAC_MASK	보낸 사람의 MAC 주소에 적용되는 마스크
dstmacaddr	MAD_ADDR	대상의 MAC 주소
dstmacmask	MAC_MASK	대상의 MAC 주소에 적용되는 마스크
srcipaddr	IP_ADDR	소스 IP 주소
srcipmask	IP_MASK	소스 IP 주소에 적용되는 마스크
dstipaddr	IP_ADDR	대상 IP 주소
dstipmask	IP_MASK	대상 IP 주소에 적용되는 마스크
srcipfrom	IP_ADDR	소스 IP 주소 범위 시작
scripto	IP_ADDR	소스 IP 주소 범위의 끝

속성 이름	datatype	정의
dstipfrom	IP_ADDR	대상 IP 주소 범위 시작
dstipto	IP_ADDR	대상 IP 주소 범위의 끝
주석	문자열	256자까지 텍스트 문자열
상태	문자열	콤마로 구분된 NEW,ESTABLISHED,RELATED,INVALID 또는 NONE
ipset	문자열	libvirt 외부에서 관리되는 IPSet의 이름입니다.
ipsetflags	IPSETFLAGS	IPSet의 플래그입니다. ipset 속성이 필요합니다.

18.12.10.10. TCP/UDP/SCTP over IPV6

프로토콜 ID: tcp-ipv6, udp-ipv6, sctp-ipv6

***chain** 매개변수는 이러한 유형의 트래픽에 대해 무시되며 생략하거나 **root**로 설정해야 합니다.*

표 18.12. IPv6 프로토콜 유형을 통한 TCP, UDP, SCTP

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcipaddr	IP_ADDR	소스 IP 주소
srcipmask	IP_MASK	소스 IP 주소에 적용되는 마스크
dstipaddr	IP_ADDR	대상 IP 주소
dstipmask	IP_MASK	대상 IP 주소에 적용되는 마스크
srcipfrom	IP_ADDR	소스 IP 주소 범위 시작
scripto	IP_ADDR	소스 IP 주소 범위의 끝
dstipfrom	IP_ADDR	대상 IP 주소 범위 시작

속성 이름	datatype	정의
dstipto	IP_ADDR	대상 IP 주소 범위의 끝
srcportstart	UINT16	유효한 소스 포트 범위 시작
srcportend	UINT16	유효한 소스 포트 범위 종료
dstportstart	UINT16	유효한 대상 포트 범위 시작
dstportend	UINT16	유효한 대상 포트 범위 종료
주석	문자열	256자까지 텍스트 문자열
상태	문자열	콤마로 구분된 NEW,ESTABLISHED,RELATED,INVALID 또는 NONE
ipset	문자열	libvirt 외부에서 관리되는 IPSet의 이름입니다.
ipsetflags	IPSETFLAGS	IPSet의 플래그입니다. ipset 속성이 필요합니다.

18.12.10.11. ICMPv6

프로토콜 ID: icmpv6

***chain** 매개변수는 이러한 유형의 트래픽에 대해 무시되며 생략하거나 **root**로 설정해야 합니다.*

표 18.13. ICMPv6 프로토콜 유형

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcipaddr	IP_ADDR	소스 IP 주소
srcipmask	IP_MASK	소스 IP 주소에 적용되는 마스크
dstipaddr	IP_ADDR	대상 IP 주소
dstipmask	IP_MASK	대상 IP 주소에 적용되는 마스크

속성 이름	datatype	정의
srcipfrom	IP_ADDR	소스 IP 주소 범위 시작
scripto	IP_ADDR	소스 IP 주소 범위의 끝
dstipfrom	IP_ADDR	대상 IP 주소 범위 시작
dstipto	IP_ADDR	대상 IP 주소 범위의 끝
type	UINT16	ICMPv6 유형
코드	UINT16	ICMPv6 코드
주석	문자열	256자까지 텍스트 문자열
상태	문자열	콤마로 구분된 NEW,ESTABLISHED,RELATED,INVALID 또는 NONE
ipset	문자열	libvirt 외부에서 관리되는 IPSet의 이름입니다.
ipsetflags	IPSETFLAGS	IPSet의 플래그입니다. ipset 속성이 필요합니다.

18.12.10.12. IPv6를 통한 IGMP, ESP, AH, UDPLITE, 'ALL'

프로토콜 ID: igmp-ipv6, esp-ipv6, ah-ipv6, udplite-ipv6, all-ipv6

***chain** 매개변수는 이러한 유형의 트래픽에 대해 무시되며 생략하거나 **root**로 설정해야 합니다.*

표 18.14. IPv 프로토콜 유형 IGMP, ESP, AH, UDPLITE, 'ALL'

속성 이름	datatype	정의
srcmacaddr	MAC_ADDR	보낸 사람의 MAC 주소
srcipaddr	IP_ADDR	소스 IP 주소
srcipmask	IP_MASK	소스 IP 주소에 적용되는 마스크
dstipaddr	IP_ADDR	대상 IP 주소

속성 이름	datatype	정의
dstipmask	IP_MASK	대상 IP 주소에 적용되는 마스크
srcipfrom	IP_ADDR	소스 IP 주소 범위 시작
scripto	IP_ADDR	소스 IP 주소 범위의 끝
dstipfrom	IP_ADDR	대상 IP 주소 범위 시작
dstipto	IP_ADDR	대상 IP 주소 범위의 끝
주석	문자열	256자까지 텍스트 문자열
상태	문자열	콤마로 구분된 NEW,ESTABLISHED,RELATED,INVALID 또는 NONE
ipset	문자열	libvirt 외부에서 관리되는 IPSet의 이름입니다.
ipsetflags	IPSETFLAGS	IPSet의 플래그입니다. ipset 속성이 필요합니다.

18.12.11. 고급 필터 구성 주제

다음 섹션에서는 고급 필터 구성 항목에 대해 설명합니다.

18.12.11.1. 연결 추적

네트워크 필터링 하위 시스템(Linux)은 IP 테이블에 대한 연결 추적 지원을 사용합니다. 이를 통해 네트워크 트래픽(상태 일치)의 방향성을 적용하고 게스트 가상 시스템에 대한 동시 연결 수를 계산 및 제한하는 데 도움이 됩니다. 예를 들어 게스트 가상 시스템에 TCP 포트 8080이 서버로 열려 있는 경우 클라이언트는 포트 8080의 게스트 가상 머신에 연결할 수 있습니다. 방향성에 대한 연결 추적 및 적용은 게스트 가상 머신이 (TCP 클라이언트) 포트 8080에서 호스트 물리적 시스템과 원격 호스트 물리적 시스템으로의 연결을 시작하지 못하도록 합니다. 더 중요한 것은 추적이 원격 공격자가 게스트 가상 머신에 다시 연결하는 것을 방지하는 데 도움이 됩니다. 예를 들어 게스트 가상 머신 내부의 사용자가 공격자 사이트에서 포트 80에 대한 연결을 설정한 경우 공격자는 게스트 가상 머신으로 다시 TCP 포트 80에서 연결을 시작할 수 없습니다. 기본적으로 연결 추적을 활성화하는 연결 상태 일치 및 트래픽 방향성 적용이 설정되어 있습니다.

> *예 18.9. TCP 포트 연결을 해제하는 XML 예*
>
> *다음은 TCP 포트 12345에 대한 수신 연결을 위해 이 기능이 꺼져 있는 XML 조각의 예를 보여줍니다.*

```
[...]
 <rule direction='in' action='accept' statematch='false'>
   <cp dstportstart='12345'/>
 </rule>
[...]
```

이제 **TCP** 포트 **12345**로 들어오는 트래픽을 허용하지만 **VM** 내에서 (클라이언트) **TCP** 포트 **12345**를 시작해도 바람직하지 않을 수도 있습니다.

18.12.11.2. 연결 수 제한

게스트 가상 머신이 설정할 수 있는 연결 수를 제한하려면 지정된 유형의 트래픽에 대한 연결 제한을 설정하는 규칙을 제공해야 합니다. 예를 들어 **VM**이 한 번에 하나의 다른 **IP** 주소만 **ping**하도록 허용되고 한 번에 하나의 활성화된 **ssh** 연결만 있어야 합니다.

예 18.10. 연결에 제한을 설정하는 **XML** 샘플 파일

다음 **XML** 조각을 사용하여 연결을 제한할 수 있습니다.

```
[...]
<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>
[...]
```

참고

제한 규칙은 트래픽을 수락하기 위한 규칙보다 먼저 XML에 나열되어야 합니다. 예 18.10. "연결에 제한을 설정하는 XML 샘플 파일" 의 XML 파일에 따르면 포트 22로 전송된 DNS 트래픽을 허용하는 추가 규칙이 게스트 가상 머신으로 이동하며 ssh 데몬이 DNS 조회 실패와 관련된 이유로 ssh 세션이 설정되지 않도록 하는 추가 규칙이 추가되었습니다. 이 규칙을 벗어나면 ssh 클라이언트가 연결을 시도하면서 예기치 않게 중단될 수 있습니다. 트래픽 추적과 관련된 시간 제한을 처리하는 것과 관련하여 추가 주의를 사용해야 합니다. 사용자가 게스트 가상 머신 내부에서 종료될 수 있는 ICMP ping은 호스트 물리적 머신의 연결 추적 시스템에 시간 초과를 가질 수 있으므로 다른 ICMP ping이 통과할 수 없습니다.

가장 좋은 해결책은 다음 명령을 사용하여 호스트 물리적 시스템의 sysfs 의 시간 제한을 조정하는 것입니다.# echo 3 > /proc/sys/netfilter/nf_conntrack_icmp_timeout. 이 명령은 ICMP 연결 추적 타임아웃을 3초로 설정합니다. 이는 한 번 ping이 종료되면 다른 ping이 3초 후에 시작될 수 있다는 것입니다.

어떤 이유로든 게스트 가상 머신이 TCP 연결을 제대로 종료하지 않은 경우 특히 호스트 물리적 시스템에서 많은 시간 동안 TCP 시간 제한이 설정된 경우 더 긴 기간 동안 연결을 열 수 있습니다. 또한 유휴 연결은 패킷이 교환되면 다시 활성화될 수 있는 연결 추적 시스템에 시간이 초과될 수 있습니다.

그러나 제한을 너무 낮게 설정하면 새로 시작된 연결은 TCP 백오프에 유휴 연결을 강제 수행할 수 있습니다. 따라서 새 TCP 연결의 변동이 유휴 연결과 관련하여 홀수 트래픽 동작을 유발하지 않도록 연결 제한이 다소 높아야 합니다.

18.12.11.3. 명령줄 툴

virsh는 네트워크 필터에 대한 라이프사이클 지원을 통해 연장되었습니다. 네트워크 필터링 하위 시스템과 관련된 모든 명령은 접두사 nwfilter 로 시작합니다. 다음 명령을 사용할 수 있습니다.

- nwfilter-list : 모든 네트워크 필터의 UUID와 이름을 나열
- nwfilter-define : 새 네트워크 필터를 정의하거나 기존 네트워크 필터를 업데이트합니다(이름을 제공해야 함)
- nwfilter-undefine : 지정된 네트워크 필터를 삭제합니다(이름을 제공해야 함). 현재 사용 중인 네트워크 필터를 삭제하지 마십시오.

- ***nwfilter-dumpxml :** 지정된 네트워크 필터를 표시합니다(이름을 제공해야 함)*

- ***nwfilter-edit :** 지정된 네트워크 필터 편집 (이름을 제공해야 함)*

18.12.11.4. 기존 네트워크 필터

*다음은 **libvirt**를 사용하여 자동으로 설치된 네트워크 필터 예제 목록입니다.*

*표 **18.15. ICMPv6** 프로토콜 유형*

명령 이름	설명
no-arp-spoofing	게스트 가상 머신이 ARP 트래픽을 스푸핑하지 못하도록 합니다. 이 필터는 ARP 요청 및 응답 메시지만 허용하며 해당 패킷에 게스트 가상 머신의 MAC 및 IP 주소를 포함하도록 강제 적용합니다.
allow-dhcp	게스트 가상 머신에서 DHCP를 통해 IP 주소를 요청할 수 있습니다(모든 DHCP 서버에서)
allow-dhcp-server	게스트 가상 머신에서 지정된 DHCP 서버의 IP 주소를 요청할 수 있습니다. DHCP 서버의 점선 IP 주소는 이 필터에 대한 참조로 제공되어야 합니다. 변수의 이름은 *DHCPSERVER* 여야 합니다.
no-ip-spoofing	게스트 가상 머신이 패킷 내부와 다른 소스 IP 주소로 IP 패킷을 보내지 못하도록 합니다.
no-ip-multicast	게스트 가상 머신이 IP 멀티 캐스트 패킷을 보내지 못하도록 합니다.
clean-traffic	MAC, IP 및 ARP 스푸핑을 방지합니다. 이 필터는 다른 여러 필터를 빌딩 블록으로 참조합니다.

*이러한 필터는 빌딩 블록일 뿐이며 유용한 네트워크 트래픽 필터링을 제공하기 위해 다른 필터와 조합해야 합니다. 위 목록에서 가장 많이 사용되는 필터는 깔끔한 트래픽 필터입니다. 이 필터 자체는 예를 들어 **no-ip-multicast** 필터와 결합하여 가상 머신이 패킷 스푸핑을 방지하여 **IP** 멀티 캐스트 트래픽을 전송하지 못하도록 할 수 있습니다.*

18.12.11.5. 자체 필터 작성

libvirt는 몇 가지 네트워킹 필터만 제공하므로 자체 작성을 고려할 수 있습니다. 이를 계획할 때 네트워크 필터링 하위 시스템과 내부적으로 작동하는 방법에 대해 알아야 할 몇 가지 사항이 있습니다. 확실히

당신은 또한 필터링하고 싶은 프로토콜을 잘 이해하고 있으므로 원하는 트래픽을 통과 할 수있는 것보다 더 이상 트래픽을 전달 할 수 없다는 것을 알고 있어야합니다.

네트워크 필터링 하위 시스템은 현재 Linux 호스트 물리적 시스템에서만 사용할 수 있으며 Qemu 및 KVM 유형의 가상 시스템에서만 작동합니다. Linux에서는 ebtables, iptables 및 ip6tables에 대한 지원을 기반으로 빌드되고 해당 기능을 사용합니다. 18.12.10절. "지원되는 프로토콜" 에 있는 목록을 고려할 때 ebtables를 사용하여 다음 프로토콜을 구현할 수 있습니다.

- mac
- STP(범위 트리 프로토콜)
- VLAN(802.1Q)
- ARP, rarp
- ipv4
- ipv6

IPv4에서 실행되는 모든 프로토콜은 iptables를 사용하여 지원되며 IPv6을 통한 프로토콜은 ip6tables를 사용하여 구현됩니다.

Linux 호스트 물리적 시스템을 사용하면 libvirt의 네트워크 필터링 하위 시스템에서 생성한 모든 트래픽 필터링 규칙이 먼저 ebtables 및 이후의 iptables 또는 ip6tables 필터를 통해 구현되는 필터링 지원을 통해 전달됩니다. 필터 트리에 프로토콜을 포함하는 규칙(mac, stp, vlan arp, ipv4 또는 ipv6)이 있는 경우 나열된 ebtable 규칙과 값이 자동으로 먼저 사용됩니다.

동일한 프로토콜을 위한 여러 체인을 생성할 수 있습니다. 체인 이름에는 이전에 열거된 프로토콜 중 하나의 접두사가 있어야 합니다. ARP 트래픽 처리를 위한 추가 체인을 만들려면 arp-test라는 체인을 지정할 수 있습니다.

예를 들어 IP 프로토콜 필터를 사용하여 소스 및 대상 포트별로 UDP 트래픽을 필터링하고 허용되는 UDP 패킷의 프로토콜, 소스 및 대상 IP 주소 및 포트의 속성을 지정할 수 있습니다. 이를 통해 ebtables를 사용하여 UDP 트래픽을 조기에 필터링할 수 있습니다. 그러나 UDP 패킷과 같은 IP 또는 IPv6 패킷은

ebtables 계층을 통과하면 iptables 또는 ip6tables 규칙을 인스턴스화하는 필터 트리에 하나 이상의 규칙이 있으면 해당 필터링 계층에 대해 UDP 패킷 전달을 허용하는 규칙도 필요합니다. 이는 적절한 udp 또는 udp-ipv6 트래픽 필터링 노드를 포함하는 규칙을 사용하여 수행할 수 있습니다.

예 18.11. 사용자 정의 필터 생성

다음 요구 사항을 충족하기 위해 필터가 필요하다고 가정합니다.

- *VM의 인터페이스가 MAC, IP 및 ARP 스푸핑을 사용하지 않음*
- *VM 인터페이스의 TCP 포트 22 및 80만 열기*
- *VM이 인터페이스에서 ping 트래픽을 보낼 수 있지만 인터페이스에서 VM을 ping하도록 허용하지 않습니다.*
- *VM에서 DNS 조회(UDP to port 53)를 수행할 수 있습니다.*

기존 스마트 트래픽 네트워크 필터에 의해 스푸핑을 방지해야 하므로 이 작업을 수행하는 방법은 사용자 정의 필터에서 참조하는 것입니다.

TCP 포트 22 및 80에 대한 트래픽을 활성화하기 위해 이러한 유형의 트래픽을 활성화하기 위해 두 개의 규칙이 추가됩니다. 게스트 가상 머신이 ping 트래픽을 보낼 수 있도록 하려면 ICMP 트래픽에 대한 규칙이 추가됩니다. 간단한 이유로, 일반 ICMP 트래픽을 게스트 가상 시스템에서 시작할 수 있으며 ICMP 에코 요청 및 응답 메시지에 지정되지 않습니다. 다른 모든 트래픽은 게스트 가상 머신에 도달하거나 시작할 수 없습니다. 이렇게 하려면 다른 모든 트래픽을 삭제하는 규칙이 추가됩니다. 게스트 가상 시스템이 test 이고 필터를 연결할 인터페이스가 eth0 이라고 가정하면 test- eth0 이라는 필터가 생성됩니다.

이러한 고려 사항의 결과는 다음과 같은 네트워크 필터 XML입니다.

```
<filter name='test-eth0'>
  <!- - This rule references the clean traffic filter to prevent MAC, IP and ARP spoofing. By not
providing an IP address parameter, libvirt will detect the IP address the guest virtual machine is
using. - ->
  <filterref filter='clean-traffic'/>

  <!- - This rule enables TCP ports 22 (ssh) and 80 (http) to be reachable - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='22'/>
```

```
  </rule>

  <rule action='accept' direction='in'>
    <tcp dstportstart='80'/>
  </rule>

  <!- - This rule enables general ICMP traffic to be initiated by the guest virtual machine including
ping traffic - ->
  <rule action='accept' direction='out'>
    <icmp/>
  </rule>>

  <!- - This rule enables outgoing DNS lookups using UDP - ->
  <rule action='accept' direction='out'>
    <udp dstportstart='53'/>
  </rule>

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

</filter>
```

18.12.11.6. 사용자 정의 필터 샘플

위 **XML**의 규칙 중 하나에는 게스트 가상 시스템의 **IP** 주소가 소스 또는 대상 주소로 포함되어 있지만 트래픽 필터링은 올바르게 작동합니다. 그 이유는 규칙의 평가가 인터페이스별로 내부적으로 발생하는 반면, 규칙은 소스 또는 대상 **IP** 주소가 아닌 패킷을 전송하거나 수신하는지 여부에 따라 추가로 평가되기 때문입니다.

예 18.12. 네트워크 인터페이스 설명을 위한 XML 샘플

테스트 게스트 가상 머신의 도메인 **XML** 내에서 가능한 네트워크 인터페이스 설명에 대한 **XML** 조각은 다음과 같습니다.

```
 [...]
  <interface type='bridge'>
    <source bridge='mybridge'/>
    <filterref filter='test-eth0'/>
  </interface>
 [...]
```

ICMP 트래픽을 보다 엄격하게 제어하고 게스트 가상 머신에서 **ICMP** 에코 요청만 보낼 수 있고 게스트 가상 머신에서 **ICMP** 에코 응답만 수신하도록 위의 **ICMP** 규칙은 다음 두 가지 규칙으로 교체될 수 있습니다.

```
<!- - 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>
```

예 18.13. 두 번째 예제 사용자 정의 필터

이 예제에서는 위의 예제와 같이 유사한 필터를 빌드하는 방법을 설명하지만 게스트 가상 시스템 내에 있는 **ftp** 서버를 사용하여 요구 사항 목록을 확장합니다. 이 필터의 요구 사항은 다음과 같습니다.

- 게스트 가상 머신의 인터페이스가 **MAC, IP** 및 **ARP** 스푸핑을 사용하지 않도록 합니다.
- 게스트 가상 머신 인터페이스에서 **TCP** 포트 **22** 및 **80**만 열기
- 게스트 가상 머신이 인터페이스에서 **ping** 트래픽을 보낼 수 있지만 인터페이스에서 게스트 가상 머신을 **ping**할 수 없습니다.
- 게스트 가상 머신이 **DNS** 조회를 수행할 수 있음 **(UDP ~ 53)**
- 게스트 가상 머신 내부에서 실행할 수 있도록 **ftp** 서버(활성 모드)를 활성화합니다.

게스트 가상 머신 내부에서 **FTP** 서버를 실행하도록 허용하는 추가 요구 사항은 **FTP** 제어 트래픽에 대해 포트 **21**에 연결할 수 있도록 허용하며 게스트 가상 머신이 게스트 가상 시스템의 **TCP** 포트 **20**에서 **FTP** 클라이언트(**FTP** 활성 모드)로 다시 생성되는 발신 **TCP** 연결을 설정할 수 있도록 허용하는 요구 사항입니다. 이 필터를 작성하는 방법에는 여러 가지가 있으며 이 예제에는 두 가지 가능한 솔루션이 포함됩니다.

첫 번째 솔루션은 **Linux** 호스트 물리적 머신의 연결 추적 프레임워크에 후크를 제공하는 **TCP** 프로토콜의 **state** 속성을 사용합니다. 게스트 가상 머신 시작 **FTP** 데이터 연결 **(FTP** 활성 모드**)**의 경우 **RELATED** 상태는 게스트 가상 머신 시작 **FTP** 데이터 연결이 기존 **FTP** 제어 연결의 결과 **(**또는 **'has relationship with')**로 인해 방화벽을 통해 패킷을 전달할 수 있음을 탐지하는 데 사용됩니다. 그러나 **RELATED** 상태는 **FTP** 데이터 경로에 대해 발신 **TCP** 연결의 첫 번째 패킷에만 유효합니다. 그 후, 상태는 **ESTABLISHED**이며, 이는 들어오고 나가는 방향에 동일하게 적용됩니다. 이 모든 것은 게스트

*가상 시스템의 **TCP** 포트 **20**에서 시작되는 **FTP** 데이터 트래픽과 관련이 있습니다. 그러면 다음과 같은 해결 방법이 있습니다.*

```
<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** 상태를 사용하여 필터를 시도하기 전에 적절한 연결 추적 모듈이 호스트 물리적 시스*

*템의 커널에 로드되었는지 확인해야 합니다. 커널 버전에 따라 게스트 가상 시스템과의 **FTP** 연결이 설정되기 전에 다음 두 명령 중 하나를 실행해야 합니다.*

- ***#modprobe nf_conntrack_ftp -** 사용 가능한 경우 **OR***
- ***#modprobe ip_conntrack_ftp** 를 위의 코드를 사용할 수 없는 경우*

***FTP** 이외의 프로토콜을 **RELATED** 상태와 함께 사용하는 경우 해당 모듈을 로드해야 합니다. 모듈은 **ftp, tftp, irc, sip, sctp** 및 **amanda** 프로토콜에서 사용할 수 있습니다.*

*두 번째 솔루션은 이전 솔루션보다 연결의 상태 플래그를 사용합니다. 이 솔루션은 트래픽 흐름의 첫 번째 패킷이 탐지될 때 **NEW** 연결 상태가 유효하다는 사실을 활용합니다. 그 후, 흐름의 첫 번째 패킷이 허용되면 흐름은 연결이 되므로 **ESTABLISHED** 상태가 됩니다. 따라서 **ESTABLISHED** 연결의 패킷이 게스트 가상 머신에 도달하거나 게스트 가상 머신에 의해 보낼 수 있도록 일반 규칙을 작성할 수 있습니다. 이는 **NEW** 상태로 식별되는 첫 번째 패킷에 대한 특정 규칙을 작성하고 데이터가 허용되도록 포트에 지시합니다. 명시적으로 승인되지 않은 포트에 대한 모든 패킷은 삭제됩니다, 따라서 **ESTABLISHED** 상태에 도달하지 않습니다. 해당 포트에서 전송된 후속 패킷도 삭제됩니다.*

```
<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>
```

18.12.12. 제한 사항

다음은 네트워크 필터링 하위 시스템의 현재 알려진 제한 사항 목록입니다.

- 게스트 가상 시스템의 최상위 수준 필터에서 참조하는 전체 필터 트리도 대상 호스트 물리적 시스템에서 사용할 수 있는 경우에만 **VM** 마이그레이션이 지원됩니다. 예를 들어 네트워크 필터 정리**-traffic** 는 모든 **libvirt** 설치에서 사용할 수 있으므로 이 필터를 참조하는 게스트 가상 머신의 마이그레이션이 활성화되어야 합니다. 버전 호환성을 보장하기 위해 패키지를 정기적으로 업데이트하여 최신 버전의 **libvirt**를 사용하고 있는지 확인하십시오.

- 인터페이스와 연결된 네트워크 트래픽 필터를 손실하지 않으려면 버전 **0.8.1** 이상을 설치하는 경우 마이그레이션이 수행되어야 합니다.

- **VLAN(802.1Q)** 패킷은 게스트 가상 머신에서 전송되는 경우 **rp, rarp, ipv4** 및 **ipv6**에 대한 프로토콜 **ID** 규칙으로 필터링할 수 없습니다. 프로토콜 **ID, MAC** 및 **VLAN**으로만 필터링할 수 있습니다. 따라서 예제 필터 **clean-traffic** 예 18.1. “네트워크 필터링의 예” 가 예상대로 작동하지 않습니다.

18.13. 터널 생성

이 섹션에서는 다양한 터널링 시나리오를 구현하는 방법을 설명합니다.

18.13.1. 멀티 캐스트 터널 생성

멀티 캐스트 그룹은 가상 네트워크를 표시하도록 설정됩니다. 네트워크 장치가 동일한 멀티 캐스트 그룹에 있는 게스트 가상 머신은 호스트 물리적 시스템에서도 서로 통신할 수 있습니다. 이 모드는 권한이

없는 사용자가도 사용할 수 있습니다. 기본 **DNS** 또는 **DHCP** 지원이 없으며 나가는 네트워크 액세스가 없습니다. 발신 네트워크 액세스를 제공하려면 게스트 가상 머신 중 하나에 연결된 두 번째 **NIC**가 있어야 첫 번째 네 가지 네트워크 유형 중 하나에 적절한 라우팅을 제공해야 합니다. 멀티 캐스트 프로토콜은 게스트 가상 머신 사용자 모드와 호환됩니다. 제공하는 소스 주소는 멀티 캐스트 주소 블록에 사용되는 주소에서 사용해야 합니다.

멀티 캐스트 터널을 만들려면 **<devices>** 요소에 다음 **XML** 세부 정보를 지정합니다.

그림 **18.28.** 멀티 캐스트 터널 **XML** 예

```
...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
    <source address='230.0.0.1' port='5558'/>
  </interface>
</devices>
...
```

18.13.2. TCP 터널 생성

TCP 클라이언트/서버 아키텍처는 가상 네트워크를 제공합니다. 이 구성에서 하나의 게스트 가상 머신은 네트워크의 서버 끝을 제공하며 다른 모든 게스트 가상 시스템은 클라이언트로 구성됩니다. 모든 네트워크 트래픽은 게스트 가상 머신 서버를 통해 게스트 가상 머신 클라이언트 간에 라우팅됩니다. 이 모드는 권한이 없는 사용자에게도 사용할 수 있습니다. 이 모드는 기본 **DNS** 또는 **DHCP**를 지원하지 않거나 발신 네트워크 액세스를 제공하지 않습니다. 발신 네트워크 액세스를 제공하려면 게스트 가상 머신 중 하나에 연결된 두 번째 **NIC**가 있어야 첫 번째 네 가지 네트워크 유형 중 하나에 적절한 라우팅을 제공해야 합니다.

TCP 터널을 만들려면 다음 **XML** 세부 정보를 **<devices>** 요소에 배치합니다.

그림 18.29. TCP 터널 도메인 XMl 예

```
    ...
  <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>
  ...
```

18.14. VLAN 태그 설정

virtual local area network (vLAN) 태그는 virsh net-edit 명령을 사용하여 추가됩니다. 이 태그는 SR-IOV 장치를 사용하여 PCI 장치 할당과 함께 사용할 수도 있습니다. 자세한 내용은 9.1.7절. "SR-IOV 장치를 사용하여 PCI 할당(Passthrough) 구성" 에서 참조하십시오.

그림 18.30. v setting VLAN 태그(지원되는 네트워크 유형에서만)

```
<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>
```

네트워크 유형이 게스트에 vlan 태깅을 투명하게 지원하는 경우 선택적 <vlan> 요소에서는 이 네트워크를 사용하여 모든 게스트의 트래픽에 적용되도록 하나 이상의 vlan 태그를 지정할 수 있습니다. (openvswitch 및 type='hostdev' SR-IOV 네트워크는 게스트 트래픽의 투명 VLAN 태그 지정을 지원합니다. 표준 Linux 브리지 및 libvirt의 자체 가상 네트워크를 포함한 기타 모든 기능은 지원하지 않습니다. 802.1QBH(vn-link) 및 802.1Qbg(VEPA) 스위치는 특정 vlans에 게스트 트래픽을 태그하는 고유한 방법(libvirt 제외)을 제공합니다. tag 속성은 사용할 vlan 태그를 지정합니다. 네트워크에 둘 이상의 <vlan> 요

소가 정의되어 있는 경우 사용자가 지정된 모든 태그를 사용하여 VLAN 트렁크를 수행하는 것으로 가정합니다. 단일 태그로 VLAN 트렁크를 원하는 경우 선택적 속성 trunk='yes'를 VLAN 요소에 추가할 수 있습니다.

openvswitch를 사용하여 네트워크 연결의 경우 'native-tagged' 및 'native-untagged' VLAN 모드를 구성할 수 있습니다. 이는 <태그> 요소에서 선택적 nativeMode 특성을 사용합니다. nativeMode는 'tagged' 또는 'untagged'로 설정할 수 있습니다. 요소의 id 속성은 기본 vlan을 설정합니다.

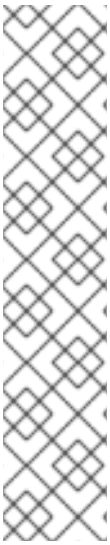
<VLAN> 요소는 <portgroup> 요소와 도메인의 <인터페이스> 요소에 직접 지정할 수도 있습니다. vlan 태그가 여러 위치에 지정된 경우 인터페이스의 설정이 우선한 다음 <인터페이스> 구성에서 선택한 <portgroup> 의 설정이 우선합니다. <네트워크> 의 <vlan> 은 <포트 그룹> 또는 <인터페이스에> 제공되지 않은 경우에만 선택됩니다.

18.15. 가상 네트워크에 QOS 적용

QoS(Quality of Service) 는 네트워크 상의 모든 사용자에게 최적의 환경을 보장하는 리소스 제어 시스템을 참조하여 지연, 지터 또는 패킷 손실이 발생하지 않도록 합니다. QoS는 특정 애플리케이션 또는 사용자/그룹 특정일 수 있습니다. 자세한 내용은 20.16.9.14절. “서비스 품질” 를 참조하십시오.

19장. QEMU-KVM 명령, 플래그 및 인수

19.1. 소개



참고

이 장의 주요 목적은 **Red Hat Enterprise Linux 6**에서 에뮬레이터 및 하이퍼바이저로 사용되는 **qemu-kvm** 유틸리티 명령, 플래그 및 인수 목록을 제공하는 것입니다. 이는 작동하는 것으로 알려진 옵션에 대한 포괄적인 요약이지만 자체 위험을 감수하는 데 사용됩니다. **Red Hat Enterprise Linux 6**는 **KVM**을 기본 가상화 기술로 사용합니다. 사용된 머신 에뮬레이터 및 하이퍼바이저는 **qemu-kvm**이라는 수정된 버전의 **QEMU**입니다. 이 버전에서는 원래 **QEMU**의 모든 구성 옵션을 지원하지 않으며 몇 가지 추가 옵션도 추가되었습니다.

여기에 나열 되지 않은 옵션은 수행해서는 안 됩니다.

허용 목록 형식

- **<name > -** 구문 설명에 사용된 경우 이 문자열을 사용자 정의 값으로 교체해야 합니다.
- **[a|b|c] -** 구문 설명에 사용된 경우 |로 구분된 문자열 중 하나만 사용됩니다.
- 주석이 없으면 가능한 모든 값과 함께 옵션이 지원됩니다.

19.2. 기본 옵션

이 섹션에서는 기본 옵션에 대한 정보를 제공합니다.

에뮬레이션된 시스템

-M <machine-type>

-machine <machine-type>[,<property>[=<value>][,..]]

프로세서 유형

-cpu <model>[,<FEATURE>][...]

*추가 모델은 **-cpu ?** 명령을 실행하여 볼 수 있습니다.*

- ***Opteron_G5 - AMD Opteron 63xx 클래스 CPU***
- ***Opteron_G4 - AMD Opteron 62xx 클래스 CPU***
- ***Opteron_G3 - AMD Opteron 23xx(AMD Opteron Gen 3)***
- ***Opteron_G2 - AMD Opteron 22xx(AMD Opteron Gen 2)***
- ***Opteron_G1 - AMD Opteron240(AMD Opteron Gen 1)***
- ***Westmere - 서부 E56xx/L56xx/X56xx (Nehalem-C)***
- ***Haswell - Intel Core Processor (Haswell)***
- ***SandyBridge - Intel Xeon E312xx (Sandy Bridge)***
- ***Nehalem - Intel Core i7 9xx (Nehalem Class Core i7)***
- ***Penryn - Intel Core 2 Duo P9xxx (Penryn Class Core 2)***
- ***Conroe - Intel Celeron_4x0 (Conroe/Merom Class Core 2)***
- ***cpu64-rhel5 - Red Hat Enterprise Linux 5에서 QEMU 가상 CPU 버전 지원***
- ***cpu64-rhel6 - Red Hat Enterprise Linux 6 지원 QEMU 가상 CPU 버전***

- *default - special 옵션은 위의 기본 옵션을 사용합니다.*

프로세서 토폴로지

-smp <n>[,cores=<ncores>][,threads=<nthreads>][,sockets=<nsocks>][,maxcpus=<maxcpus>]

프로세서 토폴로지에 대한 하이퍼 바이저 및 게스트 운영 체제 제한이 적용됩니다.

***NUMA** 시스템*

-numa <nodes>[,mem=<size>][,cpus=<cpu[-cpu>]][,nodeid=<node>]

프로세서 토폴로지에 대한 하이퍼 바이저 및 게스트 운영 체제 제한이 적용됩니다.

메모리 크기

-m <megs>

지원되는 값은 게스트 최소 및 최대 값과 하이퍼바이저 제한에 의해 제한됩니다.

키보드 레이아웃

-k <language>

게스트 이름

-name <name>

*게스트 **UUID***

-uuid <uuid>

19.3. 디스크 옵션

이 섹션에서는 디스크 옵션에 대한 정보를 제공합니다.

일반 드라이브

-drive <option>[,<option>[,<option>[,...]]]

다음 옵션과 함께 지원됩니다.

- **readonly[on|off]**
- **werror[enospc|report|stop|ignore]**
- **rerror[report|stop|ignore]**
- **id=<id>**

  드라이브 ID에는 if=none에 대한 다음과 같은 제한이 있습니다.

  - *IDE 디스크에는 다음과 같은 형식의 <id>가 있어야 합니다. drive-ide0-<BUS>-<UNIT>*

    올바른 형식의 예:

    -drive if=none,id=drive-ide0-<BUS>-<UNIT>,.. -device ide-ide,drive=ide0-<BUS>-<UNIT>,bus=ide.<BUS>,unit=<UNIT>

- **file=<file>**

  <file> 값은 다음 규칙을 사용하여 구문 분석됩니다.

  - *플로피 장치를 <file>으로 전달하는 것은 지원되지 않습니다.*
  - *cd-rom 장치를 <file>으로 전달하는 것은 cdrom 미디어 유형(media=cdrom)에서만 지원되며, IDE 드라이브(if=ide 또는 if=none + -device ide-drive)로만 지원됩니다.*

- *<file>이 블록이나 문자 장치가 없는 경우 ':'을 포함하지 않아야 합니다.*

- ***if=<interface>***

  다음 인터페이스가 지원됩니다. ***none, ide, virtio,*** *플로피.*

- ***index=<index>***

- ***media=<media>***

- ***cache=<cache>***

  지원되는 값: ***none, writeback*** *또는* ***writethrough.***

- ***copy-on-read=[on|off]***

- ***snapshot=[yes|no]***

- *직렬 =<* ***serial>***

- ***aio=<aio>***

- ***format=<format>***

  이 옵션은 필수가 아니며 생략할 수 있습니다. 그러나 보안 위험을 나타내므로 원시 이미지에는 권장되지 않습니다. 지원되는 형식은 다음과 같습니다.

  - ***qcow2***

- raw

부팅 옵션

-boot [order=<drives>][,menu=[on|off]]

스냅샷 모드

-snapshot

19.4. 표시 옵션

이 섹션에서는 표시 옵션에 대한 정보를 제공합니다.

그래픽 비활성화

-nographic

VGA 카드 에뮬레이션

-vga <type>

지원되는 유형:

- Cirrus - Cirrus logic GD5446 비디오 카드
- STD - Bochs VBE 확장이 포함된 표준 VGA 카드.
- QXL - Spice paravirtual 카드.
- none - VGA 카드 비활성화.

VNC 디스플레이

-vnc <display>[,<option>[,<option>[,...]]]

지원되는 표시 값:

- ***[<host>]:<port>***
- ***unix:<path>***
- ***share[allow-exclusive|force-shared|ignore]***
- ***none - 다른 옵션이 지정되지 않은 상태에서 지원됩니다.***

지원되는 옵션은 다음과 같습니다.

- ***to=<port>***
- ***reverse***
- *암호*
- ***tls***
- ***X 509 =</path/to/certificate/dir> - tls** 가 지정될 때 지원됨*
- ***X 509verify =</path/to/certificate/dir> - tls** 가 지정될 때 지원됨*
- ***sasl***
- ***acl***

***SPICE** 데스크탑*

-pic e 옵션[,option[...]]

지원되는 옵션은 다음과 같습니다.

- ***port=<number>***
- ***addr=<addr>***
- ***ipv4***

  ipv6
- ***password=<secret>***
- ***disable-ticketing***
- ***disable-copy-paste***
- ***tls-port=<number>***
- ***x509-dir=</path/to/certificate/dir>***
- ***x509-key-file=<file>***

  x509-key-password=<file>

  x509-cert-file=<file>

x509-cacert-file=<file>

x509-dh-key-file=<file>

- *tls-cipher=<list>*
- *tls-channel[main|display|cursor|inputs|record|playback]*

  plaintext-channel[main|display|cursor|inputs|record|playback]
- *image-compression=<compress>*
- *jpeg-wan-compression=<value>*

  zlib-glz-wan-compression=<value>
- *streaming-video=[off|all|filter]*
- *agent-mouse=[on|off]*
- *playback-compression=[on|off]*
- *seamless-migratio=[on|off]*

19.5. 네트워크 옵션

이 섹션에서는 네트워크 옵션에 대한 정보를 제공합니다.

***TAP** 네트워크*

-netdev tap,id=<id>[,<options>...]

*다음 옵션이 지원됩니다(모든 **name=value** 형식 사용).*

- **ifname**
- **fd**
- **script**
- **downscript**
- **sndbuf**
- **vnet_hdr**
- **vhost**
- **vhostfd**
- **vhostforce**

19.6. 장치 옵션

이 섹션에서는 장치 옵션에 대한 정보를 제공합니다.

일반 장치

-device <driver>[,<prop>[=<value>][,...]]

모든 드라이버는 다음과 같은 속성을 지원합니다.

- ***id***
- ***bus***

지원되는 드라이버는 다음과 같습니다(사용 가능한 속성).

- ***pci-assign***
  - ***host***
  - ***bootindex***
  - ***configfd***
  - ***addr***
  - ***rombar***
  - ***romfile***
  - ***Multifunction***

    장치에 여러 기능이 있는 경우 해당 장치를 모두 동일한 게스트에 할당해야 합니다.

- ***rtl8139***

- ***mac***
- ***netdev***
- ***bootindex***
- ***addr***

- ***e1000***
  - ***mac***
  - ***netdev***
  - ***bootindex***
  - ***addr***

- ***virtio-net-pci***
  - ***ioeventfd***
  - ***vectors***
  - *간접*
  - ***event_idx***

- ***csum***
- ***guest_csum***
- ***gso***
- ***guest_tso4***
- ***guest_tso6***
- ***guest_ecn***
- ***guest_ufo***
- ***host_tso4***
- ***host_tso6***
- ***host_ecn***
- ***host_ufo***
- ***mrg_rxbuf***
- ***status***
- ***ctrl_vq***

- ***ctrl_rx***
- ***ctrl_vlan***
- ***ctrl_rx_extra***
- ***mac***
- ***netdev***
- ***bootindex***
- ***x-txtimer***
- ***x-txburst***
- ***tx***
- ***addr***

- ***qxl***
  - ***ram_size***
  - ***vram_size***
  - *버전*

- ***cmdlog***
- ***addr***

- ***ide-drive***
  - *단위*
  - *드라이브*
  - ***physical_block_size***
  - ***bootindex***
  - ***ver***
  - ***wwn***

- ***virtio-blk-pci***
  - *클래스*
  - *드라이브*
  - ***logical_block_size***
  - ***physical_block_size***

- ***min_io_size***
- ***opt_io_size***
- ***bootindex***
- ***ioeventfd***
- ***vectors***
- ***indirect_desc***
- ***event_idx***
- ***scsi***
- ***addr***

- ***virtio-scsi-pci - 6.3의 기술 프리뷰는 6.4 이후 지원됩니다.***

  Windows 게스트의 경우 기술 프리뷰인 Windows Server 2003은 6.5부터 더 이상 지원되지 않습니다. 그러나 Windows Server 2008 및 2012 및 Windows 데스크탑 7 및 8은 6.5 이후 완전히 지원됩니다.

  - ***vectors***
  - ***indirect_desc***
  - ***event_idx***

- ***num_queues***
- ***addr***

- ***isa-debugcon***
- ***isa-serial***
  - ***index***
  - ***iobase***
  - ***IRQ***
  - ***chardev***
- ***virtserialport***
  - ***nr***
  - ***chardev***
  - ***name***
- ***virtconsole***
  - ***nr***

    - ***chardev***
    - ***name***
- ***virtio-serial-pci***
    - ***vectors***
    - *클래스*
    - ***indirect_desc***
    - ***event_idx***
    - ***max_ports***
    - ***flow_control***
    - ***addr***
- ***ES1370***
    - ***addr***
- ***AC97***
    - ***addr***

- ***intel-hda***
  - ***addr***
- ***hda-duplex***
  - ***cabf***
- ***hda-micro***
  - ***cabf***
- ***hda-output***
  - ***cabf***
- ***i6300esb***
  - ***addr***
- ***ib700 - 속성이 없음***
- ***SGA - 속성이 없음***
- ***virtio-balloon-pci***
  - ***indirect_desc***

    - ***event_idx***
    - ***addr***
- ***usb-tablet***
    - ***migrate***
    - ***port***
- ***usb-kbd***
    - ***migrate***
    - ***port***
- ***usb-mouse***
    - ***migrate***
    - ***port***
- ***USB-ccid - 6.2 이후 지원***
    - ***port***
    - *슬롯*

- *USB-호스트 - 6.2 이후 기술 프리뷰*
  - *hostbus*
  - *hostaddr*
  - *hostport*
  - *vendorID*
  - *ProductID*
  - *isobufs*
  - *port*
- *USB-hub - 6.2에서 지원*
  - *port*
- *USB-ehci - 6.2 이후 기술 프리뷰*
  - *freq*
  - *maxframes*
  - *port*

- *USB-storage - 6.2 이후 기술 프리뷰*
  - *드라이브*
  - ***bootindex***
  - ***serial***
  - ***removable***
  - ***port***
- *USB-redir - 6.4에서 지원되는 6.3 기술 프리뷰*
  - ***chardev***
  - ***filter***
- *SCSI-cd - 6.3의 기술 프리뷰, 6.4 이후 지원*
  - *드라이브*
  - ***logical_block_size***
  - ***physical_block_size***
  - ***min_io_size***

- ***opt_io_size***
- ***bootindex***
- ***ver***
- ***serial***
- ***scsi-id***
- ***LUN***
- ***channel-scsi***
- ***wwn***

- ***SCSI-hd - 6.4에서 지원되는 6.3의 기술 프리뷰***
  - *드라이브*
  - ***logical_block_size***
  - ***physical_block_size***
  - ***min_io_size***
  - ***opt_io_size***

- ***bootindex***
- ***ver***
- ***serial***
- ***scsi-id***
- ***LUN***
- ***channel-scsi***
- ***wwn***

- ***SCSI-block - 6.4 이후 지원되는 6.3의 기술 프리뷰***
  - *드라이브*
  - ***bootindex***

- ***SCSI-disk - 6.3의 기술 프리뷰***
  - ***drive=drive***
  - ***logical_block_size***
  - ***physical_block_size***

- ***min_io_size***
- ***opt_io_size***
- ***bootindex***
- ***ver***
- ***serial***
- ***scsi-id***
- ***LUN***
- ***channel-scsi***
- ***wwn***

- ***piix3-usb-uhci***
- ***piix4-usb-uhci***
- ***ccid-card-passthru***

글로벌 장치 설정

-global <device>.<property>=<value>

지원되는 장치 및 속성 "일반 장치" 섹션에 다음과 같이 추가 장치가 있습니다.

- *isa-fdc*
  - *driveA*
  - *driveB*
  - *bootindexA*
  - *bootindexB*
- *qxl-vga*
  - *ram_size*
  - *vram_size*
  - *버전*
  - *cmdlog*
  - *addr*

문자 장치

-chardev 백엔드,id=<id>[,<options>]

지원되는 백엔드는 다음과 같습니다.

- *null,id=<id> - null 장치*

- *socket,id=<id>,port=<port>[,host=<host>][,ipv4][,ipv6][,nodelay][,server][,nowait][,telnet] - tcp socket*

- *socket,id=<id>,path=<path>[,server][,nowait][,telnet] - unix socket*

- *파일,id=<id>,path=<path> - 파일에 trafit.*

- *stdio,id=<id> - standard i/o*

- *spicevmc,id=<id>,name=<name> - spice 채널*

USB 활성화

-SAML

19.7. LINUX/MULTIBOOT BOOT

이 섹션에서는 Linux 및 다중 부팅 부팅에 대한 정보를 제공합니다.

커널 파일

-kernel <bzImage>

참고: 다중 부팅 이미지는 지원되지 않습니다.

RAM 디스크

-initrd <file>

명령줄 매개 변수

-append <cmdline>

19.8. 전문가 옵션

이 섹션에서는 전문가 옵션에 대한 정보를 제공합니다.

KVM 가상화

-enable-kvm

QEMU-KVM은 KVM 가상화만 지원하며 사용 가능한 경우 기본적으로 사용됩니다. -enable-kvm이 사용되고 KVM을 사용할 수 없는 경우 qemu-kvm이 실패합니다. 그러나 -enable-kvm이 사용되지 않고 KVM을 사용할 수 없는 경우 qemu-kvm 은 TCG 모드에서 실행되지 않으며 지원되지 않습니다.

커널 모드 PIT Reinjection 비활성화

-no-kvm-pit-reinjection

no shutdown

-no-shutdown

재부팅 없음

-no-reboot

직렬 포트, 모니터, QMP

-serial <dev>

-monitor <dev>

-qmp <dev>

지원되는 장치는 다음과 같습니다.

- stdio - 표준 입력/출력
- null - null 장치

- *file:<filename> - 파일로 출력됩니다.*
- *TCP:[<host>]:<port>[,server][,nowait][,nodelay] - TCP Net console*
- *Unix:<path>[,server][,nowait] - Unix 도메인 소켓.*
- *Mon:<dev_string> - 위의 모든 장치도 멀티플렉스 모니터에 사용됩니다.*
- *None - disable, -serial에만 유효합니다.*
- *chardev:<id> - -chardev로 생성된 문자 장치입니다.*

리디렉션 모니터링

-mon <chardev_id>[,mode=[readline|control]][,default=[on|off]]

수동 CPU 시작

-S

RTC

-rtc [base=utc|localtime|date][,clock=host|vm][,driftfix=none|slew]

Watchdog

-watchdog 모델

워치독 재작업

-watchdog-action <action>

게스트 메모리 백업

-mem-prealloc -mem-path /dev/hugepages

SMBIOS Entry

-smbios type=0[,vendor=<str>][,<version=str>][,date=<str>][,release=%d.%d]

- SMBIOS type=1[,manufacturer=<str>][,product=<str>][,version=<str>][,serial=<str>][,uuid=<uuid>][,sku=<str>][,family=<str>] [,family=<str>]

19.9. 도움말 및 정보 옵션

이 섹션에서는 도움말 및 정보 옵션에 대한 정보를 제공합니다.

help

-h

-help

버전

-version

오디오 도움말

-audio-help

19.10. 기타 옵션

이 섹션에서는 기타 옵션에 대한 정보를 제공합니다.

Migration

-incoming

기본 설정 없음

-nodefconfig

-nodefaults

***-nodefaults** 없이 실행은 지원되지 않습니다.*

장치 설정 파일

-readconfig <file>

-writeconfig <file>

로드된 저장 상태

-loadvm <file>

20장. 도메인 XML 조작

이 섹션에서는 도메인을 나타내는 데 사용되는 **XML** 형식에 대해 설명합니다. 여기서 **domain** 이라는 용어는 모든 게스트 가상 시스템에 필요한 **root <도메인>** 요소를 나타냅니다. 도메인 **XML**에는 두 가지 속성이 있습니다. **type** 은 도메인 실행에 사용되는 하이퍼바이저를 지정합니다. 허용되는 값은 드라이버별로 고유하지만 **KVM** 등을 포함합니다. **ID** 는 실행 중인 게스트 가상 머신의 고유 정수 식별자입니다. 비활성 머신에는 **id** 값이 없습니다. 이 장의 섹션은 도메인 **XML**의 구성 요소를 다룹니다. 이 설명서의 추가 장에서는 도메인 **XML**을 조작해야 할 때 이 장을 참조할 수 있습니다.

참고

이 장에서는 *libvirt 업스트림 문서* 를 기반으로 합니다.

20.1. 일반 정보 및 메타데이터

이 정보는 도메인 **XML**의 다음 부분에 있습니다.

그림 **20.1.** 도메인 **XML** 메타데이터

```
<domain type='xen' id='3'>
  <name>fv0</name>
  <uuid>4dea22b31d52d8f32516782e98ab3fa0</uuid>
  <title>A short description - title - of the domain</title>
  <description>Some 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>
```

도메인 **XML**의 이 섹션의 구성 요소는 다음과 같습니다.

표 **20.1.** 일반 메타데이터 요소

element	설명
<name>	가상 머신의 이름을 할당합니다. 이 이름은 alpha-numeric 문자로만 구성되어야 하며 단일 호스트 물리적 시스템의 범위 내에서 고유해야 합니다. 종종 영구 구성 파일을 저장하기 위한 파일 이름을 형성하는 데 사용됩니다.

element	설명
<uuid>	가상 시스템의 전역적으로 고유한 식별자를 할당합니다. 형식은 RFC 4122 호환, eg **3e3fce45-4f53-4fa7-bb32-11f34168b82b** 여야 합니다. 새 시스템을 정의/생성할 때 생략된 경우 임의의 UUID가 생성됩니다. UUID에 sysinfo 사양을 제공할 수도 있습니다.
<title>	**제목:** 도메인에 대한 간단한 설명을 위한 공간이 생성됩니다. 제목에는 줄 바꿈이 포함되어 있지 않습니다.
<description>	제목과 달리 이 데이터는 libvirt에서 사용하지 않으며 사용자가 표시하려는 정보를 포함할 수 있습니다.
<metadata>	애플리케이션에서 사용자 지정 메타데이터를 XML 노드/트리 형식으로 저장하는 데 사용할 수 있습니다. 애플리케이션에서는 네임스페이스당 최상위 요소만 있는 XML 노드/트리에서 사용자 지정 네임스페이스를 사용해야 합니다(애플리케이션에 구조가 필요한 경우 네임스페이스 요소에 하위 요소가 있어야 함)

20.2. 운영 체제 부팅

*가상 머신을 각각 고유한 장단점으로 부팅하는 방법에는 여러 가지가 있습니다. 각 섹션은 **BIOS** 부트 로더, 호스트 물리적 머신 부트 로더 및 직접 커널 부팅 의 하위 섹션에 설명되어 있습니다.*

20.2.1. BIOS Boot loader

***BIOS**를 통한 부팅은 전체 가상화를 지원하는 하이퍼바이저에 사용할 수 있습니다. 이 경우 **BIOS**는 부팅 순서 우선 순위(**floppy, harddisk, cdrom, network**)가 부팅 이미지를 획득/검색할 위치를 결정합니다. 도메인 **XML**의 **OS** 섹션에는 다음과 같은 정보가 포함되어 있습니다.*

*그림 **20.2. BIOS** 부트 로더 도메인 **XML***

```
...
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <boot dev='hd'/>
  <boot dev='cdrom'/>
  <bootmenu enable='yes'/>
  <smbios mode='sysinfo'/>
  <bios useserial='yes' rebootTimeout='0'/>
</os>
...
```

*도메인 **XML**의 이 섹션의 구성 요소는 다음과 같습니다.*

표 20.2. BIOS 부트 로더 요소

element	설명
<type>	게스트 가상 머신에서 부팅할 운영 체제 유형을 지정합니다. H **VM** 은 OS가 베어 메탈에서 실행되도록 설계된 것이므로 완전한 가상화가 필요하다는 것을 나타냅니다. **Linux** 는 Xen 3 하이퍼바이저 게스트 ABI를 지원하는 OS를 나타냅니다. **arch** 는 CPU 아키텍처를 가상화에 지정하는 것과 시스템 유형을 참조하는 **시스템** 등 두 가지 선택적 특성도 있습니다. 자세한 내용은 *드라이버 기능을* 참조하십시오.
<Loader>	도메인 생성 프로세스를 지원하는 데 사용되는 펌웨어 부분을 나타냅니다. Xen 완전히 가상화된 도메인 사용에만 필요합니다.
<boot>	**fd**,**hd**,**cdrom** 또는 **network** 값 중 하나를 사용하고 다음 부팅 장치를 지정하는 데 사용됩니다. 부팅 요소는 차례로 시도하도록 부팅 장치의 우선 순위 목록을 설정하기 위해 여러 번 반복할 수 있습니다. 동일한 유형의 여러 장치가 버스 순서를 유지하면서 목표에 따라 정렬됩니다. 도메인을 정의한 후 libvirt에서 반환된 XML 구성(virDomainGetXMLDesc)는 정렬된 순서로 장치를 나열합니다. 정렬되면 첫 번째 장치가 부팅 가능한 것으로 표시됩니다. 자세한 내용은 *BIOS 부트로더* 를 참조하십시오.
<bootmenu>	게스트 가상 머신 시작 시 대화형 부팅 메뉴 프롬프트를 활성화할지 여부를 결정합니다. **enable** 속성은 **yes** 또는 **no** 일 수 있습니다. 지정하지 않으면 하이퍼바이저 기본값이 사용됩니다.
<SMBIOS>	게스트 가상 시스템에서 SMBIOS 정보를 표시하는 방법을 결정합니다. **호스트**물리적 시스템의 SMBIOS 값에서 UUID를 제외한 모든 블록 0 및 Block 1을 **에뮬레이션** 하여 **mode** 속성을 지정해야 합니다. virConnectGetSysinfo 호출을 사용하여 복사되는 값을 확인할 수 있습니다. **지정하지 않으면 하이퍼바이저 기본 설정이 사용됩니다.**

element	설명
<BIOS>	이 요소에는 가능한 값이 **yes** 또는 **no** 인 attribute **useserial** 이 있습니다. 이 특성은 직렬 포트에서 BIOS 메시지를 볼 수 있는 직렬 그래픽 어댑터를 활성화하거나 비활성화합니다. 따라서 직렬 포트가 정의되어 있어야 합니다. 또 다른 속성이 있습니다. **rebootTimeout** 은 부팅이 실패하는 경우 게스트 가상 머신이 다시 부팅을 시작하는지 여부와 이후에 부팅을 시작하는지 여부를 제어합니다. 값은 밀리초 단위이며 최대 **65535** 이고 특수 값**-1** 은 재부팅을 비활성화합니다.

20.2.2. 호스트 물리적 시스템 부팅 로더

*반가상화를 사용하는 하이퍼바이저는 일반적으로 **BIOS**를 에뮬레이션하지 않고 호스트 물리적 머신이 운영 체제 부팅을 담당합니다. 이렇게 하면 호스트 물리적 시스템에서 **pseudo-bootloader**를 사용하여 게스트 가상 머신의 커널을 선택할 수 있습니다. 예를 들어, **Xen**을 통한 **pygrub**이 있습니다.*

그림 20.3. 호스트 물리적 시스템 부트 로더 도메인 XML

```
...
<bootloader>/usr/bin/pygrub</bootloader>
<bootloader_args>--append single</bootloader_args>
...
```

*도메인 **XML**의 이 섹션의 구성 요소는 다음과 같습니다.*

표 20.3. BIOS 부트 로더 요소

element	설명
<bootloader>	호스트 물리적 머신 OS에서 실행 가능한 부트 로더에 대한 정규화된 경로를 제공합니다. 이 부트 로더는 부팅할 커널을 선택합니다. 부트 로더의 필요한 출력은 사용 중인 하이퍼바이저에 따라 다릅니다.
<bootloader_args>	명령줄 인수를 부트 로더에 전달할 수 있음(선택 사항)

20.2.3. 직접 커널 부팅

*새 게스트 가상 머신 **OS**를 설치할 때 호스트 물리적 머신 **OS**에 저장된 커널 및 **initrd**에서 직접 부팅하여 명령줄 인수를 설치 프로그램에 직접 전달할 수 있습니다. 이 기능은 일반적으로 반가상화 및 전체 가상화 게스트 가상 머신 모두에서 사용할 수 있습니다.*

그림 20.4. 커널 부팅 직접

```
...
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <kernel>/root/f8-i386-vmlinuz</kernel>
  <initrd>/root/f8-i386-initrd</initrd>
  <cmdline>console=ttyS0 ks=http://example.com/f8-i386/os/</cmdline>
  <dtb>/root/ppc.dtb</dtb>
</os>
...
```

도메인 XML의 이 섹션의 구성 요소는 다음과 같습니다.

표 20.4. 직접 커널 부팅 요소

element	설명
<type>	BIOS 부팅 섹션에 설명된 것과 동일합니다.
<Loader>	BIOS 부팅 섹션에 설명된 것과 동일합니다.
<kernel>	호스트 물리적 머신 OS에서 커널 이미지의 정규화된 경로를 지정합니다.
<initrd>	호스트 물리적 머신 OS의 (선택 사항) ramdisk 이미지의 정규화된 경로를 지정합니다.
<cmdline>	부팅 시 커널(또는 설치 프로그램)에 전달할 인수를 지정합니다. 이는 종종 대체 기본 콘솔(예: 직렬 포트) 또는 설치 미디어 소스 / kickstart 파일을 지정하는 데 사용됩니다.

20.3. SMBIOS 시스템 정보

일부 하이퍼바이저는 게스트 가상 머신에 제공되는 시스템 정보를 제어할 수 있습니다(예: 하이퍼바이저를 사용하여 SMBIOS 필드를 채운 후 게스트 가상 머신의 midecode 명령을 사용하여 검사할 수 있음). 선택적 sysinfo 요소는 이러한 모든 범주의 정보를 다룹니다.

그림 20.5. SMBIOS 시스템 정보

```
...
<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> 요소에는 하위 요소의 레이아웃을 결정하는 필수 특성 유형이 있으며 다음과 같이 정의할 수 있습니다.

- **SMBIOS -** 하위 요소가 특정 **SMBIOS** 값을 호출하면 **<os>** 요소의 하위 요소와 함께 사용되는 경우 게스트 가상 머신에 영향을 미칩니다. **sysinfo**의 각 하위 요소 이름은 **SMBIOS** 블록이며 이러한 요소 내에 블록 내의 필드를 설명하는 항목 요소 목록이 될 수 있습니다. 다음 블록 및 항목이 인식됩니다.
  - **BIOS -** 이는 블록 **0**의 **SMBIOS**이며 공급업체,버전,날짜 및 릴리즈에서 가져온 입력 이름이 제공됩니다.
  - **<시스템> -** 이는 **SMBIOS** 블록 **1**이며, 제조업체,제품, 버전 ,버전,직렬,**sku, family** 에서 가져온 입력 이름이 있습니다. **uuid** 항목이 최상위 **uuid** 요소와 함께 제공되는 경우 두 값이 일치해야 합니다.

20.4. CPU 할당

그림 20.6. CPU 할당

```
<domain>
  ...
  <vcpu placement='static' cpuset="1-4,^3,6" current="1">2</vcpu>
  ...
</domain>
```

<cpu> 요소는 게스트 가상 머신 운영 체제에 할당된 최대 가상 **CPU(vCPU)** 수를 **1**에서 하이퍼바이저

에서 지원하는 최대값이어야 합니다. 이 요소에는 도메인 프로세스 및 가상 CPU를 기본적으로 고정할 수 있는 물리적 CPU 번호의 쉼표로 구분된 목록인 선택적 cpuset 속성이 포함될 수 있습니다.

cputune 특성을 사용하여 도메인 프로세스 및 가상 CPU의 고정 정책을 별도로 지정할 수 있습니다. emulatorpin 속성이 <cputune> 에 지정되면 <vcpu> 에서 지정한 cpuset 값이 무시됩니다.

마찬가지로 vcpupin 에 대한 값을 설정한 가상 CPU로 인해 cpuset 설정이 무시됩니다. vcpupin 이 지정되지 않은 가상 CPU는 cpuset 에 의해 지정된 물리적 CPU에 고정됩니다. cpuset 목록의 각 요소는 단일 CPU 번호, CPU 번호 범위 또는 캐럿(^) 다음에 이전 범위에서 제외할 CPU 번호입니다. current 속성을 사용하여 최대 가상 CPU 수를 활성화할지 여부를 지정할 수 있습니다.

선택적 특성 배치 를 사용하여 도메인 프로세스의 CPU 배치 모드를 지정할 수 있습니다. 배치 를 정적 또는 auto 로 설정할 수 있습니다. <vcpu placement='auto'> 를 설정하면 시스템은 numad를 쿼리하고 <numatune> 태그에 지정된 설정을 사용하고 <vcpu> 의 다른 설정을 무시합니다. <vcpu placement='static'> 을 설정하면 시스템은 <numatune> 의 설정 대신 <vcpu 배치> 태그에 지정된 설정을 사용합니다.

20.5. CPU 튜닝

그림 20.7. CPU 튜닝

```
<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>
```

모두 선택 사항이지만 도메인 XML의 이 섹션의 구성 요소는 다음과 같습니다.

표 20.5. CPU 튜닝 요소

element	설명
<cputune>	도메인의 CPU 튜닝 가능 매개변수에 대한 세부 정보를 제공합니다. 이는 선택 사항입니다.
<vcpupin>	도메인의 VCPU가 고정되는 호스트 물리적 시스템의 물리적 CPU를 지정합니다. 이 값을 생략하고 **<vcpu>** 요소의 **cpuset** 속성이 지정되지 않은 경우 vCPU는 기본적으로 모든 물리적 CPU에 고정됩니다. 여기에는 두 개의 필수 속성이 포함되어 있습니다. **vcpu** 는 **id** 를 지정하고 **cpuset** 속성은 **<vcpu>** 요소의 **cpuset** 속성과 동일합니다.
<emulatorpin>	vcpu를 포함하지 않는 도메인의 하위 집합인 호스트 물리적 시스템 CPU, "emulator" 중 어느 것이 고정되는지를 지정합니다. 이 값을 생략하고 요소 **<vcpu>** 의 특성 **cpuset** 가 지정되지 않은 경우 기본적으로 모든 물리적 CPU에 "emulator"가 고정되어 있습니다. 고정할 물리적 CPU를 지정하는 **cpuset** 라는 하나의 필수 특성이 포함되어 있습니다. 요소 **<vcpu>** 의 특성 **배치 자동** 경우 **emulatorpin** 은 허용되지 않습니다.
<공유>	도메인의 비례 가중치 공유를 지정합니다. 이 값을 생략하면 운영 체제에 고유한 기본값이 설정됩니다. 값에 대한 단위가 없는 경우 다른 게스트 가상 시스템의 설정을 기준으로 계산됩니다. 예를 들어 게스트 가상 시스템이 2048로 구성된 경우 값이 1024인 게스트 가상 머신의 처리 시간이 두 배 증가합니다.
<기간>	적용 간격을 마이크로초로 지정합니다. **기간을** 사용하여 각 도메인의 vcpu는 런타임 시 할당된 할당량보다 더 많이 사용할 수 없습니다. 이 값은 다음 범위 내에 있어야 합니다: **1000-1000000**. 값이 **0 인 period** >는 값이 없음을 의미합니다.
<quota>	마이크로초에 허용되는 최대 대역폭을 지정합니다. **할당량** 을 음수 값으로 사용하는 도메인은 도메인에 무한 대역폭이 있으며 대역폭이 제어되지 않음을 나타냅니다. 값은**1000 - 18446744073709551** 또는 **0** 미만이어야 합니다. 값이 **0** 인 **할당량** 은 값이 없음을 의미합니다. 이 기능을 사용하여 모든 vcpu가 동일한 속도로 실행되도록 할 수 있습니다.
<emulator_period>	적용 간격을 마이크로초로 지정합니다. **<emulator_period>** 내에서 도메인의 에뮬레이터 스레드(vcpus를 제외하고) 에뮬레이터 스레드는 런타임의 **<emulator_quota>** 값 이상을 사용할 수 없습니다. **<emulator_period>** 값은 다음 범위에 있어야 합니다. **1000 - 1000000**. 값이 **0** 인 **<emulator_period>** 는 값이 없음을 의미합니다.

element	설명
<emulator_quota>	도메인의 에뮬레이터 스레드(vcpu 제외)에 대해 허용되는 최대 대역폭을 지정합니다. **<에뮬레이터_quota>** 값이 음수인 도메인은 도메인에 에뮬레이터 스레드(vcpus 제외)에 대한 무한 대역폭이 있으며 대역폭이 제어되지 않음을 나타냅니다. 값은 **1000 - 18446744073709551** 또는 **0** 미만이어야 합니다. 값이 **0** 인 **<emulator_quota>** 는 값이 없음을 의미합니다.

20.6. 메모리 백업

메모리 지원을 통해 하이퍼바이저는 게스트 가상 머신 내의 대규모 페이지를 올바르게 관리할 수 있습니다.

*선택적 **<memoryBacking>** 요소에는 **<hugepages>** 요소가 설정될 수 있습니다. 이는 하이퍼바이저에 일반 기본 페이지 크기 대신 **hugepages**를 사용하여 메모리를 할당해야 함을 나타냅니다.*

*그림 **20.8.** 메모리 백업*

```
<domain>
  ...
  <memoryBacking>
    <hugepages/>
  </memoryBacking>
  ...
</domain>
```

20.7. 메모리 튜닝

*그림 **20.9.** 메모리 튜닝*

```
<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>
```

*모두 선택 사항이지만 도메인 **XML**의 이 섹션의 구성 요소는 다음과 같습니다.*

*표 **20.6.** 메모리 튜닝 요소*

element	설명
<memtune>	도메인의 메모리 조정 가능한 매개변수에 대한 세부 정보를 제공합니다. 이 값을 생략하면 기본값으로 OS에서 제공하는 기본값으로 설정됩니다. 이 매개변수는 제한을 설정할 때 전체 프로세스에 적용되므로 게스트 가상 머신 RAM, 게스트 가상 머신 비디오 RAM을 추가하고 일부 메모리 오버헤드를 허용해야 합니다. 마지막 부분은 하나의 평가판과 오류를 결정하기가 어렵습니다. 각 튜닝 가능 항목의 경우 **<메모리>** 와 동일한 값을 사용하여 입력에 있는 단위를 지정할 수 있습니다. 이전 버전과의 호환성을 위해 출력은 항상 KiB 단위입니다.
<hard_limit>	게스트 가상 머신에서 사용할 수 있는 최대 메모리입니다. 이 값의 **단위는 kibibytes** (24 바이트의 블록)로 표현됩니다.
<soft_limit>	이는 메모리 경합 중에 시행되는 메모리 제한입니다. 이 값의 **단위는** kibibytes (24 바이트의 블록)로 표현됩니다.
<swap_hard_limit>	이는 게스트 가상 머신에서 사용할 수 있는 최대 메모리 및 스왑 메모리입니다. 이 값의 **단위는** kibibytes (24바이트 블록)로 표시됩니다. 이 값은 **<hard_limit>** 값보다 많이 제공해야 합니다.
<min_guarantee>	이는 게스트 가상 머신에 대해 보장된 최소 메모리 할당입니다. 이 값의 단위는 kibibytes (24 바이트의 블록)로 표현됩니다.

20.8. NUMA 노드 튜닝

*일반적인 관리 툴을 사용하여 **NUMA** 노드 튜닝을 완료하면 다음과 같은 도메인 **XML** 매개 변수가 적용됩니다.*

*그림 **20.10. NUMA** 노드 튜닝*

```
>
<domain>
  ...
  <numatune>
    <memory mode="strict" nodeset="1-4,^3"/>
  </numatune>
  ...
</domain>
```

모두 선택 사항이지만 도메인 XML의 이 섹션의 구성 요소는 다음과 같습니다.

표 20.7. NUMA 노드 튜닝 요소

element	설명
<numatune>	도메인 프로세스에 대한 NUMA 정책을 제어하여 NUMA 호스트 물리적 시스템의 성능을 튜닝하는 방법에 대한 세부 정보를 제공합니다.
<memory>	NUMA 호스트 물리적 시스템에서 도메인 프로세스에 메모리를 할당하는 방법을 지정합니다. 여기에는 몇 가지 선택적 속성이 포함되어 있습니다. 특성 **모드는 interleave**,**strict** 또는 **preferred** 중 하나입니다. 값을 지정하지 않으면 기본값은 **strict** 입니다. 특성 **nodeset** 은 **<vcpu>** 요소의 **cpuset** 속성과 동일한 구문을 사용하여 NUMA 노드를 지정합니다. 특성 **배치**를 사용하여 도메인 프로세스의 메모리 배치 모드를 나타낼 수 있습니다. 값은 **static** 또는 **auto** 일 수 있습니다. 특성 **<nodeset>** 이 지정된 경우 기본값은 **<vcpu>** 또는 **정적** 의 **<배치>** 로 설정됩니다. **Auto** 는 도메인 프로세스에서 numad 쿼리에서 반환된 권고 노드 집합의 메모리만 할당하며 특성 노드 집합의 값은 지정된 경우 무시됩니다. **vcpu** 의 속성 **배치** 가 **자동** 이고, 특성 **<numatune>** 이 지정되지 않은 경우 **<배치> 자동** 및 모드 Strict인 기본 numatune이 암시적으로 추가됩니다.

20.9. 블록 I/O 튜닝

그림 20.11. 블록 I/O 튜닝

```
<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>
```

모두 선택 사항이지만 도메인 XML의 이 섹션의 구성 요소는 다음과 같습니다.

표 20.8. 블록 I/O 튜닝 요소

element	설명
<blkiotune>	이 선택적 요소는 도메인의 Blkio cgroup 조정 가능 매개변수를 조정하는 기능을 제공합니다. 이 값을 생략하면 기본값으로 OS에서 제공하는 기본값으로 설정됩니다.
<weight>	이 선택적 가중치 요소는 게스트 가상 머신의 전체 I/O 가중치입니다. 값은 100~1000 범위 내에 있어야 합니다.
<device>	도메인에는 도메인에서 사용 중인 각 호스트 물리적 시스템 블록 **<장치에>** 대한 가중치를 추가로 조정하는 여러 장치 요소가 있을 수 있습니다. 여러 게스트 가상 머신 디스크는 단일 호스트 물리적 시스템 블록 장치를 공유할 수 있습니다. 또한 동일한 호스트 물리적 머신 파일 시스템 내의 파일에 의해 지원되므로 이 튜닝 매개변수는 각 게스트 가상 머신 디스크 장치(단일 **<디스크에>**적용할 수 있는**<iotune>** 요소에 제한) 대신 글로벌 도메인 수준에 있습니다. 각 장치 요소에는 두 개의 필수 하위 요소, 장치의 절대 경로를 설명하는 경로, 장치의 절대 경로를 설명하는 가중치, 허용 가능한 범위 100~1000이 있는 해당 장치의 상대적 가중치가 있습니다. **<> <>**

20.10. 리소스 파티셔닝

하이퍼바이저를 사용하면 가상 머신을 리소스 파티션에 배치할 수 있으며 파티션은 중첩될 수 있습니다. <리소스 요소는 리소스> 파티션과 관련된 구성을 함께 그룹화합니다. 현재 콘텐츠가 도메인을 배치할 리소스 파티션의 경로를 정의하는 하위 요소 파티션을 지원합니다. 파티션이 나열되어 있지 않으면 도메인이 기본 파티션에 배치됩니다. 게스트 가상 머신을 시작하기 전에 앱이/관리자로 파티션이 존재하는지 확인해야 합니다. (하이퍼바이저 특정) 기본 파티션만 기본적으로 존재한다고 가정할 수 있습니다.

그림 20.12. 리소스 파티션 설정

```
<resource>
  <partition>/virtualmachines/production</partition>
</resource>
```

현재 QEMU 및 LXC 드라이버에서 리소스 파티션을 지원합니다. 이 드라이버는 마운트된 모든 컨트롤러의 cgroup 디렉터리에 파티션 경로를 매핑합니다.

20.11. CPU 모델 및 토폴로지

이 섹션에서는 **CPU** 모델의 요구 사항을 다룹니다. 모든 하이퍼바이저에는 기본적으로 **CPU** 기능 게스트가 표시되는 자체 정책이 있습니다. **QEMU/KVM**에서 게스트에 제공하는 **CPU** 기능 세트는 게스트 가상 머신 구성에서 선택한 **CPU** 모델에 따라 다릅니다. **qemu32** 및 **qemu64** 는 기본 **CPU** 모델이지만 다른 모델(추가 기능 포함)을 사용할 수 있습니다. 각 모델 및 해당 토폴로지는 도메인 **XML**의 다음 요소를 사용하여 지정됩니다.

그림 **20.13. CPU** 모델 및 토폴로지 예 **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>
```

그림 **20.14. CPU** 모델 및 토폴로지 예 **2**

```
<cpu mode='host-model'>
   <model fallback='forbid'/>
   <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

그림 **20.15. CPU** 모델 및 토폴로지 예 **3**

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

CPU 모델이나 해당 기능에 제한이 없는 경우 다음과 같은 간단한 **cpu** 요소가 사용될 수 있습니다.

그림 **20.16. CPU** 모델 및 토폴로지 예 **4**

```
<cpu>
   <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

도메인 **XML**의 이 섹션의 구성 요소는 다음과 같습니다.

표 **20.9. CPU** 모델 및 토폴로지 요소

element	설명
<cpu>	이 요소에는 vCPU 기능 세트에 대한 모든 매개변수가 포함되어 있습니다.

element	설명
<match>	**<cpu>** 요소에 표시된 기능이 사용 가능한 vCPU와 얼마나 밀접하게 일치해야 하는지 지정합니다. **<topology>** 가 **<cpu>** 요소에 중첩된 유일한 요소인 경우 **match** 특성을 생략할 수 있습니다.**match** 속성에 사용 가능한 값은 다음과 같습니다. ● **Minimum** - 나열된 기능은 최소 요구사항입니다. vCPU에 더 많은 기능을 사용할 수 있는 기능이 더 있을 수 있지만 이는 허용되는 최소 기능입니다. 최소 요구 사항이 충족되지 않으면 이 값이 실패합니다. ● **정확한** - 게스트 가상 머신에 제공되는 가상 CPU는 지정된 기능과 정확히 일치해야 합니다. 일치하는 항목이 없으면 오류가 발생합니다. ● **Strict** - 호스트 물리적 머신 CPU가 사양과 정확히 일치하지 않는 한 게스트 가상 머신이 생성되지 않습니다. **<cpu>** 요소에서 **match** 속성이 생략되면 기본 설정 **match='exact'** 가 사용됩니다.

element	설명
<mode>	이 선택적 특성은 게스트 가상 머신 CPU를 호스트 물리적 머신 CPU에 최대한 가깝게 구성하는 데 더 쉽게 사용할 수 있습니다. mode 특성에 가능한 값은 다음과 같습니다. ● **Custom** - CPU가 게스트 가상 머신에 제공되는 방법을 설명합니다. **mode** 속성이 지정되지 않은 경우 기본 설정입니다. 이 모드를 사용하면 게스트 가상 머신이 부팅되는 호스트 물리적 시스템과 관계없이 영구 게스트 가상 머신이 동일한 하드웨어를 볼 수 있습니다. ● **호스트 모델 -** 기본적으로 호스트 물리적 머신 CPU 정의를 기능 XML에서 도메인 XML로 복사하는 바로 가기입니다. 도메인을 시작하기 직전에 CPU 정의가 복사되므로 서로 다른 호스트 물리적 시스템에서 동일한 XML을 사용할 수 있으며 각 호스트 물리적 시스템이 지원하는 게스트 가상 머신 CPU를 계속 제공할 수 있습니다. **match** 속성과 모든 기능 요소를 이 모드에서 사용할 수 없습니다. 자세한 내용은 libvirt 도메인 XML CPU 모델을참조하십시오. ● 이 모드에서 **호스트 전달을 통해** 게스트 가상 머신에 표시되는 CPU는 libvirt 내에 오류를 유발하는 요소를 포함하여 호스트 물리적 시스템 CPU와 정확히 동일합니다. 이 모드의 단점은 게스트 가상 머신 환경을 다른 하드웨어에서 재현할 수 없으므로 이 모드를 신중하게 사용하는 것이 좋습니다. 이 모드에서는 **모델** 이나 **기능** 요소가 허용되지 않습니다. ● **host-model** 및 **host-passthrough** 모드에서는 virDomainGetXMLDesc API를 호출할 때 VIR_DOMAIN_XML_UPDATE_CPU 플래그를 지정하여 현재 호스트 물리적 시스템에서 사용되는 실제(호스트-passthrough 모드에서 적용) CPU 정의를 결정할 수 있습니다. 다른 하드웨어에서 사용할 수 있는 게스트 가상 머신을 실행할 때 다른 하드웨어가 표시되고 다른 기능이 있는 호스트 물리적 시스템 간에 마이그레이션될 예정인 게스트 가상 머신을 실행하는 경우 이 출력을 사용하여 보다 강력한 마이그레이션을 위해 사용자 지정 모드로 XML을 다시 작성할 수 있습니다.

element	설명
<model>	게스트 가상 머신에서 요청한 CPU 모델을 지정합니다. 사용 가능한 CPU 모델 및 해당 정의 목록은 libvirt의 data 디렉터리에 설치된 **cpu_map.xml** 파일에서 확인할 수 있습니다. 하이퍼바이저가 정확한 CPU 모델을 사용할 수 없는 경우 libvirt는 CPU 기능 목록을 유지하면서 하이퍼바이저에서 지원하는 가장 가까운 모델로 자동 대체됩니다. 선택적 **fallback** 속성은 이 동작을 금지하는 데 사용할 수 있으며, 이 경우 지원되지 않는 CPU 모델을 요청하는 도메인을 시작하려고 하면 실패합니다. fallback 속성에 지원되는 값은 **allow** (기본값) 및 **forbid** 입니다. 선택적 **vendor_id** 특성은 게스트 가상 머신에 표시된 벤더 ID를 설정하는 데 사용할 수 있습니다. 정확히 12자 길이여야 합니다. 설정되지 않은 경우 호스트 물리적 시스템의 벤더 ID가 사용됩니다. 일반적인 가능한 값은 **AuthenticAMD** 및 **GenuineIntel** 입니다.
<vendor>	게스트 가상 머신에서 요청한 CPU 벤더를 지정합니다. 이 요소가 누락된 경우 게스트 가상 머신은 벤더에 관계없이 지정된 기능과 일치하는 CPU에서 실행됩니다. 지원되는 공급 업체 목록은 **cpu_map.xml** 에서 확인할 수 있습니다.
<토폴로지>	게스트 가상 머신에 제공된 가상 CPU의 요청된 토폴로지를 지정합니다. 소켓, 코어, 스레드 수, 소켓당 코어 수, 코어당 코어 수, 코어당 3개의 스레드 수를 각각 지정해야 합니다.

element	설명
<기능>	선택한 CPU 모델에서 제공하는 기능을 미세 조정하는 데 사용되는 0개 이상의 요소를 포함할 수 있습니다. 알려진 기능 이름 목록은 CPU 모델과 동일한 파일에서 확인할 수 있습니다. 각 feature 요소의 의미는 다음 값 중 하나로 설정해야 하는 policy 특성에 따라 달라집니다. ● **force** - 호스트 물리적 시스템 CPU에서 실제로 지원되는지 여부에 관계없이 가상이 계속 지원됩니다. ● **require** - 호스트 물리적 머신 CPU에서 기능을 지원하지 않는 한 게스트 가상 머신 생성이 실패합니다. 기본 설정입니다. ● **선택 사항** - 이 기능은 가상 CPU에서 지원되지만 호스트 물리적 머신 CPU에서 지원되는 경우에만 해당합니다. ● **비활성화** - 가상 CPU에서 지원되지 않습니다. ● **forbid** - 호스트 물리적 머신 CPU에서 기능을 지원하는 경우 게스트 가상 머신 생성이 실패합니다.

20.11.1. 게스트 가상 머신 NUMA 토폴로지

<numa> 요소와 도메인 XML에서 다음을 사용하여 게스트 가상 머신 NUMA 토폴로지를 지정할 수 있습니다.

그림 20.17. 게스트 가상 머신 NUMA 토폴로지

```
<cpu>
  <numa>
    <cell cpus='0-3' memory='512000'/>
    <cell cpus='4-7' memory='512000'/>
  </numa>
</cpu>
...
```

각 셀 요소는 NUMA 셀 또는 NUMA 노드를 지정합니다. CPU 는 노드에 포함된 CPU의 CPU 또는 범위를 지정합니다. memory 는 노드 메모리를 kibibytes(24바이트의 블록)로 지정합니다. 각 셀 또는 노드에는 0부터 시작하는 순서에 따라 cellid 또는 nodeid 가 할당됩니다.

20.12. 이벤트 구성

도메인 XML의 다음 섹션을 사용하면 다양한 이벤트에서 수행된 기본 작업을 덮어쓸 수 있습니다.

그림 20.18. 이벤트 구성

```
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<on_lockfailure>poweroff</on_lockfailure>
```

다음 요소 컬렉션을 사용하면 게스트 가상 머신 OS에서 라이프사이클 작업을 트리거할 때 작업을 지정할 수 있습니다. 일반적인 사용 사례는 초기 OS 설치를 수행할 때 재부팅을 강제로 전원 끄기로 처리하는 것입니다. 이를 통해 VM을 첫 번째 설치 후 부팅 시 다시 구성할 수 있습니다.

도메인 XML의 이 섹션의 구성 요소는 다음과 같습니다.

표 20.10. 이벤트 구성 요소

상태	설명
<on_poweroff>	게스트 가상 머신이 전원을 요청할 때 실행할 작업을 지정합니다. 가능한 네 가지 인수를 사용할 수 있습니다. • **destroy** - 이 작업은 도메인을 완전히 종료하고 모든 리소스를 해제합니다. • **재시작** - 이 작업은 도메인을 완전히 종료하고 동일한 구성으로 재시작합니다. • **보존** - 이 작업은 도메인을 완전히 종료하지만 향후 분석을 위해 해당 리소스가 보존됩니다. • **rename-restart** - 이 작업은 도메인을 완전히 종료한 다음 새 이름으로 다시 시작합니다.

상태	설명
<on_reboot>	게스트 가상 머신이 재부팅을 요청할 때 실행할 작업을 지정합니다. 가능한 네 가지 인수를 사용할 수 있습니다. ● **destroy** - 이 작업은 도메인을 완전히 종료하고 모든 리소스를 해제합니다. ● **재시작** - 이 작업은 도메인을 완전히 종료하고 동일한 구성으로 재시작합니다. ● **보존** - 이 작업은 도메인을 완전히 종료하지만 향후 분석을 위해 해당 리소스가 보존됩니다. ● **rename-restart** - 이 작업은 도메인을 완전히 종료한 다음 새 이름으로 다시 시작합니다.
<on_crash>	게스트 가상 머신이 충돌할 때 실행할 작업을 지정합니다. 또한 다음과 같은 추가 작업을 지원합니다. ● **coredump-destroy** - 크래시된 도메인의 코어가 덤프되고, 도메인이 완전히 종료되고, 모든 리소스가 해제됩니다. ● **coredump-restart** - 크래시된 도메인의 코어가 덤프되고 도메인이 동일한 구성 설정으로 다시 시작됩니다. 가능한 네 가지 인수를 사용할 수 있습니다. ● **destroy** - 이 작업은 도메인을 완전히 종료하고 모든 리소스를 해제합니다. ● **재시작** - 이 작업은 도메인을 완전히 종료하고 동일한 구성으로 재시작합니다. ● **보존** - 이 작업은 도메인을 완전히 종료하지만 향후 분석을 위해 해당 리소스가 보존됩니다. ● **rename-restart** - 이 작업은 도메인을 완전히 종료한 다음 새 이름으로 다시 시작합니다.

상태	설명
<on_lockfailure>	잠금 관리자에서 리소스 잠금이 손실될 때 수행할 작업을 지정합니다.Specifies what action should be taken when a lock manager loses resource locks. libvirt에서 다음 작업을 인식하지만 일부 작업은 개별 잠금 관리자에서 지원할 필요는 없습니다. 작업을 지정하지 않으면 각 잠금 관리자가 기본 작업을 수행합니다. 다음 인수를 사용할 수 있습니다. ● **전원 끄기** - 도메인 전원을 강제로 전원을 끕니다. ● **다시 시작** - 도메인을 다시 시작하여 잠금을 다시 시작합니다. ● **pause** - 잠금 문제가 해결될 때 수동으로 재개할 수 있도록 도메인을 일시 중지합니다. ● **무시** - 아무 일도 발생하지 않은 것처럼 도메인을 계속 실행합니다.

20.13. 전원 관리

도메인 XML의 다음 섹션에 영향을 주는 기존 관리 도구를 사용하여 게스트 가상 머신 OS에 대한 BIOS 알림을 강제 활성화하거나 비활성화할 수 있습니다.

그림 20.19. 전원 관리

```
...
<pm>
  <suspend-to-disk enabled='no'/>
  <suspend-to-mem enabled='yes'/>
</pm>
...
```

<pm> 요소는 no 인수를 사용하여 yes 또는 disabled를 사용하여 활성화할 수 있습니다. suspend-to-disk 및 S4 인수 suspend-to-mem ACPI sleep 상태를 사용하여 S3에 대해 BIOS 지원을 구현할 수 있습니다. 아무것도 지정하지 않으면 하이퍼바이저가 기본값으로 유지됩니다.

20.14. 하이퍼바이저 기능

하이퍼바이저는 특정 CPU/시스템 기능을 활성화 또는 비활성화(state='off') 할 수 있습니다 (state='off').

그림 20.20. 하이퍼바이저 기능

```
...
<features>
  <pae/>
  <acpi/>
  <apic/>
  <hap/>
  <privnet/>
  <hyperv>
    <relaxed state='on'/>
  </hyperv>
</features>
...
```

<상태가> 지정되지 않은 경우 <features> 요소 내에 모든 기능이 나열됩니다. 기능 XML을 호출하여 사용 가능한 기능을 찾을 수 있지만 완전히 가상화된 도메인에 대한 공통 세트는 다음과 같습니다.

표 20.11. 하이퍼바이저 기능 요소

상태	설명
<pae>	물리적 주소 확장 모드를 사용하면 32비트 게스트 가상 머신이 4GB 이상의 메모리를 처리할 수 있습니다.
<acpi>	전원 관리에 유용합니다(예: KVM 게스트 가상 시스템을 사용하는 경우 정상적으로 종료해야 함).
<apic>	프로그래밍 가능한 IRQ 관리를 사용할 수 있습니다. 이 요소에는 게스트 가상 머신에 **대한** EOI (End of Interrupt)의 가용성을 설정하고 값이 있는 선택적 속성 **eoi** 가 있습니다.
<ap>	하드웨어에서 사용할 수 있는 경우 Hardware Assisted Paging을 사용할 수 있습니다.
HyperV	Microsoft Windows를 실행하는 게스트 가상 머신의 동작을 개선하기 위해 다양한 기능을 사용할 수 있습니다. 또는 **off** 의 값으로 **완화** 된 선택적 속성을 사용하여 타이머 **에 대한** 완화 제약 조건을 활성화하거나 비활성화합니다.

20.15. 시간 유지

게스트 가상 머신 클럭은 일반적으로 호스트 실제 시스템 클럭에서 초기화됩니다. 대부분의 운영 체제는 하드웨어 시계가 기본 설정인 UTC에 보관될 것으로 예상합니다. Windows 게스트 가상 머신의 경우

게스트 가상 머신을 현지 시간으로 설정해야 합니다.

*그림 **20.21.** 시간 유지*

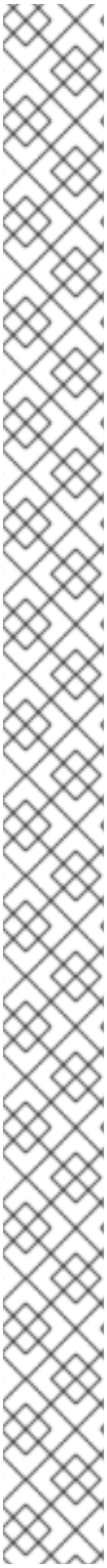
```
...
<clock offset='localtime'>
  <timer name='rtc' tickpolicy='catchup' track='guest'>
    <catchup threshold='123' slew='120' limit='10000'/>
  </timer>
  <timer name='pit' tickpolicy='delay'/>
</clock>
...
```

*도메인 **XML**의 이 섹션의 구성 요소는 다음과 같습니다.*

*표 **20.12.** 요소 유지 시간*

상태	설명

상태	설명
<clock>	**offset** 속성은 4가지 값을 사용하므로 게스트 가상 시스템 시계가 호스트 물리적 시스템에 어떻게 동기화되는지 세부적으로 제어할 수 있습니다. 모든 시간 소스에서 모든 정책을 지원할 필요는 없습니다. ● **UTC** - 부팅 시 클럭을 UTC에 동기화합니다. **UTC** 모드는 조정 특성을 사용하여 제어할 수 있는 **변수** 모드로 변환할 수 있습니다. 값을 **재설정** 하면 변환이 수행되지 않습니다. 숫자 값은 값을 초기 조정으로 사용하여 **변수** 모드로 강제 변환됩니다. 기본 조정은 하이퍼바이저 특정입니다. ● **localtime** - 부팅 시 호스트 물리적 시스템의 구성된 시간대와 게스트 가상 머신 클럭을 동기화합니다. 조정 속성은 'utc' 모드와 동일하게 작동합니다. ● **timezone** - timezone 특성을 사용하여 게스트 가상 머신 클럭을 요청된 시간대에 동기화합니다. ● **변수** - 게스트 가상 머신은 기본 속성에 따라 UTC 또는 localtime을 기준으로 적용된 임의의 오프셋을 제어합니다. UTC(또는 localtime)에 상대적인 delta는 **조정** 특성을 사용하여 초 단위로 지정됩니다. 게스트 가상 머신은 시간이 지남에 따라 RTC를 자유롭게 조정할 수 있으며 다음 재부팅 시 사용할 수 있습니다. 이는 **utc** 및 **localtime** 모드와 대조적입니다(선택적인 특성 **adjustment='reset'**), 여기서 RTC 조정은 각 재부팅 시 손실됩니다. **basis** 속성에는 **utc** (기본값) 또는 **localtime** (localtime)일 수 있습니다. **clock** 요소에는 0개 이상의 **<타이머>** 요소가 있을 수 있습니다.
<타이머>	참고 보기
<빈도>	**name="tsc"** 가 실행되는 빈도를 지정하는 서명되지 않은 정수입니다.
<mode>	**mode** 속성은 **name="tsc" <타이머>** 를 관리하는 방법을 제어합니다. **auto**,**native**,**emulate**,**paravirt**, 또는 **smpsafe** 로 설정할 수 있습니다. 다른 타이머는 항상 에뮬레이션됩니다.
<present>	게스트 가상 머신에서 특정 타이머를 사용할 수 있는지 여부를 지정합니다. **yes** 또는 **no**로 설정할 수 있습니다.



참고

각 **<timer>** 요소에는 **name** 속성이 포함되어야 하며 지정된 이름에 따라 다음 속성이 있을 수 있습니다.

- **<name> -** 수정되는 타이머 를 선택합니다. 사용할 수 있는 값은**kvmclock (QEMU-KVM), pit(QEMU-KVM),** 또는 **rtc(QEMU-KVM)** 또는 **tsc(libxl**만 해당)입니다. 플랫폼은 현재 지원되지 않습니다.

- **track -** 타이머 경로를 지정합니다. 다음 값을 사용할 수 있습니다. 부팅,게스트 또는 벽. **track** 은 **name="rtc"** 에만 유효합니다.

- **tickpolicy -** 게스트 가상 머신에 대한 틱을 삽입하는 데 필요한 데드라인이 누락된 경우를 결정합니다. 다음 값을 할당할 수 있습니다.

  - **delay -will**은 계속해서 정상 속도로 진드기를 전달합니다. 늦은 눈금으로 인해 게스트 가상 머신 시간이 지연됩니다.

  - **catchup -** 누락된 진드기를 찾기 위해 더 높은 속도로 진드를 전달합니다. **catchup**이 완료되면 게스트 가상 머신 시간이 표시되지 않습니다. 또한 각각 양의 정수(**threshold, slew, limit)**의 세 가지 선택적 특성이 있을 수 있습니다.

  - 병합 **-** 누락된 눈금을 하나의 눈금에 병합하여 삽입합니다. 병합 수행 방법에 따라 게스트 가상 머신 시간이 지연될 수 있습니다.

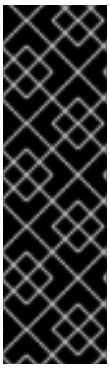
  - 삭제 **-** 누락된 눈금을 버리고 기본 간격 설정에서 향후 삽입을 계속합니다. 손실된 틱을 처리하기 위한 명시적 선언이 없는 한 게스트 가상 머신 시간이 지연될 수 있습니다.

20.16. 장치

이 **XML** 요소 세트는 모두 게스트 가상 머신 도메인에 제공되는 장치를 설명하는 데 사용됩니다. 아래의 모든 장치는 주요 장치 요소의 자식으로 표시됩니다.

다음과 같은 가상 장치가 지원됩니다.

- *virtio-scsi-pci - PCI 버스 스토리지 장치*
- *virtio-9p-pci - PCI 버스 스토리지 장치*
- *virtio-blk-pci - PCI 버스 스토리지 장치*
- *virtio-net-pci - virtio-net이라고도 하는 PCI 버스 네트워크 장치*
- *virtio-serial-pci - PCI 버스 입력 장치*
- *virtio-balloon-pci - PCI 버스 메모리 balloon 장치*
- *virtio-rng-pci - PCI 버스 가상 임의 번호 생성기 장치*



중요

virtio 장치가 32보다 큰 값으로 설정된 벡터 수가 생성되는 경우 장치는 Red Hat Enterprise Linux 6에서 0 값으로 설정되었지만 Enterprise Linux 7에서는 값이 아닌 것처럼 작동합니다. 결과 벡터 설정이 일치하지 않으면 두 플랫폼의 virtio 장치 벡터 수가 33 이상으로 설정된 경우 마이그레이션 오류가 발생합니다. 따라서 벡터 값을 32보다 크게 설정하지 않는 것이 좋습니다. virtio-balloon-pci 및 virtio-rng-pci를 제외한 모든 virtio 장치는 벡터 인수를 허용합니다.

그림 20.22. 장치 - 하위 요소

```
...
<devices>
  <emulator>/usr/lib/xen/bin/qemu-dm</emulator>
</devices>
...
```

<에뮬레이터 요소의 내용은 장치 모델 에뮬레이터> 바이너리에 대한 정규화된 경로를 지정합니다. capabilities XML은 각 특정 도메인 유형 또는 아키텍처 조합에 사용할 권장 기본 에뮬레이터를 지정합니다.

20.16.1. 하드 드라이브, Floppy 디스크, CDROM

*도메인 **XML**의 이 섹션은 디스크처럼 보이는 모든 장치를 지정합니다. 피로, 하드 디스크, **cdrom** 또는 반가상화 드라이버가 디스크 요소를 통해 지정됩니다.*

*그림 **20.23.** 장치 **-** 하드 드라이브, 플로피 디스크, **CDROM***

```
...
 <devices>
   <disk type='file' snapshot='external'>
     <driver name="tap" type="aio" cache="default"/>
     <source file='/var/lib/xen/images/fv0' startupPolicy='optional'>
       <seclabel relabel='no'/>
     </source>
     <target dev='hda' bus='ide'/>
     <iotune>
       <total_bytes_sec>10000000</total_bytes_sec>
       <read_iops_sec>400000</read_iops_sec>
       <write_iops_sec>100000</write_iops_sec>
     </iotune>
     <boot order='2'/>
     <encryption type='...'>
       ...
     </encryption>
     <shareable/>
     <serial>
       ...
     </serial>
   </disk>
     ...
   <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>
   <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>
   <disk type='block' device='cdrom'>
     <driver name='qemu' type='raw'/>
     <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>
  <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>
</devices>
...
```

20.16.1.1. 디스크 요소

<disk> 요소는 디스크를 설명하는 기본 컨테이너입니다. 특성 유형은 **<disk>** 요소와 함께 사용할 수 있습니다. 다음 유형을 사용할 수 있습니다.

- **file**
- 블록
- **dir**
- **network**

자세한 내용은 디스크 항목을 참조하십시오.

20.16.1.2. source 요소

<디스크 type='file'> 의 경우 **file** 속성은 디스크를 포함하는 파일의 정규화된 경로를 지정합니다. **<디스크 type='block'>** 의 경우 **dev** 속성은 디스크로 사용할 호스트 물리적 시스템 장치의 경로를 지정합니다. 파일과 블록 모두를 사용하면 아래에 설명된 하나 이상의 하위 요소 **seclabel** 을 사용하여 해당 소스 파일에 대한 도메인 보안 레이블 지정 정책을 재정의할 수 있습니다. 디스크 유형이 **dir** 이면 **dir** 속성은 디

*스크로 사용할 디렉터리의 정규화된 경로를 지정합니다. 디스크 유형이 네트워크 이면 **protocol** 속성은 요청된 이미지에 액세스할 프로토콜을 지정합니다. 가능한 값은 **nbd,rbd,sheepdog** 또는 **gluster** 입니다.*

*protocol 속성이 **rbd,sheepdog** 또는 **gluster** 이면 사용할 볼륨 및 또는 이미지를 지정하는 추가 속성 이름이 필요합니다. 디스크 유형이 네트워크 인 경우, 소스에 는 **type='dir'** 및 **type='network'** 를 포함하여 연결할 호스트 물리적 시스템을 지정하는 데 사용되는 호스트 하위 요소가 **0**개 이상 있을 수 있습니다. **cdrom** 또는 **coat(Device** 속성)를 나타내는 파일 디스크 유형의 경우 소스 파일에 액세스할 수 없는 경우 디스크로 수행할 작업을 정의할 수 있습니다. 이 작업은 다음 값과 함께 **startupPolicy** 특성을 조작하여 수행됩니다.*

- *어떤 이유로든 누락된 경우 필수적으로 오류가 발생합니다. 이 설정은 기본 설정입니다.*
- ***requisite** 부팅 시 누락된 경우 오류가 발생하는 경우 **migrate/restore/revert**에 누락된 경우 드롭*
- *시작 시도에 누락된 경우 선택적 드롭*

20.16.1.3. mirror 요소

*이 요소는 하이퍼바이저가 **BlockCopy** 작업을 시작한 경우 특성 파일의 **<미러>** 위치가 결국 소스와 동일한 내용이 있고 속성 형식의 파일 형식(소스 형식과 다를 수 있음)을 사용하는 경우 존재합니다. 준비가 된 속성이 있는 경우 디스크가 피벗할 준비가 된 것으로 알려져 있습니다. 그렇지 않으면 디스크를 계속 복사합니다. 현재 이 요소는 출력에서만 유효하므로 입력에서 무시됩니다.*

20.16.1.4. target 요소

***<target>** 요소는 디스크가 게스트 가상 머신 **OS**에 노출되는 버스 / 장치를 제어합니다. **dev** 속성은 논리 장치 이름을 나타냅니다. 지정된 실제 장치 이름은 게스트 가상 머신 **OS**의 장치 이름에 매핑되지 않습니다. 선택적 **bus** 속성은 에뮬레이션할 디스크 장치 유형을 지정합니다. 가능한 값은 일반적인 값 인**scsi,virtio,xen,usb** 또는 **sata** 와 함께 드라이버별로 다릅니다. 생략하면 버스 유형이 장치 이름의 스타일에서 유추됩니다. 예를 들어 **'sda'** 라는 장치는 일반적으로 **SCSI** 버스를 사용하여 내보냅니다. 선택적 속성 트레이 는 이동식 디스크(예: **CD-ROM** 또는 **Floppy** 디스크)의 트레이 상태를 나타냅니다. 값은 열거나 닫을 수 있습니다. 기본 설정은 폐쇄 입니다. 자세한 내용은 대상 항목을 참조하십시오.*

20.16.1.5. iotune

*선택적 **<iotune>** 요소는 장치별 **I/O** 튜닝을 추가로 제공할 수 있는 기능을 제공하며, 장치마다 다를 수 있는 값을 제공합니다(이 요소는 도메인에 전역적으로 적용되는 **blkiotune** 요소에 제한). 이 요소에는 다*

음과 같은 선택적 하위 요소가 있습니다. 하위 요소가 지정되거나 전혀 지정되지 않거나 0 값으로 지정되지 않거나 지정된 하위 항목은 제한이 없음을 나타냅니다.

- <total_bytes_sec> - 초당 총 처리량 제한(바이트)입니다. 이 요소는 <read_bytes_sec> 또는 <write_bytes_sec> 과 함께 사용할 수 없습니다.
- <read_bytes_sec> - 초당 읽기 처리량 제한(바이트)입니다.
- <write_bytes_sec> - 초당 쓰기 처리량 제한(바이트)입니다.
- <total_iops_sec> 초당 총 I/O 작업 수입니다. 이 요소는 <read_iops_sec> 또는 <write_iops_sec> 과 함께 사용할 수 없습니다.
- <read_iops_sec> 초당 읽기 I/O 작업 수입니다.
- <write_iops_sec> - 초당 쓰기 I/O 작업 수입니다.

20.16.1.6. 드라이버

선택적 <드라이버> 요소를 사용하면 디스크를 제공하는 데 사용되는 하이퍼바이저 드라이버와 관련된 추가 세부 정보를 지정할 수 있습니다. 다음 옵션을 사용할 수 있습니다.

- 하이퍼바이저가 여러 백엔드 드라이버를 지원하는 경우 name 속성은 기본 백엔드 드라이버 이름을 선택하는 반면 선택적 type 속성은 하위 유형을 제공합니다. 사용 가능한 유형 목록은 Driver Element를 참조하십시오.
- 선택적 캐시 속성은 캐시 메커니즘을 제어합니다. 가능한 값은 기본,none,writethrough, writeback ,writeback,directsync (호스트 물리적 머신 페이지 캐시 쓰기 가 가능) 및 안전하지 않음(호스트 물리적 시스템 페이지 캐시가 무시됨) 및 안전하지 않은 (호스트 가상 머신 가상 머신의 모든 디스크 io, 동기화 요청을 캐시할 수 있음)입니다.
- 선택적 error_policy 속성은 하이퍼바이저가 디스크 읽기 또는 쓰기 오류에서 작동하는 방식을 제어합니다. 가능한 값은 중지,보고, ,ignore, enospace. error_policy 의 기본 설정은 보고서입니다. 읽기 오류에 대한 동작을 제어하는 선택적 rerror_policy 도 있습니다. rerror_policy 가 제공되지 않으면 error_policy 가 읽기 및 쓰기 오류 모두에 사용됩니다. rerror_policy 를 지정하면 읽기 오류가 있는지 error_policy 를 덮어씁니다. 또한 enospace 는 읽기 오류에 대한 유효한

*정책이 아니므로 **error_policy** 가 **enospace** 로 설정되어 있고 **rerror_policy** 가 제공되지 않으면 기본 설정인 **report** 가 사용되는 읽기 오류입니다.*

- *선택적 **io** 속성은 **I/O**에서 특정 정책을 제어합니다. **qemu** 게스트 가상 머신 가상 머신은 스레드 및 네이티브 를 지원합니다. 선택적 **ioeventfd** 특성을 사용하면 사용자가 디스크 장치에 대한 도메인 **I/O** 비동기 처리를 설정할 수 있습니다. 기본값은 하이퍼바이저의 재량에 따라 다릅니다. 허용되는 값은 설정 및 해제 입니다. 이를 활성화하면 별도의 스레드에서 **I/O**를 처리하는 동안 게스트 가상 머신 가상 머신을 실행할 수 있습니다. 일반적으로 게스트 가상 머신 가상 머신은 **I/O** 중에 시스템 **CPU** 사용률이 높아지는 데 도움이 됩니다. 반면 과부하된 호스트 물리적 시스템은 게스트 가상 머신 가상 시스템 **I/O** 대기 시간을 늘릴 수 있습니다. **IOS**를 조작할 필요가 있다는 것을 완전히 인증하지 않는 한, 기본 설정을 변경하지 않고 하이퍼바이저가 설정을 지시할 수 있도록 하는 것이 좋습니다.*

- *선택적 **event_idx** 속성은 장치 이벤트 처리의 일부 측면을 제어하며, 켜 거나 끌 수 있습니다. 이 경우 중단 횟수를 줄이고 게스트 가상 머신 가상 시스템에 대해 종료됩니다. 기본값은 하이퍼바이저에 따라 결정되며 기본 설정은 에 있습니다. 이 동작이 낙관적 인 상황이 있는 경우 이 속성은 기능을 강제로 끌 수 있는 방법을 제공합니다. **event_idx** 를 조작해야 한다는 것을 완전히 인증하지 않는 한 기본 설정을 변경하지 않고 하이퍼바이저가 설정을 지정할 수 있도록 하는 것이 좋습니다.*

- *선택 사항인 **copy_on_read** 속성은 읽기 백업 파일을 이미지 파일에 복사할지 여부를 제어합니다. 허용되는 값은 **<설정 또는 해제>** 일 수 있습니다. **copy-on-read** 는 동일한 백업 파일 섹터에 반복적으로 액세스하는 것을 방지하고 백업 파일이 느린 네트워크를 할 때 유용합니다. 기본적으로 **copy-on-read** 는 해제 되어 있습니다.*

20.16.1.7. 추가 장치 요소

다음 특성을 장치 요소 내에서 사용할 수 있습니다.

- ***<Boot -** 디스크를 부팅**>** 할 수 있도록 지정합니다.*

  추가 부팅 값

  - ***<순서> -** 부팅 시퀀스 중에 장치를 시도하는 순서를 결정합니다.*
  - ***<장치별>** 부팅 요소는 **BIOS** 부트 로더 섹션의 일반 부트 요소와 함께 사용할 수 없습니다.*

- ***<Encryption> -** 볼륨의 암호화 방법을 지정합니다. 자세한 내용은 스토리지 암호화 페이지를 참조하십시오.*

- *<readonly> - 게스트 가상 머신 가상 머신에서 장치를 수정할 수 없음을 나타냅니다. 이 설정은 attribute device='cdrom' 이 있는 디스크의 기본값입니다.*

- *공유 가능 은 장치가 도메인 간에 공유되어야 함을 나타냅니다(하이퍼바이저와 OS 지원의 경우). 공유할 수 있는 경우 해당 장치에 cache='no' 를 사용해야 합니다.*

- *<일시적>- 게스트 가상 머신 가상 머신이 종료되면 장치 콘텐츠에 대한 변경 사항을 자동으로 되돌리야 함을 나타냅니다. 일부 하이퍼바이저에서는 디스크를 임시로 표시 하면 도메인이 마이그레이션 또는 스냅샷에 참여하지 않습니다.*

- *<serial>- 게스트 가상 머신 가상 머신의 하드 드라이브의 일련 번호를 지정합니다. 예를 들어 <직렬>WD-WMAP9A966149</직렬> 입니다.*

- *<wwn> - 가상 하드 디스크 또는 CD-ROM 드라이브의 WWN(World Wide Name)을 지정합니다. 16개의 16진수로 구성되어야 합니다.*

- *<vendor> - 가상 하드 디스크 또는 CD-ROM 장치의 벤더를 지정합니다. 8 인쇄 가능한 문자보다 길면 안 됩니다.*

- *<product> - 가상 하드 디스크 또는 CD-ROM 장치의 제품을 지정합니다. 16 인쇄 가능 문자보다 길면 안 됩니다.*

- *<Host - 4 속성 지원: 각각 호스트> 이름,포트 번호,전송 유형 및 소켓 경로를 지정하는 viz, 이름 , 포트 , 전송 및 소켓을 지원합니다. 이 요소의 의미와 요소 수는 다음과 같이 protocol 속성에 따라 달라집니다.*

  추가 호스트 속성

  - *NBD - nbd -server를 실행하는 서버를 지정하고 하나의 호스트 물리적 컴퓨터에만 사용할 수 있습니다.*

  - *RBD - RBD 유형의 서버를 모니터링하고 하나 이상의 호스트 물리적 시스템에 사용할 수 있습니다.*

  -

Sheepdog - sheepdog 서버 중 하나를 지정합니다(기본값은 localhost:7000)이며 호스트 물리적 시스템 중 하나 또는 none을 사용할 수 있습니다.

- *Gluster - iLO 데몬을 실행하는 서버를 지정하고 하나의 호스트 물리적 시스템에 대해서만 사용할 수 있습니다. transport 속성에 유효한 값은 tcp,rdma 또는 unix 입니다. 아무것도 지정되지 않은 경우 tcp 를 가정합니다. transport가 unix 인 경우 socket 속성은 unix 소켓의 경로를 지정합니다.*

- *<address> - 컨트롤러의 지정된 슬롯에 디스크를 연결합니다. 실제 <컨트롤러> 장치는 종종 가 유추할 수 있지만 명시적으로 지정할 수도 있습니다. type 속성은 필수이며 일반적으로 pci 또는 드라이브 입니다. pci 컨트롤러의 경우 버스,슬롯 및 함수에 대한 추가 속성과 선택적 도메인 및 다중 기능이 있어야 합니다. Multifun ction 의 기본값은 off 입니다. 드라이브 컨트롤러의 경우 추가 특성 컨트롤러,버스,대상 및 유닛을 사용할 수 있으며 각각 기본 설정이 0 입니다.*

- *auth - 소스에 액세스하는 데 필요한 인증 자격 증명을 제공합니다. 인증 중에 사용할 사용자 이름을 식별하는 필수 특성 사용자 이름 및 필수 속성 유형의 하위 요소가 포함되도록 하는 필수 속성 사용자 이름 이 포함됩니다. 자세한 내용은 장치 요소(Deviceelement)에서 확인할 수 있습니다.*

- *지오메트리 - 지오메트리 설정을 재정의하는 기능을 제공합니다. 이는 S390 DASD-disks 또는 이전 DOS-disks에 주로 유용합니다.*

- *Cyls - 실린더 수를 지정합니다.*

- *heads - 헤드 수를 지정합니다.*

- *secs - 트랙당 섹터 수를 지정합니다.*

- *trans - BIOS-Translation-Modus를 지정하고 다음 값을 가질 수 있습니다:none,lba 또는 auto*

- *blockio - 아래 블록 장치 속성 중 하나로 블록 장치를 덮어쓸 수 있습니다.*

blockio 옵션

- *logical_block_size- 게스트 가상 머신 가상 머신 OS에 보고하고 디스크 I/O의 최소 단위를 설명합니다.*

- **physical_block_size** - 게스트 가상 머신 가상 머신 **OS**에 보고하고 디스크 데이터의 정렬과 관련이 있는 디스크의 하드웨어 섹터 크기를 설명합니다.

20.16.2. 파일 시스템

게스트 가상 머신 가상 머신에서 직접 액세스할 수 있는 호스트 물리적 시스템의 파일 시스템 디렉터리

그림 **20.24.** 장치 **-** 파일 시스템

```
...
 <devices>
  <filesystem type='template'>
    <source name='my-vm-template'/>
    <target dir='/'/>
  </filesystem>
  <filesystem type='mount' accessmode='passthrough'>
    <driver type='path' wrpolicy='immediate'/>
    <source dir='/export/to/guest'/>
    <target dir='/import/from/host'/>
    <readonly/>
  </filesystem>
  ...
 </devices>
 ...
```

filesystem 속성에는 다음과 같은 가능한 값이 있습니다.

- **type='mount' -** 게스트 가상 머신에 마운트할 호스트 물리적 머신 디렉터리를 지정합니다. 이 값은 지정되지 않은 경우 기본 유형입니다. 이 모드에는 속성 **type='path'** 또는 **type='handle'** 이 있는 선택적 하위 요소 드라이버 도 있습니다. 드라이버 블록에 호스트 물리적 시스템 페이지 캐시와의 상호 작용을 추가로 제어하는 선택적 속성 **wrpolicy** 가 있습니다. 속성을 생략하면 속성은 기본 설정으로 되돌아갑니다. 값을 직접 지정하면 게스트 가상 머신 파일 쓰기 작업 중에 연결된 모든 페이지에 대해 호스트 물리적 시스템 쓰기가 즉시 트리거된다는 것을 의미합니다.

- **type='template' - OpenVZ** 파일 시스템 템플릿을 지정하며 **OpenVZ** 드라이버에서만 사용됩니다.

- **type='file' -** 호스트 물리적 머신 파일이 이미지로 처리되어 게스트 가상 머신에 마운트되도록 지정합니다. 이 파일 시스템 형식은 자동 탐지되며 **LXC** 드라이버에서만 사용됩니다.

- ***type='block' -** 게스트 가상 머신에 마운트할 호스트 물리적 머신 블록 장치를 지정합니다. 파일 시스템 형식은 자동 탐지되며 **LXC** 드라이버에서만 사용됩니다.*

- ***type='ram' -** 호스트 물리적 머신 **OS**의 메모리를 사용하여 메모리 내 파일 시스템을 사용하도록 지정합니다. 소스 요소에는 메모리 사용량 제한을 **kibibytes**로 제공하고 **LXC** 드라이버에서만 사용하는 단일 특성 사용이 있습니다.*

- ***type='bind' -** 게스트 가상 머신 내부의 다른 디렉터리에 바인딩되는 디렉터리를 지정합니다. 이 요소는 **LXC** 드라이버에서만 사용됩니다.*

- ***AccessMode -** 소스 액세스에 대한 보안 모드를 지정합니다. 현재 이는 **QEMU/KVM** 드라이버의 **type='mount'**에서만 작동합니다. 가능한 값은 다음과 같습니다.*

  - ***Passthrough -** 게스트 가상 머신 내부에서 설정된 사용자의 권한 설정을 사용하여 소스에 액세스하도록 지정합니다. 이 모드는 지정되지 않은 경우 기본 액세스 모드입니다.*

  - ***mapped -** 하이퍼바이저의 권한 설정을 사용하여 소스에 액세스하도록 지정합니다.*

  - *스쿼시 **- 'passthrough'** 와 유사하게, 예외는 **chown** 과 같은 권한 있는 작업의 실패가 무시된다는 것입니다. 이로 인해 루트가 아닌 하이퍼바이저를 실행하는 사용자가 **passthrough** 유사 모드를 사용할 수 있습니다.*

- ***<Source> -** 게스트 가상 머신에서 액세스 중인 호스트 물리적 머신의 리소스를 지정합니다. **name** 속성은 **<type='template'>** 과 함께 사용해야 하며 **dir** 속성은 **<type='mount'>** 와 함께 사용해야 합니다. **usage** 속성은 **<type='ram'>** 과 함께 사용하여 메모리 제한을 **KB**로 설정합니다.*

- ***target -** 게스트 가상 시스템에서 소스 드라이버에 액세스할 수 있는 위치를 파악합니다. 대부분의 드라이버의 경우 이는 자동 마운트 지점이지만 **QEMU-KVM**의 경우 이는 마운트할 위치에 대한 힌트로 게스트 가상 시스템으로 내보낸 임의의 문자열 태그일 뿐입니다.*

- ***readonly -** 기본적으로 읽기 **-**쓰기 액세스 권한이 지정되므로 게스트 가상 시스템의 읽기 전용 마운트로 합성 파일을 내보낼 수 있습니다.*

- ***space_hard_limit -** 이 게스트 가상 머신의 파일 시스템에서 사용할 수 있는 최대 공간을 지정합니다.*

- ***space_soft_limit -*** *이 게스트 가상 시스템의 파일 시스템에서 사용할 수 있는 최대 공간을 지정합니다. 컨테이너는 유예 기간 동안 소프트 제한을 초과할 수 있습니다. 그 후 하드 제한이 적용됩니다.*

20.16.3. 장치 주소

*대부분의 장치에는 가상 버스에 배치된 장치가 게스트 가상 머신에 제공되는 위치를 설명하는 선택적 **<주소>** 하위 요소가 있습니다. 입력 시 주소(또는 주소 내의 선택적 속성)를 생략하면 **libvirt**에서 적절한 주소를 생성합니다. 그러나 레이아웃을 추가로 제어해야 하는 경우 명시적 주소가 필요합니다. **address** 요소를 포함한 장치 예제는 아래를 참조하십시오.*

*모든 주소에는 장치가 있는 버스를 설명하는 필수 속성 유형이 있습니다. 지정된 장치에 사용할 주소 중 일부는 장치 및 게스트 가상 머신의 아키텍처가 제한됩니다. 예를 들어, 디스크 장치는 **type='disk'** 를 사용하지만 콘솔 장치는 **32**비트 **AMD** 및 **Intel** 아키텍처 또는 **AMD64** 및 **Intel 64** 게스트 가상 머신에서 **type='pci'** 을 사용하거나 **PowerPC64 pseries** 게스트 가상 머신의 **type='spapr-vio'** 를 사용합니다. 각 주소 **<유형에>** 는 장치가 배치될 버스 위치를 제어하는 추가 선택적 속성이 있습니다. 추가 속성은 다음과 같습니다.*

- ***type='pci' - PCI** 주소에는 다음과 같은 추가 속성이 있습니다.*
  - *도메인 (현재 **qemu**에서 사용되지 않는 **2**바이트 **16x** 정수)*
  - *버스 (**0xff**와 **0xff** 사이의 **16**진수 값)*
  - *슬롯 (**0x0**과 **0x1f** 사이의 **16**진수 값, 포함)*
  - *기능 (**0**)에서 **7** 사이의 값 (포함)*
  - *또한 사용할 수 있는 다중**function** 속성으로 **PCI** 제어 레지스터의 특정 슬롯/기능에 대한 다기능 비트 켜기를 제어합니다. 이 **multifunction** 속성은 기본적으로 **'off'** 로 설정되지만 함수 **0**에 대해 여러 함수가 사용되는 슬롯의 경우 **'on'** 로 설정되어야 합니다.*
- ***type='drive -** 드라이브 주소에는 다음과 같은 추가 속성이 있습니다.*

- *컨트롤러- (두 자리 컨트롤러 번호)*
- *버스 - (두 자리 버스 번호)*
- *대상 - (두 자리 버스 번호)*
- *단원 - (버스에 있는 2자리 단위 번호)*

- ***type='virtio-serial' -** 각 **virtio-serial** 주소에는 다음과 같은 추가 속성이 있습니다.*
  - *컨트롤러 - (두 자리 컨트롤러 번호)*
  - *버스 - (두 자리 버스 번호)*
  - *슬롯 - (버스 내의 두 자리 슬롯)*
- ***type='ccid' -** 스마트 카드에 사용되는 **CCID** 주소에는 다음과 같은 추가 속성이 있습니다.*
  - *버스 - (두 자리 버스 번호)*
  - *슬롯 속성 - (버스 내의 두 자리 슬롯)*
- ***type='usb' - USB** 주소에는 다음과 같은 추가 속성이 있습니다.*
  - ***bus - (0xfff** 사이의 **16**진수 값, 포함)*
  - *포트 - (**1.2** 또는 **2.1.3.1**과 같이 최대 **4**개의 **8**진수로 구분된 표기법)*
-

type='spapr-vio - On PowerPC pseries 게스트 가상 머신, 장치는 **SPAPR-VIO** 버스에 할당할 수 있습니다. 플랫 **64**비트 주소 공간이 있습니다. 규칙에 따라 장치는 **0**이 아닌 **0x1000**의 숫자로 일반적으로 할당되지만 다른 주소는 **libvirt**에서 유효하며 허용됩니다. 추가 특성: **reg**(시작 레지스터의 **16**진수 값)를 이 속성에 할당할 수 있습니다.

20.16.4. 컨트롤러

게스트 가상 머신 아키텍처에 따라 여러 가상 장치를 단일 버스에 할당할 수 있습니다. 정상적인 상황에서 **libvirt** 는 버스에 사용할 컨트롤러를 자동으로 유추할 수 있습니다. 그러나 게스트 가상 머신 **XML**에서 명시적 **<컨트롤러>** 요소를 제공해야 할 수도 있습니다.

그림 **20.25.** 컨트롤러 요소

```
...
<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>
...
```

각 컨트롤러에는 필수 속성 유형 이 있습니다. **"ide", "fdc", "scsi", "sata", "usb", "ccid"** 또는 **"virtio-serial"**, 및 **"virtio-serial"** , 버스 컨트롤러가 발생하는 순서를 설명하는 필수 특성 색인 **(address** 요소의 컨트롤러 속성 사용)이어야 합니다. **"virtio-serial"** 컨트롤러에는 컨트롤러를 통해 연결할 수 있는 장치 수를 제어하는 두 가지 추가 선택적 속성인 **ports** 및 **vectors** 가 있습니다.

<컨트롤러 type='scsi'> 에는 선택적 속성 모델이 있습니다. **"auto", "buslogic", "ibmvscsi", "lsilogic"** , **"lsilogic", "lsias1068 ", "virtio-scsi** 또는 **"vmpvscsi". virtio-scsi** 컨트롤러와 드라이버는 **KVM** 및 **Windows** 게스트 가상 머신 모두에서 작동합니다. 또한 **<컨트롤러 type='scsi'>** 에는 지정된 대기열 수에 대해 다중 큐 지원을 지원하는 **num_queues** 속성이 있습니다.

"usb" 컨트롤러에는 선택적 속성 모델이 있습니다. **"piix3-uhci", "piix3-uhci", "ehci", "ehci"** , **"ich9-ehci1", "ich9-u hci1"** , **"ich9-uhci1", "ich9-uhci2", , "ich9-uhci2 ", "ich9-uhci3 ", "vt82c686b-uhci", " pci-ohci"** 또는 **"nec-xhci". 또한 게스트 가상 머신에 대해 USB** 버스를 명시적으로 비활성화해야 하는 경우 **model='none'** 을 사용할 수 있습니다. **PowerPC64 "spapr-vio"** 주소에는 연결된 컨트롤러가 없습니다.

PCI 또는 **USB** 버스의 장치 자체인 컨트롤러의 경우 선택적 하위 요소 주소는 위에 제공된 의미와 함께 컨트롤러의 정확한 관계를 마스터 버스와 지정할 수 있습니다.

USB 도우미 컨트롤러는 해당 마스터 컨트롤러와의 정확한 관계를 지정하기 위한 선택적 하위 요소 마스터가 있습니다. 파트너 컨트롤러는 마스터와 동일한 버스에 있으므로 companion index 값이 동일해야 합니다.

그림 20.26. 장치 - 컨트롤러 - 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>
...
```

20.16.5. 장치 리스

잠금 관리자를 사용하는 경우 게스트 가상 머신에 대한 장치 리스를 기록하는 옵션이 있습니다. 잠금 관리자는 리스를 가져올 수 없는 한 게스트 가상 머신이 시작되지 않도록 합니다. 기존 관리 도구를 사용하여 구성하면 도메인 xml의 다음 섹션이 적용됩니다.

그림 20.27. 장치 - 장치 리스

```
...
<devices>
  ...
  <lease>
    <lockspace>somearea</lockspace>
    <key>somekey</key>
    <target path='/some/lease/path' offset='1024'/>
  </lease>
  ...
</devices>
...
```

lease 섹션에는 다음 인수가 있을 수 있습니다.

- *잠금 공간 - 키가 보관된 잠금 공간을 식별하는 임의의 문자열입니다. 잠금 관리자는 포맷 또는 잠금 공간 이름의 길이에 추가 제한을 적용할 수 있습니다.*

- **key -** 임대를 고유하게 식별하는 임의의 문자열입니다. 잠금 관리자는 키 형식 또는 길이에 추가 제한을 적용할 수 있습니다.

- **target -** 잠금 공간과 연결된 파일의 정규화된 경로입니다. 오프셋은 리스가 파일 내에 저장된 위치를 지정합니다. 잠금 관리자가 오프셋을 필요로 하지 않으면 이 값을 **0** 으로 설정합니다.

20.16.6. 호스트 물리적 시스템 장치 할당

이 섹션에서는 호스트 물리적 시스템 장치 할당에 대한 정보를 제공합니다.

20.16.6.1. USB / PCI 장치

hostdev 요소를 사용하여 호스트 물리적 시스템의 **USB** 및 **PCI** 장치를 게스트 가상 시스템으로 전달할 수 있습니다. 관리 툴을 사용하여 도메인 **xml** 파일의 다음 섹션이 구성됩니다.

그림 **20.28.** 장치 **-** 호스트 물리적 시스템 장치 할당

```
...
<devices>
  <hostdev mode='subsystem' type='usb'>
    <source startupPolicy='optional'>
      <vendor id='0x1234'/>
      <product id='0xbeef'/>
    </source>
    <boot order='2'/>
  </hostdev>
</devices>
...
```

또는 다음을 수행할 수도 있습니다.

그림 20.29. 장치 - 호스트 물리적 시스템 장치 할당 대안

```
...
<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>
...
```

도메인 XML의 이 섹션의 구성 요소는 다음과 같습니다.

표 20.13. 호스트 물리적 시스템 장치 할당 요소

매개변수	설명
hostdev	호스트 물리적 시스템 장치를 설명하는 기본 컨테이너입니다. USB 장치 패스스루 **모드** 의 경우 항상 **하위** 시스템이며 **유형은** USB 장치의 경우 **usb** 이고 PCI 장치의 경우 **pci** 입니다. PCI **장치에** 대해 **관리되는** 경우 게스트 가상 머신에 전달되기 전에 호스트 물리적 시스템에서 분리되고 게스트 가상 머신이 종료된 후 호스트 물리적 시스템에 다시 연결됩니다. **managed** 가 생략되거나 PCI 장치용 **no** 인 경우 사용자는 게스트 가상 머신을 시작하기 전에 **virNodeDeviceDettach** 인수(또는 **virsh nodedev-dettach**)를 사용하고, 게스트 가상 머신을 핫 플러그하거나 중지한 후 **virNodeDeviceReAttach** (또는 **virsh nodedev-reattach**)를 사용해야 합니다.
소스	호스트 물리적 시스템에서 볼 수 있는 장치를 설명합니다. USB 장치는 **vendor** 및 **product** 요소를 사용하여 공급 업체 / 제품 ID를 사용하거나 address 요소를 사용하여 호스트 물리적 시스템의 장치 **주소로** 처리할 수 있습니다. 반면 PCI 장치는 주소에 의해서만 설명될 수 있습니다. USB 장치의 소스 요소에는 지정된 호스트 물리적 시스템 USB 장치가 없는 경우 수행할 작업을 위한 규칙을 정의하는 데 사용할 수 있는 **startupPolicy** 속성이 포함될 수 있습니다. 속성에는 다음 값을 사용할 수 있습니다. • **mandatory** - 어떤 이유로 누락된 경우(기본값) 실패 • **requisite** - 부팅 시 누락된 경우 실패합니다. migrate/restore/revert에 없는 경우 삭제 • **선택 사항** - 시작 시도에 누락된 경우 삭제

매개변수	설명
공급 업체, 제품	이러한 요소에는 각각 USB 공급업체 및 제품 ID를 지정하는 **id** 속성이 있습니다. ID는 10진수, 16진수(0x로 시작) 또는 8진수(0)로 시작할 수 있습니다.
boot	장치를 부팅할 수 있도록 지정합니다. 특성의 순서에 따라 부팅 순서 중 장치가 시도되는 순서가 결정됩니다. 장치별 부팅 요소는 BIOS 부트 로더 섹션의 일반 부트 요소와 함께 사용할 수 없습니다.
Rom	PCI 장치의 Rom이 게스트 가상 머신에 표시되는 방식을 변경하는 데 사용됩니다. 선택 사항인 **bar** 속성을 **on** 또는 **off** 로 설정할 수 있으며, 장치의om이 게스트 가상 머신의 메모리 맵에 표시되는지 여부를 결정할 수 있습니다. (PCI 문서의 경우, **rombar** 설정은 Rom에 대한 기본 주소 등록의 존재를 제어합니다.) rom bar를 지정하지 않으면 기본 설정이 사용됩니다. 선택적 **파일** 속성은 게스트 가상 머신이 장치의 ROM BIOS로 표시되도록 바이너리 파일을 가리키는 데 사용됩니다. 이는 예를 들어 sr-iov 가능 이더넷 장치의 가상 기능에 PXE 부팅 롬을 제공하는 데 유용할 수 있습니다(VF에 대한 부팅 Rom이 없는 경우).
주소	또한 호스트 물리적 시스템의 USB 버스 및 장치 번호를 지정하는 버스 및 장치 속성이 있습니다. **이러한 속성의 값은 10진수, 16진수(0x로 시작) 또는 8진수(0으로 시작) 형식으로 지정할 수 있습니다. PCI 장치의 경우 요소는 lspci 또는 virsh nodedev-list를 사용하여 찾을 수 있는 것처럼 장치를 지정할 수 있는 3개의 속성을 전달합니다.**

20.16.6.2. 블록 / 문자 장치

*호스트 물리적 머신의 블록 / 문자 장치는 도메인 **xml hostdev** 요소를 수정하는 관리 툴을 사용하여 게스트 가상 머신에 전달할 수 있습니다. 이는 컨테이너 기반 가상화에서만 가능합니다.*

*그림 **20.30.** 장치 **-** 호스트 물리적 시스템 장치 할당 블록 문자 장치*

```
...
<hostdev mode='capabilities' type='storage'>
  <source>
    <block>/dev/sdf1</block>
  </source>
</hostdev>
...
```

대체 접근 방식은 다음과 같습니다.

그림 20.31. 장치 - 호스트 물리적 시스템 장치 할당 블록 문자 장치 대안 1

```
...
<hostdev mode='capabilities' type='misc'>
  <source>
    <char>/dev/input/event3</char>
  </source>
</hostdev>
...
```

또 다른 다른 접근 방식은 다음과 같습니다.

그림 20.32. 장치 - 호스트 물리적 시스템 장치 할당 블록 문자 장치 대안 2

```
...
<hostdev mode='capabilities' type='net'>
  <source>
    <interface>eth0</interface>
  </source>
</hostdev>
...
```

도메인 XML의 이 섹션의 구성 요소는 다음과 같습니다.

표 20.14. 블록 / 문자 장치 요소

매개변수	설명
hostdev	호스트 물리적 시스템 장치를 설명하는 기본 컨테이너입니다. 블록/문자 장치 통과 **모드** 의 경우 항상 **기능** 이며, 문자 장치의 경우 블록 장치 및 **char** 에 대한 **유형** 이 차단됩니다.
소스	이는 호스트 물리적 시스템에서 표시된 대로 장치를 설명합니다. 블록 장치의 경우 호스트 물리적 머신 OS의 블록 장치 경로는 중첩 **블록** 요소에 제공되며 문자 장치의 경우 **char** 요소가 사용됩니다.

20.16.7. 리디렉션된 장치

문자 장치를 통한 USB 장치 리디렉션은 도메인 xml의 다음 섹션을 수정하는 관리 도구로 구성하여 지원됩니다.

그림 20.33. 장치 - 리디렉션된 장치

```
...
<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>
...
```

도메인 XML의 이 섹션의 구성 요소는 다음과 같습니다.

표 20.15. 리디렉션된 장치 요소

매개변수	설명
redirdev	리디렉션된 장치를 설명하는 기본 컨테이너입니다. **버스는** USB 장치 **용** USB가 되어야 합니다. 터널의 호스트 물리적 시스템 측면을 설명하기 위해 지원되는 직렬 장치 유형 중 하나와 일치하는 추가 특성 유형이 필요합니다. **type='tcp'** 또는 **type='spicevmc'** (SPICE 그래픽 장치의 usbredir 채널을 사용하는)가 일반적입니다. redirdev 요소에는 특정 컨트롤러에 장치를 연결할 수 있는 선택적 하위 요소가 있습니다. **소스(source)와 같은 추가 서브-요소는 특정 유형에 따라 요구될 수 있지만,target 서브-요소는 필요하지 않다(상자 장치의 소비자는 게스트 가상 머신에 표시되는 장치가 아닌 하이퍼바이저 자체임).**
boot	장치를 부팅할 수 있도록 지정합니다. order 속성은 부팅 시퀀스 중 장치를 시도하는 순서를 결정합니다. 장치별 부팅 요소는 BIOS 부트 로더 섹션의 일반 부트 요소와 함께 사용할 수 없습니다.
redirfilter	이는 리디렉션에서 특정 장치를 필터링하기 위해 필터 규칙을 생성하는 데 사용됩니다. 하위 요소 **usbdev** 를 사용하여 각 필터 규칙을 정의합니다. **class** 속성은 USB 클래스 코드입니다.

20.16.8. 스마트 카드 장치

*스마트 카드 요소를 통해 가상 스마트 카드 장치를 게스트 가상 머신에 제공할 수 있습니다. 호스트 머신의 **USB** 스마트 카드 판독기 장치는 호스트와 게스트 둘 다에서 사용할 수 없으므로 간단한 장치 패스스루가 있는 게스트에서 사용할 수 없으며 게스트에서 호스트 컴퓨터를 잠글 수 있습니다. 따라서 일부 하이퍼바이저는 게스트 가상 머신에 스마트 카드 인터페이스를 제공할 수 있는 특수 가상 장치를 제공하며, 호스트 시스템에서 자격 증명을 얻는 방법 또는 타사 스마트 카드 공급자가 생성한 채널을 설명할 수 있는 여러 가지 모드를 제공합니다. 문자 장치를 통해 **USB** 장치 리디렉션에 대한 매개 변수를 설정하려면 도메인 **XML**의 다음 섹션을 편집합니다.*

*그림 **20.34.** 장치 **-** 스마트 카드 장치*

```
...
<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>
...
```

*스마트 카드 요소에는 필수 속성 모드가 있습니다. 다음 모드가 지원됩니다. 각 모드에서 게스트 가상 머신은 **USB** 버스에서 장치를 확인하여 물리적 **USB CCID(Chip/Smart Card Interface Device)** 카드처럼 작동합니다.*

모드 속성은 다음과 같습니다.

*표 **20.16.** 스마트 카드 모드 요소*

매개변수	설명
mode='host'	이 모드에서 하이퍼바이저는 게스트 가상 머신에서 NSS를 통해 호스트 물리적 시스템의 스마트 카드로 모든 직접 액세스 요청을 릴레이합니다. 다른 속성이나 하위 요소가 필요하지 않습니다. 선택적 **주소** 하위 요소 사용에 대한 아래를 참조하십시오.

매개변수	설명
mode='host-certificates'	이 모드를 사용하면 호스트 물리적 시스템에 스마트 카드를 연결할 필요 없이 호스트 물리적 시스템의 데이터베이스에 있는 세 개의 NSS 인증서 이름을 제공할 수 있습니다. 이러한 인증서는 **certutil -d /etc/pki/nssdb -x -t CT,CT -S -s CN=cert1 -n cert1** 명령을 사용하여 생성할 수 있으며 결과 세 개의 인증서 하위 요소 각각에 대한 콘텐츠로 결과 세 개의 **인증서** 이름을 제공해야 합니다. 추가 하위 요소**데이터베이스** 는 대체 디렉터리에 대한 절대 경로를 지정할 수 있습니다(인증서를 생성할 때 **certutil** 명령의 **-d** 옵션 일치) 지정하지 않으면 기본값은 **/etc/pki/nssdb** 입니다.
mode='passthrough'	이 모드를 사용하면 하이퍼바이저가 호스트 물리적 시스템과 직접 통신하는 대신 보조 문자 장치를 통해 모든 요청을 타사 공급자에게 터널링할 수 있습니다(따라서 스마트 카드 또는 세 개의 인증서 파일을 사용할 수 있음). 이 모드에서는 지원되는 직렬 장치 **유형** 중 하나와 함께, 터널의 호스트 물리적 시스템 측면을 설명하기 위해 추가 특성 유형이 필요합니다. **type='tcp'** 또는 **type='spicevmc'** (SPICE 그래픽 장치의 스마트 카드 채널을 사용하는)가 일반적입니다. 소스(**source**) 와 같은 서브-요소는 특정 유형에 따라 요구될 수 있지만, **target** 서브-요소는 필요하지 않다(자본 장치의 소비자는 게스트 가상 머신에서 볼 수 있는 장치가 아닌 하이퍼바이저 자체임).

*각 모드에서는 스마트 카드와 **ccid** 버스 컨트롤러(20.16.3절. “장치 주소”참조) 간의 상관 관계를 미세 조정하는 선택적 하위 요소 주소를 지원합니다.*

20.16.9. 네트워크 인터페이스

*네트워크 인터페이스 장치는 도메인 **XML**의 다음 부분을 구성하는 관리 도구를 사용하여 수정됩니다.*

그림 20.35. 장치 - 네트워크 인터페이스

```
...
<devices>
  <interface type='bridge'>
    <source bridge='xenbr0'/>
    <mac address='00:16:3e:5d:c7:9e'/>
    <script path='vif-bridge'/>
    <boot order='1'/>
    <rom bar='off'/>
  </interface>
</devices>
...
```

게스트 가상 머신에 표시되는 네트워크 인터페이스를 지정할 수 있는 몇 가지 가능성이 있습니다. 아래의 각 하위 섹션에서는 일반적인 설정 옵션에 대한 자세한 정보를 제공합니다. 또한 각 <인터페이스> 요소에는 인터페이스 type='pci' (20.16.3절. "장치 주소"참조)을 사용하여 인터페이스를 특정 pci 슬롯에 연결할 수 있는 선택적 <주소> 하위 요소가 있습니다.

20.16.9.1. 가상 네트워크

동적 / 무선 네트워킹 구성을 사용하는 호스트 물리적 머신의 일반적인 게스트 가상 머신 연결 (또는 호스트 물리적 머신 하드웨어 세부 정보가 <네트워크> 정의에서 별도로 설명되는 다중 호스트 물리적 머신 환경)에 대한 일반적인 게스트 가상 머신 연결에 권장되는 구성입니다. 또한 이름이 지정된 네트워크 정의에서 세부 정보를 설명하는 연결을 제공합니다. 가상 네트워크의 전달 모드 구성에 따라 네트워크는 완전히 격리되고(<전방> 요소가 제공되지 않음), NAT가 명시적 네트워크 장치 또는 기본 경로(forward mode='nat'), NAT 없이 라우팅된(forward mode='route'/), 또는 호스트 물리적 시스템의 네트워크 인터페이스(macvtap) 또는 브리지 (pass bridge) 중 하나에 직접 연결할 수 있습니다.

브리지의 전달 모드인 private, vepa 및 passthrough가 있는 네트워크의 경우 호스트 물리적 시스템에 필요한 DNS 및 DHCP 서비스가 이미 libvirt 범위 외부에 설정된 것으로 가정합니다. 분리된 네트워크 nat 및 라우팅된 네트워크의 경우 libvirt에서 DHCP 및 DNS가 가상 네트워크에 제공되며 IP 범위는 virsh net-dumpxml [networkname] 을 사용하여 가상 네트워크 구성을 검사하여 확인할 수 있습니다. NAT를 기본 경로로 설정하며 IP 범위 192.168.122.0/255.255.0이 있는 박스에서 'default' 설정이라는 하나의 가상 네트워크가 있습니다. 각 게스트 가상 머신에는 vnetN이라는 이름으로 생성된 관련 tun 장치가 있으며 이는 <대상> 요소(20.16.9.11절. "대상 요소 덮어쓰기"참조)로 덮어쓸 수도 있습니다.

인터페이스의 소스가 네트워크인 경우 네트워크 이름과 함께 portgroup을 지정할 수 있습니다. 한 네트워크에는 여러 포트 그룹이 정의될 수 있으며 각 portgroup에는 네트워크 연결 클래스마다 약간 다른 구성 정보가 포함되어 있을 수 있습니다. 또한 <직접> 네트워크 연결(아래 설명)과 유사하게, 유형 네트워크 의 연결은 <가상 포트> 요소를 지정하고, 구성 데이터를 vepa(802.1Qbg) 또는 802.1Qbh 호환 스위치 또는 Open vSwitch 가상 스위치로 전달할 수 있습니다.

실제 유형의 스위치는 호스트 물리적 시스템의 <네트워크> 구성에 따라 다를 수 있으므로, virtualport type 속성을 생략하고, 여러 가상 포트 유형(및 특정 속성 종료)에서 속성을 지정하는 것이 허용됩니다. 도메인 시작 시, 완전한 <가상 포트> 요소는 네트워크에 참조되는 포트그룹과 인터페이스에 정의된 유형 및

속성을 병합하여 구성됩니다. 새로 추가된 가상 포트는 둘 다의 조합입니다. 더 낮은 **virtualport**의 속성은 상위 가상 포트에 정의된 속성을 변경할 수 없습니다. 인터페이스는 우선 순위가 가장 높으며 **portgroup** 이 가장 낮은 우선 순위입니다.

예를 들어 **802.1Qbh** 스위치와 **Open vSwitch** 스위치를 모두 사용하여 제대로 작동하는 네트워크를 만들려면 유형을 지정하지 않아도 **profileid** 와 **interfaceid** 를 모두 제공해야 합니다. **managerid,typeid** 또는 **profileid** 와 같은 가상 포트에서 채울 다른 속성은 선택 사항입니다.

특정 유형의 스위치에만 연결하도록 게스트 가상 머신을 제한하려면 **virtualport** 유형을 지정할 수 있으며 지정된 포트 유형의 스위치만 연결됩니다. 추가 매개 변수를 지정하여 스위치 연결을 추가로 제한할 수도 있습니다. 결과적으로 포트가 지정되고 호스트 물리적 시스템의 네트워크에 다른 유형의 **virtualport** 가 있는 경우 인터페이스 연결이 실패합니다. 가상 네트워크 매개 변수는 도메인 **XML**의 다음 부분을 수정하는 관리 도구를 사용하여 정의됩니다.

그림 **20.36.** 장치 **-** 네트워크 인터페이스**-**가상 네트워크

```
...
<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>
...
```

20.16.9.2. 브릿지 LAN

정적 유선 네트워킹 구성이 있는 호스트 물리적 시스템에서 일반 게스트 가상 머신 연결에 대해 권장되는 구성 설정입니다.

Bridge와 **LAN**은 게스트 가상 시스템의 브릿지를 **LAN**에 직접 제공합니다. 호스트 물리적 머신에는 호스트 물리적 **NIC**가 하나 이상 있는 브릿지 장치가 있다고 가정합니다. 게스트 가상 머신에는 **<vnetN>** 이라는 이름으로 생성된 관련 **tun** 장치가 있으며, **<대상>** 요소(**20.16.9.11**절. “대상 요소 덮어쓰기”참조) 로 덮어쓸 수도 있습니다. **<tun>** 장치는 브리지에 종속됩니다. **IP** 범위/네트워크 구성은 **LAN**에서 사용되는 모든 항목입니다. 그러면 물리적 시스템과 마찬가지로 들어오는 게스트 가상 머신에서 들어오고 나가는 네트워크 액세스가 가능합니다.

Linux 시스템에서 브리지 장치는 일반적으로 표준 Linux 호스트 물리적 시스템 브리지입니다. Open vSwitch를 지원하는 호스트 물리적 시스템에서 virtualport type='openvswitch'/ 를 인터페이스 정의에 추가하여 open vSwitch 브리지 장치에 연결할 수도 있습니다. Open vSwitch 유형 virtualport는 Open vSwitch에 대한 이 특정 인터페이스를 고유하게 식별하는 데 사용되는 표준 uuid인 interfaceid 와 인터페이스 <포트-> 프로필으로 Open vSwitch로 전송된 선택적 profileid 를 사용자에게 생성합니다. 브리지를 LAN 설정으로 설정하려면 도메인 XML의 다음 부분을 구성하는 관리 도구를 사용합니다.

그림 20.37. 장치 - 네트워크 인터페이스 - LAN에 대한 브리지

```
...
<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>
```

20.16.9.3. 포트 마스커레이딩 범위 설정

포트 마스커레이딩 범위를 설정하려면 다음과 같이 포트를 설정할 수 있습니다.

그림 20.38. 포트 마스커레이딩 범위

```
<forward mode='nat'>
   <address start='192.0.2.1' end='192.0.2.10'/>
</forward>  ...
```

이러한 값은 다음과 같이 iptables 명령을 사용하여 설정해야 합니다. 18.3절. “네트워크 주소 변환 모드”

20.16.9.4. 사용자 공간 SLIRP 스택

사용자 공간 SLIRP 스택 매개 변수를 설정하면 NAT가 있는 가상 LAN이 외부 세계에 제공됩니다. 가상 네트워크에는 DHCP 및 DNS 서비스가 있으며 게스트 가상 시스템에 10.0.2.15부터 시작하는 IP 주소

*를 제공합니다. 기본 라우터는 **10.0.2.2**이며 **DNS** 서버는 **10.0.2.3**입니다. 이 네트워킹은 게스트 가상 머신이 나가는 액세스 권한이 필요한 권한이 없는 사용자에게 유일한 옵션입니다.*

*사용자 공간 **SLIP** 스택 매개변수는 도메인 **XML**의 다음 부분에 정의되어 있습니다.*

*그림 **20.39.** 장치 **-** 네트워크 인터페이스**-** 사용자 공간 **SLIRP** 스택*

```
...
<devices>
  <interface type='user'/>
  ...
  <interface type='user'>
    <mac address="00:11:22:33:44:55"/>
  </interface>
</devices>
...
```

20.16.9.5. 일반 이더넷 연결

*관리자가 게스트 가상 시스템의 네트워크를 **LAN**에 연결하는 임의의 스크립트를 실행할 수 있는 수단을 제공합니다. 게스트 가상 머신에는 **vnetN** 이라는 이름으로 생성되는 **tun** 장치가 있으며 이는 **target** 요소로 재정의할 수도 있습니다. **tun** 장치를 만든 후에는 필요한 호스트 물리적 시스템 네트워크 통합이 필요한 모든 작업을 수행할 것으로 예상되는 쉘 스크립트를 실행합니다. 기본적으로 이 스크립트는 **/etc/qemu-ifup** 이라고 하지만 덮어쓸 수 있습니다(**20.16.9.11**절. “대상 요소 덮어쓰기”참조).*

*일반 이더넷 연결 매개변수는 도메인 **XML**의 다음 부분에 정의되어 있습니다.*

*그림 **20.40.** 장치 **-** 네트워크 인터페이스**-** 일반 이더넷 연결*

```
...
<devices>
  <interface type='ethernet'/>
  ...
  <interface type='ethernet'>
    <target dev='vnet7'/>
    <script path='/etc/qemu-ifup-mynet'/>
  </interface>
</devices>
...
```

20.16.9.6. 물리적 인터페이스에 직접 연결

***<**인터페이스 **type='direct'>** 를 사용하면 가상 시스템의 **NIC**를 호스트의 지정된 물리적 인터페이스에 연결합니다.*

*이 설정을 사용하려면 **Linux macvtap** 드라이버를 사용할 수 있어야 합니다. 다음 모드 중 하나를 **macvtap** 장치의 운영 모드로 선택할 수 있습니다: **vepa ('Virtual Ethernet Port Aggregator'), 기본 모드, 브리지 또는 개인 모드입니다.***

*물리적 인터페이스에 직접 연결을 설정하려면 도메인 **XML**에서 다음 매개 변수를 사용합니다.*

*그림 **20.41.** 장치 - 네트워크 인터페이스 - 물리적 인터페이스에 직접 연결*

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='vepa'/>
  </interface>
</devices>
...
```

*개별 모드에서는 표 **20.17.** "물리적 인터페이스 요소에 직접 연결" 과 같이 패킷 전달이 작동합니다.*

*표 **20.17.** 물리적 인터페이스 요소에 직접 연결*

element	설명
vepa	모든 게스트 가상 머신의 패킷은 외부 브리지로 전송됩니다. 패킷이 시작된 위치와 동일한 호스트 물리적 시스템의 게스트 가상 머신인 패킷은 VEPA 가능 브릿지(일반적으로 VEPA가 사용할 수 없음)에 의해 호스트 물리적 시스템으로 다시 전송됩니다.
bridge	대상이 시작된 위치와 동일한 호스트 물리적 시스템에 있는 패킷은 대상 macvtap 장치로 직접 전달됩니다. 오리진 장치와 대상 장치 모두 직접 전달을 위해 브리지 모드에 있어야 합니다. 둘 중 하나가 **vepa** 모드에 있는 경우 VEPA 가능 브릿지가 필요합니다.
비공개	모든 패킷은 외부 브리지로 전송되며 외부 라우터 또는 게이트웨이를 통해 전송되고 해당 장치가 호스트 물리적 시스템으로 다시 전송하는 경우에만 동일한 호스트 물리적 시스템의 대상 VM으로 전달됩니다. 소스 또는 대상 장치가 개인 모드인 경우 이 절차를 따릅니다.
passthrough	이 기능은 마이그레이션 기능을 손실하지 않고 SRIOV 가능 NIC의 가상 기능을 게스트 가상 머신에 직접 연결합니다. 모든 패킷은 구성된 네트워크 장치의 VF/IF로 전송됩니다. 장치의 추가 사전 요구 사항 또는 제한 사항이 적용될 수 있습니다. 예를 들어 커널 2.6.38 이상이 필요합니다.

직접 연결된 가상 시스템의 네트워크 액세스는 호스트 물리적 시스템의 물리적 인터페이스가 연결된 하드웨어 스위치로 관리할 수 있습니다.

*스위치가 **IEEE 802.1Qbg** 표준을 준수하는 경우 인터페이스에 다음과 같이 추가 매개 변수가 있을 수 있습니다. **virtualport** 요소의 매개변수는 **IEEE 802.1Qbg** 표준에 자세히 설명되어 있습니다. 값은 네트워크에 따라 다르며 네트워크 관리자가 제공해야 합니다. **802.1Qbg** 측면에서 **VSI**(가상 스테이션 인터페이스)는 가상 머신의 가상 인터페이스를 나타냅니다.*

***IEEE 802.1Qbg**에는 **VLAN ID**에 대해 **0**이 아닌 값이 필요합니다.*

*조작할 수 있는 추가 요소는 표 **20.18.** *"물리적 인터페이스 추가 요소에 직접 연결"* 에 설명되어 있습니다.*

*표 **20.18.** 물리적 인터페이스 추가 요소에 직접 연결*

element	설명
managerid	VSI Manager ID는 VSI 유형 및 인스턴스 정의가 포함된 데이터베이스를 식별합니다. 이는 정수 값이며 값 **0** 이 예약됩니다.
typeid	VSI 유형 ID는 네트워크 액세스를 나타내는 VSI 유형을 식별합니다. VSI 유형은 일반적으로 네트워크 관리자가 관리합니다. 정수 값입니다.
typeidversion	VSI 유형 버전에서는 여러 버전의 VSI 유형을 사용할 수 있습니다. 정수 값입니다.
instanceid	VSI 인스턴스 ID 식별자는 VSI 인스턴스(가상 시스템의 가상 인터페이스)가 생성될 때 생성됩니다. 이는 전역적으로 고유한 식별자입니다.
profileid	프로필 ID에는 이 인터페이스에 적용할 포트 프로필의 이름이 포함되어 있습니다. 이 이름은 port 프로필 데이터베이스에서 포트 프로필에서 네트워크 매개 변수로 확인하며 해당 네트워크 매개 변수가 이 인터페이스에 적용됩니다.

*도메인 **XML**의 추가 매개 변수는 다음과 같습니다.*

그림 20.42. 장치 - 네트워크 인터페이스 - 물리적 인터페이스에 직접 연결 추가 매개변수

```
...
<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>
...
```

*스위치가 **IEEE 802.1Qbh** 표준을 준수하는 경우 인터페이스에 다음과 같이 추가 매개 변수가 있을 수 있습니다. 값은 네트워크에 따라 다르며 네트워크 관리자가 제공해야 합니다.*

*도메인 **XML**의 추가 매개 변수는 다음과 같습니다.*

그림 20.43. 장치 - 네트워크 인터페이스 - 물리적 인터페이스에 직접 연결 더 추가 매개변수

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private'/>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance'/>
    </virtualport>
  </interface>
</devices>
...
```

***profileid** 특성에는 이 인터페이스에 적용할 포트 프로필의 이름이 포함되어 있습니다. 이 이름은 **port** 프로필 데이터베이스에서 포트 프로필에서 네트워크 매개 변수로 확인하며 해당 네트워크 매개 변수가 이 인터페이스에 적용됩니다.*

20.16.9.7. PCI 패스스루

PCI** 네트워크 장치(소스 요소에 의해 지정됨)는 먼저 장치의 **MAC** 주소를 구성된 값으로 설정하고 선택적으로 지정된 가상 포트 요소를 사용하여 장치를 **802.1Qbhable** 스위치와 연결하는 후 일반 장치 통과를 사용하는 게스트 가상 머신에 직접 할당됩니다(**type='direct'** 네트워크 장치에 대해 위에 지정된 가상 포트의 예제 참조). 표준 단일 포트 **PCI** 이더넷 카드 설계의 제한으로 인해 **- SR-IOV(Single Root I/O

Virtualization) 가상 기능(VF) 장치만 이러한 방식으로 할당할 수 있습니다. 표준 단일 포트 PCI 또는 PCIe 이더넷 카드를 게스트 가상 머신에 할당하려면 기존 hostdev 장치 정의를 사용합니다.

네트워크 장치의 "intelligent passthrough"는 표준 hostdev 장치의 기능과 매우 유사합니다. 이 방법은 통과된 장치에 대한 MAC 주소와 가상 포트를 지정할 수 있다는 점입니다. 이러한 기능이 필요하지 않은 경우, SR-IOV를 지원하지 않는 표준 단일 포트 PCI, PCIe 또는 USB 네트워크 카드가 있는 경우(또는 게스트 가상 머신 도메인에 할당된 후 재설정 중에 구성된 MAC 주소가 손실됨) 또는 0.9.11 이전 버전의 libvirt를 사용하는 경우 표준 hostdev 를 사용하여 해당 장치를 게스트 가상 머신에 할당해야 합니다.

그림 20.44. 장치 - 네트워크 인터페이스- PCI 패스스루

```
...
<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>
...
```

20.16.9.8. 멀티 캐스트 터널

멀티 캐스트 그룹을 사용하여 가상 네트워크를 나타낼 수 있습니다. 네트워크 장치가 동일한 멀티 캐스트 그룹 내에 있는 모든 게스트 가상 머신은 여러 물리적 호스트 물리적 시스템에 있더라도 서로 통신합니다. 이 모드는 권한이 없는 사용자로 사용할 수 있습니다. 기본 DNS 또는 DHCP 지원이 없으며 나가는 네트워크 액세스가 없습니다. 나가는 네트워크 액세스를 제공하려면 게스트 가상 머신 중 하나에 적절한 라우팅을 제공하기 위해 첫 번째 4 네트워크 유형 중 하나에 연결된 두 번째 NIC가 있어야 합니다. 멀티 캐스트 프로토콜은 사용자 모드 Linux 게스트 가상 머신에서 사용하는 프로토콜과 호환됩니다. 사용되는 소스 주소는 멀티캐스트 주소 블록에서 사용해야 합니다. 멀티 캐스트 터널은 관리 툴을 사용하여 인터페이스 유형을 조작하고 mcast 로 설정/변경하고 mac 및 소스 주소를 제공하여 생성됩니다. 이 결과는 도메인 XML의 변경 사항에 표시됩니다.

그림 20.45. 장치 - 네트워크 인터페이스- 캐스트 터널

```
...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
    <source address='230.0.0.1' port='5558'/>
  </interface>
</devices>
...
```

20.16.9.9. TCP 터널

TCP 클라이언트/서버 아키텍처를 생성하는 것은 하나의 게스트 가상 머신이 네트워크의 서버 끝을 제공하고 다른 모든 게스트 가상 시스템이 클라이언트로 구성된 가상 네트워크를 제공하는 또 다른 방법입니다. 게스트 가상 머신 간의 모든 네트워크 트래픽은 서버로 구성된 게스트 가상 머신을 통해 라우팅됩니다. 이 모델은 권한이 없는 사용자에게도 사용할 수 있습니다. 기본 DNS 또는 DHCP 지원이 없으며 나가는 네트워크 액세스가 없습니다. 발신 네트워크 액세스를 제공하려면 게스트 가상 머신 중 하나에 연결된 두 번째 NIC가 있어야 첫 번째 4 네트워크 유형 중 하나에 적절한 라우팅을 제공해야 합니다. TCP 터널은 관리 도구를 사용하여 인터페이스 유형을 조작하고 서버 또는 클라이언트로 설정/변경하고 mac 및 소스 주소를 제공하여 생성됩니다. 이 결과는 도메인 XML의 변경 사항에 표시됩니다.

그림 20.46. 장치 - 네트워크 인터페이스- TCP 터널

```
...
<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>
...
```

20.16.9.10. NIC 드라이버별 옵션 설정

일부 NIC에는 조정 가능한 드라이버별 옵션이 있을 수 있습니다. 이러한 옵션은 인터페이스 정의의 드라이버 하위 요소의 속성으로 설정됩니다. 이러한 옵션은 도메인 XML의 다음 섹션을 구성하는 관리 도구를 사용하여 설정됩니다.

그림 20.47. 장치 - 네트워크 인터페이스- NIC 드라이버별 옵션 설정

```
<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>
...
```

현재 "virtio" NIC 드라이버에 다음 속성을 사용할 수 있습니다.

표 20.19. virtio NIC 드라이버 요소

매개변수	설명
name	선택적 **name** 속성은 사용할 백엔드 드라이버 유형을 강제 적용합니다. 값은 **qemu** (사용자 공간 백엔드) 또는 **vhost** (커널 백엔드)(커널에서 vhost 모듈을 제공해야 하는 커널 백엔드)일 수 있습니다. 커널 지원이 없는 vhost 드라이버가 필요합니다. vhost 드라이버가 있는 경우 기본 설정은 **vhost** 이며, 그렇지 않은 경우 자동으로 **qemu** 로 대체됩니다.
txmode	전송 버퍼가 가득 차 있을 때 패킷 전송을 처리하는 방법을 지정합니다. 값은 **iothread** 또는 **timer** 일 수 있습니다. **iothread** 로 설정하면 패킷 tx는 모두 드라이버 하단의 iothread에서 수행됩니다. 이 옵션은 **"tx=bh"** 를 **qemu** 명령줄 -device virtio-net-pci 옵션에 추가합니다. **타이머** 로 설정하면 tx 작업이 qemu에서 수행되며 현재 전송 가능한 것보다 더 많은 tx 데이터가 있는 경우 qemu가 다른 작업을 수행하기 전에 타이머가 설정됩니다. 타이머가 다른 작업을 수행할 때, 또 다른 시도는 더 많은 데이터를 전송하도록 시도합니다. 일반적으로 이 옵션을 단독으로 남겨 두는 경우 이를 변경하는 것이 절대적인 필수 사항입니다.
ioeventfd	사용자가 인터페이스 장치에 대한 도메인 I/O 비동기 처리를 설정할 수 있습니다. 기본값은 하이퍼바이저의 재량에 따라 다릅니다. 허용되는 값은 **설정 및 해제** 입니다. 이 옵션을 활성화하면 qemu가 게스트 가상 머신을 실행할 수 있으며 별도의 스레드에서 I/O를 처리할 수 있습니다. 일반적으로 게스트 가상 머신은 I/O 중에 시스템 CPU 사용률이 높아집니다. 반면 물리적 호스트 물리적 시스템의 과부하로 인해 게스트 가상 머신 I/O 대기 시간도 증가할 수 있습니다. 따라서 이 옵션을 변경하는 것이 절대적인 필수임을 확신하지 않는 한 이 옵션을 그대로 남겨 두어야 합니다.
event_idx	event_idx 속성은 장치 이벤트 처리의 일부 측면을 제어합니다. 값은 **on** 또는 **off** 일 수 있습니다. **를** 선택하면 인터럽트 수가 줄어들고 게스트 가상 머신에 대해 종료됩니다. 기본값은 **ON** 입니다. 이 동작이 낙관적 인 상황이 있는 경우 이 속성은 기능을 강제로 끌 수 있는 방법을 제공합니다. 매우 확신하지 않는 한 이 옵션을 그대로 두는 것이 절대적인 필요성입니다.

20.16.9.11. 대상 요소 덮어쓰기

대상 요소를 재정의하려면 관리 도구를 사용하여 도메인 XML을 다음과 같이 변경합니다.

그림 20.48. devices - 네트워크 인터페이스-대상 요소를 덮어씁니다.

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
  </interface>
</devices>
...
```

*대상을 지정하지 않으면 특정 하이퍼바이저가 생성된 **tun** 장치에 대한 이름을 자동으로 생성합니다. 이 이름은 수동으로 지정할 수 있지만, 이름은 **libvirt** 및 특정 하이퍼바이저에서 예약된 접두사인 **'vnet'** 또는 **'vif'**로 시작할 수 없습니다. 이러한 접두사를 사용하여 수동으로 지정된 대상은 무시됩니다.*

20.16.9.12. 부팅 순서 지정

*부팅 순서를 지정하려면 관리 툴을 사용하여 도메인 **XML**을 다음과 같이 변경합니다.*

그림 20.49. 부팅 순서 지정

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <boot order='1'/>
  </interface>
</devices>
...
```

*하이퍼바이저를 지원하는 경우 네트워크 부팅에 사용할 특정 **NIC**를 설정할 수 있습니다. 속성 순서는 부팅 시퀀스 중 장치를 시도하는 순서를 결정합니다. 장치별 부팅 요소는 **BIOS** 부트 로더 섹션의 일반 부트 요소와 함께 사용할 수 없습니다.*

20.16.9.13. 인터페이스롬 BIOS 구성

***Rom BIOS** 구성 설정을 지정하려면 관리 도구를 사용하여 도메인 **XML**을 다음과 같이 변경합니다.*

그림 20.50. 인터페이스롬 BIOS 구성

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </interface>
</devices>
...
```

이를 지원하는 하이퍼바이저의 경우 PCI 네트워크 장치의 Rom이 게스트 가상 머신에 표시되는 방식을 변경할 수 있습니다. bar 속성을 on 또는 off 로 설정할 수 있으며, 장치의om이 게스트 가상 머신의 메모리 맵에 표시되는지 여부를 결정할 수 있습니다. (PCI 문서의 경우 "로바" 설정은 Rom에 대한 기본 주소 등록의 존재를 제어합니다. rom bar를 지정하지 않으면 qemu 기본값이 사용됩니다(디렉터 버전의 qemu는 기본값을 사용했지만 최신 qemus의 기본값은)입니다. 선택적 파일 속성은 게스트 가상 머신이 장치의 ROM BIOS로 표시되도록 바이너리 파일을 가리키는 데 사용됩니다. 이는 네트워크 장치에 대한 대체 부트 롬을 제공하는 데 유용할 수 있습니다.

20.16.9.14. 서비스 품질

도메인 XML의 이 섹션에서는 서비스 품질 설정을 제공합니다. 들어오고 나가는 트래픽을 독립적으로 만들 수 있습니다. 대역폭 요소는 가장 많은 인바운드 및 최대 하나의 아웃바운드 자식 요소를 가질 수 있습니다. 이러한 하위 요소 중 하나를 벗어나면 해당 트래픽 방향에 QoS가 적용되지 않습니다. 따라서 도메인의 들어오는 트래픽만 셰이프하려면 인바운드만 사용하고 그 반대의 경우도 마찬가지입니다.

이러한 각 요소에는 하나의 필수 특성 평균 (또는 아래에 설명된 대로 바닥)이 있습니다. 평균은 형성되는 인터페이스의 평균 비트 속도를 지정합니다. 그런 다음 두 가지 선택적 속성이 있습니다. 최대 속도에서 데이터를 보낼 수 있는 최대 속도를 지정하는 최대 속도의 최대 속도를 지정하는 최대 속도이며 버스트 는 최대 속도에서 버스트될 수 있는 바이트 수를 지정합니다. 특성에 허용되는 값은 정수 번호입니다.

평균 및 최대 속성의 단위는 초당 킬로바이트이며 burst 는 킬로바이트 단위로만 설정됩니다. 또한 인바운드 트래픽은 필요한 경우Flaim 특성을 가질 수 있습니다.In addition, inbound traffic can optionally have a floor attribute. 이렇게 하면 셰이프된 인터페이스에 대한 처리량이 최소화됩니다. 플로어 를 사용하려면 모든 트래픽이 QoS 결정을 내릴 수 있는 한 지점을 통과해야 합니다. 따라서 인터페이스 type='network'/ 가 앞으로 ,nat, or no forward 를 가진 경우에만 사용될 수 있습니다. 가상 네트워크 내에서 연결된 모든 인터페이스는 최소한 인바운드 QoS 세트(최소 평균)를 보유해야 하지만, floor 속성은평균 을 지정할 필요가 없습니다. 그러나 최대 및 버스트 속성은 여전히 평균이 필요합니다. 현재 수신 qdiscs에는 클래스가 없을 수 있으므로 층은 인바운드 및 아웃 바운드 트래픽에만 적용될 수 있습니다.

QoS 구성 설정을 지정하려면 관리 도구를 사용하여 도메인 XML을 다음과 같이 변경합니다.

그림 20.51. 서비스 품질

```
...
<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>
...
```

20.16.9.15. VLAN 태그 설정 (지원되는 네트워크 유형에서만)

VLAN 태그 구성 설정을 지정하려면 관리 툴을 사용하여 도메인 **XML**을 다음과 같이 변경합니다.

그림 20.52. VLAN 태그 설정 (지원되는 네트워크 유형에서만)

```
...
<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>
...
```

게스트 가상 머신에서 사용하는 네트워크 연결이 게스트 가상 머신에 대해 **vlan** 태깅을 투명하게 지원하는 경우, 선택적 **vlan** 요소는 게스트 가상 머신의 네트워크 트래픽(**openvswitch** 및 **type='hostdev'** **SR-IOV** 인터페이스)에 적용하도록 하나 이상의 **vlan** 태그를 지정할 수 있습니다. **802.1QBH(vn-link)** 및 **802.1Qbg(VEPA)** 스위치는 게스트 가상 머신 트래픽을 특정 **vlans**에 태그하기 위한 고유한 방법(**VEPA**)을 제공합니다. 여러 태그의 사양(**vlan** 트렁크의 경우)을 허용하려면 하위 **element,** 태그, 사용할 **vlan** 태그(예: 태그 **id='42'**/)를 지정합니다. 인터페이스에 두 개 이상의 **vlan** 요소가 정의되어 있는 경우 사용자가 지정된 모든 태그를 사용하여 **VLAN** 트렁크를 수행하는 것으로 가정합니다. 단일 태그로 **vlan** 트렁크가 필요한 경우 최상위 **vlan** 요소에 선택적 속성 **trunk='yes'** 를 추가할 수 있습니다.

20.16.9.16. 가상 링크 상태 수정

이 요소는 가상 네트워크 링크의 상태를 설정하는 것을 의미합니다. 특성 상태에 가능한 값은 **up** 및 **down** 입니다. **down** 이 값으로 지정되면 인터페이스는 네트워크 케이블의 연결이 끊어진 것처럼 작동합

니다. 이 요소가 지정되지 않은 경우 기본 동작으로 링크 상태가 되게 합니다.

가상 링크 상태 구성 설정을 지정하려면 관리 툴을 사용하여 도메인 XML을 다음과 같이 변경합니다.

그림 20.53. 가상 링크 상태 수정

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
    <link state='down'/>
  </interface>
<devices>
...
```

20.16.10. 입력 장치

입력 장치를 사용하면 게스트 가상 머신 가상 머신의 그래픽 프레임 버퍼와 상호 작용할 수 있습니다. 프레임버버를 활성화하면 입력 장치가 자동으로 제공됩니다. 예를 들어 절대 커서 이동을 위한 그래픽 태블릿을 제공하는 것과 같이 장치를 명시적으로 추가할 수 있습니다.

입력 장치 구성 설정을 지정하려면 관리 툴을 사용하여 도메인 XML을 다음과 같이 변경합니다.

그림 20.54. 입력 장치

```
...
<devices>
  <input type='mouse' bus='usb'/>
</devices>
...
```

<input> 요소에는 하나의 필수 특성이 있습니다. **type: mouse** 또는 **tablet** 으로 설정할 수 있습니다. 전자는 상대 이동을 사용하는 동안 절대 커서 이동을 제공합니다. 선택적 **bus** 속성은 정확한 장치 유형을 구체화하는 데 사용할 수 있으며 **xen (para virtualized), ps2** 및 **usb** 로 설정할 수 있습니다.

input 요소에는 위에 설명된 대로 특정 **PCI** 슬롯에 장치를 연결할 수 있는 선택적 하위 요소 **<주소가>** 있습니다.

20.16.11. hub Devices

허브는 단일 포트를 여러 개 확장하여 장치를 호스트 물리적 시스템 시스템에 연결하는 데 사용할 수 있도록 하는 장치입니다.

*허브 장치 구성 설정을 지정하려면 관리 툴을 사용하여 도메인 **XML**을 다음과 같이 변경합니다.*

*그림 **20.55. hub devices***

```
...
<devices>
  <hub type='usb'/>
</devices>
...
```

***hub** 요소에는 하나의 필수 특성이 있습니다. 값이 **usb** 일 수 있는 형식입니다. **hub** 요소에는 특정 컨트롤러에 장치를 연결할 수 있는 **type='usb'**가 있는 선택적 하위 요소 주소가 있습니다.*

20.16.12. 그래픽 프레임 버퍼

*그래픽 장치를 사용하면 게스트 가상 머신 **OS**와 그래픽 상호 작용을 수행할 수 있습니다. 게스트 가상 시스템에는 일반적으로 관리자와의 상호 작용을 허용하도록 구성된 프레임버 또는 텍스트 콘솔이 있습니다.*

*그래픽 프레임버 버퍼 장치 구성 설정을 지정하려면 관리 툴을 사용하여 도메인 **XML**을 다음과 같이 변경합니다.*

*그림 **20.56.** 그래픽 프레임 버퍼*

```
...
<devices>
  <graphics type='sdl' display=':0.0'/>
  <graphics type='vnc' port='5904'>
    <listen type='address' address='192.0.2.1'/>
  </graphics>
  <graphics type='rdp' autoport='yes' multiUser='yes' />
  <graphics type='desktop' fullscreen='yes'/>
  <graphics type='spice'>
    <listen type='network' network='rednet'/>
  </graphics>
</devices>
...
```

***graphics** 요소에는 아래 설명과 같이 **sdl,vnc,rdp** 또는 **desktop** 값을 사용하는 필수 **type** 속성이 있*

습니다.

표 20.20. 그래픽 프레임버 버퍼 요소

매개변수	설명
sdl	그러면 호스트 물리적 시스템 데스크탑에 창이 표시되며, 3개의 선택적 인수를 사용할 수 있습니다. **디스플레이에** 사용할 표시 특성, 인증 식별자에 대한**xauth** 속성, 값을 수락하는 선택적 **전체 화면** 특성(**예** 또는 **없음**)이 필요합니다.
vnc	VNC 서버를 시작합니다. **port** 속성은 TCP 포트 번호(자동 할당되어야 함을 나타내는 레거시 구문으로 **-1** 포함)를 지정합니다. **autoport** 속성은 사용할 TCP 포트 자동 할당을 나타내는 새로운 기본 구문입니다. **listen** 속성은 서버가 수신 대기할 IP 주소입니다. **passwd** 속성은 일반 텍스트로 VNC 암호를 제공합니다. **keymap** 속성은 사용할 키맵을 지정합니다. 암호 유효성에 대한 제한을 설정하여 **타임 스탬프 passwdValidTo='2010-04-09T15:51:00'** 을 UTC로 지정할 수 있습니다. **연결된** 속성을 사용하면 암호 변경 중에 연결된 클라이언트를 제어할 수 있습니다. VNC는 **keep** 값만 허용하고 모든 하이퍼바이저에서 지원되지 않을 수 있습니다. QEMU는 listen/port를 사용하는 대신 unix 도메인 소켓 경로에서 수신 대기하도록 소켓 속성을 지원합니다.
SPICE	SPICE 서버를 시작합니다. **port** 속성은 TCP 포트 번호(-1를 사용하여 자동할 수 있음을 나타내는 레거시 구문으로) 지정하지만 **tlsPort** 는 대체 보안 포트 번호를 제공합니다. **autoport** 속성은 두 포트 번호의 자동 할당을 나타내는 새로운 기본 구문입니다. **listen** 속성은 서버가 수신 대기할 IP 주소입니다. **passwd** 속성은 일반 텍스트로 SPICE 암호를 제공합니다. **keymap** 속성은 사용할 키맵을 지정합니다. 암호 유효성에 대한 제한을 설정하여 **타임 스탬프 passwdValidTo='2010-04-09T15:51:00'** 을 UTC로 지정할 수 있습니다. **연결된** 속성을 사용하면 암호 변경 중에 연결된 클라이언트를 제어할 수 있습니다. SPICE는 계속 클라이언트 연결을 유지하고, 클라이언트 연결을 끊을 수 있으며, 암호 변경에 실패하게 됩니다. 모든 하이퍼바이저에서 지원되지 않습니다. **defaultMode** 속성은 기본 채널 보안 정책, 유효한 값은 **안전하지 않은** 보안 및 기본값**any** 입니다(가능한 경우 **안전하지** 않은 경우 안전하지만 보안 경로가 없는 경우 오류 발생 대신 **비보안** 으로 대체됨).

***SPICE*에 일반 및 *TLS* 보안 *TCP* 포트가 모두 구성되어 있는 경우 각 포트에서 실행할 수 있는 채널을 제한하는 것이 좋습니다. 이는 기본 그래픽 요소 내에 하나 이상의 *channel* 요소를 추가하여 달성됩니다.**

*유효한 채널 이름에는 주요 채널 이름,디스플레이,입력,커서,재생,기록, 스마트 카드; 및 **usbredir** 이 포함됩니다.*

*SPICE 구성 설정을 지정하려면 **mangement** 툴을 사용하여 도메인 **XML**을 다음과 같이 변경합니다.*

*그림 **20.57. SPICE** 구성*

```
<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>
```

*SPICE는 오디오, 이미지 및 스트리밍에 대한 변수 압축 설정을 지원합니다. 이러한 설정은 모든 요소에서 압축 속성을 사용하여 액세스할 수 있습니다. 이미지 압축을 설정하는 이미지(수용 **auto_glz, auto_lz, quick, glz, lz, off**), **wan**(수명, 절대, 절대)을 구성하여 이미지의 **JPEG** 압축을 위한 **jpeg** 를 사용할 수 있습니다(수신 이미지 압축 구성(수동, 절대, 절대) 및 외부 스트림 압축을 구성하는 경우 **zlib**.*

*스트리밍 모드는 **streaming** 요소에 의해 설정되며, **mode** 특성을 필터,**all** 또는 **off** 중 하나로 설정합니다.*

*또한 클립보드 요소에 의해 복사 및 붙여넣기 기능(**SPICE** 에이전트 사용)이 설정됩니다. 기본적으로 활성화되어 있으며 **copy paste** 속성을 **no** 로 설정하여 비활성화할 수 있습니다.*

*마우스 요소에 의해 설정되며, 해당 **mode** 특성을 서버 또는 클라이언트 중 하나로 설정합니다. 모드를 지정하지 않으면 **qemu** 기본값이 사용됩니다(클라이언트 모드).*

추가 요소는 다음과 같습니다.

*표 **20.21.** 추가 그래픽 프레임 버퍼 요소*

매개변수	설명

매개변수	설명
rdp	RDP 서버를 시작합니다. port 속성은 TCP 포트 번호(자동 할당되어야 함을 나타내는 레거시 구문으로 -1 포함)를 지정합니다. autoport 속성은 사용할 TCP 포트 자동 할당을 나타내는 새로운 기본 구문입니다. replaceUser 속성은 VM에 대한 여러 동시 연결이 허용되는지 여부를 결정하는 부울입니다. 기존 연결을 삭제해야 하는지 여부와 새 연결은 새 클라이언트가 단일 연결 모드로 연결할 때 VRDP 서버에서 설정해야 합니다.
desktop	이 값은 현재Space 도메인에 대해 예약되어 있습니다. "sdl"과 마찬가지로 호스트 물리적 머신 데스크탑에 창을 표시하지만 VirtualBox 뷰어를 사용합니다. "sdl"과 마찬가지로 선택적 특성 표시 및 전체 화면을 허용합니다.

매개변수	설명
listen	그래픽 유형 vnc와 spice에 대한 수신 대기 소켓을 설정하는 데 사용되는 주소 정보를 두는 대신 **listen** 속성, listen 특성, **listen** 이라는 **그래픽** 의 개별 하위 요소를 지정할 수 있습니다(위의 예제 참조). **listen 은 다음 속성을 허용합니다.** • **type - address 또는 network로 설정합니다. 이는 이 listen 요소가 직접 사용할 주소를 지정하는지 또는 네트워크 이름을 지정하여(다음에 수신 대기할 적절한 주소를 결정하는 데 사용됨)를 나타냅니다.** • **address - 이 속성에는 수신 대기할 IP 주소 또는 호스트 이름(DNS 쿼리를 통해 IP 주소로 확인됨)이 포함됩니다. 실행 중인 도메인의 "live" XML에서 이 속성은 type='network' 에서도 수신 대기에 사용되는 IP 주소로 설정됩니다.** • **network - type='network' 인 경우 network 속성에 구성된 네트워크 목록에 있는 네트워크 이름이 포함됩니다. 이름이 지정된 네트워크 구성을 검사하여 적절한 수신 대기 주소를 결정합니다. 예를 들어, 네트워크에 IPv4 주소가 구성에 있는 경우(예: 경로 유형, nat 또는 전달 유형(isolated)이 없는 경우) 네트워크의 구성에 나열된 첫 번째 IPv4 주소가 사용됩니다. 네트워크가 호스트 물리적 시스템 브릿지를 설명하는 경우, 해당 브리지 장치와 연관된 첫 번째 IPv4 주소가 사용되며, 네트워크가 'direct'(macvtap) 모드 중 하나를 설명하는 경우 첫 번째 전달 dev의 첫 번째 IPv4 주소가 사용됩니다.**

20.16.13. 비디오 장치

비디오 장치입니다.

비디오 장치 구성 설정을 지정하려면 관리 툴을 사용하여 도메인 XML을 다음과 같이 변경합니다.

그림 20.58. 비디오 장치

```
...
<devices>
  <video>
    <model type='vga' vram='8192' heads='1'>
      <acceleration accel3d='yes' accel2d='yes'/>
    </model>
  </video>
</devices>
...
```

graphics 요소에는 아래에 설명된 대로 "sdl", "vnc", "rdp" 또는 "desktop" 값을 사용하는 필수 유형 속성이 있습니다.

표 20.22. 그래픽 프레임버 버퍼 요소

매개변수	설명
video	**비디오 요소는 비디오** 장치를 설명하는 컨테이너입니다. 이전 버전과의 호환성을 위해 동영상이 설정되지 않지만 domain xml에 **그래픽** 요소가 있는 경우 libvirt는 게스트 가상 머신 유형에 따라 기본 **비디오를** 추가합니다. "ram" 또는 "vram"이 기본값을 제공하지 않으면 사용됩니다.
model	여기에는 사용 가능한 하이퍼바이저 기능에 따라 **vga**,**cirrus**,**vmvga**,**xen**,**vbox** 또는 **qxl** 값을 사용하는 필수 **유형** 속성이 있습니다. vram 및 heads가 있는 그림의 수를 사용하여 kibibytes(24바이트의 블록)에서 비디오 메모리 크기를 제공할 수도 있습니다.
가속	가속도가 지원되는 경우 가속 요소의 **accel3d** 및 **accel2d** 특성을 사용하여 활성화해야 합니다.
주소	선택적 주소 하위 요소를 사용하여 비디오 장치를 특정 PCI 슬롯에 연결할 수 있습니다.

20.16.14. 콘솔, 직렬, 병렬 및 채널 장치

문자 장치는 가상 머신과 상호 작용할 수 있는 방법을 제공합니다. 반가상화 콘솔, 직렬 포트, 병렬 포트 및 채널은 모두 문자 장치로 분류되므로 동일한 구문을 사용하여 표시됩니다.

consols, channel 및 기타 장치 구성 설정을 지정하려면 관리 도구를 사용하여 도메인 **XML**을 다음과 같이 변경합니다.

그림 **20.59.** 콘솔, 직렬, 병렬 및 채널 장치

```
...
<devices>
  <parallel type='pty'>
    <source path='/dev/pts/2'/>
    <target port='0'/>
  </parallel>
  <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>
...
```

이러한 각 지시문에서 최상위 요소 이름**(parallel, serial, console, channel)**은 게스트 가상 머신에 장치가 표시되는 방법을 설명합니다. 게스트 가상 머신 인터페이스는 **target** 요소에 의해 구성됩니다. 호스트 물리적 시스템에 제공되는 인터페이스는 최상위 요소의 **type** 속성에 제공됩니다. 호스트 물리적 시스템 인터페이스는 소스 요소에서 구성합니다. 소스 요소에는 레이블링이 소켓 경로에서 수행되는 방식을 덮어쓰는 선택적 **seclabel**이 포함될 수 있습니다. 이 요소가 없으면 보안 레이블은 도메인별 설정에서 상속됩니다. 각 문자 장치 요소에는 특정 컨트롤러 또는 **PCI** 슬롯에 장치를 연결할 수 있는 선택적 하위 요소 주소가 있습니다.

20.16.15. 게스트 가상 머신 인터페이스

문자 장치는 게스트 가상 시스템에 다음 유형 중 하나로 표시됩니다.

병렬 포트를 설정하려면 관리 도구를 사용하여 도메인 **XML**로 다음과 같이 변경합니다.

그림 20.60. 게스트 가상 머신 인터페이스 병렬 포트

```
...
 <devices>
  <parallel type='pty'>
   <source path='/dev/pts/2'/>
   <target port='0'/>
  </parallel>
 </devices>
 ...
```

<target> 에는 포트 번호를 지정하는 **port** 속성이 있을 수 있습니다. 포트가 **0**에서 시작하여 번호가 매겨집니다. 일반적으로 **0, 1** 또는 **2**개의 병렬 포트가 있습니다.

직렬 포트를 설정하려면 관리 도구를 사용하여 도메인 **XML**을 다음과 같이 변경합니다.

그림 20.61. 게스트 가상 머신 인터페이스 직렬 포트

```
...
<devices>
  <serial type='pty'>
    <source path='/dev/pts/3'/>
    <target port='0'/>
  </serial>
</devices>
...
```

<target> 에는 포트 번호를 지정하는 **port** 속성이 있을 수 있습니다. 포트가 **0**에서 시작하여 번호가 매겨집니다. 일반적으로 **0, 1** 또는 **2**개의 직렬 포트가 있습니다. 값에 대해 두 가지 선택 사항이 있는 선택적 **type** 속성도 있습니다. 하나는 **isa-serial** 이고 다른 하나는 **usb-serial** 입니다. **type** 이 없는 경우 **isa-serial** 은 기본적으로 사용됩니다. **usb-serial**의 **type='usb'** 가 있는 선택적 하위 요소 **<주소는>** 위에 설명된 특정 컨트롤러에 장치를 연결할 수 있습니다.

<console> 요소는 대화형 콘솔을 나타내는 데 사용됩니다. 사용 중인 게스트 가상 머신 유형에 따라 콘솔이 반가상화 장치이거나 다음 규칙에 따라 직렬 장치의 복제일 수 있습니다.

- **targetType** 속성이 설정되지 않은 경우 기본 장치 유형은 하이퍼바이저의 규칙에 따라 다릅니다. **XML**을 **libvirt**에 다시 쿼리할 때 기본 유형이 추가됩니다. 완전히 가상화된 게스트 가상 머신의 경우 기본 장치 유형은 일반적으로 직렬 포트입니다.

- **targetType** 속성이 **serial** 이면 **<serial>** 요소가 없는 경우 콘솔 요소가 **<serial>** 요소에 복사됩니다. **<serial>** 요소가 이미 있는 경우 **Console** 요소는 무시됩니다.

- *targetType 속성이 직렬 로 지정되지 않은 경우 일반적으로 처리됩니다.*

- *첫 번째 <콘솔> 요소만 serial 의 targetType 을 사용할 수 있습니다. 보조 콘솔은 모두 반가상화여야 합니다.*

- *s390에서 콘솔 요소는 sclp 또는 sclplm(line mode)의 targetType을 사용할 수 있습니다. SCLP는 s390의 기본 콘솔 유형입니다. SCLP 콘솔과 연결된 컨트롤러가 없습니다.*

아래 예에서 virtio 콘솔 장치는 게스트 가상 머신에 /dev/hvc[0-7]로 노출됩니다(자세한 내용은 http://fedoraproject.org/wiki/Features/VirtioSerial를 참조하십시오).

그림 20.62. 게스트 가상 머신 인터페이스 - virtio 콘솔 장치

```
...
<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>
...
```

콘솔이 직렬 포트로 제공되는 경우 <target> 요소는 직렬 포트와 동일한 특성을 갖습니다. 일반적으로 하나의 콘솔만 있습니다.

20.16.16. 채널

이는 호스트 물리적 시스템과 게스트 가상 머신 간의 개인 통신 채널을 나타내며 도메인 **xml**의 다음 섹션을 변경하는 관리 도구를 사용하여 게스트 가상 머신 가상 머신을 변경하여 조작됩니다.

그림 **20.63.** 채널

```
...
<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/qemu/f16x86_64.agent'/>
    <target type='virtio' name='org.qemu.guest_agent.0'/>
  </channel>
  <channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0'/>
  </channel>
</devices>
...
```

이는 다양한 방법으로 구현할 수 있습니다. 특정 유형의 **<채널은> <대상>** 요소의 **type** 특성에 지정됩니다. 채널 유형에는 다음과 같이 대상 속성이 다릅니다.

- **guestfwd -** 게스트 가상 시스템에서 지정된 **IP** 주소로 전송한 **TCP** 트래픽을 호스트 물리적 시스템의 채널 장치로 전달되도록 합니다. **target** 요소에는 **address** 및 **port** 속성이 있어야 합니다.

- **virtio -** 반가상화 **virtio** 채널. **<채널은> /dev/vport*** 아래의 게스트 가상 머신에 노출되며 선택적 요소 이름이지정된 경우 **/dev/virtio-ports/$name** (자세한 내용은 **http://fedoraproject.org/wiki/Features/VirtioSerial**을 참조하십시오). 선택적 요소 주소는 위에 설명된 특정 **type='virtio-serial'** 컨트롤러에 채널을 연결할 수 있습니다. **QEMU**를 사용하면 이름이 **"org.qemu.guest_agent.0"**인 경우 **libvirt**는 게스트 가상 머신에 설치된 게스트 가상 머신 에이전트와 상호 작용하여 게스트 가상 머신 종료 또는 파일 시스템 격리와 같은 작업을 수행할 수 있습니다.

- **spicevmc -** 반가상화 **SPICE** 채널. 도메인에는 그래픽 장치로 **SPICE** 서버도 있어야 합니다. 이 장치에는 기본 채널의 물리적 머신 비둘기 메시지 호스트도 있어야 합니다. 속성 **type='virtio'** 과 함께 **target** 요소가 있어야 합니다. 선택적 속성 이름은 게스트 가상 머신이 채널에 액세스하는 방법을 제어하고 기본값은 **name='com.redhat.spice.0'** 입니다. 선택적 **<address>** 요소는 채널을 특정 **type='virtio-serial'** 컨트롤러에 연결할 수 있습니다.

20.16.17. 호스트 물리적 시스템 인터페이스

문자 장치는 호스트 물리적 시스템에 다음 유형 중 하나로 표시됩니다.

표 20.23. 문자 장치 요소

매개변수	설명	XML 스니펫

도메인 로그 파일	문자 장치의 모든 입력을 비활성화하고 가상 시스템의 logfile로 출력을 보냅니다.	`<devices>`
		`<console type='stdio'>`
		`<target port='1'/>`
		`</console>`
		`</devices>`

장치 로그 파일	파일이 열리고 문자 장치로 전송된 모든 데이터가 파일에 기록됩니다.	`<devices>`
		`<serial type="file">`
		`<source path="/var/log/vm/vm-serial.log"/>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

가상 콘솔	가상 콘솔의 그래픽 프레임 버퍼에 문자 장치를 연결합니다. 이는 일반적으로 "ctrl+alt+3"과 같은 특수 핫키 시퀀스를 통해 액세스됩니다.	`<devices>`
		`<serial type='vc'>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

null 장치	void에 문자 장치를 연결합니다. 입력에 데이터가 제공되지 않습니다. 기록된 모든 데이터는 삭제됩니다.	`<devices>`
		`<serial type='null'>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

매개변수	설명	XML 스니펫
의사 TTY	Pseudo TTY는 **/dev/ptmx** 를 사용하여 할당됩니다. **virsh 콘솔** 과 같은 적절한 클라이언트는 직렬 포트와 로컬로 상호 작용하기 위해 연결할 수 있습니다.	<devices> <serial type="pty"> <source path="/dev/pts/3"/> <target port="1"/> </serial> </devices>
NB 특별 케이스	**<콘솔 type='pty'>** 인 경우 NB 특수 케이스는 TTY 경로도 최상위 **<콘솔>** 태그의 **tty='/dev/pts/3** 속성으로 중복됩니다. 이렇게 하면 **<콘솔>** 태그에 대한 기존 구문이 표시됩니다.	
호스트 물리적 머신 장치 프록시	문자 장치는 기본 물리적 문자 장치로 전달됩니다. 장치 유형이 일치해야 합니다. 에뮬레이션된 직렬 포트는 호스트 물리적 시스템 직렬 포트에만 연결해야 합니다. 직렬 포트를 병렬 포트에 연결하지 마십시오.	<devices> <serial type="dev"> <source path="/dev/ttyS0"/> <target port="1"/> </serial> </devices>
명명된 파이프	문자 장치는 명명된 파이프에 출력을 씁니다. 자세한 내용은 pipe(7) 매뉴얼 페이지를 참조하십시오.	<devices> <serial type="pipe"> <source path="/tmp/mypipe"/> <target port="1"/> </serial> </devices>
TCP 클라이언트/서버	문자 장치는 원격 서버에 연결하는 TCP 클라이언트 역할을 합니다.	<devices> <serial type="tcp"> <source mode="connect" host="0.0.0.0"

매개변수	설명	XML 스니펫
		service="2445"/> <protocol type="raw"/> <target port="1"/> </serial> </devices> 또는 TCP 서버가 클라이언트 연결을 대기 중인 경우. <devices> <serial type="tcp"> <source mode="bind" host="127.0.0.1" service="2445"/> <protocol type="raw"/> <target port="1"/> </serial> </devices> 또는 raw TCP 대신 telnet을 사용할 수 있습니다. 또한 telnets(보안 telnet) 및 tls를 사용할 수도 있습니다. <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>

매개변수	설명	XML 스니펫
UDP 네트워크 콘솔	문자 장치는 UDP netconsole 서비스 역할을 하며 패킷을 전송 및 수신합니다. 이것은 손실된 서비스입니다.	<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 도메인 소켓 클라이언트/서버	문자 장치는 로컬 클라이언트의 연결을 수락하는 UNIX 도메인 소켓 서버 역할을 합니다.	<devices> <serial type="unix"> <source mode="bind" path="/tmp/foo"/> <target port="1"/> </serial> </devices>

20.17. 사운드 장치

*가상 사운드 카드는 **sound** 요소를 사용하여 호스트 물리적 머신에 연결할 수 있습니다.*

*그림 **20.64.** 가상 사운드 카드*

```
...
<devices>
  <sound model='es1370'/>
</devices>
...
```

*사운드 요소에는 실제 사운드 장치가 에뮬레이션된 것을 나타내는 하나의 필수 속성인 **model** 이 있습니다. 유효한 값은 기본 하이퍼바이저에 한정되지만 일반적인 선택 사항은 **'es1370', 'sb16', 'ac97', 'ich6'** 입니다. 또한 **ich6** 모델의 사운드 요소에는 다양한 오디오 코덱을 오디오 장치에 첨부하기 위한 선택적인 하위 요소 **codec** 가 있을 수 있습니다. 지정하지 않으면 재생 및 레코딩을 허용하도록 기본 **codec**가 첨부됩니다. 유효한 값은 **'duplex'** (경로 및 줄 아웃) 및 '마이크아웃 ' (고유자 및 마이크로폰을 반전)입니다.*

그림 20.65. 사운드 장치

```
...
<devices>
  <sound model='ich6'>
    <codec type='micro'/>
  <sound/>
</devices>
...
```

각 사운드 요소에는 위에 설명된 특정 PCI 슬롯에 장치를 연결할 수 있는 선택적 하위 요소 <주소가> 있습니다.

20.18. 워치독 장치

워치독 요소를 통해 가상 하드웨어 워치독 장치를 게스트 가상 머신에 추가할 수 있습니다. <> 워치독 장치에는 게스트 가상 머신에 추가 드라이버 및 관리 데몬이 필요합니다. libvirt 구성에서 워치독을 활성화하면 자체적으로 유용한 작업은 없습니다. 현재 워치독이 실행되는 경우 지원 알림이 없습니다.

그림 20.66. 워치독 장치

```
...
<devices>
  <watchdog model='i6300esb'/>
</devices>
...

...
<devices>
  <watchdog model='i6300esb' action='poweroff'/>
</devices>
</domain>
```

다음 속성은 이 XML에서 선언됩니다.

- *모델 - 필요한 모델 속성은 실제 워치독 장치가 에뮬레이션된 것을 지정합니다. 유효한 값은 기본 하이퍼바이저에 따라 다릅니다.*

- *model 속성은 다음 값을 사용할 수 있습니다.*

  - *i6300esb - PCI Intel 6300baremetal를 에뮬레이션하는 권장 장치*

- *ib700 - wrong iBase IB700을 에뮬레이션*

- *동작 - 선택적 action 속성은 워치독이 만료될 때 수행할 작업을 설명합니다. 유효한 값은 기본 하이퍼바이저에 따라 다릅니다. action 속성에는 다음 값이 있을 수 있습니다.*

  - *reset - 기본 설정, 게스트 가상 머신 강제 재설정*

  - *shutdown - guest 가상 머신을 정상적으로 종료합니다(권장하지 않음)*

  - *poweroff - 게스트 가상 머신의 전원을 강제로 끕니다.*

  - *pause - 게스트 가상 머신 일시 정지*

  - *없음 - 아무것도 하지 않음*

  - *dump - guest 가상 머신을 자동으로 덤프합니다.*

'shutdown' 작업을 수행하려면 게스트 가상 머신이 ACPI 신호에 반응해야 합니다. 워치독이 만료된 종류의 상황에서는 게스트 가상 머신이 일반적으로 ACPI 신호에 응답할 수 없습니다. 따라서 'shutdown'을 사용하는 것은 권장되지 않습니다. 또한 덤프 파일을 저장할 디렉토리는 /etc/libvirt/qemu.conf 파일에 auto_dump_path로 설정할 수 있습니다.

20.19. 메모리 BALLOON 장치

가상 메모리 풍선 장치는 모든 Xen 및 KVM/QEMU 가상 머신에 추가됩니다. 이는 <memballoon> 요소라고 볼 수 있습니다. 적절한 경우 자동으로 추가되므로 특정 PCI 슬롯을 할당해야 하는 경우를 제외하고 게스트 가상 머신 XML에 이 요소를 명시적으로 추가할 필요가 없습니다. memballoon 장치를 명시적으로 비활성화해야하는 경우 model='none' 을 사용할 수 있습니다.

다음 예제가 KVM을 사용하여 자동으로 추가된 장치

그림 20.67. 메모리 풍선 장치

```
...
<devices>
  <memballoon model='virtio'/>
</devices>
...
```

다음은 정적 PCI 슬롯 2 요청이 있는 장치가 수동으로 추가되는 예입니다.

그림 20.68. 메모리 풍선 장치가 수동으로 추가됨

```
...
<devices>
  <memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
  </memballoon>
</devices>
</domain>
```

필수 모델 속성은 제공되는 풍선 장치의 유형을 지정합니다. 유효한 값은 가상화 플랫폼에 따라 다릅니다. 'virtio' 은 Xen 하이퍼바이저의 기본 설정인 KVM 하이퍼바이저 또는 'xen' 의 기본 설정인 'virtio'입니다.

20.20. 보안 레이블

<seclabel> 요소를 사용하면 보안 드라이버 작업을 제어할 수 있습니다. libvirt에서 고유한 보안 레이블을 자동으로 생성하는 '동적'의 기본 모드는 세 가지입니다. 여기서 libvirt는 고유한 보안 레이블을 자동으로 생성합니다. 애플리케이션/관리자가 레이블을 선택하거나, 제한을 해제하는 'none' 입니다. 동적 레이블 생성을 사용하면 libvirt는 항상 가상 머신과 연결된 모든 리소스의 레이블을 자동으로 다시 지정합니다. 정적 레이블 할당에서는 기본적으로 관리자 또는 애플리케이션에서 모든 리소스에 레이블이 올바르게 설정되어 있는지 확인해야 합니다. 그러나 원하는 경우 자동 재레이블을 활성화할 수 있습니다.

libvirt에서 둘 이상의 보안 드라이버를 사용하는 경우 각 드라이버마다 여러 seclabel 태그를 사용할 수 있으며, 각 태그에서 참조하는 보안 드라이버는 최상위 보안 레이블에 대해 특성 모델 Valid input XML 구성을 사용하여 정의할 수 있습니다.

그림 20.69. 보안 레이블

```
<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'/>
```

입력 XML에 'type' 속성이 제공되지 않으면 보안 드라이버 기본 설정이 사용되며 'none' 또는 'dynamic' 일 수 있습니다. <baselabel> 이 설정되어 있지만 'type' 이 설정되지 않은 경우 유형은 '동시 '로 간주됩니다. 자동 리소스가 활성 상태인 실행 중인 게스트 가상 머신의 XML을 볼 때 추가 XML 요소인 imagelabel 이 포함됩니다. 이는 출력 전용 요소이므로 사용자가 제공한 XML 문서에서 무시됩니다.

다음 요소는 다음 값으로 조작할 수 있습니다.

- *type - 정적 , 동적 또는 none 을 선택하여 libvirt가 고유한 보안 레이블을 자동으로 생성하는지 여부를 확인합니다.*
- *모델 - 현재 활성화된 보안 모델과 일치하는 유효한 보안 모델 이름입니다.*
- *레이블 다시 지정 - 예 또는 아니요 입니다. 동적 레이블 할당이 사용되는 경우 항상 예 여야 합니다. 정적 레이블을 할당하면 기본값은 no 입니다.*
- *<label> - 정적 라벨을 사용하는 경우 가상 도메인에 할당할 전체 보안 레이블을 지정해야 합니다. 콘텐츠 형식은 사용 중인 보안 드라이버에 따라 다릅니다.*
  - *selinux: SELinux 컨텍스트.*
  - *AppArmor: AppArmor 프로필.*

- **DAC:** 소유자와 그룹은 콜론으로 구분됩니다. 사용자/그룹 이름 또는 **uid**/**gid**로 모두 정의할 수 있습니다. 드라이버는 먼저 이러한 값을 이름으로 구문 분석하려고 하지만 선행 기호를 사용하여 드라이버를 **uid** 또는 **gid**로 구문 분석하도록 할 수 있습니다.

- **<baselabel> -** 동적 라벨을 사용하는 경우 선택적으로 기본 보안 레이블을 지정하는 데 사용할 수 있습니다. 콘텐츠 형식은 사용 중인 보안 드라이버에 따라 다릅니다.

- **<imagelabel> -** 출력 전용 요소이며, 가상 도메인과 연결된 리소스에 사용되는 보안 레이블을 표시합니다. 콘텐츠 형식은 레이블을 비활성화할 때 사용 중인 보안 드라이버를 사용하는 보안 드라이버에 따라 달라지며, 레이블링을 비활성화하여(파일이 보안 레이블링이 없는 경우 **NFS** 또는 기타 파일 시스템에 존재하는 경우 사용) 또는 대체 라벨을 요청하는 방식으로 특정 소스 파일 이름에 대해 수행되는 레이블을 미세 조정할 수 있습니다(관리 애플리케이션에서 일부 리소스를 공유하도록 허용하는 경우 특수 레이블이 생성됨). **seclabel** 요소가 최상위 도메인 할당이 아닌 특정 경로에 연결된 경우 특성 재레이블 또는 하위 요소 레이블만 지원됩니다.

20.21. 도메인 XML 구성 예

QEMU 에뮬레이션 게스트 가상 머신 AMD64 및 Intel

그림 20.70. 도메인 XML 구성 예

```
<domain type='qemu'>
  <name>QEmu-fedora-i686</name>
  <uuid>c7a5fdbd-cdaf-9455-926a-d65c16db1809</uuid>
  <memory>219200</memory>
  <currentMemory>219200</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='cdrom'/>
  </os>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='cdrom'>
      <source file='/home/user/boot.iso'/>
      <target dev='hdc'/>
      <readonly/>
    </disk>
    <disk type='file' device='disk'>
      <source file='/home/user/fedora.img'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
    </interface>
    <graphics type='vnc' port='-1'/>
  </devices>
</domain>
```

***KVM* 하드웨어가 *i686*에서 게스트 가상 머신 가속화**

그림 20.71. 도메인 XML 구성 예

```
<domain type='kvm'>
  <name>demo2</name>
  <uuid>4dea24b3-1d52-d8f3-2516-782e98a23fa0</uuid>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch="i686">hvm</type>
  </os>
  <clock sync="localtime"/>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/demo2.img'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
      <mac address='24:42:53:21:52:45'/>
    </interface>
    <graphics type='vnc' port='-1' keymap='de'/>
  </devices>
</domain>
```

21장. 문제 해결

이 장에서는 **Red Hat Enterprise Linux 6** 가상화 문제에 대한 일반적인 문제 및 솔루션에 대해 설명합니다.

가상화 기술과 관련된 일반적인 문제에 대한 이해를 돕기 위해 이 장을 읽어보십시오. 문제 해결은 문서에서 배우기 어려운 실습과 경험을 필요로 합니다. 문제 해결 기술을 개발하기 위해 **Red Hat Enterprise Linux 6**에서 가상화를 실험하고 테스트하는 것이 좋습니다.

이 문서에서 답을 찾을 수 없는 경우 가상화 커뮤니티의 온라인 응답이 있을 수 있습니다. **Linux** 가상화 웹 사이트 목록은 **B.1절. “온라인 리소스”** 를 참조하십시오.

21.1. 디버깅 및 문제 해결 툴

이 섹션에는 **System Administrator** 애플리케이션, 네트워킹 유틸리티 및 디버깅 툴이 요약되어 있습니다. 이러한 표준 시스템 관리 툴과 로그를 사용하여 문제 해결을 지원할 수 있습니다.

- **kvm_stat** - 참조 **21.4절. “kvm_stat”**
- **trace-cmd**
- **ftrace** 참조 **Red Hat Enterprise Linux 개발자 가이드**
- **vmstat**
- **iostat**
- **lsof**
- **systemtap**
- **crash**

- ***sysrq***

- ***sysrq t***

- ***sysrq w***

이러한 네트워킹 툴은 가상화 네트워킹 문제 해결을 지원할 수 있습니다.

- ***ifconfig***

- ***tcpdump***

  ***tcpdump** 명령 **'sniffs'** 네트워크 패킷. **tcpdump** 는 네트워크 비정상 및 네트워크 인증 문제를 찾는 데 유용합니다. **wireshark** 라는 그래픽 버전의 **tcpdump** 가 있습니다.*

- ***brctl***

  ***brctl** 은 **Linux** 커널에서 이더넷 브리지 구성을 검사하고 구성하는 네트워킹 도구입니다. 다음 예제 명령을 수행하기 전에 **root** 액세스 권한이 있어야 합니다.*

```
# 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
```

다음은 가상화 문제 해결에 유용한 몇 가지 유용한 명령입니다.

- **strace** 는 다른 프로세스에서 수신하고 사용하는 시스템 호출 및 이벤트를 추적하는 명령입니다.

- **vncviewer:** 서버 또는 가상 머신에서 실행 중인 **VNC** 서버에 연결합니다. **yum installwnvnc** 명령을 사용하여 **vncviwer** 를 설치합니다.

- **vncserver:** 서버에서 원격 데스크탑을 시작합니다. 원격 세션을 통해 **virt-manager**와 같은 그래픽 사용자 인터페이스를 실행할 수 있는 기능을 제공합니다. **yum installpxevnc-server** 명령을 사용하여 **vncserver** 를 설치합니다.

21.2. 재해 복구 준비

가능한 경우 날씨 또는 다른 이유로 인해 장비가 손상되는 상황을 준비하는 것이 가장 좋습니다. 호스트 물리적 시스템에서 다음 파일 및 디렉터리를 백업을 수행하는 것이 좋습니다.

- **/etc/libvirt** 디렉토리에서 모든 파일.

- **/var/lib/libvirt** 디렉토리에서 다음 항목을 백업하십시오.

  - 현재 **dnsmasq DHCP** 리스는 **/var/lib/libvirt/dnsmasq**에 있습니다.

  - **/var/lib/libvirt/network**에 있는 실행 중인 가상 네트워크 구성 파일

  - 게스트의 현재 상태를 저장할 때 **virt-manager** 에서 생성한 게스트 가상 머신 파일(있는 경우). 해당 파일은 **/var/lib/libvirt/qemu/save/** 디렉터리에서 찾을 수 있습니다. **virsh save** 명령을 사용하여 가상 시스템을 생성하는 경우 사용자가 **virsh save** 에 대해 지정된 위치에서 파일을 찾을 수 있습니다.

  - **qemu-img create** 및 **virsh snapshot-create** 명령을 통해 생성된 게스트 가상 시스템

스냅샷 파일은 사용자가 명령에 지정된 위치에 있습니다.

- **virt-manager** 에서 생성한 게스트 가상 머신 디스크 이미지(있는 경우)는 **/var/lib/libvirt/images/** 디렉토리에 있습니다. **virsh pool-define** 명령이 가상 스토리지를 생성하는 데 사용된 경우, 이미지 파일은 사용자가 **virsh pool-define** 에 지정된 위치에서 찾을 수 있습니다. 게스트 이미지 파일을 백업하는 방법에 대한 지침은 절차 21.1. “재해 복구를 위해 게스트 가상 머신의 디스크 이미지 백업 생성” 에 설명된 단계를 사용하십시오.

- 브리지를 사용하는 경우 **/etc/sysconfig/network-scripts/ifcfg-<bridge_name**에 있는 파일도 백업해야 합니다.

- 선택적으로 **/var/lib/libvirt/qemu/dump** 에 있는 게스트 가상 머신 코어 덤프 파일은 실패의 원인을 분석하기 위해 사용할 수도 있습니다. 그러나 이러한 파일은 일부 시스템에서 매우 커질 수 있습니다.

절차 21.1. 재해 복구를 위해 게스트 가상 머신의 디스크 이미지 백업 생성

다음 절차에서는 다양한 디스크 이미지 유형을 백업하는 방법에 대해 설명합니다.

1. 게스트 가상 머신 디스크 이미지만 백업하려면 **/var/lib/libvirt/images** 에 있는 파일을 백업하십시오. **LVM** 논리 볼륨으로 게스트 가상 머신 디스크 이미지를 백업하려면 다음 명령을 실행합니다.

```
# lvcreate --snapshot --name snap --size 8G /dev/vg0/data
```

이 명령은 **64G** 볼륨의 일부로 크기가 **8G**인 **snap**라는 스냅샷 볼륨을 생성합니다.

2. 이 명령과 유사한 명령을 사용하여 스냅샷에 대한 파일을 생성합니다.

```
# mkdir /mnt/virt.snapshot
```

3. 다음 명령을 사용하여 생성한 디렉터리 및 스냅샷 볼륨을 마운트합니다.

```
# mount /dev/vg0/snap /mnt/virt.snapshot
```

4. 다음 명령 중 하나를 사용하여 볼륨을 백업합니다.

a. # tar -pzc -f /mnt/backup/virt-snapshot-MM-DD-YYYY.tgz /mnt/virt.snapshot++++++++++++

b. # rsync -a /mnt/virt.snapshot/ /mnt/backup/virt-snapshot.MM-DD-YYYY/

21.3. VIRSH DUMP 파일 생성

virsh dump 명령을 실행하면 게스트 가상 시스템의 코어를 파일에 덤프하도록 요청을 전송하여 가상 시스템의 오류를 진단할 수 있습니다. 이 명령을 실행하려면 **corefilepath** 인수로 지정된 파일과 경로에 대한 적절한 권한을 수동으로 확인해야 할 수 있습니다. **virsh dump** 명령은 **coredump**(또는 **crash** 유틸리티)와 유사합니다. **virsh dump** 파일을 생성하려면 다음을 실행합니다.

```
#virsh dump <domain> <corefilepath> [--bypass-cache] { [--live] | [--crash] | [--reset] } [--verbose] [--memory-only]
```

도메인(게스트 가상 시스템 도메인 이름) 및 **corefilepath**(새로 생성된 코어 덤프 파일의 위치)는 필수지만 다음 인수는 선택 사항입니다.

- **--live** 는 실행 중인 머신에 덤프 파일을 생성하고 일시 중지하지 않습니다.
- **--crash** 는 **guest** 가상 머신을 중지하고 덤프 파일을 생성합니다. 주요 차이점은 게스트 가상 머신이 중지됨으로 나열되지 않으며 이유는 **Crashed**입니다. **virt-manager** 에서는 상태가 **Paused**로 나열됩니다.
- **--reset** 은 성공적인 덤프로 게스트 가상 머신을 재설정합니다. 이 세 스위치는 함께 사용할 수 없습니다.
- **--bypass-cache** 는 **O_DIRECT**를 사용하여 파일 시스템 캐시를 바이패스합니다.
- **--memory-only** 덤프 파일은 **elf** 파일로 저장되며 도메인의 메모리 및 **cpu** 공통 레지스터 값만 포함합니다. 이 옵션은 도메인이 호스트 장치를 직접 사용하는 경우 매우 유용합니다.
- **--verbose** 는 덤프의 진행 상황을 표시합니다.

전체 덤프 프로세스는 **virsh domjobinfo** 명령을 사용하여 모니터링할 수 있으며 **virsh domjobabort** 을 실행하여 취소할 수 있습니다.

21.4. KVM_STAT

kvm_stat 명령은 **kvm** 커널 모듈에서 런타임 통계를 검색하는 **python** 스크립트입니다. **kvm_stat** 명령은 **kvm** 에 표시되는 게스트 동작을 진단하는 데 사용할 수 있습니다. 특히 게스트와 관련된 성능 관련 문제는 다음과 같습니다. 현재 보고된 통계는 전체 시스템에 대한 것이며 실행 중인 모든 게스트의 동작이 보고됩니다. 이 스크립트를 실행하려면 **qemu-kvm-tools** 패키지를 설치해야 합니다.

kvm_stat 명령을 사용하려면 **kvm** 커널 모듈이 로드되고 **debugfs** 가 마운트됩니다. 이러한 기능 중 하나를 활성화하지 않으면 명령은 **debugfs** 또는 **kvm** 모듈을 활성화하는 데 필요한 단계를 출력합니다. 예를 들어 다음과 같습니다.

```
# kvm_stat
Please mount debugfs ('mount -t debugfs debugfs /sys/kernel/debug')
and ensure the kvm modules are loaded
```

필요한 경우 **debugfs** 를 마운트합니다.

```
# mount -t debugfs debugfs /sys/kernel/debug
```

kvm_stat 출력

kvm_stat 명령은 모든 게스트 및 호스트에 대한 통계를 출력합니다. 명령이 종료될 때까지 출력이 업데이트됩니다(**Ctrl+c** 또는 **q** 키 사용).

```
# kvm_stat

kvm statistics

efer_reload               94       0
exits                  4003074   31272
fpu_reload             1313881   10796
halt_exits               14050     259
halt_wakeup               4496     203
host_state_reload      1638354   24893
hypercalls                   0       0
insn_emulation         1093850    1909
insn_emulation_fail          0       0
invlpg                   75569       0
io_exits               1596984   24509
irq_exits                21013     363
irq_injections           48039    1222
irq_window               24656     870
largepages                   0       0
mmio_exits               11873       0
mmu_cache_miss           42565       8
mmu_flooded              14752       0
mmu_pde_zapped           58730       0
```

```
mmu_pte_updated          6      0
mmu_pte_write          138795      0
mmu_recycled             0      0
mmu_shadow_zapped       40358      0
mmu_unsync             793      0
nmi_injections           0      0
nmi_window               0      0
pf_fixed            697731   3150
pf_guest            279349      0
remote_tlb_flush         5      0
request_irq              0      0
signal_exits             1      0
tlb_flush           200190      0
```

변수에 대한 설명:

efer_reload

***EFER(Extended Feature Enable Register)**가 다시 로드되는 수입니다.*

종료

*모든 **VMEXIT** 호출 수입니다.*

fpu_reload

***VMENTRY** 가 **FPU** 상태를 다시 로드한 횟수입니다. 게스트가**FPU**(유동 포인트 단위)를 사용하는 경우 **fpu_reload** 가 증가합니다.*

halt_exits

중지 호출으로 인한 게스트 종료 수입니다. 이러한 유형의 종료는 일반적으로 게스트가 유휴 상태일 때 표시됩니다.

halt_wakeup

중지에서 발생한 오류 발생 수 입니다.

host_state_reload

*호스트 상태에 대한 전체 재로드 수(현재 **MSR** 설정 및 게스트 **MSR** 읽기)입니다.*

하이퍼 호출

게스트 하이퍼바이저 서비스 호출 수입니다.

insn_emulation

호스트에서 에뮬레이션한 게스트 명령 수입니다.

insn_emulation_fail

*실패한 **insn_emulation** 시도 수입니다.*

io_exits

I/O 포트 액세스에서 종료되는 게스트 수입니다.

irq_exits

외부 인터럽트로 인해 종료 수입니다.

irq_injections

게스트에 전송된 인터럽트 수입니다.

irq_window

게스트 수는 적용되지 않은 인터럽트 창에서 종료됩니다.

largepages

현재 사용 중인 큰 페이지 수입니다.

mmio_exits

***Memory mapped I/O (MMIO)** 액세스로 인해 게스트 수가 종료됩니다.*

mmu_cache_miss

*생성된 **KVM MMU** 섀도우 페이지 수입니다.*

mmu_flooded

MMU 페이지에 대한 과도한 쓰기 작업의 탐지 수입니다. 이 수는 개별 쓰기 작업이 아닌 쓰기 작업을 감지했습니다.

mmu_pde_zapped

PDE(Page Directory entry) 삭제 작업 수입니다.

mmu_pte_updated

페이지 테이블 항목 (PTE) 삭제 작업 수.

mmu_pte_write

게스트 페이지 테이블 항목(PTE) 쓰기 작업 수입니다.

mmu_recycled

회수할 수 있는 그림자 페이지 수입니다.

mmu_shadow_zapped

유효하지 않은 그림자 페이지 수.

mmu_unsync

아직 연결되지 않은 동기화되지 않은 페이지 수입니다.

nmi_injections

nMI(Non-maskable Interrupt) 삽입 수입니다.

nmi_window

게스트 수는 (거울) NMI(Non-maskable Interrupt) 창에서 종료됩니다.

pf_fixed

고정되지 않은 페이지 테이블 항목(PTE) 수입니다.

pf_guest

게스트에 삽입된 페이지 폴트 수입니다.

remote_tlb_flush

원격(sibling CPU) Translation lookaside Buffer (TLB) 플러시 요청 수입니다.

request_irq

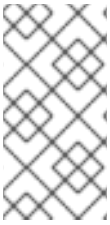
게스트 중단 창 요청 수가 종료됩니다.

signal_exits

호스트에서 보류 중인 신호 때문에 게스트 수가 종료됩니다.

tlb_flush

*하이퍼바이저에서 수행하는 **tlb_flush** 작업의 수입니다.*



참고

***kvm_stat** 명령의 출력 정보는 **/sys/kernel/debug/kvm/** 디렉터리에 있는 의사 파일로 **KVM** 하이퍼바이저에 의해 내보냅니다.*

21.5. 게스트 가상 시스템 종료 실패

*전통적으로 **virsh shutdown** 명령을 실행하면 **power** 버튼 **ACPI** 이벤트가 전송되므로 다른 사람이 물리적 시스템에서 전원 버튼을 누를 때와 동일한 작업을 복사합니다. 모든 물리적 컴퓨터에서는 이 이벤트를 처리하기 위해 **OS**에 달려 있습니다. 이전에는 운영 체제가 자동으로 종료됩니다. 오늘날 가장 일반적인 조치는 수행해야 할 작업을 묻는 대화 상자를 표시하는 것입니다. 일부 운영 체제는 특히 사용자가 로그인하지 않은 경우에도 이 이벤트를 완전히 무시합니다. 이러한 운영 체제가 게스트 가상 시스템에 설치된 경우 **virsh shutdown** 을 실행하면 작동하지 않습니다(무시되거나 가상 디스플레이에 대화 상자가 표시됨). 그러나 **qemu-guest-agent** 채널이 게스트 가상 머신에 추가되고 이 에이전트가 게스트 가상 머신의 **OS** 내에서 실행 중인 경우 **virsh shutdown** 명령은 **ACPI** 이벤트를 보내는 대신 게스트 **OS**를 종료하도록 요청합니다. 에이전트는 게스트 가상 시스템 **OS** 내부에서 종료를 호출하고 모든 것이 예상대로 작동합니다.*

*절차 **21.2.** 게스트 가상 머신에서 게스트 에이전트 채널 구성*

1. 게스트 가상 머신을 중지합니다.

2. 게스트 가상 머신용 도메인 XML을 열고 다음 스니펫을 추가합니다.

그림 21.1. 게스트 에이전트 채널 구성

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

3. **virsh start [domain]** 을 실행하여 게스트 가상 시스템을 시작합니다.

4. 게스트 가상 머신(**yum install qemu-guest-agent**)에 **qemu-guest-agent**를 설치하고 서비스(**qemu-guest-agent.service**)로 부팅할 때마다 자동으로 실행되도록 합니다. 자세한 내용은 *10장. QEMU-img 및 QEMU 게스트 에이전트* 를 참조하십시오.

21.6. 직렬 콘솔 문제 해결

Linux 커널은 직렬 포트에 정보를 출력할 수 있습니다. 이는 비디오 장치 또는 헤드리스 서버의 커널 패닉 및 하드웨어 문제를 디버깅하는 데 유용합니다. 이 섹션의 하위 섹션에서는 **KVM** 하이퍼바이저를 사용하여 호스트 물리적 시스템의 직렬 콘솔 출력 설정을 다룹니다.

이 섹션에서는 완전히 가상화된 게스트의 직렬 콘솔 출력을 활성화하는 방법에 대해 설명합니다.

완전한 가상화된 게스트 직렬 콘솔 출력은 **virsh console** 명령을 사용하여 볼 수 있습니다.

완전히 가상화된 게스트 직렬 콘솔에는 몇 가지 제한 사항이 있습니다. 현재 제한 사항은 다음과 같습니다.

- 출력 데이터는 삭제하거나 스크램블될 수 있습니다.

직렬 포트는 **Linux**에서 **ttyS0** 또는 **Windows**의 **COM1** 이라고 합니다.

정보를 가상 직렬 포트로 출력하도록 가상화된 운영 체제를 구성해야 합니다.

완전히 가상화된 **Linux** 게스트의 커널 정보를 도메인에 출력하려면 **/boot/grub/grub.conf** 파일을 수정합니다. 커널 행에 다음을 추가합니다. **console=tty0 console=ttyS0,115200.**

```
title Red Hat Enterprise Linux Server (2.6.32-36.x86-64)
 root (hd0,0)
 kernel /vmlinuz-2.6.32-36.x86-64 ro root=/dev/volgroup00/logvol00 \
 console=tty0 console=ttyS0,115200
 initrd /initrd-2.6.32-36.x86-64.img
```

게스트를 재부팅합니다.

호스트에서 다음 명령을 사용하여 직렬 콘솔에 액세스합니다.

```
# virsh console
```

virt-manager 를 사용하여 가상 텍스트 콘솔을 표시할 수도 있습니다. 게스트 콘솔 창의 텍스트 콘솔에서 직렬 **1** 을 선택합니다.

21.7. 가상화 로그 파일

- 각 완전 가상화된 게스트 로그는 **/var/log/libvirt/qemu/** 디렉토리에 있습니다. 각 게스트 로그의 이름은 **GuestName.log**로 지정되며 크기 제한에 도달하면 정기적으로 압축됩니다.

Virtual Machine Manager에 오류가 발생하면 **$HOME/.virt-manager** 디렉터리에 상주하는 **virt-manager.log** 파일에서 생성된 데이터를 검토할 수 있습니다.

21.8. 장치 오류 루프

파일 기반 게스트 이미지를 사용하는 경우 구성된 루프 장치 수를 늘려야 할 수 있습니다. 기본 구성을 사용하면 최대 **8**개의 활성 루프 장치를 사용할 수 있습니다. 파일 기반 게스트 또는 루프 장치가 **8**개 이상 필요한 경우 **/etc/modprobe.d/**디렉터리에서 구성된 루프 장치 수를 조정할 수 있습니다. 다음 행을 추가합니다.

```
options loop max_loop=64
```

이 예제에서는 64를 사용하지만 다른 숫자를 지정하여 최대 루프 값을 설정할 수 있습니다. 또한 시스템에서 루프 장치 지원 게스트를 구현해야 할 수도 있습니다. 전체 가상화 시스템에 루프 장치 지원 게스트를 사용하려면 phy: device 또는 file: file 명령을 사용합니다.

21.9. 실시간 마이그레이션 오류

실시간 마이그레이션으로 인해 메모리 콘텐츠가 반복적으로 다시 전송되도록 하는 경우가 있을 수 있습니다. 이 프로세스에서는 게스트가 메모리에 지속적으로 기록되어 마이그레이션 속도가 느려지는 상태가 됩니다. 이 문제가 발생할 경우 게스트에서 초당 수십MB 이상을 작성하는 경우 실시간 마이그레이션이 완료되지 못할 수 있습니다. 이 문제는 현재 Red Hat Enterprise Linux 6에서 해결될 예정이 아니며 Red Hat Enterprise Linux 7에서 수정될 예정입니다.

현재 실시간 마이그레이션 구현에는 기본 마이그레이션 시간이 30ms로 설정되어 있습니다. 이 값은 왼쪽을 전송하기 위해 마이그레이션이 끝날 때 게스트 일시 중지 시간을 결정합니다. 값이 높으면 실시간 마이그레이션이 수월할 확률을 증가시킵니다.

21.10. BIOS에서 INTEL VT-X 및 AMD-V 가상화 하드웨어 확장 활성화

참고

전문 지식을 확장하려면 Red Hat Virtualization(RH318) 교육 과정에 관심이 있을 수 있습니다.

이 섹션에서는 사용되지 않는 경우 하드웨어 가상화 확장을 식별하고 BIOS에서 활성화하는 방법을 설명합니다.

BIOS에서 Intel VT-x 확장 기능을 비활성화할 수 있습니다. 특정 노트북 공급 업체가 CPU에서 기본적으로 Intel VT-x 확장을 비활성화했습니다.

AMD-V의 경우 BIOS에서 가상화 확장을 비활성화할 수 없습니다.

비활성화된 가상화 확장 활성화에 대한 지침은 다음 섹션을 참조하십시오.

BIOS에서 가상화 확장 기능이 활성화되어 있는지 확인합니다. Intel VT 또는 AMD-V의 BIOS 설정은 일반적으로 Chipset 또는 Processor 메뉴에 있습니다. 메뉴 이름은 이 가이드에 따라 다를 수 있으며, 가상화 확장 설정은 보안 설정 또는 기타 표준 메뉴 이름에서 찾을 수 있습니다.

절차 21.3. BIOS에서 가상화 확장 활성화

1. *컴퓨터를 재부팅하고 시스템의 BIOS 메뉴를 엽니다. 이 작업은 일반적으로 시스템에 따라 삭제 키, F1 키 또는 Alt 및 F4 키를 눌러 수행할 수 있습니다.*

2. *BIOS에서 가상화 확장 활성화*

   참고

   아래의 대부분의 단계는 마더보드, 프로세서 유형, 칩셋 및 OEM에 따라 다를 수 있습니다. 시스템 구성에 대한 올바른 정보는 시스템 관련 설명서를 참조하십시오.

   a. *Processor 하위 메뉴를 열고 Chipset,Advanced CPU Configuration 또는 Northbridge 에서 프로세서 설정 메뉴를 숨길 수 있습니다.*

   b. *Intel VT-x라고도 하는 Intel Virtualization Technology 를 활성화합니다. AMD-V 확장은 BIOS에서 비활성화할 수 없으며 이미 활성화되어 있어야 합니다. 가상화 확장 기능은 OEM 및 시스템 BIOS에 따라 가상화 확장 기능,Vanderpool 또는 기타 다양한 이름으로 레이블이 지정될 수 있습니다.*

   c. *옵션을 사용할 수 있는 경우 Intel VT-d 또는 AMD IOMMU를 활성화합니다. Intel VT-d 및 AMD IOMMU는 PCI 장치 할당에 사용됩니다.*

   d. *Save & Exit를 선택합니다.*

3. *시스템을 재부팅합니다.*

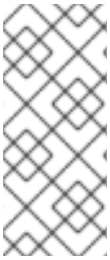
4. *머신이 부팅되면 cat /proc/cpuinfo |grep -E "vmx|svm" 을 실행합니다. --color 를 지정하는 것은 선택 사항이지만 검색 용어를 강조 표시하려는 경우 유용합니다. 명령이 출력되면 이제 가상화 확장이 활성화됩니다. 출력이 없는 경우 시스템에 가상화 확장 기능이 없거나 올바른 BIOS 설정이 활성화되어 있지 않을 수 있습니다.*

21.11. KVM 네트워킹 성능

기본적으로 KVM 가상 시스템에는 가상 Realtek 8139(rtl8139) NIC(네트워크 인터페이스 컨트롤러)가 할당됩니다. Red Hat Enterprise Linux 게스트는 기본적으로 virtio NIC를 할당하지만 Windows 게스트 또는 게스트 유형은 지정되지 않습니다.

rtl8139 가상화 NIC는 대부분의 환경에서 잘 작동하지만 이 장치는 10 기가비트 이더넷과 같은 일부 네트워크에서 성능 저하 문제가 발생할 수 있습니다.

성능을 향상시키기 위해 반가상화 네트워크 드라이버로 전환할 수 있습니다.



참고

가상화된 Intel PRO/1000(e1000) 드라이버도 에뮬레이션된 드라이버 선택으로 지원됩니다. e1000 드라이버를 사용하려면 아래 절차의 virtio 를 e1000 으로 교체합니다. 최상의 성능을 위해서는 virtio 드라이버를 사용하는 것이 좋습니다.

절차 21.4. virtio 드라이버로 전환

1. 게스트 운영 체제를 종료합니다.

2. virsh 명령을 사용하여 게스트의 설정 파일을 편집합니다(여기서 GUEST 는 게스트 이름임).

```
# virsh edit GUEST
```

virsh edit 명령은 $EDITOR 쉘 변수를 사용하여 사용할 편집기를 결정합니다.

3. 구성의 네트워크 인터페이스 섹션을 찾습니다. 이 섹션은 아래 코드 조각과 유사합니다.

```
<interface type='network'>
  [output truncated]
  <model type='rtl8139' />
</interface>
```

4. model 요소의 type 속성을 'rtl8139' 에서 'virtio' 으로 변경합니다. 이렇게 하면 드라이버가 rtl8139 드라이버에서 e1000 드라이버로 변경됩니다.

-

```
<interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

5. 변경 사항을 저장하고 텍스트 편집기를 종료합니다.

6. 게스트 운영 체제를 다시 시작합니다.

다른 네트워크 드라이버를 사용하여 새 게스트 생성

또는 다른 네트워크 드라이버를 사용하여 새 게스트를 생성할 수 있습니다. 네트워크 연결을 통해 게스트를 설치하는 데 문제가 있는 경우 이 작업이 필요할 수 있습니다. 이 방법을 사용하려면 하나 이상의 게스트가 이미 생성되어(CD 또는 DVD에서 설치될 수 있음) 템플릿으로 사용해야 합니다.

1. 기존 게스트에서 XML 템플릿을 생성합니다(이 예에서는 Guest1).

```
# virsh dumpxml Guest1 > /tmp/guest-template.xml
```

2. XML 파일을 복사 및 편집하고 가상 시스템 이름, UUID, 디스크 이미지, MAC 주소 및 기타 고유한 매개변수 등 고유한 필드를 업데이트합니다. UUID 및 MAC 주소 행을 삭제할 수 있으며 virsh는 UUID 및 MAC 주소를 생성합니다.

```
# cp /tmp/guest-template.xml /tmp/new-guest.xml
# vi /tmp/new-guest.xml
```

네트워크 인터페이스 섹션에 model 행을 추가합니다.

```
<interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

3. 새 가상 머신을 생성합니다.

```
# virsh define /tmp/new-guest.xml
# virsh start new-guest
```

21.12. LIBVIRT를 사용하여 외부 스냅샷 생성 해결 방법

QEMU 게스트의 스냅샷에는 두 가지 클래스가 있습니다. 내부 스냅샷은 qcow2 파일에 완전히 포함되며 libvirt 에서 완전히 지원하므로 스냅샷을 생성, 삭제, 되돌릴 수 있습니다. 이 설정은 특히 옵션이 지정되지 않은 경우 스냅샷을 만들 때 libvirt 에서 사용하는 기본 설정입니다. 이 파일 유형은 스냅샷을 만드는데 다른 파일보다 약간 오래 걸리지만, libvirt 에서 qcow2 디스크를 사용해야 합니다. 이 파일 유형의 또 다른 단점은 qcow2 디스크의 QEMU 개선 사항을 받을 수 없다는 것입니다.

반면, 외부 스냅샷은 원래 디스크 이미지 유형에서 작동하며 게스트 다운타임 없이 수행할 수 있으며 QEMU에서 적극적인 개선 사항을 받을 수 있습니다. libvirt 에서는 --disk-only 옵션을 snapshot-create-as 에 사용할 때 생성됩니다(또는 동일한 작업을 수행하는 snapshot-create 에 명시적 XML 파일을 지정할 때). 현재 외부 스냅샷은 libvirt 에서 생성할 수 있지만 더 이상 아무것도 수행할 수 없기 때문에 단방향 작업입니다.

21.13. 일본어 키보드를 사용하는 게스트 콘솔에서 누락된 문자

Red Hat Enterprise Linux 6 호스트에서 일본어 키보드를 시스템에 로컬로 연결하면 게스트 콘솔에 밑줄(_ 문자)이 올바르게 표시되지 않을 수 있습니다. 이는 기본적으로 필수 키맵이 올바르게 설정되지 않았기 때문에 발생합니다.

Red Hat Enterprise Linux 3 및 Red Hat Enterprise Linux 6 게스트의 경우 관련 키를 누르면 일반적으로 오류 메시지가 생성되지 않습니다. 그러나 Red Hat Enterprise Linux 4 및 Red Hat Enterprise Linux 5 게스트에 다음과 유사한 오류가 표시될 수 있습니다.

```
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 에서 이 문제를 해결하려면 다음 단계를 수행합니다.

- *virt-manager 에서 영향을 받는 게스트를 엽니다.*
- *세부 정보+보기를 클릭합니다.*
- *목록에서 VNC 디스플레이를 선택합니다.*
- *Keymap 풀다운 메뉴에서 Auto 를 ja 로 변경합니다.*

- **Apply** 버튼을 클릭합니다.

또는 대상 게스트에서 **virsh edit** 명령을 사용하여 이 문제를 해결하려면 다음을 수행합니다.

- **virsh edit <target guest>** 실행

- 태그에 다음 속성을 추가합니다 **{c>. keymap='ja'.** 예를 들어 다음과 같습니다.

```
<graphics type='vnc' port='-1' autoport='yes' keymap='ja'/>
```

21.14. 가상화 확장 확인

이 섹션을 사용하여 시스템에 하드웨어 가상화 확장 기능이 있는지 확인합니다. 완전한 가상화를 위해서는 가상화 확장 기능**(Intel VT-x** 또는 **AMD-V)**이 필요합니다.

1. 다음 명령을 실행하여 **CPU** 가상화 확장을 사용할 수 있는지 확인합니다.

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

2. 출력을 분석합니다.

- 다음 출력에는 **Intel VT-x** 확장 기능이 있는 **Intel** 프로세서를 나타내는 **vmx** 항목이 포함되어 있습니다.

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

- 다음 출력에는 **AMD-V** 확장 기능이 있는 **AMD** 프로세서를 나타내는 **svm** 항목이 포함되어 있습니다.

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

출력을 수신하면 프로세서에 하드웨어 가상화 확장 기능이 있습니다. 그러나 경우에 따라 제조업체가 **BIOS**에서 가상화 확장을 사용하지 않도록 설정합니다.

"출력 콘텐츠는 시스템의 각 하이퍼스레드, 코어 또는 **CPU**에 대해 한 번 여러 번 나타날 수 있습니다.

BIOS에서 가상화 확장 기능을 비활성화할 수 있습니다. 확장 기능이 나타나지 않거나 전체 가상화가 작동하지 않는 경우 절차 21.3. “BIOS에서 가상화 확장 활성화” 에서 참조하십시오.

3. **KVM** 하위 시스템이 로드되었는지 확인

   추가 검사로 **kvm** 모듈이 커널에 로드되었는지 확인합니다.

```
# lsmod | grep kvm
```

출력에 **kvm_intel** 또는 **kvm_amd** 가 포함된 경우 **kvm** 하드웨어 가상화 모듈이 로드되고 시스템이 요구 사항을 충족합니다.

참고

libvirt 패키지가 설치된 경우 **virsh** 명령은 가상화 시스템 기능의 전체 목록을 출력할 수 있습니다. 전체 목록을 받으려면 **virsh** 기능을 **root**로 실행합니다.

부록 A. 가상 호스트 지표 데몬(VHOSTMD)

vhostmd (가상 호스트 지표 데몬)를 사용하면 가상 머신에서 실행 중인 호스트에 대한 제한된 정보를 볼 수 있습니다. 이 데몬은 **SAP**용 **Red Hat Enterprise Linux**에서만 제공됩니다.

호스트에서 데몬(**vhostmd**)은 메트릭을 디스크 이미지에 주기적으로 쓰는 데몬을 실행합니다. 이 디스크 이미지는 게스트 가상 머신에 읽기 전용으로 내보냅니다. 게스트 가상 머신은 디스크 이미지를 읽고 지표를 확인할 수 있습니다. 간단한 동기화는 게스트 가상 머신이 최신 또는 손상된 메트릭이 표시되지 않도록 합니다.

시스템 관리자는 게스트 가상 시스템별로 사용할 수 있는 메트릭을 선택합니다. 또한 시스템 관리자는 하나 이상의 게스트 가상 머신을 지표 구성에 액세스할 수 없도록 차단할 수 있습니다.

따라서 **vhostmd** 및 **vm-dump-metrics** 를 사용하려는 고객은 **"AWS for SAP** 비즈니스 애플리케이션" 서브스크립션을 사용하려면 고객 포털 또는 **Red Hat** 서브스크립션 관리에서 **SAP**를 실행하는 **RHEL** 시스템을 **" SAP** 용 **SAP"** 채널에 등록해야 합니다. 고객 포털의 다음 **kbase** 문서에서는 **RHEL**에서 **vhostmd**의 설정에 대해 설명합니다. **https://access.redhat.com/knowledge/solutions/41566**

부록 B. 추가 리소스

가상화 및 Red Hat Enterprise Linux에 대한 자세한 내용은 다음 리소스를 참조하십시오.

B.1. 온라인 리소스

- http://www.libvirt.org/ 은(는) libvirt 가상화 API의 공식 웹 사이트입니다.
- https://virt-manager.org/ 는 가상 머신을 관리하기 위한 그래픽 애플리케이션인 Virtual Machine Manager (virt-manager)의 프로젝트 웹 사이트입니다.
- Red Hat Virtualization - http://www.redhat.com/products/cloud-computing/virtualization/
- Red Hat 제품 설명서 - https://access.redhat.com/documentation/en/
- Virtualization technologies overview - http://virt.kernelnewbies.org

B.2. 설치된 문서

- man virsh 및 /usr/share/doc/libvirt- <version-number > - virsh 가상 시스템 관리 유틸리티에 대한 하위 명령 및 옵션뿐만 아니라 libvirt 가상화 라이브러리 API에 대한 포괄적인 정보를 포함합니다.
- /usr/share/doc/gnome-applet-vm- <version-number > - 로컬 실행 가상 머신을 모니터링하고 관리하는 GNOME 그래픽 패널 애플릿에 대한 설명서입니다.
- /usr/share/doc/libvirt-python- <version-number > - libvirt 라이브러리의 Python 바인딩에 대한 세부 정보를 제공합니다. libvirt-python 패키지를 사용하면 python 개발자가 libvirt 가상화 관리 라이브러리와 인터페이스하는 프로그램을 만들 수 있습니다.
- /usr/share/doc/python-virtinst- <version-number > - 가상 머신 내부에서 Fedora 및 Red Hat Enterprise Linux 관련 배포판을 설치하는 데 도움이 되는 virt-install 명령에 대한 설명서를 제공합니다.

- */usr/share/doc/virt-manager- <version-number > - 가상 머신을 관리하기 위한 그래픽 도구를 제공하는 Virtual Machine Manager에 대한 설명서를 제공합니다.*

부록 C. 개정 내역

고침 1-502 6.9 GA 릴리스 업데이트	**Mon Mar 08 2017**	**Jiri Herrmann**
고침 1-501 6.8 GA 릴리스 업데이트	**Mon May 02 2016**	**Jiri Herrmann**
고침 1-500 6.8 베타 릴리스에 대한 여러 업데이트	**Thu Mar 01 2016**	**Jiri Herrmann**
고침 1-449 Revision History 정리	**Thu Oct 08 2015**	**Jiri Herrmann**
고침 1-447 6.7 GA 릴리스에 대한 업데이트입니다.	**Fri Jul 10 2015**	**Dayle Parker**