Red Hat Enterprise Linux 9

가상화 구성 및 관리

호스트 설정, 가상 머신 생성 및 관리, Red Hat Enterprise Linux 9의 가상화 기능 이해

Last Updated: 2022-08-29

Red Hat Enterprise Linux 9 가상화 구성 및 관리

호스트 설정, 가상 머신 생성 및 관리, Red Hat Enterprise Linux 9의 가상화 기능 이해

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

이 문서에서는 RHEL 9(Red Hat Enterprise Linux 9)에서 가상화를 관리하는 방법을 설명합니다. 가상화에 대한 일반적인 정보 외에도 명령줄 유틸리티를 사용하여 가상화를 관리하는 방법과 웹 콘솔을 사용하는 방법을 설명합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 를 참조하십시오.

RED HAT 문서에 관한 피드백 제공

문서 개선을 위한 의견을 보내 주십시오. 문서를 개선할 수 있는 방법에 관해 알려주십시오.

- 특정 문구에 대한 간단한 의견 작성 방법은 다음과 같습니다.
  1. 문서가 *Multi-page HTML* 형식으로 표시되는지 확인합니다. 또한 문서 오른쪽 상단에 **피드백** 버튼이 있는지 확인합니다.
  2. 마우스 커서를 사용하여 주석 처리하려는 텍스트 부분을 강조 표시합니다.
  3. 강조 표시된 텍스트 아래에 표시되는 **피드백 추가** 팝업을 클릭합니다.
  4. 표시된 지침을 따릅니다.
- Bugzilla를 통해 피드백을 제출하려면 새 티켓을 생성하십시오.
  1. Bugzilla 웹 사이트로 이동하십시오.
  2. 구성 요소로 **문서**를 사용합니다.
  3. **설명** 필드에 문서 개선을 위한 제안 사항을 기입하십시오. 관련된 문서의 해당 부분 링크를 알려주십시오.
  4. **버그 제출**을 클릭합니다.

1장. RHEL의 가상화 소개

가상화 또는 Linux 구현의 개념에 익숙하지 않은 경우 다음 섹션에서는 RHEL 9의 가상화에 대한 일반적인 개요를 제공합니다. 다음 섹션에서는 Red Hat에서 제공하는 기본 사항, 이점, 구성 요소 및 기타 가능한 가상화 솔루션 등 RHEL 9에서의 가상화에 대한 일반적인 개요를 제공합니다.

1.1. 가상화란 무엇인가?

RHEL 9는 RHEL 9를 실행하는 시스템에서 *게스트* 라고도 하는 여러 가상 머신(VM)을 *호스팅할* 수 있는 *가상화* 기능을 제공합니다. VM은 호스트의 물리적 하드웨어 및 컴퓨팅 리소스를 사용하여 호스트의 운영 체제에서 별도의 가상화 운영 체제(*게스트* OS)를 사용자 공간 프로세스로 실행합니다.

즉, 가상화는 운영 체제 내에 운영 체제를 사용할 수 있도록 합니다.

VM을 사용하면 소프트웨어 구성 및 기능을 안전하게 테스트하고, 기존 소프트웨어를 실행하거나, 하드웨어의 워크로드 효율성을 최적화할 수 있습니다. 혜택에 대한 자세한 내용은 가상화 Advantages를 참조하십시오.

가상화에 대한 자세한 내용은 Red Hat 고객 포털을 참조하십시오.

다음 단계

- Red Hat Enterprise Linux 9에서 가상화를 사용하기 시작하려면 Red Hat Enterprise Linux 9에서 가상화 활성화를 참조하십시오.
- Red Hat은 Red Hat Enterprise Linux 9 가상화 외에도 여러 가지 특수 가상화 솔루션을 제공합니다. 각 솔루션은 서로 다른 사용자 중심 및 기능을 제공합니다. 자세한 내용은 Red Hat 가상화 솔루션을 참조하십시오.

1.2. 가상화의 이점

VM(가상 머신)을 사용하면 물리적 머신 사용과 비교하여 다음과 같은 이점이 있습니다.

- **유연하고 세분화된 리소스 할당**
  VM은 일반적으로 물리적 호스트 시스템에서 실행되며 게스트 OS가 사용할 물리적 하드웨어도 할당할 수 있습니다. 그러나 VM에 물리적 리소스를 할당하는 것은 소프트웨어 수준에서 수행되므로 매우 유연합니다. VM은 호스트 메모리, CPU 또는 스토리지 공간의 구성 가능한 일부 부분을 사용하며, 해당 구성은 매우 세분화된 리소스 요청을 지정할 수 있습니다.

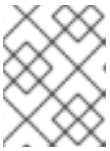
  예를 들어 게스트 OS가 호스트 파일 시스템에서 파일로 나타낼 수 있는 게스트 OS는 무엇이고, 해당 디스크의 크기는 물리 디스크에 사용 가능한 크기보다 적게 제한됩니다.

- **소프트웨어 제어 구성**
  VM의 전체 구성은 호스트에 데이터로 저장되고 소프트웨어 제어에 있습니다. 따라서 VM을 쉽게 생성, 제거, 복제, 마이그레이션, 운영하거나 원격 스토리지에 연결할 수 있습니다.

- **호스트에서 분리**
  게스트 OS는 호스트 OS와는 별도로 가상화된 커널에서 실행됩니다. 즉, 게스트 OS가 불안정하거나 손상된 경우에도 VM에 모든 OS를 설치할 수 있습니다.

- **공간 및 비용 효율성**
  단일 물리적 시스템은 다수의 VM을 호스팅할 수 있습니다. 따라서 여러 물리적 시스템이 동일한 작업을 수행할 필요가 없어 물리적 하드웨어와 관련된 공간, 전력 및 유지 관리 요구 사항이 줄어듭니다.

- **소프트웨어 호환성**
  VM은 호스트와 다른 OS를 사용할 수 있으므로 가상화에서는 호스트 OS용으로 처음 릴리스되지 않은 애플리케이션을 실행할 수 있습니다. 예를 들어 RHEL 7 게스트 OS를 사용하면 RHEL 9 호스트 시스템에서 RHEL 7용으로 릴리스된 애플리케이션을 실행할 수 있습니다.



참고

일부 운영 체제가 RHEL 9 호스트에서 게스트 OS로 지원되는 것은 아닙니다. 자세한 내용은 RHEL 9 가상화의 권장 기능을 참조하십시오.

1.3. 가상 머신 구성 요소 및 상호 작용

RHEL 9의 가상화는 다음과 같은 주요 소프트웨어 구성 요소로 구성됩니다.

하이퍼바이저

RHEL 9에서 VM(가상 머신)을 생성하는 것은 *하이퍼바이저* 이며, 하드웨어를 제어하고 호스트 머신에서 여러 운영 체제를 실행할 수 있도록 하는 소프트웨어 계층입니다.

하이퍼바이저에는 **KVM(커널 기반 가상 시스템)** 모듈 및 가상화 커널 드라이버가 포함되어 있습니다. 이러한 구성 요소를 통해 호스트 시스템의 Linux 커널이 사용자 공간 소프트웨어에 가상화를 위한 리소스를 제공할 수 있습니다.

사용자 공간 수준에서 **QEMU** 에뮬레이터는 게스트 운영 체제를 실행할 수 있는 전체 가상화된 하드웨어 플랫폼을 시뮬레이션하고 호스트에 리소스를 할당하고 게스트에 제공되는 방법을 관리합니다.

또한 **libvirt** 소프트웨어 제품군은 관리 및 통신 계층 역할을 하여 QEMU와의 상호 작용, 보안 규칙 적용 및 VM 구성 및 실행을 위한 다양한 추가 툴을 제공합니다.

XML 구성

호스트 기반 XML 구성 파일(*도메인 XML* 파일이라고도 함)은 특정 VM의 모든 설정 및 장치를 결정합니다. 구성에는 다음이 포함됩니다.

- VM의 이름, 시간대 및 VM에 대한 기타 정보와 같은 메타데이터입니다.
- 가상 CPU(vCPUS), 스토리지 장치, 입력/출력 장치, 네트워크 인터페이스 카드 및 기타 하드웨어, 실제 및 가상을 포함한 VM의 장치에 대한 설명입니다.
- VM 설정에서 사용할 수 있는 최대 메모리 양, 다시 시작 설정 및 VM 동작에 대한 기타 설정입니다.

XML 구성의 콘텐츠에 대한 자세한 내용은 샘플 가상 머신 XML 구성을 참조하십시오.

구성 요소 상호 작용

VM이 시작되면 하이퍼바이저는 XML 구성을 사용하여 호스트에서 사용자 공간 프로세스로 VM 인스턴스를 생성합니다. 또한 하이퍼바이저를 사용하면 **virsh**,**virt-install**, RuntimeClass 유틸리티 또는 웹 콘솔 GUI와 같은 호스트 기반 인터페이스에서 VM 프로세스에 액세스할 수 있습니다.

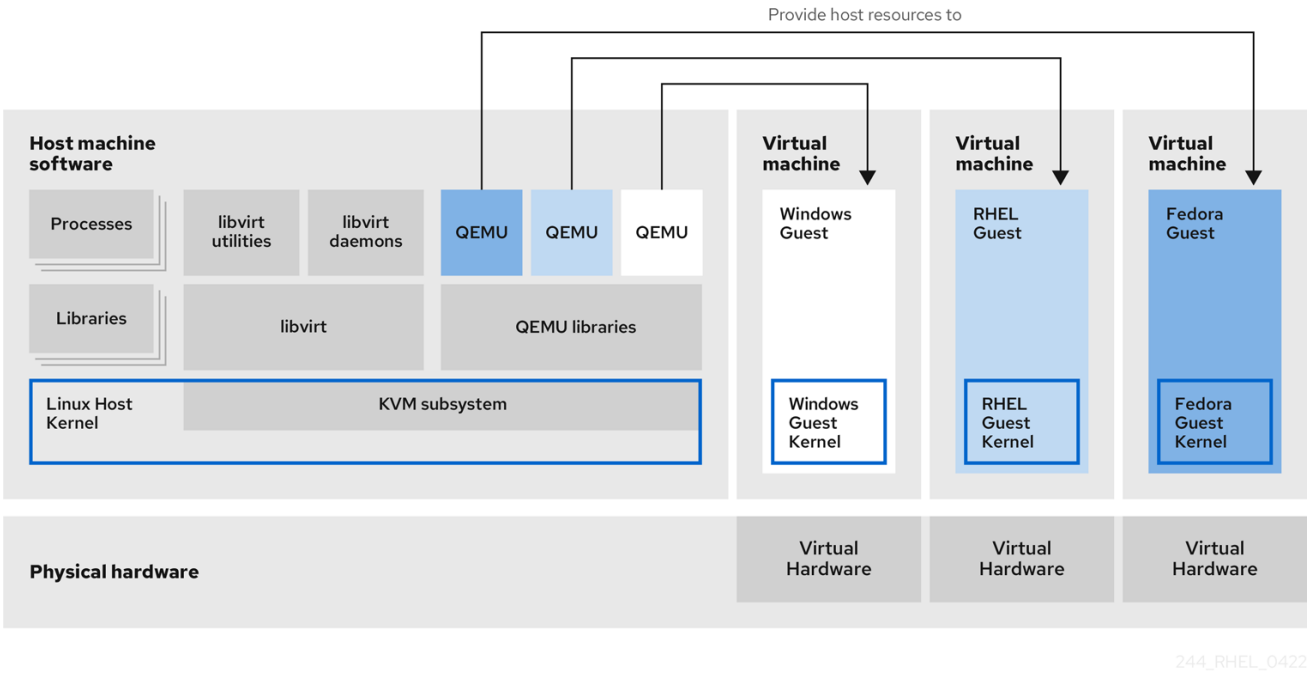
이러한 가상화 도구를 사용하면 libvirt에서 입력을 QEMU에 대한 지침으로 변환합니다. QEMU는 KVM에 지침을 전달하므로 커널이 지침을 수행하는 데 필요한 리소스를 적절하게 할당할 수 있습니다. 결과적으로 QEMU는 VM 생성 또는 수정과 같은 해당 사용자 공간 변경을 실행하거나 VM의 게스트 운영 체제에서 작업을 수행할 수 있습니다.

참고

QEMU는 아키텍처의 필수 구성 요소이지만 보안 문제로 인해 **RHEL 9** 시스템에서 직접 사용할 수 없습니다. 따라서 **qemu-*** 명령을 사용하는 것은 **Red Hat**에서 지원되지 않으며 **libvirt**를 사용하여 **QEMU**와 상호 작용하는 것이 좋습니다.

호스트 기반 인터페이스에 대한 자세한 내용은 가상화 관리를 위한 도구 및 인터페이스를 참조하십시오.

그림 1.1. RHEL 9 가상화 아키텍처



1.4. 가상화 관리를 위한 툴 및 인터페이스

RHEL 9에서 **CLI**(명령줄 인터페이스) 또는 여러 개의 **GUI**(그래픽 사용자 인터페이스)를 사용하여 가상화를 관리할 수 있습니다.

명령줄 인터페이스

CLI는 **RHEL 9**에서 가상화를 관리하는 가장 강력한 방법입니다. **VM**(가상 머신) 관리를 위한 **CLI** 명령은 다음과 같습니다.

- **virsh** - 제공된 인수에 따라 다양한 용도로 제공되는 가상화 명령줄 유틸리티 및 셸입니다. 예를 들면 다음과 같습니다.

- VM 시작 및 종료 - virsh start and virsh shutdown
- 사용 가능한 VM 나열 - virsh 목록
- 구성 파일에서 VM 생성 - virsh create
- 가상화 쉘 입력 - virsh

자세한 내용은 virsh(1) 매뉴얼 페이지를 참조하십시오.

- virt-install - 새 VM 생성을 위한 CLI 유틸리티입니다. 자세한 내용은 virt-install(1) 매뉴얼 페이지를 참조하십시오.
- virt-xml - VM 구성을 편집하기 위한 유틸리티입니다.
- RuntimeClass - VM 디스크 이미지 검사 및 수정에 대한 유틸리티입니다. 자세한 내용은 RuntimeClass (1) 매뉴얼 페이지를 참조하십시오.

그래픽 인터페이스

다음 GUI를 사용하여 RHEL 9에서 가상화를 관리할 수 있습니다.

- *Cockpit* 라고도 하는 RHEL 9 웹 콘솔 은 VM 및 가상화 호스트를 관리하기 위해 그래픽 사용자 인터페이스를 사용하기 쉽고 원격으로 액세스 할 수 있습니다.

  웹 콘솔을 사용한 기본 가상화 관리에 대한 자세한 내용은 웹 콘솔 의 가상 머신 관리를 참조하십시오.

1.5. RED HAT 가상화 솔루션

다음 Red Hat 제품은 RHEL 9 가상화 기능을 기반으로 하며 RHEL 9에서 사용할 수 있는 KVM 가상화 기능을 확장합니다. 또한 RHEL 9 가상화의 많은 제한 사항은 이러한 제품에는 적용되지 않습니다.

OpenShift Virtualization

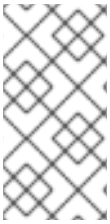
KubeVirt 기술을 기반으로 OpenShift Virtualization은 Red Hat OpenShift Container Platform의 일부이며 컨테이너에서 가상 머신을 실행할 수 있습니다.

OpenShift Virtualization에 대한 자세한 내용은 Red Hat Hybrid Cloud 페이지를 참조하십시오.

Red Hat OpenStack Platform (RHOSP)

Red Hat OpenStack Platform은 안전하고 신뢰할 수 있는 퍼블릭 또는 프라이빗 OpenStack 클라우드를 구축, 배포, 확장할 수 있는 통합 기반을 제공합니다.

Red Hat OpenStack Platform에 대한 자세한 내용은 Red Hat 고객 포털 또는 Red Hat OpenStack Platform 설명서 제품군 을 참조하십시오.



참고

RHEL에서 지원되지만 다른 Red Hat 가상화 솔루션에서 지원되지 않는 가상화 기능에 대한 자세한 내용은 RHEL 9 가상화에서 지원되지 않는 기능을 참조하십시오.

2장. 가상화 활성화

RHEL 9에서 가상화를 사용하려면 가상화 패키지를 설치하고 VM(가상 머신)을 호스팅하도록 시스템이 구성되어 있는지 확인해야 합니다. 이를 수행하는 특정 단계는 CPU 아키텍처에 따라 다릅니다.

2.1. AMD64 및 INTEL 64에서 가상화 활성화

RHEL 9를 실행하는 AMD64 또는 Intel 64 시스템에 KVM 하이퍼바이저를 설정하고 VM(가상 머신)을 생성하려면 다음 지침을 따르십시오.

사전 요구 사항

- Red Hat Enterprise Linux 9가 호스트 시스템에 설치 및 등록되어 있습니다.
- 시스템은 가상화 호스트로 작동하기 위해 다음 하드웨어 요구 사항을 충족합니다.
  - 호스트 시스템의 아키텍처는 KVM 가상화를 지원합니다.
  - 다음과 같은 최소 시스템 리소스를 사용할 수 있습니다.
    - 6GB의 호스트 여유 디스크 공간과 각 VM에 대해 6GB의 여유 디스크 공간
    - 호스트용 2GB RAM과 각 VM에 대해 2GB의 RAM.

절차

1. 가상화 하이퍼바이저 패키지를 설치합니다.

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

2. 가상화 서비스를 시작합니다.

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

검증

1. 시스템이 가상화 호스트로 준비되었는지 확인합니다.

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

2. 모든 **virt-host-validate** 검사에서 **STATUSS** 값을 반환하는 경우 시스템이 VM을 생성할 준비가 되어 있습니다.

검사 중 **FAIL** 값을 반환하는 경우 표시된 지침에 따라 문제를 해결합니다.

검사 중 **WARN** 값을 반환하는 경우 표시된 지침에 따라 가상화 기능을 개선하는 것이 좋습니다.

문제 해결

- 호스트 **CPU**에서 **KVM** 가상화를 지원하지 않는 경우 **virt-host-validate** 는 다음 출력을 생성합니다.

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

그러나 이러한 호스트 시스템의 **VM**은 성능 문제가 아닌 부팅에 실패합니다.

이 문제를 해결하려면 **VM**의 **XML** 구성에서 **& lt;domain type >** 값을 **qemu** 로 변경할 수 있습니다. 그러나 **Red Hat**은 **qemu** 도메인 유형을 사용하는 **VM**을 지원하지 않으며 프로덕션 환경에서는 권장되지 않습니다.

다음 단계

- RHEL 9 호스트에서 가상 머신 생성

2.2. IBM Z에서 가상화 활성화

RHEL 9를 실행하는 IBM Z 시스템에서 KVM 하이퍼바이저를 설정하고 VM(가상 머신)을 생성하려면 아래 지침을 따르십시오.

사전 요구 사항

- 다음과 같은 최소 시스템 리소스를 사용할 수 있습니다.
  - 6GB의 호스트 여유 디스크 공간과 각 VM에 대해 6GB의 여유 디스크 공간
  - 호스트용 2GB RAM과 각 VM에 대해 2GB의 RAM.
  - 호스트의 CPU 4개 VM은 일반적으로 단일 할당된 vCPU로 실행할 수 있지만, 부하가 많은 VM이 응답하지 않는 것을 방지하기 위해 VM당 2개 이상의 vCPU를 할당하는 것이 좋습니다.
- IBM Z 호스트 시스템은 z13 CPU를 사용하고 있습니다.
- RHEL 9는 LPAR(Logical partition)에 설치됩니다. 또한 LPAR는 SIE(*Start-interpretive execution*) 가상화 기능을 지원합니다.

  이를 확인하려면 /proc/cpuinfo 파일에서 sie 를 검색합니다.

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

절차

1. 가상화 패키지를 설치합니다.

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

2. 가상화 서비스를 시작합니다.

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

검증

1. 시스템이 가상화 호스트로 준비되었는지 확인합니다.

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

2. 모든 **virt-host-validate** 검사에서 **STATUSS** 값을 반환하는 경우 시스템이 VM을 생성할 준비가 되어 있습니다.

   검사 중 **FAIL** 값을 반환하는 경우 표시된 지침에 따라 문제를 해결합니다.

   검사 중 **WARN** 값을 반환하는 경우 표시된 지침에 따라 가상화 기능을 개선하는 것이 좋습니다.

문제 해결

- 호스트 **CPU**에서 **KVM** 가상화를 지원하지 않는 경우 **virt-host-validate** 는 다음 출력을 생성합니다.

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

  그러나 이러한 호스트 시스템의 **VM**은 성능 문제가 아닌 부팅에 실패합니다.

  이 문제를 해결하려면 **VM**의 **XML** 구성에서 **& lt;domain type >** 값을 **qemu** 로 변경할 수 있습니다. 그러나 **Red Hat**은 **qemu** 도메인 유형을 사용하는 **VM**을 지원하지 않으며 프로덕션 환경에서는 권장되지 않습니다.

2.3. ARM 64에서 가상화 활성화

RHEL 9를 실행하는 ARM 64 시스템에서 가상 머신(VM)을 생성하기 위한 KVM 하이퍼바이저를 설정하려면 아래 지침을 따르십시오.

중요

ARM 64의 가상화는 RHEL 9에서 기술 프리뷰 로만 제공되므로 지원되지 않습니다.

사전 요구 사항

- 다음과 같은 최소 시스템 리소스를 사용할 수 있습니다.
  - 6GB의 호스트를 위한 디스크 여유 공간, 의도한 게스트 각각에 대해 6GB의 6GB.
  - 호스트용 4GB RAM과 의도한 게스트 각각에 대해 4GB의 RAM.

절차

1. 가상화 패키지를 설치합니다.

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

2. 가상화 서비스를 시작합니다.

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

검증

1. 시스템이 가상화 호스트로 준비되었는지 확인합니다.

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

```
QEMU: Checking for cgroup 'blkio' controller mount-point    : PASS
QEMU: Checking if IOMMU is enabled by kernel                : WARN (Unknown if this
platform has IOMMU support)
```

2. 모든 **virt-host-validate** 검사에서 **passwordS** 값을 반환하는 경우 시스템에서 가상 머신을 생성할 준비가 된 것입니다.

   검사 중 **FAIL** 값을 반환하는 경우 표시된 지침에 따라 문제를 해결합니다.

   검사 중 **WARN** 값을 반환하는 경우 표시된 지침에 따라 가상화 기능을 개선하는 것이 좋습니다.

문제 해결

- 호스트 **CPU**에서 **KVM** 가상화를 지원하지 않는 경우 **virt-host-validate** 는 다음 출력을 생성합니다.

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

그러나 이러한 호스트 시스템의 **VM**은 성능 문제가 아닌 부팅에 실패합니다.

이 문제를 해결하려면 **VM**의 **XML** 구성에서 **& lt;domain type >** 값을 **qemu** 로 변경할 수 있습니다. 그러나 **Red Hat**은 **qemu** 도메인 유형을 사용하는 **VM**을 지원하지 않으며 프로덕션 환경에서는 권장되지 않습니다.

다음 단계

- 가상 머신 생성

추가 리소스

- **ARM 64**의 가상화가 **AMD64** 및 **Intel 64**와 다른 방법

3장. 가상 머신 생성

RHEL 9에서 VM(가상 머신)을 생성하려면 명령줄 인터페이스 또는 RHEL 9 웹 콘솔 을 사용합니다.

사전 요구 사항

- 시스템에 가상화가 설치 및 활성화되어 있습니다.
- 디스크 공간, RAM 또는 CPU와 같이 VM에 할당할 수 있는 충분한 시스템 리소스가 있습니다. 권장 값은 VM의 의도된 작업 및 워크로드에 따라 크게 다를 수 있습니다.

주의

RHEL 9에서는 호스트 CD-ROM 또는 DVD-ROM 장치에서 설치할 수 없습니다. RHEL 9에서 사용 가능한 VM 설치 방법을 사용할 때 CD-ROM 또는 DVD-ROM을 설치 소스로 선택하면 설치에 실패합니다. 자세한 내용은 Red Hat 지식베이스 를 참조하십시오.

3.1. 명령줄 인터페이스를 사용하여 가상 머신 생성

virt-install 유틸리티를 사용하여 RHEL 9 호스트에서 VM(가상 머신)을 만들려면 아래 지침을 따르십시오.

사전 요구 사항

- 호스트 시스템에서 가상화가 활성화되어 있습니다.
- 디스크 공간, RAM 또는 CPU와 같이 VM에 할당할 수 있는 충분한 시스템 리소스가 있습니다. 권장 값은 VM의 의도된 작업 및 워크로드에 따라 크게 다를 수 있습니다.
- 운영 체제(OS) 설치 소스는 로컬 또는 네트워크에서 사용할 수 있습니다. 다음 중 하나일 수 있습니다.

- 설치 미디어의 ISO 이미지
- 기존 VM 설치의 디스크 이미지

주의

RHEL 9에서는 호스트 CD-ROM 또는 DVD-ROM 장치에서 설치할 수 없습니다. RHEL 9에서 사용할 수 있는 VM 설치 방법을 사용할 때 CD-ROM 또는 DVD-ROM을 설치 소스로 선택하면 설치에 실패합니다. 자세한 내용은 Red Hat 지식베이스 를 참조하십시오.

- 선택 사항: Kickstart 파일을 더 빠르고 쉽게 설치할 수 있도록 제공할 수 있습니다.

절차

VM을 생성하고 OS 설치를 시작하려면 다음 필수 인수와 함께 virt-install 명령을 사용하십시오.

- 새 머신의 이름(--name)
- 할당된 메모리 양(--memory)
- 할당된 가상 CPU 수(--vcpus)
- 할당된 스토리지의 유형 및 크기(--disk)
- OS 설치 소스의 유형 및 위치 (--cdrom 또는 --location)

선택한 설치 방법에 따라 필요한 옵션과 값이 다를 수 있습니다. 예를 보려면 아래를 참조하십시오.

-

다음은 /home/username/Downloads/Win10install.iso 파일에 로컬로 저장된 ISO 이미지에서 Windows 10 OS를 설치하는 demo-guest1 이라는 VM을 생성합니다. 또한 이 VM은 2048MiB RAM 및 2개의 vCPU로 할당되며 VM에 대해 80GiB qcow2 가상 디스크가 자동으로 구성됩니다.

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

- 다음은 /home/username/Downloads/rhel9.iso 이미지를 사용하여 라이브 CD에서 RHEL 9 OS를 실행하는 demo-guest2 라는 VM을 생성합니다. 이 VM에 디스크 공간이 할당되지 않으므로 세션 중에 변경한 내용은 유지되지 않습니다. 또한 VM에 4096MiB RAM 및 4개의 vCPU가 할당됩니다.

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

- 다음은 기존 디스크 이미지 /home/username/backup/disk.qcow2 에 연결하는 demo-guest3 이라는 RHEL 9 VM을 생성합니다. 이는 시스템 간에 하드 드라이브를 물리적으로 이동하는 것과 유사하므로 demo-guest3에서 사용할 수 있는 OS와 데이터는 이전에 이미지를 처리하는 방법에 따라 결정됩니다. 또한 이 VM에는 2048MiB RAM 및 2개의 vCPU가 할당됩니다.

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

디스크 이미지를 가져올 때 --os-variant 옵션을 사용하는 것이 좋습니다. 제공되지 않으면 생성된 VM의 성능에 부정적인 영향을 미칩니다.

- 다음은 http://example.com/OS-install URL에서 설치하는 demo-guest4 라는 VM을 생성합니다. 설치가 성공적으로 시작되면 URL에 작동 중인 OS 설치 트리가 포함되어 있어야 합니다. 또한 OS는 /home/username/ks.cfg kickstart 파일을 사용하여 자동으로 구성됩니다. 또한 이 VM은 2048MiB RAM, 2개의 vCPU 및160GiB qcow2 가상 디스크로 할당됩니다.

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

- 다음은 그래픽 없이 RHEL9.iso 이미지 파일에서 설치하는 demo-guest5 라는 VM을 텍스트 전용 모드로 생성합니다. 게스트 콘솔을 직렬 콘솔에 연결합니다. VM에는 16384MiB의 메모리, 16개의 vCPU 및 280GiB 디스크가 있습니다. 이러한 종류의 설치는 느린 네트워크 링크를 통해 호스트에 연결할 때 유용합니다.

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

- 다음은 **demo-guest5**와 동일한 구성이 있지만 **10.0.0.1** 원격 호스트에 상주하는 **demo-guest6** 이라는 **VM**을 생성합니다.

```
# virt-install --connect qemu+ssh://root@10.0.0.1/system --name demo-guest6 --memory 16384 --vcpus 16 --disk size=280 --os-variant rhel9.0 --location RHEL9.iso --graphics none --extra-args='console=ttyS0'
```

- 다음은 **demo-guest5**와 동일한 구성이 있는 **demo-guest-7** 이라는 **VM**을 생성하지만 해당 스토리지의 경우 **DASD** 중재 장치 **mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8**를 사용하여 장치 **1111** 을 할당합니다.

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

virsh nodedev-list --cap mdev 명령을 사용하여 설치할 수 있는 중재 장치의 이름을 검색할 수 있습니다.

검증

- **VM**이 성공적으로 생성되면 **VM**의 그래픽 콘솔로 **virt-viewer** 창이 열리고 게스트 **OS** 설치를 시작합니다.

문제 해결

- **virt-install** 에 오류가 발생하면 기본 네트워크 오류를 찾을 수 없습니다.

a. ***libvirt-daemon-config-network*** 패키지가 설치되었는지 확인합니다.

```
# dnf info libvirt-daemon-config-network
Installed Packages
Name         : libvirt-daemon-config-network
[...]
```

b. **libvirt** 기본 네트워크가 활성 상태이고 자동으로 시작하도록 구성되었는지 확인합니다.

```
# virsh net-list --all
 Name      State    Autostart   Persistent
----------------------------------------
```

```
default   active   yes         yes
```

c. 이 네트워크가 없는 경우 기본 네트워크를 활성화하고 **auto-start**로 설정합니다.

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

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

i. 다음 오류와 함께 기본 네트워크 활성화에 실패하면 ***libvirt-daemon-config-network*** 패키지가 올바르게 설치되지 않았습니다.

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

이 문제를 해결하려면 ***libvirt-daemon-config-network*** 를 다시 설치합니다.

```
# dnf reinstall libvirt-daemon-config-network
```

ii. 다음과 유사한 오류와 함께 기본 네트워크 활성화가 실패하면 기본 네트워크의 서브넷과 호스트의 기존 인터페이스 간에 충돌이 발생했습니다.

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

이 문제를 해결하려면 **virsh net-edit** 기본 명령을 사용하고 구성의 **192.168.122.*** 값을 호스트에서 아직 사용되지 않는 서브넷으로 변경합니다.

추가 리소스

- **man virt-install** 명령

3.2. 웹 콘솔을 사용하여 가상 머신 생성 및 게스트 운영 체제 설치

RHEL 9 호스트의 **GUI**에서 **VM**(가상 시스템)을 관리하려면 웹 콘솔을 사용합니다. 다음 섹션에서는 **RHEL 9** 웹 콘솔을 사용하여 **VM**을 생성하고 게스트 운영 체제를 설치하는 방법에 대한 정보를 제공합니다.

3.2.1. 웹 콘솔을 사용하여 가상 머신 생성

웹 콘솔이 연결된 호스트 머신에 VM(가상 머신)을 생성하려면 아래 지침을 따르십시오.

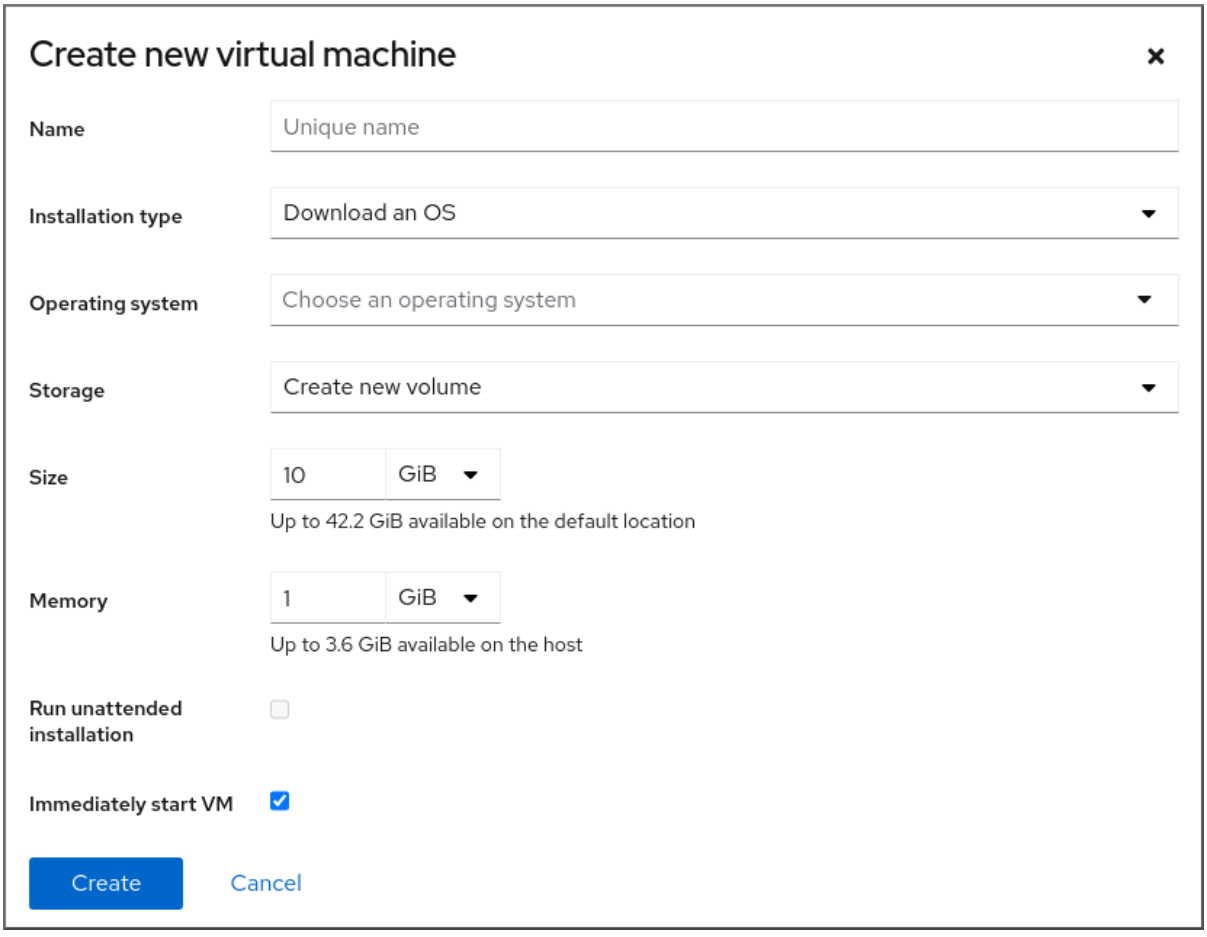
사전 요구 사항

- 호스트 시스템에서 가상화가 활성화되어 있습니다.
- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 디스크 공간, RAM 또는 CPU와 같이 VM에 할당할 수 있는 충분한 시스템 리소스가 있습니다. 권장 값은 VM의 의도된 작업 및 워크로드에 따라 크게 다를 수 있습니다.

절차

1. 웹 콘솔 의 가상 머신 인터페이스에서 VM 생성 을 클릭합니다.

   Create new virtual machine(새 가상 머신 만들기) 대화 상자가 표시됩니다.



2. 생성할 VM의 기본 구성을 입력합니다.

- name - VM의 이름입니다.
- 연결 - libvirt 연결 유형, 시스템 또는 세션입니다. 자세한 내용은 시스템 및 세션 연결을 참조하십시오.
- 설치 유형 - 설치 시 로컬 설치 매체, URL, PXE 네트워크 부팅, 클라우드 기본 이미지 또는 제한된 운영 체제 세트에서 OS를 다운로드할 수 있습니다.
- 운영 체제 - VM의 운영 체제. Red Hat은 제한된 게스트 운영 체제 세트에 대해서만 지원을 제공합니다.
- 스토리지 - VM을 구성할 스토리지 유형입니다.
- size - VM을 구성할 스토리지 공간의 크기입니다.
- memory - VM을 구성할 메모리 양입니다.
- 자동 설치 실행 - 설치를 자동으로 실행할지 여부입니다. 이 옵션은 설치 유형이 OS 다운로드인 경우에만 사용할 수 있습니다.
- VM을 즉시 시작 - VM이 생성된 직후 시작될지 여부입니다.

3. 생성을 클릭합니다.

VM이 생성됩니다. Immediately Start VM 확인란을 선택하면 VM이 즉시 시작되고 게스트 운영 체제 설치가 시작됩니다.

추가 리소스

- VM에 운영 체제 설치

3.2.2. 웹 콘솔을 사용하여 디스크 이미지를 가져와 가상 머신 생성

기존 VM 설치의 디스크 이미지를 가져와 VM(가상 머신)을 생성하려면 아래 지침을 따르십시오.

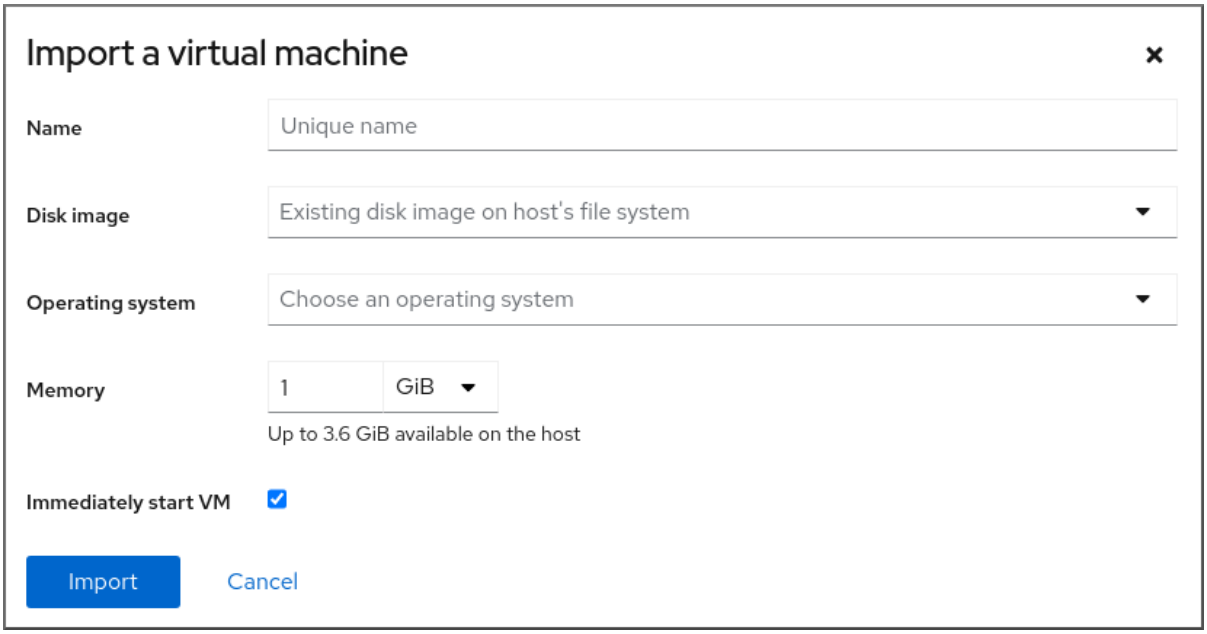
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 디스크 공간, RAM 또는 CPU와 같이 VM에 할당할 수 있는 충분한 시스템 리소스가 있습니다. 권장 값은 VM의 의도된 작업 및 워크로드에 따라 크게 다를 수 있습니다.
- 기존 VM 설치의 디스크 이미지가 있는지 확인합니다.

절차

1. 웹 콘솔의 Virtual Machines(가상 시스템) 인터페이스에서 Import VM (VM 가져오기)을 클릭합니다.

   Import a virtual machine(가상 머신 가져오기) 대화 상자가 표시됩니다.



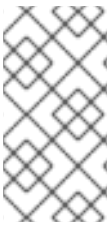
2. 생성할 VM의 기본 구성을 입력합니다.

- **name - VM**의 이름입니다.
- 연결 **- libvirt** 연결 유형, 시스템 또는 세션입니다. 자세한 내용은 시스템 및 세션 연결을 참조하십시오.
- 디스크 이미지 **-** 호스트 시스템에 있는 **VM**의 기존 디스크 이미지 경로입니다.
- 운영 체제 **- VM**의 운영 체제. **Red Hat**은 제한된 게스트 운영 체제 세트에 대해서만 지원을 제공합니다.
- **memory - VM**을 구성할 메모리 양입니다.
- **VM** 즉시 시작 **- VM**이 생성된 직후에 **VM**이 시작되는지 여부입니다.

3. 가져오기 를 클릭합니다.

3.2.3. 웹 콘솔을 사용하여 게스트 운영 체제 설치

VM(가상 머신**)**을 처음 로드할 때는 **VM**에 운영 체제를 설치해야 합니다.



참고

Create New Virtual Machine(새 가상 머신 생성**)** 대화 상자의 ***Immediately Start VM*** 확인란을 선택하면 **VM**이 생성될 때 운영 체제의 설치 루틴이 자동으로 시작됩니다.

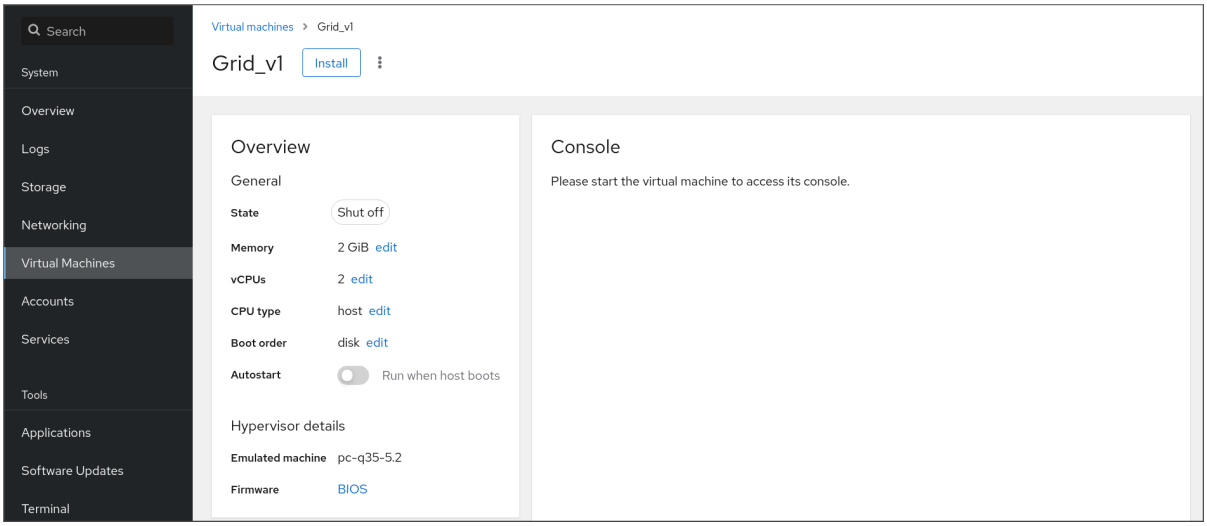
사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 시스템에 설치되어 있습니다.
- 운영 체제를 설치할 **VM**을 사용할 수 있어야 합니다.

절차

1. **Virtual Machines** (가상 시스템) 인터페이스에서 게스트 **OS**를 설치할 **VM**을 클릭합니다.

선택한 **VM**에 대한 기본 정보와 **VM**의 다양한 측면을 관리하기 위한 컨트롤이 포함된 새 페이지가 열립니다.



2. 선택 사항: 펌웨어를 변경합니다.

참고

Create New Virtual Machine (새 가상 머신 생성) 대화 상자에서 ***Immediately Start VM*** 확인란을 선택하지 않은 경우에만 펌웨어를 변경할 수 있으며 **OS**가 **VM**에 아직 설치되지 않은 경우에만 해당 펌웨어를 변경할 수 있습니다.

a. 펌웨어를 클릭합니다.

b. 펌웨어 변경 창에서 원하는 펌웨어를 선택합니다.

c. 저장을 클릭합니다.

3. 설치를 클릭합니다.

운영 체제의 설치 루틴은 **VM** 콘솔에서 실행됩니다.

문제 해결

- 설치 루틴이 실패하면 VM을 삭제하고 다시 생성해야 합니다.

3.2.4. 웹 콘솔을 사용하여 클라우드 이미지 인증으로 가상 머신 생성

기본적으로 클라우드 이미지에는 로그인 계정이 없습니다. 그러나 RHEL 웹 콘솔을 사용하여 이제 VM(가상 머신)을 생성하고 root 및 사용자 계정 로그인 인증 정보를 지정할 수 있습니다. 그러면 cloud-init로 전달됩니다.

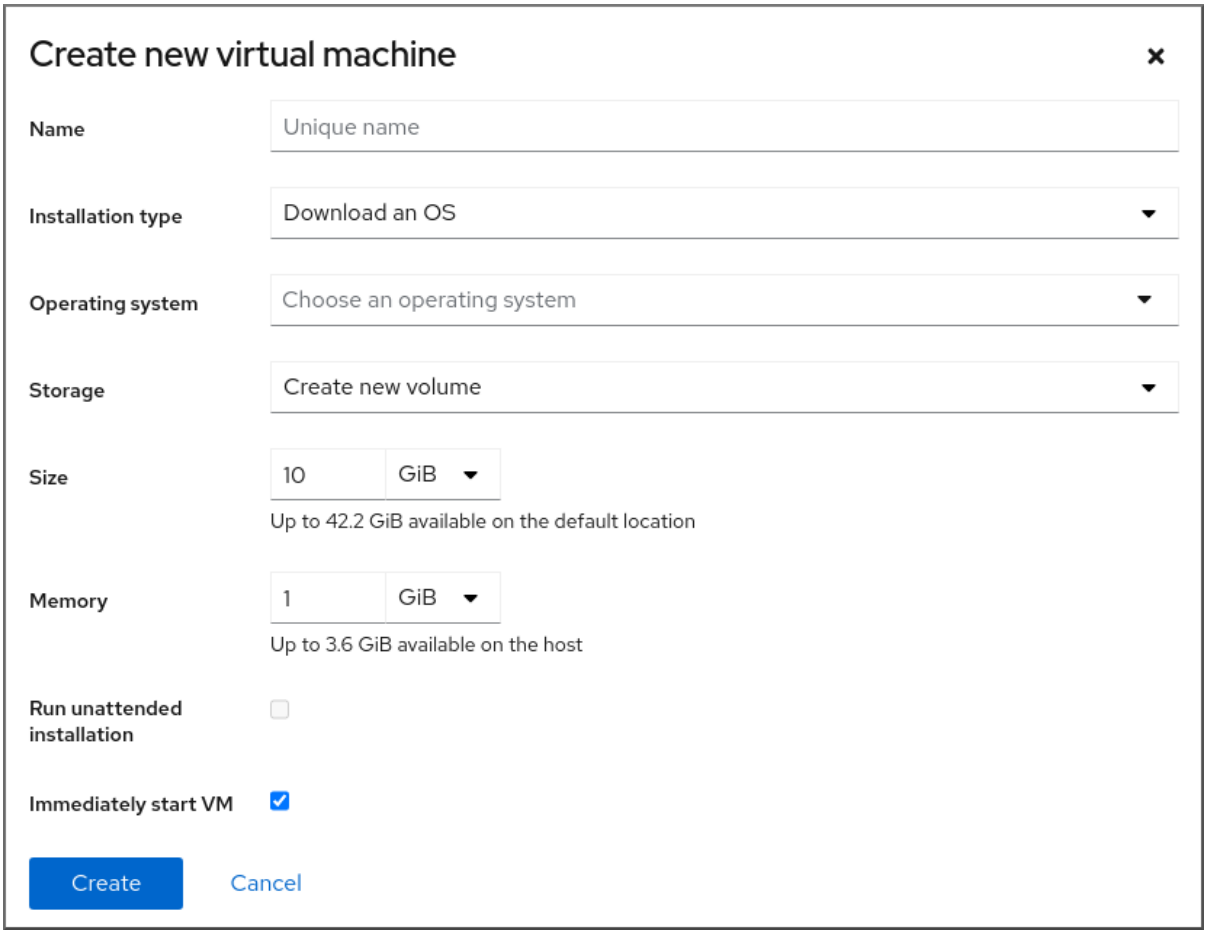
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 호스트 시스템에서 가상화가 활성화되어 있습니다.
- 디스크 공간, RAM 또는 CPU와 같이 VM에 할당할 충분한 시스템 리소스가 있습니다. 권장 값은 VM의 의도된 작업 및 워크로드에 따라 크게 다를 수 있습니다.

절차

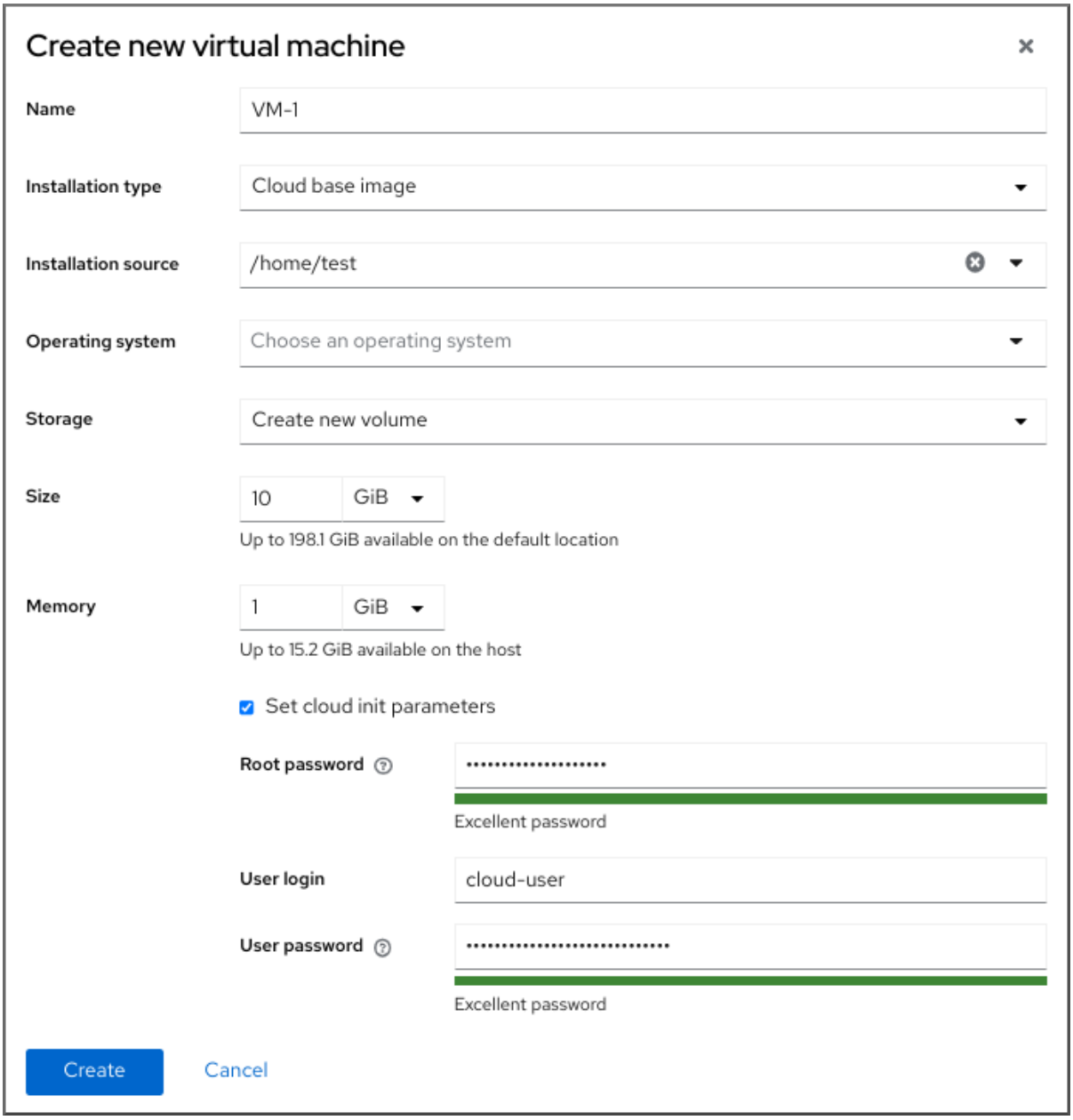
1. 웹 콘솔 의 가상 머신 인터페이스에서 VM 생성 을 클릭합니다.

   Create new virtual machine(새 가상 머신 만들기) 대화 상자가 표시됩니다.



2. **이름 필드에 VM의 이름을 입력합니다.**

3. **설치 유형 필드에서 Cloud base image 를 선택합니다.**



4. 설치 소스 필드에서 호스트 시스템의 이미지 파일 경로를 설정합니다.

5. 생성할 VM의 구성을 입력합니다.

- 운영 체제 - VM의 운영 체제. Red Hat은 제한된 게스트 운영 체제 세트에 대해서만 지원을 제공합니다.
- 스토리지 - VM을 구성할 스토리지 유형입니다.
- size - VM을 구성할 스토리지 공간의 크기입니다.

- **memory - VM**을 구성할 메모리 양입니다.

6. 클라우드 **init** 매개변수 설정을 선택합니다.

   클라우드 인증 자격 증명을 설정합니다.

   - 루트 암호 **- VM**의 루트 암호를 입력합니다. 루트 암호를 설정하지 않으려면 필드를 비워 둡니다.

   - 사용자 로그인 **- cloud-init** 사용자 로그인을 입력합니다.

   - 사용자 암호 **-** 암호를 입력합니다. 암호를 설정하지 않으려면 필드를 비워 둡니다.

7. 생성을 클릭합니다.

   VM이 생성됩니다.

추가 리소스

- VM에 운영 체제 설치

4장. 가상 머신 시작

RHEL 9에서 VM(가상 머신)을 시작하려면 명령줄 인터페이스 또는 웹 콘솔 GUI 를 사용할 수 있습니다.

사전 요구 사항

- VM을 시작하려면 먼저 VM을 생성해야 하며 OS를 사용하여 설치해야 합니다. 이를 수행하는 방법은 가상 머신 생성을 참조하십시오.

4.1. 명령줄 인터페이스를 사용하여 가상 머신 시작

CLI(명령줄 인터페이스)를 사용하여 종료한 VM(가상 머신)을 시작하거나 저장된 VM을 복원할 수 있습니다. CLI를 사용하면 로컬 및 원격 VM을 모두 시작할 수 있습니다.

사전 요구 사항

- 이미 정의된 비활성 VM입니다.
- VM의 이름입니다.
- 원격 VM의 경우:
  - VM이 있는 호스트의 IP 주소입니다.
  - 호스트에 대한 루트 액세스 권한입니다.

절차

- 로컬 VM의 경우 virsh start 유틸리티를 사용합니다.

  예를 들어 다음 명령은 *demo-guest1* VM을 시작합니다.

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

- 원격 호스트에 있는 VM의 경우 호스트의 QEMU+SSH 연결과 함께 virsh start 유틸리티를 사용합니다.

  예를 들어 다음 명령은 192.168.123.123 호스트에서 *demo-guest1* VM을 시작합니다.

```
# virsh -c qemu+ssh://root@192.168.123.123/system start demo-guest1

root@192.168.123.123's password:

Domain 'demo-guest1' started
```

추가 리소스

- virsh start --help 명령
- 원격 가상화 호스트에 쉽게 액세스 가능
- 호스트가 시작될 때 자동으로 가상 머신 시작

4.2. 웹 콘솔을 사용하여 가상 머신 시작

VM(가상 머신)이 *종료된* 상태인 경우 RHEL 9 웹 콘솔을 사용하여 시작할 수 있습니다. 호스트가 시작될 때 VM이 자동으로 시작하도록 구성할 수도 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 이미 정의된 비활성 VM입니다.
- VM의 이름입니다.

절차

1. Virtual Machines (가상 시스템) 인터페이스에서 시작할 VM을 클릭합니다.

선택한 VM에 대한 자세한 정보와 VM 종료 및 삭제를 위한 컨트롤이 포함된 새 페이지가 열립니다.

2. 실행 을 클릭합니다.

VM이 시작되고 해당 콘솔 또는 그래픽 출력에 연결할 수 있습니다.

3. 선택 사항: 호스트가 시작될 때 자동으로 시작하도록 VM을 구성하려면 Autostart 확인란을 클릭합니다.

libvirt에서 관리하지 않는 네트워크 인터페이스를 사용하는 경우 systemd 구성도 변경해야 합니다. 그렇지 않으면 영향을 받는 VM이 시작되지 못할 수 있습니다. 호스트가 시작될 때 자동으로 가상 시스템 시작을 참조하십시오.

추가 리소스

- 웹 콘솔에서 가상 머신 종료
- 웹 콘솔을 사용하여 가상 머신 재시작

4.3. 호스트가 시작될 때 자동으로 가상 시스템 시작

VM(가상 머신)이 실행 중인 호스트가 다시 시작되면 VM이 종료되며 기본적으로 다시 시작해야 합니다. 호스트가 실행될 때마다 VM이 활성화되도록 하려면 VM을 자동으로 시작하도록 구성할 수 있습니다.

사전 요구 사항

- 생성된 가상 머신

절차

1. virsh 자동 시작 유틸리티를 사용하여 호스트가 시작될 때 자동으로 시작하도록 VM을 구성합니다.

예를 들어 다음 명령은 자동으로 시작되도록 *demo-guest1* VM을 구성합니다.

■

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

2. libvirt 에서 관리하지 않는 네트워크 인터페이스를 사용하는 경우 systemd 구성을 추가로 변경해야 합니다. 그렇지 않으면 영향을 받는 VM이 시작되지 못할 수 있습니다.

참고

이러한 인터페이스는 예를 들면 다음과 같습니다.

- NetworkManager에서 만든 브리지 장치
- <forward mode='bridge'/>를 사용하도록 구성된 네트워크

a. systemd 구성 디렉터리 트리에서 virtqemud.service.d 디렉터리가 아직 없는 경우 생성합니다.

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

b. 이전에 생성한 디렉터리에 10-network-online.conf systemd 장치 재정의 파일을 만듭니다. 이 파일의 내용은 virtqemud 서비스의 기본 systemd 구성을 덮어씁니다.

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

c. 10-network-online.conf 파일에 다음 행을 추가합니다. 이 구성 변경으로 인해 호스트의 네트워크가 준비된 후에 systemd가 virtqemud 서비스를 시작할 수 있습니다.

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

검증

1. VM 구성을 보고 *autostart* 옵션이 활성화되었는지 확인합니다.

예를 들어 다음 명령은 *autostart* 옵션을 포함하여 *demo-guest1* VM에 대한 기본 정보를 표시합니다.

-

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

2. **libvirt**에서 관리하지 않는 네트워크 인터페이스를 사용하는 경우 **10-network-online.conf** 파일의 콘텐츠가 다음 출력과 일치하는지 확인합니다.

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

추가 리소스

- **virsh autostart --help** 명령
- 웹 콘솔을 사용하여 가상 머신 시작.

5장. 가상 머신 연결

RHEL 9에서 VM(가상 머신)과 상호 작용하려면 다음 중 하나를 수행하여 가상 머신에 연결해야 합니다.

- 웹 콘솔 인터페이스를 사용하는 경우 웹 콘솔 인터페이스에서 Virtual Machines(가상 머신) 창을 사용합니다. 자세한 내용은 웹 콘솔을 사용하여 가상 머신 상호 작용을 참조하십시오.
- 웹 콘솔을 사용하지 않고 VM 그래픽 디스플레이와 상호 작용해야 하는 경우 Virt Viewer 애플리케이션을 사용합니다. 자세한 내용은 Virt Viewer를 사용하여 가상 시스템 그래픽 콘솔 열기를 참조하십시오.
- 그래픽 디스플레이가 불가능하거나 필요하지 않은 경우 SSH 터미널 연결을 사용합니다.
- 네트워크를 사용하여 시스템에서 가상 시스템에 연결할 수 없는 경우 virsh 콘솔을 사용합니다.

연결 중인 VM이 로컬 호스트가 아닌 원격 호스트에 있는 경우 선택적으로 원격 호스트에 보다 편리하게 액세스할 수 있도록 시스템을 구성할 수 있습니다.

사전 요구 사항

- 상호 작용하려는 VM이 설치되고 시작됩니다.

5.1. 웹 콘솔을 사용하여 가상 머신과 상호 작용

RHEL 9 웹 콘솔에서 VM(가상 머신)과 상호 작용하려면 VM의 콘솔에 연결해야 합니다. 여기에는 그래픽 콘솔과 직렬 콘솔이 모두 포함됩니다.

- 웹 콘솔에서 VM의 그래픽 인터페이스와 상호 작용하려면 그래픽 콘솔을 사용합니다.
- 원격 뷰어에서 VM의 그래픽 인터페이스와 상호 작용하려면 원격 뷰어 에서 그래픽 콘솔을 사용합니다.

- 웹 콘솔에서 VM의 CLI와 상호 작용하려면 직렬 콘솔을 사용합니다.

5.1.1. 웹 콘솔에서 가상 머신 그래픽 콘솔 보기

VM(가상 머신) 콘솔 인터페이스를 사용하면 RHEL 9 웹 콘솔에서 선택한 VM의 그래픽 출력을 볼 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 호스트와 VM 모두 그래픽 인터페이스를 지원하는지 확인합니다.

절차

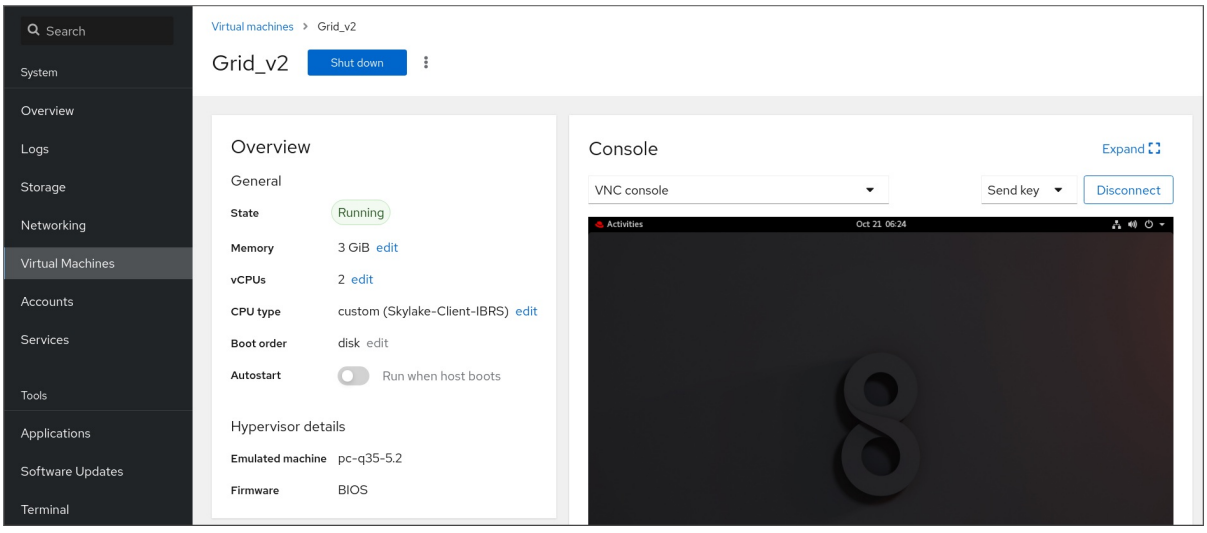
1. Virtual Machines (가상 시스템) 인터페이스에서 볼 그래픽 콘솔이 있는 VM을 클릭합니다.

   VM에 대한 개요 및 콘솔 섹션이 포함된 새 페이지가 열립니다.

2. 콘솔 드롭다운 메뉴에서 VNC 콘솔을 선택합니다.

   VNC 콘솔은 웹 인터페이스의 메뉴 아래에 표시됩니다.

   그래픽 콘솔이 웹 인터페이스에 나타납니다.



3. **Expand**(확장)를클릭합니다.

이제 실제 시스템과 동일한 방식으로 마우스와 키보드를 사용하여 VM 콘솔과 상호 작용할 수 있습니다. VM 콘솔의 디스플레이는 VM에서 수행되는 활동을 반영합니다.

참고

웹 콘솔이 실행 중인 호스트는 **Ctrl+Alt+Del** 과 같은 특정 키 조합을 인터셉트하여 VM으로 전송되지 않을 수 있습니다.

이러한 키 조합을 보내려면 키 보내기 메뉴를 클릭하고 전송할 키 시퀀스를 선택합니다.

예를 들어 **Ctrl+Alt+Del** 조합을 VM으로 보내려면 **Send**(보내기) 키를 클릭하고 **Ctrl+Alt+Del** 메뉴 항목을 선택합니다.

문제 해결

- 그래픽 콘솔을 클릭해도 효과가 없으면 콘솔을 전체 화면으로 확장하십시오. 이는 마우스 커서 오프셋의 알려진 문제입니다.

추가 리소스

- 웹 콘솔을 사용하여 원격 뷰어에서 그래픽 콘솔 보기
- 웹 콘솔에서 가상 머신 직렬 콘솔 보기

5.1.2. 웹 콘솔을 사용하여 원격 뷰어에서 그래픽 콘솔 보기

웹 콘솔 인터페이스를 사용하여 Virt Viewer와 같은 원격 뷰어에서 선택한 VM(가상 시스템)의 그래픽 콘솔을 표시할 수 있습니다.

참고

웹 콘솔에서 Virt Viewer를 시작할 수 있습니다. 다른 VNC 원격 뷰어는 수동으로 시작할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 호스트와 VM 모두 그래픽 인터페이스를 지원하는지 확인합니다.
- Virt Viewer에서 그래픽 콘솔을 보려면 웹 콘솔이 연결된 시스템에 Virt Viewer를 설치해야 합니다.

1. Launch remote viewer(원격 뷰어 시작)를 클릭합니다.

   a .vv 파일이 다운로드됨.

2. 파일을 열어 Virt Viewer를 시작합니다.

참고

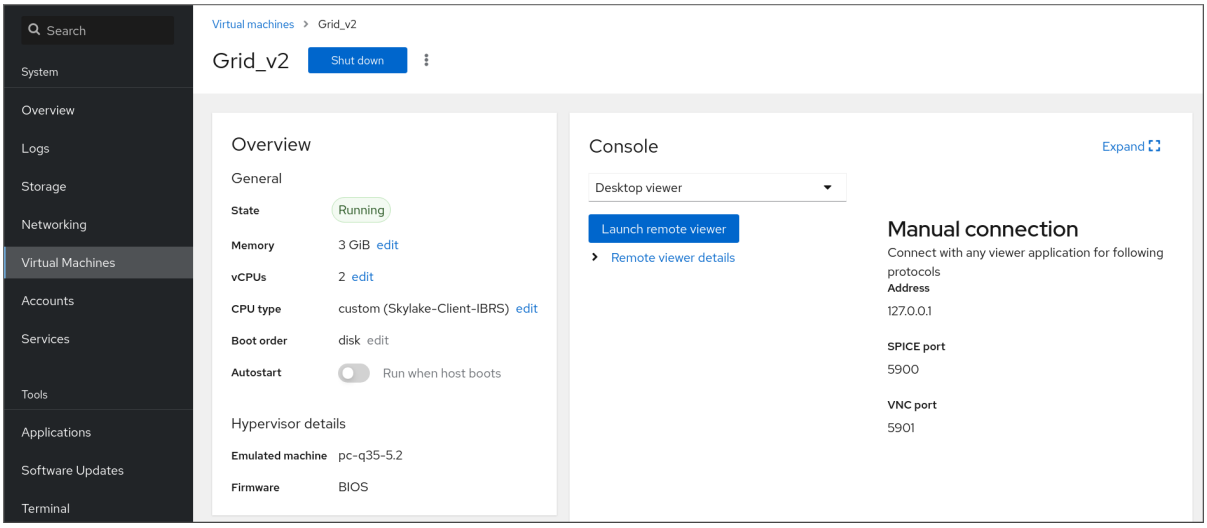
원격 뷰어는 대부분의 운영 체제에서 사용할 수 있습니다. 그러나 일부 브라우저 확장 및 플러그인에서는 웹 콘솔에서 Virt Viewer를 열 수 없습니다.

절차

1. Virtual Machines (가상 시스템) 인터페이스에서 볼 그래픽 콘솔이 있는 VM을 클릭합니다.

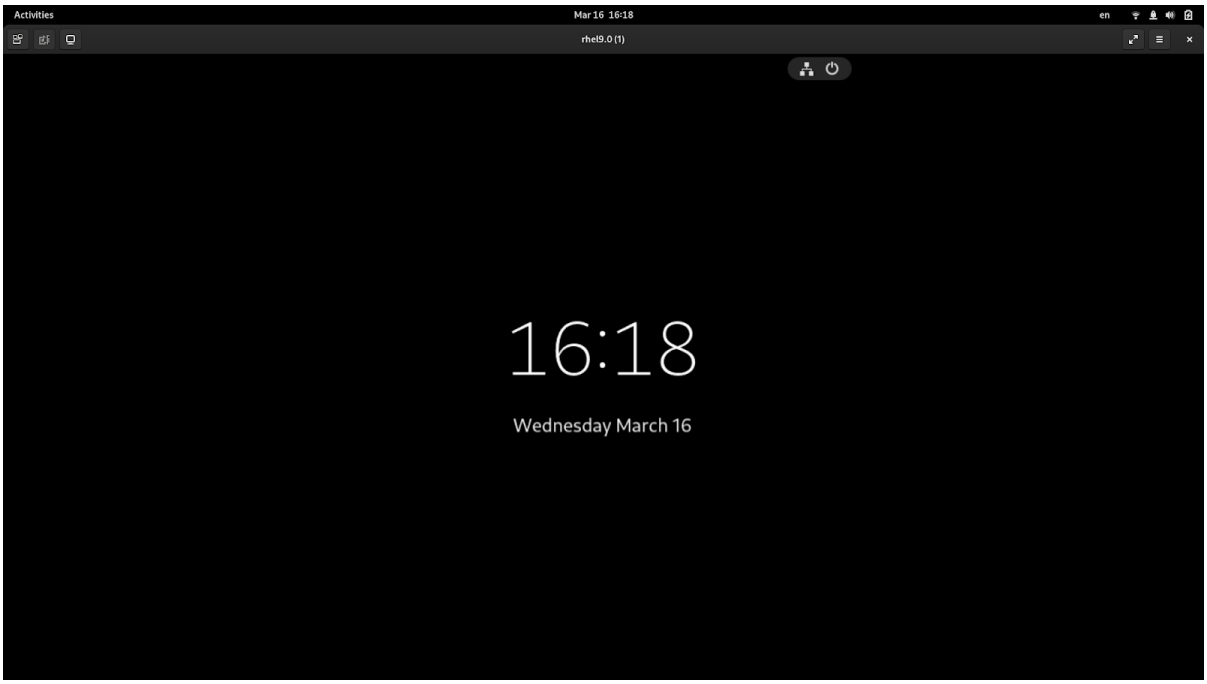
   VM에 대한 개요 및 콘솔 섹션이 포함된 새 페이지가 열립니다.

2. 콘솔 드롭다운 메뉴에서 Desktop Viewer (데스크탑 뷰어)를 선택합니다.

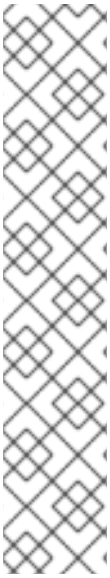


3. 원격 뷰어 시작을 클릭합니다.

그래픽 콘솔이 **Virt Viewer**(가상 뷰어)에서 열립니다.



실제 시스템과 상호 작용하는 것과 동일한 방식으로 마우스와 키보드를 사용하여 **VM** 콘솔과 상호 작용할 수 있습니다. **VM** 콘솔의 디스플레이는 **VM**에서 수행되는 활동을 반영합니다.



참고

웹 콘솔이 실행 중인 서버는 **Ctrl+Alt+Del** 과 같은 특정 키 조합을 인터셉트하여 **VM**으로 전송되지 않을 수 있습니다.

이러한 키 조합을 보내려면 키 보내기 메뉴를 클릭하고 전송할 키 시퀀스를 선택합니다.

예를 들어 **Ctrl+Alt+Del** 조합을 **VM**에 보내려면 **Send** 키 메뉴를 클릭하고 **Ctrl+Alt+Del** 메뉴 항목을 선택합니다.

문제 해결

- 그래픽 콘솔을 클릭해도 효과가 없으면 콘솔을 전체 화면으로 확장하십시오. 이는 마우스 커서 오프셋의 알려진 문제입니다.
- 웹 콘솔에서 **Remote Viewer**를 시작하는 것이 작동하지 않거나 최적이 아닌 경우 다음 프로토콜을 사용하여 뷰어 애플리케이션에 수동으로 연결할 수 있습니다.
  - 주소 **-** 기본 주소는 **127.0.0.1** 입니다. **/etc/libvirt/qemu.conf** 에서 **vnc_listen** 매개변수를 수정하여 호스트의 **IP** 주소로 변경할 수 있습니다.
  - **VNC** 포트 **- 5901**

추가 리소스

- 웹 콘솔에서 가상 머신 그래픽 콘솔 보기
- 웹 콘솔에서 가상 머신 직렬 콘솔 보기

5.1.3. 웹 콘솔에서 가상 머신 직렬 콘솔 보기

RHEL 9 웹 콘솔에서 선택한 **VM**(가상 머신)의 직렬 콘솔을 볼 수 있습니다. 이는 호스트 시스템 또는 **VM**이 그래픽 인터페이스로 구성되지 않은 경우 유용합니다.

직렬 콘솔에 대한 자세한 내용은 가상 머신 직렬 콘솔 열기를 참조하십시오.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

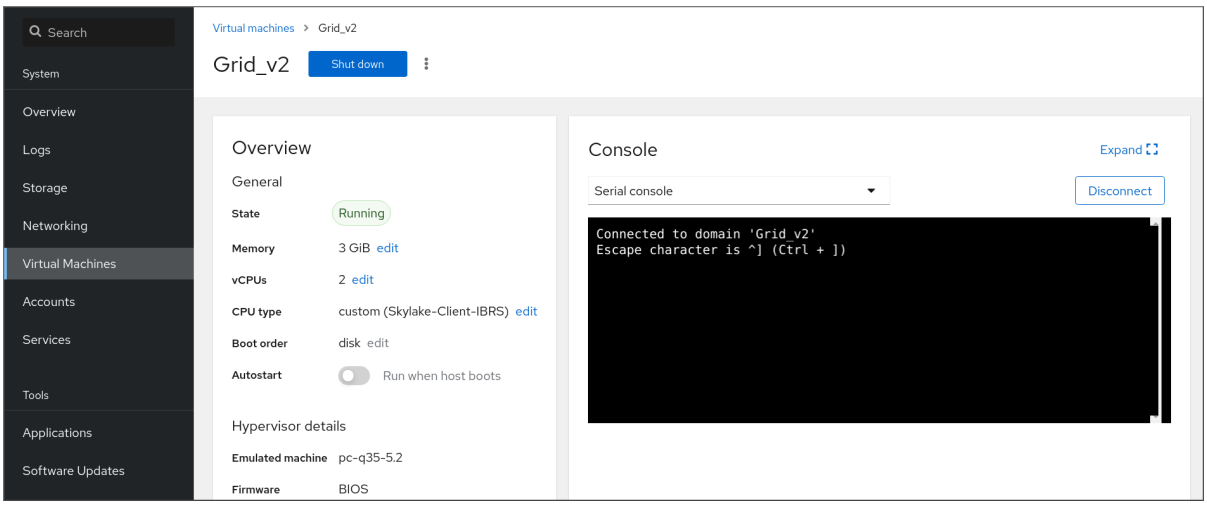
절차

1. Virtual Machines (가상 시스템) 창에서 볼 직렬 콘솔이 있는 VM을 클릭합니다.

   VM에 대한 개요 및 콘솔 섹션이 포함된 새 페이지가 열립니다.

2. 콘솔 드롭다운 메뉴에서 직렬 콘솔을 선택합니다.

   그래픽 콘솔이 웹 인터페이스에 나타납니다.



VM에서 직렬 콘솔의 연결을 해제하고 다시 연결할 수 있습니다.

- VM에서 직렬 콘솔의 연결을 끊으려면 Disconnect 를 클릭합니다.
- 직렬 콘솔을 VM에 다시 연결하려면 연결을 다시 클릭합니다.

추가 리소스

- 웹 콘솔에서 가상 머신 그래픽 콘솔 보기
- 웹 콘솔을 사용하여 원격 뷰어에서 그래픽 콘솔 보기

5.2. VIRT VIEWER를 사용하여 가상 머신 그래픽 콘솔 열기

KVM 가상 머신(VM)의 그래픽 콘솔에 연결하여 Virt Viewer 데스크탑 애플리케이션에서 시작하려면 아래 절차를 따르십시오.

사전 요구 사항

- 연결하는 VM과 시스템뿐만 아니라 그래픽 디스플레이를 지원해야 합니다.
- 대상 VM이 원격 호스트에 있는 경우 호스트에 대한 연결 및 루트 액세스 권한이 필요합니다.
- 선택 사항: 대상 VM이 원격 호스트에 있는 경우 원격 호스트에 보다 편리하게 액세스할 수 있도록 libvirt 및 SSH를 설정합니다.

절차

- 로컬 VM에 연결하려면 다음 명령을 사용하고 *guest-name* 을 연결할 VM의 이름으로 교체합니다.

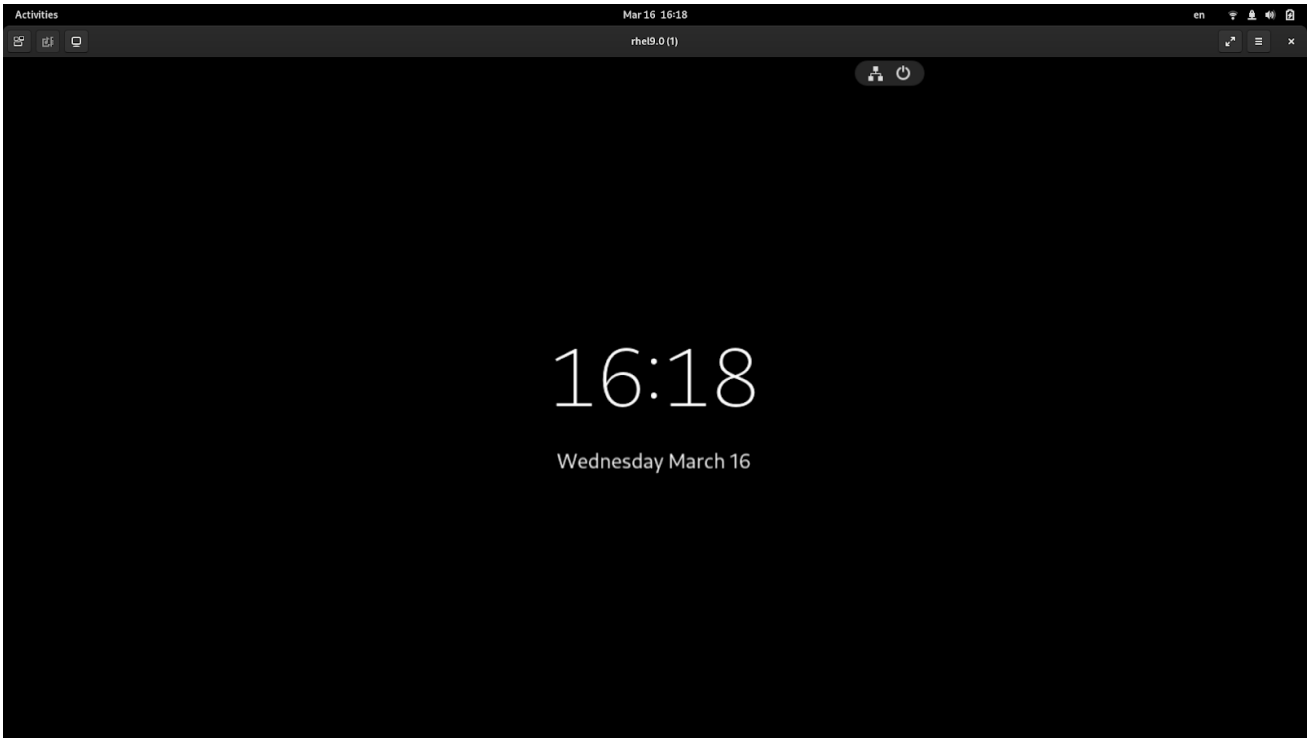
```
# virt-viewer guest-name
```

- 원격 VM에 연결하려면 SSH 프로토콜과 함께 virt-viewer 명령을 사용합니다. 예를 들어 다음 명령은 원격 시스템 10.0.0.1에 있는 *guest-name* 이라는 VM에 root로 연결합니다. 또한 연결에는 10.0.0.1에 대한 루트 인증도 필요합니다.

```
# virt-viewer --direct --connect qemu+ssh://root@10.0.0.1/system guest-name
root@10.0.0.1's password:
```

검증

연결이 올바르게 작동하는 경우 VM 디스플레이가 Virt Viewer 창에 표시됩니다.



실제 시스템과 상호 작용하는 것과 동일한 방식으로 마우스와 키보드를 사용하여 VM 콘솔과 상호 작용할 수 있습니다. VM 콘솔의 디스플레이는 VM에서 수행되는 활동을 반영합니다.

문제 해결

- 그래픽 콘솔을 클릭해도 효과가 없으면 콘솔을 전체 화면으로 확장하십시오. 이는 마우스 커서 오프셋의 알려진 문제입니다.

추가 리소스

- **virt-viewer man** 페이지
- 원격 가상화 호스트에 쉽게 액세스 가능
- 웹 콘솔을 사용하여 가상 머신과 상호 작용

5.3. SSH를 사용하여 가상 머신에 연결

SSH 연결 프로토콜을 사용하여 VM(가상 머신)의 터미널과 상호 작용하려면 다음 절차를 따르십시오.

사전 요구 사항

- 대상 VM에 대한 네트워크 연결 및 루트 액세스 권한이 있습니다.
- 대상 VM이 원격 호스트에 있는 경우 해당 호스트에 대한 연결 및 루트 액세스 권한도 있습니다.
- VM 네트워크는 libvirt 에서 생성된 dnsmasq 에 의해 IP 주소를 할당합니다. 예를 들어 libvirt NAT 네트워크 의 경우입니다.

  특히 VM에서 다음 네트워크 구성 중 하나를 사용하는 경우 SSH를 사용하여 VM에 연결할 수 없습니다.

  - hostdev 인터페이스
  - 직접 인터페이스
  - 브릿지 상호 작용
- libvirt-nss 구성 요소가 VM의 호스트에 설치 및 활성화되어 있습니다. 그렇지 않은 경우 다음을 수행합니다.

  a. libvirt-nss 패키지를 설치합니다.

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

  b. /etc/nsswitch.conf 파일을 편집하고 libvirt_guest 를 hosts 행에 추가합니다.

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

절차

1. 원격 VM에 연결할 때 먼저 물리적 호스트에 SSH를 연결합니다. 다음 예제는 루트 자격 증명

을 사용하여 호스트 시스템 10.0.0.1에 연결하는 방법을 보여줍니다.

```
# ssh root@10.0.0.1
root@10.0.0.1's password:
Last login: Mon Sep 24 12:05:36 2021
root~#
```

2. VM의 이름과 사용자 액세스 자격 증명을 사용하여 연결합니다. 예를 들어 다음은 루트 자격 증명을 사용하여 testguest1 VM에 연결됩니다.

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

문제 해결

- VM의 이름을 모르는 경우 virsh list --all 명령을 사용하여 호스트에서 사용 가능한 모든 VM을 나열할 수 있습니다.

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

추가 리소스

- 업스트림 libvirt 문서

5.4. 가상 머신 직렬 콘솔 열기

virsh console 명령을 사용하면 VM(가상 머신)의 직렬 콘솔에 연결할 수 있습니다.

이는 VM에 유용합니다.

- VNC 프로토콜을 제공하지 않으므로 GUI 툴에 대한 비디오 디스플레이를 제공하지 않습니다.
- 네트워크 연결이 없으므로 SSH를 사용하여 상호 작용할 수 없습니다.

사전 요구 사항

- VM에는 콘솔 type='pty'와 같이 직렬 콘솔 장치가 구성되어 있어야 합니다. 확인하려면 다음을 수행하십시오.

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

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

- VM에 커널 명령줄에 직렬 콘솔이 구성되어 있어야 합니다. 이를 확인하려면 VM의 cat /proc/cmdline 명령 출력에 *console=ttyS0* 이 포함되어야 합니다. 예를 들면 다음과 같습니다.

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

직렬 콘솔이 VM에 제대로 설정되지 않은 경우 virsh 콘솔 을 사용하여 VM에 연결하면 응답하지 않는 게스트 콘솔에 연결됩니다. 그러나 Ctrl+] 바로 가기를 사용하여 응답하지 않는 콘솔을 계속 종료할 수 있습니다.

- VM에서 직렬 콘솔을 설정하려면 다음을 수행합니다.

a. VM에서 /etc/default/grub 파일을 편집하고 console=ttyS0 을 GRUB_CMDLINE_LINUX 로 시작하는 행에 추가합니다.

b. 변경 사항이 적용되지 않을 수 있는 커널 옵션을 지웁니다.

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

c. Grub 구성을 다시 로드합니다.

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.0-948.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0-948.el7.x86_64.img
[...]
done
```

d.

VM을 재부팅합니다.

절차

1.

호스트 시스템에서 **virsh console** 명령을 사용합니다. 다음 예제에서는 **libvirt** 드라이버가 안전한 콘솔 처리를 지원하는 경우 ***guest1*** **VM**에 연결됩니다.

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

Subscription-name
Kernel 3.10.0-948.el7.x86_64 on an x86_64

localhost login:
```

2.

표준 명령줄 인터페이스와 동일한 방식으로 **virsh** 콘솔과 상호 작용할 수 있습니다.

추가 리소스

- **virsh man** 페이지

5.5. 원격 가상화 호스트에 쉽게 액세스 설정

libvirt 유틸리티를 사용하여 원격 호스트 시스템의 **VM**을 관리할 때 **-c qemu+ssh://root@hostname/system** 구문을 사용하는 것이 좋습니다. 예를 들어 **10.0.0.1** 호스트에서 **virsh list** 명령을 **root**로 사용하려면 다음을 실행합니다.

```
# virsh -c qemu+ssh://root@10.0.0.1/system list

root@10.0.0.1's password:

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

그러나 편의를 위해 **SSH** 및 **libvirt** 구성을 수정하여 연결 세부 정보를 전체적으로 지정할 필요가 없습니다. 예를 들어 다음을 수행할 수 있습니다.

```
# virsh -c remote-host list

root@10.0.0.1's password:
```

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

이 개선 사항을 활성화하려면 아래 지침을 따르십시오.

절차

1. ~/.ssh/config 파일을 편집하거나 생성하고 다음 내용을 추가합니다. 여기서 *host-alias* 는 특정 원격 호스트와 연결된 이름이 단축되고 *hosturl* 은 호스트의 URL 주소입니다.

```
Host host-alias
    User            root
    Hostname        hosturl
```

예를 들어 다음에서는 root@10.0.0.1의 *tyrannosaurus* 별칭을 설정합니다.

```
Host tyrannosaurus
    User            root
    Hostname        10.0.0.1
```

2. /etc/libvirt/libvirt.conf 파일을 편집하거나 생성하고 다음 내용을 추가합니다. 여기서 *qemu-host-alias* 는 QEMU 및 libvirt 유틸리티가 의도한 호스트와 연결할 호스트 별칭입니다.

```
uri_aliases = [
  "qemu-host-alias=qemu+ssh://host-alias/system",
]
```

예를 들어 다음에서는 이전 단계에서 구성한 *tyrannosaurus* 별칭을 사용하여 qemu+ssh://10.0.0.1/system 을 나타내는 *t-rex* 별칭을 설정합니다.

```
uri_aliases = [
  "t-rex=qemu+ssh://tyrannosaurus/system",
]
```

검증

1. 추가된 -c *qemu-host-alias* 매개변수가 있는 로컬 시스템에서 libvirt 기반 유틸리티를 사용하여 원격 VM을 관리할 수 있는지 확인합니다. 이 명령은 원격 호스트에서 SSH를 통해 자동으로 명령을 수행합니다.

예를 들어 다음에 10.0.0.1 원격 호스트의 VM이 나열되는지 확인합니다. 이 연결은 이전 단계에서 *t-rex* 로 설정된 연결입니다.

```
$ virsh -c t-rex list

root@10.0.0.1's password:

Id   Name           State
--------------------------------
1    velociraptor     running
```

참고

virsh 외에 -c (또는 --connect) 옵션과 위에서 설명한 원격 호스트 액세스 구성은 다음 유틸리티에서 사용할 수 있습니다.

- virt-install
- virt-viewer

다음 단계

- 단일 원격 호스트에서만 libvirt 유틸리티를 사용하려면 특정 연결을 libvirt 기반 유틸리티의 기본 대상으로 설정할 수도 있습니다. 이를 위해 /etc/libvirt/libvirt.conf 파일을 편집하고 uri_default 매개변수 값을 *qemu-host-alias* 로 설정합니다. 예를 들어 다음에서는 이전 단계에서 기본 libvirt 대상으로 설정된 *t-rex* 호스트 별칭을 사용합니다.

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

결과적으로 모든 libvirt 기반 명령이 지정된 원격 호스트에서 자동으로 수행됩니다.

```
$ virsh list
root@10.0.0.1's password:

Id   Name           State
--------------------------------
1    velociraptor     running
```

그러나 로컬 호스트 또는 다른 원격 호스트에서 VM을 관리하려면 이 방법은 권장되지 않습니

다.

- 원격 호스트에 연결할 때 원격 시스템에 루트 암호를 제공하지 않아도 됩니다. 이를 수행하려면 다음 방법 중 하나 이상을 사용합니다.
  - 원격 호스트에 대한 키 기반 SSH 액세스 설정
  - 원격 시스템에 연결하기 위해 SSH 연결 멀티플렉싱 사용
  - Identity Management의 Kerberos 인증
- -c (또는 --connect) 옵션을 사용하여 원격 호스트에서 virt-install, virt-viewer, virsh 명령을 실행할 수 있습니다.

6장. 가상 머신 종료

RHEL 9에서 호스팅되는 실행 중인 가상 머신 을 종료하려면 명령줄 인터페이스 또는 웹 콘솔 GUI를 사용합니다.

6.1. 명령줄 인터페이스를 사용하여 가상 머신 종료

반응형 가상 머신(VM)을 종료하려면 다음 중 하나를 수행합니다.

- 게스트에 연결된 동안 게스트 OS에 적합한 shutdown 명령을 사용합니다.
- 호스트에서 virsh shutdown 명령을 사용합니다.
  - VM이 로컬 호스트에 있는 경우:

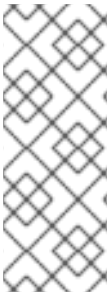
```
# virsh shutdown demo-guest1
Domain 'demo-guest1' is being shutdown
```

  - VM이 원격 호스트에 있는 경우 이 예에서는 10.0.0.1입니다.

```
# virsh -c qemu+ssh://root@10.0.0.1/system shutdown demo-guest1

root@10.0.0.1's password:
Domain 'demo-guest1' is being shutdown
```

VM이 종료되도록 하려면 예를 들어 응답하지 않는 경우 호스트에서 virsh destroy 명령을 사용합니다.

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



참고

virsh destroy 명령은 VM 구성 또는 디스크 이미지를 실제로 삭제하거나 제거하지 않습니다. 실제 시스템에서 전원을 가져오는 것과 마찬가지로 VM의 실행 중인 VM 인스턴스만 종료합니다. 따라서 드문 경우에서 virsh destroy 는 VM의 파일 시스템이 손상될 수 있으므로 다른 모든 종료 방법이 실패한 경우에만 이 명령을 사용하는 것이 좋습니다.

6.2. 웹 콘솔을 사용하여 가상 머신 종료 및 재시작

RHEL 9 웹 콘솔을 사용하여 실행 중인 가상 머신을 종료 하거나 다시 시작할 수 있습니다. 비마스크할 수 없는 인터럽트를 응답하지 않는 가상 머신에 보낼 수도 있습니다.

6.2.1. 웹 콘솔에서 가상 머신 종료

VM(가상 머신)이 running 상태인 경우 RHEL 9 웹 콘솔을 사용하여 종료할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. Virtual Machines (가상 시스템) 인터페이스에서 종료할 VM의 행을 찾습니다.
2. 행 오른쪽에서 Shut Down 을 클릭합니다.

   VM이 종료됩니다.

문제 해결

- VM이 종료되지 않은 경우 종료 버튼 옆에 있는 메뉴 버튼을 클릭하고 Force Shut down 을 선택합니다.
- 응답하지 않는 VM을 종료하려면 마스킹 불가능한 인터럽트를 보낼 수도 있습니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 시작
- 웹 콘솔을 사용하여 가상 머신 재시작

6.2.2. 웹 콘솔을 사용하여 가상 머신 재시작

VM(가상 머신)이 running 상태인 경우 RHEL 9 웹 콘솔을 사용하여 다시 시작할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. 가상 머신 인터페이스에서 재시작할 VM 행을 찾습니다.
2. 행 오른쪽에서 메뉴 버튼을 클릭합니다 .

   작업 드롭다운 메뉴가 나타납니다.
3. 드롭다운 메뉴에서 재부팅 을 클릭합니다.

   VM이 종료되고 다시 시작됩니다.

문제 해결

- VM을 재시작하지 않으면 Restart 버튼 옆에 있는 메뉴 버튼을 클릭하고 Restart 를 선택합니다.
- 응답하지 않는 VM을 종료하려면 마스킹 불가능한 인터럽트를 보낼 수도 있습니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 시작
- 웹 콘솔에서 가상 머신 종료

6.2.3. 웹 콘솔을 사용하여 VM으로 마스킹할 수 없는 인터럽트 전송

NMI(Non-maskable interrupt)를 전송하면 응답하지 않는 VM(가상 머신)이 응답하지 않거나 종료될

수 있습니다. 예를 들어, Ctrl+Alt+Del NMI를 표준 입력에 응답하지 않는 VM에 보낼 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. 가상 머신 인터페이스에서 NMI를 보낼 VM 행을 찾습니다.
2. 행 오른쪽에서 메뉴 버튼을 클릭합니다 .

   작업 드롭다운 메뉴가 나타납니다.
3. 드롭다운 메뉴에서 Send Non-Maskable Interrupt 를 클릭합니다.

   NMI가 VM으로 전송됩니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 시작
- 웹 콘솔을 사용하여 가상 머신 재시작
- 웹 콘솔에서 가상 머신 종료

7장. 가상 머신 삭제

RHEL 9에서 가상 머신을 삭제하려면 명령줄 인터페이스 또는 웹 콘솔 GUI 를 사용하십시오.

7.1. 명령줄 인터페이스를 사용하여 가상 머신 삭제

VM(가상 머신)을 삭제하려면 명령줄을 사용하여 호스트에서 XML 구성 및 관련 스토리지 파일을 제거할 수 있습니다. 아래 절차를 따르십시오.

사전 요구 사항

- VM에서 중요한 데이터를 백업합니다.
- VM을 종료합니다.
- 다른 VM에서 동일한 관련 스토리지를 사용하지 않도록 합니다.

절차

- virsh undefine 유틸리티를 사용합니다.

  예를 들어 다음 명령은 *guest1* VM, 연결된 스토리지 볼륨 및 비휘발성 RAM을 제거합니다.

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

추가 리소스

- virsh undefine --help 명령
- virsh man 페이지

7.2. 웹 콘솔을 사용하여 가상 머신 삭제

RHEL 9 웹 콘솔이 연결된 호스트에서 VM(가상 머신) 및 관련 스토리지 파일을 삭제하려면 다음 절차를 따르십시오.

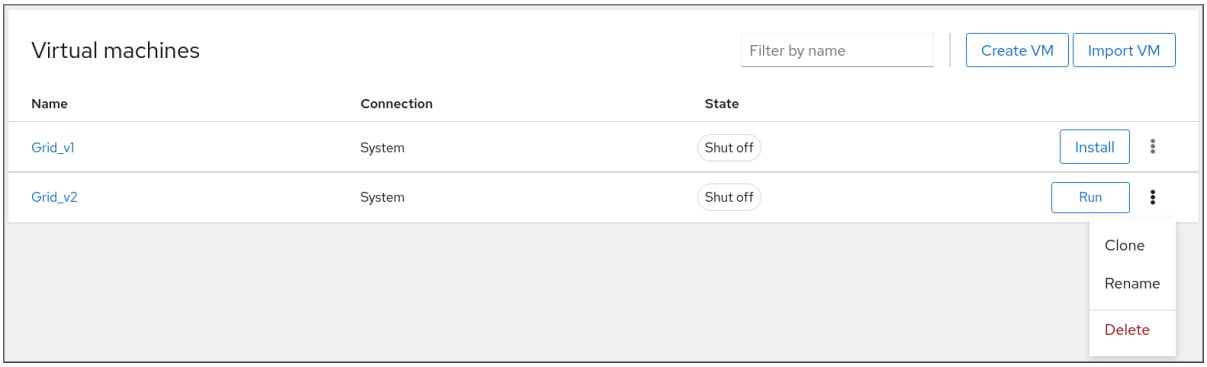
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- VM에서 중요한 데이터를 백업합니다.
- 다른 VM이 동일한 연결된 스토리지를 사용하지 않는지 확인합니다.
- 선택 사항: VM을 종료합니다.

절차

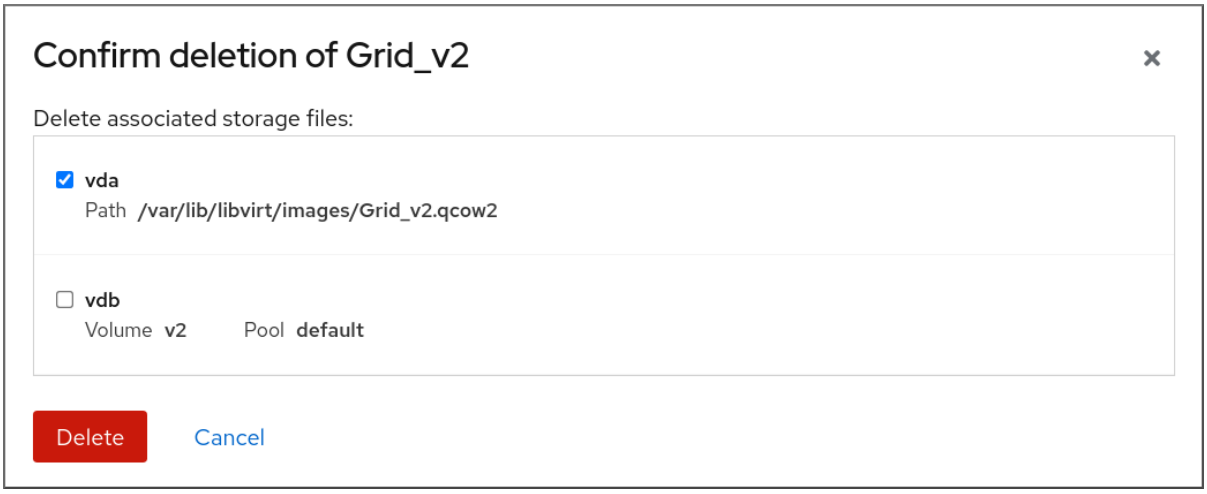
1. Virtual Machines 인터페이스에서 삭제할 VM 의 메뉴 버튼을 클릭합니다.

   다양한 VM 작업에 대한 제어와 함께 드롭다운 메뉴가 표시됩니다.



2. 삭제를 클릭합니다.

   확인 대화 상자가 나타납니다.



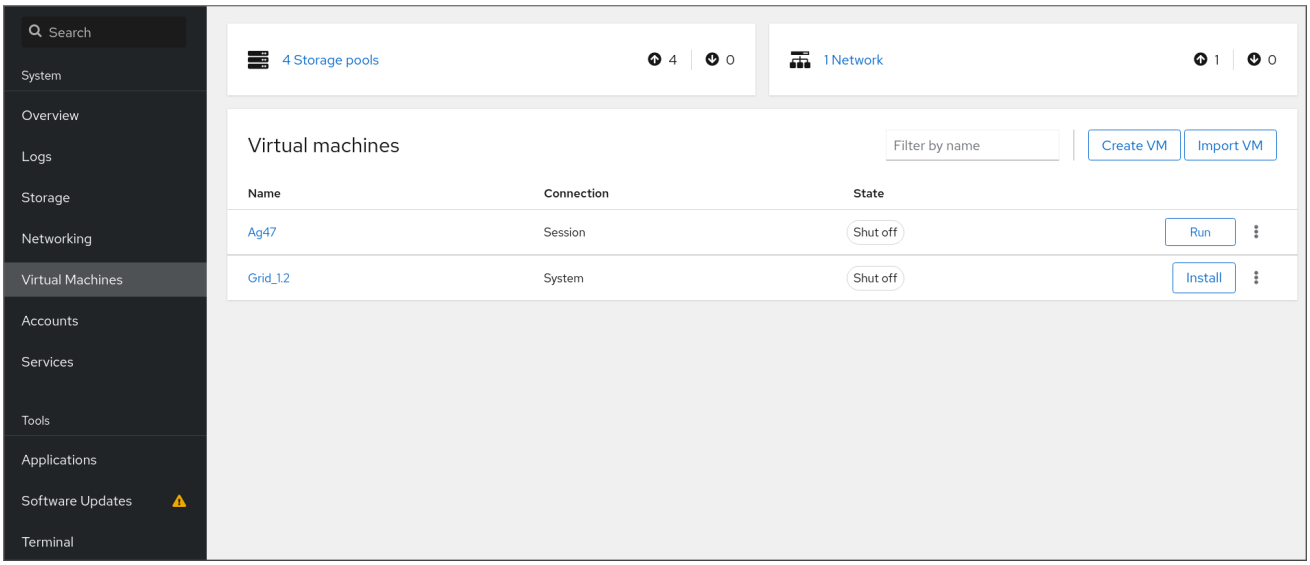
3. **선택 사항: VM과 연결된 스토리지 파일 전체 또는 일부를 삭제하려면 삭제하려는 스토리지 파일 옆에 있는 확인란을 선택합니다.**

4. **삭제를 클릭합니다.**

VM 및 선택한 스토리지 파일이 삭제됩니다.

8장. 웹 콘솔에서 가상 머신 관리

RHEL 9 호스트의 그래픽 인터페이스에서 가상 머신을 관리하려면 RHEL 9 웹 콘솔에서 Virtual Machines 창을 사용할 수 있습니다.



8.1. 웹 콘솔을 사용한 가상 머신 관리 개요

RHEL 9 웹 콘솔은 시스템 관리를 위한 웹 기반 인터페이스입니다. 웹 콘솔은 해당 기능 중 하나로 호스트 시스템에서 VM(가상 머신)의 그래픽 보기를 제공하며 이러한 VM을 생성, 액세스 및 구성할 수 있습니다.

웹 콘솔을 사용하여 RHEL 9에서 VM을 관리하려면 먼저 가상화용 웹 콘솔 플러그인을 설치해야 합니다.

다음 단계

- 웹 콘솔에서 VM 관리를 활성화하는 방법에 대한 지침은 가상 시스템을 관리하기 위한 웹 콘솔 설정을 참조하십시오.
- 웹 콘솔에서 제공하는 VM 관리 작업의 포괄적인 목록은 웹 콘솔에서 사용 가능한 가상 머신 관리 기능을 참조하십시오.

8.2. 가상 머신 관리를 위한 웹 콘솔 설정

RHEL 9 웹 콘솔을 사용하여 VM(가상 머신)을 관리하려면 호스트에 웹 콘솔 가상 머신 플러그인을 설치해야 합니다.

사전 요구 사항

- 웹 콘솔이 시스템에 설치되어 활성화되어 있는지 확인합니다.

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

이 명령에서 **Unit cockpit.socket**을 반환할 수 없는 경우 웹 콘솔 설치 문서를 따라 웹 콘솔을 활성화합니다.

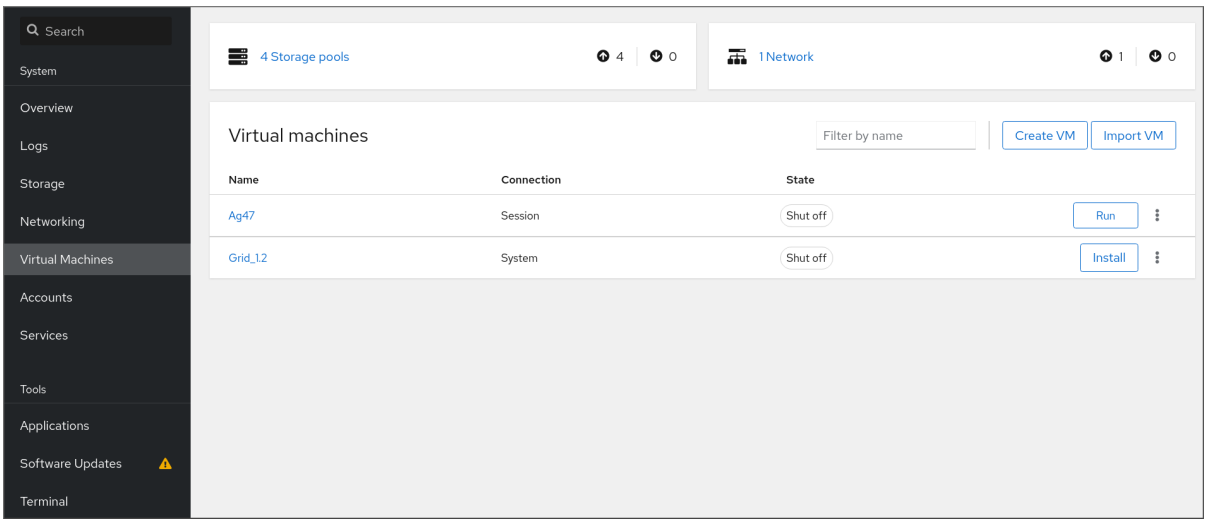
절차

- **cockpit-machines** 플러그인을 설치합니다.

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

검증

1. 웹 콘솔에 액세스합니다(예: 브라우저에서 **https://localhost:9090** 주소를 입력하여).
2. 로그인합니다.
3. 설치에 성공하면 **Virtual Machines**(가상 시스템) 가 웹 콘솔 사이드 메뉴에 나타납니다.



추가 리소스

- RHEL 9 웹 콘솔을 사용하여 시스템 관리

8.3. 웹 콘솔을 사용하여 가상 머신 이름 변경

VM(가상 머신)을 생성한 후 충돌을 피하도록 VM의 이름을 바꾸거나 사용 사례에 따라 새로운 고유 이름을 할당할 수 있습니다. RHEL 웹 콘솔을 사용하여 VM의 이름을 변경할 수 있습니다.

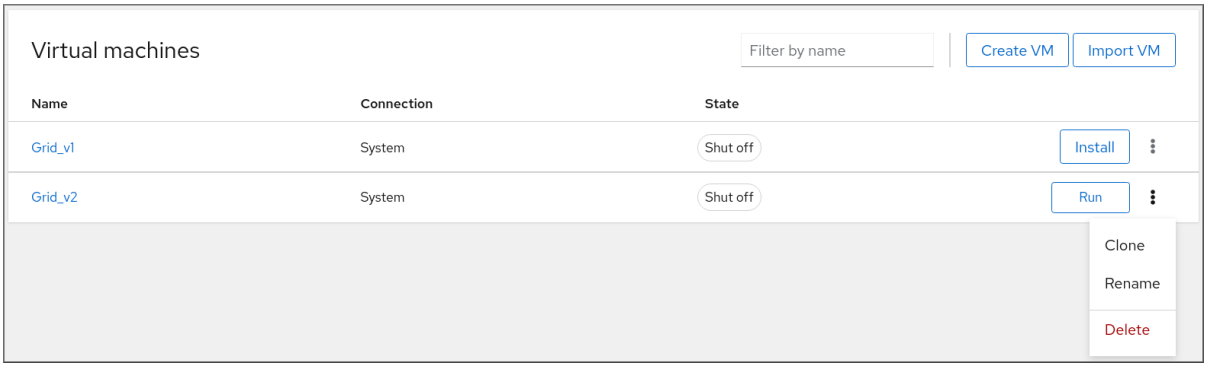
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- VM이 종료되었는지 확인합니다.

절차

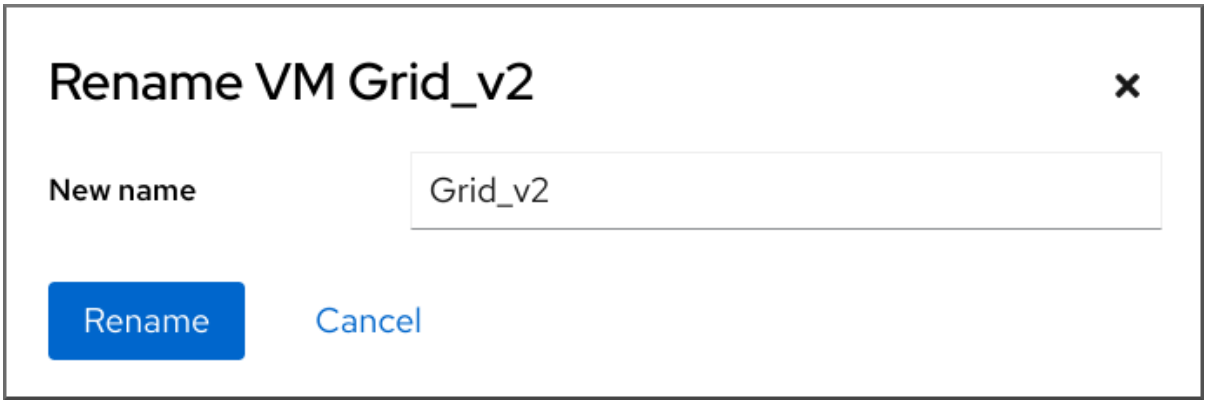
1. Virtual Machines (가상 머신) 인터페이스에서 이름을 변경할 VM의 메뉴 버튼을 클릭합니다.

   다양한 VM 작업에 대한 제어와 함께 드롭다운 메뉴가 표시됩니다.



2. Rename 을 클릭합니다.

   Rename a VM 대화 상자가 나타납니다.



3. **New name (새 이름) 필드에 VM의 이름을 입력합니다.**

4. **Rename 을 클릭합니다.**

검증

- **새 VM 이름이 Virtual Machines 인터페이스에 표시되어야 합니다.**

8.4. 웹 콘솔에서 사용할 수 있는 가상 머신 관리 기능

RHEL 9 웹 콘솔을 사용하여 시스템에서 가상 머신(VM)을 관리하기 위해 다음 작업을 수행할 수 있습니다.

표 8.1. RHEL 9 웹 콘솔에서 수행할 수 있는 VM 관리 작업

Task	자세한 내용은 참조하십시오.
VM을 생성하고 게스트 운영 체제로 설치	웹 콘솔을 사용하여 가상 머신 생성 및 게스트 운영 체제 설치
VM 삭제	웹 콘솔을 사용하여 가상 머신 삭제
VM을 시작, 종료, 재시작	웹 콘솔 을 사용하여 가상 머신 시작 및 웹 콘솔을 사용하여 가상 머신 종료 및 재시작
다양한 콘솔을 사용하여 VM에 연결하고 상호 작용	웹 콘솔을 사용하여 가상 머신과 상호 작용
VM에 대한 다양한 정보 보기	웹 콘솔을 사용하여 가상 머신 정보 보기
VM에 할당된 호스트 메모리 조정	웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거
VM의 네트워크 연결 관리	가상 머신 네트워크 인터페이스 관리에 웹 콘솔 사용

Task	자세한 내용은 참조하십시오.
호스트에서 사용 가능한 VM 스토리지를 관리하고 가상 디스크를 VM에 연결합니다.	웹 콘솔을 사용하여 가상 머신용 스토리지 관리
VM의 가상 CPU 설정 구성	웹 콘솔을 사용하여 virtal CPU 관리
VM 실시간 마이그레이션	웹 콘솔을 사용하여 가상 머신 실시간 마이그레이션
호스트 장치 관리	웹 콘솔을 사용하여 호스트 장치 관리

9장. 가상 머신에 대한 정보 보기

RHEL 9에서 가상화 배포의 측면을 조정하거나 해결해야 할 첫 번째 단계는 일반적으로 가상 머신의 현재 상태 및 구성에 대한 정보를 확인하는 것입니다. 이를 위해 명령줄 인터페이스 또는 웹 콘솔을 사용할 수 있습니다. VM의 XML 구성에서 정보를 볼 수도 있습니다.

9.1. 명령줄 인터페이스를 사용하여 가상 머신 정보 보기

호스트 및 해당 구성에서 VM(가상 시스템)에 대한 정보를 검색하려면 다음 명령 중 하나 이상을 사용합니다.

절차

- 호스트에서 VM 목록을 가져오려면 다음을 수행합니다.

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

- 특정 VM에 대한 기본 정보를 얻으려면 다음을 수행합니다.

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

- 특정 VM의 전체 XML 구성을 얻으려면 다음을 수행합니다.

```
# virsh dumpxml testguest2

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

- VM 디스크 및 기타 블록 장치에 대한 정보를 보려면 다음을 수행합니다.

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

- VM의 파일 시스템 및 해당 마운트 지점에 대한 정보를 얻으려면 다음을 수행합니다.

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

- 특정 VM의 vCPU에 대한 자세한 내용을 확인하려면 다음을 수행하십시오.

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

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

VM에서 vCPU를 구성하고 최적화하려면 가상 머신 CPU 성능 최적화를 참조하십시오.

- 호스트의 모든 가상 네트워크 인터페이스를 나열하려면 다음을 수행합니다.

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

특정 인터페이스에 대한 자세한 내용은 다음을 수행합니다.

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

네트워크 인터페이스, VM 네트워크 및 구성 방법에 대한 자세한 내용은 가상 머신 네트워크 연결 구성을 참조하십시오.

9.2. 웹 콘솔을 사용하여 가상 머신 정보 보기

RHEL 9 웹 콘솔을 사용하면 웹 콘솔 세션이 액세스할 수 있는 모든 VM 및 스토리지 풀에 대한 정보를 볼 수 있습니다.

웹 콘솔 세션이 연결된 선택된 VM에 대한 정보를 볼 수 있습니다. 여기에는 디스크,가상 네트워크 인터페이스 및 리소스 사용량에 대한 정보가 포함됩니다.

9.2.1. 웹 콘솔에서 가상화 개요 보기

웹 콘솔을 사용하여 사용 가능한 VM(가상 머신), 스토리지 풀 및 네트워크에 대한 요약된 정보가 포함된 가상화 개요에 액세스할 수 있습니다.

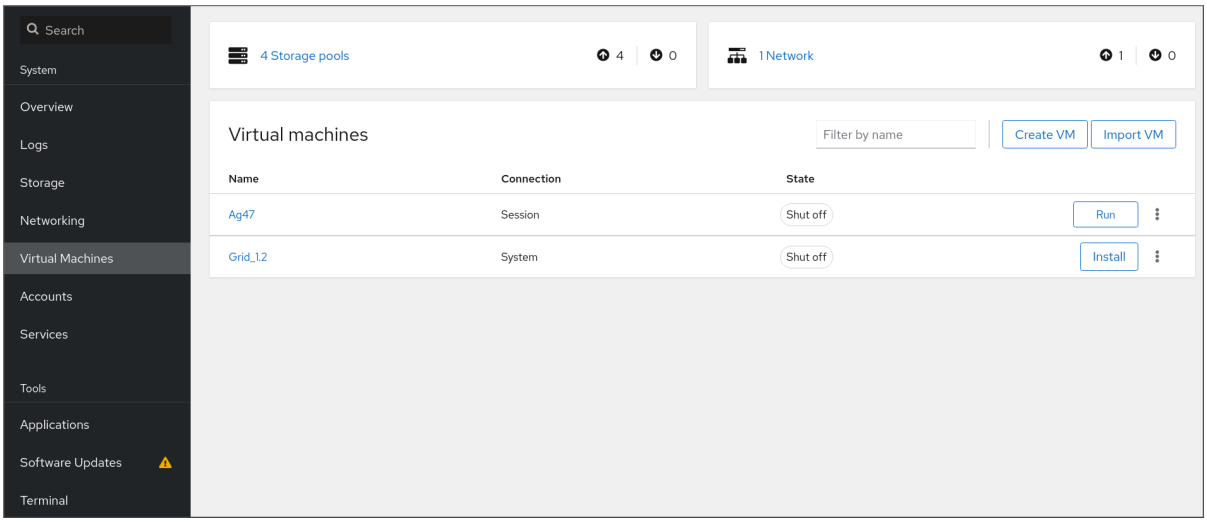
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

- 웹 콘솔의 사이드 메뉴에서 Virtual Machines (가상 머신)를 클릭합니다.

사용 가능한 스토리지 풀, 사용 가능한 네트워크 및 웹 콘솔이 연결된 VM에 대한 정보가 포함된 대화 상자가 나타납니다.



정보에는 다음이 포함됩니다.

- 스토리지 풀 - 웹 콘솔 및 해당 상태로 액세스할 수 있는 활성 또는 비활성 스토리지 풀 수입니다.
- 네트워크 - 웹 콘솔 및 해당 상태를 통해 액세스할 수 있는 활성 또는 비활성 네트워크 수입니다.
- name - VM의 이름입니다.
- 연결 - libvirt 연결 유형, 시스템 또는 세션입니다.
- 상태 - VM의 상태입니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 정보 보기

9.2.2. 웹 콘솔을 사용하여 스토리지 풀 정보 보기

웹 콘솔을 사용하면 시스템에서 사용 가능한 스토리지 풀에 대한 자세한 정보를 볼 수 있습니다. 스토

리지 풀을 사용하여 가상 머신의 디스크 이미지를 생성할 수 있습니다.

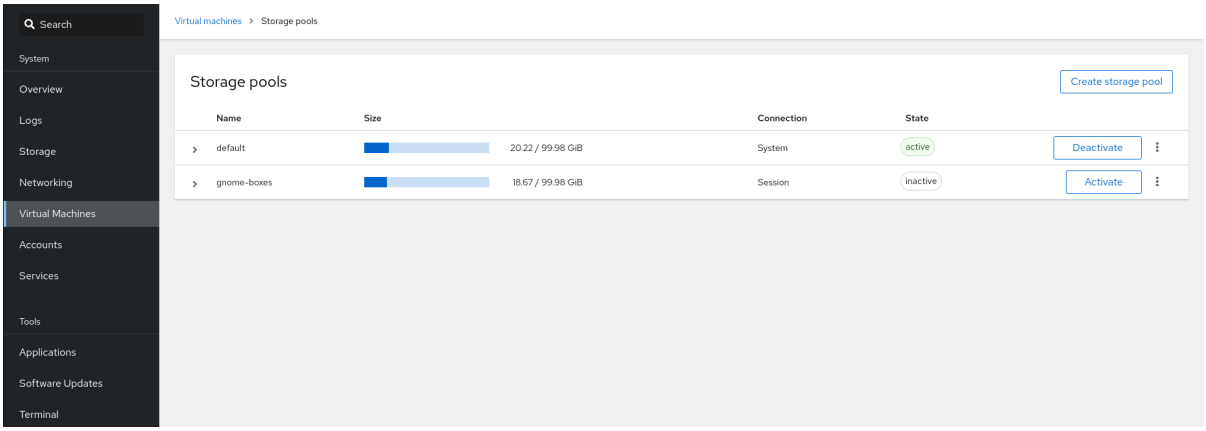
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. 가상 머신 인터페이스 상단에 있는 스토리지 풀 을 클릭합니다.

   구성된 스토리지 풀 목록을 보여주는 스토리지 풀 창이 표시됩니다.

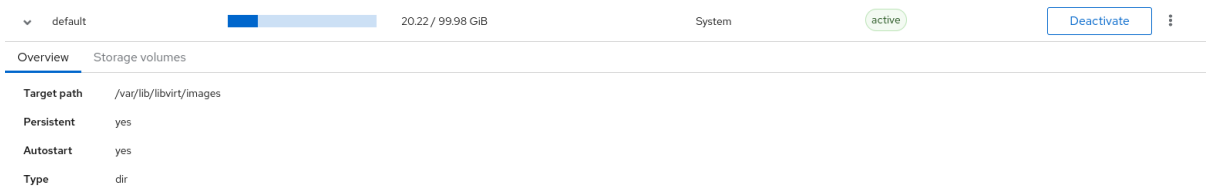


   정보에는 다음이 포함됩니다.

   - **Name -** 스토리지 풀의 이름입니다.
   - **size -** 스토리지 풀의 현재 할당 및 총 용량입니다.
   - 연결 - 스토리지 풀에 액세스하는 데 사용되는 연결입니다.
   - 상태 - 스토리지 풀의 상태입니다.

2. 정보를 보려는 스토리지 풀 옆에 있는 화살표를 클릭합니다.

행이 확장되어 선택한 스토리지 풀에 대한 자세한 정보가 포함된 개요 창을 표시합니다.

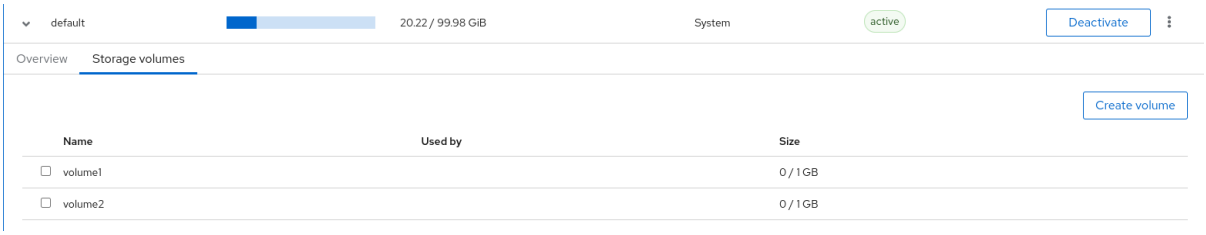


이 정보에는 다음이 포함됩니다.

- 대상 경로 - 디렉터리에서 지원하는 스토리지 풀 유형 소스(예: **dir** 또는 **netfs**)입니다.
- **persistent** - 스토리지 풀에 영구 구성이 있는지 여부를 나타냅니다.
- **autostart** - 시스템 부팅 시 스토리지 풀이 자동으로 시작되는지 여부를 나타냅니다.
- **type** - 스토리지 풀의 유형입니다.

3. 스토리지 풀과 연결된 스토리지 볼륨 목록을 보려면 **스토리지 볼륨** 을 클릭합니다.

구성된 스토리지 볼륨 목록을 보여주는 **Storage Volumes(스토리지 볼륨)** 창이 표시됩니다.



이 정보에는 다음이 포함됩니다.

- **name** - 스토리지 볼륨의 이름입니다.
- 사용됨 - 현재 스토리지 볼륨을 사용 중인 VM입니다.

- **size -** 볼륨의 크기입니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 정보 보기

9.2.3. 웹 콘솔에서 기본 가상 머신 정보 보기

웹 콘솔을 사용하면 선택한 VM(가상 머신)에 대한 할당된 리소스 또는 하이퍼바이저 세부 정보와 같은 기본 정보를 볼 수 있습니다.

사전 요구 사항

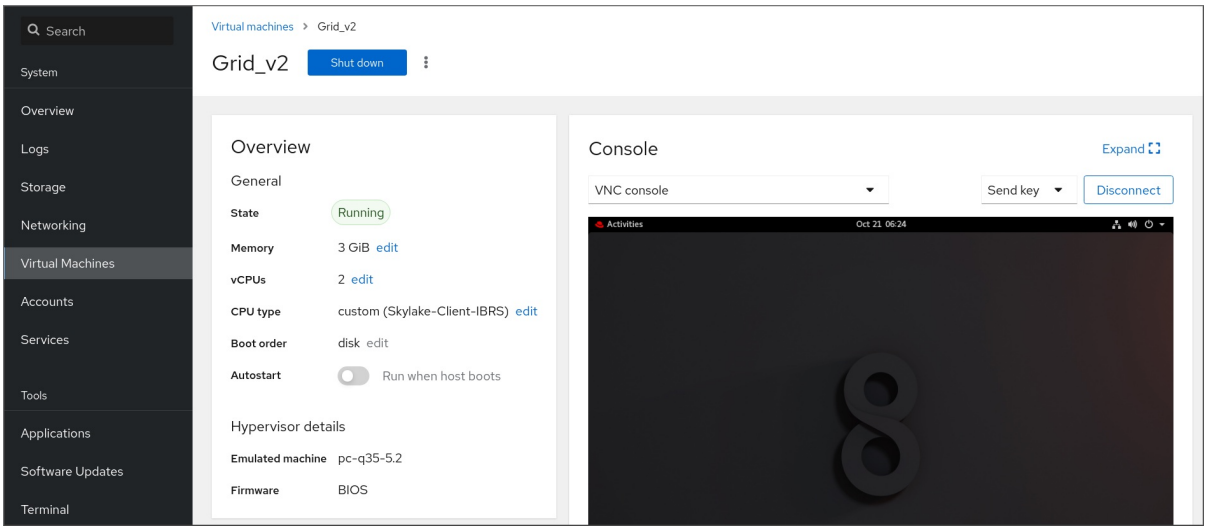
- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. 웹 콘솔의 사이드 메뉴에서 **Virtual Machines** (가상 머신)를 클릭합니다.

2. 보려는 정보를 원하는 VM을 클릭합니다.

VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 **Overview**(개요) 섹션이 포함된 새 페이지가 열립니다.



개요 섹션에는 다음과 같은 일반적인 VM 세부 정보가 포함됩니다.

- 상태 - VM 상태, 실행 중 또는 종료됨.
- memory - VM에 할당된 메모리 양입니다.
- vCPU - VM에 대해 구성된 가상 CPU 수입니다.
- CPU 유형 - VM에 대해 구성된 가상 CPU의 아키텍처입니다.
- 부팅 순서 - VM에 대해 구성된 부팅 순서입니다.
- auto start - VM에 대해 자동 시작이 활성화되었는지 여부를 나타냅니다.

이 정보에는 다음과 같은 하이퍼바이저 세부 정보도 포함됩니다.

- 에뮬레이션된 시스템 - VM에서 에뮬레이션한 시스템 유형입니다.
- 펌웨어 - VM의 펌웨어입니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 정보 보기
- 웹 콘솔을 사용하여 가상 CPU 관리

9.2.4. 웹 콘솔에서 가상 머신 리소스 사용량 보기

웹 콘솔을 사용하여 선택한 VM(가상 머신)의 메모리 및 가상 CPU 사용량을 볼 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

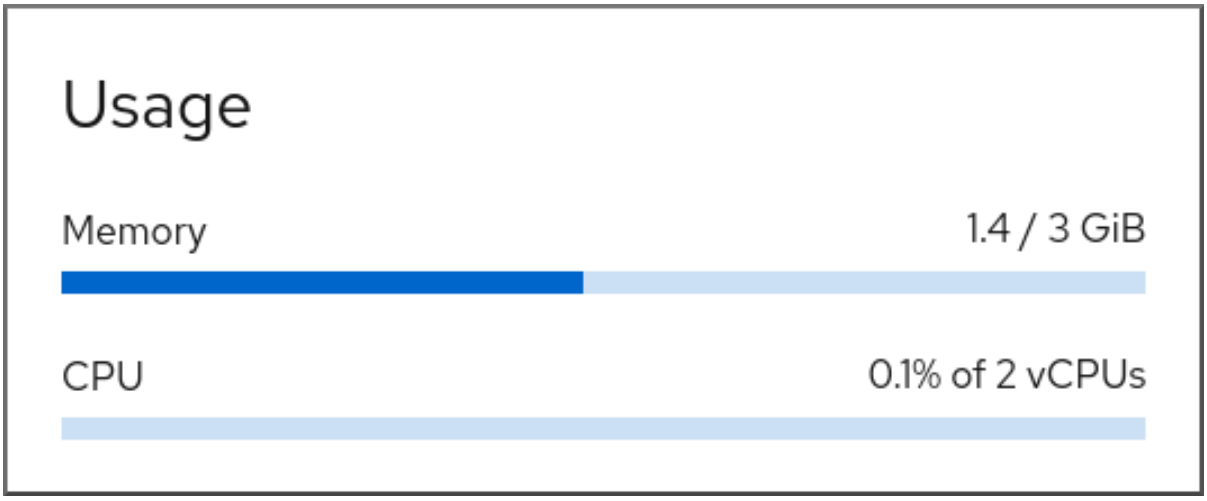
절차

1. Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

2. 사용법 으로 스크롤합니다.

   Usage(사용) 섹션에는 VM의 메모리 및 가상 CPU 사용에 대한 정보가 표시됩니다.



추가 리소스

- 웹 콘솔을 사용하여 가상 머신 정보 보기

9.2.5. 웹 콘솔에서 가상 머신 디스크 정보 보기

웹 콘솔을 사용하면 선택한 VM(가상 머신)에 할당된 디스크에 대한 자세한 정보를 볼 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

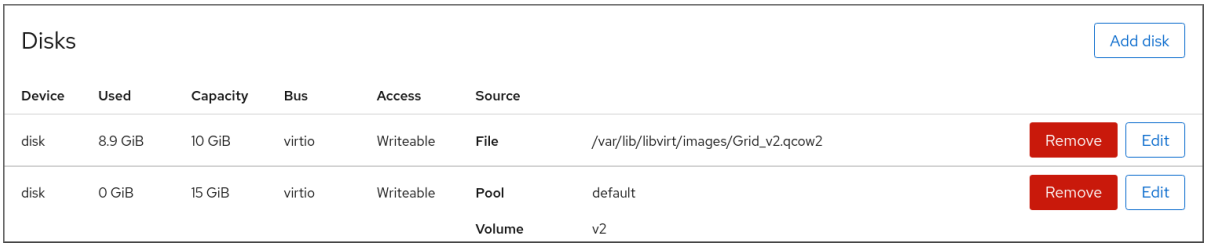
절차

1. 보려는 정보를 원하는 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

2. 디스크로 스크롤합니다.

   디스크 섹션에는 VM에 할당된 디스크 정보와 디스크 추가,제거 또는 편집 옵션이 표시됩니다.



정보에는 다음이 포함됩니다.

- device - 디스크의 장치 유형입니다.
- used - 현재 할당된 디스크 양입니다.
- capacity - 스토리지 볼륨의 최대 크기입니다.
- bus - 에뮬레이션된 디스크 장치의 유형입니다.
- 액세스 - 디스크가 쓰기 가능 하거나 읽기 전용인지 여부입니다. 원시 디스크의 경우 쓰기 가능 및 공유에 대한 액세스를 설정할 수도 있습니다.

- **Source** - 디스크 장치 또는 파일입니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 정보 보기

9.2.6. 웹 콘솔에서 가상 네트워크 인터페이스 정보 보기 및 편집

RHEL 9 웹 콘솔을 사용하면 선택한 VM(가상 머신)에서 가상 네트워크 인터페이스를 보고 수정할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. **Virtual Machines** (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 **Overview**(개요) 섹션이 포함된 새 페이지가 열립니다.

2. 네트워크 인터페이스로 스크롤합니다.

   Networks Interfaces(네트워크 인터페이스) 섹션에는 VM에 대해 구성된 가상 네트워크 인터페이스와 네트워크 인터페이스 추가,삭제,편집 또는 연결 해제 옵션이 표시됩니다 .

+ 정보에는 다음이 포함됩니다.

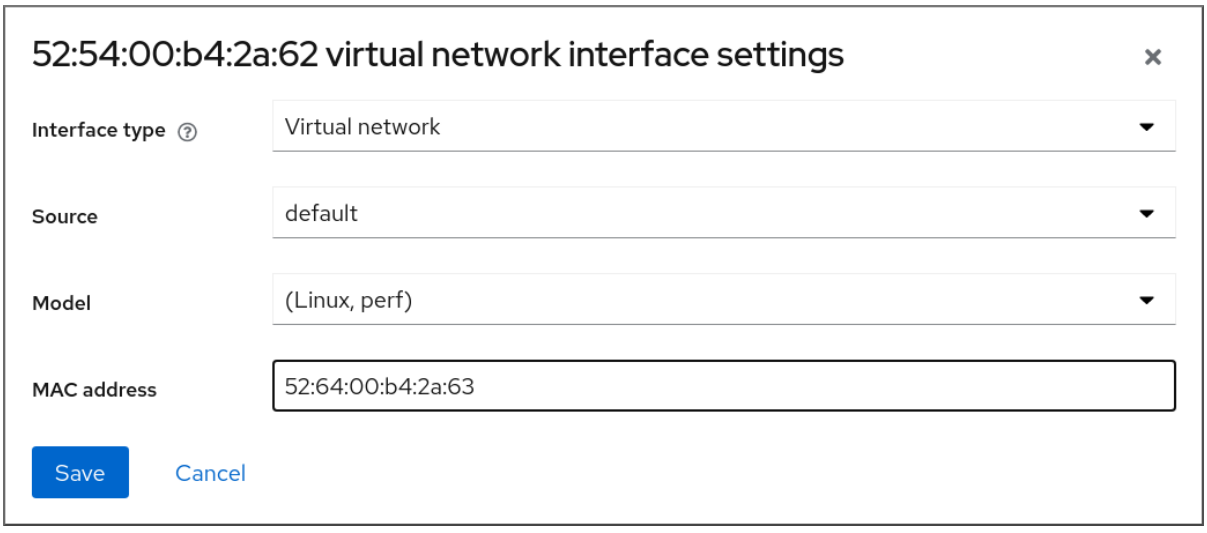
- **Type** - VM의 네트워크 인터페이스 유형입니다. 유형에는 가상 네트워크, LAN에 대한 브리지, 직접 연결이 포함됩니다.

참고

RHEL 9 이상에서는 일반 이더넷 연결이 지원되지 않습니다.

- 모델 유형 - 가상 네트워크 인터페이스의 모델입니다.
- MAC 주소 - 가상 네트워크 인터페이스의 MAC 주소입니다.
- IP 주소 - 가상 네트워크 인터페이스의 IP 주소입니다.
- Source - 네트워크 인터페이스의 소스입니다. 이는 네트워크 유형에 따라 다릅니다.
- State - 가상 네트워크 인터페이스의 상태입니다.

3. 가상 네트워크 인터페이스 설정을 편집하려면 편집 을 클릭합니다. 가상 네트워크 인터페이스 설정 대화 상자가 열립니다.



4. 인터페이스 유형, 소스, 모델 또는 MAC 주소를 변경합니다.

5. 저장을 클릭합니다. 네트워크 인터페이스가 변경되었습니다.

참고

가상 네트워크 인터페이스 설정 변경 사항은 VM을 다시 시작한 후에만 적용됩니다.

또한 MAC 주소는 VM이 종료된 경우에만 수정할 수 있습니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 정보 보기

9.3. 가상 머신 XML 구성 샘플

VM의 XML 구성은 *도메인 XML* 이라고도 하며 VM의 설정 및 구성 요소를 결정합니다. 다음 표에서는 VM(가상 머신)의 샘플 XML 구성 섹션을 보여주고 내용을 설명합니다.

VM의 XML 구성을 얻으려면 virsh dumpxml 명령 뒤에 VM 이름을 사용할 수 있습니다.

```
# virsh dumpxml testguest1
```

표 9.1. 샘플 XML 구성

도메인 XML 섹션	설명
<domain type='kvm'> <name>Testguest1</name> <uuid>ec6fbaa1-3eb4-49da-bf61-bb02fbec4967</uuid> <memory unit='KiB'>1048576</memory> <currentMemory unit='KiB'>1048576</currentMemory>	이 가상 머신은 *Testguest1* 이라는 KVM 가상 머신이며 1024MiB RAM이 할당됩니다.
<vcpu placement='static'>1</vcpu>	VM은 단일 가상 CPU(vCPU)로 할당됩니다. vCPU 구성에 대한 자세한 내용은 가상 머신 CPU 성능 최적화를 참조하십시오.

도메인 XML 섹션	설명
<os> <type arch='x86_64' machine='pc-q35-rhel9.0.0'>hvm</type> <boot dev='hd'/> </os>	머신 아키텍처는 AMD64 및 Intel 64 아키텍처로 설정되고 Intel Q35 시스템 유형을 사용하여 기능 호환성을 결정합니다. OS는 하드 드라이브에서 부팅되도록 설정되어 있습니다. 설치된 OS를 사용하여 VM을 생성하는 방법에 대한 자세한 내용은 웹 콘솔을 사용하여 가상 머신 생성 및 게스트 운영 체제 설치를 참조하십시오.
<features> <acpi/> <apic/> </features>	**acpi** 및 **apic** 하이퍼바이저 기능은 비활성화되어 있습니다.
<cpu mode='host-model' check='partial'/>	기능 XML의 호스트 CPU 정의(**virsh domcapabilities**로 가져올 수 있음)는 VM의 XML 구성에 자동으로 복사됩니다. 따라서 VM이 부팅되면 **libvirt** 에서 호스트 CPU와 유사한 CPU 모델을 선택한 다음 호스트 모델에 최대한 가깝게 추가 기능을 추가합니다.
<clock offset='utc'> <timer name='rtc' tickpolicy='catchup'/> <timer name='pit' tickpolicy='delay'/> <timer name='hpet' present='no'/> </clock>	VM의 가상 하드웨어 클럭은 UTC 시간대를 사용합니다. 또한 QEMU 하이퍼바이저와 동기화하기 위해 세 가지 다른 타이머가 설정됩니다.
<on_poweroff>destroy</on_poweroff> <on_reboot>restart</on_reboot> <on_crash>destroy</on_crash>	VM의 전원을 끄거나 OS가 예기치 않게 종료되면 **libvirt** 는 VM을 종료하고 할당된 모든 리소스를 해제합니다. VM이 재부팅되면 **libvirt** 가 동일한 구성으로 다시 시작합니다.
<pm> <suspend-to-mem enabled='no'/> <suspend-to-disk enabled='no'/> </pm>	이 VM에 대해 S3 및 S4 ACPI 절전 상태가 비활성화됩니다.

도메인 XML 섹션	설명
<devices> <emulator>/usr/libexec/qemu-kvm</emulator> <disk type='file' device='disk'> <driver name='qemu' type='qcow2'/> <source file='/var/lib/libvirt/images/Testguest.qcow2'/> <target dev='vda' bus='virtio'/> </disk> <disk type='file' device='cdrom'> <driver name='qemu' type='raw'/> <target dev='sdb' bus='sata'/> <readonly/> </disk>	VM은 에뮬레이션에 **/usr/libexec/qemu-kvm** 바이너리 파일을 사용하며 두 개의 디스크 장치가 연결되어 있습니다. 첫 번째 디스크는 호스트에 저장된 **/var/libvirt/images/Testguest.qcow2** 를 기반으로 가상화된 하드 드라이브이며 해당 논리적 장치 이름은 **vda** 로 설정됩니다. Windows 게스트에서는 **virtio** 대신 **sata** 버스를 사용하는 것이 좋습니다. 두 번째 디스크는 가상화된 CD-ROM 이며 해당 논리적 장치 이름은 **sdb** 로 설정됩니다.

도메인 XML 섹션	설명
<controller type='usb' index='0' model='qemu-xhci' ports='15'/> <controller type='sata' index='0'/> <controller type='pci' index='0' model='pcie-root'/> <controller type='pci' index='1' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='1' port='0x10'/> </controller> <controller type='pci' index='2' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='2' port='0x11'/> </controller> <controller type='pci' index='3' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='3' port='0x12'/> </controller> <controller type='pci' index='4' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='4' port='0x13'/> </controller> <controller type='pci' index='5' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='5' port='0x14'/> </controller> <controller type='pci' index='6' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='6' port='0x15'/> </controller> <controller type='pci' index='7' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='7' port='0x16'/> </controller> <controller type='virtio-serial' index='0'/>	VM은 단일 컨트롤러를 사용하여 USB 장치를 연결하고 PCI-Express (PCI-Express) 장치를 위한 루트 컨트롤러를 사용합니다. 또한 **virtio-serial** 컨트롤러를 사용하면 VM이 직렬 콘솔과 같은 다양한 방식으로 호스트와 상호 작용할 수 있습니다. 가상 장치에 대한 자세한 내용은 가상 장치 유형을 참조하십시오.
<interface type='network'> <mac address='52:54:00:65:29:21'/> <source network='default'/> <model type='virtio'/> </interface>	네트워크 인터페이스는 **기본** 가상 네트워크와 **virtio** 네트워크 장치 모델을 사용하는 VM에 설정됩니다. Windows 게스트에서는 **virtio** 대신 **e1000e** 모델을 사용하는 것이 좋습니다. 네트워크 인터페이스 구성에 대한 자세한 내용은 가상 머신 네트워크 성능 최적화를 참조하십시오.

도메인 XML 섹션	설명
<serial type='pty'> <target type='isa-serial' port='0'> <model name='isa-serial'/> </target> </serial> <console type='pty'> <target type='serial' port='0'/> </console> <channel type='unix'> <target type='virtio' name='org.qemu.guest_agent.0'/> <address type='virtio-serial' controller='0' bus='0' port='1'/> </channel>	**pty** 직렬 콘솔은 VM에 설정되어 호스트와의 기본 VM 통신을 활성화합니다. 콘솔은 포트 1에서 **UNIX** 채널을 사용합니다. 이 설정은 자동으로 설정되고 이러한 설정을 변경하는 것은 권장되지 않습니다. VM과의 상호 작용에 대한 자세한 내용은 웹 콘솔을 사용하여 가상 머신과 상호 작용을 참조하십시오.
<input type='tablet' bus='usb'> <address type='usb' bus='0' port='1'/> </input> <input type='mouse' bus='ps2'/> <input type='keyboard' bus='ps2'/>	VM은 태블릿 입력을 수신하도록 설정된 가상 **USB** 포트와 마우스 및 키보드 입력을 수신하도록 설정된 가상 **ps2** 포트를 사용합니다. 이 설정은 자동으로 설정되고 이러한 설정을 변경하는 것은 권장되지 않습니다.
<graphics type='vnc' port='-1' autoport='yes' listen='127.0.0.1'> <listen type='address' address='127.0.0.1'/> </graphics>	VM은 그래픽 출력을 렌더링하기 위해 **vnc** 프로토콜을 사용합니다.
<redirdev bus='usb' type='tcp'> <source mode='connect' host='localhost' service='4000'/> <protocol type='raw'/> </redirdev> <memballoon model='virtio'> <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/> </memballoon> </devices> </domain>	VM은 **TCP** 리디렉션을 사용하여 USB 장치를 원격으로 연결하고 메모리 증대가 켜집니다. 이 설정은 자동으로 설정되고 이러한 설정을 변경하는 것은 권장되지 않습니다.

10장. 가상 머신 저장 및 복원

시스템 리소스를 확보하기 위해 해당 시스템에서 실행 중인 VM(가상 머신)을 종료할 수 있습니다. 그러나 VM이 다시 필요한 경우 게스트 운영 체제(OS)를 부팅하고 애플리케이션을 다시 시작해야 합니다. 이 경우 상당한 시간이 걸릴 수 있습니다. 이 다운타임을 줄이고 VM 워크로드를 더 빨리 실행할 수 있도록 하려면 저장 및 복원 기능을 사용하여 OS 종료 및 부팅 시퀀스를 완전히 방지할 수 있습니다.

이 섹션에서는 VM 저장 및 완전한 VM 부팅 없이 동일한 상태로 복원하는 방법에 대한 정보를 제공합니다.

10.1. 가상 머신 저장 및 복원 방법

VM(가상 머신)을 저장하면 메모리와 장치 상태를 호스트의 디스크에 저장하고 VM 프로세스가 즉시 중지됩니다. 실행 중이거나 일시 중지된 상태인 VM을 저장할 수 있으며 복원 시 VM이 해당 상태로 돌아갑니다.

이 프로세스에서는 호스트 시스템의 RAM 및 CPU 리소스를 디스크 공간을 교환하여 호스트 시스템 성능을 향상시킬 수 있습니다. 게스트 OS를 부팅할 필요가 없기 때문에 VM을 복원할 때 긴 부팅 기간을 방지할 수 있습니다.

VM을 저장하려면 CLI(명령줄 인터페이스)를 사용할 수 있습니다. 자세한 내용은 명령줄 인터페이스를 사용하여 가상 시스템 탐색을 참조하십시오.

VM을 복원하려면 CLI 또는 웹 콘솔 GUI 를 사용할 수 있습니다.

10.2. 명령줄 인터페이스를 사용하여 가상 머신 저장

VM(가상 머신)과 해당 현재 상태를 호스트의 디스크에 저장할 수 있습니다. 예를 들어 호스트의 리소스를 다른 용도로 사용해야 하는 경우 유용합니다. 그런 다음 저장된 VM을 이전 실행 상태로 신속하게 복원할 수 있습니다.

명령줄을 사용하여 VM을 저장하려면 아래 절차를 따르십시오.

사전 요구 사항

- VM 및 해당 구성을 저장할 충분한 디스크 공간이 있는지 확인합니다. VM에 사용되는 공간은 해당 VM에 할당된 RAM 용량에 따라 다릅니다.

- VM이 지속되는지 확인합니다.

- 선택 사항: 필요한 경우 VM에서 중요한 데이터를 백업합니다.

절차

- virsh managedsave 유틸리티를 사용합니다.

  예를 들어 다음 명령은 *demo-guest1* VM을 중지하고 해당 구성을 저장합니다.

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

  저장된 VM 파일은 기본적으로 /var/lib/libvirt/qemu/save 디렉터리에 demo-guest1.save 로 있습니다.

  다음에 VM을 시작하면 위 파일에서 저장된 상태를 자동으로 복원합니다.

검증

- VM이 저장된 상태에 있는지 확인하거나 virsh list 유틸리티를 사용하여 종료합니다.

  저장이 활성화된 VM을 나열하려면 다음 명령을 사용합니다. *저장된* VM에 나열된 VM에는 관리 저장이 활성화됩니다.

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

  관리 저장소 이미지가 있는 VM을 나열하려면 다음을 수행합니다.

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

-

  종료된 상태의 저장된 VM을 나열하려면 명령에 **--all** 또는 **--inactive** 옵션을 사용해야 합니다.

문제 해결

-

  저장된 VM 파일이 손상되거나 읽을 수 없는 경우 VM을 복원하면 대신 표준 VM 부팅이 시작됩니다.

추가 리소스

-

  virsh managedsave --help 명령

-

  명령줄 인터페이스를 사용하여 저장된 VM 복원

-

  웹 콘솔을 사용하여 저장된 VM 복원

10.3. 명령줄 인터페이스를 사용하여 가상 머신 시작

CLI(명령줄 인터페이스)를 사용하여 종료한 VM(가상 머신)을 시작하거나 저장된 VM을 복원할 수 있습니다. CLI를 사용하면 로컬 및 원격 VM을 모두 시작할 수 있습니다.

사전 요구 사항

-

  이미 정의된 비활성 VM입니다.

-

  VM의 이름입니다.

-

  원격 VM의 경우:

  -

    VM이 있는 호스트의 IP 주소입니다.

  -

    호스트에 대한 루트 액세스 권한입니다.

절차

- 로컬 VM의 경우 **virsh start** 유틸리티를 사용합니다.

  예를 들어 다음 명령은 ***demo-guest1*** VM을 시작합니다.

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

- 원격 호스트에 있는 VM의 경우 호스트의 **QEMU+SSH** 연결과 함께 **virsh start** 유틸리티를 사용합니다.

  예를 들어 다음 명령은 **192.168.123.123** 호스트에서 ***demo-guest1*** VM을 시작합니다.

```
# virsh -c qemu+ssh://root@192.168.123.123/system start demo-guest1

root@192.168.123.123's password:

Domain 'demo-guest1' started
```

추가 리소스

- **virsh start --help** 명령
- 원격 가상화 호스트에 쉽게 액세스 가능
- 호스트가 시작될 때 자동으로 가상 머신 시작

10.4. 웹 콘솔을 사용하여 가상 머신 시작

VM(가상 머신)이 *종료된* 상태인 경우 **RHEL 9** 웹 콘솔을 사용하여 시작할 수 있습니다. 호스트가 시작될 때 VM이 자동으로 시작하도록 구성할 수도 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

- 이미 정의된 비활성 VM입니다.
- VM의 이름입니다.

절차

1. Virtual Machines (가상 시스템) 인터페이스에서 시작할 VM을 클릭합니다.

   선택한 VM에 대한 자세한 정보와 VM 종료 및 삭제를 위한 컨트롤이 포함된 새 페이지가 열립니다.

2. 실행 을 클릭합니다.

   VM이 시작되고 해당 콘솔 또는 그래픽 출력에 연결할 수 있습니다.

3. 선택 사항: 호스트가 시작될 때 자동으로 시작하도록 VM을 구성하려면 Autostart 확인란을 클릭합니다.

   libvirt에서 관리하지 않는 네트워크 인터페이스를 사용하는 경우 systemd 구성도 변경해야 합니다. 그렇지 않으면 영향을 받는 VM이 시작되지 못할 수 있습니다. 호스트가 시작될 때 자동으로 가상 시스템 시작을 참조하십시오.

추가 리소스

- 웹 콘솔에서 가상 머신 종료
- 웹 콘솔을 사용하여 가상 머신 재시작

11장. 가상 머신 복제

특정 속성 세트를 사용하여 새 VM(가상 머신)을 빠르게 생성하기 위해 기존 VM을 *복제* 할 수 있습니다.

복제는 스토리지에 자체 디스크 이미지를 사용하는 새 VM을 생성하지만 대부분의 복제본 구성과 저장된 데이터는 소스 VM과 동일합니다. 따라서 각 VM을 개별적으로 최적화하지 않고도 특정 작업에 최적화된 다수의 VM을 준비할 수 있습니다.

11.1. 가상 머신 복제의 작동 방식

VM(가상 머신)을 복제하면 소스 VM 및 디스크 이미지의 XML 구성이 복사되고 구성을 조정하여 새 VM의 고유성을 보장합니다. 여기에는 VM의 이름 변경 및 디스크 이미지 복제가 사용되는지 확인하는 작업이 포함됩니다. 그러나 복제본의 가상 디스크에 저장된 데이터는 소스 VM과 동일합니다.

이 프로세스는 새 VM을 생성하고 게스트 운영 체제로 설치하는 것보다 빠르며 특정 구성 및 콘텐츠를 사용하여 VM을 빠르게 생성하는 데 사용할 수 있습니다.

VM 복제본 여러 개를 생성하려는 경우 먼저 포함되지 않은 VM *템플릿* 을 생성합니다.

- 복제본이 올바르게 작동하지 못하도록 하는 영구적인 네트워크 MAC 구성과 같은 고유한 설정입니다.
- SSH 키 및 암호 파일과 같은 민감한 데이터입니다.

자세한 내용은 가상 머신 템플릿 생성을 참조하십시오.

추가 리소스

- 명령줄 인터페이스를 사용하여 가상 머신 복제
- 웹 콘솔을 사용하여 가상 머신 복제

11.2. 가상 머신 템플릿 생성

여러 VM(가상 머신)에서 올바르게 작동하는 복제본을 생성하려면 SSH 키 또는 영구 네트워크 MAC 구성과 같이 소스 VM에 고유한 정보와 구성을 제거할 수 있습니다. 그러면 VM 복제를 쉽고 안전하게 생성하는 데 사용할 수 있는 VM 템플릿이 생성됩니다.

virt-sysprep 유틸리티를 사용하여 VM 템플릿 을 생성하거나 요구 사항에 따라 수동으로 생성할 수 있습니다.

11.2.1. virt-sysprep을 사용하여 가상 머신 템플릿 생성

기존 VM(가상 머신)에서 템플릿을 생성하려면 virt-sysprep 유틸리티를 사용하여 게스트 VM을 신속하게 구성 해제하여 복제를 준비할 수 있습니다. virt-sysprep 유틸리티는 복제에 복사해서는 안 되는 VM에서 특정 구성을 자동으로 제거하여 템플릿을 생성합니다.

사전 요구 사항

- *virt-sysprep 유틸리티가 호스트에 설치되어 있습니다.*

```
# dnf install /usr/bin/virt-sysprep
```

- *템플릿으로 사용할 VM이 종료됩니다.*

- *소스 VM의 디스크 이미지가 있는 위치와 VM 디스크 이미지 파일의 소유자여야 합니다.*

  libvirt의 시스템 연결에 생성된 VM의 디스크 이미지는 기본적으로 /var/lib/libvirt/images 디렉터리에 있으며 root 사용자가 소유합니다.

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

- *선택 사항: VM 디스크의 중요한 데이터가 백업되었습니다. 소스 VM을 그대로 유지하려면 먼저 복제하고 복제본 을 편집하여 템플릿을 생성합니다.*

절차

1. *VM 디스크 이미지의 소유자로 로그인했는지 확인합니다.*

```
# whoami
root
```

2. *선택 사항: VM의 디스크 이미지를 복사합니다.*

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

나중에 VM이 템플릿으로 변환되었는지 확인하는 데 사용됩니다.

3. *다음 명령을 사용하고 /var/lib/libvirt/images/a-really-important-vm.qcow2 를 소스 VM의 디스크 이미지 경로로 바꿉니다.*

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

검증

- *프로세스가 성공했는지 확인하려면 수정된 디스크 이미지를 원래 디스크 이미지와 비교합니다. 다음 예제에서는 성공적으로 템플릿을 생성하는 방법을 보여줍니다.*

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

추가 리소스

- *virt-sysprep 도움말 페이지의 OPERATIONS 섹션*

- 명령줄 인터페이스를 사용하여 가상 머신 복제

11.2.2. 수동으로 가상 머신 템플릿 생성

기존 VM(가상 머신)에서 템플릿을 생성하려면 복제를 위해 게스트 VM을 수동으로 재설정하거나 구성 해제하면 됩니다.

사전 요구 사항

- 소스 VM의 디스크 이미지 위치를 알고 있으며 VM 디스크 이미지 파일의 소유자입니다.

  libvirt의 시스템 연결에 생성된 VM의 디스크 이미지는 기본적으로 /var/lib/libvirt/images 디렉터리에 있으며 root 사용자가 소유합니다.

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

- VM이 종료되었는지 확인합니다.

- 선택 사항: VM 디스크의 중요한 데이터가 백업되었습니다. 소스 VM을 그대로 유지하려면 먼저 복제하고 복제본 을 편집하여 템플릿을 생성합니다.

절차

1. 복제를 위해 VM을 구성합니다.

   a. 복제본에 필요한 소프트웨어를 설치합니다.

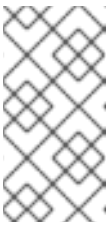
   b. 운영 체제에 대한 고유하지 않은 설정을 구성합니다.

   c. 고유하지 않은 애플리케이션 설정을 구성합니다.

2. *네트워크 구성을 제거합니다.*

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** 이름이 **eth0** 대신 **eth1** 일 수 있습니다.*

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

i. ***HWADDR** 및 정적 줄을 제거합니다.*

참고

***HWADDR**이 새 게스트의 **MAC** 주소와 일치하지 않으면 **ifcfg** 가 무시됩니다.*

```
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**를 구성하지만 **HWADDR** 또는 기타 고유 정보는 포함하지 마십시오.*

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

c. *다음 파일에도 시스템에 있는 경우 동일한 콘텐츠가 포함되어 있는지 확인합니다.*

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

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

참고

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

3. *등록 세부 정보 제거:*

- *RHN(Red Hat Network)에 등록된 VM의 경우:*

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

- *RHSM(Red Hat Subscription Manager)에 등록된 VM의 경우:*
  - *원래 VM을 사용하지 않는 경우 다음을 수행합니다.*

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

  - *원래 VM을 사용하려는 경우:*

```
# subscription-manager clean
```

참고

원래 RHSM 프로필은 ID 코드와 함께 포털에 남아 있습니다. 다음 명령을 사용하여 복제된 후 VM에서 RHSM 등록을 다시 활성화합니다.

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

4. *다른 고유한 세부 정보 제거:*

a.

ssh 공개/개인 키 쌍 제거:

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

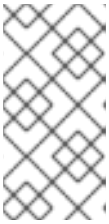
b.

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

5.

gnome-initial-setup-done 파일을 제거하여 다음 부팅 시 구성 마법사를 실행하도록 **VM**을 구성합니다.

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



참고

다음 부팅 시 실행되는 마법사는 **VM**에서 제거된 구성에 따라 다릅니다. 또한 복제본의 첫 번째 부팅 시 호스트 이름을 변경하는 것이 좋습니다.

11.3. 명령줄 인터페이스를 사용하여 가상 머신 복제

테스트 목적으로 특정 속성 세트를 사용하여 새 **VM**(가상 머신)을 빠르게 생성하려면 기존 **VM**을 복제할 수 있습니다. **CLI**를 사용하여 이를 수행하려면 아래 지침을 따르십시오.

사전 요구 사항

- 소스 **VM**이 종료되었습니다.
- 복제된 디스크 이미지를 저장할 충분한 디스크 공간이 있는지 확인합니다.
- 선택 사항: 여러 **VM** 복제본을 생성할 때 소스 **VM**에서 고유한 데이터 및 설정을 제거하여 복제된 **VM**이 제대로 작동하는지 확인합니다. 자세한 내용은 가상 머신 템플릿 생성을 참조하십시오.

절차

1.

환경과 사용 사례에 적합한 옵션과 함께 **virt-clone** 유틸리티를 사용합니다.

사용 사례 샘플

- 다음 명령은 **doppelganger** 라는 로컬 VM을 복제하고 **doppelganger-clone** VM을 생성합니다. 또한 원래 VM의 디스크 이미지와 동일한 위치에 **doppelganger-clone.qcow2** 디스크 이미지를 생성하고 동일한 데이터를 사용합니다.

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

Clone 'doppelganger-clone' created successfully.
```

- 다음 명령은 **geminus1** 이라는 VM을 복제하고 **geminus2** 라는 로컬 VM을 생성합니다. 이는 **geminus1**의 두 개의 **geminus1** 디스크만 사용합니다.

```
# virt-clone --original geminus1 --name geminus2 --file
/var/lib/libvirt/images/disk1.qcow2 --file /var/lib/libvirt/images/disk2.qcow2
Allocating 'disk1-clone.qcow2'                           | 78.0 GB  00:05:37
Allocating 'disk2-clone.qcow2'                           | 80.0 GB  00:05:37

Clone 'geminus2' created successfully.
```

- VM을 다른 호스트로 복제하려면 로컬 호스트에서 태그를 정의하지 않고 VM을 마이그레이션합니다. 예를 들어, 다음 명령은 로컬 디스크를 포함하여 이전에 생성된 **geminus2** VM을 **10.0.0.1** 원격 시스템에 복제합니다. 이러한 명령을 사용하려면 **10.0.0.1**에 대한 **root** 권한도 필요합니다.

```
# virsh migrate --offline --persistent geminus2 qemu+ssh://root@10.0.0.1/system
root@10.0.0.1's password:

# scp /var/lib/libvirt/images/disk1-clone.qcow2
root@10.0.0.1/user@remote_host.com://var/lib/libvirt/images/

# scp /var/lib/libvirt/images/disk2-clone.qcow2
root@10.0.0.1/user@remote_host.com://var/lib/libvirt/images/
```

검증

VM이 성공적으로 복제되었으며 올바르게 작동하는지 확인하려면 다음을 수행하십시오.

1. 호스트의 VM 목록에 복제본이 추가되었는지 확인합니다.

```
# virsh list --all
Id   Name                State
```

```
 ----------------------------------
 -   doppelganger         shut off
 -   doppelganger-clone   shut off
```

2. 복제를 시작하고 부팅되는지 관찰합니다.

```
# virsh start doppelganger-clone
Domain 'doppelganger-clone' started
```

추가 리소스

- **virt-clone** 도움말 페이지
- 가상 머신 마이그레이션

11.4. 웹 콘솔을 사용하여 가상 머신 복제

특정 속성 세트로 새 VM(가상 머신)을 빠르게 생성하기 위해 이전에 구성한 VM을 복제할 수 있습니다. 다음 지침은 웹 콘솔을 사용하여 이를 수행하는 방법을 설명합니다.

참고

VM 복제는 해당 VM과 연결된 디스크도 복제합니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 복제하려는 VM이 종료되었는지 확인합니다.

절차

1. 웹 콘솔의 **Virtual Machines** 인터페이스에서 복제할 VM 의 메뉴 버튼을 클릭합니다.

   다양한 VM 작업에 대한 제어와 함께 드롭다운 메뉴가 표시됩니다.

2. ***Clone (복제)을 클릭합니다.***

   복제 VM 생성 대화 상자가 표시됩니다.

3. ***선택 사항: VM 복제본의 새 이름을 입력합니다.***

4. ***Clone (복제)을 클릭합니다.***

   소스 VM을 기반으로 새 VM이 생성됩니다.

검증

- ***복제된 VM이 호스트에서 사용 가능한 VM 목록에 표시되는지 확인합니다.***

12장. 가상 머신 마이그레이션

VM(가상 머신)의 현재 호스트가 적합하지 않거나 더 이상 사용할 수 없거나 호스팅 워크로드를 재배포하려는 경우 VM을 다른 KVM 호스트로 마이그레이션할 수 있습니다.

12.1. 가상 머신 마이그레이션 작동 방식

VM(가상 머신) 마이그레이션의 필수 부분은 VM의 XML 구성을 다른 호스트 머신에 복사하는 것입니다. 마이그레이션된 VM이 종료되지 않으면 마이그레이션에서 VM 메모리 및 모든 가상화 장치의 상태도 대상 호스트 시스템으로 전송합니다. VM이 대상 호스트에서 계속 작동하려면 VM의 디스크 이미지를 사용할 수 있어야 합니다.

기본적으로 마이그레이션된 VM은 대상 호스트에서 일시적이며 소스 호스트에서도 정의됩니다.

실시간 또는 비 라이브 마이그레이션을 사용하여 실행 중인 VM을 마이그레이션할 수 있습니다. 종료 VM을 마이그레이션하려면 오프라인 마이그레이션을 사용해야 합니다. 자세한 내용은 다음 표를 참조하십시오.

표 12.1. VM 마이그레이션 유형

마이그레이션 유형	설명	사용 사례	스토리지 요구사항
실시간 마이그레이션	KVM이 VM의 메모리 페이지를 대상 호스트로 전송하는 동안 VM은 소스 호스트 머신에서 계속 실행됩니다. 마이그레이션이 거의 완료되면 KVM은 VM을 잠시 중단하고 대상 호스트에서 다시 시작합니다.	지속적인 가동 시간이 필요한 VM에 유용합니다. 그러나 KVM보다 메모리 페이지를 수정하는 VM은 I/O 로드가 많은 VM과 같이 실시간 마이그레이션할 수 없으며 실시간이 *아닌 마이그레이션*을 대신 사용해야 합니다.	VM의 디스크 이미지는 소스 호스트와 대상 호스트에 모두 액세스할 수 있는 공유 네트워크 에 있어야 합니다.
실시간이 아닌 마이그레이션	VM을 일시 중지하고 구성 및 해당 메모리를 대상 호스트에 복사하고 VM을 다시 시작합니다.	VM에 대한 다운타임을 생성하지만 일반적으로 실시간 마이그레이션보다 안정적입니다. I/O 로드가 많은 VM에 권장됩니다.	VM의 디스크 이미지는 소스 호스트와 대상 호스트에 모두 액세스할 수 있는 공유 네트워크 에 있어야 합니다.
오프라인 마이그레이션	VM의 구성을 대상 호스트로 이동합니다.	종료 VM에 사용하는 것이 좋습니다.	VM의 디스크 이미지는 공유 네트워크에서 사용할 수 없으며 대상 호스트로 수동으로 복사하거나 이동할 수 있습니다.

추가 리소스

- 가상 머신 마이그레이션의 이점
- 다른 호스트와 가상 머신 디스크 이미지 공유

12.2. 가상 머신 마이그레이션의 이점

VM(가상 머신) 마이그레이션은 다음에 유용할 수 있습니다.

로드 밸런싱

호스트가 과부하되는 경우 또는 다른 호스트가 활용도가 낮은 경우 VM을 호스트 시스템으로 이동할 수 있습니다.

하드웨어 독립성

호스트 시스템에서 하드웨어 장치를 업그레이드, 추가 또는 제거해야 하는 경우 VM을 다른 호스트에 안전하게 재배치할 수 있습니다. 따라서 VM은 하드웨어 개선 시 다운타임이 발생하지 않습니다.

더 많은 비용 절감

VM은 다른 호스트에 재배포할 수 있으며 로드되지 않은 호스트 시스템의 전원을 꺼 사용 기간이 짧은 동안 전력을 절약하고 비용을 절감할 수 있습니다.

지리적 마이그레이션

대기 시간이 적거나 다른 이유로 필요한 경우 VM을 다른 물리적 위치로 이동할 수 있습니다.

12.3. 가상 머신 마이그레이션에 대한 제한 사항

RHEL 9에서 VM(가상 머신)을 마이그레이션하기 전에 마이그레이션의 제한 사항을 알고 있어야 합니다.

- libvirt의 세션 연결 또는 세션 연결 로 VM을 마이그레이션하는 것은 신뢰할 수 없으므로 권장되지 않는 것이 좋습니다.
- 마이그레이션된 경우 특정 기능 및 구성을 사용하는 VM이 제대로 작동하지 않거나 마이그레이션이 실패합니다. 이러한 기능은 다음과 같습니다.

- *장치 패스스루*
- ***SR-IOV** 장치 할당*
- ***vGPU**와 같은 중재 장치*

- ***NUMA(Non-Uniform Memory Access)** 고정 고정을 사용하는 호스트 간의 마이그레이션은 호스트에 유사한 토폴로지가 있는 경우에만 작동합니다. 그러나 워크로드 실행에 대한 성능은 마이그레이션의 영향을 받을 수 있습니다.*

- *소스 **VM**과 대상 **VM**의 에뮬레이션된 **CPU**는 동일해야 합니다. 그렇지 않으면 마이그레이션이 실패할 수 있습니다. 다음 **CPU** 관련 영역에 있는 **VM** 간의 차이로 인해 마이그레이션에 문제가 발생할 수 있습니다.*

  - ***CPU** 모델*

    참고

    ***x86-64** 명령을 공유하더라도 **Intel 64** 호스트와 **AMD64** 호스트 간에 마이그레이션하는 것은 지원되지 않습니다.*

  - *펌웨어 설정*
  - *마이크로 코드 버전*
  - ***BIOS** 버전*
  - ***BIOS** 설정*
  - ***QEMU** 버전*

- 커널 버전

- 1TB 이상의 메모리를 사용하는 VM을 실시간 마이그레이션하는 경우 안정적으로 작동하지 않을 수 있습니다. 이러한 마이그레이션의 안정성은 다음에 따라 다릅니다.

  - VM의 현재 워크로드

  - 호스트가 마이그레이션에 사용할 수 있는 네트워크 대역폭

  - 배포에서 지원할 수 있는 다운 타임

  1TB 이상의 메모리가 있는 VM과 관련된 실시간 마이그레이션 시나리오는 Red Hat에 문의해야 합니다.

12.4. 다른 호스트와 가상 머신 디스크 이미지 공유

지원되는 KVM 호스트 간에 VM(가상 머신) 실시간 마이그레이션을 수행하려면 공유 VM 스토리지가 필요합니다. 이 섹션에서는 NFS 프로토콜을 사용하여 로컬에 저장된 VM 이미지를 소스 호스트 및 대상 호스트와 공유하는 지침을 제공합니다.

사전 요구 사항

- 마이그레이션을 위한 VM이 종료됩니다.

- 선택 사항: 호스트 시스템은 소스 또는 대상 호스트가 아닌 스토리지를 호스팅하는 데 사용할 수 있지만 소스와 대상 호스트 모두 네트워크를 통해 도달할 수 있습니다. 이는 공유 스토리지에 가장 적합한 솔루션이며 Red Hat에서 권장합니다.

- KVM에서 지원되지 않으므로 NFS 파일 잠금이 사용되지 않는지 확인합니다.

- NFS는 소스 및 대상 호스트에 설치 및 활성화됩니다. 그렇지 않은 경우 다음을 수행합니다.

a. *NFS 패키지를 설치합니다.*

```
# dnf install nfs-utils
```

b. *NFS(2049)의 포트가 방화벽에서 열려 있는지 확인합니다.*

```
# firewall-cmd --permanent --add-service=nfs
# firewall-cmd --permanent --add-service=mountd
# firewall-cmd --permanent --add-service=rpc-bind
# firewall-cmd --permanent --add-port=2049/tcp
# firewall-cmd --permanent --add-port=2049/udp
# firewall-cmd --reload
```

c. *NFS 서비스를 시작합니다.*

```
# systemctl start nfs-server
```

절차

1. *공유 스토리지를 제공할 호스트에 연결합니다. 이 예에서는 loads -bay 호스트입니다.*

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

2. *디스크 이미지를 보유하고 마이그레이션 호스트와 공유할 디렉터리를 생성합니다.*

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

3. *소스 호스트에서 새로 생성된 디렉터리로 VM의 디스크 이미지를 복사합니다. 예를 들어, 다음에서는 wanderer1 VM의 디스크 이미지를'cargo-bay 호스트의 /var/lib/libvirt/shared-images/ 디렉터리에 복사합니다.*

```
# scp /var/lib/libvirt/images/wanderer1.qcow2 root@cargo-bay:/var/lib/libvirt/shared-images/wanderer1.qcow2
```

4. *스토리지를 공유하는 데 사용할 호스트에서 공유 디렉토리를 /etc/exports 파일에 추가합니다. 다음 예제에서는 /var/lib/libvirt/shared-images 디렉터리를 source-example 및 dest-*

example 호스트와 공유합니다.

```
/var/lib/libvirt/shared-images source-example(rw,no_root_squash) dest-example(rw,no_root_squash)
```

5. *소스 및 대상 호스트 모두에서 /var/lib/libvirt/images 디렉터리에 공유 디렉터리를 마운트합니다.*

```
# mount cargo-bay:/var/lib/libvirt/shared-images /var/lib/libvirt/images
```

검증

- *프로세스가 성공했는지 확인하려면 소스 호스트에서 VM을 시작하고 올바르게 부팅되는지 확인합니다.*

12.5. 명령줄 인터페이스를 사용하여 가상 머신 마이그레이션

VM(가상 머신)의 현재 호스트가 적합하지 않거나 더 이상 사용할 수 없거나 호스팅 워크로드를 재배포하려는 경우 VM을 다른 KVM 호스트로 마이그레이션할 수 있습니다. 이 섹션에서는 이러한 마이그레이션의 다양한 시나리오에 대한 지침 및 예제를 제공합니다.

사전 요구 사항

- *소스 호스트와 대상 호스트는 모두 KVM 하이퍼바이저를 사용합니다.*
- *소스 호스트와 대상 호스트는 네트워크를 통해 서로 연결할 수 있습니다. ping 유틸리티를 사용하여 이를 확인합니다.*
- *대상 호스트에서 다음 포트가 열려 있는지 확인합니다.*
  - *SSH를 사용하여 대상 호스트에 연결하는 데 포트 22가 필요합니다.*
  - *TLS를 사용하여 대상 호스트에 연결하려면 포트 16509가 필요합니다.*
  - *TCP를 사용하여 대상 호스트에 연결하려면 포트 16514가 필요합니다.*

- 메모리 및 디스크 마이그레이션 데이터를 전송하려면 **QEMU**에 **49152-49215** 포트가 필요합니다.

- **Red Hat**에서 마이그레이션을 지원하려면 소스 호스트 및 대상 호스트에서 특정 운영 체제 및 시스템 유형을 사용해야 합니다. 이 문제가 발생하지 않도록 하려면 *가상 머신 마이그레이션에 대한 지원 호스트를* 참조하십시오.

- 마이그레이션할 **VM**의 디스크 이미지는 소스 호스트와 대상 호스트에 모두 액세스할 수 있는 별도의 네트워크 위치에 있습니다. 이는 오프라인 마이그레이션의 경우 선택 사항이지만 실행 중인 **VM**을 마이그레이션하는 데 필요합니다.

  이러한 공유 **VM** 스토리지를 설정하는 방법은 *다른 호스트와 가상 머신 디스크 이미지 공유를* 참조하십시오.

- 실행 중인 **VM**을 마이그레이션할 때 **VM**이 더티 메모리 페이지를 생성하는 속도보다 네트워크 대역폭이 커야 합니다.

  실시간 마이그레이션을 시작하기 전에 **VM**의 더티 페이지 속도를 얻으려면 다음을 수행하십시오.

  a. 짧은 기간 동안 **VM**의 더티 페이지 생성 속도를 모니터링합니다.

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

  b. 모니터링이 완료되면 결과를 가져옵니다.

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

  이 예에서 **VM**은 초당 **2MB**의 더티 메모리 페이지를 생성하고 있습니다. 대역폭이 **2MB/s** 이하인 네트워크에서 **VM**을 실시간 마이그레이션하려고 하면 **VM**을 일시 중지하거나 워크로드를 낮추지 않으면 실시간 마이그레이션이 진행되지 않습니다.

  실시간 마이그레이션이 성공적으로 완료되도록 **Red Hat**은 네트워크 대역폭이 **VM**의 더티 페이지 생성 속도보다 큰 영향을 미칠 것을 권장합니다.

- 공용 브리지 탭 네트워크에서 기존 VM을 마이그레이션하는 경우 소스 및 대상 호스트가 동일한 네트워크에 있어야 합니다. 그렇지 않으면 마이그레이션 후 VM 네트워크가 작동하지 않습니다.

- VM 마이그레이션을 수행할 때 소스 호스트의 virsh 클라이언트는 여러 프로토콜 중 하나를 사용하여 대상 호스트의 libvirt 데몬에 연결할 수 있습니다. 다음 절차의 예제에서는 SSH 연결을 사용하지만 다른 연결을 선택할 수 있습니다.

  - libvirt에서 SSH 연결을 사용하도록 하려면 대상 호스트에서 virtqemud 소켓이 활성화되어 실행 중인지 확인합니다.

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

  - libvirt에서 TLS 연결을 사용하도록 하려면 대상 호스트에서 virtproxyd-tls 소켓이 활성화되어 실행 중인지 확인합니다.

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

  - libvirt에서 TCP 연결을 사용하도록 하려면 대상 호스트에서 virtproxyd-tcp 소켓이 활성화되어 실행 중인지 확인합니다.

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

절차

1. 마이그레이션 요구 사항에 적합한 옵션과 함께 virsh migrate 명령을 사용합니다.

   - 다음에서는 SSH 터널을 사용하여 로컬 호스트에서 dest-example 호스트의 시스템 연결로 wanderer1 VM을 마이그레이션합니다. 마이그레이션 중에 VM이 계속 실행됩니다.

     ```
     # virsh migrate --persistent --live wanderer1 qemu+ssh://dest-example/system
     ```

   - 다음은 로컬 호스트에서 실행 중인 wanderer2 VM의 구성을 수동으로 조정한 다음 VM을 dest-example 호스트로 마이그레이션할 수 있습니다. 마이그레이션된 VM은 업데이트된 구성을 자동으로 사용합니다.

     ```
     # virsh dumpxml --migratable wanderer2 >wanderer2.xml
     ```

```
# vi wanderer2.xml
# virsh migrate --live --persistent --xml wanderer2.xml wanderer2 qemu+ssh://dest-example/system
```

다음 절차는 대상 호스트에서 다른 경로를 사용하여 공유 VM 스토리지에 액세스하거나 대상 호스트와 관련된 기능을 구성할 때 유용할 수 있습니다.

- 다음은 source-example 호스트에서 wander 3 VM을 중단하고 dest-example 호스트로 마이그레이션한 다음 wander 3-alt.xml 파일에서 제공하는 조정된 XML 구성을 사용하도록 지시합니다. 마이그레이션이 완료되면 libvirt 가 대상 호스트에서 VM을 다시 시작합니다.

```
# virsh migrate wanderer3 qemu+ssh://source-example/system qemu+ssh://dest-example/system --xml wanderer3-alt.xml
```

마이그레이션 후 VM은 소스 호스트의 종료 상태가 되고 마이그레이션된 복사는 종료된 후 삭제됩니다.

- 다음은 source-example 호스트에서 종료 종료 wanderer4 VM을 삭제하고 해당 구성을 dest-example 호스트로 이동합니다.

```
# virsh migrate --offline --persistent --undefinesource wanderer4 qemu+ssh://source-example/system qemu+ssh://dest-example/system
```

이러한 유형의 마이그레이션에서는 VM의 디스크 이미지를 공유 스토리지로 이동할 필요가 없습니다. 그러나 대상 호스트에서 VM을 사용할 수 있으려면 VM의 디스크 이미지도 마이그레이션해야 합니다. 예를 들면 다음과 같습니다.

```
# scp root@source-example:/var/lib/libvirt/images/wanderer4.qcow2 root@dest-example:/var/lib/libvirt/images/wanderer4.qcow2
```

2. 마이그레이션이 완료될 때까지 기다립니다. 이 프로세스는 네트워크 대역폭, 시스템 로드 및 VM 크기에 따라 다소 시간이 걸릴 수 있습니다. virsh migrate 에 --verbose 옵션을 사용하지 않으면 CLI에서 오류를 제외한 진행률 지표를 표시하지 않습니다.

마이그레이션이 진행 중인 경우 virsh domjobinfo 유틸리티를 사용하여 마이그레이션 통계를 표시할 수 있습니다.

검증

- 대상 호스트에서 사용 가능한 VM을 나열하여 VM이 마이그레이션되었는지 확인합니다.

```
# virsh list
Id Name                 State
----------------------------------
10 wanderer1            running
```

마이그레이션이 계속 실행 중인 경우 이 명령은 VM 상태를 일시 중지됨 으로 나열합니다.

문제 해결

- 경우에 따라 대상 호스트는 마이그레이션된 VM의 XML 구성(예: 네트워크 이름 또는 CPU 유형)의 특정 값과 호환되지 않습니다. 결과적으로 VM이 대상 호스트에서 부팅되지 않습니다. 이러한 문제를 해결하려면 virsh edit 명령을 사용하여 문제가 있는 값을 업데이트할 수 있습니다. 값을 업데이트한 후 변경 사항을 적용하려면 VM을 다시 시작해야 합니다.

- 실시간 마이그레이션이 완료하는 데 시간이 오래 걸리는 경우 VM이 로드되어 있고 실시간 마이그레이션이 가능한 메모리 페이지가 너무 많기 때문일 수 있습니다. 이 문제를 해결하려면 VM을 일시 중지하여 라이브가 아닌로의 마이그레이션을 변경합니다.

```
# virsh suspend wanderer1
```

추가 리소스

- virsh migrate --help 명령

- virsh man 페이지

12.6. 웹 콘솔을 사용하여 가상 머신 실시간 마이그레이션

지속적으로 실행 중이어야 하는 작업을 수행하는 VM(가상 머신)을 마이그레이션하려면 해당 VM을 종료하지 않고 다른 KVM 호스트로 마이그레이션할 수 있습니다. 이를 실시간 마이그레이션이라고도 합니다. 다음 지침은 웹 콘솔을 사용하여 이를 수행하는 방법을 설명합니다.

주의

KVM이 많은 I/O 로드 작업과 같이 메모리 페이지를 더 빠르게 수정하는 작업의 경우 VM을 실시간 마이그레이션하지 않는 것이 좋습니다.

사전 요구 사항

- *웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.*
- *소스 및 대상 호스트가 실행 중입니다.*
- *대상 호스트에서 다음 포트가 열려 있는지 확인합니다.*
  - *SSH를 사용하여 대상 호스트에 연결하는 데 포트 22가 필요합니다.*
  - *TLS를 사용하여 대상 호스트에 연결하려면 포트 16509가 필요합니다.*
  - *TCP를 사용하여 대상 호스트에 연결하려면 포트 16514가 필요합니다.*
  - *메모리 및 디스크 마이그레이션 데이터를 전송하려면 QEMU에 49152-49215 포트가 필요합니다.*
- *VM의 디스크 이미지는 소스 호스트 및 대상 호스트에서 액세스할 수 있는 공유 스토리지에 있습니다.*
- *실행 중인 VM을 마이그레이션할 때 VM이 더티 메모리 페이지를 생성하는 속도보다 네트워크 대역폭이 커야 합니다.*

  실시간 마이그레이션을 시작하기 전에 VM의 더티 페이지 속도를 얻으려면 명령줄 인터페이스에서 다음을 수행하십시오.

a.

짧은 기간 동안 VM의 더티 페이지 생성 속도를 모니터링합니다.

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

b.

모니터링이 완료되면 결과를 가져옵니다.

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

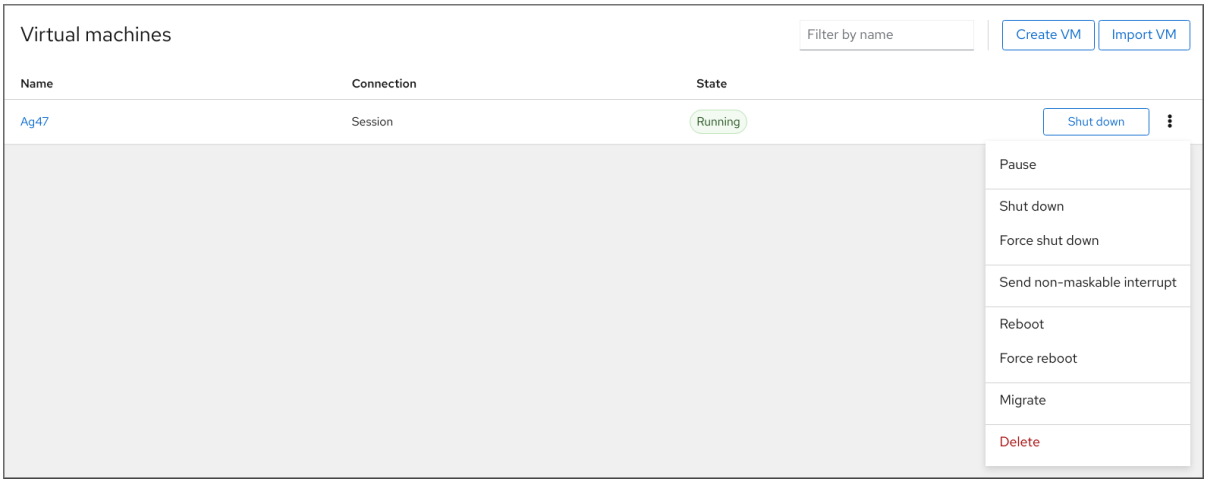
이 예에서 VM은 초당 2MB의 더티 메모리 페이지를 생성하고 있습니다. 대역폭이 2MB/s 이하인 네트워크에서 VM을 실시간 마이그레이션하려고 하면 VM을 일시 중지하거나 워크로드를 낮추지 않으면 실시간 마이그레이션이 진행되지 않습니다.

실시간 마이그레이션이 성공적으로 완료되도록 Red Hat은 네트워크 대역폭이 VM의 더티 페이지 생성 속도보다 큰 영향을 미칠 것을 권장합니다.

절차

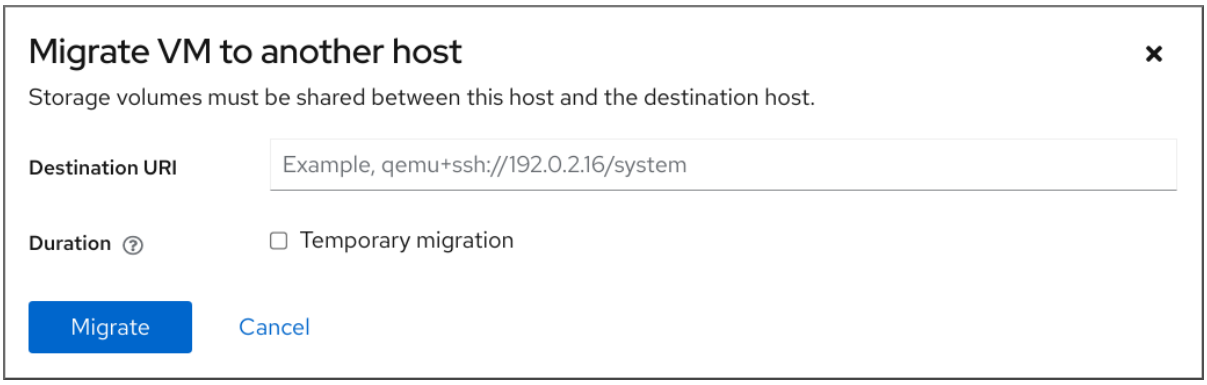
1. 웹 콘솔의 Virtual Machines 인터페이스에서 마이그레이션할 VM 의 메뉴 버튼을 클릭합니다.

다양한 VM 작업에 대한 제어와 함께 드롭다운 메뉴가 표시됩니다.



2. 마이그레이션을 클릭합니다.

Migrate VM to another host(VM 마이그레이션) 대화 상자가 나타납니다.



3. *대상 호스트의 URI를 입력합니다.*

4. *마이그레이션 기간을 구성합니다.*

   - *permanent - VM을 영구적으로 마이그레이션하려는 경우 상자를 선택하지 마십시오. 영구 마이그레이션은 소스 호스트에서 VM 구성을 완전히 제거합니다.*

   - *임시 - 임시 마이그레이션은 VM의 사본을 대상 호스트로 마이그레이션합니다. 이 사본은 VM이 종료되면 대상 호스트에서 삭제됩니다. 원래 VM은 소스 호스트에 남아 있습니다.*

5. *마이그레이션을 클릭합니다.*

   VM이 대상 호스트로 마이그레이션됩니다.

검증

VM이 성공적으로 마이그레이션되었는지 확인하려면 다음을 수행하십시오.

- *VM이 대상 호스트에서 사용 가능한 VM 목록에 표시되는지 확인합니다.*

- *마이그레이션된 VM을 시작하고 부팅되는지 확인합니다.*

12.7. 가상 머신 마이그레이션에 지원되는 호스트

VM(가상 머신) 마이그레이션이 제대로 작동하고 Red Hat에서 지원을 받으려면 소스 및 대상 호스트는 특정 RHEL 버전 및 머신 유형이어야 합니다. 다음 표에서는 지원되는 VM 마이그레이션 경로를 보여줍니다.

표 12.2. 실시간 마이그레이션 호환성

마이그레이션 방법	릴리스 유형	향후 버전 예	지원 상태
forward	마이너 릴리스	9.0.1 → 9.1	지원되는 RHEL 9 시스템에서 시스템 유형 **q35**.
Backward	마이너 릴리스	9.1 → 9.0.1	지원되는 RHEL 9 시스템에서 시스템 유형 **q35**.

참고

지원 수준은 RHOSP 및 OpenShift Virtualization을 포함하여 Red Hat에서 제공하는 다른 가상화 솔루션에 따라 다릅니다.

13장. 가상 장치 관리

VM(가상 머신)의 기능, 기능 및 성능을 관리하는 가장 효과적인 방법 중 하나는 가상 장치를 조정하는 것입니다.

다음 섹션에서는 가상 장치가 무엇인지와 CLI 또는 웹 콘솔 을 사용하여 관리하는 방법에 대한 일반적인 개요 를 제공합니다.

13.1. 가상 장치 작동 방식

실제 머신과 마찬가지로 VM(가상 머신)에는 처리 전원, 메모리, 스토리지, 네트워킹 또는 그래픽과 같은 시스템에 기능을 제공하기 위해 특수 장치가 필요합니다. 물리적 시스템은 일반적으로 이러한 목적으로 하드웨어 장치를 사용합니다. 그러나 VM은 소프트웨어 구현으로 작동하므로 가상 장치라고 하는 장치의 소프트웨어 추상화를 사용해야 합니다.

기본 사항

VM에 연결된 가상 장치는 VM 을 생성할 때 구성할 수 있으며 기존 VM 에서 관리할 수도 있습니다. 일반적으로 VM이 종료된 경우에만 VM에서 가상 장치를 연결하거나 분리할 수 있지만 VM이 실행 중일 때 일부는 추가하거나 제거할 수 있습니다. 이 기능을 장치 핫 플러그라고 하며 핫 언플러그 라고 합니다.

새 VM을 생성할 때 libvirt 는 사용자가 별도로 지정하지 않는 한 기본 필수 가상 장치 세트를 자동으로 생성하고 구성합니다. 호스트 시스템 아키텍처 및 시스템 유형을 기반으로 하며 일반적으로 다음과 같습니다.

- CPU
- memory
- 키보드
- NIC(네트워크 인터페이스 컨트롤러)
- 다양한 장치 컨트롤러

- 비디오 카드
- 사운드 카드

VM을 생성한 후 가상 장치를 관리하려면 CLI(명령줄 인터페이스)를 사용합니다. 그러나 가상 스토리지 장치 및 NIC를 관리하기 위해 RHEL 9 웹 콘솔을 사용할 수도 있습니다.

성능 또는 유연성

일부 유형의 장치의 경우 RHEL 9는 여러 구현을 지원하며 성능과 유연성 간에 절충되는 경우가 많습니다.

예를 들어 가상 디스크에 사용되는 물리 스토리지는 qcow2 또는 raw 와 같은 다양한 형식으로 파일로 표시하고 다양한 컨트롤러를 사용하여 VM에 표시할 수 있습니다.

- 에뮬레이션된 컨트롤러
- virtio-scsi
- virtio-blk

virtio 장치는 가상화 목적으로 특별히 설계되었기 때문에 에뮬레이션된 컨트롤러는 virtio 컨트롤러보다 느립니다. 반면 에뮬레이션된 컨트롤러는 virtio 장치용 드라이버가 없는 운영 체제를 실행할 수 있습니다. 마찬가지로 virtio-scsi 는 SCSI 명령을 보다 완벽하게 지원하므로 많은 디스크를 VM에 연결할 수 있습니다. 마지막으로 virtio-blk 는 virtio-scsi 및 에뮬레이션된 컨트롤러보다 더 나은 성능을 제공하지만 보다 제한된 사용 사례도 있습니다. 예를 들어 virtio-blk 를 사용하는 경우 물리적 디스크를 LUN 장치로 VM에 연결할 수 없습니다.

가상 장치 유형에 대한 자세한 내용은 가상 장치 유형을 참조하십시오.

13.2. 가상 장치 유형

RHEL 9의 가상화는 가상 머신 (VM)에 연결할 수 있는 여러 가지 유형의 가상 장치를 표시할 수 있습니다.

에뮬레이션된 장치

에뮬레이션된 장치는 널리 사용되는 물리적 장치의 소프트웨어 구현입니다. 물리적 장치를 위해 설계된 드라이버는 에뮬레이션된 장치와도 호환됩니다. 따라서 에뮬레이션된 장치는 매우 유연하게 사용할 수 있습니다.

그러나 특정 유형의 하드웨어를 견고하게 에뮬레이션해야 하므로 에뮬레이션된 장치는 해당 물리적 장치 또는 더 최적화된 가상 장치에 비해 상당한 성능 손실이 발생할 수 있습니다.

다음과 같은 유형의 에뮬레이션 장치가 지원됩니다.

- *가상 **CPU(vCPU)** 및 다양한 **CPU** 모델을 선택할 수 있습니다. 에뮬레이션의 성능 영향은 호스트 **CPU**와 에뮬레이션된 **vCPU**의 차이점에 따라 크게 달라집니다.*
- ***PCI** 버스 컨트롤러와 같은 에뮬레이션된 시스템 구성 요소입니다.*
- ***SATA, SCSI** 또는 **IDE**와 같은 에뮬레이션된 스토리지 컨트롤러.*
- ***ICH9, ICH6** 또는 **AC97**과 같은 에뮬레이션된 음향 장치.*
- ***VGA** 카드와 같은 에뮬레이션된 그래픽 카드.*
- ***rtl8139**와 같은 에뮬레이션된 네트워크 장치.*

반가상화 장치

*반가상화는 가상 장치를 **VM**에 노출할 수 있는 빠르고 효율적인 방법을 제공합니다. 반가상화 장치는 **VM**에서 사용하도록 특별히 설계된 인터페이스를 노출하므로 장치 성능이 크게 향상됩니다. **RHEL 9**는 **virtio API**를 하이퍼바이저와 **VM** 간의 계층으로 사용하는 **VM**에 반가상화 장치를 제공합니다. 이 접근 방식의 단점은 게스트 운영 체제에서 특정 장치 드라이버가 필요하다는 것입니다.*

***I/O** 집약적 애플리케이션을 실행하는 경우 가능한 한 항상 **VM**에 에뮬레이션된 장치를 사용하지 않고 반가상화 장치를 사용하는 것이 좋습니다. 반가상화 장치는 **I/O** 대기 시간을 줄이고 **I/O** 처리량을 높이며 경우에 따라 베어 메탈 성능에 매우 근접합니다. 다른 반가상화 장치도 사용할 수 없는 **VM**에 기능을 추가합니다.*

다음과 같은 유형의 반가상화 장치가 지원됩니다.

- 반가상화 네트워크 장치(**virtio-net**).
- 반가상화 스토리지 컨트롤러:
  - **virtio-blk -** 블록 장치 에뮬레이션을 제공합니다.
  - **virtio-scsi -** 더 완전한 **SCSI** 에뮬레이션을 제공합니다.
- 반가상화된 시계입니다.
- 반가상화 직렬 장치(**virtio-serial**).
- **balloon** 장치(**virtio-balloon**)는 **VM**과 호스트 간에 메모리를 동적으로 배포하는 데 사용됩니다.
- 반가상화 임의 번호 생성기(**virtio-rng**).

물리적으로 공유되는 장치

특정 하드웨어 플랫폼을 사용하면 **VM**이 다양한 하드웨어 장치 및 구성 요소에 직접 액세스할 수 있습니다. 이 프로세스를 장치 할당 또는 패스스루(**passthrough**)라고 합니다.

이러한 방식으로 연결된 경우 물리적 장치의 일부 측면을 물리적 시스템과 마찬가지로 **VM**에서 직접 사용할 수 있습니다. 이는 **VM**에서 사용할 때 장치에 대해 우수한 성능을 제공합니다. 그러나 **VM**에 물리적으로 연결된 장치를 호스트에서 사용할 수 없게 되므로 마이그레이션할 수도 없습니다.

그러나 일부 장치는 여러 **VM**에서 공유할 수 있습니다. 예를 들어, 단일 물리적 장치는 특정 상황에서 여러 개의 중재 장치를 제공할 수 있으며, 이를 개별 **VM**에 할당할 수 있습니다.

다음과 같은 유형의 패스스루 장치가 지원됩니다.

- **USB, PCI 및 SCSI 패스스루 -** 게스트 소프트웨어에서 특정 기능을 사용할 수 있도록 일반 산업 표준 버스를 VM에 직접 노출합니다.

- **SR-IOV(Single-root I/O Virtualization) - PCI Express** 리소스의 하드웨어 적용 가능 격리를 활성화하는 사양입니다. 이렇게 하면 단일 물리 PCI 리소스를 가상 PCI 함수로 분할하는 것이 안전하고 효율적입니다. NIC(네트워크 인터페이스 카드)에 일반적으로 사용됩니다.

- **N_Port ID 가상화(NPIV) -** 단일 물리적 HBA(호스트 버스 어댑터)를 여러 가상 포트와 공유하는 Fibre Channel 기술입니다.

- **GPU 및 vGPUs -** 특정 종류의 그래픽 또는 컴퓨팅 워크로드에 대한 가속기입니다. 일부 GPU는 VM에 직접 연결할 수 있지만 특정 유형은 기본 물리적 하드웨어를 공유하는 가상 GPU(vGPU)를 생성하는 기능도 제공합니다.

13.3. CLI를 사용하여 가상 머신에 연결된 장치 관리

VM(가상 머신) 기능을 수정하려면 CLI(명령줄 인터페이스)를 사용하여 VM에 연결된 장치를 관리할 수 있습니다.

CLI를 사용하여 다음을 수행할 수 있습니다.

- 장치 연결
- 장치 수정
- 장치 제거

13.3.1. 가상 머신에 장치 연결

새 가상 장치를 연결하여 VM(가상 머신)에 특정 기능을 추가할 수 있습니다.

다음 절차에서는 CLI(명령줄 인터페이스)를 사용하여 가상 장치를 생성하고 VM(가상 머신)에 연결하는 방법을 설명합니다. 일부 장치는 RHEL 웹 콘솔을 사용하여 VM에 연결할 수도 있습니다.

예를 들어 새 가상 디스크 장치를 연결하여 VM의 스토리지 용량을 늘릴 수 있습니다. 이를 메모리 핫 플러그 라고도 합니다.

주의

VM에서 메모리 장치 제거(메모리 핫 플러그 해제 라고도 함)는 RHEL 9에서 지원되지 않으며 Red Hat은 사용이 좋지 않습니다.

사전 요구 사항

- VM에 연결하려는 장치에 필요한 옵션을 가져옵니다. 특정 장치에 사용 가능한 옵션을 보려면 virt-xml --device=? 명령을 사용하십시오. 예를 들면 다음과 같습니다.

```
# virt-xml --network=?
--network options:
[...]
address.unit
boot_order
clearxml
driver_name
[...]
```

절차

1. 장치를 VM에 연결하려면 장치 정의 및 필수 옵션을 포함하여 virt-xml --add-device 명령을 사용하십시오.

   - 예를 들어 다음 명령은 /var/lib/libvirt/images/ 디렉터리에 20GB newdisk qcow2 디스크 이미지를 생성하고 다음 VM 시작 시 실행 중인 testguest VM에 가상 디스크로 연결합니다.

     ```
     # virt-xml testguest --add-device --disk
     /var/lib/libvirt/images/newdisk.qcow2,format=qcow2,size=20
     Domain 'testguest' defined successfully.
     Changes will take effect after the domain is fully powered off.
     ```

   - 다음은 VM이 실행되는 동안 호스트의 버스 002에서 장치 004로 연결된 USB 플래시 드라이브를 testguest2 VM에 연결합니다.

```
# virt-xml testguest2 --add-device --update --hostdev 002.004
Device hotplug successful.
Domain 'testguest2' defined successfully.
```

USB를 정의하는 버스 장치 조합은 lsusb 명령을 사용하여 얻을 수 있습니다.

검증

장치가 추가되었는지 확인하려면 다음 중 하나를 수행합니다.

- virsh dumpxml 명령을 사용하여 장치의 XML 정의가 VM의 XML 구성의 < devices > 섹션에 추가되었는지 확인합니다.

  예를 들어 다음 출력에서는 testguest VM의 구성을 보여주고 002.004 USB 플래시 디스크 장치가 추가되었는지 확인합니다.

```
# virsh dumpxml testguest
[...]
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x4146'/>
    <product id='0x902e'/>
    <address bus='2' device='4'/>
  </source>
  <alias name='hostdev0'/>
  <address type='usb' bus='0' port='3'/>
</hostdev>
[...]
```

- VM을 실행하고 장치가 있고 제대로 작동하는지 테스트합니다.

추가 리소스

- man virt-xml 명령

13.3.2. 가상 머신에 연결된 장치 수정

연결된 가상 장치 구성을 편집하여 VM(가상 머신) 기능을 변경할 수 있습니다. 예를 들어 VM의 성능을 최적화하려면 가상 CPU 모델을 변경하여 호스트 CPU의 CPU와 더 잘 일치시킬 수 있습니다.

다음 절차에서는 CLI(명령줄 인터페이스)를 사용하여 가상 장치를 수정하는 일반적인 지침을 제공합

니다. 디스크 및 NIC와 같이 VM에 연결된 일부 장치도 RHEL 9 웹 콘솔을 사용하여 수정할 수 있습니다.

사전 요구 사항

- VM에 연결하려는 장치에 필요한 옵션을 가져옵니다. 특정 장치에 사용 가능한 옵션을 보려면 virt-xml --device=? 명령을 사용하십시오. 예를 들면 다음과 같습니다.

```
# virt-xml --network=?
--network options:
[...]
address.unit
boot_order
clearxml
driver_name
[...]
```

- 선택 사항: virsh dumpxml vm-name 을 사용하여 VM의 XML 구성을 백업하고 해당 출력을 파일에 전송합니다. 예를 들어, 다음에서는 Motoko VM의 구성을 motoko.xml 파일로 백업합니다.

```
# virsh dumpxml Motoko > motoko.xml
# cat motoko.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>Motoko</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

절차

1. 장치 정의 및 필수 옵션을 포함하여 virt-xml --edit 명령을 사용합니다.

   예를 들어 다음에서는 종료 testguest VM의 < cpu > 구성을 지우고 host-model 로 설정합니다.

   ```
   # virt-xml testguest --edit --cpu host-model,clearxml=yes
   Domain 'testguest' defined successfully.
   ```

검증

장치가 수정되었는지 확인하려면 다음 중 하나를 수행합니다.

- VM을 실행하고 장치가 있는지 테스트하고 수정 사항을 반영합니다.

- virsh dumpxml 명령을 사용하여 장치의 XML 정의가 VM의 XML 구성에서 수정되었는지 확인합니다.

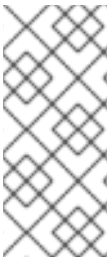
  예를 들어 다음 출력에서는 testguest VM의 구성을 보여주고 CPU 모드가 host-model 로 구성되었는지 확인합니다.

```
# virsh dumpxml testguest
[...]
<cpu mode='host-model' check='partial'>
  <model fallback='allow'/>
</cpu>
[...]
```

문제 해결

- 장치를 수정하면 VM을 부팅할 수 없게 되는 경우 virsh define 유틸리티를 사용하여 이전에 백업한 XML 구성 파일을 다시 로드하여 XML 구성을 복원합니다.

```
# virsh define testguest.xml
```



참고

VM의 XML 구성을 약간 변경하려면 virsh edit 명령(예: virsh edit testguest)을 사용할 수 있습니다. 그러나 VM이 부팅되지 않도록 할 수 있는 방식으로 구성이 중단될 가능성이 많기 때문에 이 방법을 사용하지 마십시오.

추가 리소스

- man virt-xml 명령

13.3.3. 가상 머신에서 장치 제거

가상 장치를 제거하여 VM(가상 머신) 기능을 변경할 수 있습니다. 예를 들어 VM 중 하나에서 가상 디스크 장치를 더 이상 필요하지 않은 경우 제거할 수 있습니다.

다음 절차에서는 CLI(명령줄 인터페이스)를 사용하여 VM(가상 머신)에서 가상 장치를 제거하는 방법을 보여줍니다. 디스크 또는 NIC와 같은 일부 장치는 RHEL 9 웹 콘솔을 사용하는 VM에서도 제거할 수 있

습니다.

사전 요구 사항

- *선택 사항: **virsh dumpxml vm-name** 을 사용하여 VM의 XML 구성을 백업하고 해당 출력을 파일에 전송합니다. 예를 들어, 다음에서는 **Motoko VM**의 구성을 **motoko.xml** 파일로 백업합니다.*

```
# virsh dumpxml Motoko > motoko.xml
# cat motoko.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>Motoko</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

절차

1. *장치 정의를 포함하여 **virt-xml --remove-device** 명령을 사용합니다. 예를 들면 다음과 같습니다.*

   - *다음은 실행 중인 **testguest VM**에서 **elasticsearch** 로 표시된 스토리지 장치를 종료한 후 제거합니다.*

     ```
     # virt-xml testguest --remove-device --disk target=vdb
     Domain 'testguest' defined successfully.
     Changes will take effect after the domain is fully powered off.
     ```

   - *다음은 실행 중인 **testguest2 VM**에서 **USB** 플래시 드라이브 장치를 즉시 제거합니다.*

     ```
     # virt-xml testguest2 --remove-device --update --hostdev type=usb
     Device hotunplug successful.
     Domain 'testguest2' defined successfully.
     ```

문제 해결

- *장치를 제거하면 **VM**을 부팅할 수 없게 되는 경우 **virsh define** 유틸리티를 사용하여 이전에 백업한 **XML** 구성 파일을 다시 로드하여 **XML** 구성을 복원합니다.*

  ```
  # virsh define testguest.xml
  ```

추가 리소스

- *man virt-xml 명령*

13.4. 웹 콘솔을 사용하여 호스트 장치 관리

VM(가상 머신)의 기능을 수정하려면 Red Hat Enterprise Linux 9 웹 콘솔을 사용하여 VM에 연결된 호스트 장치를 관리할 수 있습니다.

호스트 장치는 호스트 시스템에 연결된 물리적 장치입니다. 요구 사항에 따라 VM이 이러한 하드웨어 장치 및 구성 요소에 직접 액세스할 수 있습니다.

웹 콘솔을 사용하여 다음을 수행할 수 있습니다.

- *장치 보기*
- *장치 연결*
- *장치 제거*

13.4.1. 웹 콘솔을 사용하여 가상 머신에 연결된 장치 보기

VM(가상 머신)에 연결된 장치를 추가하거나 수정하기 전에 VM에 이미 연결된 장치를 볼 수 있습니다. 다음 절차에서는 웹 콘솔을 사용하여 이러한 장치를 확인하는 지침을 제공합니다.

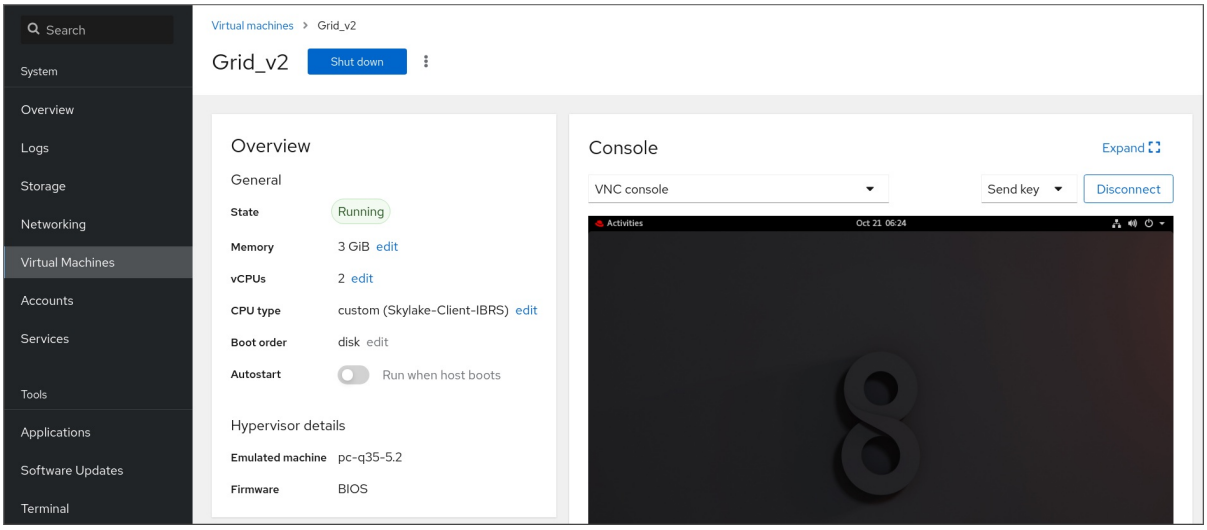
사전 요구 사항

- *웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.*

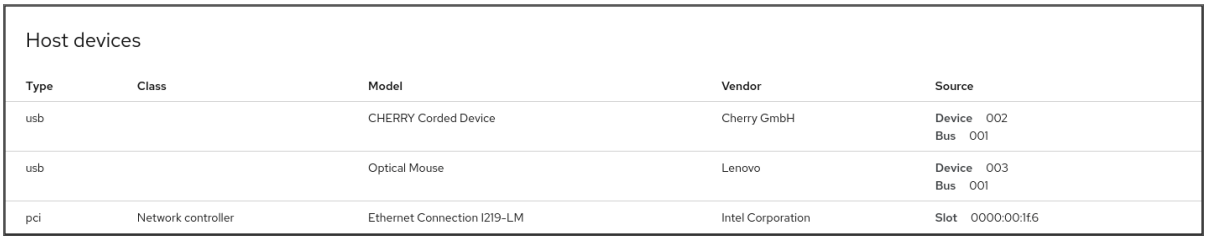
절차

1. *Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.*

   VM에 대한 자세한 정보가 포함된 새 페이지가 열립니다.



2. ***Host devices* (호스트 장치) 섹션으로 스크롤합니다.**



추가 리소스

- *가상 장치 관리*

13.4.2. 웹 콘솔을 사용하여 가상 머신에 장치 연결

***VM*(가상 머신)에 특정 기능을 추가하려면 웹 콘솔을 사용하여 호스트 장치를 *VM*에 연결할 수 있습니다.**

참고

여러 호스트 장치를 동시에 연결할 수 없습니다. 한 번에 하나의 장치만 연결할 수 있습니다.

*자세한 내용은 **RHEL 9 Known Issues**를 참조하십시오.*

사전 요구 사항

- ***PCI* 장치를 연결하는 경우 *hostdev* 요소의 *managed* 속성 상태가 *yes*로 설정되어 있는지 확인합니다.**

참고

PCI 장치를 VM에 연결할 때 hostdev 요소의 관리 속성을 생략하거나 no 로 설정하지 마십시오. 이렇게 하면 VM에 전달할 때 PCI 장치는 호스트에서 자동으로 분리할 수 없습니다. 또한 VM을 끄면 호스트에 자동으로 다시 연결할 수 없습니다.

결과적으로 호스트가 응답하지 않거나 예기치 않게 종료될 수 있습니다.

VM의 XML 구성에서 managed 속성의 상태를 찾을 수 있습니다. 다음 예제에서는 Ag47 VM 의 XML 구성을 엽니다.

```
# virsh edit Ag47
```

- VM에서 중요한 데이터를 백업합니다.
- 선택 사항: VM의 XML 구성을 백업합니다. 예를 들어 Centurion VM을 백업하려면 다음을 수행합니다.

```
# virsh dumpxml Centurion > Centurion.xml
```

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

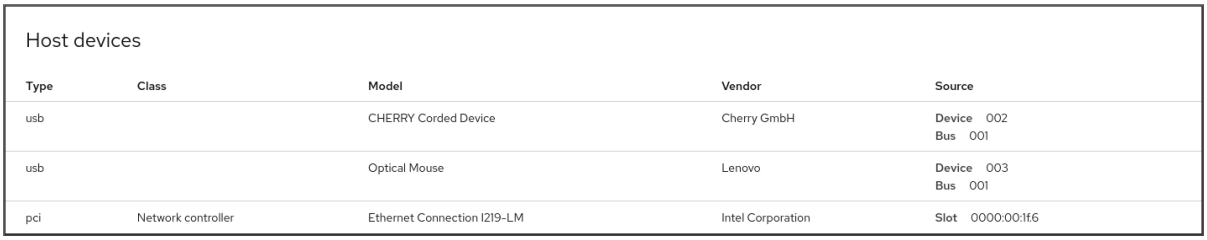
절차

1. Virtual Machines (가상 시스템) 인터페이스에서 호스트 장치를 연결할 VM을 클릭합니다.

VM의 그래픽 인터페이스에 액세스할 수 있도록 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 개요 섹션이 포함된 새 페이지가 열립니다.

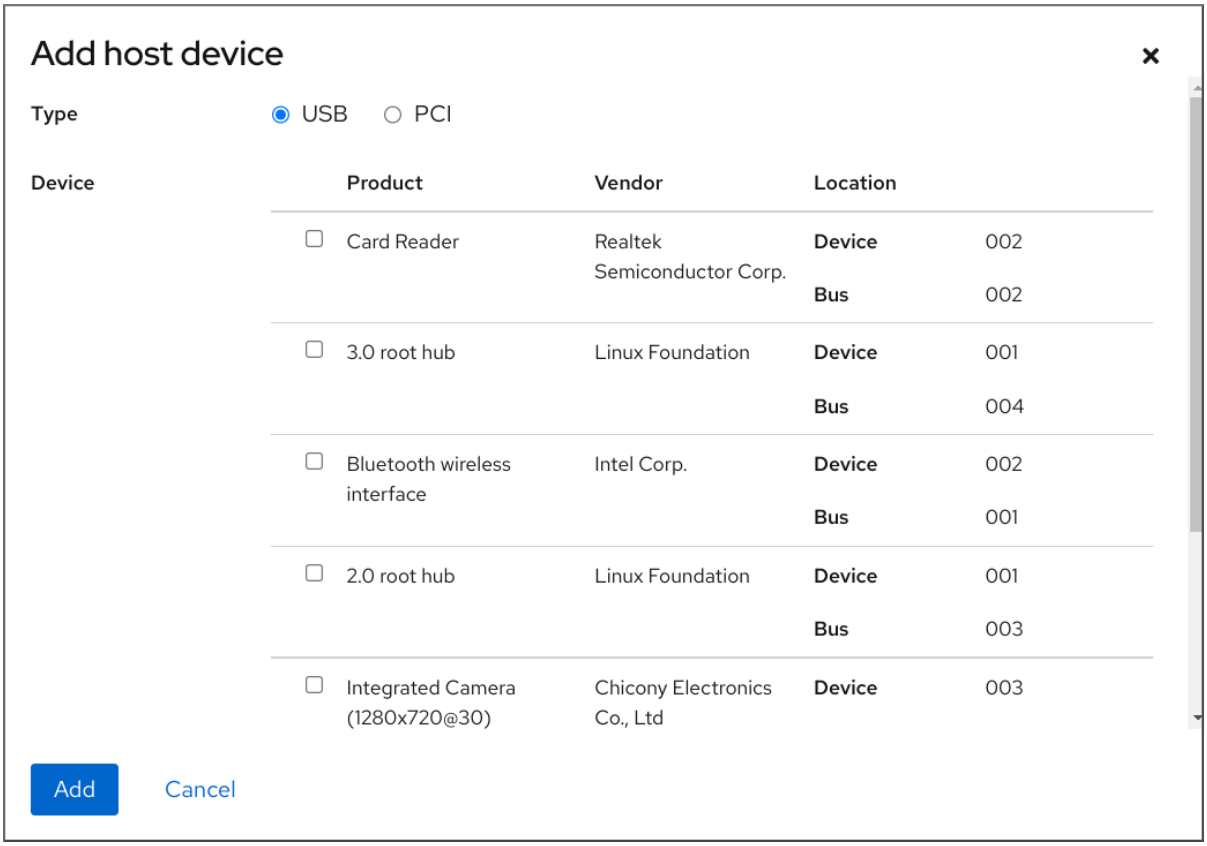
2. 호스트 장치로 스크롤합니다.

Host devices (호스트 장치) 섹션에는 VM에 연결된 장치와 장치 추가 또는 제거 옵션이 표시됩니다.



3. *호스트 장치 추가를 클릭합니다.*

   호스트 장치 추가 대화 상자가 나타납니다.



4. ***VM에 연결할 장치를 선택합니다.***

5. *추가를 클릭합니다.*

   *선택한 장치가 **VM**에 연결되어 있습니다.*

검증

- ***VM을 실행하고 장치가 Host devices (호스트 장치) 섹션에 표시되는지 확인합니다.***

13.4.3. 웹 콘솔을 사용하여 가상 머신에서 장치 제거

리소스를 확보하거나 VM의 기능을 수정하려면, 웹 콘솔을 사용하여 VM을 수정하고 더 이상 필요하지 않은 호스트 장치를 제거할 수 있습니다.

주의

웹 콘솔을 사용하여 연결된 USB 호스트 장치를 제거하는 것은 USB 장치의 장치와 버스 번호 간의 잘못된 상관 관계로 인해 실패할 수 있습니다.

자세한 내용은 RHEL 9 Known Issues를 참조하십시오.

해결 방법으로 "virsh" 유틸리티를 사용하여 VM의 XML 구성에서 USB 장치의 <hostdev> 부분을 제거합니다. 다음 예제에서는 Ag47 VM의 XML 구성을 엽니다.

```
# virsh edit Ag47
```

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 선택 사항: virsh dumpxml vm-name 을 사용하여 VM의 XML 구성을 백업하고 해당 출력을 파일에 전송합니다. 예를 들어, 다음에서는 Motoko VM의 구성을 motoko.xml 파일로 백업합니다.

```
# virsh dumpxml Motoko > motoko.xml
# cat motoko.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>Motoko</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

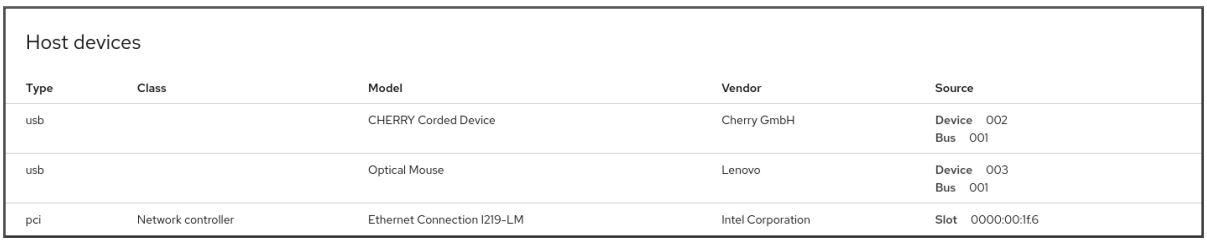
절차

1. 가상 머신 인터페이스에서 호스트 장치를 제거할 VM을 클릭합니다.

VM의 그래픽 인터페이스에 액세스할 수 있도록 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 개요 섹션이 포함된 새 페이지가 열립니다.

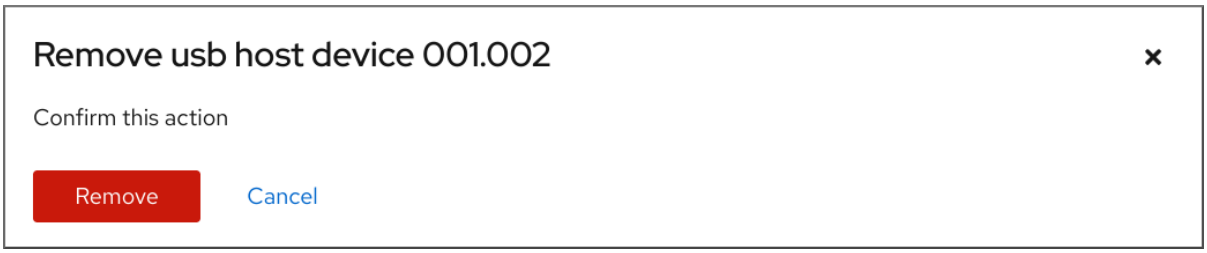
2. *호스트 장치로 스크롤합니다.*

   Host devices (호스트 장치) 섹션에는 VM에 연결된 장치와 장치 추가 또는 제거 옵션이 표시됩니다.



3. *VM에서 제거할 장치 옆에 있는 Remove (제거) 버튼을 클릭합니다.*

   제거 장치 확인 대화 상자가 나타납니다.



4. *제거를 클릭합니다.*

   장치가 VM에서 제거됩니다.

문제 해결

- *호스트 장치를 제거하면 VM을 부팅할 수 없게 되는 경우 virsh define 유틸리티를 사용하여 이전에 백업한 XML 구성 파일을 다시 로드하여 XML 구성을 복원합니다.*

```
# virsh define motoko.xml
```

13.5. 가상 USB 장치 관리

VM(가상 머신)을 사용하는 경우 호스트 시스템에 연결된 플래쉬 드라이브 또는 웹 카메라와 같은 USB 장치에 액세스하고 제어할 수 있습니다. 이 시나리오에서는 호스트 시스템이 장치에 대한 제어를 VM에 전달합니다. 이를 USB-passthrough라고도 합니다.

다음 섹션에서는 다음에 명령줄을 사용하는 방법에 대한 정보를 제공합니다.

- VM 에 USB 장치 연결
- VM 에서 USB 장치 제거

13.5.1. 가상 머신에 USB 장치 연결

USB 장치를 VM(가상 머신)에 연결하려면 VM의 XML 구성 파일에 USB 장치 정보를 포함할 수 있습니다.

사전 요구 사항

- VM에 전달할 장치가 호스트에 연결되어 있는지 확인합니다.

절차

1. VM에 연결할 USB의 버스 및 장치 값을 찾습니다.

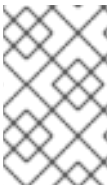
   예를 들어 다음 명령은 호스트에 연결된 USB 장치 목록을 표시합니다. 이 예제에서 사용할 장치는 버스 001에 장치 005로 연결됩니다.

   ```
   # lsusb
   [...]
   Bus 001 Device 003: ID 2567:0a2b Intel Corp.
   Bus 001 Device 005: ID 0407:6252 Kingston River 2.0
   [...]
   ```

2. --add-device 인수와 함께 virt-xml 유틸리티를 사용합니다.

   예를 들어 다음 명령은 USB 플래시 드라이브를 Library VM에 연결합니다.

```
# virt-xml Library --add-device --hostdev 001.005
Domain 'Library' defined successfully.
```



참고

실행 중인 VM에 USB 장치를 연결하려면 이전 명령에 --update 인수를 추가합니다.

검증

- VM을 실행하고 장치가 있고 예상대로 작동하는지 테스트합니다.
- virsh dumpxml 명령을 사용하여 장치의 XML 정의가 VM의 XML 구성 파일의 <devices> 섹션에 추가되었는지 확인합니다.

```
# virsh dumpxml Library
[...]
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x0407'/>
    <product id='0x6252'/>
    <address bus='1' device='5'/>
  </source>
  <alias name='hostdev0'/>
  <address type='usb' bus='0' port='3'/>
</hostdev>
[...]
```

추가 리소스

- man virt-xml 명령
- 가상 머신에 장치 연결

13.5.2. 가상 머신에서 USB 장치 제거

VM(가상 머신)에서 USB 장치를 제거하려면 VM의 XML 구성에서 USB 장치 정보를 제거할 수 있습니다.

절차

1. VM에서 제거할 USB의 버스 및 장치 값을 찾습니다.

   예를 들어 다음 명령은 호스트에 연결된 USB 장치 목록을 표시합니다. 이 예제에서 사용할 장치는 버스 001에 장치 005로 연결됩니다.

```
# lsusb
[...]
Bus 001 Device 003: ID 2567:0a2b Intel Corp.
Bus 001 Device 005: ID 0407:6252 Kingston River 2.0
[...]
```

2. --remove-device 인수와 함께 virt-xml 유틸리티를 사용합니다.

   예를 들어 다음 명령은 호스트가 001에서 장치 005로 연결된 USB 플래시 드라이브를 Library VM에서 제거합니다.

```
# virt-xml Library --remove-device --hostdev 001.005
Domain 'Library' defined successfully.
```

참고

실행 중인 VM에서 USB 장치를 제거하려면 이전 명령에 --update 인수를 추가합니다.

검증

- VM을 실행하고 장치 목록에서 장치가 제거되었는지 확인합니다.

추가 리소스

- man virt-xml 명령
- 가상 머신에 장치 연결

13.6. 가상 광 드라이브 관리

VM(가상 머신)을 사용하는 경우 호스트의 ISO 이미지에 저장된 정보에 액세스할 수 있습니다. 이렇게 하려면 CD 드라이브 또는 DVD 드라이브와 같은 가상 광 드라이브로 VM에 ISO 이미지를 연결합니다.

다음 섹션에서는 다음에 명령줄을 사용하는 방법에 대한 정보를 제공합니다.

- 드라이브 및 ISO 이미지를 VM에 연결
- 가상 광 드라이브의 ISO 이미지 교체
- 가상 광 드라이브에서 ISO 이미지 제거
- VM 에서 드라이브 제거

13.6.1. 가상 머신에 광 드라이브 연결

ISO 이미지를 가상 광 드라이브로 연결하려면 VM(가상 머신)의 XML 구성 파일을 편집하고 새 드라이브를 추가합니다.

사전 요구 사항

- ISO 이미지를 로컬 호스트에 저장해야 합니다.
- ISO 이미지의 경로를 알아야 합니다.

절차

- --add-device 인수와 함께 virt-xml 유틸리티를 사용합니다.

  예를 들어 다음 명령은 /MC/tank/ 디렉터리에 저장된 Doc10 ISO 이미지를 DN1 VM에 연결합니다.

```
# virt-xml DN1 --add-device --disk /MC/tank/Doc10.iso,device=cdrom
Domain 'DN1' defined successfully.
```

검증

- VM을 실행하고 장치가 있고 예상대로 작동하는지 테스트합니다.

추가 리소스

- man virt-xml 명령
- 가상 머신에 장치 연결

13.6.2. 가상 광 드라이브에서 ISO 이미지 교체

VM(가상 머신에 가상 광 드라이브)에 연결된 ISO 이미지를 교체하려면 VM의 XML 구성 파일을 편집하고 교체를 지정합니다.

사전 요구 사항

- ISO 이미지를 로컬 호스트에 저장해야 합니다.
- ISO 이미지의 경로를 알아야 합니다.

절차

1. CD-ROM이 VM에 연결된 대상 장치를 찾습니다. 이 정보는 VM의 XML 구성 파일에서 확인할 수 있습니다.

   예를 들어 다음 명령은 CD-ROM의 대상 장치가 sda 인 DN1 VM의 XML 구성 파일을 표시합니다.

```
# virsh dumpxml DN1
...
<disk>
  ...
  <source file='/MC/tank/Doc10.iso'/>
  <target dev='sda' bus='sata'/>
  ...
</disk>
...
```

2. edit 인수와 함께 virt-xml 유틸리티를 사용합니다.

   예를 들어 다음 명령은 대상 sda 에서 DN1 VM에 연결된 Doc10 ISO 이미지를 /Dvrs/current/ 디렉터리에 저장된 DrDN ISO 이미지로 교체합니다.

```
# virt-xml DN1 --edit target=sda --disk /Dvrs/current/DrDN.iso
Domain 'DN1' defined successfully.
```

검증

- *VM을 실행하고 장치가 교체되어 예상대로 작동하는지 테스트합니다.*

추가 리소스

- *man virt-xml 명령*

13.6.3. 가상 광 드라이브에서 ISO 이미지 제거

VM(가상 머신)에 연결된 가상 광 드라이브에서 ISO 이미지를 제거하려면 VM의 XML 구성 파일을 편집합니다.

절차

1. *CD-ROM이 VM에 연결된 대상 장치를 찾습니다. 이 정보는 VM의 XML 구성 파일에서 확인할 수 있습니다.*

   예를 들어 다음 명령은 CD-ROM의 대상 장치가 sda 인 DN1 VM의 XML 구성 파일을 표시합니다.

   ```
   # virsh dumpxml DN1
   ...
   <disk>
     ...
     <source file='/Dvrs/current/DrDN'/>
     <target dev='sda' bus='sata'/>
     ...
   </disk>
   ...
   ```

2. *edit 인수와 함께 virt-xml 유틸리티를 사용합니다.*

   예를 들어 다음 명령은 DN1 VM에 연결된 CD 드라이브에서 DrDN ISO 이미지를 제거합니다.

   ```
   # virt-xml DN1 --edit target=sda --disk path=
   Domain 'DN1' defined successfully.
   ```

검증

- VM을 실행하고 이미지를 더 이상 사용할 수 없는지 확인합니다.

추가 리소스

- man virt-xml 명령

13.6.4. 가상 머신에서 광 드라이브 제거

VM(가상 머신)에 연결된 광 드라이브를 제거하려면 VM의 XML 구성 파일을 편집합니다.

절차

1. CD-ROM이 VM에 연결된 대상 장치를 찾습니다. 이 정보는 VM의 XML 구성 파일에서 확인할 수 있습니다.

   예를 들어 다음 명령은 CD-ROM의 대상 장치가 sda 인 DN1 VM의 XML 구성 파일을 표시합니다.

```
# virsh dumpxml DN1
...
<disk type='file' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <target dev='sda' bus='sata'/>
  ...
</disk>
...
```

2. --remove-device 인수와 함께 virt-xml 유틸리티를 사용합니다.

   예를 들어 다음 명령은 DN1 VM에서 대상 sda 로 연결된 광섬유 드라이브를 제거합니다.

```
# virt-xml DN1 --remove-device --disk target=sda
Domain 'DN1' defined successfully.
```

검증

- 장치가 VM의 XML 구성 파일에 더 이상 나열되지 않는지 확인합니다.

추가 리소스

- man virt-xml 명령

13.7. SR-IOV 장치 관리

에뮬레이션된 가상 장치는 하드웨어 네트워크 장치보다 많은 CPU와 메모리를 사용하는 경우가 많습니다. 이렇게 하면 VM(가상 머신)의 성능이 제한될 수 있습니다. 그러나 가상화 호스트의 모든 장치가 SR-IOV(Single Root I/O Virtualization)를 지원하는 경우 이 기능을 사용하여 장치 성능을 개선하고 VM의 전체 성능도 향상시킬 수 있습니다.

13.7.1. SR-IOV는 무엇입니까?

SR-IOV(Single-root I/O virtualization)는 단일 PCI Express(PCI Express) 장치가 호스트 시스템에 VF(가상 기능)라는 여러 개의 별도의 PCI 장치를 제공할 수 있도록 하는 사양입니다. 이러한 각 장치는 다음과 같습니다.

- 원래 PCIe 장치와 동일하거나 유사한 서비스를 제공할 수 있습니다.
- 호스트 PCI 버스의 다른 주소에 나타납니다.
- VFIO 할당을 사용하여 다른 VM에 할당할 수 있습니다.

예를 들어 단일 SR-IOV 가능 네트워크 장치는 VF를 여러 VM에 제공할 수 있습니다. 모든 VF는 동일한 물리적 카드, 동일한 네트워크 연결, 동일한 네트워크 케이블을 사용하지만 각 VM은 자체 하드웨어 네트워크 장치를 직접 제어하고 호스트의 추가 리소스를 사용하지 않습니다.

SR-IOV의 작동 방식

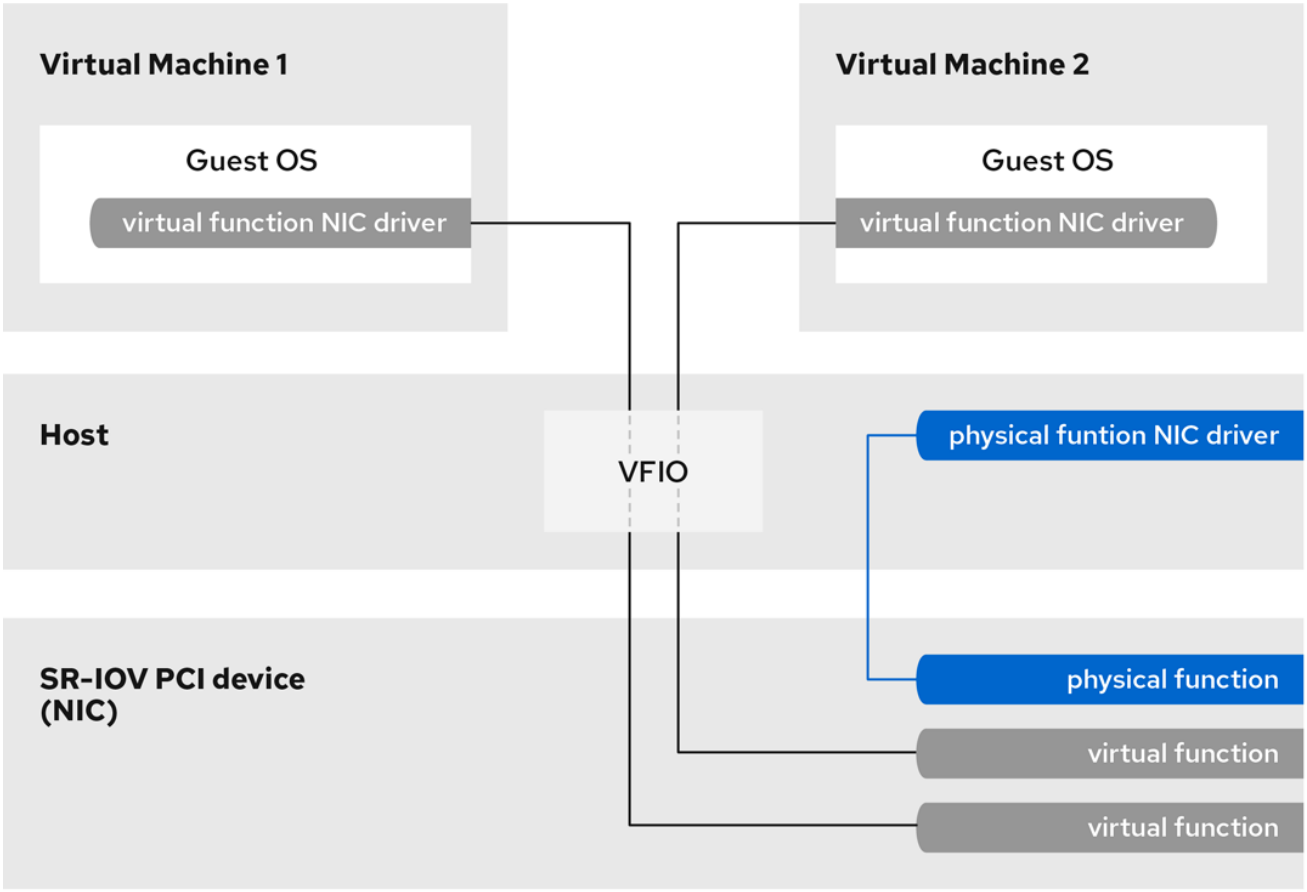
SR-IOV 기능은 다음 PCIe 함수를 도입하여 사용할 수 있습니다.

- PF(물리적 기능) - 호스트에 장치(예: 네트워킹) 기능을 제공하는 PCIe 함수이지만 VF 세트를 생성하고 관리할 수도 있습니다. 각 SR-IOV 가능 장치에는 하나 이상의 PF가 있습니다.
- VF(가상 기능) - 독립적인 장치로 작동하는 경량 PCIe 기능입니다. 각 VF는 PF에서 파생됩니다. 장치가 가질 수 있는 최대 VF 수는 장치 하드웨어에 따라 다릅니다. 각 VF는 한 번에 단일 VM

에만 할당할 수 있지만 VM에 여러 VF를 할당할 수 있습니다.

VM은 VF를 가상 장치로 인식합니다. 예를 들어 SR-IOV 네트워크 장치에서 생성된 VF는 실제 네트워크 카드가 호스트 시스템에 표시되는 것과 동일한 방식으로 할당된 VM에 대한 네트워크 카드로 나타납니다.

그림 13.1. SR-IOV 아키텍처



혜택

에뮬레이션된 장치가 아닌 SR-IOV VF를 사용할 때의 주요 장점은 다음과 같습니다.

- 성능 개선
- 호스트 CPU 및 메모리 리소스의 사용 감소

예를 들어 vNIC로 VM에 연결된 VF는 물리적 NIC와 거의 동일한 수준에서 작동하며 반가상화 또는 에뮬레이션된 NIC보다 훨씬 더 좋습니다. 특히 단일 호스트에서 여러 VF를 동시에 사용하는 경우 성능상의 이점이 중요할 수 있습니다.

단점

- PF의 구성을 수정하려면 먼저 PF에서 노출된 VF 수를 0으로 변경해야 합니다. 따라서 이러한 VF에서 제공하는 장치를 할당된 VM에서 제거해야 합니다.

- SR-IOV VF를 포함하여 연결된 VFIO 할당 장치가 있는 VM은 다른 호스트로 마이그레이션할 수 없습니다. 경우에 따라 할당된 장치를 에뮬레이트된 장치와 쌍으로 연결하여 이러한 제한을 해결할 수 있습니다. 예를 들어 할당된 네트워킹 VF를 에뮬레이션된 vNIC에 연결하고 마이그레이션 전에 VF를 제거할 수 있습니다. https://access.redhat.com/solutions/67546

- 또한 VFIO 할당 장치에는 VM 메모리 고정이 필요하므로 VM의 메모리 소비가 증가하고 VM에서 메모리 증대를 방지할 수 있습니다.

추가 리소스

- SR-IOV 할당에 지원되는 장치

13.7.2. 가상 머신에 SR-IOV 네트워킹 장치 연결

SR-IOV 네트워킹 장치를 Intel 또는 AMD 호스트의 가상 머신(VM)에 연결하려면 호스트의 SR-IOV 가능 네트워크 인터페이스에서 VF(가상 기능)를 생성하고 VF를 지정된 VM에 할당해야 합니다. 자세한 내용은 다음 지침을 참조하십시오.

사전 요구 사항

- 호스트의 CPU 및 펌웨어는 IOMMU(I/O Memory Management Unit)를 지원합니다.

  - Intel CPU를 사용하는 경우 Intel Virtualization Technology for Directed I/O(VT-d)를 지원해야 합니다.

  - AMD CPU를 사용하는 경우 AMD-Vi 기능을 지원해야 합니다.

- 호스트 시스템은 AS(Access Control Service)를 사용하여 PCIe 토폴로지에 대한 직접 메모리 액세스(DMA) 격리를 제공합니다. 시스템 공급 업체에서 확인하십시오.

  자세한 내용은 SR-IOV 구현에 대한 하드웨어 고려 사항을 참조하십시오.

- *물리적 네트워크 장치는 **SR-IOV**를 지원합니다. 시스템의 네트워크 장치가 **SR-IOV**를 지원하는지 확인하려면 **lspci -v** 명령을 사용하고 출력에서 **SR-IOV(Single Root I/O Virtualization)** 를 찾습니다.*

```
# lspci -v
[...]
02:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev
01)
 Subsystem: Intel Corporation Gigabit ET Dual Port Server Adapter
 Flags: bus master, fast devsel, latency 0, IRQ 16, NUMA node 0
 Memory at fcba0000 (32-bit, non-prefetchable) [size=128K]
[...]
 Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
 Capabilities: [160] Single Root I/O Virtualization (SR-IOV)
 Kernel driver in use: igb
 Kernel modules: igb
[...]
```

- *VF를 생성하는 데 사용할 호스트 네트워크 인터페이스가 실행 중입니다. 예를 들어 **eth1** 인터페이스를 활성화하고 실행 중인지 확인하려면 다음을 수행합니다.*

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

- *SR-IOV 장치 할당이 작동하려면 호스트 **BIOS** 및 커널에서 **IOMMU** 기능을 활성화해야 합니다. 이를 위해 다음을 수행합니다.*

  - *Intel 호스트에서 **VT-d**를 활성화합니다.*

    - *Intel 호스트에서 여러 부팅 항목을 사용하는 경우:*

      A. */etc/default/grub 파일을 편집하고 **GRUB_CMDLINE_LINUX** 행 끝에 **intel_iommu=on** 및 **iommu=pt** 매개변수를 추가합니다.*

```
GRUB_CMDLINE_LINUX="crashkernel=auto
resume=/dev/mapper/rhel_dell-per730-27-swap rd.lvm.lv=rhel_dell-per730-
27/root rd.lvm.lv=rhel_dell-per730-27/swap console=ttyS0,115200n81
```

```
intel_iommu=on iommu=pt"
```

B. *GRUB 설정을 다시 생성합니다.*

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

C. *호스트를 재부팅합니다.*

- *Intel 호스트가 단일 부팅 항목을 사용하는 경우:*

A. *intel_iommu=on iommu=pt 매개 변수를 사용하여 GRUB 설정을 다시 생성합니다.*

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

B. *호스트를 재부팅합니다.*

- *AMD 호스트에서 AMD-Vi를 활성화합니다.*

- *AMD 호스트에서 여러 부팅 항목을 사용하는 경우:*

A. */etc/default/grub 파일을 편집하고 GRUB_CMDLINE_LINUX 행 끝에 iommu=pt 매개변수를 추가합니다.*

```
GRUB_CMDLINE_LINUX="crashkernel=auto
resume=/dev/mapper/rhel_dell-per730-27-swap rd.lvm.lv=rhel_dell-per730-
27/root rd.lvm.lv=rhel_dell-per730-27/swap console=ttyS0,115200n81
iommu=pt"
```

B. *GRUB 설정을 다시 생성합니다.*

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

C. *호스트를 재부팅합니다.*

- ***AMD** 호스트에서 단일 부팅 항목을 사용하는 경우:*

  A. ***iommu=pt** 매개변수를 사용하여 **GRUB** 설정을 다시 생성합니다.*

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

  B. *호스트를 재부팅합니다.*

절차

1. *선택 사항: 네트워크 장치에서 사용할 수 있는 최대 **VF** 수를 확인합니다. 이렇게 하려면 다음 명령을 사용하고 **eth1** 을 **SR-IOV** 호환 네트워크 장치로 교체합니다.*

   ```
   # cat /sys/class/net/eth1/device/sriov_totalvfs
   7
   ```

2. *다음 명령을 사용하여 **VF**(가상 기능)를 생성합니다.*

   ```
   # echo VF-number > /sys/class/net/network-interface/device/sriov_numvfs
   ```

   명령에서 다음을 교체합니다.

   - ***PF**에서 생성할 **VF** 수가 있는 **VF** 번호입니다.*
   - ***VF**가 생성될 네트워크 인터페이스의 이름을 사용하여 **network-interface** 입니다.*

   *다음 예제는 **eth1** 네트워크 인터페이스에서 **2**개의 **VF**를 생성합니다.*

   ```
   # echo 2 > /sys/class/net/eth1/device/sriov_numvfs
   ```

3. ***VF**가 추가되었는지 확인합니다.*

   ```
   # lspci | grep Ethernet
   82:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network
   Connection (rev 01)
   ```

```
82:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network
Connection (rev 01)
82:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
82:10.2 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
```

4. *VF를 생성하는 데 사용한 네트워크 인터페이스에 대한 udev 규칙을 생성하여 생성된 VF를 영구적으로 만듭니다. 예를 들어 eth1 인터페이스의 경우 /etc/udev/rules.d/eth1.rules 파일을 생성하고 다음 행을 추가합니다.*

```
ACTION=="add", SUBSYSTEM=="net", ENV{ID_NET_DRIVER}=="ixgbe",
ATTR{device/sriov_numvfs}="2"
```

이렇게 하면 호스트가 시작될 때 ixgbe 드라이버를 사용하는 두 개의 VF를 eth1 인터페이스에서 자동으로 사용할 수 있습니다. 영구 SR-IOV 장치가 필요하지 않은 경우 이 단계를 건너뜁니다.

주의

현재 Broadcom NetXtreme II BCM57810 어댑터에서 VF를 영구적으로 만들려고 할 때 위에서 설명한 설정이 제대로 작동하지 않습니다. 또한 이러한 어댑터를 Windows VM에 기반으로 VF를 연결하는 것은 현재 신뢰할 수 없습니다.

5. *새로 추가된 VF 인터페이스 장치 중 하나를 실행 중인 VM에 핫플러그합니다.*

```
# virsh attach-interface testguest1 hostdev 0000:82:10.0 --managed --live --config
```

검증

- *절차가 성공적으로 수행되면 게스트 운영 체제는 새 네트워크 인터페이스 카드를 감지합니다.*

13.7.3. SR-IOV 할당에 지원되는 장치

일부 장치를 SR-IOV에 사용할 수 있는 것은 아닙니다. 다음 장치는 RHEL 9에서 SR-IOV와 호환되는 것으로 테스트 및 검증되었습니다.

네트워킹 장치

- Intel 82599ES 10 Gigabit Ethernet Controller - ixgbe 드라이버를 사용합니다.
- Intel Ethernet Controller XL710 Series - i40e 드라이버 사용
- Mellanox ConnectX-5 Ethernet Adapter Cards - mlx5_core 드라이버 사용
- Intel Ethernet Network Adapter XXV710 - i40e 드라이버를 사용합니다.
- Intel 82576 기가비트 이더넷 컨트롤러 - igb 드라이버 사용
- Broadcom NetXtreme II BCM57810 - bnx2x 드라이버를 사용합니다.

13.8. IBM Z의 가상 머신에 DASD 장치 연결

vfio-ccw 기능을 사용하면 DASD(직접 액세스 스토리지 장치)를 IBM Z 호스트의 VM(가상 머신)에 중재된 장치로 할당할 수 있습니다. 예를 들어 VM이 z/OS 데이터 집합에 액세스하거나 할당된 DASD를 z/OS 머신에 제공할 수 있습니다.

사전 요구 사항

- 호스트 시스템은 IBM Z 하드웨어 아키텍처를 사용하고 있으며 FICON 프로토콜을 지원합니다.
- 대상 VM은 Linux 게스트 운영 체제를 사용합니다.
- mdevctl 패키지가 설치됩니다.

```
# dnf install mdevctl
```

- *driverctl 패키지가 설치되어 있습니다.*

```
# dnf install driverctl
```

- *필요한 커널 모듈이 호스트에 로드되었습니다. 확인하려면 다음을 사용합니다.*

```
# lsmod | grep vfio
```

출력에는 다음 모듈이 포함되어야 합니다.

  - *vfio_ccw*
  - *vfio_mdev*
  - *vfio_iommu_type1*

- *VM에서 독점적으로 사용할 수 있는 예비 DASD 장치가 있으며 장치의 식별자를 알고 있습니다.*

이 절차에서는 예제로 0.0.002c 를 사용합니다. 명령을 수행할 때 0.0.002c 를 DASD 장치의 식별자로 바꿉니다.

절차

1. *DASD 장치의 하위 채널 식별자를 가져옵니다.*

```
# lscss -d 0.0.002c
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.002c 0.0.29a8  3390/0c 3990/e9 yes  f0  f0  ff   02111221 00000000
```

이 예에서 하위 채널 식별자는 0.0.29a8 로 탐지됩니다. 이 절차의 다음 명령에서 0.0.29a8 을 장치의 감지된 하위 채널 식별자로 바꿉니다.

2. *이전 단계의 **lscss** 명령이 헤더 출력과 장치 정보가 없는 경우 다음 단계를 수행합니다.*

   a. ***cio_ignore** 목록에서 장치를 제거합니다.*

   ```
   # cio_ignore -r 0.0.002c
   ```

   b. *게스트 **OS**에서 **VM** 의 커널 명령줄을 편집하고 아직 없는 경우 **cio_ignore=** 로 시작하는 줄에 **!** 기호가 있는 장치 식별자를 추가합니다.*

   ```
   cio_ignore=all,!condev,!0.0.002c
   ```

   c. *호스트에서 **1**단계를 반복하여 하위 채널 식별자를 가져옵니다.*

3. *하위 채널을 **vfio_ccw passthrough** 드라이버에 바인딩합니다.*

   ```
   # driverctl -b css set-override 0.0.29a8 vfio_ccw
   ```

   참고

   *그러면 **0.0.29a8** 하위 채널을 **vfio_ccw** 에 영구적으로 바인딩하므로 호스트에서 **DASD**를 사용할 수 없습니다. 호스트에서 장치를 사용해야 하는 경우 먼저 **'vfio_ccw'**에 대한 자동 바인딩을 제거하고 하위 채널을 기본 드라이버에 다시 바인딩해야 합니다.*

   ```
   # driverctl -b cs unset-override 0.0.29a8
   ```

4. ***UUID**를 생성합니다.*

   ```
   # uuidgen
   30820a6f-b1a5-4503-91ca-0c10ba12345a
   ```

5. *생성된 **UUID**를 사용하여 **DASD** 중재 장치를 생성합니다.*

   ```
   # mdevctl start --uuid 30820a6f-b1a5-4503-91ca-0c10ba12345a --parent 0.0.29a8 --type
   vfio_ccw-io
   ```

6. 중재된 장치를 영구적으로 만듭니다.

```
# mdevctl define --auto --uuid 30820a6f-b1a5-4503-91ca-0c10ba12345a
```

7. 실행 중인 경우 VM을 종료합니다.

8. 중재된 장치를 VM에 연결합니다. 이렇게 하려면 virsh edit 유틸리티를 사용하여 VM의 XML 구성을 편집하고, 다음 섹션을 XML에 추가하고, uuid 값을 이전 단계에서 생성한 UUID로 교체합니다.

```
<hostdev mode='subsystem' type='mdev' model='vfio-ccw'>
  <source>
    <address uuid="30820a6f-b1a5-4503-91ca-0c10ba12345a"/>
  </source>
</hostdev>
```

검증

1. 중재된 DASD 장치에 할당된 libvirt 의 식별자를 가져옵니다. 이렇게 하려면 VM의 XML 구성을 표시하고 vfio-ccw 장치를 찾습니다.

```
# virsh dumpxml vm-name

<domain>
[...]
    <hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ccw'>
      <source>
        <address uuid='10620d2f-ed4d-437b-8aff-beda461541f9'/>
      </source>
      <alias name='hostdev0'/>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0009'/>
    </hostdev>
[...]
</domain>
```

이 예에서 장치의 할당된 식별자는 0.0.0009 입니다.

2. VM을 시작하고 게스트 OS에 로그인합니다.

3. 게스트 OS에서 DASD 장치가 나열되었는지 확인합니다. 예를 들면 다음과 같습니다.

```
# lscss | grep 0.0.0009
0.0.0009 0.0.0007  3390/0c 3990/e9      f0  f0  ff   12212231 00000000
```

4. 게스트 OS에서 장치를 온라인으로 설정합니다. 예를 들면 다음과 같습니다.

```
# chccwdev -e 0.0009
Setting device 0.0.0009 online
Done
```

추가 리소스

- cio_ignore에 대한 IBM 문서
- 런타임 시 커널 매개변수 구성

14장. 가상 머신용 스토리지 관리

VM(가상 머신)은 실제 시스템과 마찬가지로 데이터, 프로그램 및 시스템 파일을 위한 스토리지가 필요합니다. VM 관리자는 가상 스토리지로 물리적 또는 네트워크 기반 스토리지를 VM에 할당할 수 있습니다. 기본 하드웨어에 관계없이 VM에 스토리지가 표시되는 방식을 수정할 수도 있습니다.

다음 섹션에서는 다양한 유형의 VM 스토리지, 작동 방식 및 CLI 또는 웹 콘솔을 사용하여 이를 관리하는 방법에 대한 정보를 제공합니다.

14.1. 가상 머신 스토리지 이해

가상 머신(VM) 스토리지를 처음 사용하거나 작동 방식에 대해 잘 모르는 경우 다음 섹션에서는 VM 스토리지의 다양한 구성 요소, 작동 방법, 관리 기본 사항 및 Red Hat에서 제공하는 지원되는 솔루션에 대한 일반적인 개요를 제공합니다.

다음에 대한 정보를 찾을 수 있습니다.

- 스토리지 풀
- 스토리지 볼륨
- libvirt를 사용하여 스토리지 관리
- VM 스토리지 개요
- 지원되지 않는 스토리지 풀 유형

14.1.1. 스토리지 풀 소개

스토리지 풀은 가상 시스템(VM)에 스토리지를 제공하기 위해 libvirt 에서 관리하는 파일, 디렉터리 또는 스토리지 장치입니다. 스토리지 풀을 스토리지 볼륨으로 분리하여 VM 이미지를 저장하거나 VM에 추가 스토리지로 연결할 수 있습니다.

또한 여러 VM이 동일한 스토리지 풀을 공유할 수 있어 스토리지 리소스를 더 효율적으로 할당할 수 있

습니다.

- *스토리지 풀은 영구 또는 임시일 수 있습니다.*
  - *영구 스토리지 풀은 시스템이 호스트 머신을 재시작해도 유지됩니다. **virsh pool-define** 을 사용하여 영구 스토리지 풀을 생성할 수 있습니다.*
  - *일시적인 스토리지 풀은 호스트가 재부팅될 때까지만 존재합니다. **virsh pool-create** 명령을 사용하여 임시 스토리지 풀을 생성할 수 있습니다.*

스토리지 풀 스토리지 유형

스토리지 풀은 로컬 또는 네트워크 기반(공유)일 수 있습니다.

- *로컬 스토리지 풀*

  *로컬 스토리지 풀은 호스트 서버에 직접 연결됩니다. 여기에는 로컬 디렉터리, 직접 연결된 디스크, 물리 파티션, 로컬 장치에 논리 볼륨 관리(**LVM**) 볼륨 그룹이 포함됩니다.*

  *로컬 스토리지 풀은 마이그레이션이 필요하지 않거나 다수의 **VM**이 없는 개발, 테스트 및 소규모 배포에 유용합니다.*

- *네트워크(공유) 스토리지 풀*

  네트워크로 연결된 스토리지 풀에는 표준 프로토콜을 사용하여 네트워크를 통해 공유하는 스토리지 장치가 포함됩니다.

14.1.2. 스토리지 볼륨 소개

*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. 스토리지 볼륨은 기본 하드웨어에 관계없이 디스크와 같은 로컬 스토리지 장치로 **VM**에 제공됩니다.*

*호스트 시스템에서 스토리지 볼륨은 이름 및 파생되는 스토리지 풀의 식별자로 참조합니다. **virsh** 명령 줄에서 **--pool storage_pool volume_name** 형식을 사용합니다.*

예를 들어 guest_images 풀에 firstimage 라는 볼륨에 대한 정보를 표시하려면 다음을 수행합니다.

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

14.1.3. libvirt를 사용한 스토리지 관리

libvirt 원격 프로토콜을 사용하여 VM 스토리지의 모든 측면을 관리할 수 있습니다. 이러한 작업은 원격 호스트에서도 수행할 수 있습니다. 결과적으로 RHEL 웹 콘솔과 같이 libvirt 를 사용하는 관리 애플리케이션을 사용하여 VM의 스토리지를 구성하는 데 필요한 모든 작업을 수행할 수 있습니다.

libvirt API를 사용하여 스토리지 풀의 볼륨 목록을 쿼리하거나 해당 스토리지 풀에서 사용 가능한 용량, 할당 및 사용 가능한 스토리지에 대한 정보를 가져올 수 있습니다. 이를 지원하는 스토리지 풀의 경우 libvirt API를 사용하여 스토리지 볼륨을 생성, 복제, 크기 조정, 삭제할 수도 있습니다. 또한 libvirt API를 사용하여 스토리지 볼륨에 데이터를 업로드하거나 스토리지 볼륨에서 데이터를 다운로드하거나 스토리지 볼륨에서 데이터를 초기화할 수 있습니다.

14.1.4. 스토리지 관리 개요

스토리지 관리에 사용 가능한 옵션을 설명하기 위해 다음 예제에서는 mount -t nfs.example.com:/path/to/share /path/to/data 를 사용하는 샘플 NFS 서버에 대해 설명합니다.

스토리지 관리자로서:

- 내보낸 서버 경로 및 클라이언트 대상 경로를 설명하도록 가상화 호스트에서 NFS 스토리지 풀을 정의할 수 있습니다. 따라서 libvirt 는 libvirt 가 시작될 때 또는 libvirt 를 실행하는 동안 필요에 따라 스토리지를 자동으로 마운트할 수 있습니다.

- 간단히 스토리지 풀 및 스토리지 볼륨을 VM에 이름으로 추가할 수 있습니다. 볼륨에 대상 경로를 추가할 필요가 없습니다. 따라서 대상 클라이언트 경로가 변경되더라도 VM에 영향을 미치지 않습니다.

- 자동 시작을 위해 스토리지 풀을 구성할 수 있습니다. 이렇게 하면 libvirt 가 시작될 때 지정된 디렉터리에 NFS 공유 디스크를 자동으로 마운트합니다. libvirt 는 nfs.example.com:/path/to/share /vmdata 와 유사하게 지정된 디렉터리에 공유를 마운트합니다.

- *libvirt API를 사용하여 스토리지 볼륨 경로를 쿼리할 수 있습니다. 이러한 스토리지 볼륨은 기본적으로 NFS 공유 디스크에 있는 파일입니다. 그런 다음 VM 블록 장치의 소스 스토리지를 설명하는 VM의 XML 정의 섹션에 이러한 경로를 복사할 수 있습니다.*

- *NFS의 경우 libvirt API를 사용하는 애플리케이션을 사용하여 풀 크기(공유의 스토리지 용량)까지 스토리지 풀(NFS 공유의 파일)에서 스토리지 볼륨을 생성하고 삭제할 수 있습니다.*

  일부 스토리지 풀 유형은 볼륨 생성 및 삭제를 지원하는 것은 아닙니다.

- *더 이상 필요하지 않은 경우 스토리지 풀을 중지할 수 있습니다. 스토리지 풀(pool-destroy)을 중지하면 시작 작업을 취소합니다. 이 경우 NFS 공유를 마운트 해제합니다. 공유의 데이터는 명령 이름이 제안한 경우에도 destroy 작업에 의해 수정되지 않습니다. 자세한 내용은 man virsh을 참조하십시오.*

14.1.5. 지원되지 않는 스토리지 풀 유형

지원되는 스토리지 풀 유형

다음은 RHEL에서 지원하는 스토리지 풀 유형 목록입니다.

- *디렉터리 기반 스토리지 풀*
- *디스크 기반 스토리지 풀*
- *파티션 기반 스토리지 풀*
- *iSCSI 기반 스토리지 풀*
- *LVM 기반 스토리지 풀*
- *NFS 기반 스토리지 풀*

- vHBA 장치가 있는 SCSI 기반 스토리지 풀
- 다중 경로 기반 스토리지 풀
- RBD 기반 스토리지 풀

지원되지 않는 스토리지 풀 유형

다음은 RHEL에서 지원하지 않는 libvirt 스토리지 풀 유형 목록입니다.

- Sheepdog 기반 스토리지 풀
- Vstorage 기반 스토리지 풀
- aware-based storage pool
- iSCSI-direct 스토리지 풀
- GlusterFS 스토리지 풀

14.2. CLI를 사용하여 가상 머신 스토리지 풀 관리

CLI를 사용하여 스토리지 풀의 다음 측면을 관리하여 VM(가상 머신)에 스토리지를 할당할 수 있습니다.

- 스토리지 풀 정보 보기
- 스토리지 풀 생성
  - CLI를 사용하여 디렉터리 기반 스토리지 풀 생성

- CLI를 사용하여 디스크 기반 스토리지 풀 생성
- CLI를 사용하여 파일 시스템 기반 스토리지 풀 생성
- CLI를 사용하여 iSCSI 기반 스토리지 풀 생성
- CLI를 사용하여 LVM 기반 스토리지 풀 생성
- CLI를 사용하여 NFS 기반 스토리지 풀 생성
- CLI를 사용하여 vHBA 장치로 SCSI 기반 스토리지 풀 생성

- 스토리지 풀 제거

14.2.1. CLI를 사용하여 스토리지 풀 정보 보기

CLI를 사용하면 스토리지 풀에 대한 제한된 또는 전체 세부 정보로 모든 스토리지 풀 목록을 볼 수 있습니다. 나열된 스토리지 풀을 필터링할 수도 있습니다.

절차

- virsh pool-list 명령을 사용하여 스토리지 풀 정보를 확인합니다.

```
# virsh pool-list --all --details
 Name              State    Autostart  Persistent    Capacity  Allocation   Available
 default           running  yes        yes         48.97 GiB   23.93 GiB   25.03 GiB
 Downloads         running  yes        yes        175.62 GiB   62.02 GiB  113.60 GiB
 RHEL-Storage-Pool running  yes        yes        214.62 GiB   93.02 GiB  168.60 GiB
```

추가 리소스

- virsh pool-list --help 명령

14.2.2. CLI를 사용하여 디렉터리 기반 스토리지 풀 생성

디렉터리 기반 스토리지 풀은 기존 마운트된 파일 시스템의 디렉터리를 기반으로 합니다. 예를 들어 파일 시스템의 나머지 공간을 다른 용도로 사용하려는 경우 유용합니다. **virsh** 유틸리티를 사용하여 디렉터리 기반 스토리지 풀을 생성할 수 있습니다.

사전 요구 사항

- 하이퍼바이저가 디렉터리 스토리지 풀을 지원하는지 확인합니다.

```
# virsh pool-capabilities | grep "'dir' supported='yes'"
```

명령에서 출력을 표시하는 경우 디렉터리 풀이 지원됩니다.

절차

1. 스토리지 풀 생성

virsh pool-define-as 명령을 사용하여 디렉터리 유형 스토리지 풀을 정의하고 생성합니다. 예를 들어 **/guest_images** 디렉터리를 사용하는 **guest_images_dir** 이라는 스토리지 풀을 생성하려면 다음을 수행합니다.

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

만들 스토리지 풀의 **XML** 구성이 이미 있는 경우 **XML**을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 디렉터리 기반 스토리지 풀 매개 변수를 참조하십시오.

2. 스토리지 풀 대상 경로 생성

virsh pool-build 명령을 사용하여 사전 포맷된 파일 시스템 스토리지 풀에 대한 스토리지 풀 대상 경로를 생성하고 스토리지 소스 장치를 초기화하고 데이터 형식을 정의합니다.

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

# ls -la /guest_images
 total 8
 drwx------.  2 root root 4096 May 31 19:38 .
 dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
```

3.

풀이 생성되었는지 확인합니다.

***virsh pool-list** 명령을 사용하여 풀이 생성되었는지 확인합니다.*

```
# virsh pool-list --all

Name              State      Autostart
-----------------------------------------
default           active     yes
guest_images_dir  inactive   no
```

4.

스토리지 풀 시작

***virsh pool-start** 명령을 사용하여 스토리지 풀을 마운트합니다.*

```
# virsh pool-start guest_images_dir
 Pool guest_images_dir started
```

참고

***virsh pool-start** 명령은 영구 스토리지 풀에만 필요합니다. 임시 스토리지 풀은 생성될 때 자동으로 시작됩니다.*

5.

[선택 사항] 자동 시작 켜기

*기본적으로 **virsh** 명령으로 정의된 스토리지 풀은 가상화 서비스가 시작될 때마다 자동으로 시작하도록 설정되지 않습니다. **virsh pool-autostart** 명령을 사용하여 스토리지 풀을 **autostart**로 구성합니다.*

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

검증

- *virsh pool-info** 명령을 사용하여 스토리지 풀이 **running** 상태인지 확인합니다. 보고된 크기가 예상대로 있고 **autostart**가 올바르게 구성되었는지 확인합니다.*

```
# virsh pool-info guest_images_dir
 Name:           guest_images_dir
 UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
```

```
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       458.39 GB
Allocation:     197.91 MB
Available:      458.20 GB
```

14.2.3. CLI를 사용하여 디스크 기반 스토리지 풀 생성

디스크 기반 스토리지 풀에서 풀은 디스크 파티션을 기반으로 합니다. 예를 들어 VM(가상 머신) 스토리지 전용 디스크 파티션을 사용하려는 경우 유용합니다. virsh 유틸리티를 사용하여 디스크 기반 스토리지 풀을 생성할 수 있습니다.

사전 요구 사항

- 하이퍼바이저가 디스크 기반 스토리지 풀을 지원하는지 확인합니다.

```
# virsh pool-capabilities | grep "'disk' supported='yes'"
```

명령에서 출력을 표시하는 경우 디스크 기반 풀이 지원됩니다.

- 스토리지 풀을 기반으로 할 장치를 준비합니다. 이를 위해 파티션(예: /dev/sdb1) 또는 LVM 볼륨을 우선합니다. VM에 전체 디스크 또는 블록 장치(예: /dev/sdb)에 대한 쓰기 액세스 권한이 있는 경우 VM이 파티션하거나 자체 LVM 그룹을 생성할 수 있습니다. 이로 인해 호스트에 시스템 오류가 발생할 수 있습니다.

  그러나 스토리지 풀에 전체 블록 장치를 사용해야 하는 경우 장치의 중요한 파티션을 GRUB의 os-prober 함수에서 보호하는 것이 좋습니다. 이를 위해 /etc/default/grub 파일을 편집하고 다음 설정 중 하나를 적용합니다.

  - os-prober 를 비활성화합니다.

    ```
    GRUB_DISABLE_OS_PROBER=true
    ```

  - os-prober 가 특정 파티션을 검색하지 못하도록 합니다. 예를 들면 다음과 같습니다.

    ```
    GRUB_OS_PROBER_SKIP_LIST="5ef6313a-257c-4d43@/dev/sdb1"
    ```

- 스토리지 풀을 생성하기 전에 선택한 스토리지 장치에 데이터를 백업하십시오. 사용 중인 libvirt 버전에 따라 디스크를 스토리지 풀에 전용으로 사용하면 디스크 장치에 현재 저장된 모든

데이터를 다시 포맷하고 지울 수 있습니다.

절차

1. *스토리지 풀 생성*

   ***virsh pool-define-as** 명령을 사용하여 디스크 유형 스토리지 풀을 정의하고 생성합니다. 다음 예제에서는 **/dev/sdb** 장치를 사용하고 **/dev** 디렉터리에 마운트된 **guest_images_disk** 라는 스토리지 풀을 생성합니다.*

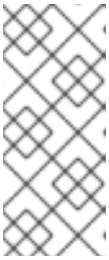
```
# virsh pool-define-as guest_images_disk disk --source-format=gpt --source-
dev=/dev/sdb --target /dev
Pool guest_images_disk defined
```

   *만들 스토리지 풀의 **XML** 구성이 이미 있는 경우 **XML**을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 디스크 기반 스토리지 풀 매개변수를 참조하십시오.*

2. *스토리지 풀 대상 경로 생성*

   ***virsh pool-build** 명령을 사용하여 사전 포맷된 파일 시스템 스토리지 풀에 대한 스토리지 풀 대상 경로를 생성하고 스토리지 소스 장치를 초기화하고 데이터 형식을 정의합니다.*

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



   참고

   *대상 경로 구축은 디스크 기반, 파일 시스템 기반 및 논리 스토리지 풀에만 필요합니다. **libvirt** 에서 소스 스토리지 장치의 데이터 형식이 선택한 스토리지 풀 유형과 다른 것을 감지하면 덮어쓰기 옵션을 지정하지 않는 한 빌드가 실패합니다.*

3. *풀이 생성되었는지 확인합니다.*

   ***virsh pool-list** 명령을 사용하여 풀이 생성되었는지 확인합니다.*

```
# 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 마운트합니다.

```
# virsh pool-start guest_images_disk
 Pool guest_images_disk started
```

참고

virsh pool-start 명령은 영구 스토리지 풀에만 필요합니다. 임시 스토리지 풀은 생성될 때 자동으로 시작됩니다.

5. [선택 사항] 자동 시작 켜기

   기본적으로 **virsh** 명령으로 정의된 스토리지 풀은 가상화 서비스가 시작될 때마다 자동으로 시작하도록 설정되지 않습니다. **virsh pool-autostart** 명령을 사용하여 스토리지 풀을 **autostart** 로 구성합니다.

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

검증

- **virsh pool-info** 명령을 사용하여 스토리지 풀이 **running** 상태인지 확인합니다. 보고된 크기가 예상대로 있고 **autostart**가 올바르게 구성되었는지 확인합니다.

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

14.2.4. CLI를 사용하여 파일 시스템 기반 스토리지 풀 생성

마운트되지 않은 파일 시스템에서 스토리지 풀을 생성하려면 파일 시스템 기반 스토리지 풀을 사용합니다. 이 스토리지 풀은 지정된 파일 시스템 마운트 지점을 기반으로 합니다. **virsh** 유틸리티를 사용하여 파일 시스템 기반 스토리지 풀을 생성할 수 있습니다.

사전 요구 사항

- 하이퍼바이저가 파일 시스템 기반 스토리지 풀을 지원하는지 확인합니다.

```
# virsh pool-capabilities | grep "'fs' supported='yes'"
```

명령에서 출력을 표시하는 경우 파일 기반 풀이 지원됩니다.

- 스토리지 풀을 기반으로 할 장치를 준비합니다. 이를 위해 파티션(예: **/dev/sdb1**) 또는 **LVM** 볼륨을 우선합니다. **VM**에 전체 디스크 또는 블록 장치(예: **/dev/sdb**)에 대한 쓰기 액세스 권한이 있는 경우 **VM**이 파티션하거나 자체 **LVM** 그룹을 생성할 수 있습니다. 이로 인해 호스트에 시스템 오류가 발생할 수 있습니다.

  그러나 스토리지 풀에 전체 블록 장치를 사용해야 하는 경우 장치의 중요한 파티션을 **GRUB**의 **os-prober** 함수에서 보호하는 것이 좋습니다. 이를 위해 **/etc/default/grub** 파일을 편집하고 다음 설정 중 하나를 적용합니다.

  - **os-prober** 를 비활성화합니다.

```
GRUB_DISABLE_OS_PROBER=true
```

  - **os-prober** 가 특정 파티션을 검색하지 못하도록 합니다. 예를 들면 다음과 같습니다.

```
GRUB_OS_PROBER_SKIP_LIST="5ef6313a-257c-4d43@/dev/sdb1"
```

절차

1. 스토리지 풀 생성

   virsh pool-define-as 명령을 사용하여 파일 시스템 유형 스토리지 풀을 정의하고 생성합니다. 예를 들어 **/dev/sdc1** 파티션을 사용하고 **/guest_images** 디렉터리에 마운트된 **guest_images_fs** 라는 스토리지 풀을 생성하려면 다음을 수행합니다.

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

만들 스토리지 풀의 **XML** 구성이 이미 있는 경우 **XML**을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 *파일 시스템 기반 스토리지 풀 매개 변수를* 참조하십시오.

2. 스토리지 풀 대상 경로 정의

virsh pool-build 명령을 사용하여 사전 포맷된 파일 시스템 스토리지 풀에 대한 스토리지 풀 대상 경로를 생성하고 스토리지 소스 장치를 초기화하고 데이터 형식을 정의합니다.

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

# ls -la /guest_images
 total 8
 drwx------.  2 root root 4096 May 31 19:38 .
 dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
```

3. 풀이 생성되었는지 확인합니다.

virsh pool-list 명령을 사용하여 풀이 생성되었는지 확인합니다.

```
# virsh pool-list --all

 Name              State      Autostart
 -----------------------------------------
 default           active     yes
 guest_images_fs   inactive   no
```

4. 스토리지 풀 시작

virsh pool-start 명령을 사용하여 스토리지 풀을 마운트합니다.

```
# virsh pool-start guest_images_fs
 Pool guest_images_fs started
```

참고

virsh pool-start 명령은 영구 스토리지 풀에만 필요합니다. 임시 스토리지 풀은 생성될 때 자동으로 시작됩니다.

5. [선택 사항] 자동 시작 켜기

기본적으로 **virsh** 명령으로 정의된 스토리지 풀은 가상화 서비스가 시작될 때마다 자동으로 시작하도록 설정되지 않습니다. **virsh pool-autostart** 명령을 사용하여 스토리지 풀을 **autostart** 로 구성합니다.

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

검증

1. **virsh pool-info** 명령을 사용하여 스토리지 풀이 **running** 상태인지 확인합니다. 보고된 크기가 예상대로 있고 **autostart**가 올바르게 구성되었는지 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
```

2. 파일 시스템의 대상 경로에 장치가 마운트되었음을 나타내는 **lost+found** 디렉터리가 있는지 확인합니다.

```
# 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
```

14.2.5. CLI를 사용하여 iSCSI 기반 스토리지 풀 생성

iSCSI(Internet Small Computer Systems Interface)는 데이터 스토리지 기능을 연결하는 IP 기반 스토리지 네트워킹 표준입니다. iSCSI 서버에 스토리지 풀을 사용하려면 virsh 유틸리티를 사용하여 iSCSI 기반 스토리지 풀을 생성할 수 있습니다.

사전 요구 사항

- *하이퍼바이저가 iSCSI 기반 스토리지 풀을 지원하는지 확인합니다.*

```
# virsh pool-capabilities | grep '"iscsi' supported='yes'"
```

명령에 출력이 표시되면 iSCSI 기반 풀이 지원됩니다.

절차

1. *스토리지 풀 생성*

virsh pool-define-as 명령을 사용하여 iSCSI 유형 스토리지 풀을 정의하고 생성합니다. 예를 들어 server1.example.com 에서 iqn.2010-05.com.example.server1:iscsirhel7guest IQN 을 사용하는 guest_images_iscsi 라는 스토리지 풀을 생성하려면 /dev/disk/by-path 경로에 마운트됩니다.

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

만들 스토리지 풀의 XML 구성이 이미 있는 경우 XML을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 iSCSI 기반 스토리지 풀 매개 변수를 참조하십시오.

2. *풀이 생성되었는지 확인합니다.*

virsh pool-list 명령을 사용하여 풀이 생성되었는지 확인합니다.

```
# virsh pool-list --all

Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_iscsi   inactive   no
```

3.

스토리지 풀 시작

***virsh pool-start** 명령을 사용하여 스토리지 풀을 마운트합니다.*

```
# virsh pool-start guest_images_iscsi
 Pool guest_images_iscsi started
```

참고

***virsh pool-start** 명령은 영구 스토리지 풀에만 필요합니다. 임시 스토리지 풀은 생성될 때 자동으로 시작됩니다.*

4.

[선택 사항] 자동 시작 켜기

*기본적으로 **virsh** 명령으로 정의된 스토리지 풀은 가상화 서비스가 시작될 때마다 자동으로 시작하도록 설정되지 않습니다. **virsh pool-autostart** 명령을 사용하여 스토리지 풀을 **autostart**로 구성합니다.*

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

검증

- *virsh pool-info 명령을 사용하여 스토리지 풀이 running 상태인지 확인합니다. 보고된 크기가 예상대로 있고 autostart가 올바르게 구성되었는지 확인합니다.*

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

14.2.6. CLI를 사용하여 LVM 기반 스토리지 풀 생성

*LVM 볼륨 그룹의 일부인 스토리지 풀을 사용하려면 **virsh** 유틸리티를 사용하여 LVM 기반 스토리지 풀을 생성할 수 있습니다.*

권장 사항

LVM 기반 스토리지 풀을 생성하기 전에 다음 사항에 유의하십시오.

- *LVM 기반 스토리지 풀은 LVM의 완전한 유연성을 제공하지 않습니다.*
- *libvirt 는 씬 논리 볼륨을 지원하지만 씬 스토리지 풀의 기능은 제공하지 않습니다.*
- *LVM 기반 스토리지 풀은 볼륨 그룹입니다. virsh 유틸리티를 사용하여 볼륨 그룹을 만들 수 있지만 이렇게 하면 생성된 볼륨 그룹에 하나의 장치만 있을 수 있습니다. 여러 장치가 있는 볼륨 그룹을 만들려면 대신 LVM 유틸리티를 사용하십시오. LVM을 사용하여 Linux에서 볼륨 그룹을 만드는 방법을 참조하십시오.*

  볼륨 그룹에 대한 자세한 내용은 Red Hat Enterprise Linux Logical Volume Manager 관리 가이드 를 참조하십시오.
- *LVM 기반 스토리지 풀에는 전체 디스크 파티션이 필요합니다. virsh 명령을 사용하여 새 파티션 또는 장치를 활성화하면 파티션이 포맷되고 모든 데이터가 삭제됩니다. 이 절차에서와 같이 호스트의 기존 볼륨 그룹을 사용하는 경우 아무것도 삭제되지 않습니다.*

사전 요구 사항

- *하이퍼바이저가 LVM 기반 스토리지 풀을 지원하는지 확인합니다.*

  ```
  # virsh pool-capabilities | grep "'logical' supported='yes'"
  ```

  명령에서 출력을 표시하는 경우 LVM 기반 풀이 지원됩니다.

절차

1. *스토리지 풀 생성*

   virsh pool-define-as 명령을 사용하여 LVM 유형 스토리지 풀을 정의하고 생성합니다. 예를 들어 다음 명령은 lvm_vg 볼륨 그룹을 사용하고 /dev/ lvm_vg 디렉터리에 마운트된 guest_images_lvm 이라는 스토리지 풀을 생성합니다.

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

만들 스토리지 풀의 **XML** 구성이 이미 있는 경우 **XML**을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 *LVM 기반 스토리지 풀 매개 변수*를 참조하십시오.

2. 풀이 생성되었는지 확인합니다.

virsh pool-list 명령을 사용하여 풀이 생성되었는지 확인합니다.

```
# virsh pool-list --all

Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_lvm     inactive   no
```

3. 스토리지 풀 시작

virsh pool-start 명령을 사용하여 스토리지 풀을 마운트합니다.

```
# virsh pool-start guest_images_lvm
 Pool guest_images_lvm started
```

참고

virsh pool-start 명령은 영구 스토리지 풀에만 필요합니다. 임시 스토리지 풀은 생성될 때 자동으로 시작됩니다.

4. [선택 사항] 자동 시작 켜기

기본적으로 **virsh** 명령으로 정의된 스토리지 풀은 가상화 서비스가 시작될 때마다 자동으로 시작하도록 설정되지 않습니다. **virsh pool-autostart** 명령을 사용하여 스토리지 풀을 **autostart**로 구성합니다.

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

검증

- *virsh pool-info 명령을 사용하여 스토리지 풀이 running 상태인지 확인합니다. 보고된 크기가 예상대로 있고 autostart가 올바르게 구성되었는지 확인합니다.*

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

14.2.7. CLI를 사용하여 NFS 기반 스토리지 풀 생성

NFS(Network File System) 서버에 스토리지 풀을 사용하려면 virsh 유틸리티를 사용하여 NFS 기반 스토리지 풀을 생성할 수 있습니다.

사전 요구 사항

- *하이퍼바이저가 NFS 기반 스토리지 풀을 지원하는지 확인합니다.*

```
# virsh pool-capabilities | grep "<value>nfs</value>"
```

명령에서 출력을 표시하는 경우 NFS 기반 풀이 지원됩니다.

절차

1. *스토리지 풀 생성*

virsh pool-define-as 명령을 사용하여 NFS 유형 스토리지 풀을 정의하고 생성합니다. 예를 들어 대상 디렉토리 /var/lib/libvirt/images/nfspool 을 사용하여 서버 디렉터리 /home/net_mount 에 마운트된 IP 111.222.111.222 가 있는 NFS 서버를 사용하는 guest_images_netfs 라는 스토리지 풀을 생성하려면 다음을 수행합니다.

```
# virsh pool-define-as --name guest_images_netfs --type netfs --source-
host='111.222.111.222' --source-path='/home/net_mount' --source-format='nfs' --
target='/var/lib/libvirt/images/nfspool'
```

만들 스토리지 풀의 XML 구성이 이미 있는 경우 XML을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 NFS 기반 스토리지 풀 매개 변수를 참조하십시오.

2.

풀이 생성되었는지 확인합니다.

***virsh pool-list** 명령을 사용하여 풀이 생성되었는지 확인합니다.*

```
# virsh pool-list --all

Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_netfs   inactive   no
```

3.

스토리지 풀 시작

***virsh pool-start** 명령을 사용하여 스토리지 풀을 마운트합니다.*

```
# virsh pool-start guest_images_netfs
 Pool guest_images_netfs started
```

참고

***virsh pool-start** 명령은 영구 스토리지 풀에만 필요합니다. 임시 스토리지 풀은 생성될 때 자동으로 시작됩니다.*

4.

[선택 사항] 자동 시작 켜기

*기본적으로 **virsh** 명령으로 정의된 스토리지 풀은 가상화 서비스가 시작될 때마다 자동으로 시작하도록 설정되지 않습니다. **virsh pool-autostart** 명령을 사용하여 스토리지 풀을 **autostart**로 구성합니다.*

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

검증

- *virsh pool-info** 명령을 사용하여 스토리지 풀이 **running** 상태인지 확인합니다. 보고된 크기가 예상대로 있고 **autostart**가 올바르게 구성되었는지 확인합니다.*

```
# virsh pool-info guest_images_netfs
```

```
Name:           guest_images_netfs
UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       458.39 GB
Allocation:     197.91 MB
Available:      458.20 GB
```

14.2.8. CLI를 사용하여 vHBA 장치로 SCSI 기반 스토리지 풀 생성

SCSI(Small Computer System Interface) 장치에 스토리지 풀을 사용하려면 호스트에서 가상 호스트 버스 어댑터(vHBA)를 사용하여 SCSI 장치에 연결할 수 있어야 합니다. 그런 다음 virsh 유틸리티를 사용하여 SCSI 기반 스토리지 풀을 생성할 수 있습니다.

사전 요구 사항

- 하이퍼바이저가 SCSI 기반 스토리지 풀을 지원하는지 확인합니다.

```
# virsh pool-capabilities | grep "'scsi' supported='yes'"
```

명령에 출력이 표시되면 SCSI 기반 풀이 지원됩니다.

- vHBA 장치를 사용하여 SCSI 기반 스토리지 풀을 생성하기 전에 vHBA를 생성합니다. 자세한 내용은 vHBAs 생성을 참조하십시오.

절차

1. 스토리지 풀 생성

virsh pool-define-as 명령을 사용하여 vHBA를 사용하여 SCSI 스토리지 풀을 정의하고 생성합니다. 예를 들어 다음은 scsi_host3 상위 어댑터로 식별되는 vHBA, 세계 전체 포트 번호 5001a4ace3ee047d, world-wide 노드 번호 5001a4a93526d0a1 을 사용하는 guest_images_vhba 라는 스토리지 풀을 생성합니다. 스토리지 풀은 /dev/disk/ 디렉터리에 마운트됩니다.

```
# virsh pool-define-as guest_images_vhba scsi --adapter-parent scsi_host3 --adapter-
wwnn 5001a4a93526d0a1 --adapter-wwpn 5001a4ace3ee047d --target /dev/disk/
Pool guest_images_vhba defined
```

만들 스토리지 풀의 XML 구성이 이미 있는 경우 XML을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 vHBA 장치가 있는 SCSI 기반 스토리지 풀 매개 변수 를 참조하십시오.

2. 풀이 생성되었는지 확인합니다.

   virsh pool-list 명령을 사용하여 풀이 생성되었는지 확인합니다.

```
# virsh pool-list --all

Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_vhba    inactive   no
```

3. 스토리지 풀 시작

   virsh pool-start 명령을 사용하여 스토리지 풀을 마운트합니다.

```
# virsh pool-start guest_images_vhba
 Pool guest_images_vhba started
```

참고

virsh pool-start 명령은 영구 스토리지 풀에만 필요합니다. 임시 스토리지 풀은 생성될 때 자동으로 시작됩니다.

4. [선택 사항] 자동 시작 켜기

   기본적으로 **virsh** 명령으로 정의된 스토리지 풀은 가상화 서비스가 시작될 때마다 자동으로 시작하도록 설정되지 않습니다. **virsh pool-autostart** 명령을 사용하여 스토리지 풀을 **autostart**로 구성합니다.

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

검증

- **virsh pool-info** 명령을 사용하여 스토리지 풀이 **running** 상태인지 확인합니다. 보고된 크기가 예상대로 있고 **autostart**가 올바르게 구성되었는지 확인합니다.

```
# virsh pool-info guest_images_vhba
```

```
Name:           guest_images_vhba
UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       458.39 GB
Allocation:     197.91 MB
Available:      458.20 GB
```

14.2.9. CLI를 사용하여 스토리지 풀 삭제

호스트 시스템에서 스토리지 풀을 제거하려면 풀을 중지하고 해당 **XML** 정의를 제거해야 합니다.

절차

1. **virsh pool-list** 명령을 사용하여 정의된 스토리지 풀을 나열합니다.

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
Downloads            active     yes
RHEL-Storage-Pool    active     yes
```

2. **virsh pool-destroy** 명령을 사용하여 삭제하려는 스토리지 풀을 중지합니다.

```
# virsh pool-destroy Downloads
Pool Downloads destroyed
```

3. 선택 사항: 일부 유형의 스토리지 풀의 경우 **virsh pool-delete** 명령을 사용하여 스토리지 풀이 있는 디렉터리를 제거할 수 있습니다. 이렇게 하려면 디렉터리가 비어 있어야 합니다.

```
# virsh pool-delete Downloads
Pool Downloads deleted
```

4. **virsh pool-undefine** 명령을 사용하여 스토리지 풀의 정의를 삭제합니다.

```
# virsh pool-undefine Downloads
Pool Downloads has been undefined
```

검증

- 스토리지 풀이 삭제되었는지 확인합니다.

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
RHEL-Storage-Pool    active     yes
```

14.3. 웹 콘솔을 사용하여 가상 머신 스토리지 풀 관리

RHEL 웹 콘솔을 사용하면 스토리지 풀을 관리하여 VM(가상 머신)에 스토리지를 할당할 수 있습니다.

웹 콘솔을 사용하여 다음을 수행할 수 있습니다.

- 스토리지 풀 정보 보기.
- 스토리지 풀을 생성합니다.
  - 디렉터리 기반 스토리지 풀 생성.
  - NFS 기반 스토리지 풀을 생성합니다.
  - iSCSI 기반 스토리지 풀을 생성합니다.
  - LVM 기반 스토리지 풀을 생성합니다.
- 스토리지 풀 제거.
- 스토리지 풀을 비활성화 합니다.

14.3.1. 웹 콘솔을 사용하여 스토리지 풀 정보 보기

웹 콘솔을 사용하면 시스템에서 사용 가능한 스토리지 풀에 대한 자세한 정보를 볼 수 있습니다. 스토

리지 풀을 사용하여 가상 머신의 디스크 이미지를 생성할 수 있습니다.

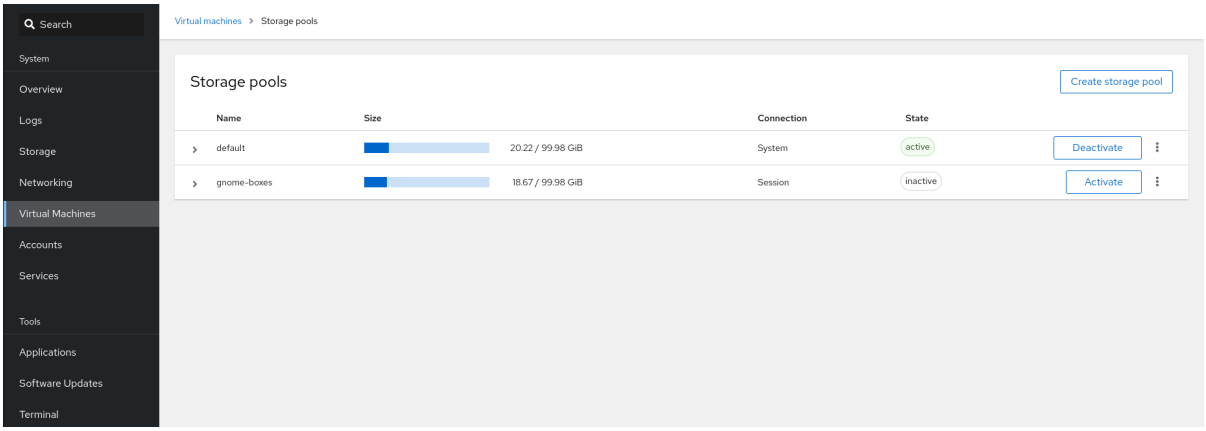
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. 가상 머신 인터페이스 상단에 있는 스토리지 풀 을 클릭합니다.

   구성된 스토리지 풀 목록을 보여주는 스토리지 풀 창이 표시됩니다.

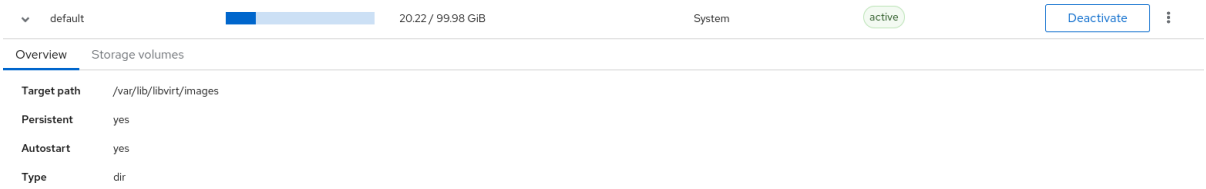


   정보에는 다음이 포함됩니다.

   - Name - 스토리지 풀의 이름입니다.
   - size - 스토리지 풀의 현재 할당 및 총 용량입니다.
   - 연결 - 스토리지 풀에 액세스하는 데 사용되는 연결입니다.
   - 상태 - 스토리지 풀의 상태입니다.

2. 정보를 보려는 스토리지 풀 옆에 있는 화살표를 클릭합니다.

행이 확장되어 선택한 스토리지 풀에 대한 자세한 정보가 포함된 개요 창을 표시합니다.

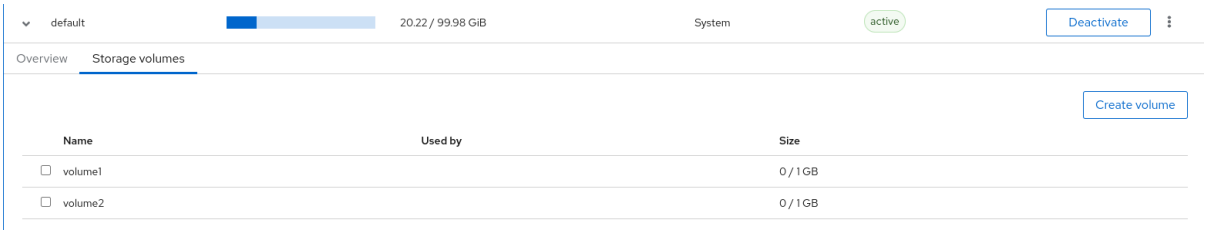


이 정보에는 다음이 포함됩니다.

- *대상 경로 - 디렉터리에서 지원하는 스토리지 풀 유형 소스(예: **dir** 또는 **netfs**)입니다.*
- ***persistent** - 스토리지 풀에 영구 구성이 있는지 여부를 나타냅니다.*
- ***autostart** - 시스템 부팅 시 스토리지 풀이 자동으로 시작되는지 여부를 나타냅니다.*
- ***type** - 스토리지 풀의 유형입니다.*

3. *스토리지 풀과 연결된 스토리지 볼륨 목록을 보려면 스토리지 볼륨 을 클릭합니다.*

   *구성된 스토리지 볼륨 목록을 보여주는 **Storage Volumes**(스토리지 볼륨) 창이 표시됩니다.*



이 정보에는 다음이 포함됩니다.

- ***name** - 스토리지 볼륨의 이름입니다.*
- *사용됨 - 현재 스토리지 볼륨을 사용 중인 **VM**입니다.*

- **size** - 볼륨의 크기입니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 정보 보기

14.3.2. 웹 콘솔을 사용하여 디렉터리 기반 스토리지 풀 생성

디렉터리 기반 스토리지 풀은 기존 마운트된 파일 시스템의 디렉터리를 기반으로 합니다. 예를 들어 파일 시스템의 나머지 공간을 다른 용도로 사용하려는 경우 유용합니다.

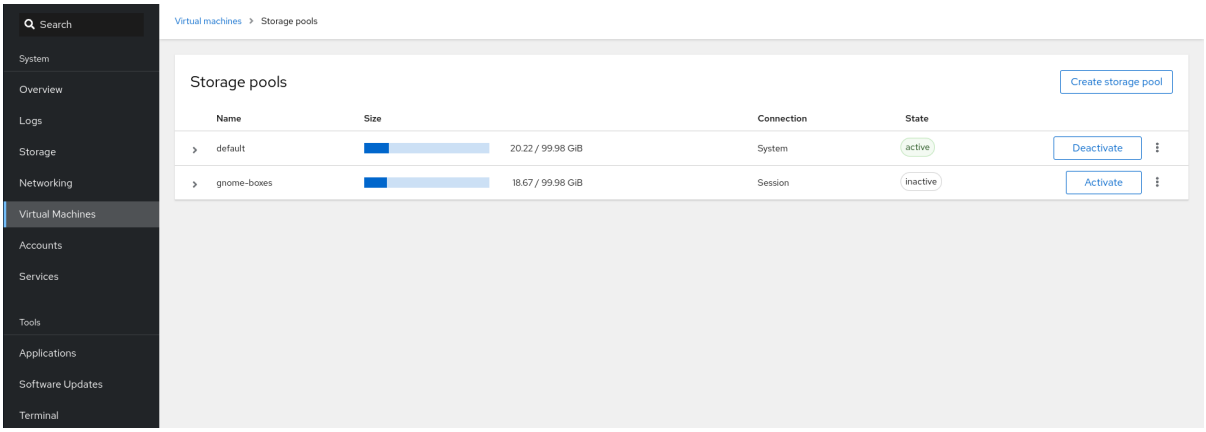
사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 시스템에 설치되어 있습니다.

절차

1. **RHEL** 웹 콘솔의 **Virtual Machines** (가상 머신) 탭에서 스토리지 풀 을 클릭합니다.

   구성된 스토리지 풀 목록을 표시하는 스토리지 풀 창이 표시됩니다(있는 경우).

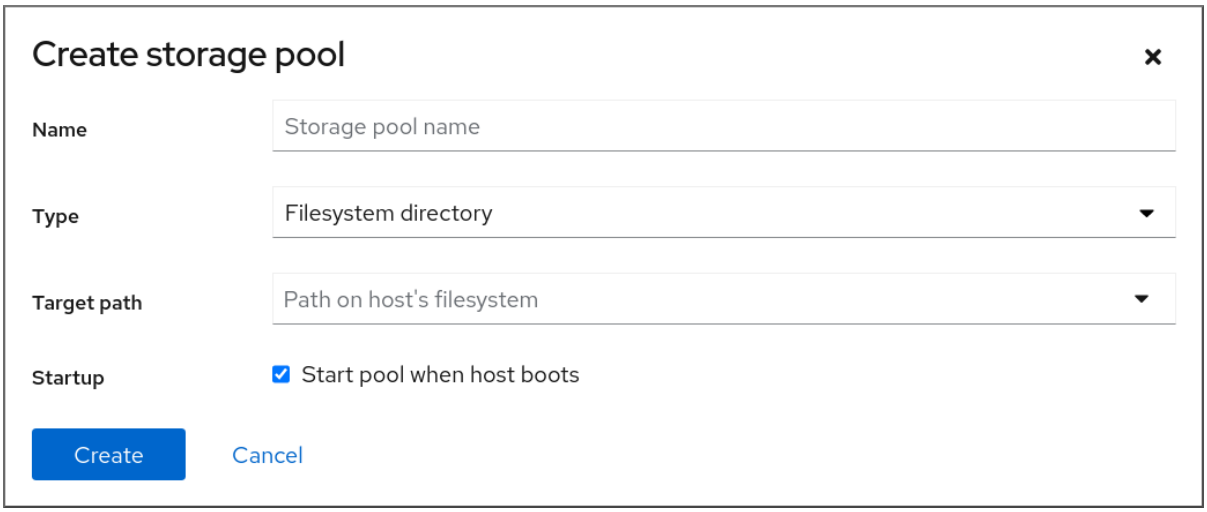


2. 스토리지 풀 생성을 클릭합니다.

   스토리지 풀 생성 대화 상자가 나타납니다.

3. 스토리지 풀의 이름을 입력합니다.

4. *유형 드롭다운 메뉴에서 **Filesystem** 디렉터리 를 선택합니다.*



참고

*드롭다운 메뉴에 **Filesystem** 디렉터리 옵션이 표시되지 않으면 하이퍼바이저에서 디렉터리 기반 스토리지 풀을 지원하지 않습니다.*

5. *다음 정보를 입력합니다.*

- *대상 경로 - 디렉터리에서 지원하는 스토리지 풀 유형 소스(예: **dir** 또는 **netfs**)입니다.*
- *시작 - 호스트가 부팅될 때 스토리지 풀이 시작되는지 여부입니다.*

6. *생성을 클릭합니다.*

스토리지 풀이 생성되고 스토리지 풀 생성 대화 상자가 닫히고 새 스토리지 풀이 스토리지 풀 목록에 나타납니다.

추가 리소스

- *스토리지 풀 이해*
- *웹 콘솔을 사용하여 스토리지 풀 정보 보기*

14.3.3. 웹 콘솔을 사용하여 NFS 기반 스토리지 풀 생성

NFS 기반 스토리지 풀은 서버에서 호스팅되는 파일 시스템을 기반으로 합니다.

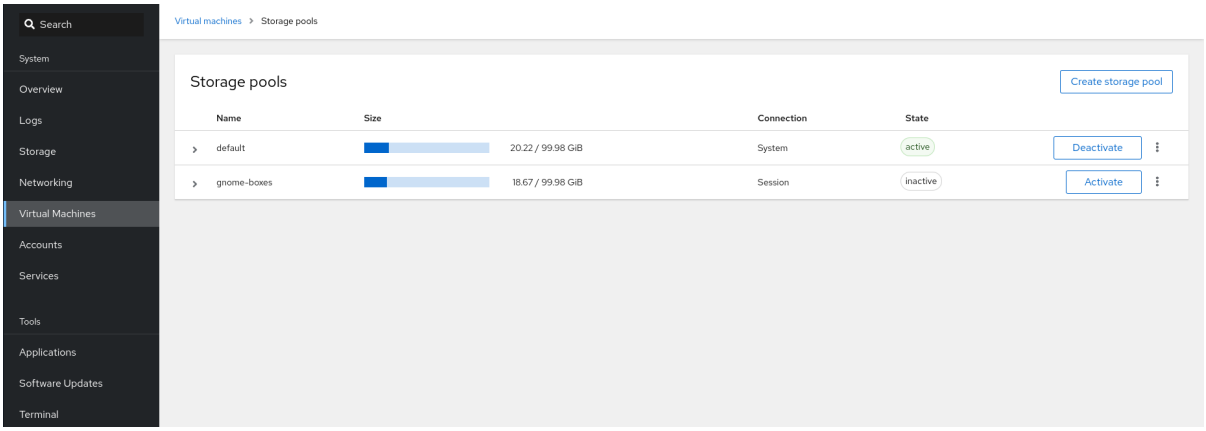
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. RHEL 웹 콘솔의 Virtual Machines (가상 머신) 탭에서 스토리지 풀 을 클릭합니다.

   구성된 스토리지 풀 목록을 표시하는 스토리지 풀 창이 표시됩니다(있는 경우).

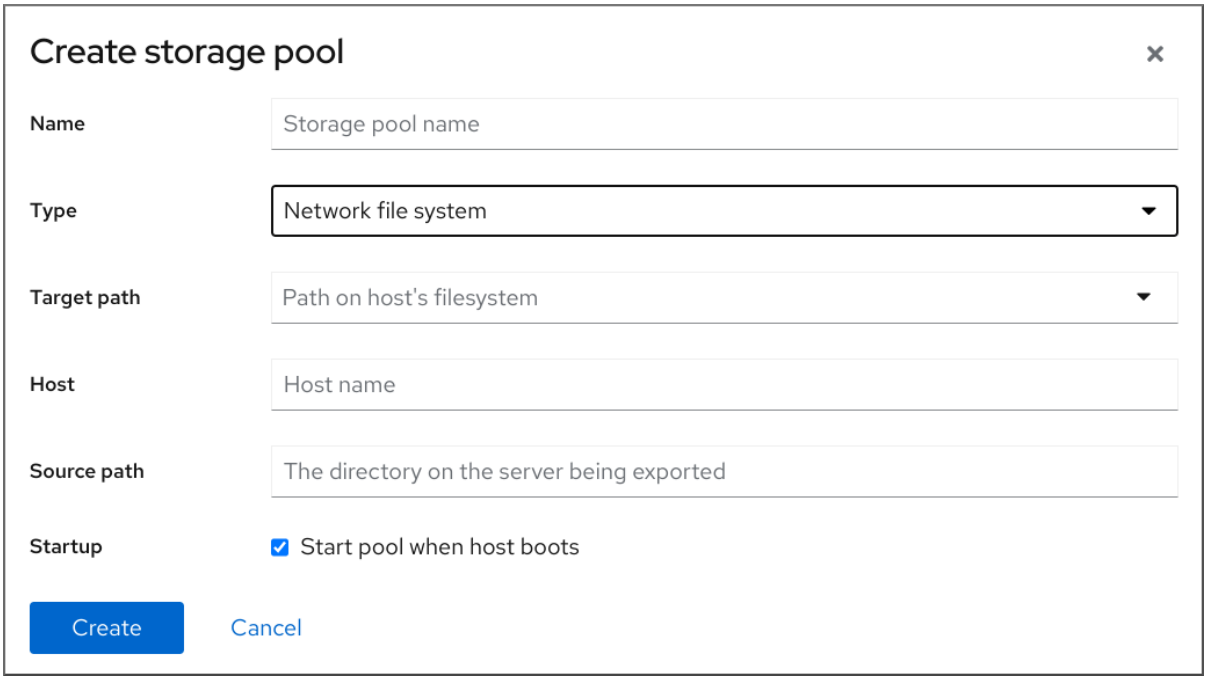


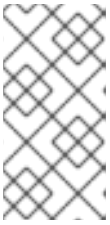
2. 스토리지 풀 생성을 클릭합니다.

   스토리지 풀 생성 대화 상자가 나타납니다.

3. 스토리지 풀의 이름을 입력합니다.

4. 유형 드롭다운 메뉴에서 네트워크 파일 시스템을 선택합니다.





참고

*드롭다운 메뉴에 네트워크 파일 시스템 옵션이 표시되지 않으면 하이퍼바이저에서 **nfs** 기반 스토리지 풀을 지원하지 않습니다.*

5. *정보의 나머지 부분을 입력하십시오:*

- ***target path -** 대상을 지정하는 경로입니다. 스토리지 풀에 사용되는 경로입니다.*
- ***host -** 마운트 지점이 있는 네트워크 서버의 호스트 이름입니다. 호스트 이름 또는 **IP** 주소일 수 있습니다.*
- ***source path -** 네트워크 서버에서 사용되는 디렉터리입니다.*
- *시작 - 호스트가 부팅될 때 스토리지 풀이 시작되는지 여부입니다.*

6. *생성을 클릭합니다.*

스토리지 풀이 생성됩니다. 스토리지 풀 생성 대화 상자가 닫히고 새 스토리지 풀은 스토리지 풀 목록에 나타납니다.

추가 리소스

- 스토리지 풀 이해
- 웹 콘솔을 사용하여 스토리지 풀 정보 보기

14.3.4. 웹 콘솔을 사용하여 iSCSI 기반 스토리지 풀 생성

iSCSI 기반 스토리지 풀은 데이터 스토리지 기능을 연결하는 IP 기반 스토리지 네트워킹 표준인 iSCSI(Internet Small Computer Systems Interface)를 기반으로 합니다.

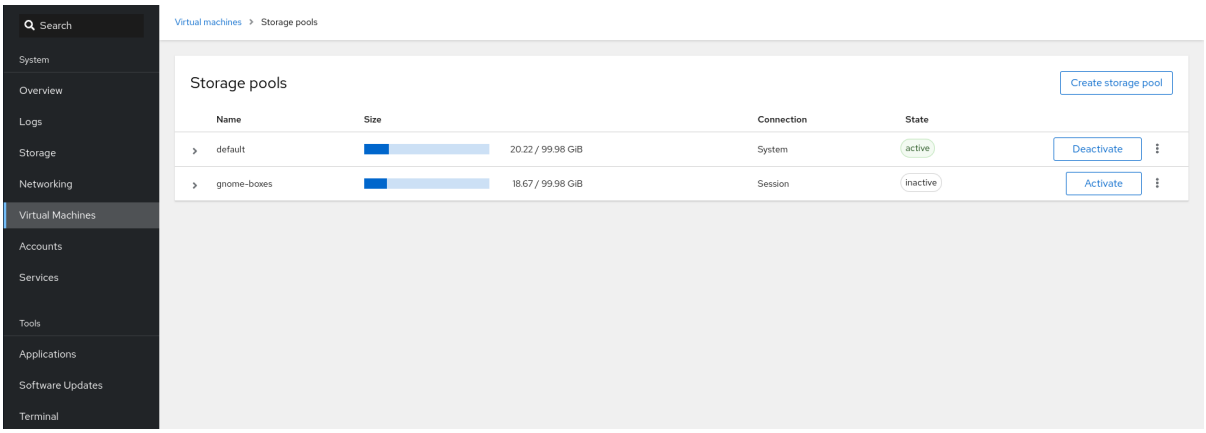
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. RHEL 웹 콘솔의 Virtual Machines (가상 머신) 탭에서 스토리지 풀 을 클릭합니다.

   구성된 스토리지 풀 목록을 표시하는 스토리지 풀 창이 표시됩니다(있는 경우).

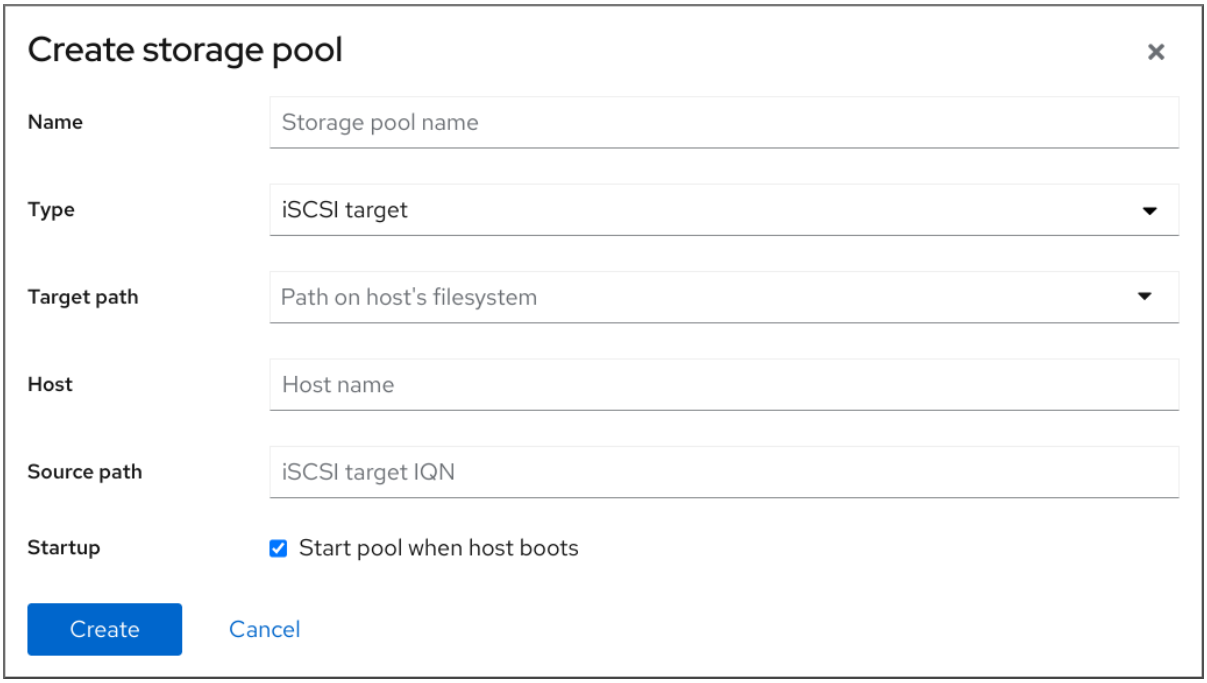


2. 스토리지 풀 생성을 클릭합니다.

   스토리지 풀 생성 대화 상자가 나타납니다.

3. 스토리지 풀의 이름을 입력합니다.

4. *유형 드롭다운 메뉴에서 **iSCSI** 대상 을 선택합니다.*



5. *정보의 나머지 부분을 입력하십시오:*

- *대상 경로 - 대상을 지정하는 경로입니다. 스토리지 풀에 사용되는 경로입니다.*
- ***host - ISCSI** 서버의 호스트 이름 또는 **IP** 주소입니다.*
- *소스 경로 - **iSCSI** 대상의 고유한 **IQN(iSCSI Qualified Name)**입니다.*
- *시작 - 호스트가 부팅될 때 스토리지 풀이 시작되는지 여부입니다.*

6. *생성을 클릭합니다.*

스토리지 풀이 생성됩니다. 스토리지 풀 생성 대화 상자가 닫히고 새 스토리지 풀은 스토리지 풀 목록에 나타납니다.

추가 리소스

- *스토리지 풀 이해*

- 웹 콘솔을 사용하여 스토리지 풀 정보 보기

14.3.5. 웹 콘솔을 사용하여 디스크 기반 스토리지 풀 생성

디스크 기반 스토리지 풀은 전체 디스크 파티션을 사용합니다.

주의

- 사용 중인 **libvirt** 버전에 따라 디스크를 스토리지 풀에 전용으로 사용하면 디스크 장치에 현재 저장된 모든 데이터를 다시 포맷하고 지울 수 있습니다. 스토리지 풀을 생성하기 전에 스토리지 장치에 데이터를 백업하는 것이 좋습니다.

- 전체 디스크 또는 블록 장치가 **VM**에 전달되면 **VM**에서 파티션을 지정하거나 자체 **LVM** 그룹을 생성합니다. 이로 인해 호스트 시스템에서 이러한 파티션 또는 **LVM** 그룹을 감지하고 오류가 발생할 수 있습니다.

  이러한 오류는 파티션 또는 **LVM** 그룹을 수동으로 생성하여 **VM**에 전달할 때도 발생할 수 있습니다.

  이러한 오류를 방지하려면 대신 파일 기반 스토리지 풀을 사용합니다.

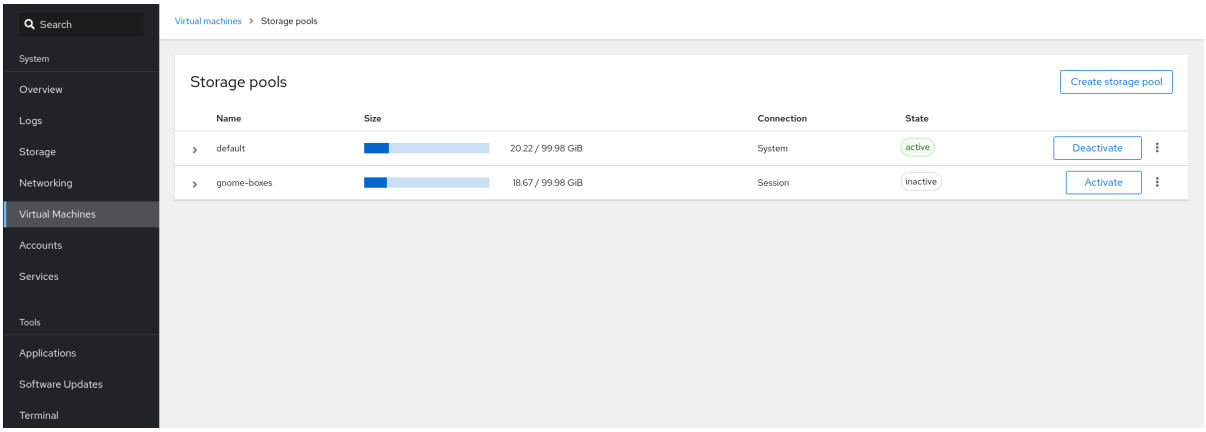
사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 시스템에 설치되어 있습니다.

절차

1. **RHEL** 웹 콘솔의 **Virtual Machines** (가상 머신) 탭에서 스토리지 풀 을 클릭합니다.

   구성된 스토리지 풀 목록을 표시하는 스토리지 풀 창이 표시됩니다(있는 경우).

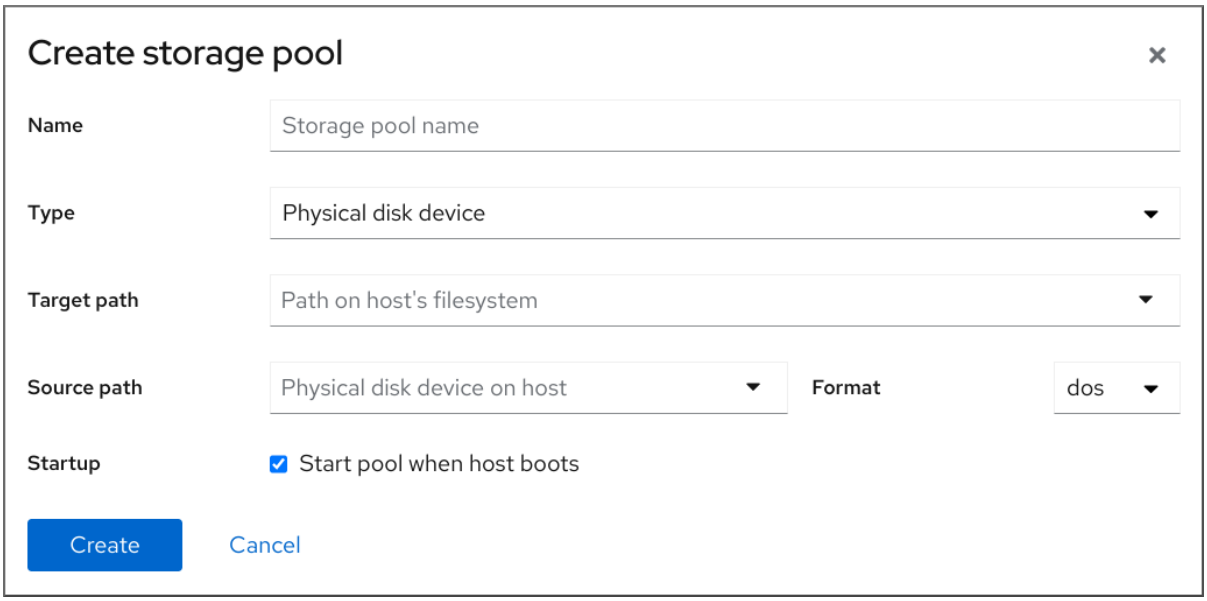


2. *스토리지 풀 생성을 클릭합니다.*

   스토리지 풀 생성 대화 상자가 나타납니다.

3. *스토리지 풀의 이름을 입력합니다.*

4. *유형 드롭다운 메뉴에서 물리적 디스크 장치를 선택합니다.*



참고

드롭다운 메뉴에 물리 디스크 장치 옵션이 표시되지 않으면 하이퍼바이저에서 디스크 기반 스토리지 풀을 지원하지 않습니다.

5. *정보의 나머지 부분을 입력하십시오:*

- *대상 경로 - 대상 장치를 지정하는 경로입니다. 스토리지 풀에 사용되는 경로입니다.*
- *소스 경로 - 스토리지 장치를 지정하는 경로입니다. 예를 들면 /dev/sdb 입니다.*
- ***Format** - 파티션 테이블의 유형입니다.*
- *시작 - 호스트가 부팅될 때 스토리지 풀이 시작되는지 여부입니다.*

6. *생성을 클릭합니다.*

스토리지 풀이 생성됩니다. 스토리지 풀 생성 대화 상자가 닫히고 새 스토리지 풀은 스토리지 풀 목록에 나타납니다.

추가 리소스

- *스토리지 풀 이해*
- *웹 콘솔을 사용하여 스토리지 풀 정보 보기*

14.3.6. 웹 콘솔을 사용하여 LVM 기반 스토리지 풀 생성

***LVM** 기반 스토리지 풀은 **LVM(Logical Volume Manager)**을 사용하여 관리할 수 있는 볼륨 그룹을 기반으로 합니다. 볼륨 그룹은 단일 스토리지 구조를 생성하는 여러 물리 볼륨을 조합한 것입니다.*

참고

- LVM 기반 스토리지 풀은 LVM의 완전한 유연성을 제공하지 않습니다.
- libvirt 는 씬 논리 볼륨을 지원하지만 씬 스토리지 풀의 기능은 제공하지 않습니다.
- LVM 기반 스토리지 풀에는 전체 디스크 파티션이 필요합니다. virsh 명령을 사용하여 새 파티션 또는 장치를 활성화하면 파티션이 포맷되고 모든 데이터가 삭제됩니다. 이 절차에서와 같이 호스트의 기존 볼륨 그룹을 사용하는 경우 아무것도 삭제되지 않습니다.
- 여러 장치가 있는 볼륨 그룹을 만들려면 대신 LVM 유틸리티를 사용하십시오. LVM을 사용하여 Linux에서 볼륨 그룹을 만드는 방법을 참조하십시오.

볼륨 그룹에 대한 자세한 내용은 Red Hat Enterprise Linux Logical Volume Manager 관리 가이드 를 참조하십시오.

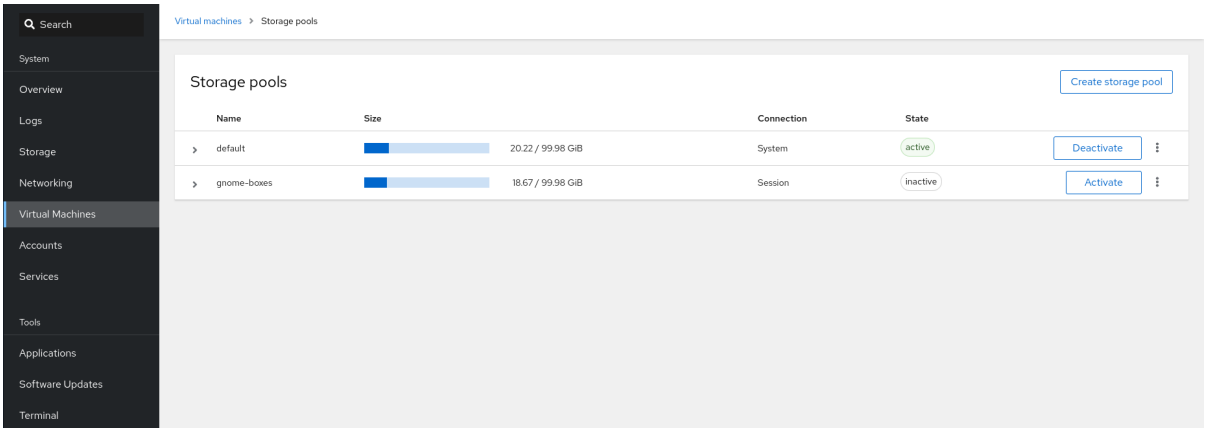
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. RHEL 웹 콘솔의 Virtual Machines (가상 머신) 탭에서 스토리지 풀 을 클릭합니다.

구성된 스토리지 풀 목록을 표시하는 스토리지 풀 창이 표시됩니다(있는 경우).

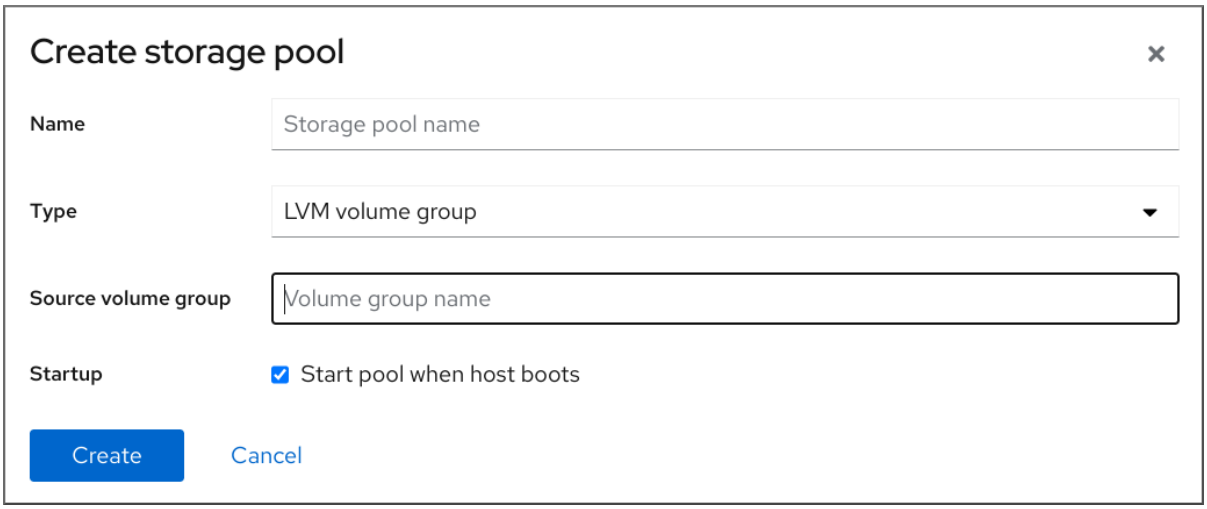


2. *스토리지 풀 생성을 클릭합니다.*

   스토리지 풀 생성 대화 상자가 나타납니다.

3. *스토리지 풀의 이름을 입력합니다.*

4. *유형 드롭다운 메뉴에서 **LVM** 볼륨 그룹 을 선택합니다.*



참고

*드롭다운 메뉴에 **LVM** 볼륨 그룹 옵션이 표시되지 않으면 하이퍼바이저에서 **LVM** 기반 스토리지 풀을 지원하지 않습니다.*

5. *정보의 나머지 부분을 입력하십시오:*

   - *소스 볼륨 그룹 - 사용하려는 **LVM** 볼륨 그룹의 이름입니다.*
   - *시작 - 호스트가 부팅될 때 스토리지 풀이 시작되는지 여부입니다.*

6. *생성을 클릭합니다.*

스토리지 풀이 생성됩니다. 스토리지 풀 생성 대화 상자가 닫히고 새 스토리지 풀은 스토리지 풀 목록에 나타납니다.

추가 리소스

- 스토리지 풀 이해
- 웹 콘솔을 사용하여 스토리지 풀 정보 보기

14.3.7. 웹 콘솔을 사용하여 스토리지 풀 제거

스토리지 풀을 제거하여 호스트 또는 네트워크에서 리소스를 확보하여 시스템 성능을 향상시킬 수 있습니다. 스토리지 풀을 삭제하면 다른 VM(가상 머신)에서 사용할 수 있는 리소스가 확보됩니다.

중요

명시적으로 지정하지 않는 한 스토리지 풀을 삭제해도 해당 풀 내의 스토리지 볼륨이 동시에 삭제되지 않습니다.

RHEL 웹 콘솔을 사용하여 스토리지 풀을 삭제하려면 다음 절차를 참조하십시오.

참고

삭제하지 않고 스토리지 풀을 일시적으로 비활성화하려면 웹 콘솔을 사용하여 스토리지 풀 비활성화를참조하십시오.

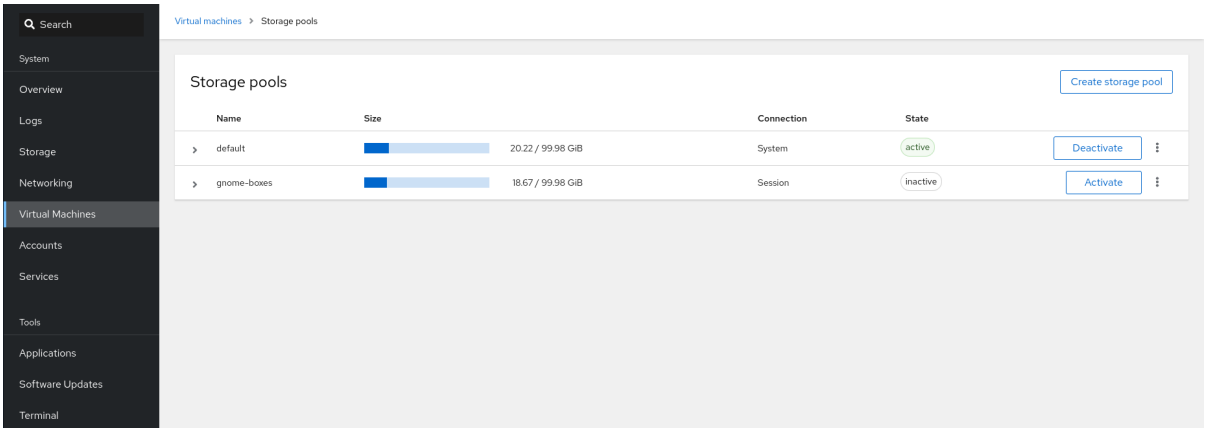
사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 풀과 함께 스토리지 볼륨을 삭제하려면 먼저 VM에서 디스크를 분리해야 합니다.

절차

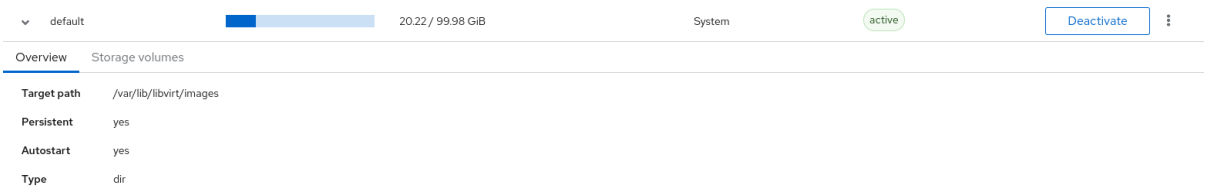
1. 가상 머신 탭에서 Storage Pools 를 클릭합니다.

   구성된 스토리지 풀 목록을 보여주는 스토리지 풀 창이 표시됩니다.



2. ***Storage Pools*** *(스토리지 풀) 창에서 삭제할 스토리지 풀을 클릭합니다.*

행이 확장되어 선택한 스토리지 풀에 대한 기본 정보와 스토리지 풀을 비활성화하거나 삭제하기 위한 제어에 대한 기본 정보가 포함된 개요 창이 표시됩니다.



3. *메뉴 버튼을 클릭하고 삭제를 클릭합니다.*

확인 대화 상자가 나타납니다.

4. *선택 사항: 풀 내의 스토리지 볼륨을 삭제하려면 대화 상자에서 확인란을 선택합니다.*

5. *삭제를 클릭합니다.*

스토리지 풀이 삭제됩니다. 이전 단계에서 확인란을 선택한 경우 연결된 스토리지 볼륨도 삭제됩니다.

추가 리소스

- *스토리지 풀 이해*
- *웹 콘솔을 사용하여 스토리지 풀 정보 보기*

14.3.8. 웹 콘솔을 사용하여 스토리지 풀 비활성화

스토리지 풀을 영구적으로 삭제하지 않으려면 대신 일시적으로 비활성화할 수 있습니다.

스토리지 풀을 비활성화하면 해당 풀에 새 볼륨을 만들 수 없습니다. 그러나 해당 풀에 볼륨이 있는 VM(가상 머신)은 계속 실행됩니다. 예를 들어 이 기능은 풀에서 생성할 수 있는 볼륨 수를 제한하여 시스템 성능을 향상시킬 수 있는 여러 가지 이유로 유용합니다.

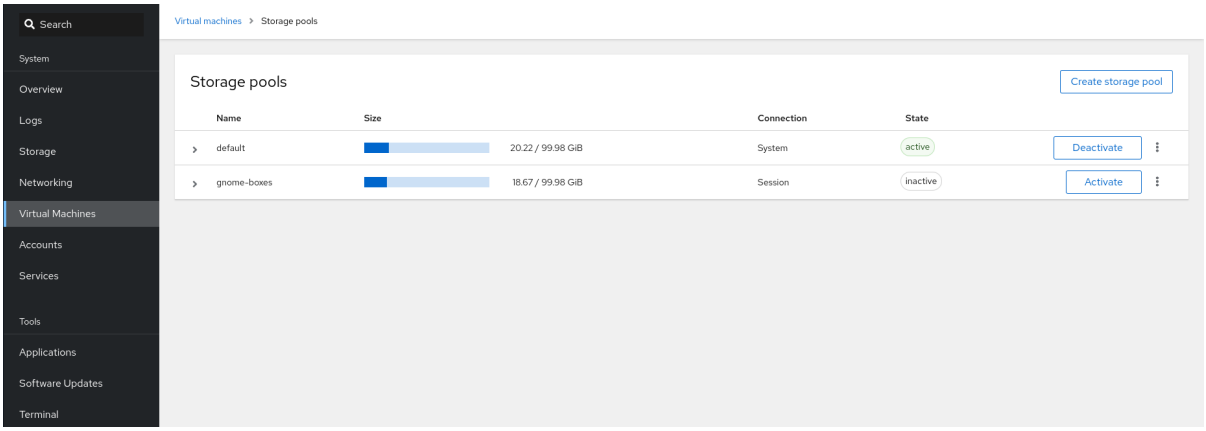
RHEL 웹 콘솔을 사용하여 스토리지 풀을 비활성화하려면 다음 절차를 참조하십시오.

사전 요구 사항

- *웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.*

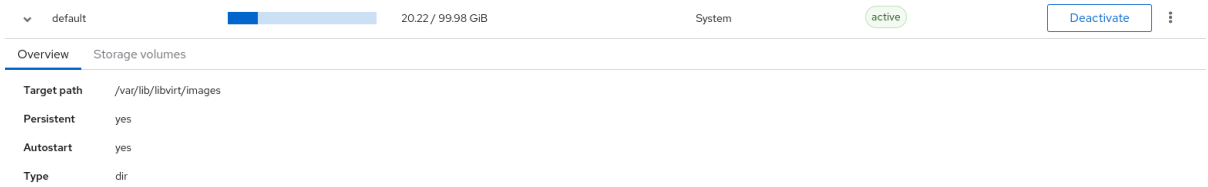
절차

1. *Virtual Machines(가상 머신) 탭 상단에서 Storage Pools (스토리지 풀)를 클릭합니다. 구성된 스토리지 풀 목록을 보여주는 스토리지 풀 창이 표시됩니다.*



2. *Storage Pools (스토리지 풀) 창에서 비활성화할 스토리지 풀을 클릭합니다.*

행이 확장되어 선택한 스토리지 풀 및 VM 비활성화 및 삭제에 대한 제어에 대한 기본 정보가 포함된 개요 창이 표시됩니다.



3. *비활성화 를 클릭합니다.*

   스토리지 풀이 비활성화됩니다.

추가 리소스

- *스토리지 풀 이해*
- *웹 콘솔을 사용하여 스토리지 풀 정보 보기*

14.4. 스토리지 풀 생성을 위한 매개변수

*필요한 스토리지 풀 유형에 따라 **XML** 구성 파일을 수정하고 특정 유형의 스토리지 풀을 정의할 수 있습니다. 이 섹션에서는 다양한 유형의 스토리지 풀을 생성하는 데 필요한 **XML** 매개 변수와 예제를 제공합니다.*

14.4.1. 디렉터리 기반 스토리지 풀 매개변수

***XML** 구성 파일을 사용하여 디렉터리 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.*

***virsh pool-define** 명령을 사용하여 지정된 파일의 **XML** 구성을 기반으로 스토리지 풀을 생성할 수 있습니다. 예를 들면 다음과 같습니다.*

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

매개 변수

*다음 표에서는 디렉터리 기반 스토리지 풀의 **XML** 파일에 필요한 매개 변수 목록을 제공합니다.*

표 14.1. 디렉터리 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀의 유형	**<pool type='dir'>**
스토리지 풀의 이름	**<name>*name*</name>**
대상을 지정하는 경로입니다. 스토리지 풀에 사용되는 경로입니다.	**<target> <path>*target_path*</path> </target>**

예제

다음은 /guest_images 디렉터리를 기반으로 스토리지 풀의 XML 파일의 예입니다.

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

추가 리소스

- *CLI를 사용하여 디렉터리 기반 스토리지 풀 생성*

14.4.2. 디스크 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 디스크 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 XML 구성을 기반으로 스토리지 풀을 생성할 수 있습니다. 예를 들면 다음과 같습니다.

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

매개 변수

다음 표에서는 디스크 기반 스토리지 풀의 XML 파일에 필요한 매개 변수 목록을 제공합니다.

표 14.2. 디스크 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀의 유형	**<pool type='disk'>**
스토리지 풀의 이름	**<name>*name*</name>**
스토리지 장치를 지정하는 경로입니다. 예를 들면 **/dev/sdb** 입니다.	**<source>** **<path>*source_path*</path>** **</source>**
대상 장치를 지정하는 경로입니다. 스토리지 풀에 사용되는 경로입니다.	**<target>** **<path>*target_path*</path>** **</target>**

예제

*다음은 디스크 기반 스토리지 풀에 대한 **XML** 파일의 예입니다.*

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

추가 리소스

- *CLI를 사용하여 디스크 기반 스토리지 풀 생성*

14.4.3. 파일 시스템 기반 스토리지 풀 매개변수

***XML** 구성 파일을 사용하여 파일 시스템 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.*

***virsh pool-define** 명령을 사용하여 지정된 파일의 **XML** 구성을 기반으로 스토리지 풀을 생성할 수 있습니다. 예를 들면 다음과 같습니다.*

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

매개 변수

*다음 표에서는 파일 시스템 기반 스토리지 풀의 **XML** 파일에 필요한 매개 변수 목록을 제공합니다.*

*표 **14.3.** 파일 시스템 기반 스토리지 풀 매개변수*

설명	XML
스토리지 풀의 유형	**<pool type='fs'>**
스토리지 풀의 이름	**<name>*name*</name>**
파티션을 지정하는 경로입니다. 예를 들면 **/dev/sdc1**	**<source>** **<device path=*device_path* />**
파일 시스템 유형(예: **ext4**)	**<format type=*fs_type* />** **</source>**
대상을 지정하는 경로입니다. 스토리지 풀에 사용되는 경로입니다.	**<target>** **<path>*path-to-pool*</path>** **</target>**

예제

*다음은 **/dev/sdc1** 파티션을 기반으로 하는 스토리지 풀에 대한 **XML** 파일의 예입니다.*

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

추가 리소스

- *CLI를 사용하여 파일 시스템 기반 스토리지 풀 생성*

14.4.4. iSCSI 기반 스토리지 풀 매개변수

***XML** 구성 파일을 사용하여 **iSCSI** 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.*

virsh pool-define 명령을 사용하여 지정된 파일의 XML 구성을 기반으로 스토리지 풀을 생성할 수 있습니다. 예를 들면 다음과 같습니다.

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

매개 변수

다음 표에서는 iSCSI 기반 스토리지 풀의 XML 파일에 필요한 매개 변수 목록을 제공합니다.

표 14.4. iSCSI 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀의 유형	**<pool type='iscsi'>**
스토리지 풀의 이름	**<name>*name*</name>**
호스트 이름	**<source>** ** <host name=*hostname* />**
iSCSI IQN	** <device path= *iSCSI_IQN* />** **</source>**
대상을 지정하는 경로입니다. 스토리지 풀에 사용되는 경로입니다.	**<target>** ** <path>*/dev/disk/by-path*</path>** **</target>**
[선택 사항] iSCSI 이니시에이터의 IQN입니다. 이는 ACL이 LUN을 특정 이니시에이터로 제한할 때만 필요합니다.	**<initiator>** ** <iqn name='*initiator0*' />** **</initiator>**

참고

iSCSI 이니시에이터의 IQN은 virsh find-storage-pool-sources-as iscsi 명령을 사용하여 확인할 수 있습니다.

예제

다음은 지정된 iSCSI 장치를 기반으로 하는 스토리지 풀의 XML 파일의 예입니다.

```
<pool type='iscsi'>
```

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

추가 리소스

- *CLI를 사용하여 iSCSI 기반 스토리지 풀 생성*

14.4.5. LVM 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 LVM 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 XML 구성을 기반으로 스토리지 풀을 생성할 수 있습니다. 예를 들면 다음과 같습니다.

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

매개 변수

다음 표에서는 LVM 기반 스토리지 풀의 XML 파일에 필요한 매개 변수 목록을 제공합니다.

표 14.5. LVM 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀의 유형	**<pool type='logical'>**
스토리지 풀의 이름	**<name>*name*</name>**
스토리지 풀의 장치 경로입니다.	**<source>** ** <device path='*device_path*' />`**
볼륨 그룹의 이름	** <name>*VG-name*</name>**
가상 그룹 형식	** <format type='lvm2' />** **</source>**

설명	XML
대상 경로	<target> <path=*target_path* /> </target>

참고

논리 볼륨 그룹이 여러 개의 디스크 파티션으로 설정된 경우 여러 소스 장치가 나열될 수 있습니다. 예를 들면 다음과 같습니다.

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

예제

*다음은 지정된 **LVM**을 기반으로 스토리지 풀에 대한 **XML** 파일의 예입니다.*

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

추가 리소스

- *CLI를 사용하여 LVM 기반 스토리지 풀 생성*

14.4.6. NFS 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 NFS 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 XML 구성을 기반으로 스토리지 풀을 생성할 수 있습니다. 예를 들면 다음과 같습니다.

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

매개 변수

다음 표에서는 NFS 기반 스토리지 풀의 XML 파일에 필요한 매개 변수 목록을 제공합니다.

표 14.6. NFS 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀의 유형	**<pool type='netfs'>**
스토리지 풀의 이름	**<name>*name*</name>**
마운트 지점이 있는 네트워크 서버의 호스트 이름입니다. 호스트 이름 또는 IP 주소일 수 있습니다.	**<source>** ** <host name=*hostname* />**
스토리지 풀 형식	다음 중 하나: **<format type='nfs' />** **<format type='cifs' />**
네트워크 서버에서 사용되는 디렉터리	** <dir path=*source_path* />** **</source>**
대상을 지정하는 경로입니다. 스토리지 풀에 사용되는 경로입니다.	**<target>** ** <path>*target_path*</path>** **</target>**

예제

다음은 file_server NFS 서버의 /home/net_mount 디렉터리를 기반으로 스토리지 풀에 대한 XML 파일의 예입니다.

```
<pool type='netfs'>
  <name>nfspool</name>
  <source>
    <host name='file_server'/>
    <format type='nfs'/>
    <dir path='/home/net_mount'/>
  </source>
  <target>
```

```
    <path>/var/lib/libvirt/images/nfspool</path>
  </target>
</pool>
```

추가 리소스

- CLI를 사용하여 NFS 기반 스토리지 풀 생성

14.4.7. vHBA 장치를 사용한 SCSI 기반 스토리지 풀의 매개변수

VHBA(가상 호스트 어댑터 버스) 장치를 사용하는 SCSi 기반 스토리지 풀에 대한 XML 구성 파일을 생성하거나 수정하려면 XML 구성 파일에 특정 필수 매개 변수를 포함해야 합니다. 필수 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 XML 구성을 기반으로 스토리지 풀을 생성할 수 있습니다. 예를 들면 다음과 같습니다.

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

매개 변수

다음 표에서는 vHBA를 사용한 SCSI 기반 스토리지 풀의 XML 파일에 필요한 매개 변수 목록을 제공합니다.

표 14.7. vHBA 장치를 사용한 SCSI 기반 스토리지 풀의 매개변수

설명	XML
스토리지 풀의 유형	**<pool type='scsi'>**
스토리지 풀의 이름	**<name>*name*</name>**
vHBA의 식별자입니다. **상위** 속성은 선택 사항입니다.	**<source> <adapter type='fc_host' [parent=*parent_scsi_device*] wwnn='*WWNN*' wwpn='*WWPN*'/> </source>**
대상 경로입니다. 스토리지 풀에 사용되는 경로입니다.	**<target> <path=*target_path* /> </target>**

중요

< path > 필드가 /dev/ 인 경우 libvirt 는 볼륨 장치 경로에 대한 고유한 단축 장치 경로를 생성합니다. 예를 들면 /dev/sdc 입니다. 그렇지 않으면 물리적 호스트 경로가 사용됩니다. 예를 들어 /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0. 고유한 짧은 장치 경로를 사용하면 여러 스토리지 풀에서 동일한 볼륨을 여러 VM(가상 머신)에 나열할 수 있습니다. 여러 VM에서 물리적 호스트 경로를 사용하는 경우 중복 장치 유형 경고가 발생할 수 있습니다.

참고

상위 속성은 < adapter > 필드에서 사용하여 NPIV LUN을 다양한 경로로 사용할 수 있는 물리 HBA 부모를 식별할 수 있습니다. 이 필드 scsi_hostN 은 vports 및 max_vports 속성과 결합하여 상위 ID를 완료합니다. 상위 , parent _wwnn,parent_wwpn 또는 parent_fabric_wwn 속성은 호스트가 동일한 HBA를 재부팅한 후 다양한 수준의 보증을 제공합니다.

- 부모가 지정되지 않은 경우 libvirt 는 NPIV를 지원하는 첫 번째 scsi_hostN 어댑터를 사용합니다.
- 부모 만 지정된 경우 구성에 추가 SCSI 호스트 어댑터가 추가되는 경우 문제가 발생할 수 있습니다.
- 호스트가 동일한 HBA를 재부팅한 후 parent_wwnn 또는 parent_wwpn 을 지정하면 됩니다.
- parent_fabric_wwn 을 사용하는 경우, 호스트가 scsi_hostN 과 관계없이 동일한 패브릭에서 HBA를 재부팅한 후.

예제

다음은 vHBA를 사용한 SCSI 기반 스토리지 풀의 XML 파일의 예입니다.

- HBA의 유일한 스토리지 풀인 스토리지 풀:

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' wwnn='5001a4a93526d0a1' wwpn='5001a4ace3ee047d'/>
  </source>
```

```
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

- 단일 **vHBA**를 사용하고 부모 특성을 사용하여 **SCSI** 호스트 장치를 식별하는 여러 스토리지 풀 중 하나인 스토리지 풀:

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' parent='scsi_host3' wwnn='5001a4a93526d0a1'
wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

추가 리소스

- **CLI**를 사용하여 **vHBA** 장치로 **SCSI** 기반 스토리지 풀 생성

14.5. CLI를 사용하여 가상 머신 스토리지 볼륨 관리

CLI를 사용하여 스토리지 볼륨의 다음 측면을 관리하여 **VM**(가상 머신)에 스토리지를 할당할 수 있습니다.

- 스토리지 볼륨 정보 보기
- 스토리지 볼륨 생성
- 스토리지 볼륨 삭제

14.5.1. CLI를 사용하여 스토리지 볼륨 정보 보기

명령줄을 사용하면 호스트에서 사용 가능한 모든 스토리지 풀 목록과 지정된 스토리지 풀에 대한 세부 정보를 볼 수 있습니다.

절차

1. *virsh vol-list 명령을 사용하여 지정된 스토리지 풀의 스토리지 볼륨을 나열합니다.*

```
# virsh vol-list --pool RHEL-Storage-Pool --details
 Name              Path                                    Type   Capacity  Allocation
--------------------------------------------------------------------------------------------
 .bash_history     /home/VirtualMachines/.bash_history     file  18.70 KiB   20.00 KiB
 .bash_logout      /home/VirtualMachines/.bash_logout      file   18.00 B    4.00 KiB
 .bash_profile     /home/VirtualMachines/.bash_profile     file  193.00 B    4.00 KiB
 .bashrc           /home/VirtualMachines/.bashrc           file   1.29 KiB    4.00 KiB
 .git-prompt.sh    /home/VirtualMachines/.git-prompt.sh    file  15.84 KiB   16.00 KiB
 .gitconfig        /home/VirtualMachines/.gitconfig        file   167.00 B    4.00 KiB
 RHEL_Volume.qcow2 /home/VirtualMachines/RHEL8_Volume.qcow2 file  60.00 GiB
13.93 GiB
```

2. *virsh vol-info 명령을 사용하여 지정된 스토리지 풀의 스토리지 볼륨을 나열합니다.*

```
# vol-info --pool RHEL-Storage-Pool --vol RHEL_Volume.qcow2
Name:           RHEL_Volume.qcow2
Type:           file
Capacity:       60.00 GiB
Allocation:     13.93 GiB
```

14.5.2. CLI를 사용하여 스토리지 볼륨 생성 및 할당

디스크 이미지를 가져와서 가상 머신(VM)에 가상 디스크로 연결하려면 스토리지 볼륨을 생성하고 해당 XML 구성을 VM에 할당합니다.

사전 요구 사항

- *할당되지 않은 공간이 있는 스토리지 풀이 호스트에 있습니다.*
  - *확인하려면 호스트의 스토리지 풀을 나열합니다.*

```
# virsh pool-list --details

Name              State    Autostart  Persistent  Capacity    Allocation  Available
-------------------------------------------------------------------------------------------
default           running  yes        yes         48.97 GiB   36.34 GiB   12.63 GiB
Downloads         running  yes        yes         175.92 GiB  121.20 GiB  54.72 GiB
VM-disks          running  yes        yes         175.92 GiB  121.20 GiB  54.72 GiB
```

  - *기존 스토리지 풀이 없는 경우 하나를 생성합니다. 자세한 내용은 가상 머신의 스토리지 관리를 참조하십시오.*

절차

1. ***virsh vol-create-as*** *명령을 사용하여 스토리지 볼륨을 생성합니다. 예를 들어* ***guest-images-fs*** *스토리지 풀을 기반으로* ***20GB qcow2*** *볼륨을 생성하려면 다음을 수행합니다.*

```
# virsh vol-create-as --pool guest-images-fs --name vm-disk1 --capacity 20 --format
qcow2
```

중요: 특정 스토리지 풀 유형은 ***virsh vol-create-as*** *명령을 지원하지 않으며 대신 스토리지 볼륨을 생성하려면 특정 프로세스가 필요합니다.*

- ***iSCSI*** *기반 -* ***iSCSI*** *서버에서 미리* ***iSCSI LUN*** *준비.*
- ***multipath*** *기반 -* ***multipathd*** *명령을 사용하여 다중 경로 를 준비하거나 관리합니다.*
- ***vHBA*** *기반 -* ***파이버 채널 카드*** *준비*

2. ***XML*** *파일을 만들고 여기에 다음 행을 추가합니다. 이 파일은 스토리지 볼륨을* ***VM***에 *디스크로 추가하는 데 사용됩니다.*

```
<disk type='volume' device='disk'>
  <driver name='qemu' type='qcow2'/>
  <source pool='guest-images-fs' volume='vm-disk1'/>
  <target dev='hdk' bus='ide'/>
</disk>
```

이 예제에서는 이전 단계에서 만든 ***vm-disk1*** *볼륨을 사용하는 가상 디스크를 지정하고 볼륨을* ***ide*** *버스에서 디스크* ***hdk*** *로 설정하도록 설정합니다. 해당 매개변수를 환경에 적절하게 수정합니다.*

중요: 특정 스토리지 풀 유형의 경우 스토리지 볼륨 디스크를 설명하려면 다른 ***XML*** *형식을 사용해야 합니다.*

- *다중 경로 기반 풀의 경우:*

```
<disk type='block' device='disk'>
```

```
<driver name='qemu' type='raw'/>
<source dev='/dev/mapper/mpatha' />
<target dev='sda' bus='scsi'/>
</disk>
```

- **RBD** 기반 스토리지 풀의 경우:

```
<disk type='network' device='disk'>
  <driver name='qemu' type='raw'/>
  <source protocol='rbd' name='pool/image'>
    <host name='mon1.example.org' port='6321'/>
  </source>
  <target dev='vdc' bus='virtio'/>
</disk>
```

3. **XML** 파일을 사용하여 스토리지 볼륨을 **VM**에 디스크로 할당합니다. 예를 들어 **~/vm-disk1.xml** 에 정의된 디스크를 **testguest1 VM**에 할당하려면 다음을 실행합니다.

```
# attach-device --config testguest1 ~/vm-disk1.xml
```

검증

- **VM**의 게스트 운영 체제에서 디스크 이미지를 포맷하지 않고 할당되지 않은 디스크로 사용할 수 있는지 확인합니다.

14.5.3. CLI를 사용하여 스토리지 볼륨 삭제

호스트 시스템에서 스토리지 볼륨을 제거하려면 풀을 중지하고 **XML** 정의를 제거해야 합니다.

사전 요구 사항

- 삭제하려는 스토리지 볼륨을 사용하는 가상 머신이 종료됩니다.

절차

1. **virsh vol-list** 명령을 사용하여 지정된 스토리지 풀의 스토리지 볼륨을 나열합니다.

```
# virsh vol-list --pool RHEL-SP
 Name              Path
------------------------------------------------------------
 .bash_history     /home/VirtualMachines/.bash_history
 .bash_logout      /home/VirtualMachines/.bash_logout
```

```
.bash_profile       /home/VirtualMachines/.bash_profile
.bashrc             /home/VirtualMachines/.bashrc
.git-prompt.sh      /home/VirtualMachines/.git-prompt.sh
.gitconfig          /home/VirtualMachines/.gitconfig
vm-disk1            /home/VirtualMachines/vm-disk1
```

2. 선택 사항: **virsh vol-wipe** 명령을 사용하여 스토리지 볼륨을 초기화합니다. 예를 들어 스토리지 풀 **RHEL-SP:**와 연결된 **vm-disk1** 이라는 스토리지 볼륨을 지우려면 다음을 수행합니다.

```
# virsh vol-wipe --pool RHEL-SP vm-disk1
Vol vm-disk1 wiped
```

3. **virsh vol-delete** 명령을 사용하여 스토리지 볼륨을 삭제합니다. 예를 들어 스토리지 풀 **RHEL-SP:**와 연결된 **vm-disk1** 이라는 스토리지 볼륨을 삭제하려면 다음을 수행합니다.

```
# virsh vol-delete --pool RHEL-SP vm-disk1
Vol vm-disk1 deleted
```

검증

- **virsh vol-list** 명령을 다시 사용하여 스토리지 볼륨이 삭제되었는지 확인합니다.

```
# virsh vol-list --pool RHEL-SP
 Name              Path
------------------------------------------------------------
.bash_history       /home/VirtualMachines/.bash_history
.bash_logout        /home/VirtualMachines/.bash_logout
.bash_profile       /home/VirtualMachines/.bash_profile
.bashrc             /home/VirtualMachines/.bashrc
.git-prompt.sh      /home/VirtualMachines/.git-prompt.sh
.gitconfig          /home/VirtualMachines/.gitconfig
```

14.6. 웹 콘솔을 사용하여 가상 머신 스토리지 볼륨 관리

RHEL을 사용하면 **VM(**가상 머신**)**에 스토리지를 할당하는 데 사용되는 스토리지 볼륨을 관리할 수 있습니다.

RHEL 웹 콘솔을 사용하여 다음을 수행할 수 있습니다.

- 스토리지 볼륨을 생성합니다.

- *스토리지 볼륨을 제거합니다.*

14.6.1. 웹 콘솔을 사용하여 스토리지 볼륨 생성

작동하는 VM(가상 머신)을 생성하려면 VM 이미지 및 VM 관련 데이터를 저장할 수 있는 VM에 할당된 로컬 스토리지 장치가 필요합니다. 스토리지 풀에서 스토리지 볼륨을 생성하여 VM에 스토리지 디스크로 할당할 수 있습니다.

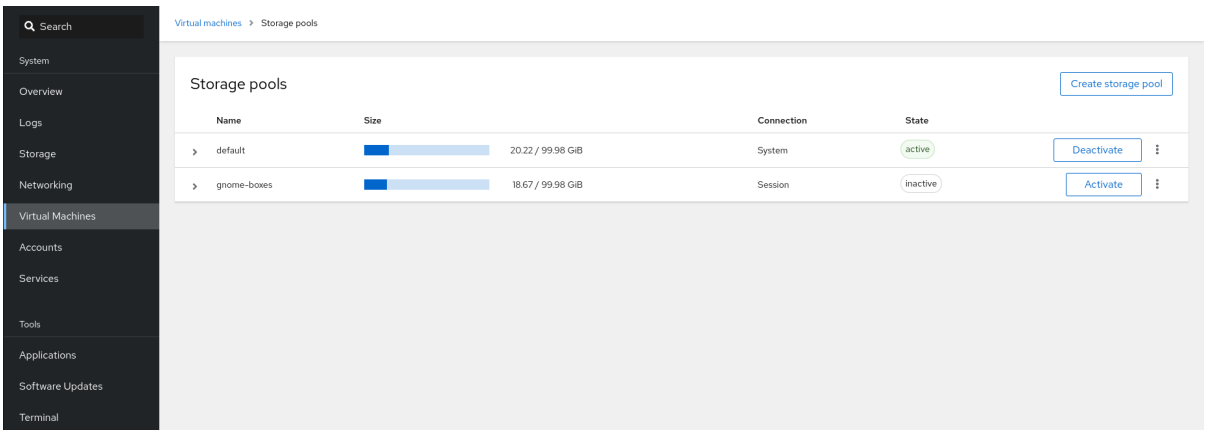
웹 콘솔을 사용하여 스토리지 볼륨을 생성하려면 다음 절차를 참조하십시오.

사전 요구 사항

- *웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.*

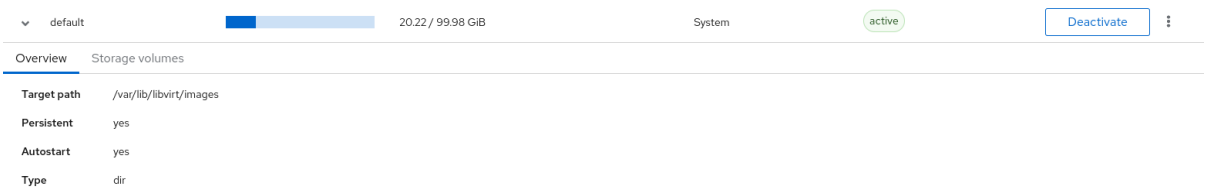
절차

1. ***Virtual Machines**(가상 머신) 탭 상단에서 **Storage Pools** (스토리지 풀)를 클릭합니다. 구성된 스토리지 풀 목록을 보여주는 스토리지 풀 창이 표시됩니다.*



2. ***Storage Pools** (스토리지 풀) 창의 스토리지 볼륨을 생성할 스토리지 풀을 클릭합니다.*

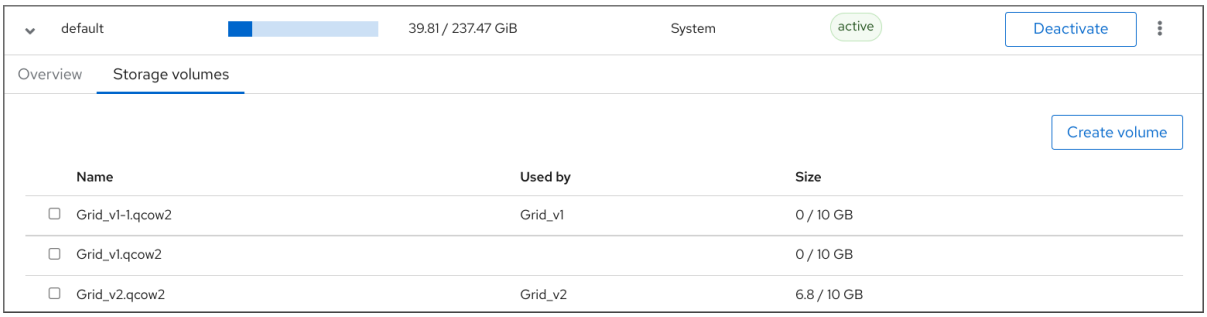
행이 확장되어 선택한 스토리지 풀에 대한 기본 정보가 포함된 개요 창을 표시합니다.



3.

*확장된 행의 **Overview**(개요) 탭 옆에 있는 **Storage Volumes** (스토리지 볼륨)를 클릭합니다.*

***Storage Volume**(스토리지 볼륨) 탭에는 기존 스토리지 볼륨에 대한 기본 정보가 있는 경우 표시됩니다.*



4. *볼륨 만들기를 클릭합니다.*

   스토리지 볼륨 만들기 대화 상자가 나타납니다.

5. *스토리지 볼륨 만들기 대화 상자에 다음 정보를 입력합니다.*

   - ***name** - 스토리지 볼륨의 이름입니다.*
   - ***size** - 스토리지 볼륨의 크기(**MiB** 또는 **GiB**)입니다.*
   - ***Format** - 스토리지 볼륨의 형식입니다. 지원되는 유형은 **qcow2** 및 **raw** 입니다.*

6. *생성을 클릭합니다.*

   *스토리지 볼륨이 생성되고, **Create Storage Volume**(스토리지 볼륨 만들기) 대화 상자가 닫*

히고 새 스토리지 볼륨이 스토리지 볼륨 목록에 표시됩니다.

추가 리소스

- *스토리지 볼륨 이해*
- *웹 콘솔을 사용하여 가상 머신에 새 디스크 추가*

14.6.2. 웹 콘솔을 사용하여 스토리지 볼륨 제거

*스토리지 볼륨을 제거하여 스토리지 풀에서 공간을 확보하거나 **defunct VM**(가상 머신)과 연결된 스토리지 항목을 제거할 수 있습니다.*

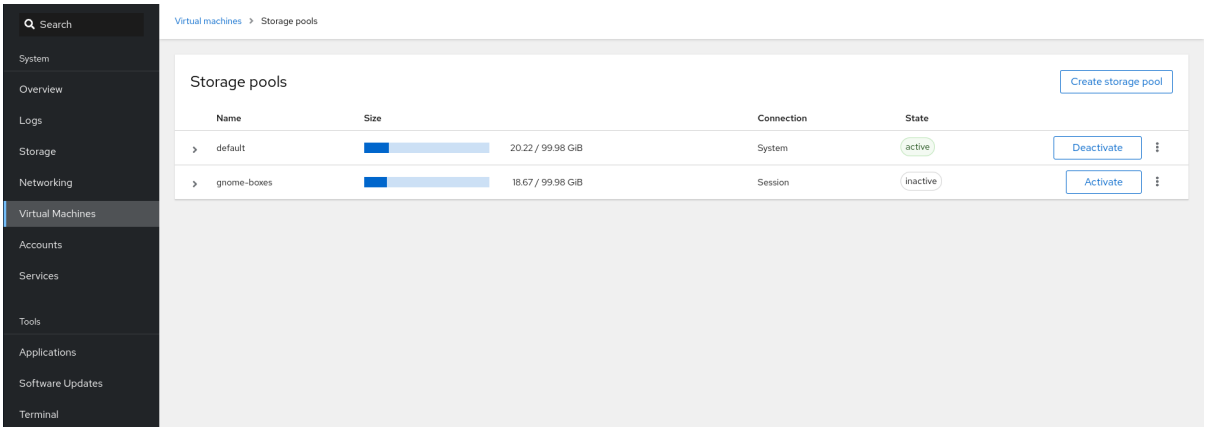
***RHEL** 웹 콘솔을 사용하여 스토리지 볼륨을 제거하려면 다음 절차를 참조하십시오.*

사전 요구 사항

- *웹 콘솔 **VM** 플러그인이 시스템에 설치되어 있습니다.*
- *삭제하려는 스토리지 볼륨을 사용하는 가상 머신이 종료됩니다.*

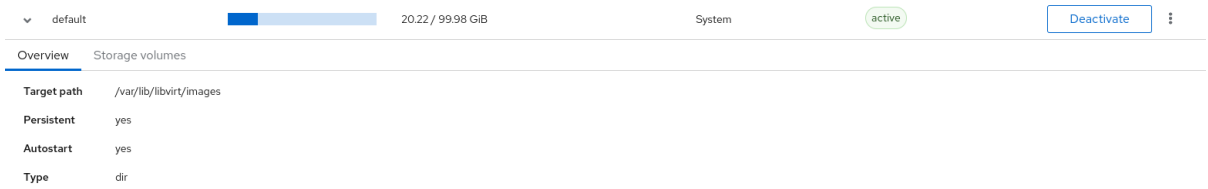
절차

1. ***Virtual Machines**(가상 머신) 탭 상단에서 **Storage Pools** (스토리지 풀)를 클릭합니다. 구성된 스토리지 풀 목록을 보여주는 스토리지 풀 창이 표시됩니다.*



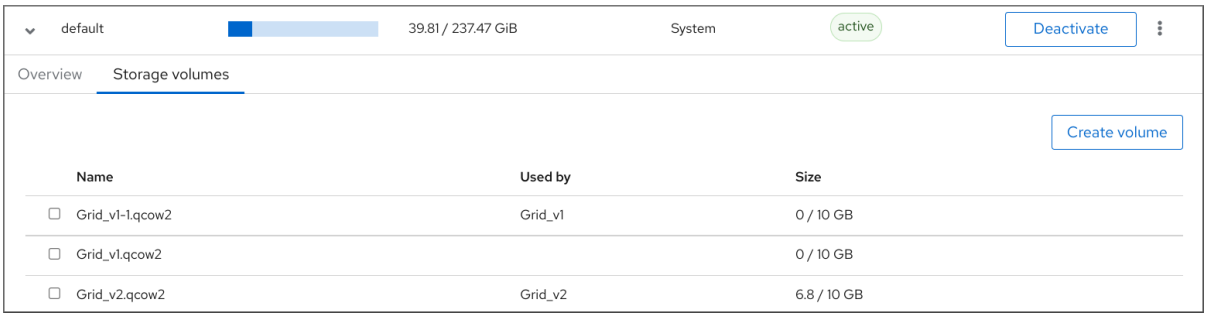
2. **Storage Pools** (스토리지 풀) 창의 스토리지 볼륨을 제거할 스토리지 풀을 클릭합니다.

   행이 확장되어 선택한 스토리지 풀에 대한 기본 정보가 포함된 개요 창을 표시합니다.

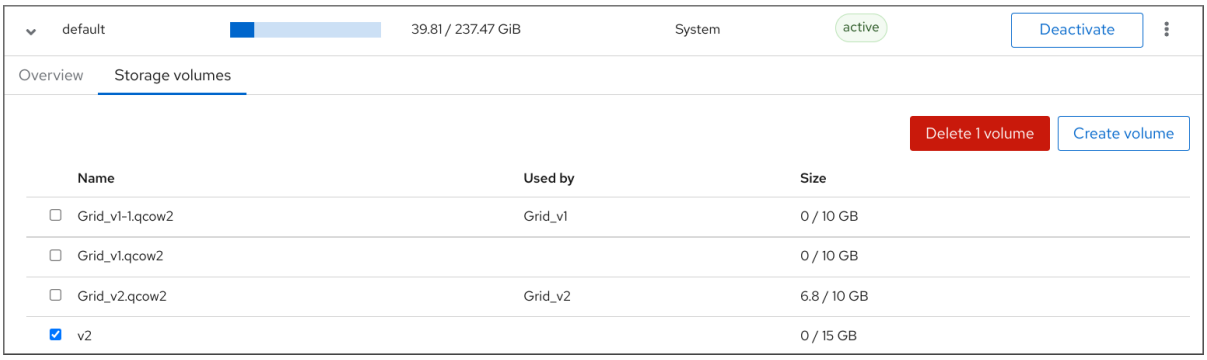


3. 확장된 행의 **Overview**(개요) 탭 옆에 있는 **Storage Volumes** (스토리지 볼륨)를 클릭합니다.

   Storage Volume(스토리지 볼륨) 탭에는 기존 스토리지 볼륨에 대한 기본 정보가 있는 경우 표시됩니다.



4. 제거할 스토리지 볼륨을 선택합니다.



5. **Delete 1 Volume**을 클릭합니다.

추가 리소스

- 스토리지 볼륨 이해

14.7. 웹 콘솔을 사용하여 가상 머신 스토리지 디스크 관리

RHEL을 사용하면 VM(가상 머신)에 연결된 스토리지 디스크를 관리할 수 있습니다.

RHEL 웹 콘솔을 사용하여 다음을 수행할 수 있습니다.

- VM 디스크 정보를 봅니다.
- 새 디스크를 VM에 추가합니다.
- 디스크를 VM에 연결합니다.
- VM에서 디스크를 분리 합니다.

14.7.1. 웹 콘솔에서 가상 머신 디스크 정보 보기

웹 콘솔을 사용하면 선택한 VM(가상 머신)에 할당된 디스크에 대한 자세한 정보를 볼 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

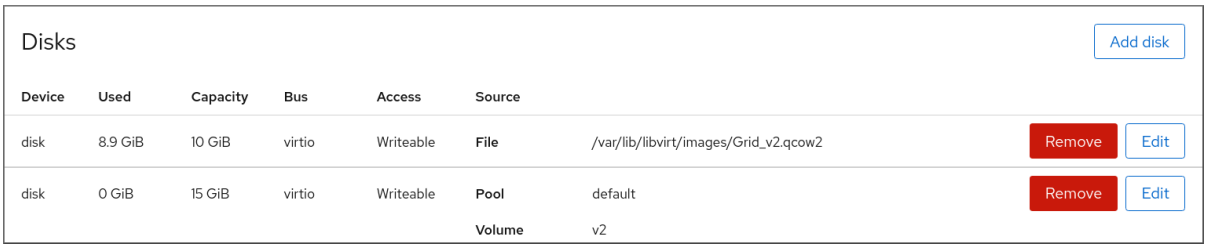
절차

1. 보려는 정보를 원하는 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

2. 디스크로 스크롤합니다.

디스크 섹션에는 VM에 할당된 디스크 정보와 디스크 추가,제거 또는 편집 옵션이 표시됩니다.



정보에는 다음이 포함됩니다.

- **device** - 디스크의 장치 유형입니다.
- **used** - 현재 할당된 디스크 양입니다.
- **capacity** - 스토리지 볼륨의 최대 크기입니다.
- **bus** - 에뮬레이션된 디스크 장치의 유형입니다.
- 액세스 - 디스크가 쓰기 가능 하거나 읽기 전용인지 여부입니다. 원시 디스크의 경우 쓰기 가능 및 공유에 대한 액세스를 설정할 수도 있습니다.
- **Source** - 디스크 장치 또는 파일입니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 정보 보기

14.7.2. 웹 콘솔을 사용하여 가상 머신에 새 디스크 추가

새 스토리지 볼륨을 생성하고 RHEL 9 웹 콘솔을 사용하여 VM에 연결하여 새 디스크를 VM에 추가할

수 있습니다.

사전 요구 사항

- *웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.*

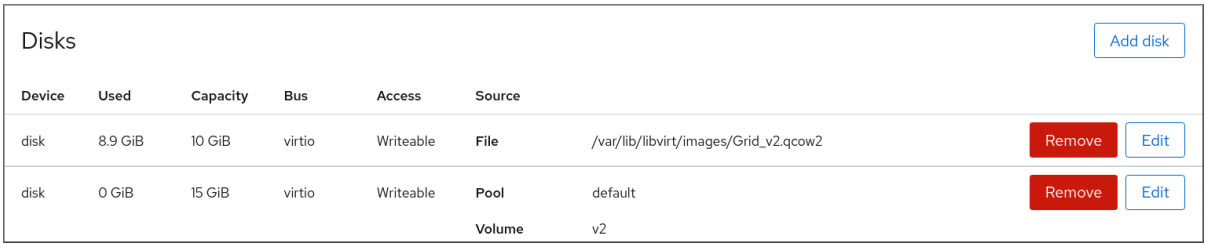
절차

1. *Virtual Machines 인터페이스에서 새 디스크를 생성하고 연결할 VM을 클릭합니다.*

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

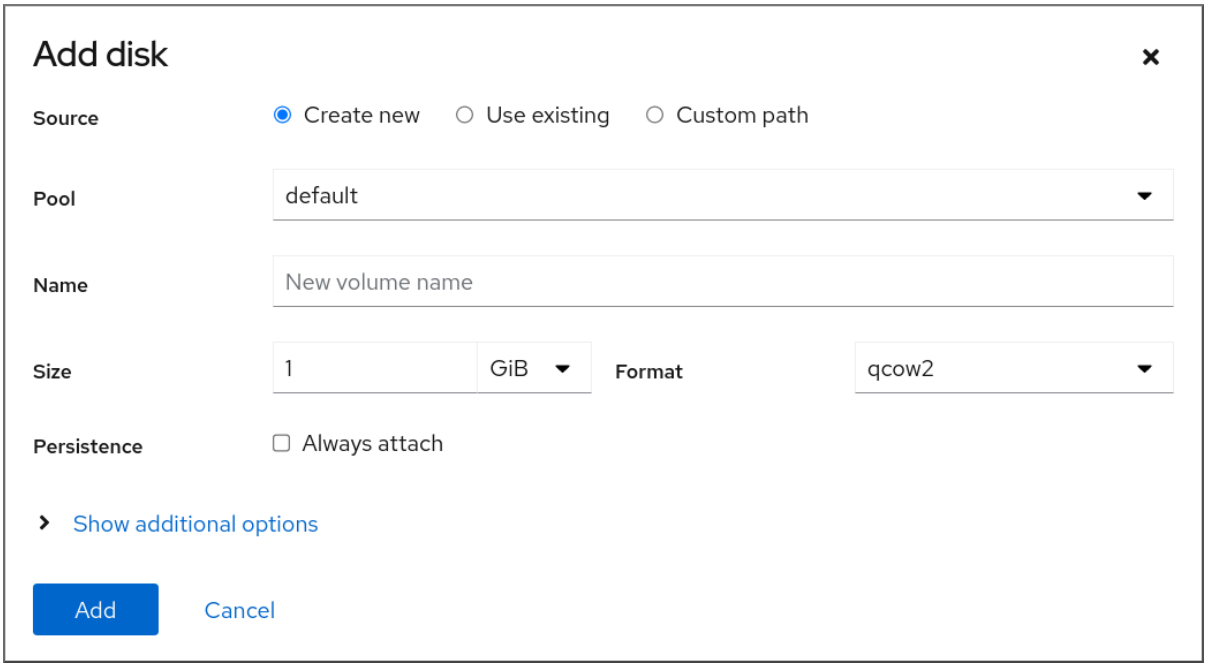
2. *디스크로 스크롤합니다.*

   디스크 섹션에는 VM에 할당된 디스크 정보와 디스크 추가,제거 또는 편집 옵션이 표시됩니다.



3. *디스크 추가를 클릭합니다.*

   디스크 추가 대화 상자가 나타납니다.



4. ***Create New (새로 만들기) 옵션을 선택합니다.***

5. *새 디스크를 구성합니다.*

- ***pool -** 가상 디스크를 생성할 스토리지 풀을 선택합니다.*
- ***Name -** 생성될 가상 디스크의 이름을 입력합니다.*
- ***size -** 크기를 입력하고 생성할 가상 디스크의 단위(**MiB** 또는 **GiB**)를 선택합니다.*
- ***Format -** 만들 가상 디스크의 형식을 선택합니다. 지원되는 유형은 **qcow2** 및 **raw** 입니다.*
- ***persistence -** 이 옵션을 선택하면 가상 디스크가 영구적입니다. 이 옵션을 선택하지 않으면 가상 디스크가 일시적입니다.*

참고

*일시적인 디스크는 실행 중인 **VM**에만 추가할 수 있습니다.*

- 추가 옵션 - 가상 디스크에 대한 추가 구성 설정.
  - **cache** - 캐시 메커니즘을 선택합니다.
  - **bus** - 에뮬레이션할 디스크 장치 유형을 선택합니다.

6. 추가를 클릭합니다.

   가상 디스크가 생성되어 VM에 연결됩니다.

추가 리소스

- 웹 콘솔에서 가상 머신 디스크 정보 보기
- 웹 콘솔을 사용하여 기존 디스크를 가상 머신에 연결
- 웹 콘솔을 사용하여 가상 머신에서 디스크 분리

14.7.3. 웹 콘솔을 사용하여 기존 디스크를 가상 머신에 연결

웹 콘솔을 사용하여 기존 스토리지 볼륨을 디스크로 VM(가상 머신)에 연결할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

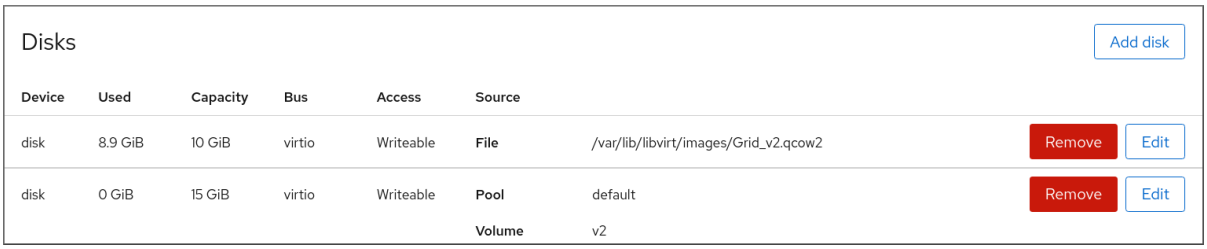
절차

1. Virtual Machines 인터페이스에서 새 디스크를 생성하고 연결할 VM을 클릭합니다.

VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

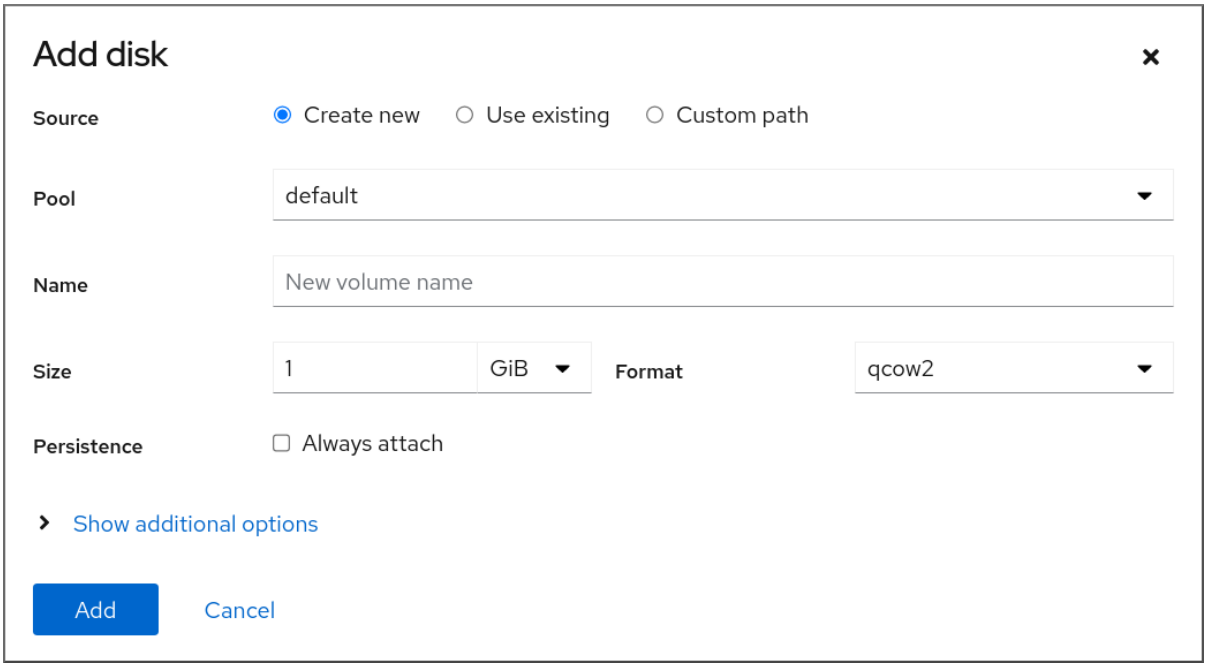
2. *디스크로 스크롤합니다.*

디스크 섹션에는 VM에 할당된 디스크 정보와 디스크 추가,제거 또는 편집 옵션이 표시됩니다.



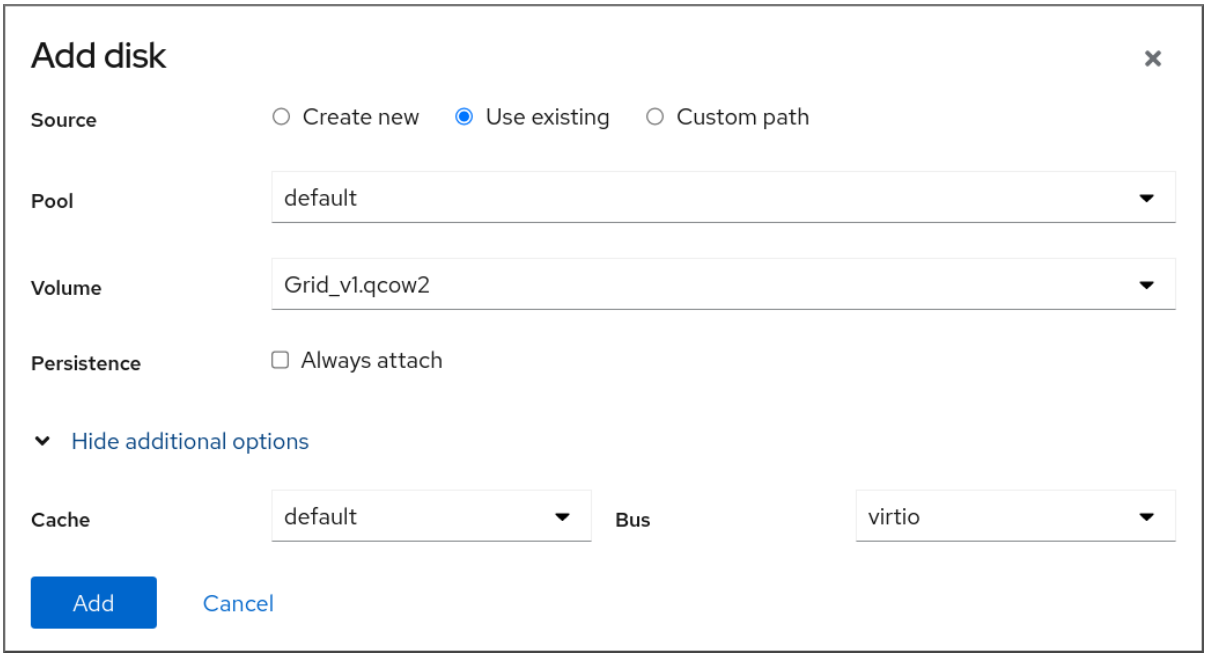
3. *디스크 추가를 클릭합니다.*

디스크 추가 대화 상자가 나타납니다.



4. ***Use Existing** 라디오 버튼을 클릭합니다.*

적절한 구성 필드가 디스크 추가 대화 상자에 나타납니다.



5. *VM의 디스크를 구성합니다.*

- ***pool** - 가상 디스크가 연결될 스토리지 풀을 선택합니다.*
- ***volume** - 연결할 스토리지 볼륨을 선택합니다.*
- *지속성 - VM이 실행 중일 때 사용 가능합니다. **Always attach** 확인란을 선택하여 가상 디스크를 영구적으로 만듭니다. 가상 디스크를 임시로 만들려면 확인란의 선택을 취소합니다.*
- *추가 옵션 - 가상 디스크에 대한 추가 구성 설정.*
  - ***cache** - 캐시 메커니즘을 선택합니다.*
  - ***bus** - 에뮬레이션할 디스크 장치 유형을 선택합니다.*

6. *추가를 클릭합니다.*

선택한 가상 디스크가 VM에 연결되어 있습니다.

추가 리소스

- 웹 콘솔에서 가상 머신 디스크 정보 보기
- 웹 콘솔을 사용하여 가상 머신에 새 디스크 추가
- 웹 콘솔을 사용하여 가상 머신에서 디스크 분리

14.7.4. 웹 콘솔을 사용하여 가상 머신에서 디스크 분리

웹 콘솔을 사용하여 VM(가상 머신)에서 디스크를 분리할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

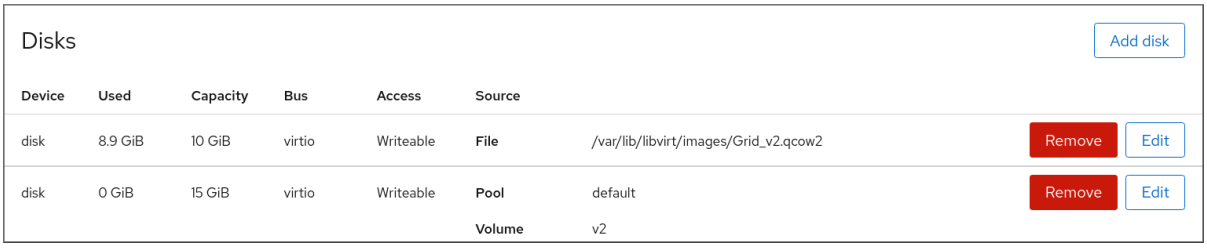
절차

1. Virtual Machines (가상 머신) 인터페이스에서 디스크를 분리할 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

2. 디스크로 스크롤합니다.

   디스크 섹션에는 VM에 할당된 디스크 정보와 디스크 추가,제거 또는 편집 옵션이 표시됩니다.



3. VM에서 분리할 디스크 옆에 있는 Remove (제거) 버튼을 클릭합니다. 디스크 제거 대화 상자가 나타납니다.

4. 확인 대화 상자에서 제거를 클릭합니다.

가상 디스크는 VM에서 분리됩니다.

추가 리소스

- 웹 콘솔에서 가상 머신 디스크 정보 보기
- 웹 콘솔을 사용하여 가상 머신에 새 디스크 추가
- 웹 콘솔을 사용하여 기존 디스크를 가상 머신에 연결

14.8. LIBVIRT 보안을 사용하여 ISCSI 스토리지 풀 보안

사용자 이름 및 암호 매개 변수는 virsh 로 구성하여 iSCSI 스토리지 풀을 보호할 수 있습니다. 풀을 정의하기 전이나 후에 구성할 수 있지만 인증 설정을 적용하려면 풀을 시작해야 합니다.

다음은 iSCSI 기반 스토리지 풀과 libvirt 시크릿을 보호하는 지침을 제공합니다.

참고

iSCSI 대상을 생성할 때 user_ID 및 암호 가 정의된 경우 이 절차가 필요합니다.

사전 요구 사항

- iSCSI 기반 스토리지 풀을 생성했는지 확인합니다. 자세한 내용은 CLI를 사용하여 iSCSI 기반 스토리지 풀 생성을 참조하십시오.

절차

1. CHAP(challenge-handshake 인증 프로토콜) 사용자 이름을 사용하여 libvirt 시크릿 파일을 만듭니다. 예를 들면 다음과 같습니다.

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

```
    <usage type='iscsi'>
      <target>iscsirhel7secret</target>
    </usage>
</secret>
```

2. **virsh secret-define** 명령을 사용하여 **libvirt** 시크릿을 정의합니다.

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

3. **virsh secret-list** 명령을 사용하여 **UUID**를 확인합니다.

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

4. **virsh secret-set-value** 명령을 사용하여 이전 단계 출력의 **UUID**에 시크릿을 할당합니다. 이렇게 하면 **CHAP** 사용자 이름과 암호가 **libvirt** 제어 비밀 목록에 있습니다. 예를 들면 다음과 같습니다.

```
# virsh secret-set-value --interactive 2d7891af-20be-4e5e-af83-190e8a922360
Enter new value for secret:
Secret value set
```

5. **virsh edit** 명령을 사용하여 스토리지 풀의 **XML** 파일에 인증 항목을 추가하고 인증 유형, 사용자 이름 , 시크릿 사용량 을 지정하여 **< auth >** 요소를 추가합니다.

예를 들면 다음과 같습니다.

```
<pool type='iscsi'>
  <name>iscsirhel7pool</name>
    <source>
      <host name='192.168.122.1'/>
      <device path='iqn.2010-05.com.example.server1:iscsirhel7guest'/>
      <auth type='chap' username='redhat'>
        <secret usage='iscsirhel7secret'/>
      </auth>
    </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

참고

< auth > 하위 요소는 가상 머신의 < pool > 및 < disk > XML 요소 내의 다른 위치에 있습니다. < pool >의 경우 < auth > 요소는 < source > 요소 내에 지정됩니다. 이는 인증이 일부 풀 소스(iSCSI 및 RBD)의 속성이므로 풀 소스를 찾을 위치를 설명합니다. 도메인 의 하위 요소인 <disk >의 경우 iSCSI 또는 RBD 디스크에 대한 인증은 디스크의 속성입니다. 또한 디스크의 <auth > 하위 요소는 스토리지 풀과 다릅니다.

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

6. 변경 사항을 활성화하려면 스토리지 풀을 활성화합니다. 풀이 이미 시작된 경우 스토리지 풀을 중지하고 다시 시작합니다.

```
# virsh pool-destroy iscsirhel7pool
# virsh pool-start iscsirhel7pool
```

14.9. VHBAS 만들기

가상 호스트 버스 어댑터(vHBA) 장치는 호스트 시스템을 SCSI 장치에 연결하고 SCSI 기반 스토리지 풀을 생성하는 데 필요합니다.

XML 구성 파일에 vHBA 장치를 정의할 수 있습니다.

절차

1. virsh nodedev-list --cap vports 명령을 사용하여 호스트 시스템에서 HBA를 찾습니다.

   다음 예제에서는 vHBA를 지원하는 HBA가 두 개 있는 호스트를 보여줍니다.

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

2. virsh nodedev-dumpxml HBA_device명령을 사용하여 HBA 의 세부 정보를 확인합니다.

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

명령의 출력에는 **vHBA**를 만드는 데 사용되는 **< name > , < wwnn >** 및 **< wwpn >** 필드가 나열됩니다. **< max_vports >**에는 지원되는 최대 **vHBA**가 표시됩니다. 예를 들면 다음과 같습니다.

```
<device>
  <name>scsi_host3</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3</path>
  <parent>pci_0000_10_00_0</parent>
  <capability type='scsi_host'>
    <host>3</host>
    <unique_id>0</unique_id>
    <capability type='fc_host'>
      <wwnn>20000000c9848140</wwnn>
      <wwpn>10000000c9848140</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
    <capability type='vport_ops'>
      <max_vports>127</max_vports>
      <vports>0</vports>
    </capability>
  </capability>
</device>
```

이 예에서 **< max_vports >** 값은 **HBA** 구성에서 사용할 수 있는 총 **127**개의 가상 포트가 있음을 보여줍니다. **& lt;vports& gt;** 값은 현재 사용 중인 가상 포트 수를 표시합니다. 이러한 값은 **vHBA**를 생성한 후 업데이트됩니다.

3.

vHBA 호스트에 대해 다음 중 하나와 유사한 **XML** 파일을 만듭니다. 이 예제에서 파일의 이름은 **vhba_host3.xml** 입니다.

이 예에서는 **scsi_host3** 을 사용하여 상위 **vHBA**를 설명합니다.

```
<device>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <capability type='fc_host'>
    </capability>
  </capability>
</device>
```

이 예제에서는 **WWNN/WWPN** 쌍을 사용하여 상위 **vHBA**를 설명합니다.

```
<device>
  <name>vhba</name>
  <parent wwnn='20000000c9848140' wwpn='10000000c9848140'/>
  <capability type='scsi_host'>
    <capability type='fc_host'>
```

```
    </capability>
  </capability>
</device>
```

참고

WWNN 및 WWPN 값은 이전 단계에서 확인한 HBA 세부 정보의 값과 일치해야 합니다.

& lt;parent > 필드는 이 vHBA 장치와 연결할 HBA 장치를 지정합니다. < device> 태그의 세부 사항은 다음 단계에서 호스트의 새 vHBA 장치를 생성하는 데 사용됩니다. nodedev XML 형식에 대한 자세한 내용은 libvirt 업스트림 페이지를 참조하십시오.

참고

virsh 명령은 parent_wwnn,parent_wwpn 또는 parent_fabric_wwn 속성을 정의하는 방법을 제공하지 않습니다.

4. virsh nodev-create 명령을 사용하여 이전 단계에서 만든 XML 파일을 기반으로 VHBA를 만듭니다.

```
# virsh nodedev-create vhba_host3
Node device scsi_host5 created from vhba_host3.xml
```

검증

- virsh nodedev-dumpxml 명령을 사용하여 새 vHBA의 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>
    <unique_id>2</unique_id>
    <capability type='fc_host'>
      <wwnn>5001a4a93526d0a1</wwnn>
      <wwpn>5001a4ace3ee047d</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
```

```
    </capability>
  </capability>
</device>
```

추가 리소스

- CLI를 사용하여 vHBA 장치로 SCSI 기반 스토리지 풀 생성

15장. 가상 머신에서 GPU 장치 관리

RHEL 9 호스트에서 VM(가상 머신)의 그래픽 성능을 향상시키기 위해 호스트 GPU를 VM에 할당할 수 있습니다.

- GPU를 호스트에서 분리하고 GPU의 전체 제어를 VM에 직접 전달할 수 있습니다.
- 물리적 GPU에서 여러 개의 중재 장치를 생성하고 이러한 장치를 여러 게스트에 가상 GPU(vGPU)로 할당할 수 있습니다. 이는 현재 선택한 NVIDIA GPU에서만 지원되며, 단일 게스트에 중재된 장치 한 개만 할당할 수 있습니다.

15.1. 가상 머신에 GPU 할당

호스트 시스템에 연결된 GPU에 액세스하고 제어하려면 GPU에 대한 직접 제어를 VM(가상 머신)으로 전달하도록 호스트 시스템을 구성해야 합니다.

참고

가상 GPU 할당에 대한 정보를 찾고 있는 경우 NVIDIA vGPU 장치 관리를 참조하십시오.

사전 요구 사항

- 호스트 시스템 커널에서 IOMMU 지원을 활성화해야 합니다.
  - Intel 호스트에서 VT-d를 활성화해야 합니다.
    1. intel_iommu=on 및 iommu=pt 매개변수를 사용하여 GRUB 설정을 다시 생성합니다.

       ```
       # grubby --args="intel_iommu=on iommu_pt" --update-kernel DEFAULT
       ```

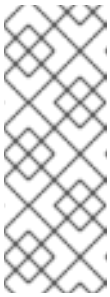
    2. 호스트를 재부팅합니다.

- *AMD 호스트에서 AMD-Vi를 활성화해야 합니다.*

  AMD 호스트에서 IOMMU는 기본적으로 활성화되어 있으므로 iommu=pt 를 추가하여 통과 모드로 전환할 수 있습니다.

1. *iommu=pt 매개변수를 사용하여 GRUB 설정을 다시 생성합니다.*

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



참고

pt 옵션은 통과 모드에서 사용되는 장치에 대해서만 IOMMU를 활성화하고 더 나은 호스트 성능을 제공합니다. 그러나 모든 하드웨어가 옵션을 지원하는 것은 아닙니다. 이 옵션이 활성화되어 있는지 여부에 관계없이 장치를 할당할 수 있습니다.

2. *호스트를 재부팅합니다.*

절차

1. *드라이버가 GPU에 바인딩되지 않도록 합니다.*

a. *GPU가 연결된 PCI 버스 주소를 식별합니다.*

```
# lspci -Dnn | grep VGA
0000:02:00.0 VGA compatible controller [0300]: NVIDIA Corporation GK106GL
[Quadro K4000] [10de:11fa] (rev a1)
```

b. *호스트의 그래픽 드라이버가 GPU를 사용하지 못하도록 합니다. 이 작업을 수행하려면 pci-stub 드라이버와 함께 GPU의 PCI ID를 사용합니다.*

예를 들어 다음 명령은 드라이버가 10de:11fa 버스에 연결된 GPU에 바인딩하지 못하도록 합니다.

```
# grubby --args="pci-stub.ids=10de:11fa" --update-kernel DEFAULT
```

c. *호스트를 재부팅합니다.*

2. *선택 사항: 지원 제한으로 인해 오디오와 같은 특정 **GPU** 함수를 **VM**으로 전달할 수 없는 경우 필요한 **GPU** 함수만 전달하도록 **IOMMU** 그룹 내의 끝점의 드라이버 바인딩을 수정할 수 있습니다.*

a. *GPU 설정을 **XML**로 변환하고 호스트 드라이버에 연결하지 못하도록 하려는 끝점의 **PCI** 주소를 기록해 둡니다.*

*이렇게 하려면 **pci_** 접두사를 주소에 추가하고 구분 기호를 밑줄로 변환하여 **GPU**의 **PCI** 버스 주소를 **libvirt** 호환 형식으로 변환합니다.*

*예를 들어 다음 명령은 **0000:02:00.0** 버스 주소에 연결된 **GPU**의 **XML** 구성을 표시합니다.*

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

```
<device>
 <name>pci_0000_02_00_0</name>
 <path>/sys/devices/pci0000:00/0000:00:03.0/0000:02:00.0</path>
 <parent>pci_0000_00_03_0</parent>
 <driver>
  <name>pci-stub</name>
 </driver>
 <capability type='pci'>
  <domain>0</domain>
  <bus>2</bus>
  <slot>0</slot>
  <function>0</function>
  <product id='0x11fa'>GK106GL [Quadro K4000]</product>
  <vendor id='0x10de'>NVIDIA Corporation</vendor>
  <iommuGroup number='13'>
   <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
   <address domain='0x0000' bus='0x02' slot='0x00' function='0x1'/>
  </iommuGroup>
  <pci-express>
   <link validity='cap' port='0' speed='8' width='16'/>
   <link validity='sta' speed='2.5' width='16'/>
  </pci-express>
 </capability>
</device>
```

b. *엔드포인트가 호스트 드라이버에 연결하지 못하도록 합니다.*

이 예에서 GPU를 VM에 할당하려면 < address domain='0x0000' bus='0x02' slot='0x00' function='0x00'/ > , 을 통해 호스트 오디오 드라이버에 연결하고 끝점을 VFIO-PCI에 연결하는 대신 VFIO- PCI에 연결하는 것을 방지합니다.

```
# driverctl set-override 0000:02:00.1 vfio-pci
```

3. GPU를 VM에 연결

a. PCI 버스 주소를 사용하여 GPU에 대한 XML 구성 파일을 만듭니다.

예를 들어 GPU 버스 주소의 매개변수를 사용하여 다음 XML 파일 GPU-Assign.xml을 생성할 수 있습니다.

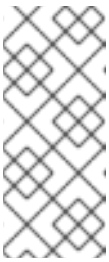
```
<hostdev mode='subsystem' type='pci' managed='yes'>
 <driver name='vfio'/>
 <source>
  <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
 </source>
</hostdev>
```

b. 호스트 시스템에 파일을 저장합니다.

c. 파일을 VM의 XML 구성과 병합합니다.

예를 들어 다음 명령은 GPU XML 파일 GPU-Assign.xml을 System1 VM의 XML 구성 파일과 병합합니다.

```
# virsh attach-device System1 --file /home/GPU-Assign.xml --persistent
Device attached successfully.
```



참고

GPU는 VM에 보조 그래픽 장치로 연결됩니다. GPU를 기본 그래픽 장치로 할당하는 것은 지원되지 않으며, VM의 XML 구성에서 주요 에뮬레이션 그래픽 장치를 제거하지 않는 것이 좋습니다.

검증

- 장치가 VM의 XML 구성의 < devices > 섹션에 나타납니다. 자세한 내용은 샘플 가상 머신 XML 구성에서 참조하십시오.

알려진 문제

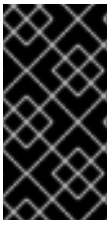
- *VM에 연결할 수 있는 GPU 수는 RHEL 9에서 현재 64인 할당된 PCI 장치의 최대 수로 제한됩니다. 그러나 여러 GPU를 VM에 연결하면 게스트에서 메모리 매핑된 I/O(MMIO)에 문제가 발생할 수 있으므로 VM에서 GPU를 사용할 수 없습니다.*

  이러한 문제를 해결하려면 더 큰 64비트 MMIO 공간을 설정하고 64비트 MMIO 공간을 확장할 수 있도록 vCPU 물리적 주소 비트를 구성합니다.

- *현재 RHEL 9 게스트 운영 체제를 사용하는 VM에 NVIDIA GPU 장치를 연결하면 해당 VM에서 Wayland 세션을 비활성화하고 대신 Xorg 세션을 로드합니다. 이는 NVIDIA 드라이버와 Wayland 간의 비호환성 때문입니다.*

15.2. NVIDIA VGPU 장치 관리

vGPU 기능을 사용하면 물리적 NVIDIA GPU 장치를 중재 장치라고 하는 여러 가상 장치로 나눌 수 있습니다. 그런 다음 이러한 중재된 장치를 가상 GPU로 여러 VM(가상 머신)에 할당할 수 있습니다. 결과적으로 이러한 VM은 단일 물리 GPU의 성능을 공유할 수 있습니다.



중요

중재 장치를 사용하거나 사용하지 않고 VM에 물리적 GPU를 할당하면 호스트가 GPU를 사용할 수 없습니다.

15.2.1. NVIDIA vGPU 장치 설정

NVIDIA vGPU 기능을 설정하려면 GPU 장치에 대해 NVIDIA vGPU 드라이버를 다운로드하여 중재 장치를 생성하여 원하는 가상 머신에 할당해야 합니다. 자세한 지침은 아래를 참조하십시오.

사전 요구 사항

- *GPU는 vGPU 중재 장치를 지원합니다. vGPU 생성을 지원하는 NVIDIA GPU의 최신 목록은 NVIDIA vGPU 소프트웨어 설명서를 참조하십시오.*

  - *호스트가 사용 중인 GPU를 모르는 경우 lshw 패키지를 설치하고 lshw -C display 명령을 사용하십시오. 다음 예에서는 해당 시스템이 NVIDIAether P4 GPU를 사용하며 vGPU와 호환되는 것입니다.*

```
# lshw -C display

*-display
       description: 3D controller
       product: GP104GL [Tesla P4]
       vendor: NVIDIA Corporation
       physical id: 0
       bus info: pci@0000:01:00.0
       version: a1
       width: 64 bits
       clock: 33MHz
       capabilities: pm msi pciexpress cap_list
       configuration: driver=vfio-pci latency=0
       resources: irq:16 memory:f6000000-f6ffffff memory:e0000000-efffffff
memory:f0000000-f1ffffff
```

절차

1. NVIDIA vGPU 드라이버를 다운로드하여 시스템에 설치합니다. 자세한 내용은 NVIDIA 문서를 참조하십시오.

2. NVIDIA 소프트웨어 설치 관리자가 /etc/modprobe.d/nvidia-installer-disable-nouveau.conf 파일을 생성하지 않은 경우 /etc/modprobe.d/ 에 이름의 conf 파일을 생성하고 파일에 다음 행을 추가합니다.

```
blacklist nouveau
options nouveau modeset=0
```

3. 현재 커널의 초기 램디스크를 다시 생성한 다음 재부팅합니다.

```
# dracut --force
# reboot
```

4. 커널이 nvidia_vgpu_vfio 모듈을 로드하고 nvidia-vgpu-mgr.service 서비스가 실행 중인지 확인합니다.

```
# lsmod | grep nvidia_vgpu_vfio
nvidia_vgpu_vfio 45011 0
nvidia 14333621 10 nvidia_vgpu_vfio
mdev 20414 2 vfio_mdev,nvidia_vgpu_vfio
vfio 32695 3 vfio_mdev,nvidia_vgpu_vfio,vfio_iommu_type1

# systemctl status nvidia-vgpu-mgr.service
nvidia-vgpu-mgr.service - NVIDIA vGPU Manager Daemon
   Loaded: loaded (/usr/lib/systemd/system/nvidia-vgpu-mgr.service; enabled; vendor
preset: disabled)
```

```
  Active: active (running) since Fri 2018-03-16 10:17:36 CET; 5h 8min ago
Main PID: 1553 (nvidia-vgpu-mgr)
[...]
```

또한 NVIDIA Ampere GPU 장치를 기반으로 vGPU를 생성하는 경우, 물리 GPU에 가상 기능을 사용하도록 설정해야 합니다. 자세한 내용은 NVIDIA 문서 를 참조하십시오.

5. 장치 UUID를 생성합니다.

```
# uuidgen
30820a6f-b1a5-4503-91ca-0c10ba58692a
```

6. 감지된 GPU 하드웨어에 따라 중재된 장치의 구성으로 XML 파일을 준비합니다. 예를 들어, 다음은 0000:01:00.0 PCI 버스에서 실행되고 이전 단계에서 생성된 UUID를 사용하는 NVIDIA Tesla P4 카드에서 중재된 nvidia-63 vGPU 유형의 중재 장치를 구성합니다.

```
<device>
  <parent>pci_0000_01_00_0</parent>
  <capability type="mdev">
    <type id="nvidia-63"/>
    <uuid>30820a6f-b1a5-4503-91ca-0c10ba58692a</uuid>
  </capability>
</device>
```

7. 준비한 XML 파일을 기반으로 vGPU 조정 장치를 정의합니다. 예를 들면 다음과 같습니다.

```
# virsh nodedev-define vgpu-test.xml
Node device mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0 created
from vgpu-test.xml
```

8. 선택 사항: 중재 장치가 비활성으로 나열되는지 확인합니다.

```
# virsh nodedev-list --cap mdev --inactive
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

9. 생성한 vGPU 중재 장치를 시작합니다.

```
# virsh nodedev-start mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Device mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0 started
```

10. 선택 사항: 중재 장치가 활성 상태로 나열되어 있는지 확인합니다.

```
# virsh nodedev-list --cap mdev
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

11. 호스트가 재부팅된 후 자동으로 시작하도록 vGPU 장치 설정

```
# virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Device mdev_d196754e_d8ed_4f43_bf22_684ed698b08b_0000_9b_00_0 marked as
autostarted
```

12. vGPU 리소스를 공유하려는 VM에 중재 장치를 연결합니다. 이렇게 하려면 이전에 의도했던 UUID와 함께 다음 행을 VM의 XML 구성의 < devices/ > 섹션에 추가합니다.

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci' display='on'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>
  </source>
</hostdev>
```

각 UUID는 한 번에 하나의 VM에만 할당할 수 있습니다. 또한 VM에 virtio-vga 와 같은 QEMU 비디오 장치가 없는 경우 < hostdev > 행에 ramfb='on' 매개변수도 추가합니다.

13. 할당된 VM에서 사용할 수 있는 vGPU 조정 장치의 모든 기능을 사용하려면 VM에서 NVIDIA vGPU 게스트 소프트웨어 라이선스를 설정합니다. 자세한 내용 및 지침은 NVIDIA Virtual GPU 소프트웨어 라이센스 서버 사용자 가이드를 참조하십시오.

검증

1. 생성한 vGPU의 기능을 쿼리하고 활성 및 지속적으로 나열되는지 확인합니다.

```
# virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Name:           virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Parent:         pci_0000_01_00_0
Active:         yes
Persistent:     yes
Autostart:      yes
```

2. VM을 시작하고 게스트 운영 체제가 중재 장치를 NVIDIA GPU로 감지하는지 확인합니다. 예를 들어 VM에서 Linux를 사용하는 경우 다음을 실행합니다.

```
# lspci -d 10de: -k
07:00.0 VGA compatible controller: NVIDIA Corporation GV100GL [Tesla V100 SXM2 32GB] (rev a1)
        Subsystem: NVIDIA Corporation Device 12ce
        Kernel driver in use: nvidia
        Kernel modules: nouveau, nvidia_drm, nvidia
```

알려진 문제

- 현재 RHEL 9 게스트 운영 체제를 사용하는 VM에 NVIDIA vGPU 중재 장치를 할당하는 경우 해당 VM에서 Wayland 세션을 비활성화하고 대신 Xorg 세션을 로드합니다. 이는 NVIDIA 드라이버와 Wayland 간의 비호환성 때문입니다.

추가 리소스

- NVIDIA vGPU 소프트웨어 문서
- man virsh 명령

15.2.2. NVIDIA vGPU 장치 제거

할당된 vGPU 중재 장치의 구성을 변경하려면 할당된 VM에서 기존 장치를 제거해야 합니다. 자세한 내용은 아래를 참조하십시오.

사전 요구 사항

- 장치를 제거하려는 VM이 종료됩니다.

절차

1. 제거하려는 중재 장치의 ID를 가져옵니다.

```
# virsh nodedev-list --cap mdev
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

2. 실행 중인 **vGPU** 중재 장치의 인스턴스를 중지합니다.

```
# virsh nodedev-destroy
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Destroyed node device
'mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0'
```

3. 선택 사항: 중재 장치가 비활성화되었는지 확인합니다.

```
# virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Name:           virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Parent:         pci_0000_01_00_0
Active:         no
Persistent:     yes
Autostart:      yes
```

4. **VM**의 **XML** 구성에서 장치를 제거합니다. 이렇게 하려면 **virsh edit** 유틸리티를 사용하여 **VM**의 **XML** 구성을 편집하고 **mdev**의 구성 세그먼트를 제거합니다. 세그먼트는 다음과 유사합니다.

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>
  </source>
</hostdev>
```

중재된 장치를 중지 및 분리해도 해당 장치는 삭제되지 않고 정의된 대로 유지됩니다. 따라서 장치를 *재시작* 하고 다른 **VM**에 *연결할* 수 있습니다.

5. 선택 사항: 중지된 중재 장치를 삭제하려면 정의를 제거합니다.

```
# virsh nodedev-undefine
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Undefined node device
'mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0'
```

검증

- 장치를 중지 및 분리한 경우 중재 장치가 비활성으로 나열되어 있는지 확인합니다.

```
# virsh nodedev-list --cap mdev --inactive
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

- 장치를 삭제한 경우 다음 명령이 표시되지 않는지 확인합니다.

```
# virsh nodedev-list --cap mdev
```

추가 리소스

- man virsh 명령

15.2.3. 시스템에 대한 NVIDIA vGPU 정보 가져오기

사용 가능한 vGPU 기능의 기능을 평가하기 위해 다음과 같이 시스템의 중재 장치에 대한 추가 정보를 얻을 수 있습니다.

- 지정된 유형의 중재 장치 수를 생성할 수 있습니다.
- 시스템에 이미 구성된 중재 장치는 무엇입니까.

절차

- vGPU 중재 장치를 지원할 수 있는 호스트에서 사용 가능한 GPU 장치를 보려면 virsh nodedev-list --cap mdev_types 명령을 사용합니다. 예를 들어, 다음은 NVIDIA Quadro RTX6000 장치가 있는 시스템을 보여줍니다.

```
# virsh nodedev-list --cap mdev_types
pci_0000_5b_00_0
pci_0000_9b_00_0
```

- 특정 GPU 장치에서 지원하는 vGPU 유형과 추가 메타데이터를 표시하려면 virsh nodedev-dumpxml 명령을 사용합니다.

```
# virsh nodedev-dumpxml pci_0000_9b_00_0
<device>
 <name>pci_0000_9b_00_0</name>
 <path>/sys/devices/pci0000:9a/0000:9a:00.0/0000:9b:00.0</path>
 <parent>pci_0000_9a_00_0</parent>
 <driver>
  <name>nvidia</name>
 </driver>
 <capability type='pci'>
  <class>0x030000</class>
```

```
    <domain>0</domain>
    <bus>155</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x1e30'>TU102GL [Quadro RTX 6000/8000]</product>
    <vendor id='0x10de'>NVIDIA Corporation</vendor>
    <capability type='mdev_types'>
     <type id='nvidia-346'>
      <name>GRID RTX6000-12C</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>2</availableInstances>
     </type>
     <type id='nvidia-439'>
      <name>GRID RTX6000-3A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>8</availableInstances>
     </type>
     [...]
     <type id='nvidia-440'>
      <name>GRID RTX6000-4A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>6</availableInstances>
     </type>
     <type id='nvidia-261'>
      <name>GRID RTX6000-8Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>3</availableInstances>
     </type>
    </capability>
    <iommuGroup number='216'>
     <address domain='0x0000' bus='0x9b' slot='0x00' function='0x3'/>
     <address domain='0x0000' bus='0x9b' slot='0x00' function='0x1'/>
     <address domain='0x0000' bus='0x9b' slot='0x00' function='0x2'/>
     <address domain='0x0000' bus='0x9b' slot='0x00' function='0x0'/>
    </iommuGroup>
    <numa node='2'/>
    <pci-express>
     <link validity='cap' port='0' speed='8' width='16'/>
     <link validity='sta' speed='2.5' width='8'/>
    </pci-express>
   </capability>
  </device>
```

추가 리소스

- *man virsh 명령*

15.2.4. NVIDIA vGPU용 원격 데스크탑 스트리밍 서비스

다음 원격 데스크탑 스트리밍 서비스는 NVIDIA vGPU 또는 NVIDIA GPU 패스스루가 활성화된 RHEL 9 하이퍼바이저에서 지원됩니다.

- *HP ZCentral Remote Boost/Teradici*
- *NICE DCV*
- *Mechdyne TGX*

지원 세부 사항은 해당 벤더 지원 매트릭스를 참조하십시오.

15.2.5. 추가 리소스

- *NVIDIA vGPU 소프트웨어 문서*

16장. 가상 머신 네트워크 연결 구성

VM(가상 머신)이 네트워크를 통해 호스트에 연결하고 호스트의 다른 VM에 연결하고 외부 네트워크의 위치에서 VM 네트워킹을 적절하게 구성해야 합니다. VM 네트워킹을 제공하기 위해 RHEL 9 하이퍼바이저와 새로 생성된 VM에는 기본 네트워크 구성이 있어 추가로 수정할 수도 있습니다. 예를 들면 다음과 같습니다.

- VM이 호스트와 동일한 네트워크에 있는 것처럼 호스트의 VM을 검색 및 호스트 외부 위치에 연결할 수 있습니다.
- VM을 인바운드 네트워크 트래픽에서 부분적으로 또는 완전히 격리하여 보안을 향상시키고 VM에 영향을 미치는 문제의 위험을 최소화할 수 있습니다.

다음 섹션에서는 다양한 유형의 VM 네트워크 구성에 대해 설명하고 선택한 VM 네트워크 구성 설정에 대한 지침을 제공합니다.

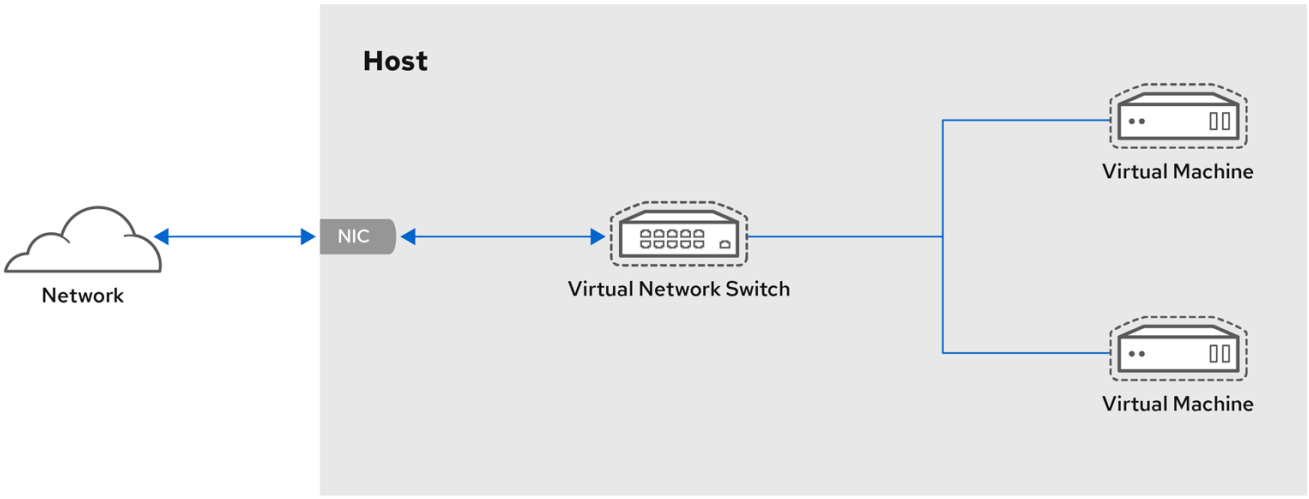
16.1. 가상 네트워킹 이해

가상 머신(VM)을 다른 장치 및 네트워크의 위치와 호스트 하드웨어에 쉽게 연결할 수 있어야 합니다. 다음 섹션에서는 VM 네트워크 연결 메커니즘을 설명하고 기본 VM 네트워크 설정을 설명합니다.

16.1.1. 가상 네트워크 작동 방식

가상 네트워킹은 가상 네트워크 스위치의 개념을 사용합니다. 가상 네트워크 스위치는 호스트 시스템에서 작동하는 소프트웨어 구조입니다. VM은 가상 네트워크 스위치를 통해 네트워크에 연결합니다. 가상 스위치의 구성에 따라 VM은 하이퍼바이저 또는 다른 네트워크 연결 방법으로 관리되는 기존 가상 네트워크를 사용할 수 있습니다.

다음 그림은 두 개의 VM을 네트워크에 연결하는 가상 네트워크 스위치를 보여줍니다.



*게스트 운영 체제 관점에서 가상 네트워크 연결은 물리적 네트워크 연결과 동일합니다. 호스트 시스템은 가상 네트워크 스위치를 네트워크 인터페이스로 봅니다. **virtnetworkd** 서비스가 처음 설치 및 시작되면 **VM**의 기본 네트워크 인터페이스인 **virbr0** 을 생성합니다.*

*이 인터페이스에 대한 정보를 보려면 호스트에서 **ip** 유틸리티를 사용합니다.*

```
$ 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
```

*기본적으로 단일 호스트의 모든 **VM**은 **virbr0** 인터페이스를 사용하는 **default** 라는 동일한 **NAT-type** 가상 네트워크에 연결됩니다. 자세한 내용은 가상 네트워킹 기본 구성을 참조하십시오.*

***VM**의 기본 아웃바운드 전용 네트워크 액세스의 경우 기본 네트워크가 **libvirt-daemon-config-network** 패키지와 함께 설치되고 **virtnetworkd** 서비스가 시작될 때 자동으로 시작되므로 추가 네트워크 설정이 필요하지 않습니다.*

*다른 **VM** 네트워크 기능이 필요한 경우 추가 가상 네트워크 및 네트워크 인터페이스를 생성하고 이를 사용하도록 **VM**을 구성할 수 있습니다. 기본 **NAT** 외에도 다음 모드 중 하나를 사용하도록 이러한 네트워크 및 인터페이스를 구성할 수 있습니다.*

- *라우팅 모드*
- *브리지 모드*

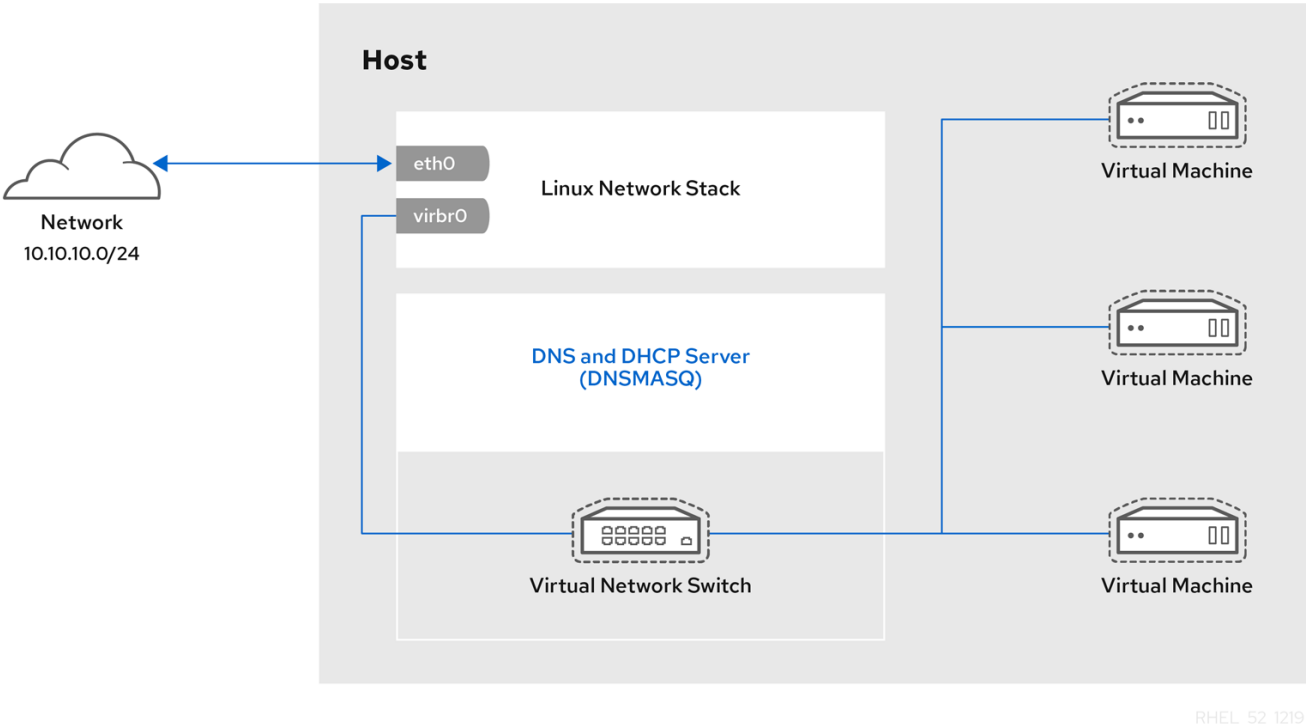
- *isolated 모드*
- *열려 있는 모드*

16.1.2. 가상 네트워킹 기본 구성

***virtnetworkd** 서비스가 가상화 호스트에 처음 설치되면 **NAT**(네트워크 주소 변환) 모드의 초기 가상 네트워크 구성이 포함됩니다. 기본적으로 호스트의 모든 **VM**은 **default** 라는 동일한 **libvirt** 가상 네트워크에 연결됩니다. 이 네트워크의 **VM**은 호스트 및 호스트 이외의 네트워크의 위치에 연결할 수 있지만 다음과 같은 제한 사항이 있습니다.*

- *네트워크의 **VM**은 호스트 및 호스트의 다른 **VM**에 표시되지만, 네트워크 트래픽은 게스트 운영 체제의 네트워크 스택과 게스트 인터페이스에 연결된 **libvirt** 네트워크 필터링 규칙의 영향을 받습니다.*
- *네트워크의 **VM**은 호스트 외부의 위치에 연결할 수 있지만 표시되지 않습니다. 아웃바운드 트래픽은 **NAT** 규칙 및 호스트 시스템의 방화벽의 영향을 받습니다.*

*다음 다이어그램에서는 기본 **VM** 네트워크 구성을 보여줍니다.*



16.2. 가상 머신 네트워크 인터페이스를 관리하기 위해 웹 콘솔 사용

RHEL 9 웹 콘솔을 사용하면 웹 콘솔이 연결된 가상 머신의 가상 네트워크 인터페이스를 관리할 수 있습니다. 다음을 수행할 수 있습니다.

- 네트워크 인터페이스에 대한 정보를 보고 편집합니다.
- 네트워크 인터페이스를 가상 시스템에 추가하고 인터페이스의 연결을 끊거나 삭제합니다.

16.2.1. 웹 콘솔에서 가상 네트워크 인터페이스 정보 보기 및 편집

RHEL 9 웹 콘솔을 사용하면 선택한 VM(가상 머신)에서 가상 네트워크 인터페이스를 보고 수정할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

2. 네트워크 인터페이스로 스크롤합니다.

   Networks Interfaces(네트워크 인터페이스) 섹션에는 VM에 대해 구성된 가상 네트워크 인터페이스와 네트워크 인터페이스 추가,삭제,편집 또는 연결 해제 옵션이 표시됩니다 .

+ 정보에는 다음이 포함됩니다.

- Type - VM의 네트워크 인터페이스 유형입니다. 유형에는 가상 네트워크, LAN에 대한

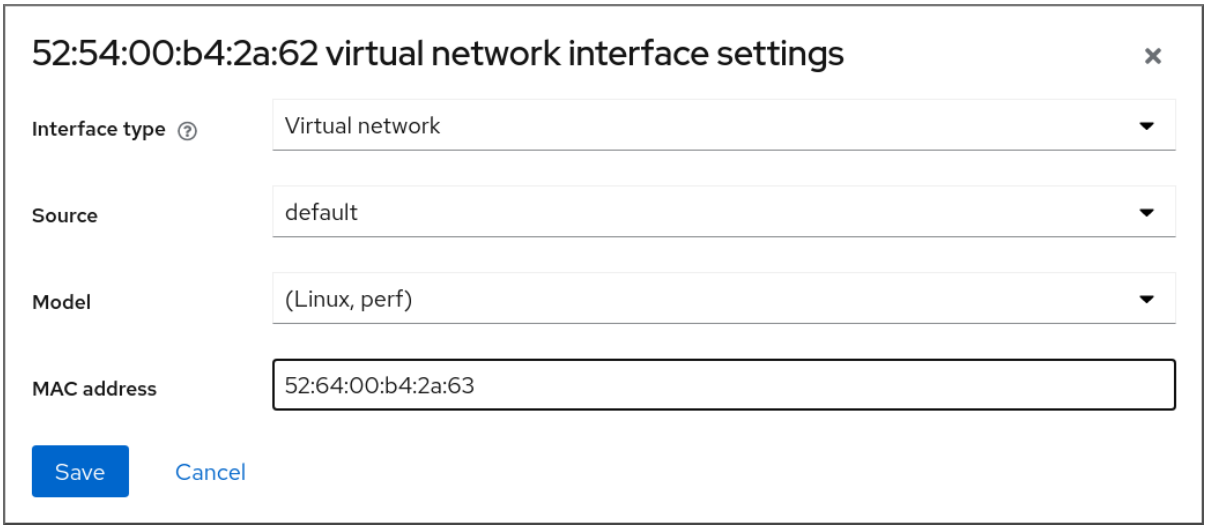
브리지, 직접 연결이 포함됩니다.

참고

***RHEL 9** 이상에서는 일반 이더넷 연결이 지원되지 않습니다.*

- *모델 유형 - 가상 네트워크 인터페이스의 모델입니다.*
- ***MAC** 주소 - 가상 네트워크 인터페이스의 **MAC** 주소입니다.*
- ***IP** 주소 - 가상 네트워크 인터페이스의 **IP** 주소입니다.*
- ***Source** - 네트워크 인터페이스의 소스입니다. 이는 네트워크 유형에 따라 다릅니다.*
- ***State** - 가상 네트워크 인터페이스의 상태입니다.*

3. *가상 네트워크 인터페이스 설정을 편집하려면 편집 을 클릭합니다. 가상 네트워크 인터페이스 설정 대화 상자가 열립니다.*



4. *인터페이스 유형, 소스, 모델 또는 **MAC** 주소를 변경합니다.*

5. *저장을 클릭합니다. 네트워크 인터페이스가 변경되었습니다.*

참고

가상 네트워크 인터페이스 설정 변경 사항은 VM을 다시 시작한 후에만 적용됩니다.

또한 MAC 주소는 VM이 종료된 경우에만 수정할 수 있습니다.

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 정보 보기

16.2.2. 웹 콘솔에서 가상 네트워크 인터페이스 추가 및 연결

RHEL 9 웹 콘솔을 사용하여 가상 네트워크 인터페이스를 생성하고 가상 머신(VM)을 연결할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

2. 네트워크 인터페이스로 스크롤합니다.

   Networks Interfaces(네트워크 인터페이스) 섹션에는 VM에 대해 구성된 가상 네트워크 인터페이스와 네트워크 인터페이스 추가,삭제,편집 또는 연결 옵션이 표시됩니다.

3. 연결하려는 가상 네트워크 인터페이스 행에서 Plug 를 클릭합니다.

   선택한 가상 네트워크 인터페이스가 VM에 연결됩니다.

16.2.3. 웹 콘솔에서 가상 네트워크 인터페이스 연결 해제 및 제거

RHEL 9 웹 콘솔을 사용하면 선택한 VM(가상 머신)에 연결된 가상 네트워크 인터페이스의 연결을 끊을 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

2. 네트워크 인터페이스로 스크롤합니다.

   Networks Interfaces(네트워크 인터페이스) 섹션에는 VM에 대해 구성된 가상 네트워크 인터페이스와 네트워크 인터페이스 추가,삭제,편집 또는 연결 해제 옵션이 표시됩니다 .

3. 연결 해제하려 는 가상 네트워크 인터페이스 행에서 연결 해제를 클릭합니다.

   선택한 가상 네트워크 인터페이스가 VM에서 연결을 끊습니다.

16.3. 권장되는 가상 머신 네트워킹 구성

대부분의 시나리오에서는 기본 VM 네트워킹 구성으로 충분합니다. 그러나 구성을 조정해야 하는 경우 CLI(명령줄 인터페이스) 또는 RHEL 9 웹 콘솔을 사용하여 이를 수행할 수 있습니다. 다음 섹션에서는 이러한 상황에 대해 선택한 VM 네트워크 설정에 대해 설명합니다.

16.3.1. 명령줄 인터페이스를 사용하여 외부적으로 표시되는 가상 머신 구성

기본적으로 새로 생성된 VM은 호스트의 기본 가상 브릿지인 virbr0 을 사용하는 NAT-type 네트워크에 연결됩니다. 이렇게 하면 VM에서 외부 네트워크에 연결하는 데 호스트의 NIC(네트워크 인터페이스 컨트롤러)를 사용할 수 있지만 외부 시스템에서 VM에 연결할 수 없습니다.

VM이 하이퍼바이저와 동일한 외부 네트워크에 표시되도록 하려면 대신 bridged 모드를 사용해야 합니다. 이를 위해 VM을 하이퍼바이저의 물리적 네트워크 장치에 연결된 브릿지 장치에 연결합니다. 이를 위해 명령줄 인터페이스를 사용하려면 아래 지침을 따르십시오.

사전 요구 사항

- 기본 NAT 설정을 사용하는 기존 VM 종료.
- 하이퍼바이저의 IP 구성입니다. 이는 호스트의 네트워크 연결에 따라 다릅니다. 예를 들어, 이 절차에서는 이더넷 케이블을 사용하여 호스트가 네트워크에 연결된 시나리오를 사용하고, 호스트의 물리적 NIC MAC 주소는 DHCP 서버의 고정 IP에 할당됩니다. 따라서 이더넷 인터페이스는 하이퍼바이저 IP로 처리됩니다.

  ethernet 인터페이스의 IP 구성을 가져오려면 ip addr 유틸리티를 사용합니다.

```
# ip addr
[...]
enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP group default qlen 1000
    link/ether 54:ee:75:49:dc:46 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.148/24 brd 10.0.0.255 scope global dynamic noprefixroute enp0s25
```

절차

1. 호스트에서 물리적 인터페이스에 대한 브리지 연결을 만들고 설정합니다. 자세한 내용은 네트워크 브리지 구성을 참조하십시오.

   고정 IP 할당이 사용되는 시나리오에서는 물리적 이더넷 인터페이스의 IPv4 설정을 브리지 인터페이스로 이동해야 합니다.

2. 생성된 브릿지 인터페이스를 사용하도록 VM의 네트워크를 수정합니다. 예를 들어, 다음에서는 bridge0 을 사용하도록 testguest 를 설정합니다.

```
# virt-xml testguest --edit --network bridge=bridge0
Domain 'testguest' defined successfully.
```

3. *VM을 시작합니다.*

```
# virsh start testguest
```

4. *게스트 운영 체제에서 VM이 하이퍼바이저와 동일한 네트워크의 다른 물리적 시스템이었던 것처럼 시스템 네트워크 인터페이스의 IP 및 DHCP 설정을 조정합니다.*

이 특정 단계는 VM에서 사용하는 게스트 OS에 따라 다릅니다. 예를 들어 게스트 OS가 RHEL 9 인 경우 이더넷 연결 구성을 참조하십시오.

검증

1. *새로 생성된 브리지가 실행 중이고 호스트의 물리적 인터페이스와 VM의 인터페이스를 모두 포함합니다.*

```
# ip link show master bridge0
2: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
master bridge0 state UP mode DEFAULT group default qlen 1000
    link/ether 54:ee:75:49:dc:46 brd ff:ff:ff:ff:ff:ff
10: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
master bridge0 state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether fe:54:00:89:15:40 brd ff:ff:ff:ff:ff:ff
```

2. *VM이 하이퍼바이저와 동일한 외부 네트워크에 표시되는지 확인합니다.*

   a. *게스트 운영 체제에서 시스템의 네트워크 ID를 가져옵니다. 예를 들어 Linux 게스트인 경우 다음을 수행합니다.*

```
# ip addr
[...]
enp0s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
state UP group default qlen 1000
    link/ether 52:54:00:09:15:46 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.150/24 brd 10.0.0.255 scope global dynamic noprefixroute enp0s0
```

   b. *로컬 네트워크에 연결된 외부 시스템에서 가져온 ID를 사용하여 VM에 연결합니다.*

```
# ssh root@10.0.0.150
root@10.0.0.150's password:
Last login: Mon Sep 24 12:05:36 2019
root~#*
```

연결이 작동하면 네트워크가 성공적으로 구성되어 있습니다.

문제 해결

- VM이 클라이언트에서 호스팅되는 동안 클라이언트-사이트 VPN을 사용할 때와 같은 특정 상황에서는 VM을 외부 위치에서 사용할 수 있도록 브리지 모드를 사용할 수 없습니다.

  이 문제를 해결하려면 VM에 nftables를 사용하여 대상 NAT를 설정할 수 있습니다.

추가 리소스

- 웹 콘솔을 사용하여 외부에서 볼 수 있는 가상 머신 구성
- 브리지 모드의 가상 네트워킹

16.3.2. 웹 콘솔을 사용하여 외부적으로 표시되는 가상 머신 구성

기본적으로 새로 생성된 VM은 호스트의 기본 가상 브릿지인 virbr0 을 사용하는 NAT-type 네트워크에 연결됩니다. 이렇게 하면 VM에서 외부 네트워크에 연결하는 데 호스트의 NIC(네트워크 인터페이스 컨트롤러)를 사용할 수 있지만 외부 시스템에서 VM에 연결할 수 없습니다.

VM이 하이퍼바이저와 동일한 외부 네트워크에 표시되도록 하려면 대신 bridged 모드를 사용해야 합니다. 이를 위해 VM을 하이퍼바이저의 물리적 네트워크 장치에 연결된 브릿지 장치에 연결합니다. RHEL 9 웹 콘솔을 사용하려면 아래 지침을 따르십시오.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 기본 NAT 설정을 사용하는 기존 VM 종료.
- 하이퍼바이저의 IP 구성입니다. 이는 호스트의 네트워크 연결에 따라 다릅니다. 예를 들어, 이 절차에서는 이더넷 케이블을 사용하여 호스트가 네트워크에 연결된 시나리오를 사용하고, 호스트의 물리적 NIC MAC 주소는 DHCP 서버의 고정 IP에 할당됩니다. 따라서 이더넷 인터페이스는 하이퍼바이저 IP로 처리됩니다.

이더넷 인터페이스의 IP 구성을 가져오려면 웹 콘솔의 네트워킹 탭으로 이동하여 Interfaces 섹션을 참조하십시오.

절차

1. 호스트에서 물리적 인터페이스에 대한 브리지 연결을 만들고 설정합니다. 자세한 내용은 웹 콘솔의 네트워크 브리지 구성을 참조하십시오.

   고정 IP 할당이 사용되는 시나리오에서는 물리적 이더넷 인터페이스의 IPv4 설정을 브리지 인터페이스로 이동해야 합니다.

2. 브리지 인터페이스를 사용하도록 VM의 네트워크를 수정합니다. VM의 네트워크 인터페이스 탭에서 다음을 수행합니다.

   a. 네트워크 인터페이스 추가를클릭합니다.

   b. 가상 네트워크 인터페이스 추가 대화 상자에서 다음을 설정합니다.

      - LAN으로 브리지의 인터페이스 유형
      - 새로 생성된 브리지로의 소스(예: bridge0)

   c. 추가를 클릭합니다.

   d. 선택 사항: VM에 연결된 다른 모든 인터페이스에 대해 Unplug 를 클릭합니다.

3. Run (실행)을 클릭하여 VM을 시작합니다.

4. 게스트 운영 체제에서 VM이 하이퍼바이저와 동일한 네트워크의 다른 물리적 시스템이었던 것처럼 시스템 네트워크 인터페이스의 IP 및 DHCP 설정을 조정합니다.

   이 특정 단계는 VM에서 사용하는 게스트 OS에 따라 다릅니다. 예를 들어 게스트 OS가 RHEL 9 인 경우 이더넷 연결 구성을 참조하십시오.

검증

1. 호스트의 웹 콘솔의 네트워킹 탭에서 새로 생성된 브리지가 있는 행을 클릭하여 실행 중이며 호스트의 물리적 인터페이스와 VM 인터페이스가 모두 포함되어 있는지 확인합니다.

2. VM이 하이퍼바이저와 동일한 외부 네트워크에 표시되는지 확인합니다.

   a. 게스트 운영 체제에서 시스템의 네트워크 ID를 가져옵니다. 예를 들어 Linux 게스트인 경우 다음을 수행합니다.

   ```
   # ip addr
   [...]
   enp0s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
   state UP group default qlen 1000
      link/ether 52:54:00:09:15:46 brd ff:ff:ff:ff:ff:ff
      inet 10.0.0.150/24 brd 10.0.0.255 scope global dynamic noprefixroute enp0s0
   ```

   b. 로컬 네트워크에 연결된 외부 시스템에서 가져온 ID를 사용하여 VM에 연결합니다.

   ```
   # ssh root@10.0.0.150
   root@110.34.5.18's password:
   Last login: Mon Sep 24 12:05:36 2019
   root~#*
   ```

   연결이 작동하면 네트워크가 성공적으로 구성되어 있습니다.

문제 해결

- VM이 클라이언트에서 호스팅되는 동안 클라이언트-사이트 VPN을 사용할 때와 같은 특정 상황에서는 VM을 외부 위치에서 사용할 수 있도록 브리지 모드를 사용할 수 없습니다.

추가 리소스

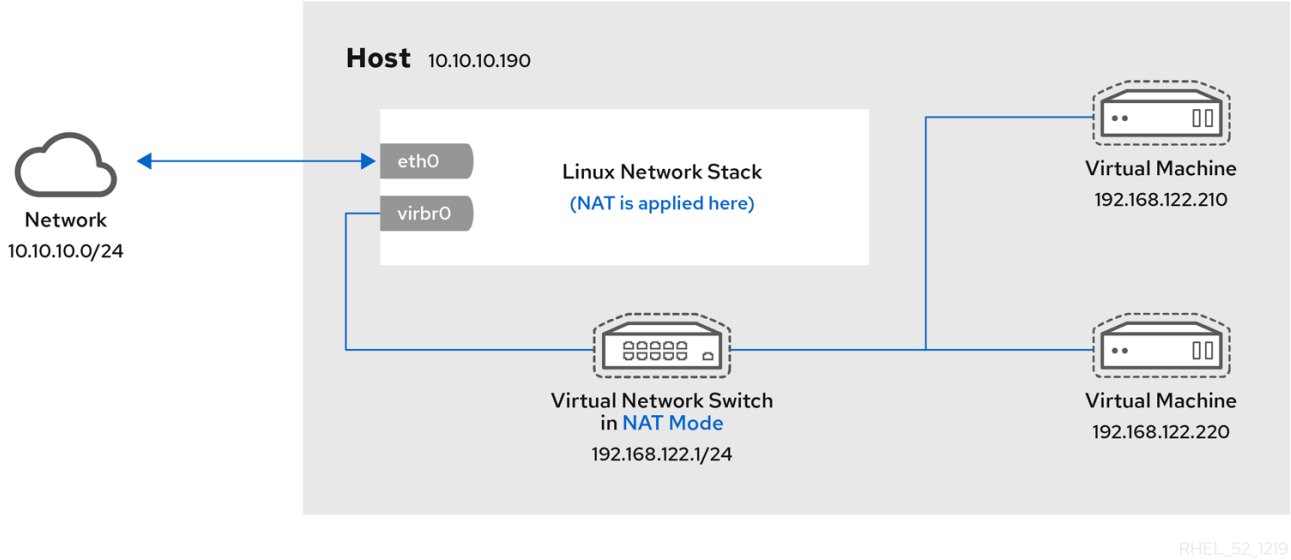
- 명령줄 인터페이스를 사용하여 외부적으로 표시되는 가상 머신 구성
- 브리지 모드의 가상 네트워킹

16.4. 가상 머신 네트워크 연결 유형

VM의 네트워킹 속성 및 동작을 수정하려면 가상 네트워크 또는 VM에서 사용하는 인터페이스의 유형을 변경합니다. 다음 섹션에서는 RHEL 9의 VM에서 사용할 수 있는 연결 유형에 대해 설명합니다.

16.4.1. 네트워크 주소 변환을 사용하는 가상 네트워킹

기본적으로 가상 네트워크 스위치는 NAT(네트워크 주소 변환) 모드에서 작동합니다. NAT(Source-NAT) 또는 Destination-NAT(Destination-NAT) 대신 IP 마스커레이딩을 사용합니다. IP 마스커레이딩을 사용하면 연결된 VM이 호스트 시스템의 IP 주소를 사용하여 외부 네트워크와 통신할 수 있습니다. 가상 네트워크 스위치가 NAT 모드에서 작동할 때 호스트 외부의 컴퓨터는 호스트 내부의 VM과 통신할 수 없습니다.



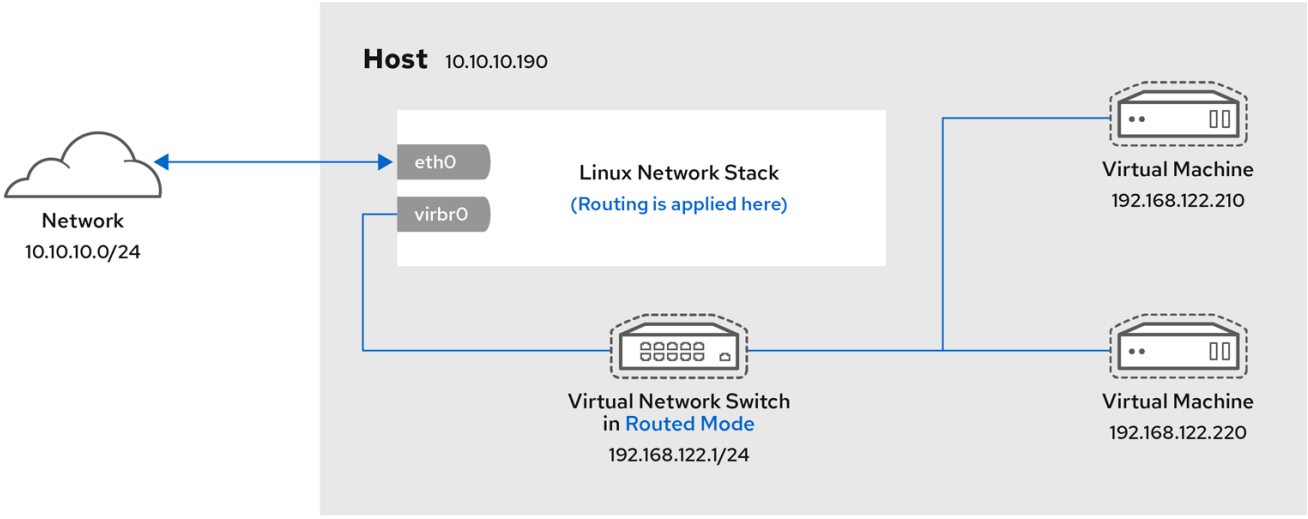
주의

가상 네트워크 스위치는 방화벽 규칙으로 구성된 NAT를 사용합니다. 스위치가 실행되는 동안 이러한 규칙을 편집하는 것은 바람직하지 않습니다. 잘못된 규칙으로 인해 스위치가 통신할 수 없게 될 수 있습니다.

16.4.2. 라우팅 모드의 가상 네트워킹

Routed 모드를 사용하면 가상 스위치가 호스트 시스템에 연결된 실제 LAN에 연결하여 NAT를 사용하지 않고 트래픽을 다시 전달합니다. 가상 스위치는 모든 트래픽을 검사하고 네트워크 패킷에 포함된 정보를 사용하여 라우팅 결정을 내릴 수 있습니다. 이 모드를 사용하는 경우 VM(가상 머신)은 모두 단일 서브넷에 있으며 호스트 시스템과는 분리되어 있습니다. VM 서브넷은 호스트 시스템에 존재하는 가상 스위치를 통해 라우팅됩니다. 이렇게 하면 들어오는 연결을 사용할 수 있지만 외부 네트워크의 시스템에 대해 추가 라우팅 테이블 항목이 필요합니다.

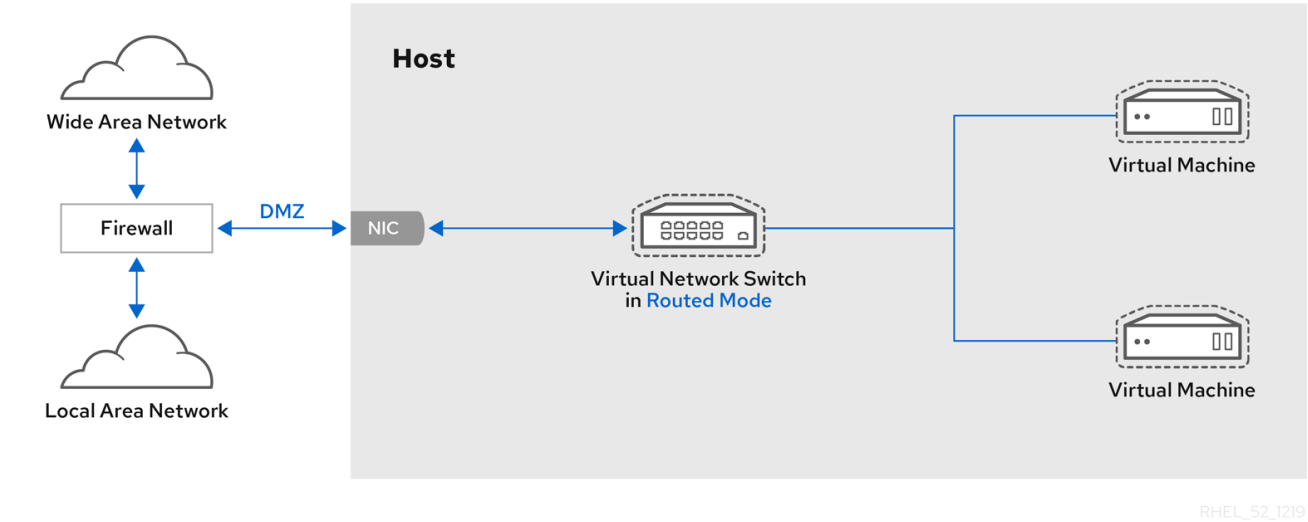
*라우팅 모드는 **IP** 주소를 기반으로 라우팅을 사용합니다.*



*라우팅 모드를 사용하는 일반적인 토폴로지에는 **RuntimeClass** 및 가상 서버 호스팅이 포함됩니다.*

DMZ

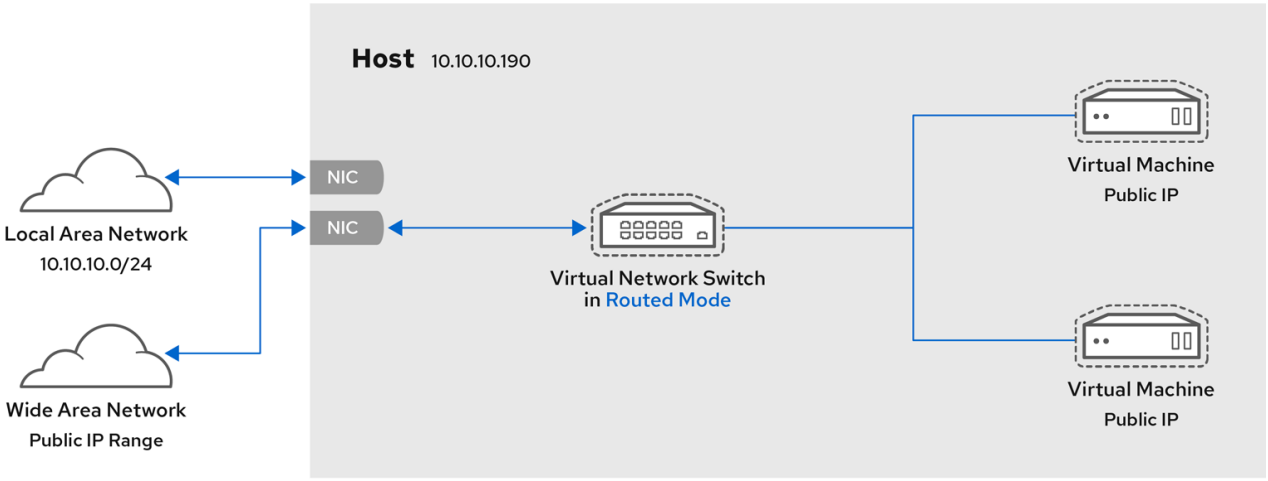
*보안상의 이유로 하나 이상의 노드가 제어되는 하위 네트워크에 배치되는 네트워크를 생성할 수 있습니다. 이러한 하위 네트워크를 **DMZ(demilitarized zone)**라고 합니다.*



***RuntimeClass**의 호스트 머신은 일반적으로 **LAN**(내부) 호스트 시스템과 함께 **WAN**(외부) 호스트 머신에 서비스를 제공합니다. 이를 위해서는 여러 위치에서 액세스할 수 있어야 하며 이러한 위치가 보안 및 신뢰 수준에 따라 제어 및 운영된다는 점을 고려하여 라우팅 모드는 이 환경에 가장 적합한 구성입니다.*

가상 서버 호스팅

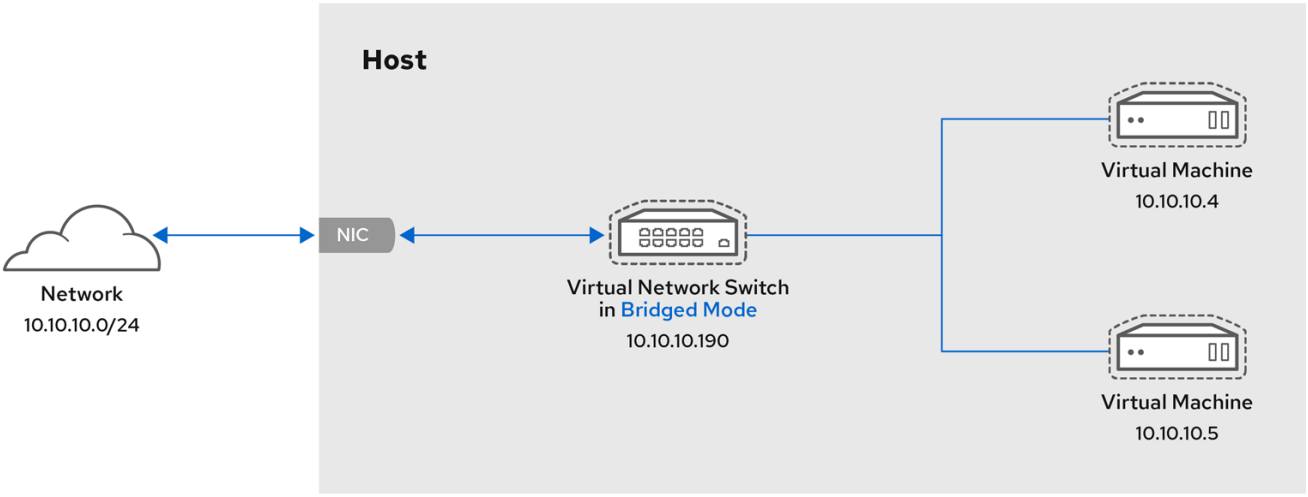
가상 서버 호스팅 공급자에는 각각 두 개의 물리적 네트워크 연결이 있는 여러 호스트 시스템이 있을 수 있습니다. 하나의 인터페이스는 VM이 연결되는 관리 및 회계에 사용됩니다. 각 VM에는 자체 공용 IP 주소가 있지만 호스트 시스템은 내부 관리자만 VM을 관리할 수 있도록 개인 IP 주소를 사용합니다.



16.4.3. 브릿지 모드의 가상 네트워킹

대부분의 VM 네트워킹 모드에서 VM은 virbr0 가상 브릿지를 자동으로 생성하고 연결합니다. 반면 브리지 모드에서 VM은 호스트의 기존 Linux 브리지에 연결됩니다. 결과적으로 VM이 실제 네트워크에 직접 표시됩니다. 이렇게 하면 들어오는 연결이 활성화되지만 추가 라우팅 테이블 항목이 필요하지 않습니다.

bridged 모드에서는 MAC 주소를 기반으로 연결 전환을 사용합니다.



브리지 모드에서 VM이 호스트 시스템과 동일한 서브넷 내에 나타납니다. 동일한 물리적 네트워크에 있는 다른 모든 물리 시스템은 VM을 감지하고 액세스할 수 있습니다.

브리지된 네트워크 본딩

본딩과 함께 결합하여 하이퍼바이저에서 여러 물리적 브릿지 인터페이스를 사용할 수 있습니다. 그런 다음 VM을 브리지에 추가할 수 있는 본딩을 브리지에 추가할 수 있습니다. 그러나 본딩 드라이버에는 여러 가지 작동 모드가 있으며 이러한 모드가 모두 VM이 사용 중인 브릿지와 함께 작동하지는 않습니다.

다음 본딩 모드를 사용할 수 있습니다.

- *모드 1*
- *모드 2*
- *모드 4*

반면 모드 0, 3, 5 또는 6을 사용하면 연결이 실패할 가능성이 높습니다. 또한 ARP(Address Resolution Protocol) 모니터링이 제대로 작동하지 않기 때문에 MII(media-independent interface) 모니터링을 사용하여 본딩 모드를 모니터링해야 합니다.

본딩 모드에 대한 자세한 내용은 Red Hat 지식베이스 를 참조하십시오.

일반적인 시나리오

bridged 모드의 가장 일반적인 사용 사례는 다음과 같습니다.

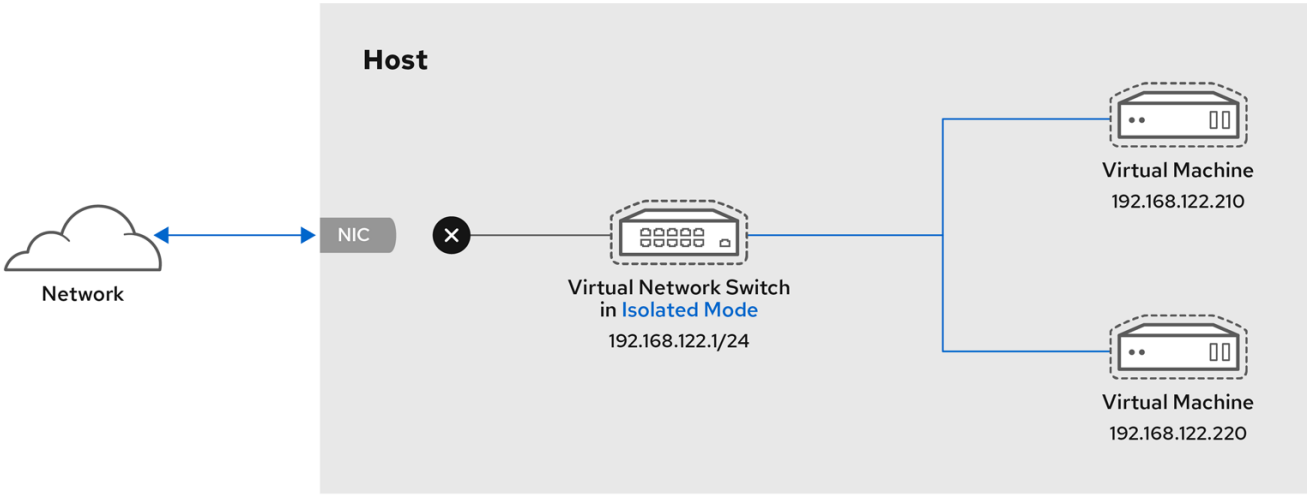
- *호스트 시스템과 함께 기존 네트워크에 VM을 배포하여 가상 시스템과 물리적 시스템 간의 차이를 최종 사용자에게 표시합니다.*
- *기존 물리적 네트워크 구성을 변경하지 않고 VM 배포.*
- *기존 물리적 네트워크에서 쉽게 액세스할 수 있어야 하는 VM 배포. VM을 물리적 네트워크에 배치하여 DHCP 서비스에 액세스해야 합니다.*
- *VLAN(가상 LAN)이 사용되는 기존 네트워크에 VM을 연결합니다.*

추가 리소스

- 명령줄 인터페이스를 사용하여 외부적으로 표시되는 가상 머신 구성
- 웹 콘솔을 사용하여 외부에서 볼 수 있는 가상 머신 구성
- **bridge_opts** 매개변수에 대한 설명

16.4.4. 격리된 모드의 가상 네트워킹

격리된 모드를 사용하는 경우 가상 스위치에 연결된 가상 시스템이 서로 통신할 수 있지만 트래픽은 호스트 시스템 외부에 전달되지 않으며 호스트 시스템 외부에서 트래픽을 수신할 수 없습니다. **DHCP**와 같은 기본 기능을 사용하려면 이 모드에서 **dnsmasq** 를 사용해야 합니다.



16.4.5. 오픈 모드의 가상 네트워킹

네트워킹에 **open** 모드를 사용하는 경우 **libvirt** 는 네트워크에 대한 방화벽 규칙을 생성하지 않습니다. 따라서 **libvirt** 는 호스트에서 제공하는 방화벽 규칙을 덮어쓰지 않으므로 사용자가 **VM**의 방화벽 규칙을 수동으로 관리할 수 있습니다.

16.4.6. 가상 머신 연결 유형 비교

다음 표에서는 선택한 **VM**(가상 머신) 네트워크 구성이 연결할 수 있는 위치와 표시되는 위치에 대한 정보를 제공합니다.

표 **16.1.** 가상 머신 연결 유형

	호스트에 대한 연결	호스트의 다른 VM에 연결	외부 위치에 연결	외부 위치에서 볼 수 있음
브리지 모드	YES	YES	YES	YES
NAT	YES	YES	YES	*제공되지 않음*
라우팅 모드	YES	YES	YES	YES
isolated 모드	YES	YES	*제공되지 않음*	*제공되지 않음*
열려 있는 모드	*호스트의 방화벽 규칙에 따라 달라집니다*			

16.5. PXE 서버에서 가상 머신 부팅

***PXE(Preboot Execution Environment)**를 사용하는 **VM**(가상 머신)은 네트워크에서 구성을 부팅 및 로드할 수 있습니다. 이 장에서는 **libvirt** 를 사용하여 가상 또는 브리지 네트워크의 **PXE** 서버에서 **VM**을 부팅하는 방법을 설명합니다.*

주의

이러한 절차는 예제로만 제공됩니다. 진행하기 전에 충분한 백업이 있는지 확인하십시오.

16.5.1. 가상 네트워크에서 PXE 부팅 서버 설정

*이 절차에서는 **PXE(Preboot Execution Environment)**를 제공하도록 **libvirt** 가상 네트워크를 구성하는 방법을 설명합니다. 이렇게 하면 호스트의 가상 머신을 가상 네트워크에서 사용할 수 있는 부팅 이미지에서 부팅하도록 구성할 수 있습니다.*

사전 요구 사항

- *다음과 같은 로컬 **PXE** 서버(**DHCP** 및 **TFTP**)*
  - ***libvirt** 내부 서버*

- 수동으로 구성된 **dhcpd** 및 **tftpd**
- **dnsmasq**
- 공동 작업 서버

- **Co1.8.0r** 또는 수동으로 구성한 **PXELINUX** 와 같은 **PXE** 부팅 이미지.

절차

1. **PXE** 부팅 이미지와 구성을 **/var/lib/tftpboot** 폴더에 배치합니다.
2. 폴더 권한을 설정합니다.

```
# chmod -R a+r /var/lib/tftpboot
```

3. 폴더 소유권을 설정합니다.

```
# chown -R nobody: /var/lib/tftpboot
```

4. **SELinux** 컨텍스트를 업데이트합니다.

```
# chcon -R --reference /usr/sbin/dnsmasq /var/lib/tftpboot
# chcon -R --reference /usr/libexec/libvirt_leaseshelper /var/lib/tftpboot
```

5. 가상 네트워크를 종료합니다.

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

6. 기본 편집기에서 가상 네트워크 구성 파일을 엽니다.

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

7.

적절한 주소, 네트워크 마스크, DHCP 주소 범위 및 부팅 파일을 포함하도록 < ip > 요소를 편집합니다. 여기서 BOOT_FILENAME 은 부팅 이미지 파일의 이름입니다.

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

8. 가상 네트워크를 시작합니다.

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

검증

- 기본 가상 네트워크가 활성화되어 있는지 확인합니다.

```
# virsh net-list
Name              State    Autostart   Persistent
----------------------------------------------------
default           active   no          no
```

추가 리소스

- PXE 서버에서 TFTP 및 DHCP 구성

16.5.2. PXE 및 가상 네트워크를 사용하여 가상 머신 부팅

가상 네트워크에서 사용 가능한 PXE(Preboot Execution Environment) 서버에서 VM(가상 머신)을 부팅하려면 PXE 부팅을 활성화해야 합니다.

사전 요구 사항

- PXE 부팅 서버는 가상 네트워크의 PXE 부팅 서버 설정에 설명된 대로 가상 네트워크에 설정됩니다.

절차

- PXE 부팅이 활성화된 새 VM을 생성합니다. 예를 들어 기본 가상 네트워크에서 사용 가능한

PXE에서 새 10GB qcow2 이미지 파일로 설치하려면 다음을 실행합니다.

```
# virt-install --pxe --network network=default --memory 2048 --vcpus 2 --disk size=10
```

- 또는 기존 VM의 XML 구성 파일을 수동으로 편집할 수 있습니다.

  i. < os > 요소에 < boot dev='network'/> 요소가 있는지 확인합니다.

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

  ii. 게스트 네트워크가 가상 네트워크를 사용하도록 구성되어 있는지 확인합니다.

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

검증

- virsh start 명령을 사용하여 VM을 시작합니다. PXE가 올바르게 구성된 경우 VM은 PXE 서버에서 사용 가능한 부팅 이미지에서 부팅됩니다.

16.5.3. PXE 및 브릿지 네트워크를 사용하여 가상 머신 부팅

브리지 네트워크에서 사용 가능한 PXE(Preboot Execution Environment) 서버에서 VM(가상 머신)을 부팅하려면 PXE 부팅을 활성화해야 합니다.

사전 요구 사항

- 네트워크 브리징이 활성화되었습니다.
- PXE 부팅 서버는 bridged 네트워크에서 사용할 수 있습니다.

절차

- PXE 부팅이 활성화된 새 VM을 생성합니다. 예를 들어 breth0 브릿지d 네트워크에서 사용 가능한 PXE에서 새 10GB qcow2 이미지 파일로 설치하려면 다음을 실행합니다.

```
# virt-install --pxe --network bridge=breth0 --memory 2048 --vcpus 2 --disk size=10
```

  - 또는 기존 VM의 XML 구성 파일을 수동으로 편집할 수 있습니다.

    i. < os > 요소에 < boot dev='network'/> 요소가 있는지 확인합니다.

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

    ii. VM이 bridged 네트워크를 사용하도록 구성되어 있는지 확인합니다.

```
<interface type='bridge'>
  <mac address='52:54:00:5a:ad:cb'/>
  <source bridge='breth0'/>
  <target dev='vnet0'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

검증

- virsh start 명령을 사용하여 VM을 시작합니다. PXE가 올바르게 구성된 경우 VM은 PXE 서버에서 사용 가능한 부팅 이미지에서 부팅됩니다.

추가 리소스

- 네트워크 브리지 구성

16.6. 추가 리소스

- 네트워킹 구성 및 관리

- *특정 네트워크 인터페이스 카드를 SR-IOV 장치로 연결하여 VM 성능을 향상시킵니다.*

17장. 가상 머신 성능 최적화

가상 머신(VM)은 항상 호스트와 비교하여 어느 정도의 성능 저하를 경험합니다. 다음 섹션에서는 이러한 거부의 이유를 설명하고 RHEL 9에서 가상화의 성능 영향을 최소화하는 방법에 대한 지침을 제공하여 하드웨어 인프라 리소스를 최대한 효율적으로 사용할 수 있도록 합니다.

17.1. 가상 머신 성능에 미치는 영향

VM은 호스트에서 사용자 공간 프로세스로 실행됩니다. 따라서 하이퍼바이저는 VM이 사용할 수 있도록 호스트의 시스템 리소스를 변환해야 합니다. 결과적으로 리소스의 일부가 변환에서 소비되므로 VM은 호스트와 동일한 성능 효율성을 달성할 수 없습니다.

시스템 성능에 가상화의 영향

VM 성능 손실에 대한 구체적인 이유는 다음과 같습니다.

- 가상 CPU(vCPU)는 Linux 스케줄러에서 처리하는 호스트에서 스레드로 구현됩니다.
- VM은 호스트 커널에서 NUMA 또는 대규모 페이지와 같은 최적화 기능을 자동으로 상속하지 않습니다.
- 호스트의 디스크 및 네트워크 I/O 설정은 VM에 상당한 성능 영향을 미칠 수 있습니다.
- 네트워크 트래픽은 일반적으로 소프트웨어 기반 브릿지를 통해 VM으로 이동합니다.
- 호스트 장치 및 모델에 따라 특정 하드웨어의 에뮬레이션으로 인해 상당한 오버헤드가 발생할 수 있습니다.

VM 성능에 미치는 가상화의 심각도는 다음과 같은 다양한 요인의 영향을 받습니다.

- 동시에 실행되는 VM 수입니다.
- 각 VM에서 사용하는 가상 장치의 양입니다.

- VM에서 사용하는 장치 유형입니다.

VM 성능 손실 감소

RHEL 9는 가상화의 부정적인 성능 영향을 줄이는 데 사용할 수 있는 다양한 기능을 제공합니다. 주요 사항:

- TuneD 서비스는 VM의 리소스 배포 및 성능을 자동으로 최적화할 수 있습니다.
- 블록 I/O 튜닝을 사용하면 디스크와 같은 VM 블록 장치의 성능을 향상시킬 수 있습니다.
- NUMA 튜닝 은 vCPU 성능을 향상시킬 수 있습니다.
- 가상 네트워킹 을 다양한 방식으로 최적화할 수 있습니다.

중요

VM 성능 튜닝은 다른 가상화 기능에 부정적인 영향을 미칠 수 있습니다. 예를 들어 수정된 VM을 보다 쉽게 마이그레이션할 수 있습니다.

17.2. TUNED를 사용하여 가상 머신 성능 최적화

TuneD 유틸리티는 CPU 집약적인 작업 또는 스토리지 네트워크 처리량 응답성 요구 사항과 같은 특정 워크로드 특성에 RHEL을 조정하는 튜닝 프로필 전달 메커니즘입니다. 성능을 향상시키고 여러 가지 특정 사용 사례에서 전력 소비를 줄이기 위해 사전 구성된 여러 튜닝 프로필을 제공합니다. 이러한 프로필을 편집하거나 새 프로필을 생성하여 가상화 환경을 비롯한 환경에 맞는 성능 솔루션을 생성할 수 있습니다.

가상화를 위해 RHEL 9를 최적화하려면 다음 프로필을 사용하십시오.

- RHEL 9 가상 머신의 경우 virtual-guest 프로필을 사용합니다. 일반적으로 적용 가능한 처리량-성능 프로필을 기반으로 하지만 가상 메모리의 스왑성을 줄입니다.
- RHEL 9 가상화 호스트의 경우 virtual-host 프로필을 사용합니다. 이를 통해 더 적극적으로 더티 메모리 페이지의 쓰기를 통해 호스트 성능이 향상됩니다.

사전 요구 사항

- **TuneD** 서비스가 *설치되고 활성화됩니다*.

절차

*특정 **TuneD** 프로필을 활성화하려면 다음을 수행합니다.*

1. *사용 가능한 **TuneD** 프로필을 나열합니다.*

```
# tuned-adm list

Available profiles:
- balanced              - General non-specialized TuneD profile
- desktop               - Optimize for the desktop use-case
[...]
- virtual-guest         - Optimize for running inside a virtual guest
- virtual-host          - Optimize for running KVM guests
Current active profile: balanced
```

2. *선택 사항: 새 **TuneD** 프로필을 생성하거나 기존 **TuneD** 프로필을 편집합니다.*

   *자세한 내용은 **TuneD** 프로필 사용자 지정을 참조하십시오.*

3. ***TuneD** 프로필을 활성화합니다.*

```
# tuned-adm profile selected-profile
```

   - *가상화 호스트를 최적화하려면 **virtual-host** 프로필을 사용합니다.*

```
# tuned-adm profile virtual-host
```

   - ***RHEL** 게스트 운영 체제에서 **virtual-guest** 프로필을 사용합니다.*

```
# tuned-adm profile virtual-guest
```

추가 리소스

- *시스템 상태 및 성능 모니터링 및 관리*

17.3. LIBVIRT 데몬 최적화

libvirt 가상화 제품군은 **RHEL** 하이퍼바이저의 관리 계층으로 작동하며 **libvirt** 구성은 가상화 호스트에 큰 영향을 미칩니다. 특히 **RHEL 9**에는 두 가지 유형의 **libvirt** 데몬, 모놀리식 또는 모듈식, 사용하는 데몬 유형이 개별 가상화 드라이버를 구성하는 방법에 영향을 미칩니다.

17.3.1. libvirt 데몬 유형

RHEL 9에서는 다음과 같은 **libvirt** 데몬 유형을 지원합니다.

모놀리식 **libvirt**

기존 **libvirt** 데몬인 **libvirtd** 는 단일 구성 파일인 **/etc/libvirt/libvirtd.conf** 를 사용하여 다양한 가상화 드라이버를 제어합니다.

따라서 **libvirtd** 는 중앙 집중식 하이퍼바이저 구성을 허용하지만 시스템 리소스를 효율적으로 사용할 수 있습니다. 따라서 **libvirtd** 는 향후 **RHEL**의 주요 릴리스에서 지원되지 않습니다.

그러나 **RHEL 8**에서 **RHEL 9**로 업데이트한 경우 호스트는 기본적으로 **libvirtd** 를 사용합니다.

모듈식 **libvirt**

RHEL 9에 새로 도입된 모듈식 **libvirt** 는 각 가상화 드라이버에 대해 특정 데몬을 제공합니다. 여기에는 다음이 포함됩니다.

- **virtqemud -** 하이퍼바이저 관리를 위한 기본 데몬
- **virtinterfaced -** 호스트 **NIC** 관리를 위한 보조 데몬
- **virtnetworkd -** 가상 네트워크 관리를 위한 보조 데몬
- **virtnodedevd -** 호스트 물리적 장치 관리를 위한 보조 데몬

- **virtnwfilterd** - 호스트 방화벽 관리를 위한 보조 데몬
- **virtsecretd** - 호스트 시크릿 관리를 위한 보조 데몬
- **virtstoraged** - 스토리지 관리를 위한 보조 데몬

각 데몬에는 별도의 구성 파일(예: **/etc/libvirt/virtqemud.conf**)이 있습니다. 따라서 모듈식 **libvirt** 데몬은 **libvirt** 리소스 관리를 위한 더 나은 옵션을 제공합니다.

RHEL 9 새로 설치를 수행한 경우 기본적으로 모듈식 **libvirt** 가 구성됩니다.

다음 단계

- **RHEL 9**에서 **libvirtd** 을 사용하는 경우 모듈식 데몬으로 전환하는 것이 좋습니다. 자세한 내용은 모듈식 **libvirt** 데몬 활성화를 참조하십시오.

17.3.2. 모듈식 libvirt 데몬 활성화

RHEL 9에서 **libvirt** 라이브러리는 호스트의 개별 가상화 드라이버 세트를 처리하는 모듈식 데몬을 사용합니다. 예를 들어 **virtqemud** 데몬은 **QEMU** 드라이버를 처리합니다.

RHEL 9 호스트 새로 설치를 수행한 경우 하이퍼바이저는 기본적으로 모듈식 **libvirt** 데몬을 사용합니다. 그러나 호스트를 **RHEL 8**에서 **RHEL 9**로 업그레이드한 경우 하이퍼바이저는 **RHEL 8**의 기본값인 모놀리식 **libvirtd** 데몬을 사용합니다.

이러한 경우 **libvirt** 리소스 관리를 위해 더 나은 옵션을 제공하기 때문에 **Red Hat**은 모듈화된 **libvirt** 데몬을 활성화하는 것이 좋습니다. 또한 **libvirtd** 는 향후 **RHEL**의 주요 릴리스에서 지원되지 않습니다.

사전 요구 사항

- 하이퍼바이저가 모놀리식 **libvirtd** 서비스를 사용하고 있습니다. 이 경우인지 알아보려면 다음을 수행합니다.

```
# systemctl is-active libvirtd.service
active
```

이 명령이 활성 을 표시하는 경우 **libvirtd** 를 사용하고 있습니다.

- 가상 머신이 종료됩니다.

절차

1. **libvirtd** 및 해당 소켓을 중지합니다.

```
# systemctl stop libvirtd.service
# systemctl stop libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

2. **libvirtd** 를 비활성화하여 부팅 시 시작되지 않도록 합니다.

```
$ systemctl disable libvirtd.service
$ systemctl disable libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

3. 모듈식 **libvirt** 데몬을 활성화합니다.

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

4. 모듈식 데몬의 소켓을 시작합니다.

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

5. 선택 사항: 원격 호스트에서 호스트에 연결해야 하는 경우 가상화 프록시 데몬을 활성화하고 시작합니다.

```
# systemctl unmask virtproxyd.service
# systemctl unmask virtproxyd{,-ro,-admin,-tls}.socket
# systemctl enable virtproxyd.service
# systemctl enable virtproxyd{,-ro,-admin,-tls}.socket
# systemctl start virtproxyd{,-ro,-admin,-tls}.socket
```

검증

1. 활성화된 가상화 데몬을 활성화합니다.

```
# virsh uri
qemu:///system
```

2. 호스트가 **virtqemud** 모듈식 데몬을 사용하고 있는지 확인합니다.

```
# systemctl is-active virtqemud.service
active
```

이 명령에서 활성 을 표시하는 경우 모듈식 **libvirt** 데몬을 성공적으로 활성화했습니다.

17.4. 가상 머신 메모리 구성

VM(가상 머신)의 성능을 개선하기 위해 **VM**에 호스트 **RAM**을 추가로 할당할 수 있습니다. 마찬가지로 **VM**에 할당된 메모리 양을 줄일 수 있으므로 호스트 메모리를 다른 **VM** 또는 작업에 할당할 수 있습니다.

이러한 작업을 수행하려면 웹 콘솔 또는 명령줄 인터페이스를 사용할 수 있습니다.

17.4.1. 웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거

VM(가상 머신)의 성능을 개선하거나 사용 중인 호스트 리소스를 확보하려면 웹 콘솔을 사용하여 **VM**에 할당된 메모리 양을 조정할 수 있습니다.

사전 요구 사항

- 게스트 **OS**는 메모리 **balloon** 드라이버를 실행하고 있습니다. 이를 확인하려면 다음을 수행하십시오.

  1. **VM** 구성에 **memballoon** 장치가 포함되어 있는지 확인합니다.

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

이 명령이 출력을 표시하고 모델이 **none** 으로 설정되지 않은 경우 **memballoon** 장치가 있습니다.

2. guest OS에서 balloon 드라이버가 실행 중인지 확인합니다.

   - Windows 게스트에서 드라이버는 virtio-win 드라이버 패키지의 일부로 설치됩니다. 자세한 내용은 Windows 가상 머신 용 KVM 반가상화 드라이버 설치를 참조하십시오.

   - Linux 게스트에서는 일반적으로 드라이버는 기본적으로 포함되어 있으며 memballoon 장치가 있는 경우 활성화됩니다.

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. 선택 사항: 최대 메모리와 현재 VM에 사용된 메모리에 대한 정보를 가져옵니다. 이는 변경 사항에 대한 기준선 역할을 하고 확인에도 적용됩니다.

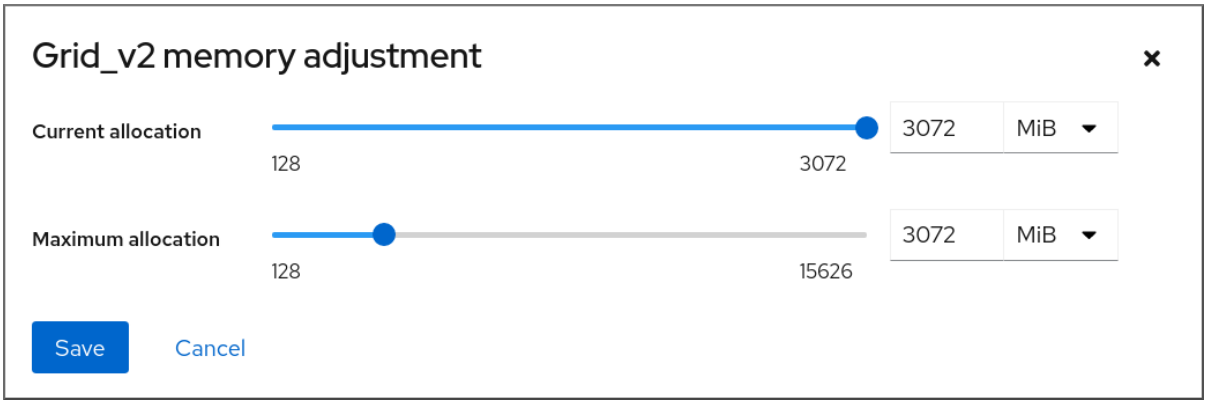
```
# virsh dominfo testguest
Max memory:     2097152 KiB
Used memory:    2097152 KiB
```

2. Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

3. Overview(개요) 창에서 Memory(메모리) 행 옆에 있는 Edit(편집)를 클릭합니다.

   메모리 조정 대화 상자가 나타납니다.



4. 선택한 VM의 가상 CPU를 구성합니다.

- 최대 할당 - VM에서 프로세스에 사용할 수 있는 최대 호스트 메모리 양을 설정합니다. VM을 생성할 때 최대 메모리를 지정하거나 나중에 늘릴 수 있습니다. 메모리를 MiB 또는 GiB의 배수로 지정할 수 있습니다.

  최대 메모리 할당 조정은 종료 VM에서만 가능합니다.

- 현재 할당 - VM에 할당된 실제 메모리 양을 설정합니다. 이 값은 최대 할당보다 작을 수 있지만 초과할 수는 없습니다. 프로세스에 대해 VM에서 사용 가능한 메모리를 규제하도록 값을 조정할 수 있습니다. 메모리를 MiB 또는 GiB의 배수로 지정할 수 있습니다.

  이 값을 지정하지 않으면 기본 할당은 최대 할당 값입니다.

5. 저장을 클릭합니다.

  VM의 메모리 할당이 조정됩니다.

추가 리소스

- 명령줄 인터페이스를 사용하여 가상 머신 메모리 추가 및 제거
- 가상 머신 CPU 성능 최적화

17.4.2. 명령줄 인터페이스를 사용하여 가상 머신 메모리 추가 및 제거

VM(가상 머신)의 성능을 개선하거나 사용 중인 호스트 리소스를 확보하려면 CLI를 사용하여 VM에 할당된 메모리 양을 조정할 수 있습니다.

사전 요구 사항

- *게스트 OS는 메모리 balloon 드라이버를 실행하고 있습니다. 이를 확인하려면 다음을 수행하십시오.*

  1. *VM 구성에 memballoon 장치가 포함되어 있는지 확인합니다.*

     ```
     # virsh dumpxml testguest | grep memballoon
     <memballoon model='virtio'>
       </memballoon>
     ```

     이 명령이 출력을 표시하고 모델이 none 으로 설정되지 않은 경우 memballoon 장치가 있습니다.

  2. *게스트 OS에서 볼론 드라이버가 설치되어 있는지 확인합니다.*

     - *Windows 게스트에서 드라이버는 virtio-win 드라이버 패키지의 일부로 설치됩니다. 자세한 내용은 Windows 가상 머신 용 KVM 반가상화 드라이버 설치를 참조하십시오.*

     - *Linux 게스트에서는 일반적으로 드라이버는 기본적으로 포함되어 있으며 memballoon 장치가 있는 경우 활성화됩니다.*

절차

1. *선택 사항: 최대 메모리와 현재 VM에 사용된 메모리에 대한 정보를 가져옵니다. 이는 변경 사항에 대한 기준선 역할을 하고 확인에도 적용됩니다.*

   ```
   # virsh dominfo testguest
   Max memory:     2097152 KiB
   Used memory:    2097152 KiB
   ```

2. *VM에 할당된 최대 메모리를 조정합니다. 이 값을 늘리면 VM의 성능이 향상되고 VM이 호스트에 있는 성능 풋프린트를 줄일 수 있습니다. 이러한 변경은 종료 VM에서만 수행할 수 있으므로 실행 중인 VM을 조정하려면 재부팅이 필요합니다.*

   예를 들어 testguest VM에서 4096MiB로 사용할 수 있는 최대 메모리를 변경하려면 다음을

수행합니다.

```
# virt-xml testguest --edit --memory memory=4096,currentMemory=4096
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

실행 중인 VM의 최대 메모리를 늘리려면 메모리 장치를 VM에 연결할 수 있습니다. 이를 메모리 핫 플러그 라고도 합니다. 자세한 내용은 메모리 장치를 가상 머신에 연결을 참조하십시오.

주의

실행 중인 VM에서 메모리 장치 제거(메모리 핫 플러그라고도 함)은 지원되지 않으며 Red Hat에서 사용하지 않는 경우가 많습니다.

3. 선택 사항: VM에서 현재 사용하는 메모리를 최대 할당까지 조정할 수도 있습니다. 이렇게 하면 최대 VM 할당을 변경하지 않고 다음 재부팅까지 VM에 있는 메모리 로드가 조정됩니다.

```
# virsh setmem testguest --current 2048
```

검증

1. VM에서 사용하는 메모리가 업데이트되었는지 확인합니다.

```
# virsh dominfo testguest
Max memory:     4194304 KiB
Used memory:    2097152 KiB
```

2. 선택 사항: 현재 VM 메모리를 조정한 경우 VM의 메모리 balloon 통계를 가져와 메모리 사용량을 얼마나 효과적으로 규제하는지 평가할 수 있습니다.

```
 # virsh domstats --balloon testguest
Domain: 'testguest'
  balloon.current=365624
  balloon.maximum=4194304
  balloon.swap_in=0
  balloon.swap_out=0
  balloon.major_fault=306
  balloon.minor_fault=156117
```

```
balloon.unused=3834448
balloon.available=4035008
balloon.usable=3746340
balloon.last-update=1587971682
balloon.disk_caches=75444
balloon.hugetlb_pgalloc=0
balloon.hugetlb_pgfail=0
balloon.rss=1005456
```

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거
- 가상 머신 CPU 성능 최적화

17.4.3. 추가 리소스

- 가상 머신에 장치 연결.

17.5. 가상 머신 I/O 성능 최적화

VM(가상 머신)의 입력 및 출력(I/O) 기능은 VM의 전체 효율성을 크게 제한할 수 있습니다. 이를 해결하기 위해 블록 I/O 매개 변수를 구성하여 VM의 I/O를 최적화할 수 있습니다.

17.5.1. 가상 머신에서 블록 I/O 튜닝

하나 이상의 VM에서 여러 블록 장치를 사용하는 경우 I/O 가중치 를 수정하여 특정 가상 장치의 I/O 우선 순위를 조정해야 할 수 있습니다.

장치의 I/O 가중치를 늘리면 I/O 대역폭에 대한 우선순위가 높아지고 호스트 리소스가 추가됩니다. 마찬가지로 장치의 가중치를 줄이면 호스트 리소스가 줄어듭니다.

참고

각 장치의 가중치 값은 100 ~1000 범위 내에 있어야 합니다. 또는 값은 장치별 목록에서 해당 장치를 제거하는 0 일 수 있습니다.

절차

VM의 블록 I/O 매개 변수를 표시하고 설정하려면 다음을 수행합니다.

1. VM의 현재 <blkio> 매개변수를 표시합니다.

```
# virsh dumpxml VM-name
```

```
<domain>
 [...]
 <blkiotune>
  <weight>800</weight>
  <device>
   <path>/dev/sda</path>
   <weight>1000</weight>
  </device>
  <device>
   <path>/dev/sdb</path>
   <weight>500</weight>
  </device>
 </blkiotune>
 [...]
</domain>
```

2. 지정된 장치의 I/O 가중치를 편집합니다.

```
# virsh blkiotune VM-name --device-weights device, I/O-weight
```

예를 들어, 다음은 리프트 VM의 /dev/sda 장치의 가중치를 500으로 변경합니다.

```
# virsh blkiotune liftbrul --device-weights /dev/sda, 500
```

17.5.2. 가상 머신의 디스크 I/O 제한

여러 VM이 동시에 실행되는 경우 과도한 디스크 I/O를 사용하여 시스템 성능을 방해할 수 있습니다. KVM 가상화의 디스크 I/O 제한은 VM에서 호스트 시스템으로 전송된 디스크 I/O 요청에 대한 제한을 설정하는 기능을 제공합니다. 이렇게 하면 VM이 공유 리소스를 과도하게 활용하고 다른 VM의 성능에 영향을 미칠 수 있습니다.

디스크 I/O 제한을 활성화하려면 VM에 연결된 각 블록 장치에서 호스트 시스템으로 전송된 디스크 I/O 요청에 제한을 설정합니다.

절차

1. *virsh domblklist 명령을 사용하여 지정된 VM의 모든 디스크 장치의 이름을 나열합니다.*

```
# virsh domblklist rollin-coal
Target     Source
------------------------------------------------
vda        /var/lib/libvirt/images/rollin-coal.qcow2
sda        -
sdb        /home/horridly-demanding-processes.iso
```

2. *제한하려는 가상 디스크가 마운트된 호스트 블록 장치를 찾습니다.*

예를 들어 이전 단계에서 sdb 가상 디스크를 제한하려면 다음 출력에서 디스크가 /dev/nvme0n1p3 파티션에 마운트되었음을 보여줍니다.

```
$ lsblk
NAME                                          MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
zram0                                         252:0    0     4G  0 disk  [SWAP]
nvme0n1                                       259:0    0 238.5G  0 disk
├─nvme0n1p1                                   259:1    0   600M  0 part  /boot/efi
├─nvme0n1p2                                   259:2    0     1G  0 part  /boot
└─nvme0n1p3                                   259:3    0 236.9G  0 part
  └─luks-a1123911-6f37-463c-b4eb-fxzy1ac12fea 253:0    0 236.9G  0 crypt /home
```

3. *virsh blkiotune 명령을 사용하여 블록 장치에 대한 I/O 제한을 설정합니다.*

```
# virsh blkiotune VM-name --parameter device,limit
```

다음 예제에서는 Rollin-coal VM의 sdb 디스크를 초당 1000 읽기 및 쓰기 I/O 작업과 초당 50MB로 제한합니다.

```
# virsh blkiotune rollin-coal --device-read-iops-sec /dev/nvme0n1p3,1000 --device-
write-iops-sec /dev/nvme0n1p3,1000 --device-write-bytes-sec
/dev/nvme0n1p3,52428800 --device-read-bytes-sec /dev/nvme0n1p3,52428800
```

추가 정보

- *디스크 I/O 제한은 서로 다른 고객에 속하는 VM이 동일한 호스트에서 실행 중인 경우 또는 다른 VM에 대해 서비스 품질 보장이 제공되는 경우와 같이 다양한 상황에서 유용할 수 있습니다. 디스크 I/O 제한을 사용하여 더 느린 디스크를 시뮬레이션할 수도 있습니다.*

- *I/O 제한은 VM에 연결된 각 블록 장치에 독립적으로 적용할 수 있으며 처리량 및 I/O 작업에 대한 제한을 지원합니다.*

- Red Hat은 virsh blkdeviotune 명령을 사용하여 VM에서 I/O 제한을 구성할 수 없습니다. RHEL 9를 VM 호스트로 사용할 때 지원되지 않는 기능에 대한 자세한 내용은 RHEL 9 가상화의 지원되지 않는 기능을 참조하십시오.

17.5.3. 다중 대기열 virtio-scsi 활성화

가상 머신(VM)에서 virtio-scsi 스토리지 장치를 사용하는 경우 다중 대기열 virtio-scsi 기능은 향상된 스토리지 성능과 확장성을 제공합니다. 이를 통해 각 가상 CPU(vCPU)는 다른 vCPU에 영향을 주지 않고 별도의 대기열과 인터럽트를 사용할 수 있습니다.

절차

- 특정 VM에 대한 다중 대기열 virtio-scsi 지원을 활성화하려면 VM의 XML 구성에 다음을 추가합니다. 여기서 N 은 vCPU 대기열의 총 수입니다.

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

17.6. 가상 머신 CPU 성능 최적화

호스트 시스템의 물리적 CPU와 마찬가지로 vCPU는 VM(가상 머신) 성능에 매우 중요합니다. 따라서 vCPU를 최적화하면 VM의 리소스 효율성에 큰 영향을 미칠 수 있습니다. vCPU를 최적화하려면 다음을 수행합니다.

1. VM에 할당되는 호스트 CPU 수를 조정합니다. CLI 또는 웹 콘솔 을 사용하여 이 작업을 수행할 수 있습니다.

2. vCPU 모델이 호스트의 CPU 모델과 일치하는지 확인합니다. 예를 들어 호스트의 CPU 모델을 사용하도록 testguest1 VM을 설정하려면 다음을 수행합니다.

```
# virt-xml testguest1 --edit --cpu host-model
```

3. 커널 동일 페이지 병합(KSM)을 관리합니다.

4. 호스트 시스템에서 NUMA(Non-Uniform Memory Access)를 사용하는 경우 해당 VM에 대해 NUMA를 구성할 수도 있습니다. 이는 호스트의 CPU 및 메모리 프로세스를 VM의 CPU 및 메

모리 프로세스에 가능한 한 가깝게 매핑합니다. 실제로 NUMA 튜닝에서는 VM에 할당된 시스템 메모리에 보다 효율적으로 액세스할 수 있는 vCPU를 제공하므로 vCPU 처리 효율성을 향상시킬 수 있습니다.

자세한 내용은 가상 머신에서 NUMA 구성 및 샘플 vCPU 성능 튜닝 시나리오를 참조하십시오.

17.6.1. 명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거

VM(가상 머신)의 CPU 성능을 늘리거나 최적화하기 위해 VM에 할당된 가상 CPU(vCPU)를 추가하거나 제거할 수 있습니다.

실행 중인 VM에서 수행할 때 vCPU 핫 플러그 및 핫플러그라고도 합니다. 그러나 RHEL 9에서는 vCPU 핫 플러그가 지원되지 않으며 Red Hat은 사용을 권장하지 않습니다.

사전 요구 사항

- 선택 사항: 대상 VM에서 vCPU의 현재 상태를 확인합니다. 예를 들어 testguest VM에 vCPU 수를 표시하려면 다음을 수행합니다.

```
# virsh vcpucount testguest
maximum      config         4
maximum      live           2
current      config         2
current      live           1
```

이 출력은 testguest 가 현재 1 vCPU를 사용하고 있으며 VM 성능을 높이기 위해 1개 이상의 vCPu를 핫 플러그로 연결할 수 있음을 나타냅니다. 그러나 재부팅 후 vCPU 테스트 게스트 수가 2로 변경되며 2개의 vCPU를 더 핫플러그할 수 있습니다.

절차

1. VM에 연결할 수 있는 최대 vCPU 수를 조정합니다. VM의 다음 부팅에 적용됩니다.

예를 들어 testguest VM의 최대 vCPU 수를 8로 늘리려면 다음을 수행합니다.

```
# virsh setvcpus testguest 8 --maximum --config
```

CPU 토폴로지, 호스트 하드웨어, 하이퍼바이저 및 기타 요인에 따라 최대값을 제한할 수 있습니다.

2. VM에 연결된 현재 vCPU 수를 이전 단계에서 구성한 최대값까지 조정합니다. 예를 들면 다음과 같습니다.

- 실행 중인 testguest VM에 연결된 vCPU 수를 4로 늘리려면 다음을 실행합니다.

```
# virsh setvcpus testguest 4 --live
```

이로 인해 VM의 다음 부팅이 될 때까지 testguest 의 VM 성능 및 호스트 로드 공간이 증가합니다.

- testguest VM에 연결된 vCPU 수를 1로 영구적으로 축소하려면 다음을 수행합니다.

```
# virsh setvcpus testguest 1 --config
```

이렇게 하면 VM의 다음 부팅 후 testguest 의 VM 성능 및 호스트 로드 공간이 줄어듭니다. 그러나 필요한 경우 VM에 추가 vCPU를 핫플러그하여 일시적으로 성능을 향상시킬 수 있습니다.

검증

- VM의 현재 vCPU 상태가 변경 사항을 반영하는지 확인합니다.

```
# virsh vcpucount testguest
maximum      config         8
maximum      live           4
current      config         1
current      live           4
```

추가 리소스

- 웹 콘솔을 사용하여 가상 CPU 관리

17.6.2. 웹 콘솔을 사용하여 가상 CPU 관리

RHEL 9 웹 콘솔을 사용하면 웹 콘솔이 연결된 VM(가상 머신)에서 사용하는 가상 CPU를 검토하고 구성할 수 있습니다.

사전 요구 사항

- *웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.*

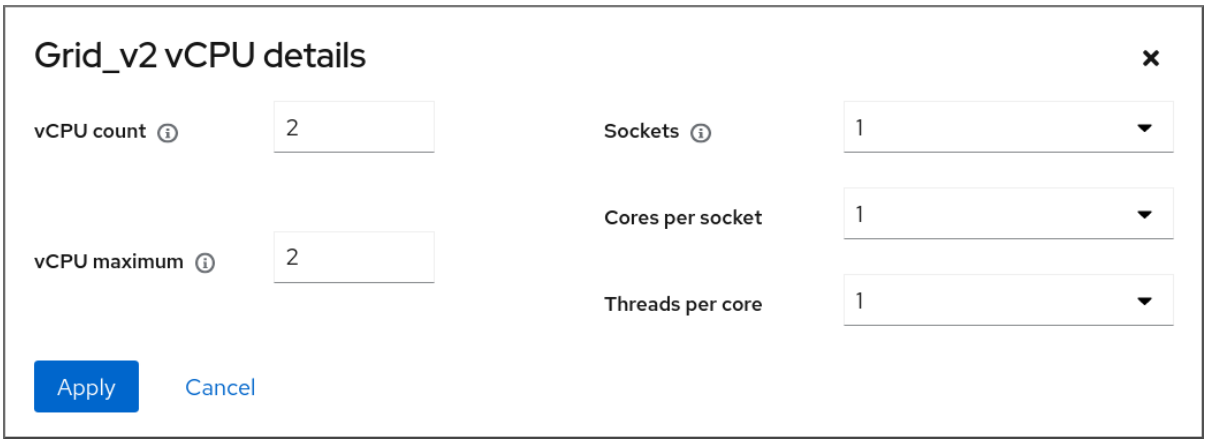
절차

1. *Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.*

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

2. *Overview(개요) 창에서 vCPU의 수 옆에 있는 Edit(편집)를 클릭합니다.*

   vCPU 세부 정보 대화 상자가 나타납니다.



1. *선택한 VM의 가상 CPU를 구성합니다.*

   - *vCPU 개수 - 현재 사용 중인 vCPU 수입니다.*

     참고

     vCPU 수는 vCPU 최대값보다 클 수 없습니다.

   - *vCPU Maximum - VM에 대해 구성할 수 있는 최대 가상 CPU 수입니다. 이 값이 vCPU 개수 보다 크면 VM에 추가 vCPU를 연결할 수 있습니다.*

- *sockets - VM에 노출할 소켓 수입니다.*

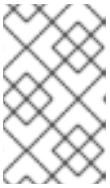
- *소켓당 코어 수 - VM에 표시할 각 소켓의 코어 수입니다.*

- *코어당 스레드 - VM에 노출할 각 코어의 스레드 수입니다.*

소켓당 소켓,소켓당 코어 및 코어 옵션당 스레드 는 VM의 CPU 토폴로지를 조정합니다. 이는 vCPU 성능에 유용할 수 있으며 게스트 OS에서 특정 소프트웨어의 기능에 영향을 미칠 수 있습니다. 배포에서 다른 설정이 필요하지 않은 경우 기본값을 유지합니다.

2. *Apply(적용)를 클릭합니다.*

VM의 가상 CPU가 구성됩니다.



참고

가상 CPU 설정에 대한 변경 사항은 VM을 재시작한 후에만 적용됩니다.

추가 리소스

- *명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거*

17.6.3. 가상 머신에서 NUMA 구성

다음 방법을 사용하여 RHEL 9 호스트에서 VM(가상 머신) 설정을 NUMA(Non-Uniform Memory Access) 설정을 구성할 수 있습니다.

사전 요구 사항

- *호스트는 NUMA 호환 시스템입니다. 이 경우 virsh nodeinfo 명령을 사용하여 NUMA 셀 을 확인합니다.*

```
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              48
CPU frequency:       1200 MHz
CPU socket(s):       1
```

```
Core(s) per socket:  12
Thread(s) per core:  2
NUMA cell(s):        2
Memory size:         67012964 KiB
```

행 값이 2 이상이면 호스트는 NUMA와 호환됩니다.

절차

쉽게 사용할 수 있도록 자동 유틸리티 및 서비스를 사용하여 VM의 NUMA 구성을 설정할 수 있습니다. 그러나 수동 NUMA 설정은 상당한 성능을 향상시킬 가능성이 높습니다.

자동 방법

- VM의 NUMA 정책을 Preferred 로 설정합니다. 예를 들어 testguest5 VM에 대해 다음을 수행하려면 다음을 수행합니다.

```
# virt-xml testguest5 --edit --vcpus placement=auto
# virt-xml testguest5 --edit --numatune mode=preferred
```

- 호스트에서 자동 NUMA 분산을 활성화합니다.

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

- numad 명령을 사용하여 VM CPU를 메모리 리소스와 자동으로 정렬합니다.

```
# numad
```

수동 방법

1. 특정 호스트 CPU 또는 CPU 범위에 특정 vCPU 스레드를 고정합니다. 이는 NUMA가 아닌 호스트 및 VM에서도 발생할 수 있으며 vCPU 성능 향상을 위한 안전한 방법으로 권장됩니다.

   예를 들어 다음 명령은 vCPU 스레드 0~5개의 testguest6 VM을 각각 고정하여 CPU 1, 3, 5, 7, 9 및 11을 각각 호스팅합니다.

```
# virsh vcpupin testguest6 0 1
# virsh vcpupin testguest6 1 3
```

```
# virsh vcpupin testguest6 2 5
# virsh vcpupin testguest6 3 7
# virsh vcpupin testguest6 4 9
# virsh vcpupin testguest6 5 11
```

그런 다음 이것이 성공했는지 확인할 수 있습니다.

```
# virsh vcpupin testguest6
VCPU   CPU Affinity
----------------------
0      1
1      3
2      5
3      7
4      9
5      11
```

2. vCPU 스레드를 고정한 후 지정된 VM과 연결된 QEMU 프로세스 스레드를 특정 호스트 CPU 또는 CPU 범위에 고정할 수도 있습니다. 예를 들어 다음 명령은 testguest6 의 QEMU 프로세스 스레드를 CPU 13 및 15에 고정하고 이것이 성공했는지 확인합니다.

```
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
----------------------------------
       *: 13,15
```

3. 마지막으로 특정 VM에 할당할 호스트 NUMA 노드를 지정할 수도 있습니다. 이렇게 하면 VM 의 vCPU에서 호스트 메모리 사용량이 향상될 수 있습니다. 예를 들어 다음 명령은 호스트 NUMA 노드 3을 5로 사용하도록 testguest6 을 설정하고 이것이 성공했는지 확인합니다.

```
# virsh numatune testguest6 --nodeset 3-5
# virsh numatune testguest6
```

참고

최상의 성능 결과를 얻으려면 위에 나열된 모든 수동 튜닝 방법을 사용하는 것이 좋습니다.

확인된 문제

- 현재 IBM Z 호스트에서는 NUMA 튜닝을 수행할 수 없습니다.

추가 리소스

- vCPU 성능 튜닝 시나리오 샘플
- numastat 유틸리티를 사용하여 시스템의 현재 NUMA 구성 보기

17.6.4. vCPU 성능 튜닝 시나리오 샘플

가능한 최상의 vCPU 성능을 얻으려면 다음 시나리오에서와 같이 수동 vcpupin,emulatorpin 및 numatune 설정을 함께 사용하는 것이 좋습니다.

시나리오 시작

- 호스트에는 다음과 같은 하드웨어 관련 사항이 있습니다.
  - NUMA 노드 2개
  - 각 노드의 CPU 코어 3개
  - 각 코어의 2개 스레드

  이러한 머신의 virsh nodeinfo 출력은 다음과 유사합니다.

```
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              12
CPU frequency:       3661 MHz
CPU socket(s):       2
Core(s) per socket:  3
Thread(s) per core:  2
NUMA cell(s):        2
Memory size:         31248692 KiB
```

- 기존 VM을 8개의 vCPU가 있도록 수정하려고 합니다. 즉, 단일 NUMA 노드에 맞지 않습니다.

  따라서 각 NUMA 노드에 4개의 vCPU를 배포하고 vCPU 토폴로지가 호스트 토폴로지와 최

*대한 유사하도록 해야 합니다. 즉, 지정된 물리적 **CPU**의 형제 스레드로 실행되는 **vCPU**는 동일한 코어에서 스레드를 호스트하도록 고정해야 합니다. 자세한 내용은 아래 솔루션을 참조하십시오.*

해결책

1. *호스트 토폴로지에 대한 정보를 가져옵니다.*

```
# virsh capabilities
```

출력에는 다음과 유사한 섹션이 포함되어야 합니다.

```
<topology>
  <cells num="2">
    <cell id="0">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="10" />
        <sibling id="1" value="21" />
      </distances>
      <cpus num="6">
        <cpu id="0" socket_id="0" core_id="0" siblings="0,3" />
        <cpu id="1" socket_id="0" core_id="1" siblings="1,4" />
        <cpu id="2" socket_id="0" core_id="2" siblings="2,5" />
        <cpu id="3" socket_id="0" core_id="0" siblings="0,3" />
        <cpu id="4" socket_id="0" core_id="1" siblings="1,4" />
        <cpu id="5" socket_id="0" core_id="2" siblings="2,5" />
      </cpus>
    </cell>
    <cell id="1">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="21" />
        <sibling id="1" value="10" />
      </distances>
      <cpus num="6">
        <cpu id="6" socket_id="1" core_id="3" siblings="6,9" />
        <cpu id="7" socket_id="1" core_id="4" siblings="7,10" />
        <cpu id="8" socket_id="1" core_id="5" siblings="8,11" />
        <cpu id="9" socket_id="1" core_id="3" siblings="6,9" />
        <cpu id="10" socket_id="1" core_id="4" siblings="7,10" />
        <cpu id="11" socket_id="1" core_id="5" siblings="8,11" />
      </cpus>
```

```
    </cell>
  </cells>
</topology>
```

2. 선택 사항: 적용 가능한 툴 및 유틸리티를 사용하여 VM의 성능을 테스트합니다.

3. 호스트에 1GiB 대규모 페이지를 설정 및 마운트합니다.

   a. 호스트의 커널 명령줄에 다음 행을 추가합니다.

   ```
   default_hugepagesz=1G hugepagesz=1G
   ```

   b. 다음 콘텐츠를 사용하여 /etc/systemd/system/hugetlb-gigantic-pages.service 파일을 만듭니다.

   ```
   [Unit]
   Description=HugeTLB Gigantic Pages Reservation
   DefaultDependencies=no
   Before=dev-hugepages.mount
   ConditionPathExists=/sys/devices/system/node
   ConditionKernelCommandLine=hugepagesz=1G

   [Service]
   Type=oneshot
   RemainAfterExit=yes
   ExecStart=/etc/systemd/hugetlb-reserve-pages.sh

   [Install]
   WantedBy=sysinit.target
   ```

   c. 다음 콘텐츠를 사용하여 /etc/systemd/hugetlb-reserve-pages.sh 파일을 만듭니다.

   ```
   #!/bin/sh

   nodes_path=/sys/devices/system/node/
   if [ ! -d $nodes_path ]; then
    echo "ERROR: $nodes_path does not exist"
    exit 1
   fi

   reserve_pages()
   {
    echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
   }
   ```

```
reserve_pages 4 node1
reserve_pages 4 node2
```

*이는 **node1** 에서 **4GiB**의 대규모 페이지를 예약하고 **node2** 의 **4**개의 대규모 페이지를 예약합니다.*

d. *이전 단계에서 생성한 스크립트를 실행 가능하게 만듭니다.*

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

e. *부팅 시 대규모 페이지 예약을 활성화합니다.*

```
# systemctl enable hugetlb-gigantic-pages
```

4. *이 예에서는 **super-VM** 을 사용하여 최적화하려는 **VM**의 **XML** 구성을 편집하려면 **virsh edit** 명령을 사용합니다.*

```
# virsh edit super-vm
```

5. *다음과 같은 방식으로 **VM**의 **XML** 구성을 조정합니다.*

a. ***8**개의 정적 **vCPU**를 사용하도록 **VM**을 설정합니다. 이 작업을 수행하려면 **<vcpu/>** 요소를 사용합니다.*

b. *각 **vCPU** 스레드를 토폴로지에서 미러링하는 해당 호스트 **CPU** 스레드에 고정합니다. 이렇게 하려면 **<cputune>** 섹션의 **<vcpupin/>** 요소를 사용합니다.*

*위의 **virsh capabilities** 유틸리티에서와 같이 호스트 **CPU** 스레드는 해당 코어에서 순차적으로 정렬되지 않습니다. 또한 **vCPU** 스레드를 동일한 **NUMA** 노드에서 사용 가능한 최고 호스트 코어 세트에 고정해야 합니다. 테이블 그림에서는 아래의 샘플 토폴로지 섹션을 참조하십시오.*

***a.** 및 **b.** 단계에 대한 **XML** 구성은 다음과 유사합니다.*

```
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='4'/>
```

```
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='7'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='8'/>
  <vcpupin vcpu='7' cpuset='11'/>
  <emulatorpin cpuset='6,9'/>
</cputune>
```

c. *1GiB 대규모 페이지를 사용하도록 VM을 설정합니다.*

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'/>
  </hugepages>
</memoryBacking>
```

d. *호스트의 해당 NUMA 노드의 메모리를 사용하도록 VM의 NUMA 노드를 구성합니다. 이렇게 하려면 <numatune/> 섹션의 <memnode/> 요소를 사용하십시오.*

```
<numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

e. *CPU 모드가 host-passthrough 로 설정되어 있고 CPU에서 passthrough 모드에서 캐시를 사용하는지 확인합니다.*

```
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
```

검증

1. *결과적으로 VM의 XML 구성에 다음과 유사한 섹션이 포함되어 있는지 확인합니다.*

```
[...]
  <memoryBacking>
    <hugepages>
      <page size='1' unit='GiB'/>
    </hugepages>
  </memoryBacking>
  <vcpu placement='static'>8</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='1'/>
```

```
  <vcpupin vcpu='1' cpuset='4'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='7'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='8'/>
  <vcpupin vcpu='7' cpuset='11'/>
  <emulatorpin cpuset='6,9'/>
 </cputune>
 <numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
 </numatune>
 <cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
  <numa>
   <cell id="0" cpus="0-3" memory="2" unit="GiB">
    <distances>
     <sibling id="0" value="10"/>
     <sibling id="1" value="21"/>
    </distances>
   </cell>
   <cell id="1" cpus="4-7" memory="2" unit="GiB">
    <distances>
     <sibling id="0" value="21"/>
     <sibling id="1" value="10"/>
    </distances>
   </cell>
  </numa>
 </cpu>
</domain>
```

2. *선택 사항: 적용 가능한 툴 및 유틸리티를 사용하여 VM의 성능을 테스트하여 VM 최적화의 영향을 평가합니다.*

샘플 토폴로지

- *다음 표에서는 고정해야 하는 vCPU와 호스트 CPU 간의 연결을 보여줍니다.*

표 17.1. 호스트 토폴로지

CPU 스레드	0	3	1	4	2	5	6	9	7	10	8	11
코어	0		1		2		3		4		5	
소켓	0						1					
NUMA 노드	0						1					

표 17.2. VM 토폴로지

vCPU 스레드	0	1	2	3	4	5	6	7
코어	0		1		2		3	
소켓	0				1			
NUMA 노드	0				1			

표 17.3. 결합된 호스트 및 VM 토폴로지

vCPU 스레드			0	1	2	3			4	5	6	7
호스트 CPU 스레드	0	3	1	4	2	5	6	9	7	10	8	11
코어	0		1		2		3		4		5	
소켓	0						1					
NUMA 노드	0						1					

이 시나리오에서는 NUMA 노드 2개와 8개의 vCPU가 있습니다. 따라서 각 노드에 4개의 vCPU 스레드를 고정해야 합니다.

또한 호스트 시스템 작업을 위해 각 노드에서 하나 이상의 단일 CPU 스레드를 사용할 수 있는 것이 좋습니다.

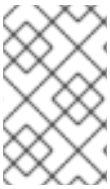
이 예에서 각 NUMA 노드에는 각각 2개의 호스트 CPU 스레드가 있는 3개의 코어가 포함되어 있으므로 노드 0에 대한 세트는 다음과 같이 변환됩니다.

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

17.6.5. 커널 동일한 페이지 병합 관리

KSM(커널 동일 페이지 병합)은 가상 머신(VM) 간에 동일한 메모리 페이지를 공유하여 메모리 밀도를 향상시킵니다. 그러나 KSM을 활성화하면 CPU 사용률이 증가하며 워크로드에 따라 전체 성능에 부정적인 영향을 미칠 수 있습니다.

요구 사항에 따라 단일 세션 또는 영구적으로 **KSM**을 활성화하거나 비활성화할 수 있습니다.



참고

RHEL 9 이상에서 **KSM**은 기본적으로 비활성화되어 있습니다.

사전 요구 사항

- 호스트 시스템에 대한 루트 액세스.

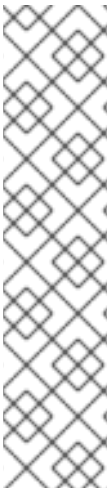
절차

- **KSM**을 비활성화합니다.
  - 단일 세션의 **KSM**을 비활성화하려면 **systemctl** 유틸리티를 사용하여 **ksm** 및 **ksmtuned** 서비스를 중지합니다.

```
# systemctl stop ksm

# systemctl stop ksmtuned
```

  - **KSM**을 영구적으로 비활성화하려면 **systemctl** 유틸리티를 사용하여 **ksm** 및 **ksmtuned** 서비스를 비활성화합니다.

```
# systemctl disable ksm
Removed /etc/systemd/system/multi-user.target.wants/ksm.service.
# systemctl disable ksmtuned
Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
```



참고

KSM을 비활성화하기 전에 **VM** 간에 공유되는 메모리 페이지가 공유됩니다. 공유를 중지하려면 다음 명령을 사용하여 시스템의 **PageKSM** 페이지를 모두 삭제합니다.

```
# echo 2 > /sys/kernel/mm/ksm/run
```

익명 페이지가 **KSM** 페이지를 교체하면 **khugepaged** 커널 서비스에서 **VM**의 물리적 메모리에서 투명한 **hugepages**를 다시 빌드합니다.

- KSM을 활성화합니다.

주의

KSM을 활성화하면 CPU 사용률이 증가하고 전체 CPU 성능에 영향을 미칩니다.

1. ksmtuned 서비스를 설치합니다.

```
# yum install ksmtuned
```

2. 서비스를 시작합니다.

- 단일 세션으로 KSM을 활성화하려면 systemctl 유틸리티를 사용하여 ksm 및 ksmtuned 서비스를 시작합니다.

```
# systemctl start ksm
# systemctl start ksmtuned
```

- KSM을 영구적으로 활성화하려면 systemctl 유틸리티를 사용하여 ksm 및 ksmtuned 서비스를 활성화합니다.

```
# systemctl enable ksm
Created symlink /etc/systemd/system/multi-user.target.wants/ksm.service →
/usr/lib/systemd/system/ksm.service

# systemctl enable ksmtuned
Created symlink /etc/systemd/system/multi-user.target.wants/ksmtuned.service →
/usr/lib/systemd/system/ksmtuned.service
```

17.7. 가상 머신 네트워크 성능 최적화

VM의 NIC(네트워크 인터페이스 카드)의 가상 특성으로 인해 VM이 할당된 호스트 네트워크 대역폭의 일부가 손실되어 VM의 전체 워크로드 효율성을 줄일 수 있습니다. 다음 팁은 가상 NIC(vNIC) 처리량에 대한 가상화의 부정적인 영향을 최소화할 수 있습니다.

절차

다음 방법 중 하나를 사용하고 VM 네트워크 성능에 유용한 영향을 미치는지 확인합니다.

vhost_net 모듈 활성화

호스트에서 vhost_net 커널 기능이 활성화되어 있는지 확인합니다.

```
# lsmod | grep vhost
vhost_net           32768  1
vhost               53248  1 vhost_net
tap                 24576  1 vhost_net
tun                 57344  6 vhost_net
```

이 명령의 출력이 비어 있으면 vhost_net 커널 모듈을 활성화합니다.

```
# modprobe vhost_net
```

멀티 큐 virtio-net 설정

VM에 다중 대기열 virtio-net 기능을 설정하려면 virsh edit 명령을 사용하여 VM의 XML 구성에 대해 편집합니다. XML에서 <devices> 섹션에 다음을 추가하고 N 을 VM의 vCPU 수로 바꿉니다(최대 16개).

```
<interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
    <driver name='vhost' queues='N'/>
</interface>
```

VM이 실행 중인 경우 변경 사항을 적용하려면 VM을 다시 시작합니다.

네트워크 패킷 배치

긴 전송 경로가 있는 Linux VM 구성에서 패킷을 배치하기 전에 패킷을 배치하면 캐시 사용률이 향상될 수 있습니다. 패킷 배치를 설정하려면 호스트에서 다음 명령을 사용하고 tap0 을 VM에서 사용하는 네트워크 인터페이스 이름으로 교체합니다.

```
# ethtool -C tap0 rx-frames 64
```

SR-IOV

호스트 NIC에서 SR-IOV를 지원하는 경우 vNICs에 SR-IOV 장치 할당을 사용합니다. 자세한 내용

은 SR-IOV 장치 관리를 참조하십시오.

추가 리소스

- 가상 네트워킹 이해

17.8. 가상 머신 성능 모니터링 툴

VM 리소스를 가장 많이 사용하는 것과 최적화가 필요한 VM 성능의 측면을 식별하려면 일반 및 VM 고유의 성능 진단 툴을 사용할 수 있습니다.

기본 OS 성능 모니터링 툴

표준 성능 평가의 경우 호스트 및 게스트 운영 체제에서 기본적으로 제공하는 유틸리티를 사용할 수 있습니다.

- RHEL 9 호스트에서 root로 top 유틸리티 또는 시스템 모니터 애플리케이션을 사용하고 출력에서 qemu 및 virt 을 찾습니다. 이는 VM이 사용하는 호스트 시스템 리소스의 양을 보여줍니다.
  - 모니터링 툴이 호스트 CPU 또는 메모리 용량의 많은 부분을 사용하는 qemu 또는 virt 프로세스가 표시되는 경우 perf 유틸리티를 사용하여 조사합니다. 자세한 내용은 아래를 참조하십시오.
  - 또한 vhost_net-1234 와 같은 vhost_net 스레드 프로세스가 과도한 호스트 CPU 용량을 소비하는 것으로 표시되는 경우 가상 네트워크 최적화 기능 (예: multi-queue virtio-net) 을 사용하는 것이 좋습니다.
- 게스트 운영 체제에서는 시스템에서 사용 가능한 성능 유틸리티 및 애플리케이션을 사용하여 가장 많은 시스템 리소스를 사용하는 프로세스를 평가합니다.
  - Linux 시스템에서는 top 유틸리티를 사용할 수 있습니다.
  - Windows 시스템에서는 작업 관리자 애플리케이션을 사용할 수 있습니다.

perf kvm

perf 유틸리티를 사용하여 RHEL 9 호스트 성능에 대한 가상화 관련 통계를 수집하고 분석할 수 있습

니다. 이를 위해 다음을 수행합니다.

1. 호스트에서 **perf** 패키지를 설치합니다.

```
# dnf install perf
```

2. **perf kvm stat** 명령 중 하나를 사용하여 가상화 호스트에 대한 **perf** 통계를 표시합니다.

- 하이퍼바이저를 실시간으로 모니터링하려면 **perf kvm stat live** 명령을 사용합니다.
- 일정 기간 동안 하이퍼바이저의 **perf** 데이터를 기록하려면 **perf kvm stat record** 명령을 사용하여 로깅을 활성화합니다. 명령이 취소되거나 중단되면 데이터가 **perf.data.guest** 파일에 저장되며 **perf kvm stat report** 명령을 사용하여 분석할 수 있습니다.

3. **VM-EXIT** 이벤트 유형 및 배포에 대한 **perf** 출력을 분석합니다. 예를 들어 **PAUSE_INSTRUCTION** 이벤트는 드물지만 다음 출력에서는 이 이벤트의 높은 경우 호스트 **CPU**가 실행 중인 **vCPU**를 잘 처리하지 않음을 나타냅니다. 이러한 시나리오에서는 활성 **VM** 일부 종료, 이러한 **VM**에서 **vCPU** 제거 또는 **vCPU**의 성능을 튜닝 하는 것이 좋습니다.

```
# perf kvm stat report

Analyze events for all VMs, all VCPUs:


             VM-EXIT    Samples  Samples%     Time%    Min Time    Max Time         Avg time

  EXTERNAL_INTERRUPT     365634    31.59%    18.04%      0.42us  58780.59us    204.08us ( +-   0.99% )
            MSR_WRITE     293428    25.35%     0.13%      0.59us  17873.02us      1.80us ( +-   4.63% )
     PREEMPTION_TIMER     276162    23.86%     0.23%      0.51us  21396.03us      3.38us ( +-   5.19% )
      PAUSE_INSTRUCTION     189375    16.36%    11.75%      0.72us  29655.25us    256.77us ( +-   0.70% )
                    HLT      20440     1.77%    69.83%      0.62us  79319.41us  14134.56us ( +-   0.79% )
                 VMCALL      12426     1.07%     0.03%      1.02us   5416.25us      8.77us ( +-   7.36% )
          EXCEPTION_NMI         27     0.00%     0.00%      0.69us      1.34us      0.98us ( +-   3.50% )
          EPT_MISCONFIG          5     0.00%     0.00%      5.15us     10.85us      7.88us ( +-  11.67% )

Total Samples:1157497, Total events handled time:413728274.66us.
```

perf kvm stat 의 출력에서 문제를 나타낼 수 있는 기타 이벤트 유형은 다음과 같습니다.

- *INSN_EMULATION - 차선 VM I/O 구성 을 제안합니다.*

perf 를 사용하여 가상화 성능을 모니터링하는 방법에 대한 자세한 내용은 perf-kvm 도움말 페이지를 참조하십시오.

numastat

시스템의 현재 NUMA 구성을 보려면 numactl 패키지를 설치하여 제공되는 numastat 유틸리티를 사용할 수 있습니다.

다음은 각각 여러 NUMA 노드에서 메모리를 가져오는 4개의 VM이 있는 호스트를 보여줍니다. vCPU 성능에는 적합하지 않으며 조정이 보장됩니다.

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51722 (qemu-kvm)     68     16    357   6936      2      3    147    598  8128
51747 (qemu-kvm)    245     11      5     18   5172   2532      1     92  8076
53736 (qemu-kvm)     62    432   1661    506   4851    136     22    445  8116
53773 (qemu-kvm)   1393      3      1      2     12      0      0   6702  8114
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total              1769    463   2024   7462  10037   2672    169   7837 32434
```

반대로 다음에서는 단일 노드에서 각 VM에 제공되는 메모리를 보여줍니다. 이는 훨씬 더 효율적입니다.

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51747 (qemu-kvm)      0      0      7      0   8072      0      1      0  8080
53736 (qemu-kvm)      0      0      7      0      0      0   8113      0  8120
53773 (qemu-kvm)      0      0      7      0      0      0      1   8110  8118
59065 (qemu-kvm)      0      0   8050      0      0      0      0      0  8051
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total                 0      0   8072      0   8072      0   8114   8110 32368
```

17.9. 추가 리소스

- *Windows 가상 머신 최적화*

18장. 가상 머신 보안

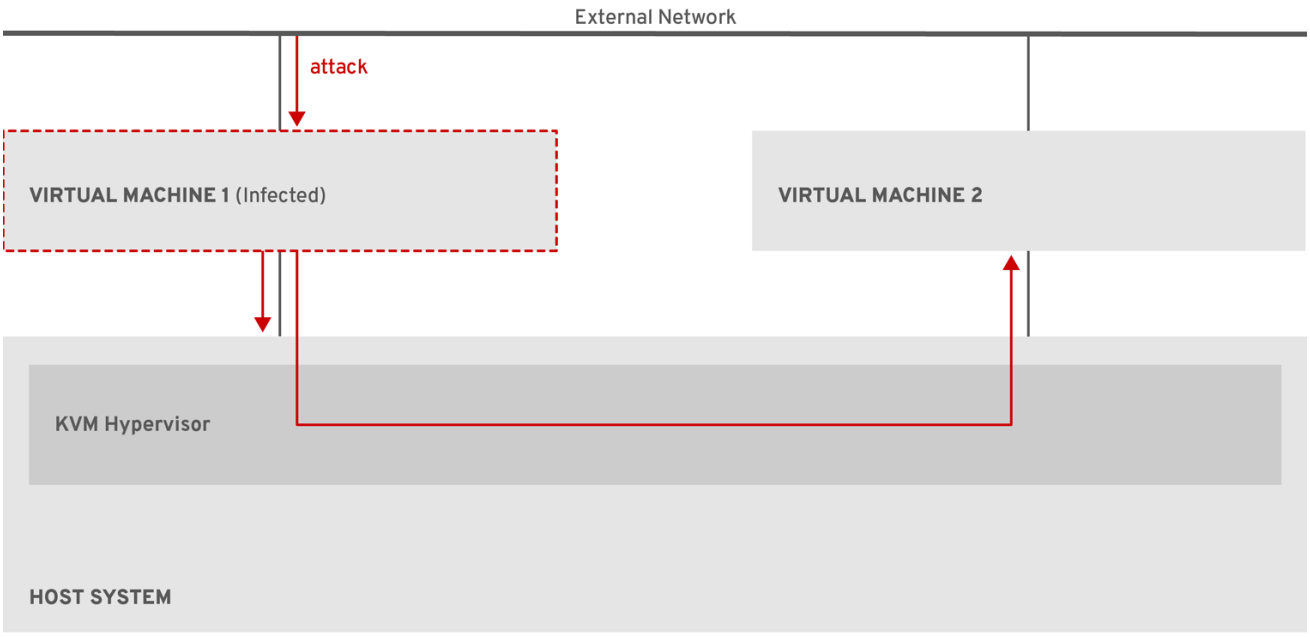
VM(가상 머신)을 사용하는 RHEL 9 시스템 관리자로서 VM을 최대한 안전하게 보호함으로써 게스트 및 호스트 OS가 악성 소프트웨어에 의해 감염된 위험을 크게 줄일 수 있습니다.

이 문서에서는 RHEL 9 호스트에서 VM을 보호하는 메커니즘 을 간략하게 설명하고 VM 의 보안을 향상시키는 방법 목록을 제공합니다.

18.1. 가상 시스템에서 보안이 작동하는 방법

VM(가상 머신)을 사용하는 경우 단일 호스트 시스템 내에 여러 운영 체제를 보유할 수 있습니다. 이러한 시스템은 하이퍼바이저를 통해 호스트와 연결되어 있으며 일반적으로 가상 네트워크를 통해 연결됩니다. 결과적으로 각 VM은 악성 소프트웨어로 호스트를 공격하기 위한 벡터로 사용될 수 있으며 호스트는 VM을 공격하기 위한 벡터로 사용될 수 있습니다.

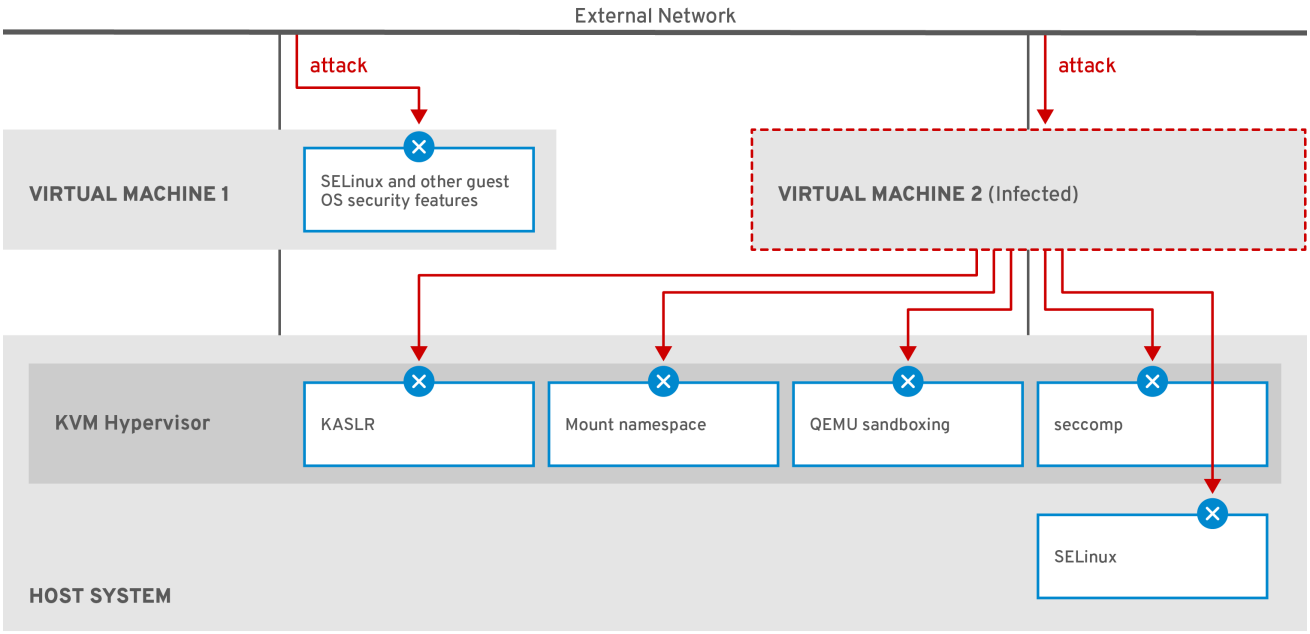
그림 18.1. 가상화 호스트에서 잠재적인 악성 코드 공격 벡터



하이퍼바이저는 호스트 커널을 사용하여 VM을 관리하므로 VM의 운영 체제에서 실행되는 서비스는 호스트 시스템에 악성 코드를 삽입하는 데 자주 사용됩니다. 그러나 호스트 및 게스트 시스템의 여러 보안 기능을 사용하여 이러한 보안 위협으로부터 시스템을 보호할 수 있습니다.

SELinux 또는 QEMU 샌드박스와 같은 이러한 기능은 악성 코드가 하이퍼바이저를 공격하고 호스트와 VM 간에 전송하는 것을 더 어렵게 만드는 다양한 조치를 제공합니다.

그림 18.2. 가상화 호스트에 대한 악성 코드 공격 방지



RHEL 9에서 **VM** 보안을 위해 제공하는 대부분의 기능은 항상 활성화되어 활성화되어 구성할 필요가 없습니다. 자세한 내용은 가상 머신 보안의 자동 기능을 참조하십시오.

또한 다양한 모범 사례를 준수하여 **VM**과 하이퍼바이저의 취약점을 최소화할 수 있습니다. 자세한 내용은 가상 머신 보안을 위한 모범 사례를 참조하십시오.

18.2. 가상 머신 보안을 위한 모범 사례

아래 지침에 따라 가상 머신이 악성 코드로 감염되고 호스트 시스템을 감염시키기 위해 공격 벡터로 사용되는 위험이 크게 감소합니다.

게스트 쪽에서:

- 가상 머신을 실제 머신처럼 보호합니다. 보안을 강화하기 위해 사용할 수 있는 특정 방법은 게스트 **OS**에 따라 다릅니다.

  VM이 **RHEL 9** 를 실행하는 경우 게스트 시스템의 보안 개선에 대한 자세한 내용은 **Red Hat Enterprise Linux 9** 보안을 참조하십시오.

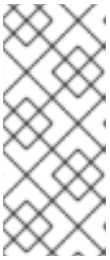
호스트 측에서:

- VM을 원격으로 관리할 때 VM에 연결하기 위한 SSL 과 같은 암호화 유틸리티(예: SSL)를 사용합니다.

- SELinux가 강제 모드에 있는지 확인합니다.

```
# getenforce
Enforcing
```

SELinux가 비활성화되었거나 허용 모드에서 실행되는 경우 강제 모드 활성화에 대한 지침은 SELinux 문서 사용을 참조하십시오.



참고

SELinux Enforcing 모드를 사용하면 TSX RHEL 9 기능을 사용할 수 있습니다. 가상화를 위한 특수 SELinux 부울 집합입니다. VM 보안 관리를 위해 수동으로 조정할 수 있습니다.

- SecureBoot 에서 VM 사용:

SecureBoot는 VM이 암호화 방식으로 서명된 OS를 실행할 수 있도록 하는 기능입니다. 이로 인해 OS가 악성 코드 공격을 통해 변경된 VM이 부팅되지 않습니다.

SecureBoot는 OVMF 펌웨어를 사용하는 Linux VM을 설치할 때만 적용할 수 있습니다. 자세한 내용은 SecureBoot 가상 머신 생성을 참조하십시오.

- qemu-kvm 과 같은 qemu-* 명령을 사용하지 마십시오.

QEMU는 RHEL 9의 가상화 아키텍처의 필수 구성 요소이지만 수동으로 관리하기가 어렵고 부적절한 QEMU 구성으로 인해 보안 취약점이 발생할 수 있습니다. 따라서 qemu-* 명령 사용은 Red Hat에서 지원되지 않습니다. 대신, 모범 사례에 따라 QEMU를 오케스트레이션하므로 virsh,virt-install, virt-xml 과 같은 libvirt 유틸리티를 사용하십시오.

추가 리소스

- RHEL에서 가상화를 위한 SELinux 부울

18.3. SECUREBOOT 가상 머신 생성

VM이 암호화 방식으로 서명된 OS를 실행하고 있는지 확인하는 SecureBoot 기능을 사용하는 Linux VM(가상 머신)을 만들 수 있습니다. 이 기능은 VM의 게스트 OS가 악성 코드에 의해 변경된 경우에 유용할 수 있습니다. 이러한 시나리오에서는 SecureBoot가 VM이 부팅되지 않도록 하여 호스트 시스템에 악성 코드가 발생할 수 없도록 합니다.

사전 요구 사항

- VM에서 Q35 시스템 유형을 사용합니다.
- edk2-OVMF 패키지가 설치됩니다.

```
# dnf install edk2-ovmf
```

- 운영 체제(OS) 설치 소스는 로컬 또는 네트워크에서 사용할 수 있습니다. 다음 형식 중 하나일 수 있습니다.
  - 설치 미디어의 ISO 이미지
  - 기존 VM 설치의 디스크 이미지

> **주의**
>
> RHEL 9에서는 호스트 CD-ROM 또는 DVD-ROM 장치에서 설치할 수 없습니다. RHEL 9에서 사용할 수 있는 VM 설치 방법을 사용할 때 CD-ROM 또는 DVD-ROM을 설치 소스로 선택하면 설치에 실패합니다. 자세한 내용은 Red Hat 지식베이스 를 참조하십시오.

- 선택 사항: Kickstart 파일을 더 빠르고 쉽게 설치할 수 있도록 제공할 수 있습니다.

절차

1.

virt-install 명령을 사용하여 명령줄 인터페이스를 사용하여 가상 머신 생성에 자세히 설명된 VM을 생성합니다. --boot 옵션의 경우 uefi,nvram_template=/usr/share/OVMF/OVMF_VARS.secboot.fd 값을 사용합니다. 이는 OVMF_VARS.secboot.fd 및 OVMF_CODE.secboot.fd 파일을 VM의 NVRAM(Non-volatile RAM) 설정 템플릿으로 사용하여 SecureBoot 기능을 활성화합니다.

예를 들면 다음과 같습니다.

```
# virt-install --name rhel8sb --memory 4096 --vcpus 4 --os-variant rhel9.0 --boot uefi,nvram_template=/usr/share/OVMF/OVMF_VARS.secboot.fd --disk boot_order=2,size=10 --disk boot_order=1,device=cdrom,bus=scsi,path=/images/RHEL-9.0-installation.iso
```

2. 화면의 지침에 따라 OS 설치 절차를 따르십시오.

검증

1. 게스트 OS가 설치되면 그래픽 게스트 콘솔에서 터미널을 열거나 SSH를 사용하여 게스트 OS에 연결하여 VM의 명령줄에 액세스합니다.

2. VM에서 SecureBoot가 활성화되었는지 확인하려면 mokutil --sb-state 명령을 사용합니다.

```
# mokutil --sb-state
SecureBoot enabled
```

추가 리소스

- AMD64, Intel 64 및 64비트 ARM에 RHEL 9 설치

18.4. 가상 머신 사용자가 수행할 수 있는 작업 제한

경우에 따라 RHEL 9에서 호스팅되는 VM(가상 머신) 사용자가 기본적으로 수행할 수 있는 조치로 인해 보안 위험이 발생할 수 있습니다. 이 경우 호스트 시스템에서 polkit 정책 툴킷을 사용하도록 libvirt 데몬을 구성하여 VM 사용자가 사용할 수 있는 작업을 제한할 수 있습니다.

절차

1. 선택 사항: libvirt 와 관련된 시스템의 polkit 제어 정책이 기본 설정에 따라 설정되어 있는지 확인합니다.

a. */usr/share/polkit-1/actions/* 및 */usr/share/polkit-1/rules.d/* 디렉토리에서 모든 **libvirt** 관련 파일을 찾습니다.

```
# ls /usr/share/polkit-1/actions | grep libvirt
# ls /usr/share/polkit-1/rules.d | grep libvirt
```

b. 파일을 열고 규칙 설정을 검토합니다.

polkit 제어 정책의 구문을 읽는 방법에 대한 자세한 내용은 **man polkit** 을 사용합니다.

c. **libvirt** 제어 정책을 수정합니다. 이를 위해 다음을 수행합니다.

i. */etc/polkit-1/rules.d/* 디렉터리에 새 **.conf** 파일을 만듭니다.

ii. 이 파일에 사용자 지정 정책을 추가하고 저장합니다.

libvirt 제어 정책의 자세한 내용 및 예는 **libvirt** 업스트림 문서를 참조하십시오.

2. **polkit** 에 의해 결정된 액세스 정책을 사용하도록 **VM**을 구성합니다.

a. */etc/libvirt/* 디렉터리에서 모든 가상화 드라이버 구성 파일을 찾습니다.

```
# ls /etc/libvirt/ | grep virt*d.conf
```

b. 각 파일에서 **access_drivers = ["polkit"]** 행의 주석을 제거하고 파일을 저장합니다.

3. 이전 단계에서 수정한 각 파일에 대해 해당 서비스를 다시 시작합니다.

예를 들어 */etc/libvirt/virtqemud.conf* 를 수정한 경우 **virtqemud** 서비스를 다시 시작합니다.

```
# systemctl try-restart virtqemud
```

검증

- 제한하려는 VM 작업이 있는 사용자로 제한된 작업 중 하나를 수행합니다.

예를 들어 권한이 없는 사용자가 시스템 세션에서 생성된 VM을 볼 수 없는 경우 다음을 실행합니다.

```
$ virsh -c qemu:///system list --all
Id   Name         State
------------------------------
```

시스템에 하나 이상의 VM이 있어도 이 명령에 VM이 나열되지 않은 경우 polkit 은 권한이 없는 사용자에 대한 작업을 성공적으로 제한합니다.

문제 해결

- 현재 polkit 을 사용하도록 libvirt 를 구성하면 libvirt-dbus 서비스와 의 호환성으로 인해 RHEL 9 웹 콘솔을 사용하여 VM에 연결할 수 없습니다.

웹 콘솔에서 VM에 대한 액세스를 세부적으로 제어해야 하는 경우 사용자 지정 D-Bus 정책을 생성하는 것이 좋습니다. 자세한 내용은 Red Hat 지식베이스 의 Cockpit에서 가상 시스템 제어를 구성하는 방법을 참조하십시오.

추가 리소스

- man polkit 명령
- polkit 액세스 제어 정책의 libvirt 매뉴얼 페이지

18.5. 가상 머신 보안을 위한 자동 기능

가상 시스템 보안 유지에 나열된 가상 시스템의 보안을 수동으로 개선하는 방법 외에도 libvirt 소프트웨어 제품군에서 여러 보안 기능을 제공하며 RHEL 9에서 가상화를 사용할 때 자동으로 활성화됩니다. 여기에는 다음이 포함됩니다.

시스템 및 세션 연결

RHEL 9에서 가상 머신 관리에 사용 가능한 모든 유틸리티에 액세스하려면 libvirt (qemu:///system)의 시스템 연결을사용해야 합니다. 이렇게 하려면 시스템에 대한 root 권한이 있거나 libvirt 사용자 그룹의 일부로 있어야 합니다.

libvirt 그룹에 없는 루트가 아닌 사용자는 리소스에 액세스할 때 로컬 사용자의 액세스 권한을 존중해야 하는 libvirt(qemu:///session)의 세션 연결에 만 액세스할 수 있습니다. 예를 들어 세션 연결을 사용하면 시스템 연결 또는 다른 사용자가 만든 VM을 검색하거나 액세스할 수 없습니다. 또한 사용 가능한 VM 네트워킹 구성 옵션은 상당히 제한됩니다.

참고

RHEL 9 문서에서는 시스템 연결 권한이 있다고 가정합니다.

가상 머신 분리

개별 VM은 호스트에서 격리된 프로세스로 실행되며 호스트 커널에서 적용되는 보안을 사용합니다. 따라서 VM은 동일한 호스트에 있는 다른 VM의 메모리 또는 스토리지를 읽거나 액세스할 수 없습니다.

QEMU 샌드박스

QEMU 코드가 호스트 보안을 손상시킬 수 있는 시스템 호출을 실행하지 못하도록 하는 기능입니다.

Kernel Address Space Randomization (KASLR)

커널 이미지의 압축을 풀기 위한 물리적 주소 및 가상 주소를 임의로 설정할 수 있습니다. 따라서 KASLR은 커널 개체의 위치를 기반으로 게스트 보안 취약점을 악용하는 것을 방지합니다.

18.6. 가상화를 위한 SELINUX 부울

RHEL 9 시스템에서 가상 시스템 보안을 세부적으로 구성하려면 호스트에서 SELinux 부울을 구성하여 하이퍼바이저가 특정 방식으로 작동하도록 할 수 있습니다.

모든 가상화 관련 부울과 해당 상태를 나열하려면 getsebool -a | grep virt 명령을 사용합니다.

```
$ getsebool -a | grep virt
[...]
virt_sandbox_use_netlink --> off
virt_sandbox_use_sys_admin --> off
virt_transition_userdomain --> off
virt_use_comm --> off
virt_use_execmem --> off
virt_use_fusefs --> off
[...]
```

특정 부울을 활성화하려면 root 에서 setsebool -P boolean_name 을 사용합니다. 부울을 비활성화하

*려면 **setsebool -P boolean_name off** 를 사용합니다.*

*다음 표에는 **RHEL 9**에서 사용할 수 있는 가상화 관련 부울과 활성화된 경우 수행하는 사항이 나열되어 있습니다.*

*표 **18.1. SELinux** 가상화 부울*

SELinux Boolean	설명
staff_use_svirt	루트가 아닌 사용자가 VMs를 VMDK로 생성 및 전환할 수 있습니다.
unprivuser_use_svirt	권한이 없는 사용자가 VMs를 추가로 생성 및 transition 할 수 있습니다.
virt_sandbox_use_audit	샌드박스 컨테이너를 사용하여 감사 메시지를 보낼 수 있습니다.
virt_sandbox_use_netlink	샌드박스 컨테이너가 netlink 시스템 호출을 사용할 수 있도록 합니다.
virt_sandbox_use_sys_admin	샌드박스 컨테이너가 mount와 같은 sys_admin 시스템 호출을 사용할 수 있도록 합니다.
virt_transition_userdomain	가상 프로세스를 사용자 도메인으로 실행할 수 있습니다.
virt_use_comm	virt에서 직렬/parallel 통신 포트를 사용할 수 있도록 합니다.
virt_use_execmem	제한된 가상 게스트를 사용하여 실행 가능한 메모리 및 실행 가능 스택을 사용할 수 있습니다.
virt_use_fusefs	virt을 사용하여 FUSE 마운트된 파일을 읽을 수 있습니다.
virt_use_nfs	virt에서 NFS 마운트 파일을 관리할 수 있습니다.
virt_use_rawip	virt이 rawip 소켓과 상호 작용할 수 있습니다.
virt_use_samba	virt을 사용하여 CIFS 마운트 파일을 관리할 수 있습니다.
virt_use_sanlock	제한된 가상 게스트가 sanlock과 상호 작용할 수 있습니다.
virt_use_usb	virt이 USB 장치를 사용할 수 있도록 합니다.

SELinux Boolean	설명
virt_use_xserver	가상 머신이 X 창 시스템과 상호 작용할 수 있습니다.

18.7. IBM Z에서 IBM SECURE EXECUTION 설정

IBM Z 하드웨어를 사용하여 RHEL 9 호스트를 실행하는 경우 VM에 대해 IBM Secure Execution를 구성하여 VM(가상 머신)의 보안을 개선할 수 있습니다.

보호 가상화라고도 하는 IBM Secure Execution를 통해 호스트 시스템이 VM의 상태 및 메모리 콘텐츠에 액세스할 수 없습니다. 결과적으로 호스트가 손상되더라도 게스트 운영 체제를 공격하기 위한 벡터로 사용할 수 없습니다. 또한 신뢰할 수 없는 호스트가 VM에서 중요한 정보를 얻지 못하는 데 Secure Execution를 사용할 수 있습니다.

다음 절차에서는 IBM Z 호스트의 기존 VM을 보안 VM으로 변환하는 방법을 설명합니다.

사전 요구 사항

- *시스템 하드웨어는 다음 중 하나입니다.*
  - *IBM z15 이상*
  - *IBM LinuxONE III 이상*
- *시스템에 대해 보안 실행 기능이 활성화되어 있습니다. 확인하려면 다음을 사용합니다.*

```
# grep facilities /proc/cpuinfo | grep 158
```

이 명령을 실행하면 CPU가 Secure Execution와 호환됩니다.

- *커널에는 보안 실행 지원이 포함되어 있습니다. 확인하려면 다음을 사용합니다.*

```
# ls /sys/firmware | grep uv
```

명령에서 출력을 생성하면 커널이 **Secure Execution**를 지원합니다.

- 호스트 **CPU** 모델에는 압축 해제 기능이 포함되어 있습니다. 확인하려면 다음을 사용합니다.

```
# virsh domcapabilities | grep unpack
<feature policy='require' name='unpack'/>
```

명령에서 위의 출력을 생성하면 **CPU** 호스트 모델이 **Secure Execution**와 호환됩니다.

- **VM**의 **CPU** 모드는 **host-model** 로 설정됩니다. 이를 확인하려면 다음을 사용하고 **vm-name** 을 **VM** 이름으로 교체합니다.

```
# virsh dumpxml vm-name | grep "<cpu mode='host-model'/>"
```

명령에서 출력을 생성하면 **VM**의 **CPU** 모드가 올바르게 설정됩니다.

- **genprotimg** 패키지가 호스트에 설치되어 있어야 합니다.

```
# dnf install genprotimg
```

- **IBM Z** 호스트 키 문서를 취득하고 확인했습니다. 이 작업을 수행하는 방법은 **IBM** 문서의 호스트 키 문서 확인을 참조하십시오.

절차

호스트에서 다음 단계를 수행합니다.

1. **prot_virt=1** 커널 매개 변수를 호스트의 부팅 구성에 추가합니다.

```
# grubby --update-kernel=ALL --args="prot_virt=1"
```

2. 보호할 **VM**에서 공유 버퍼를 사용하도록 **virtio** 장치를 활성화합니다. 이렇게 하려면 **virsh edit** 를 사용하여 **VM**의 **XML** 구성을 수정하고 하나의 장치가 있는 모든 장치의 **< driver >** 행에 **iommu='on'** 을 추가합니다. 예를 들면 다음과 같습니다.

```
<interface type='network'>
  <source network='default'/>
  <model type='virtio'/>
  <driver name='vhost' iommu='on'/>
</interface>
```

*장치 구성에 **< driver >** 줄이 없는 경우 대신 **< driver iommu='on'\& gt;**를 추가합니다.*

3. *기능이 **Secure Execution**와 호환되지 않으므로 **VM**에서 메모리 증대를 비활성화합니다. 이렇게 하려면 **VM**의 **XML** 구성에 다음 행을 추가합니다.*

```
<memballoon model='none'/>
```

*보호하려는 **VM** 의 게스트 운영 체제에서 다음 단계를 수행합니다.*

1. *매개 변수 파일을 생성합니다. 예를 들면 다음과 같습니다.*

```
# touch ~/secure-parameters
```

2. */boot/loader/entries* *디렉토리에서 최신 버전으로 부트 로더 항목을 식별합니다.*

```
# ls /boot/loader/entries -l
[...]
-rw-r--r--. 1 root root  281 Oct  9 15:51 3ab27a195c2849429927b00679db15c1-4.18.0-
240.el8.s390x.conf
```

3. *부트 로더 항목의 커널 옵션 행을 검색합니다.*

```
# cat /boot/loader/entries/3ab27a195c2849429927b00679db15c1-4.18.0-
240.el8.s390x.conf | grep options
options root=/dev/mapper/rhel-root
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
```

4. *옵션 줄의 내용을 추가하고 생성된 매개 변수 파일에 **swiotlb=262144** 를 추가합니다.*

```
# echo "root=/dev/mapper/rhel-root rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
swiotlb=262144" > ~/secure-parameters
```

5.

IBM 보안 실행 이미지를 생성합니다.

예를 들어, 다음에서는 /boot/secure-image 보안 이미지를 /boot/vmlinuz-4.18.0-240.el8.s390x 이미지, /boot/initramfs - 4.18.0-240.el8.s390x.img 초기 RAM 디스크 파일, HKD-86-00020C2.4.4.41C.crt 1을 기반으로 합니다.

```
# genprotimg -i /boot/vmlinuz-4.18.0-240.el8.s390x -r /boot/initramfs-4.18.0-240.el8.s390x.img -p ~/secure-parameters -k HKD-8651-00020089A8.crt -o /boot/secure-image
```

genprotimg 유틸리티를 사용하면 커널 매개 변수, 초기 RAM 디스크 및 부팅 이미지가 포함된 보안 이미지가 생성됩니다.

6.

VM의 부팅 메뉴를 업데이트하여 보안 이미지에서 부팅합니다. 또한 필요하지 않으므로 initrd 및 옵션으로 시작하는 행을 제거합니다.

예를 들어 RHEL 8.3 VM에서 /boot/loader/entries/ 디렉토리에서 부팅 메뉴를 편집할 수 있습니다.

```
# cat /boot/loader/entries/3ab27a195c2849429927b00679db15c1-4.18.0-240.el8.s390x.conf
title Red Hat Enterprise Linux 8.3
version 4.18.0-240.el8.s390x
linux /boot/secure-image
[...]
```

7.

부팅 가능한 디스크 이미지 생성

```
# zipl -V
```

8.

보호되지 않은 원본 파일을 안전하게 제거합니다. 예를 들면 다음과 같습니다.

```
# shred /boot/vmlinuz-4.18.0-240.el8.s390x
# shred /boot/initramfs-4.18.0-240.el8.s390x.img
# shred secure-parameters
```

원래 부팅 이미지, 초기 RAM 이미지 및 커널 매개 변수 파일은 보호되지 않으며 제거되지 않는 경우에도 Secure Execution가 활성화된 VM은 시도 또는 중요한 데이터 마이닝에 취약할 수 있습니다.

검증

- 호스트에서 **virsh dumpxml** 유틸리티를 사용하여 보안 VM의 XML 구성을 확인합니다. 구성에는 **< driver iommu='on'/ >** 및 **< memballoon model='none'/ >** 요소가 포함되어야 합니다.

```
# virsh dumpxml vm-name
[...]
 <cpu mode='host-model'/>
 <devices>
  <disk type='file' device='disk'>
   <driver name='qemu' type='qcow2' cache='none' io='native' iommu='on'>
   <source file='/var/lib/libvirt/images/secure-guest.qcow2'/>
   <target dev='vda' bus='virtio'/>
  </disk>
  <interface type='network'>
   <driver iommu='on'/>
   <source network='default'/>
   <model type='virtio'/>
  </interface>
  <console type='pty'/>
  <memballoon model='none'/>
 </devices>
</domain>
```

추가 리소스

- 커널 명령줄 매개변수 구성
- genprotimg의 IBM 문서

18.8. IBM Z의 가상 머신에 암호화 COPROCESSORS 연결

IBM Z 호스트의 VM(가상 머신)에서 하드웨어 암호화를 사용하려면 암호화 coprocessor 장치에서 중재된 장치를 생성하여 원하는 VM에 할당합니다. 자세한 지침은 아래를 참조하십시오.

사전 요구 사항

- 호스트는 IBM Z 하드웨어에서 실행됩니다.
- 암호화 coprocessor는 장치 할당과 호환됩니다. 이를 확인하려면 coprocessor 유형이 CEX4 이상으로 나열되어 있는지 확인하십시오.

```
# lszcrypt -V
```

```
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH
FUNCTIONS  DRIVER
--------------------------------------------------------------------------------
05         CEX5C CCA-Coproc  online       1        0     11     08 S--D--N--  cex4card
05.0004    CEX5C CCA-Coproc  online       1        0     11     08 S--D--N--  cex4queue
05.00ab    CEX5C CCA-Coproc  online       1        0     11     08 S--D--N--  cex4queue
```

- *mdevctl 패키지가 설치됩니다.*

- *vfio_ap 커널 모듈이 로드됩니다. 확인하려면 다음을 사용합니다.*

```
# lsmod | grep vfio_ap
vfio_ap         24576  0
[...]
```

모듈을 로드하려면 다음을 사용합니다.

```
# modprobe vfio_ap
```

절차

1. *호스트에서 vfio-ap 드라이버에 암호화 장치를 다시 할당합니다. 다음 예제에서는 비트마스크 ID (0x05, 0x0004) 및 (0x05, 0x00ab) 를 vfio-ap 에 사용하는 두 암호화 장치를 할당합니다.*

```
#  echo -0x05 > /sys/bus/ap/apmask
#  echo -0x0004, -0x00ab > /sys/bus/ap/aqmask
```

비트마스크 ID 값을 식별하는 방법에 대한 자세한 내용은 IBM의 KVM 가상 서버 관리 문서에 있는 암호화 어댑터 리소스에 대한 패스스루(pass-through) 장치 준비를 참조하십시오.

2. *암호화 장치가 올바르게 할당되었는지 확인합니다.*

```
# lszcrypt -V

CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH
FUNCTIONS  DRIVER
--------------------------------------------------------------------------------
05         CEX5C CCA-Coproc  -            1        0     11     08 S--D--N--  cex4card
05.0004    CEX5C CCA-Coproc  -            1        0     11     08 S--D--N--  vfio_ap
05.00ab    CEX5C CCA-Coproc  -            1        0     11     08 S--D--N--  vfio_ap
```

*도메인 대기열의 **DRIVER** 값이 **vfio_ap** 로 변경된 경우 재할당에 성공했습니다.*

3. *장치 **UUID**를 생성합니다.*

```
# uuidgen
669d9b23-fe1b-4ecb-be08-a2fabca99b71
```

*이 절차의 다음 단계에서 **669d9b23-fe1b-4ecb-be08-a2fabca99b71** 을 생성된 **UUID**로 바꿉니다.*

4. ***UUID**를 사용하여 새 **vfio_ap** 장치를 만듭니다.*

*다음 예제에서는 영구 중재 장치를 생성하고 큐를 할당하는 방법을 보여줍니다. 예를 들어, 다음 명령은 도메인 어댑터 **0x05** 및 도메인 대기열 **0x0004** 및 **0x00ab** 을 장치 **669d9b23-fe1b-4ecb-be08-a2fab99b71** 에 할당합니다.*

```
# mdevctl define --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --parent matrix --type
vfio_ap-passthrough
# mdevctl modify --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --
addattr=assign_adapter --value=0x05
# mdevctl modify --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --
addattr=assign_domain --value=0x0004
# mdevctl modify --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --
addattr=assign_domain --value=0x00ab
```

5. *중재된 장치를 시작합니다.*

```
# mdevctl start --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71
```

6. *구성이 올바르게 적용되었는지 확인합니다.*

```
# cat /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-
passthrough/devices/669d9b23-fe1b-4ecb-be08-a2fabca99b71/matrix
05.0004
05.00ab
```

*출력에 이전에 **vfio-ap** 에 할당한 대기열의 숫자 값이 포함된 경우 프로세스가 성공적으로 수행되었습니다.*

7. *virsh edit 명령을 사용하여 암호화 장치를 사용하려는 VM의 XML 구성을 엽니다.*

```
# virsh edit vm-name
```

8. *XML 구성의 &lt ;devices& gt; 섹션에 다음 행을 추가하고 저장합니다.*

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ap'>
  <source>
    <address uuid='669d9b23-fe1b-4ecb-be08-a2fabca99b71'/>
  </source>
</hostdev>
```

각 UUID는 한 번에 하나의 VM에만 할당할 수 있습니다.

9. *중재된 장치에 필요한 제어 도메인을 할당합니다.*

```
# mdevctl modify --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --
addattr=assign_control_domain --value=0x00ab
# mdevctl modify --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --
addattr=assign_control_domain --value=0x0004
```

검증

1. *중재된 장치가 할당된 VM을 시작합니다.*

2. *OS(게스트 운영 체제)가 부팅된 후 할당된 암호화 장치를 감지하는지 확인합니다.*

```
# lszcrypt -V

CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH
FUNCTIONS  DRIVER
--------------------------------------------------------------------------------
05          CEX5C CCA-Coproc  online        1        0    11     08 S--D--N--  cex4card
05.0004     CEX5C CCA-Coproc  online        1        0    11     08 S--D--N--  cex4queue
05.00ab     CEX5C CCA-Coproc  online        1        0    11     08 S--D--N--  cex4queue
```

게스트 OS에서 이 명령의 출력은 사용 가능한 동일한 암호화 coprocessor 장치가 있는 호스트 논리 파티션의 출력과 동일합니다.

3.

게스트 OS에서 제어 도메인이 암호화 장치에 성공적으로 할당되었는지 확인합니다.

```
# lszcrypt -d C

DOMAIN 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
------------------------------------------------------
  00 . . . . . . . . . . . . . . . .
  10 . . . . . . . . . . . . . . . .
  20 . . . . . . . . . . . B . . . .
  30 . . B  B . . . . . . . . . . . .
  40 . . . . . . . . . . . . . . . .
  50 . . . . . . . . . . . . . . . .
  60 . . . . . . . . . . . . . . . .
  70 . . . . . . . . . . . . . . . .
  80 . . . . . . . . . . . . . . . .
  90 . . . . . . . . . . . . . . . .
  a0 . . . . . . . . . . . . . . . .
  b0 . . . . . . . . . . . . . . . .
  c0 . . . . . . . . . . . . . . . .
  d0 . . . . . . . . . . . . . . . .
  e0 . . . . . . . . . . . . . . . .
  f0 . . . . . . . . . . . . . . . .
------------------------------------------------------
```

lszcrypt -d C 가 암호화 장치 매트릭스에 B 교집합을 표시하는 경우 컨트롤 도메인 할당에 성공했습니다.

18.9. WINDOWS 가상 머신에서 표준 하드웨어 보안 활성화

Windows VM(가상 머신)을 보호하려면 Windows 장치의 표준 하드웨어 기능을 사용하여 기본 수준 보안을 활성화할 수 있습니다.

사전 요구 사항

- 최신 WHQL 인증 VirtIO 드라이버를 설치했는지 확인하십시오.
- VM의 펌웨어가 UEFI 부팅을 지원하는지 확인합니다.
- 호스트 시스템에 edk2-OVMF 패키지를 설치합니다.

```
# dnf install edk2-ovmf
```

- 호스트 시스템에 vTPM 패키지를 설치합니다.

```
# dnf install swtpm libtpms
```

- VM에서 Q35 시스템 아키텍처를 사용하고 있는지 확인합니다.

- Windows 설치 미디어가 있는지 확인합니다.

절차

1. VM의 XML 구성의 < devices > 섹션에 다음 매개 변수를 추가하여 TPM 2.0을 활성화합니다.

```
<devices>
[...]
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'/>
  </tpm>
[...]
</devices>
```

2. UEFI 모드에서 Windows를 설치합니다. 이를 수행하는 방법에 대한 자세한 내용은 SecureBoot 가상 머신 생성을 참조하십시오.

3. Windows VM에 VirtIO 드라이버를 설치합니다. 이를 수행하는 방법에 대한 자세한 내용은 Windows 게스트에 virtio 드라이버 설치를 참조하십시오.

4. UEFI에서 Secure Boot를 활성화합니다. 이 작업을 수행하는 방법에 대한 자세한 내용은 Secure Boot 를 참조하십시오.

검증

- Windows 머신의 장치 보안 페이지에 다음 메시지가 표시되는지 확인합니다.

설정 > 업데이트 및 보안 > Windows 보안 > 장치 보안

```
Your device meets the requirements for standard hardware security.
```

18.10. WINDOWS 가상 머신에서 향상된 하드웨어 보안 활성화

Windows VM(Windows 가상 머신)을 보다 안전하게 보호하려면 **HVCI(**하이퍼바이저 보호 코드 무결성)라고도 하는 코드 무결성의 가상화 기반 보호를 활성화할 수 있습니다.

사전 요구 사항

- 표준 하드웨어 보안이 활성화되어 있는지 확인합니다. 자세한 내용은 **Windows** 가상 머신에서 표준 하드웨어 보안 활성화를 참조하십시오.
- **Hyper-V** 시행을 활성화했는지 확인합니다. 자세한 내용은 **Enabling Hyper-V Enlightenments**를 참조하십시오.

절차

1. **Windows VM**의 **XML** 구성을 엽니다. 다음 예제에서는 **Example-L1 VM**의 구성을 엽니다.

```
# virsh edit Example-L1
```

2. **&lt ;cpu& gt;** 섹션에서 **CPU** 모드를 지정하고 정책 플래그를 추가합니다.

중요

- **Intel CPU**의 경우 **vmx** 정책 플래그를 활성화합니다.
- **AMD CPU**의 경우 **svm** 정책 플래그를 활성화합니다.
- 사용자 정의 **CPU**를 지정하지 않으려면 **< cpu mode >**를 **host-passthrough** 로 설정할 수 있습니다.

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>Skylake-Client-IBRS</model>
  <topology sockets='1' dies='1' cores='4' threads='1'/>
  <feature policy='require' name='vmx'/>
</cpu>
```

3. *XML 구성을 저장하고 VM을 재부팅합니다.*

4. *VM 운영 체제에서 코어 격리 세부 정보 페이지로 이동합니다.*

   설정 > 업데이트 및 보안 > Windows 보안 > 장치 보안 > 핵심 격리 세부 정보

5. *스위치를 전환하여 메모리 무결성을 활성화합니다.*

6. *VM을 재부팅합니다.*

참고

HVCI를 활성화하는 다른 방법은 관련 Microsoft 문서를 참조하십시오.

검증

- *Windows VM의 장치 보안 페이지에 다음 메시지가 표시되는지 확인합니다.*

  설정 > 업데이트 및 보안 > Windows 보안 > 장치 보안

  > ***Your device meets the requirements for enhanced hardware security.***

- *또는 Windows VM의 시스템 정보를 확인합니다.*

  a. *명령 프롬프트에서 msinfo32.exe 를 실행합니다.*

  b. *Credential Guard를 확인하고, Hypervisor 강제 코드 무결성 이 가상화 기반 보안 서비스 실행 아래에 나열되어 있는지 확인합니다.*

19장. 호스트와 가상 머신 간 파일 공유

호스트 시스템과 실행되는 VM(가상 머신) 간에 데이터를 공유해야 하는 경우가 많습니다. 이를 위해서는 시스템에 NFS 또는 Samba 파일 공유를 빠르고 효율적으로 설정할 수 있습니다. RHEL 9에서 새로 지원되는 기능은 virtiofs 파일 시스템을 사용하여 Linux VM과 데이터를 공유할 수도 있습니다.

19.1. VIRTIOFS를 사용하여 호스트와 해당 가상 머신 간 파일 공유

RHEL 9를 하이퍼바이저로 사용하는 경우 virtiofs 기능을 사용하여 호스트 시스템과 해당 VM(가상 머신) 간에 파일을 효율적으로 공유할 수 있습니다.

사전 요구 사항

- RHEL 9 호스트에 가상화가 설치 및 활성화되어 있습니다.
- VM과 공유할 디렉터리입니다. 기존 디렉터리를 공유하지 않으려면 새 디렉터리(예: shared-files)를 만듭니다.

```
# mkdir /root/shared-files
```

- 데이터를 공유하려는 VM은 Linux 배포를 게스트 OS로 사용하는 것입니다.

절차

1. VM과 공유할 호스트의 각 디렉터리에 대해 VM의 XML 구성에서 virtiofs 파일 시스템으로 설정합니다.

   a. 원하는 VM의 XML 구성을 엽니다.

   ```
   # virsh edit vm-name
   ```

   b. 다음과 유사한 항목을 VM XML 구성의 <devices > 섹션에 추가합니다.

   ```
   <filesystem type='mount' accessmode='passthrough'>
     <driver type='virtiofs'/>
     <binary path='/usr/libexec/virtiofsd' xattr='on'/>
   ```

```
  <source dir='/root/shared-files'/>
  <target dir='host-file-share'/>
</filesystem>
```

이 예에서는 호스트의 **/root/shared-files** 디렉터리를 **host-file-share** 로 VM으로 표시되도록 설정합니다.

2. **XML** 구성에 공유 메모리의 **NUMA** 토폴로지를 추가합니다. 다음 예제에서는 모든 **CPU** 및 모든 **RAM**에 대한 기본 토폴로지를 추가합니다.

```
<cpu mode='host-passthrough' check='none'>
 <numa>
  <cell id='0' cpus='0-{number-vcpus - 1}' memory='{ram-amount-KiB}' unit='KiB'
memAccess='shared'/>
 </numa>
</cpu>
```

3. **XML** 구성의 **< domain>** 섹션에 공유 메모리 백업을 추가합니다.

```
<domain>
[...]
 <memoryBacking>
  <access mode='shared'/>
 </memoryBacking>
[...]
</domain>
```

4. **VM**을 부팅합니다.

```
# virsh start vm-name
```

5. 게스트 운영 체제(**OS**)에 파일 시스템을 마운트합니다. 다음 예제는 이전에 구성된 **host-file-share** 디렉터리를 **Linux** 게스트 **OS**와 함께 마운트합니다.

```
# mount -t virtiofs host-file-share /mnt
```

검증

- **VM**에서 공유 디렉터리에 액세스할 수 있게 되고 이제 디렉터리에 저장된 파일을 열 수 있는지 확인합니다.

알려진 문제 및 제한 사항

- *noatime* 및 *strictatime* 과 같은 액세스 시간과 관련된 파일 시스템 마운트 옵션은 **virtiofs**와 호환되지 않으며 **Red Hat**의 사용을 권장하지 않습니다.

문제 해결

- **virtiofs** 가 사용 사례에 적합하지 않거나 시스템에 대해 지원되는 경우 **NFS** 또는 **Samba** 를 대신 사용할 수 있습니다.

19.2. VIRTIOFS를 사용하여 호스트와 가상 머신 간에 파일을 공유하는 웹 콘솔을 사용합니다.

RHEL 웹 콘솔을 사용하여 **virtiofs** 기능을 사용하여 호스트 시스템과 **VM**(가상 머신) 간에 파일을 효율적으로 공유할 수 있습니다.

사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 시스템에 설치되어 있습니다.

- **VM**과 공유할 디렉터리입니다. 기존 디렉터리를 공유하지 않으려면 새 디렉터리를 만듭니다 (예: **centurion**).

```
# mkdir /home/centurion
```

- 데이터를 공유하려는 **VM**은 **Linux** 배포를 게스트 **OS**로 사용하는 것입니다.

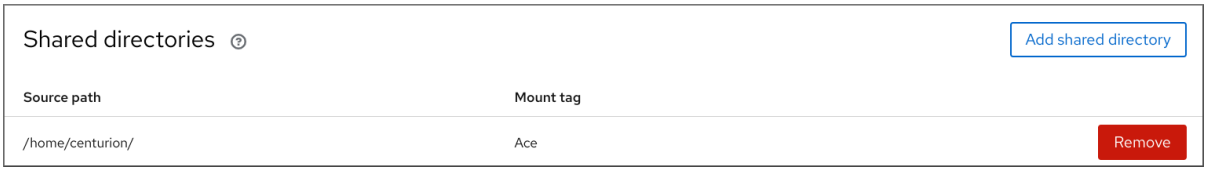
절차

1. **Virtual Machines** (가상 시스템) 인터페이스에서 파일을 공유할 **VM**을 클릭합니다.

   선택한 **VM** 및 콘솔 섹션에 대한 기본 정보가 포함된 개요 섹션이 포함된 새 페이지가 열립니다.

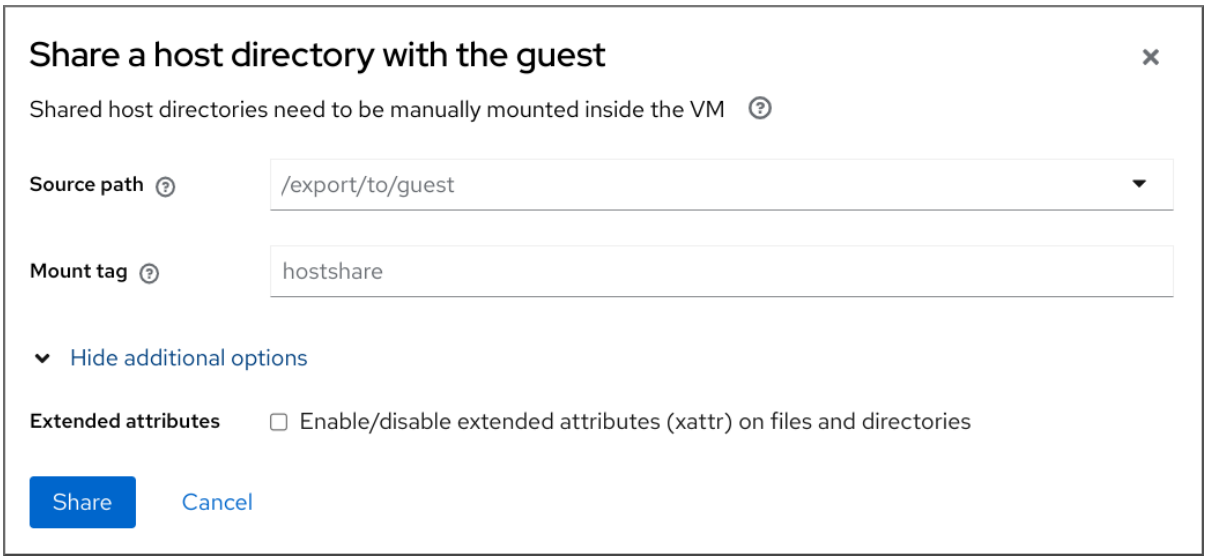
2. 공유 디렉터리 로 스크롤합니다.

   공유 디렉터리 섹션에는 공유 디렉터리를 추가 또는 제거하는 해당 **VM** 및 옵션과 함께 공유하는 호스트 파일 및 디렉터리에 대한 정보가 표시됩니다.



3. *공유 디렉터리 추가 를 클릭합니다.*

게스트를 사용하여 호스트 디렉터리 공유 대화 상자가 나타납니다.



4. *다음 정보를 입력합니다.*

- ***source path** - 공유할 호스트 디렉터리의 경로입니다.*
- ***Mount tag** - VM에서 디렉터리를 마운트하는 데 사용하는 태그입니다.*

5. *추가 옵션을 설정합니다.*

- *확장 속성 - 공유 파일 및 디렉터리에서 확장 속성 **xattr** 을 활성화할지 여부를 설정합니다.*

6. *공유를 클릭합니다.*

선택한 디렉터리가 VM과 공유됩니다.

검증

- VM에서 공유 디렉터리에 액세스할 수 있고 이제 해당 디렉터리에 저장된 파일을 열 수 있는지 확인합니다.

19.3. 웹 콘솔을 사용하여 VIRTIOFS를 사용하여 호스트와 해당 가상 머신 간의 공유 파일을 제거

RHEL 웹 콘솔을 사용하여 virtiofs 기능을 사용하여 호스트 시스템과 해당 VM(가상 머신) 간에 공유되는 파일을 제거할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.
- 디렉터리는 VM에서 더 이상 사용되지 않습니다.

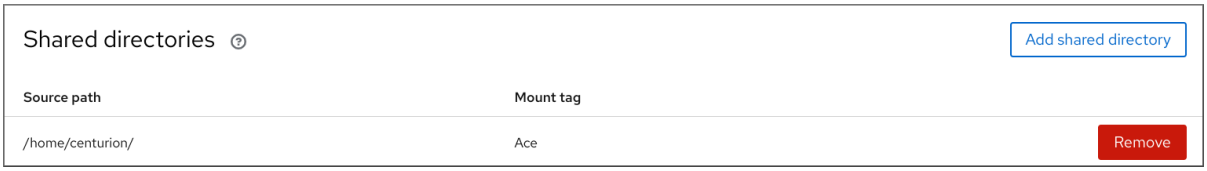
절차

1. Virtual Machines (가상 머신) 인터페이스에서 공유 파일을 제거할 VM을 클릭합니다.

   선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 개요 섹션이 포함된 새 페이지가 열립니다.

2. 공유 디렉터리 로 스크롤합니다.

   공유 디렉터리 섹션에는 공유 디렉터리를 추가 또는 제거하는 해당 VM 및 옵션과 함께 공유하는 호스트 파일 및 디렉터리에 대한 정보가 표시됩니다.



3. VM과 공유할 디렉토리 옆에 있는 Remove (제거)를 클릭합니다.

   파일 시스템 제거 대화 상자가 나타납니다.

4. *제거를 클릭합니다.*

   선택한 디렉토리는 VM과 공유되지 않습니다.

검증

- *공유 디렉터리는 더 이상 사용할 수 없으며 VM에서 액세스할 수 있습니다.*

19.4. NFS를 사용하여 호스트와 LINUX 가상 시스템 간 파일 공유

RHEL 9 호스트 시스템과 연결된 Linux VM 간의 파일 공유를 효율적으로 공유하려면 virtiofs 기능을 사용합니다. 그러나 virtiofs 가 사용자에게 작동하지 않거나 사용 사례에 적합하지 않은 경우 VM에서 마운트하고 액세스할 수 있는 NFS 공유를 내보낼 수 있습니다.

사전 요구 사항

- ***nfs-utils** 패키지가 호스트에 설치되어 있습니다.*

- *VM과 공유할 디렉터리입니다. 기존 디렉터리를 공유하지 않으려면 새 디렉터리(예: **shared-files**)를 만듭니다.*

```
# mkdir shared-files
```

- *VM의 네트워크를 통해 호스트를 표시하고 연결할 수 있습니다. 일반적으로 VM이 NAT 및 가상 네트워크의 브리지 유형을 사용하여 연결된 경우입니다. 그러나 macvtap 연결의 경우 먼저 호스트에서 macvlan 기능을 설정해야 합니다. 이를 위해 다음을 수행합니다.*

  1. *호스트의 /etc/systemd/network/ 디렉터리에 네트워크 장치 파일을 만듭니다(예: vm-macvlan.netdev).*

```
# vim /etc/systemd/network/vm-macvlan.netdev
```

2. *다음 콘텐츠를 포함하도록 네트워크 장치 파일을 편집합니다. **vm-macvlan** 을 네트워크 장치에 대해 선택한 이름으로 교체할 수 있습니다.*

```
[NetDev]
Name=vm-macvlan
Kind=macvlan

[MACVLAN]
Mode=bridge
```

3. ***macvlan** 네트워크 장치의 네트워크 구성 파일을 생성합니다(예: **vm-macvlan.network**).*

```
# vim /etc/systemd/network/vm-macvlan.network
```

4. *다음 콘텐츠를 포함하도록 네트워크 구성 파일을 편집합니다. **vm-macvlan** 을 네트워크 장치에 대해 선택한 이름으로 교체할 수 있습니다.*

```
[Match]
Name=_vm-macvlan_

[Network]
IPForward=yes
Address=192.168.250.33/24
Gateway=192.168.250.1
DNS=192.168.250.1
```

5. *실제 네트워크 인터페이스에 대한 네트워크 구성 파일을 만듭니다. 예를 들어 인터페이스가 **enp4s0** 인 경우 :*

```
# vim /etc/systemd/network/enp4s0.network
```

*사용할 인터페이스 이름이 확실하지 않은 경우 호스트에서 **ifconfig** 명령을 사용하여 활성 네트워크 인터페이스 목록을 가져올 수 있습니다.*

6. *물리적 네트워크 구성 파일을 편집하여 물리적 네트워크를 **macvlan** 인터페이스(이 경우 **vm-macvlan**)의 일부로 만듭니다.*

```
[Match]
Name=enp4s0

[Network]
MACVLAN=vm-macvlan
```

7. *호스트를 재부팅합니다.*

- *선택 사항: 향상된 보안을 위해 VM이 NFS 버전 4 이상과 호환되는지 확인합니다.*

절차

1. *호스트에서 NFS(네트워크 파일 시스템)로 공유할 파일이 있는 디렉터리를 내보냅니다.*

a. *파일을 공유하려는 각 가상 머신의 IP 주소를 가져옵니다. 다음 예제에서는 testguest1 및 testguest2 의 IP를 가져옵니다.*

```
# virsh domifaddr testguest1
Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
vnet0      52:53:00:84:57:90    ipv4         192.168.124.220/24

# virsh domifaddr testguest2
Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
vnet1      52:53:00:65:29:21    ipv4         192.168.124.17/24
```

b. *호스트에서 /etc/exports 파일을 편집하고 공유할 디렉터리, 공유할 VM IP 및 공유 옵션을 포함하는 행을 추가합니다.*

```
Shared directory VM1-IP(options) VM2-IP(options) [...]
```

예를 들어 다음에서는 testguest1 및 testguest2 를 사용하여 호스트의 /usr/local/shared-files 디렉터리를 공유하고 VM이 디렉터리의 콘텐츠를 편집할 수 있습니다.

```
/usr/local/shared-files/ 192.168.124.220(rw,sync) 192.168.124.17(rw,sync)
```

c. *업데이트된 파일 시스템을 내보냅니다.*

```
# exportfs -a
```

d. *NFS 프로세스가 시작되었는지 확인합니다.*

-

```
# systemctl start nfs-server
```

e. 호스트 시스템의 **IP** 주소를 가져옵니다. 이는 나중에 **VM**에 공유 디렉터리를 마운트하는 데 사용됩니다.

```
# ip addr
[...]
5: virbr0: [BROADCAST,MULTICAST,UP,LOWER_UP] mtu 1500 qdisc noqueue
state UP group default qlen 1000
    link/ether 52:54:00:32:ff:a5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.124.1/24 brd 192.168.124.255 scope global virbr0
       valid_lft forever preferred_lft forever
[...]
```

관련 네트워크는 파일을 공유하려는 **VM**에서 호스트 연결에 사용하는 네트워크입니다. 일반적으로 **virbr0** 입니다.

2. **/etc/exports** 파일에 지정된 **VM**의 게스트 **OS**에서 내보낸 파일 시스템을 마운트합니다.

a. 공유 파일 시스템의 마운트 지점으로 사용할 디렉토리를 만듭니다(예: **/mnt/host-share**):

```
# mkdir /mnt/host-share
```

b. 호스트에서 내보낸 디렉터리를 마운트 지점에 마운트합니다. 이 예제에서는 게스트의 **/mnt/host-share** 에 **192.168.124.1** 호스트에서 내보낸 **/usr/local/shared-files** 디렉터리를 마운트합니다.

```
# mount 192.168.124.1:/usr/local/shared-files /mnt/host-share
```

검증

- 마운트가 성공했는지 확인하려면 액세스한 후 마운트 지점의 공유 디렉토리를 살펴봅니다.

```
# cd /mnt/host-share
# ls
shared-file1  shared-file2  shared-file3
```

19.5. SAMBA를 사용하여 호스트와 WINDOWS 가상 시스템 간에 파일 공유

RHEL 9 호스트 시스템과 연결된 Windows VM 간의 파일 공유를 효율적으로 공유하려면 virtiofs 기능을 사용합니다. 그러나 virtiofs 가 사용자를 위해 작동하지 않거나 사용 사례에 적합하지 않은 경우 VM에서 액세스할 수 있는 Samba 서버를 준비할 수 있습니다.

사전 요구 사항

- samba 패키지가 호스트에 설치되어 있습니다. 그렇지 않은 경우 다음을 수행합니다.

```
# dnf install samba
```

- VM의 네트워크를 통해 호스트를 표시하고 연결할 수 있습니다. 일반적으로 VM이 NAT 및 가상 네트워크의 브리지 유형을 사용하여 연결된 경우입니다. 그러나 macvtap 연결의 경우 먼저 호스트에서 macvlan 기능을 설정해야 합니다. 이를 위해 다음을 수행합니다.

1. 호스트의 /etc/systemd/network/ 디렉터리에 vm-macvlan.netdev 라는 네트워크 장치 파일을 만듭니다.

```
# vim /etc/systemd/network/vm-macvlan.netdev
```

2. 다음 콘텐츠를 포함하도록 네트워크 장치 파일을 편집합니다. vm-macvlan 을 네트워크 장치에 대해 선택한 이름으로 교체할 수 있습니다.

```
[NetDev]
Name=vm-macvlan
Kind=macvlan

[MACVLAN]
Mode=bridge
```

3. macvlan 네트워크 장치의 네트워크 구성 파일을 생성합니다(예: vm-macvlan.network).

```
# vim /etc/systemd/network/vm-macvlan.network
```

4. 다음 콘텐츠를 포함하도록 네트워크 구성 파일을 편집합니다. vm-macvlan 을 네트워크 장치에 대해 선택한 이름으로 교체할 수 있습니다.

```
[Match]
Name=_vm-macvlan_

[Network]
```

```
IPForward=yes
Address=192.168.250.33/24
Gateway=192.168.250.1
DNS=192.168.250.1
```

5. *실제 네트워크 인터페이스에 대한 네트워크 구성 파일을 만듭니다. 예를 들어 인터페이스가 **enp4s0** 인 경우 :*

```
# vim /etc/systemd/network/enp4s0.network
```

*사용할 인터페이스가 확실하지 않은 경우 호스트에서 **ifconfig** 명령을 사용하여 활성 네트워크 인터페이스 목록을 가져올 수 있습니다.*

6. *물리적 네트워크 구성 파일을 편집하여 물리적 네트워크를 **macvlan** 인터페이스(이 경우 **vm-macvlan**)의 일부로 만듭니다.*

```
[Match]
Name=enp4s0

[Network]
MACVLAN=vm-macvlan
```

7. *호스트를 재부팅합니다.*

절차

1. *호스트에서 **Samba** 공유를 만들고 외부 시스템에서 액세스할 수 있도록 합니다.*

a. ***Samba**에 대한 방화벽 권한을 추가합니다.*

```
# firewall-cmd --permanent --zone=public --add-service=samba
success
# firewall-cmd --reload
success
```

b. */etc/samba/smb.conf* *파일을 편집합니다.*

i. *[global] 섹션에 다음을 추가합니다.*

```
map to guest = Bad User
```

ii. 파일 끝에 다음을 추가합니다.

```
#=== Share Definitions ===
[VM-share]
path = /samba/VM-share
browsable = yes
guest ok = yes
read only = no
hosts allow = 192.168.122.0/24
```

호스트 allow 행은 VM 네트워크의 호스트에만 공유 액세스 가능성을 제한합니다. 모든 사용자가 공유에 액세스할 수 있도록 하려면 행을 제거합니다.

c. /hiera/VM-share 디렉터리를 생성합니다.

```
# mkdir -p /samba/VM-share
```

d. Samba 서비스를 활성화합니다.

```
# systemctl enable smb.service
Created symlink /etc/systemd/system/multi-user.target.wants/smb.service →
/usr/lib/systemd/system/smb.service.
```

e. Samba 서비스를 다시 시작합니다.

```
# systemctl restart smb.service
```

f. VM-share 디렉터리에 액세스할 수 있고 VM에서 수정할 수 있도록 허용합니다.

```
# chmod -R 0755 /samba/VM-share/
# chown -R nobody:nobody /samba/VM-share/
```

g. SELinux Samba 공유 레이블을 /etc/hiera/VM-share/에 추가합니다.

```
# chcon -t samba_share_t /samba/VM-share/
```

2.

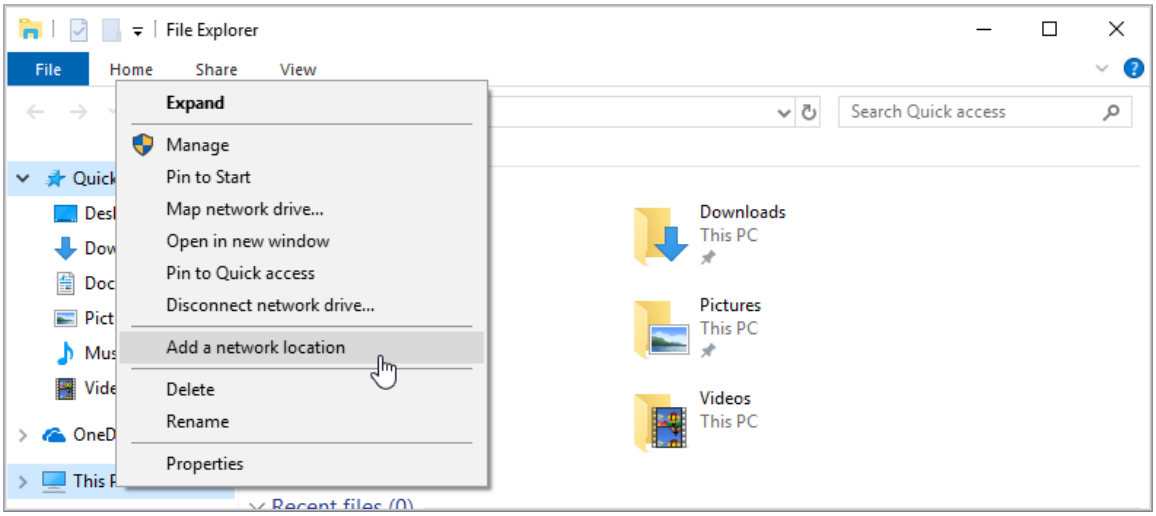
***Windows** 게스트 운영 체제에서는 **Samba** 공유를 네트워크 위치로 연결합니다.*

a.

*파일 탐색기를 열고 **"This PC"**를 마우스 오른쪽 버튼으로 클릭합니다.*

b.

컨텍스트 메뉴에서 네트워크 위치 추가를 클릭합니다.

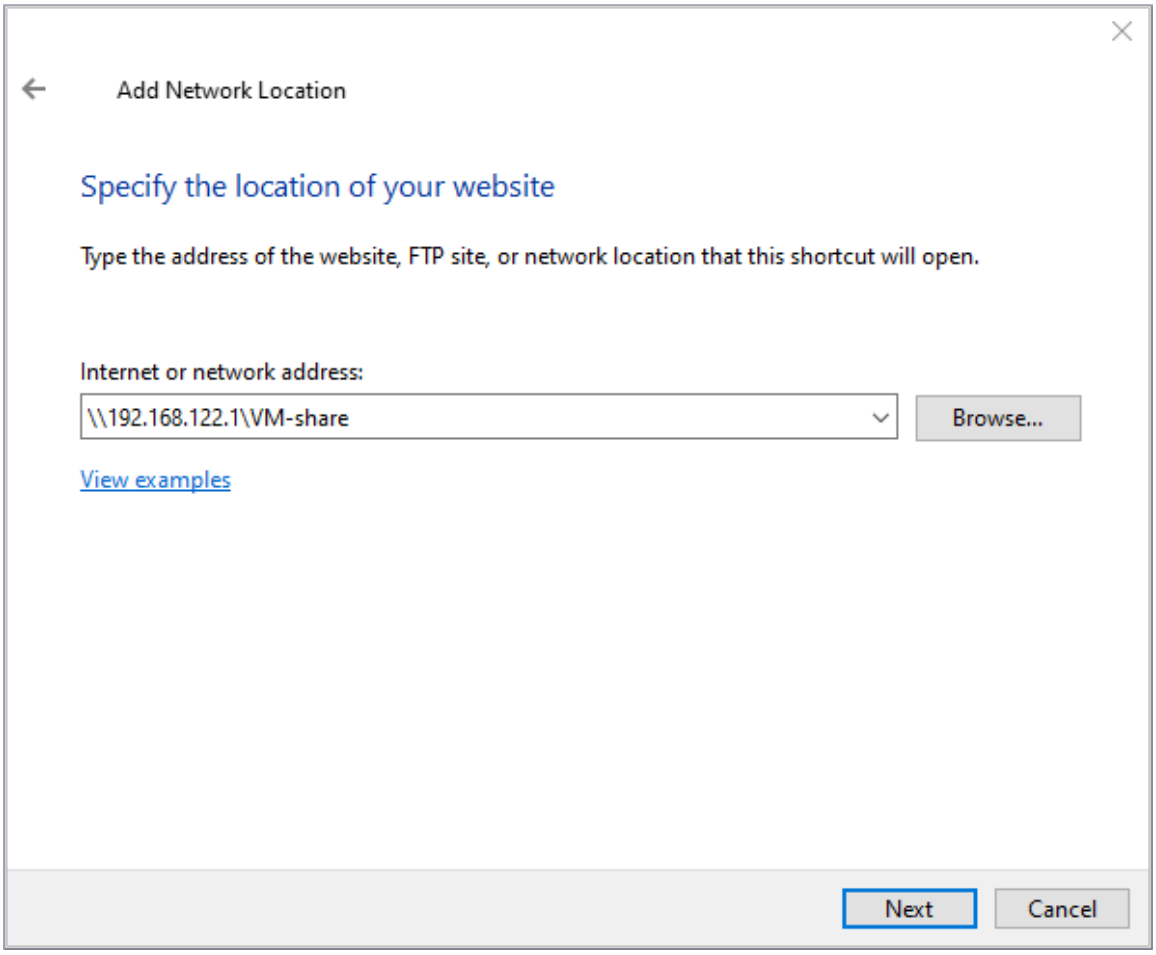


c.

네트워크 위치 추가 마법사에서 "사용자 지정 네트워크 위치 선택"를 선택하고 다음을 클릭합니다.

d.

*"Internet or network address" 필드에 **host-IP/VM-share**를 입력합니다. 여기서 **host-IP** 는 호스트의 **IP** 주소입니다. 일반적으로 호스트 **IP**는 **VM**의 기본 게이트웨이입니다. 그런 다음 다음을 클릭합니다.*



e.

*마법사에서 공유 디렉터리의 이름을 변경할지 묻는 경우 기본 이름을 유지합니다. 이렇게 하면 **VM** 및 게스트에서 파일 공유 구성의 일관성이 유지됩니다. 다음을 클릭합니다.*

f.

네트워크 위치에 액세스하는 경우 완료 를 클릭하고 공유 디렉터리를 열 수 있습니다.

20장. WINDOWS 가상 머신 설치 및 관리

RHEL 9 호스트에서 VM(가상 머신)의 게스트 운영 체제로 Microsoft Windows를 사용하려면 이러한 VM이 올바르게 실행되도록 추가 단계를 수행하는 것이 좋습니다.

이 목적을 위해 다음 섹션에서는 호스트에 Windows VM을 설치 및 최적화하고 이러한 VM에 드라이버를 설치 및 구성하는 데 대한 정보를 제공합니다.

20.1. WINDOWS 가상 머신 설치

RHEL 9 호스트에서 완전히 가상화된 Windows 머신을 생성하고 VM(가상 머신) 내에서 그래픽 Windows 설치 프로그램을 시작하고 설치된 Windows 게스트 운영 체제(OS)를 최적화할 수 있습니다.

VM을 생성하고 Windows 게스트 OS를 설치하려면 virt-install 명령 또는 RHEL 9 웹 콘솔을 사용합니다.

사전 요구 사항

- 다음 중 하나일 수 있으며 로컬 또는 네트워크에서 사용할 수 있는 Windows OS 설치 소스입니다.
  - 설치 미디어의 ISO 이미지
  - 기존 VM 설치의 디스크 이미지
- KVM virtio 드라이버가 있는 스토리지 매체입니다.

  이 미디어를 생성하려면 호스트 시스템의 virtio 드라이버 설치 미디어 준비를 참조하십시오.
- Windows 11을 설치하는 경우 edk2-ovmf,swtpm 및 libtpms 패키지를 호스트에 설치해야 합니다.

절차

1. VM을 생성합니다. 자세한 내용은 가상 머신 생성을 참조하십시오. 그러나 다음과 같은 세부

사항을 고려하십시오.

- virt-install 유틸리티를 사용하여 VM을 생성하는 경우 명령에 다음 옵션을 추가합니다.
  - KVM virtio 드라이버가 있는 스토리지 매체입니다. 예를 들면 다음과 같습니다.

    ```
    --disk path=/usr/share/virtio-win/virtio-win.iso,device=cdrom
    ```

  - 설치할 Windows 버전입니다. 예를 들어 Windows 10 및 11의 경우:

    ```
    --os-variant win10
    ```

    사용 가능한 Windows 버전 및 적절한 옵션 목록은 다음 명령을 사용합니다.

    ```
    # osinfo-query os
    ```

  - Windows 11을 설치하는 경우 UEFI(Unified Extensible Firmware Interface) 및 vTPM(가상 신뢰할 수 있는 플랫폼 모듈)을 활성화합니다.

    ```
    --boot uefi
    ```

- 웹 콘솔을 사용하여 VM을 생성하는 경우 새 가상 머신 생성 창의 운영 체제 필드에 Windows 버전을 지정합니다.
  - Windows 11 및 Windows Server 2022 이전의 Windows 버전을 설치하는 경우 생성을 클릭하고 을 실행하여 설치를 시작합니다.
  - Windows 11을 설치하거나 추가 Windows Server 2022 기능을 사용하려면 CLI를 사용하여 UEFI 및 vTPM을 생성하고 활성화하여 확인하십시오.

    A. VM의 XML 구성을 엽니다.

    ```
    # virsh edit windows-vm
    ```

B.

os 요소에 firmware='efi' 옵션을 추가합니다.

```
<os firmware='efi'>
  <type arch='x86_64' machine='pc-q35-6.2'>hvm</type>
  <boot dev='hd'/>
</os>
```

C.

devices 요소 내에 tpm 장치를 추가합니다.

```
<devices>
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'/>
  </tpm>
</devices>
```

D.

가상 머신 테이블에서 설치를 클릭하여 Windows 설치를 시작합니다.

2. *VM에 Windows OS를 설치합니다.*

   Windows 운영 체제를 설치하는 방법에 대한 자세한 내용은 관련 Microsoft 설치 설명서를 참조하십시오.

3. *웹 콘솔을 사용하여 VM을 생성하는 경우 Disks 인터페이스를 사용하여 virtio 드라이버가 있는 스토리지 미디어를 VM에 연결합니다. 자세한 내용은 웹 콘솔을 사용하여 기존 디스크를 가상 머신에 연결을 참조하십시오.*

4. *Windows 게스트 OS에서 KVM virtio 드라이버를 구성합니다. 자세한 내용은 Windows 가상 머신 용 KVM 반가상화 드라이버 설치를 참조하십시오.*

추가 리소스

- *Windows 가상 머신 최적화*
- *Windows 가상 머신에서 표준 하드웨어 보안 활성화*
- *Windows 가상 머신에서 향상된 하드웨어 보안 활성화*

- 샘플 가상 머신 XML 구성

20.2. WINDOWS 가상 머신 최적화

RHEL 9에서 호스팅되는 VM(가상 머신)의 게스트 운영 체제로 Microsoft Windows를 사용하는 경우 게스트의 성능에 부정적인 영향을 미칠 수 있습니다.

따라서 Red Hat은 다음 작업을 수행하여 Windows VM을 최적화할 것을 권장합니다.

- 반가상화 드라이버 사용. 자세한 내용은 Windows 가상 머신 용 KVM 반가상화 드라이버 설치를 참조하십시오.
- Hyper-V 서브스크립션 활성화. 자세한 내용은 Enabling Hyper-V Enlightenments를 참조하십시오.
- NetKVM 드라이버 매개 변수 구성. 자세한 내용은 NetKVM 드라이버 매개변수 구성을 참조하십시오.
- Windows 백그라운드 프로세스 최적화 또는 비활성화. 자세한 내용은 Windows 가상 머신에서 백그라운드 프로세스 최적화를 참조하십시오.

20.2.1. Windows 가상 머신용 KVM 반가상화 드라이버 설치

Windows VM(가상 머신)의 성능을 향상시키는 주요 방법은 게스트 운영 체제(OS)에 Windows용 KVM 반가상화(virtio) 드라이버를 설치하는 것입니다.

이를 위해 다음을 수행합니다.

1. 호스트 시스템에 설치 미디어를 준비합니다. 자세한 내용은 호스트 시스템에서 virtio 드라이버 설치 미디어 준비를 참조하십시오.
2. 설치 미디어를 기존 Windows VM에 연결하거나 새 Windows VM을 생성할 때 연결합니다.
3.

Windows 게스트 OS에 virtio 드라이버를 설치합니다. 자세한 내용은 Windows 게스트에 virtio 드라이버 설치를 참조하십시오.

20.2.1.1. Windows virtio 드라이버 작동 방식

반가상화 드라이버는 I/O 대기 시간을 줄이고 거의 베어 메탈 수준으로의 처리량을 증가시켜 VM(가상 머신)의 성능을 향상시킵니다. I/O 보존 작업 및 애플리케이션을 실행하는 VM에 반가상화 드라이버를 사용하는 것이 좋습니다.

VirtIO 드라이버는 KVM 호스트에서 실행되는 Windows VM에서 사용할 수 있는 KVM의 반가상화 장치 드라이버입니다. 이러한 드라이버는 다음 드라이버를 포함하는 virtio-win 패키지에서 제공합니다.

- 블록(스토리지) 장치
- 네트워크 인터페이스 컨트롤러
- 비디오 컨트롤러
- 메모리 증대 장치
- Paravirtual 직렬 포트 장치
- 엔트로피 소스 장치
- Paravirtual panic 장치
- 마우스, 키보드 또는 태블릿과 같은 입력 장치
- 작은 에뮬레이션 장치 세트

참고

에뮬레이션된 virtio 및 할당된 장치에 대한 자세한 내용은 가상 장치 관리를 참조하십시오.

KVM virtio 드라이버를 사용하면 다음 Microsoft Windows 버전이 물리적 시스템과 유사하게 실행되어야 합니다.

- Windows Server 버전: Certified guest operating systems for Red Hat Enterprise Linux with KVM in the Red Hat Knowledgebase를 참조하십시오.
- Windows Desktop(서버가 아닌) 버전:
  - Windows 7(32비트 및 64비트 버전)
  - Windows 8(32비트 및 64비트 버전)
  - Windows 8.1(32비트 및 64비트 버전)
  - Windows 10(32비트 및 64비트 버전)
  - Windows 11 (64비트)

20.2.1.2. 호스트 머신에서 virtio 드라이버 설치 미디어 준비

Windows VM(가상 머신)에 KVM virtio 드라이버를 설치하려면 먼저 호스트 머신에서 virtio 드라이버를 위한 설치 미디어를 준비해야 합니다. 이렇게 하려면 호스트 시스템에 virtio-win 패키지를 설치하고 VM용 스토리지로 제공되는 .iso 파일을 사용합니다.

사전 요구 사항

- RHEL 9 호스트 시스템에서 가상화가 활성화되어 있는지 확인합니다. 자세한 내용은 가상화 활성화를 참조하십시오.

절차

1. *드라이버 다운로드*

   a. *Red Hat Enterprise Linux를 다운로드하여 다운로드합니다.*

   b. *시스템 아키텍처와 관련된 **Product Variant** 를 선택합니다. 예를 들어 **Intel 64** 및 **AMD64**의 경우 **x86_64용 Red Hat Enterprise Linux**를 선택합니다.*

   c. *호스트 시스템의 버전을 선택합니다.*

   d. ***Packages** (패키지) 탭에서 **virtio-win** 을 검색합니다.*

   e. ***virtio-win AppStream** 패키지 옆에 있는 **Download Latest** 를 클릭합니다.*

   ***RPM** 파일이 다운로드됩니다.*

2. *다운로드 디렉터리에서 **virtio-win** 패키지를 설치합니다. 예를 들면 다음과 같습니다.*

```
# dnf install ~/Downloads/virtio-win-1.9.9-3.el8.noarch.rpm
[...]
Installed:
  virtio-win-1.9.9-3.el8.noarch
```

   *설치에 성공하면 **virtio-win** 드라이버 파일이 **/usr/share/virtio-win/** 디렉터리에 준비됩니다. 여기에는 디렉터리의 드라이버 파일이 있는 **ISO** 파일 및 드라이버 디렉터리, 각 아키텍처 및 지원되는 **Windows** 버전에 하나씩 포함됩니다.*

```
# ls /usr/share/virtio-win/
drivers/  guest-agent/  virtio-win-1.9.9.iso  virtio-win.iso
```

3. ***virtio-win.iso** 파일을 **Windows VM**에 연결합니다. 이렇게 하려면 다음 중 하나를 수행합니다.*

   - *새 Windows VM을 생성할 때 파일을 디스크로 사용합니다.*

- 파일을 기존 Windows VM에 CD-ROM으로 추가합니다. 예를 들면 다음과 같습니다.

```
# virt-xml WindowsVM --add-device --disk virtio-win.iso,device=cdrom
Domain 'WindowsVM' defined successfully.
```

다음 단계

- virtio-win.iso 가 Windows VM에 연결된 경우 Windows 게스트 운영 체제에 virtio 드라이버를 설치할 수 있습니다.

20.2.1.3. Windows 게스트에 virtio 드라이버 설치

Windows 게스트 운영 체제(OS)에 KVM virtio 드라이버를 설치하려면 VM(가상 머신)을 생성할 때 드라이버가 포함된 스토리지 장치를 추가하고 Windows 게스트 OS에 드라이버를 설치해야 합니다.

이 예에서는 그래픽 인터페이스를 사용하여 드라이버를 설치하는 방법을 보여줍니다. Microsoft Windows Installer(MSI) 명령줄 인터페이스도 사용할 수 있습니다.

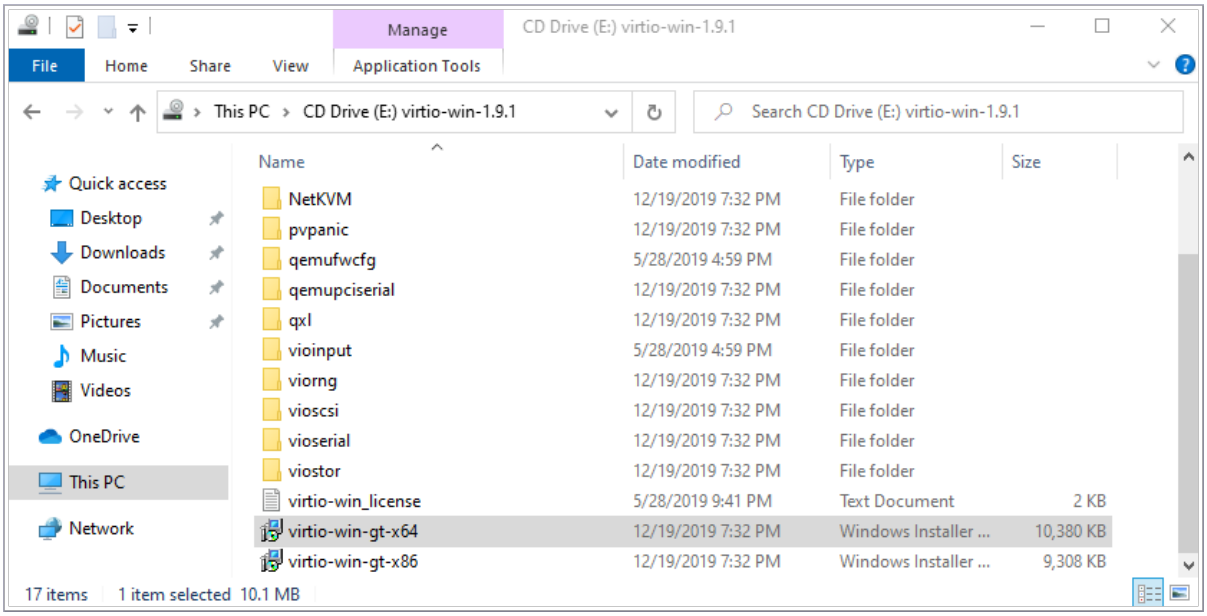
사전 요구 사항

- KVM virtio 드라이버가 있는 설치 미디어를 VM에 연결해야 합니다. 중간 준비에 대한 지침은 호스트 시스템의 virtio 드라이버 설치 미디어 준비를 참조하십시오.

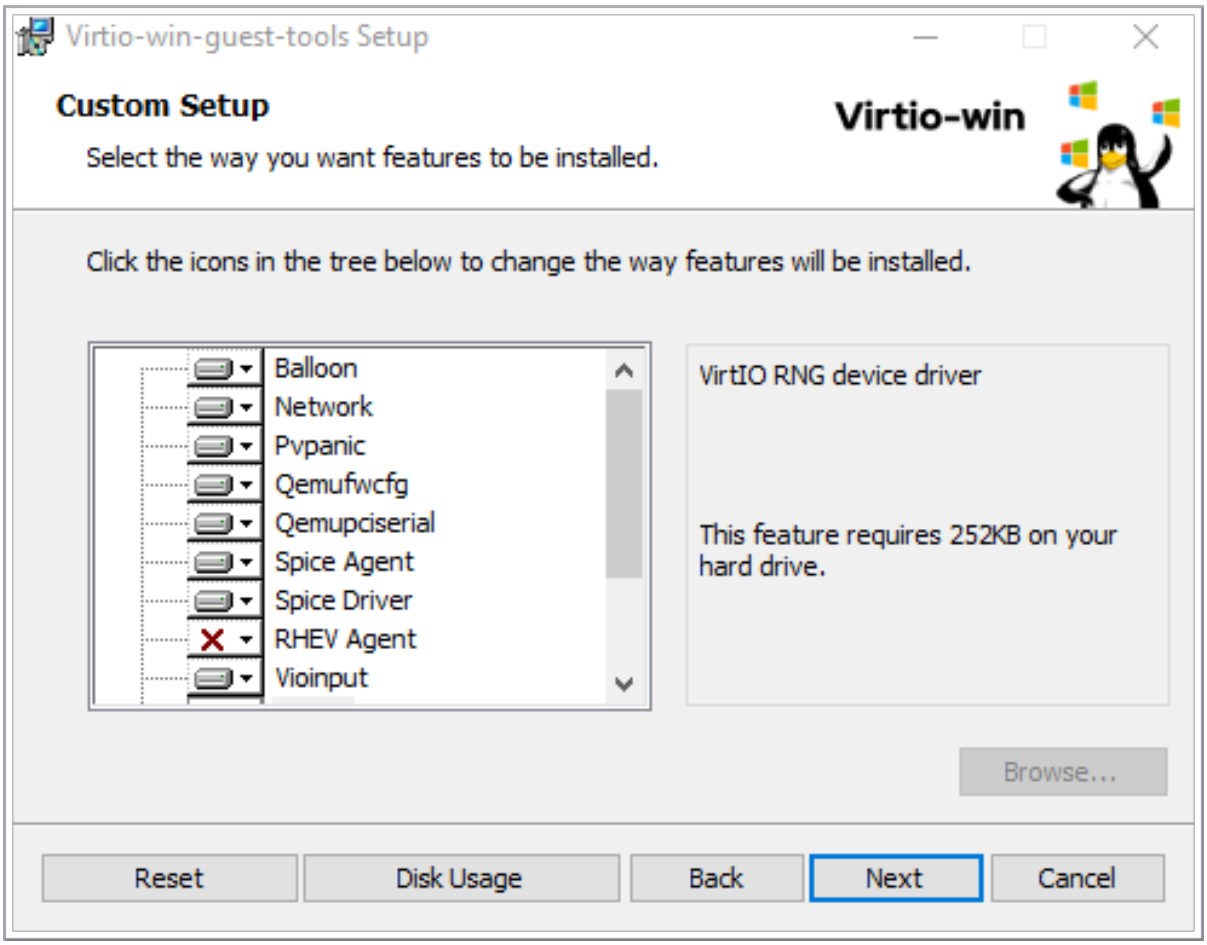
절차

1. Windows 게스트 OS에서 파일 탐색기 애플리케이션을 엽니다.
2. 이 PC 를 클릭합니다.
3. 장치 및 드라이브 창에서 virtio-win 미디어를 엽니다.
4. VM vCPU의 아키텍처에 따라 설치 프로그램 중 하나를 매체에서 실행합니다.
   - 32비트 vCPU를 사용하는 경우 virtio-win-gt-x86 설치 프로그램을 실행합니다.

- *64비트 vCPU를 사용하는 경우 virtio-win-gt-x64 설치 프로그램을 실행합니다.*



5. *열리는 Virtio-win-guest-tools 설정 마법사에서 사용자 지정 설정 단계에 도달할 때까지 표시된 지침을 따르십시오.*



6. *사용자 지정 설정 창에서 설치할 장치 드라이버를 선택합니다. 권장 드라이버 세트가 자동으로 선택되고 드라이버에 대한 설명이 목록 오른쪽에 표시됩니다.*

7. *다음 을 클릭한 다음 설치를 클릭합니다.*

8. *설치가 완료되면 완료를 클릭합니다.*

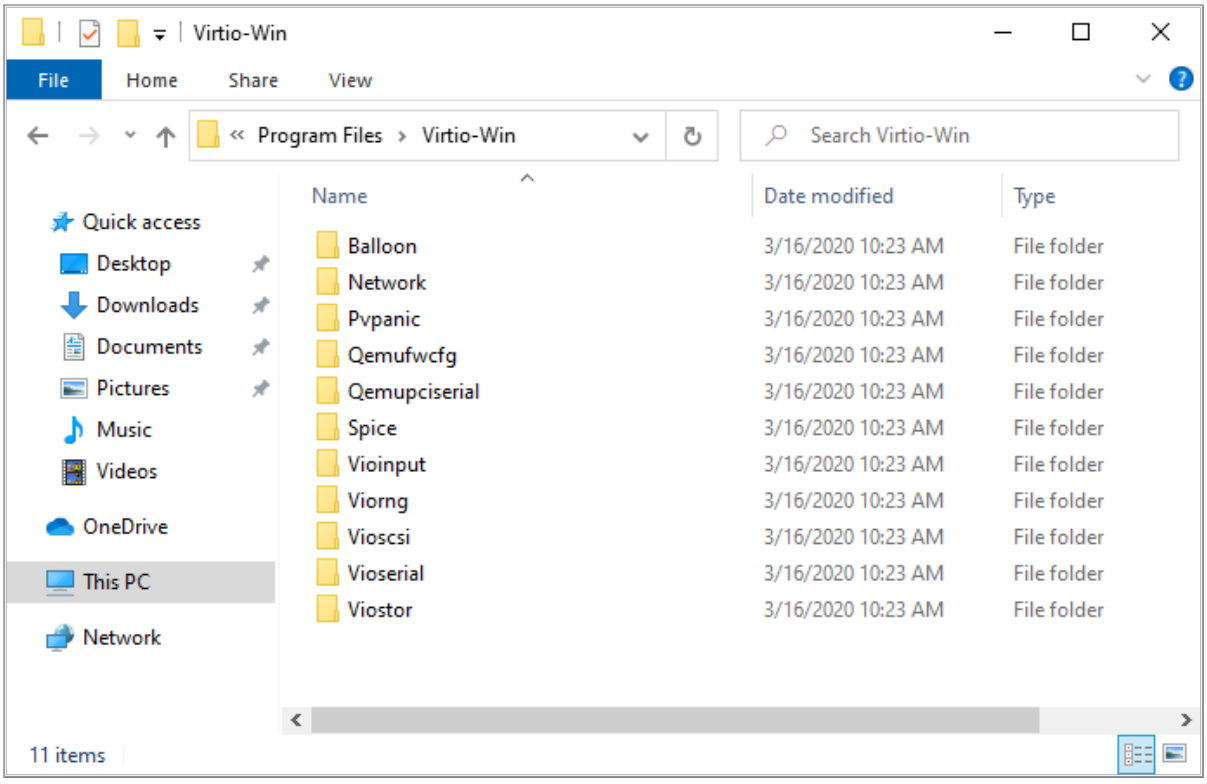
9. ***VM**을 재부팅하여 드라이버 설치를 완료합니다.*

검증

1. *이 **PC** 에서 시스템 디스크를 엽니다. 이는 일반적으로 **(C:)** 입니다.*

2. ***Program Files** 디렉토리에서 **Virtio-Win** 디렉터리를 엽니다.*

   ***Virtio-Win** 디렉터리가 있고 선택한 각 드라이버에 대한 하위 디렉터리가 포함된 경우 설치에 성공한 것입니다.*



다음 단계

- ***NetKVM** 드라이버를 설치하는 경우 **Windows** 게스트의 네트워킹 매개 변수를 구성해야 할 수도 있습니다.*

20.2.2. Hyper-V 서브스크립션 활성화

Hyper-V 서브스크립션에서는 KVM이 Microsoft Hyper-V 하이퍼바이저를 에뮬레이션할 수 있는 방법을 제공합니다. 이렇게 하면 Windows 가상 머신의 성능이 향상됩니다.

다음 섹션에서는 지원되는 Hyper-V 기능 및 이러한 기능을 활성화하는 방법에 대한 정보를 제공합니다.

20.2.2.1. Windows 가상 머신에서 Hyper-V 기능 활성화

Hyper-V 활동은 RHEL 9 호스트에서 실행 중인 Windows 가상 머신(VM)에서 더 나은 성능을 제공합니다. 이를 활성화하는 방법에 대한 지침은 다음을 참조하십시오.

절차

1. virsh edit 명령을 사용하여 VM의 XML 구성을 엽니다. 예를 들면 다음과 같습니다.

```
# virsh edit windows-vm
```

2. XML의 &lt ; features> 섹션에 다음 <hyperv & gt; 하위 섹션을 추가합니다.

```
<features>
 [...]
 <hyperv>
  <relaxed state='on'/>
  <vapic state='on'/>
  <spinlocks state='on' retries='8191'/>
  <vpindex state='on'/>
  <runtime state='on' />
  <synic state='on'/>
  <stimer state='on'>
   <direct state='on'/>
  </stimer>
  <frequencies state='on'/>
  <reset state='on'/>
  <relaxed state='on'/>
  <time state='on'/>
  <tlbflush state='on'/>
  <reenlightenment state='on'/>
  <stimer_direct state='on'/>
  <ipi state='on'/>
  <crash state='on'/>
  <evmcs state='on'/>
```

```
  </hyperv>
  [...]
</features>
```

XML에 이미 < hyperv> 하위 섹션이 포함된 경우 위에 표시된 대로 수정합니다.

3. 다음과 같이 구성의 clock 섹션을 변경합니다.

```
<clock offset='localtime'>
  ...
  <timer name='hypervclock' present='yes'/>
</clock>
```

4. XML 구성을 저장하고 종료합니다.

5. VM이 실행 중인 경우 다시 시작합니다.

검증

- virsh dumpxml 명령을 사용하여 실행 중인 VM의 XML 구성을 표시합니다. 다음 세그먼트가 포함된 경우 VM에서 Hyper-V enlightenments가 활성화됩니다.

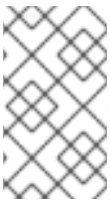
```
<hyperv>
  <relaxed state='on'/>
  <vapic state='on'/>
  <spinlocks state='on' retries='8191'/>
  <vpindex state='on'/>
  <runtime state='on' />
  <synic state='on'/>
  <stimer state='on'/>
  <frequencies state='on'/>
  <reset state='on'/>
  <relaxed state='on'/>
  <time state='on'/>
  <tlbflush state='on'/>
  <reenlightenment state='on'/>
  <stimer state='on'>
    <direct state='on'/>
  </stimer>
  <ipi state='on'/>
  <crash state='on'/>
  <evmcs state='on'/>
</hyperv>

<clock offset='localtime'>
```

```
  ...
  <timer name='hypervclock' present='yes'/>
</clock>
```

20.2.2.2. 구성 가능한 Hyper-V 개선 사항

특정 **Hyper-V** 기능을 구성하여 **Windows VM**을 최적화할 수 있습니다. 다음 표에서는 이러한 구성 가능한 **Hyper-V** 기능과 해당 값에 대한 정보를 제공합니다.

표 **20.1.** 구성 가능한 **Hyper-V** 기능

권장 사항	설명	값
크래시	VM 충돌 시 정보와 로그를 저장하는 데 사용할 수 있는 VM에 MSR을 제공합니다. QEMU 로그에서 사용 가능한 의 정보입니다.  **참고** hv_crash가 활성화되면 Windows 크래시 덤프가 생성되지 않습니다.	On, Off
evmcs	L0(KVM)과 L1(Hyper-V) 하이퍼바이저 간에 반가상화 프로토콜을 구현하므로 더 빠르게 L2가 하이퍼바이저로 종료될 수 있습니다.  **참고** 이 기능은 Intel 프로세서에서만 사용할 수 있습니다.	On, Off
빈도	Hyper-V 빈도 MSR(Hyper-V frequency Machine Specific Registers)을 활성화합니다.	On, Off
ipi	반가상화 프로세스 인터럽트(IPI) 지원을 활성화합니다.	On, Off
no-nonarch-coresharing	게스트 OS에 형제 SMT 스레드로 보고되지 않는 한 가상 프로세서가 실제 코어를 공유하지 않음을 알립니다. 이 정보는 Windows 및 Hyper-V 게스트가 SMT(동시 멀티스레딩) 관련 CPU 취약점을 적절하게 완화하기 위해 필요합니다.	On, off, auto

권장 사항	설명	값
reenlightenment	마이그레이션 중에 발생하는 타임 스탬프 카운터(TSC) 빈도 변경이 있을 때 알림을 받습니다. 또한 게스트가 새 빈도로 전환할 준비가 될 때까지 이전 빈도를 계속 사용할 수 있습니다.	On, Off
완화됨	VM이 과도하게 로드된 호스트에서 실행 중일 때 일반적으로 운영되는 Windows 온전성 점검을 비활성화합니다. 이는 Linux가 KVM에서 실행될 때 자동으로 활성화되는 Linux 커널 옵션 no_timer_check와 유사합니다.	On, Off
runtime	게스트 코드 실행 및 게스트 코드 대신 소비된 프로세서 시간을 설정합니다.	On, Off

권장 사항	설명	값
spinlocks	• VM의 운영 체제에서 사용하는 가상 프로세서에서 호출하는 가상 프로세서가 동일한 파티션 내의 다른 가상 프로세서에 의해 잠재적으로 보유되는 리소스를 획득하려고 시도한다는 사실을 Hyper-V에 알립니다.Used by a VM's operating system to notify Hyper-V that the calling virtual processor is attempting to acquire a resource that is potentially held by another virtual processor within the same partition. • Hyper-V에서 가상 머신의 운영 체제에 과도한 회전 상황을 나타내는 빈도를 설정하기 전에 가상 머신의 운영 체제를 나타내는 데 사용됩니다.Used by Hyper-V to indicate to the virtual machine's operating system the number of times a spinlock acquisition should be attempted before indicating an excessive spin situation to Hyper-V.	On, Off
stimer	가상 프로세서에 대해 합성 타이머를 활성화합니다. 특정 Windows 버전은 HPET를 사용할 수 없는 경우(또는 HPET를 사용할 수 없는 경우에도 RTC) 사용으로 되돌아오고 가상 CPU가 유휴 상태인 경우에도 상당한 CPU 소비가 발생할 수 있습니다.	On, Off
stimer-direct	만료 이벤트가 정상적인 인터럽트를 통해 전달되는 경우 합성 타이머를 활성화합니다.	On, off.
synic	stimer와 함께, 합성 타이머를 활성화합니다. Windows 8은 이 기능을 주기 모드에서 사용합니다.	On, Off

권장 사항	설명	값
time	VM에서 사용할 수 있는 다음 Hyper-V 관련 클럭 소스를 활성화합니다. ● MSR 기반 82 Hyper-V 클럭 소스 (HV_X64_MSR_TIME_REF_COUNT, 0x40000020) ● MSR (HV_X64_MSR_REFERENCE_TSC, 0x40000021)을 통해 활성화 된 TSC 83 페이지 참조	On, Off
tlbflush	가상 프로세서의 TLB를 플러시합니다.	On, Off
vapic	고성능 메모리 매핑된 Advanced Programmable Interrupt Controller(APIC) 레지스터에 가속화된 MSR 액세스를 제공하는 가상 APIC를 활성화합니다.	On, Off
vendor_id	Hyper-V 공급업체 ID를 설정합니다.	● On, Off ● ID 값 - 최대 12자의 문자열
vpindex	가상 프로세서 인덱스를 활성화합니다.	On, Off

20.2.3. NetKVM 드라이버 매개변수 구성

***NetKVM** 드라이버가 설치되면 환경에 더 잘 맞게 구성할 수 있습니다. 이 섹션에 나열된 매개 변수는 **Windows** 장치 관리자(**devmgmt.msc**)를 사용하여 구성할 수 있습니다.*

중요

*드라이버의 매개 변수를 수정하면 **Windows**에서 해당 드라이버를 다시 로드합니다. 이렇게 하면 기존 네트워크 활동이 중단됩니다.*

사전 요구 사항

- *NetKVM 드라이버가 가상 시스템에 설치되어 있습니다.*

  자세한 내용은 Windows 가상 머신 용 KVM 반가상화 드라이버 설치를 참조하십시오.

절차

1. *Windows 장치 관리자를 엽니다.*

   장치 관리자 열기에 대한 자세한 내용은 Windows 설명서를 참조하십시오.

2. *Red Hat VirtIO 이더넷 어댑터를 찾습니다.*

   a. *장치 관리자 창에서 네트워크 어댑터 옆에 있는 + 를 클릭합니다.*

   b. *네트워크 어댑터 목록에서 Red Hat VirtIO Ethernet Adapter 를 두 번 클릭합니다.*

      장치의 속성 창이 열립니다.

3. *장치 매개 변수를 확인합니다.*

   속성 창에서 고급 탭을 클릭합니다.In the Properties window, click the Advanced tab.

4. *장치 매개 변수를 수정합니다.*

   a. *수정할 매개변수를 클릭합니다.*

      해당 매개 변수의 옵션이 표시됩니다.

   b. *필요에 따라 옵션을 수정합니다.*

      NetKVM 매개변수 옵션에 대한 자세한 내용은 NetKVM 드라이버 매개 변수 를 참조하십시오.

c.

확인 을 클릭하여 변경 내용을 저장합니다.Click OK to save the changes.

20.2.4. NetKVM 드라이버 매개변수

다음 표에서는 구성 가능한 NetKVM 드라이버 로깅 매개변수에 대한 정보를 제공합니다.

표 20.2. 로깅 매개변수

매개변수	설명 2
Logging.Enable	로깅이 활성화되었는지 여부를 결정하는 부울 값입니다. 기본값은 Enabled입니다.
Logging.Level	로깅 수준을 정의하는 정수입니다. 정수가 증가하므로 로그의 세부 정보 표시를 수행합니다. ● 기본값은 0 (errors 만)입니다. ● 1~2, 구성 메시지를 추가합니다. ● 3.4 패킷 흐름 정보를 추가합니다. ● 5-6 인터럽트 및 DPC 레벨 추적 정보를 추가합니다.  **참고** 높은 로깅 수준은 가상 머신의 속도가 느려집니다.

다음 표에서는 구성 가능한 NetKVM 드라이버 초기 매개변수에 대한 정보를 제공합니다.

표 20.3. 초기 매개변수

매개변수	설명
Assign MAC	반가상화 NIC의 로컬 관리 MAC 주소를 정의하는 문자열입니다. 이 설정은 기본적으로 설정되지 않습니다.
Init.ConnectionRate(Mb)	초당 megabytesbits의 연결 속도를 나타내는 정수입니다. Windows 2008 이상의 기본값은 10G(10,000 Megabits per second)입니다.
Init.Do802.1PQ	Priority/VLAN 태그 채우기 및 제거 지원을 활성화하는 부울 값입니다. 기본값은 Enabled입니다.

매개변수	설명
Init.MTUSize	최대 전송 단위(MTU)를 정의하는 정수입니다. 기본값은 1500입니다. 500에서 65500 사이의 모든 값이 허용됩니다.
Init.MaxTxBuffers	할당될 TX 링 설명자 수를 나타내는 정수입니다. 기본값은 1024입니다. 유효한 값은 다음과 같습니다. 16, 32, 64, 128, 256, 512 및 1024.
Init.MaxRxBuffers	할당될 RX 링 설명자 수를 나타내는 정수입니다. 기본값은 256입니다. 유효한 값은 다음과 같습니다. 16, 32, 64, 128, 256, 512 및 1024.
Offload.Tx.Checksum	TX 체크섬 오프로드 모드를 지정합니다. Red Hat Enterprise Linux 9에서는 이 매개변수에 유효한 값은 다음과 같습니다. * IPv4와 IPv6 모두에 IP, TCP 및 UDP 체크섬 오프로드를 활성화하는 모든(기본값) * TCP/UDP(v4,v6) IPv4 및 IPv6 둘 다에 TCP 및 UDP 체크섬 오프로드 가능 * TCP/UDP(v4) - IPv4 전용 TCP 및 UDP 체크섬 오프로드 가능 * IPv4에서만 TCP 체크섬 오프로드를 활성화하는 TCP(v4)

20.2.5. Windows 가상 머신에서 백그라운드 프로세스 최적화

***Windows OS**를 실행하는 **VM**(가상 머신)의 성능을 최적화하기 위해 다양한 **Windows** 프로세스를 구성하거나 비활성화할 수 있습니다.*

주의

구성을 변경하는 경우 특정 프로세스가 예상대로 작동하지 않을 수 있습니다.

절차

다음 중 임의의 조합을 수행하여 Windows VM을 최적화할 수 있습니다.

- USB 또는 CD-ROM과 같은 미사용 장치를 제거하고 포트를 비활성화합니다.
- SuperFetch 및 Windows 검색과 같은 백그라운드 서비스를 비활성화합니다. 서비스 중지에 대한 자세한 내용은 시스템 서비스 비활성화 또는 중지를 참조하십시오.
- useplatformclock 을 비활성화합니다. 이렇게 하려면 다음 명령을 실행합니다.

```
# bcdedit /set useplatformclock No
```

- 예약된 디스크 조각 모음과 같은 불필요한 예약된 작업을 검토하고 비활성화합니다. 이 작업을 수행하는 방법에 대한 자세한 내용은 예약된 작업 비활성화 를 참조하십시오.
- 디스크가 암호화되지 않았는지 확인합니다.
- 서버 애플리케이션의 주기적 활동 감소. 각 타이머를 편집하여 이 작업을 수행할 수 있습니다. 자세한 내용은 Multimedia timers를 참조하십시오.
- VM에서 Server Manager 애플리케이션을 닫습니다.
- 안티바이러스 소프트웨어를 비활성화합니다. 안티바이러스를 비활성화하면 VM의 보안이 손상될 수 있습니다.

- 화면 보호기를 비활성화합니다.
- 사용하지 않을 때는 로그인 화면에 Windows OS를 유지합니다.

20.3. WINDOWS 가상 머신에서 표준 하드웨어 보안 활성화

Windows VM(가상 머신)을 보호하려면 Windows 장치의 표준 하드웨어 기능을 사용하여 기본 수준 보안을 활성화할 수 있습니다.

사전 요구 사항

- 최신 WHQL 인증 VirtIO 드라이버를 설치했는지 확인하십시오.
- VM의 펌웨어가 UEFI 부팅을 지원하는지 확인합니다.
- 호스트 시스템에 edk2-OVMF 패키지를 설치합니다.

```
# dnf install edk2-ovmf
```

- 호스트 시스템에 vTPM 패키지를 설치합니다.

```
# dnf install swtpm libtpms
```

- VM에서 Q35 시스템 아키텍처를 사용하고 있는지 확인합니다.
- Windows 설치 미디어가 있는지 확인합니다.

절차

1. VM의 XML 구성의 < devices > 섹션에 다음 매개 변수를 추가하여 TPM 2.0을 활성화합니다.

```
<devices>
[...]
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'/>
```

```
  </tpm>
[...]
</devices>
```

2. UEFI 모드에서 Windows를 설치합니다. 이를 수행하는 방법에 대한 자세한 내용은 SecureBoot 가상 머신 생성을 참조하십시오.

3. Windows VM에 VirtIO 드라이버를 설치합니다. 이를 수행하는 방법에 대한 자세한 내용은 Windows 게스트에 virtio 드라이버 설치를 참조하십시오.

4. UEFI에서 Secure Boot를 활성화합니다. 이 작업을 수행하는 방법에 대한 자세한 내용은 Secure Boot 를 참조하십시오.

검증

- Windows 머신의 장치 보안 페이지에 다음 메시지가 표시되는지 확인합니다.

  설정 > 업데이트 및 보안 > Windows 보안 > 장치 보안

  > **Your device meets the requirements for standard hardware security.**

20.4. WINDOWS 가상 머신에서 향상된 하드웨어 보안 활성화

Windows VM(Windows 가상 머신)을 보다 안전하게 보호하려면 HVCI(하이퍼바이저 보호 코드 무결성)라고도 하는 코드 무결성의 가상화 기반 보호를 활성화할 수 있습니다.

사전 요구 사항

- 표준 하드웨어 보안이 활성화되어 있는지 확인합니다. 자세한 내용은 Windows 가상 머신에서 표준 하드웨어 보안 활성화를 참조하십시오.

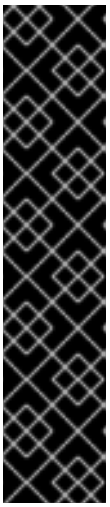
- Hyper-V 시행을 활성화했는지 확인합니다. 자세한 내용은 Enabling Hyper-V Enlightenments를 참조하십시오.

절차

1. Windows VM의 XML 구성을 엽니다. 다음 예제에서는 Example-L1 VM의 구성을 엽니다.

```
# virsh edit Example-L1
```

2. &lt ;cpu& gt; 섹션에서 CPU 모드를 지정하고 정책 플래그를 추가합니다.



중요

- Intel CPU의 경우 vmx 정책 플래그를 활성화합니다.
- AMD CPU의 경우 svm 정책 플래그를 활성화합니다.
- 사용자 정의 CPU를 지정하지 않으려면 < cpu mode >를 host-passthrough 로 설정할 수 있습니다.

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>Skylake-Client-IBRS</model>
  <topology sockets='1' dies='1' cores='4' threads='1'/>
  <feature policy='require' name='vmx'/>
</cpu>
```

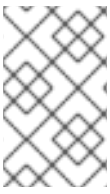
3. XML 구성을 저장하고 VM을 재부팅합니다.

4. VM 운영 체제에서 코어 격리 세부 정보 페이지로 이동합니다.

설정 > 업데이트 및 보안 > Windows 보안 > 장치 보안 > 핵심 격리 세부 정보

5. 스위치를 전환하여 메모리 무결성을 활성화합니다.

6. VM을 재부팅합니다.



참고

HVCI를 활성화하는 다른 방법은 관련 Microsoft 문서를 참조하십시오.

검증

- Windows VM의 장치 보안 페이지에 다음 메시지가 표시되는지 확인합니다.

  설정 > 업데이트 및 보안 > Windows 보안 > 장치 보안

  > Your device meets the requirements for enhanced hardware security.

- 또는 Windows VM의 시스템 정보를 확인합니다.

  a. 명령 프롬프트에서 msinfo32.exe 를 실행합니다.

  b. Credential Guard를 확인하고, Hypervisor 강제 코드 무결성 이 가상화 기반 보안 서비스 실행 아래에 나열되어 있는지 확인합니다.

20.5. 다음 단계

- Windows VM의 가상 머신 디스크 또는 기타 디스크 이미지에 액세스하는 데 유틸리티를 사용하려면 호스트 머신에 libguestfs-tools 및 libguestfs-winsupport 패키지를 설치합니다.

  ```
  $ sudo dnf install libguestfs-tools libguestfs-winsupport
  ```

- Windows VM의 가상 머신 디스크 또는 기타 디스크 이미지에 액세스하는 데 유틸리티를 사용하려면 호스트 머신에 guestfs-tools 및 guestfs-winsupport 패키지를 설치합니다.

  ```
  $ sudo dnf install guestfs-tools guestfs-winsupport
  ```

- RHEL 9 호스트와 해당 Windows VM 간에 파일을 공유하려면 virtiofs 또는 Samba 를 사용할 수 있습니다.

21장. 가상 머신 문제 진단

VM(가상 머신)으로 작업할 때 다양한 심각도 수준에 문제가 발생할 수 있습니다. 일부 문제는 빠르고 쉽게 해결할 수 있지만 다른 문제의 경우 VM 관련 데이터 및 로그를 캡처하여 문제를 보고하거나 진단해야 할 수 있습니다.

다음 섹션에서는 로그 생성 및 일부 일반적인 VM 문제 진단 및 이러한 문제 보고에 대한 자세한 정보를 제공합니다.

21.1. LIBVIRT 디버그 로그 생성

VM(가상 머신) 문제를 진단하려면 libvirt 디버그 로그를 생성하고 검토하는 것이 유용합니다. 디버그 로그를 연결하는 것도 VM 관련 문제를 해결하는 지원을 요청할 때 유용합니다.

다음 섹션에서는 디버그 로그 의 정의, 지속적인으로 설정하는 방법,런타임 중에 활성화, 문제 보고 시 이를 연결하는 방법에 대해 설명합니다.

21.1.1. libvirt 디버그 로그 이해

디버그 로그는 VM(가상 머신) 런타임 중에 발생하는 이벤트에 대한 데이터가 포함된 텍스트 파일입니다. 로그는 호스트 라이브러리 및 libvirt 데몬과 같은 기본 서버 측 기능에 대한 정보를 제공합니다. 로그 파일에는 실행 중인 모든 VM의 표준 오류 출력(stderr)도 포함되어 있습니다.

디버그 로깅은 기본적으로 활성화되어 있지 않으며 libvirt를 시작할 때 활성화해야 합니다. 단일 세션에 대한 로깅을 활성화하거나 영구적으로 활성화할 수 있습니다. 데몬 런타임 설정을 수정하여 libvirt 데몬 세션이 이미 실행 중인 경우 로깅을 활성화할 수도 있습니다.

libvirt 디버그 로그를 연결하는 것도 VM 문제에 대한 지원을 요청할 때 유용합니다.

21.1.2. libvirt 디버그 로그에 대한 영구 설정 활성화

libvirt를 시작할 때마다 libvirt 디버그 로깅이 자동으로 활성화되도록 구성할 수 있습니다. 기본적으로 virtqemud 는 RHEL 9의 기본 libvirt 데몬입니다. libvirt 구성을 영구적으로 변경하려면 /etc/libvirt 디렉터리에 있는 virtqemud.conf 파일을 편집해야 합니다.

참고

예를 들어 RHEL 8에서 업그레이드하는 경우와 같이 libvirtd 가 활성화된 libvirt 데몬인 경우도 있습니다. 이 경우 libvirtd.conf 파일을 대신 편집해야 합니다.

절차

1. 편집기에서 virtqemud.conf 파일을 엽니다.

2. 요구 사항에 따라 필터를 교체하거나 설정합니다.

표 21.1. 디버깅 필터 값

1	libvirt에서 생성한 모든 메시지를 기록합니다.
2	디버그되지 않은 모든 정보를 기록합니다.
3	모든 경고 및 오류 메시지를 기록합니다. 이는 기본값입니다.
4	오류 메시지만 기록합니다.

예 21.1. 로깅 필터를 위한 데몬 설정 샘플

다음 설정:

- 원격,util.json 의 모든 오류 및 경고 메시지를 기록하십시오.....
.....

- 이벤트 계층의 오류 메시지만 기록합니다.

- 필터링된 로그를 /var/log/libvirt/libvirt.log에 저장합니다.

```
log_filters="3:remote 4:event 3:util.json 3:rpc"
log_outputs="1:file:/var/log/libvirt/libvirt.log"
```

3. 저장하고 종료합니다.

4. **libvirt 데몬을 다시 시작합니다.**

```
$ systemctl restart virtqemud.service
```

21.1.3. 런타임 중 libvirt 디버그 로그 활성화

libvirt 데몬의 런타임 설정을 수정하여 디버그 로그를 활성화하고 출력 파일에 저장할 수 있습니다.

이 기능은 문제를 다시 시작하거나 동시에 실행 중인 마이그레이션 또는 백업과 같은 다른 프로세스가 있기 때문에 **libvirt** 데몬을 다시 시작할 수 없습니다. 구성 파일을 편집하거나 데몬을 다시 시작하지 않고 명령을 시도하려는 경우에도 런타임 설정을 수정할 수 있습니다.

사전 요구 사항

- **libvirt-admin** 패키지가 설치되어 있는지 확인합니다.

절차

1. 선택 사항: 활성 로그 필터 세트를 백업합니다.

```
# virt-admin -c virtqemud:///system daemon-log-filters >> virt-filters-backup
```

참고

로그를 생성한 후 복원할 수 있도록 활성 필터 세트를 백업하는 것이 좋습니다. 필터를 복원하지 않으면 시스템 성능에 영향을 줄 수 있는 메시지가 계속 기록됩니다.

2. **virt-admin** 유틸리티를 사용하여 디버깅을 활성화하고 요구 사항에 따라 필터를 설정합니다.

표 21.2. 디버깅 필터 값

1	libvirt에서 생성한 모든 메시지를 기록합니다.
2	디버그되지 않은 모든 정보를 기록합니다.
3	모든 경고 및 오류 메시지를 기록합니다. 이는 기본값입니다.

4	오류 메시지만 기록합니다.

예 21.2. 로깅 필터를 위한 virt-admin 설정 샘플

다음 명령:

- *원격,util.json 및 CHAP 계층의 모든 오류 및 경고 메시지를 기록합니다.*
- *이벤트 계층의 오류 메시지만 기록합니다.*

```
# virt-admin -c virtqemud:///system daemon-log-filters "3:remote 4:event 3:util.json 3:rpc"
```

3. *virt-admin 유틸리티를 사용하여 로그를 특정 파일 또는 디렉터리에 저장합니다.*

예를 들어 다음 명령은 로그 출력을 libvirt.log 파일에 /var/log/libvirt/ 디렉터리에 저장합니다.

```
# virt-admin -c virtqemud:///system daemon-log-outputs
"1:file:/var/log/libvirt/libvirt.log"
```

4. *선택 사항: 필터를 제거하여 모든 VM 관련 정보가 포함된 로그 파일을 생성할 수도 있습니다. 그러나 이 파일에는 libvirt의 모듈에서 생성된 다수의 중복 정보가 포함될 수 있으므로 권장되지 않습니다.*

- *virt-admin 유틸리티를 사용하여 빈 필터 세트를 지정합니다.*

```
# virt-admin -c virtqemud:///system daemon-log-filters
  Logging filters:
```

5. *선택 사항: 백업 파일을 사용하여 필터를 원래 상태로 복원합니다. 저장된 값을 사용하여 두 번째 단계를 수행하여 필터를 복원합니다.*

21.1.4. 요청을 지원하기 위해 libvirt 디버그 로그 연결

VM(가상 머신) 문제를 진단하고 해결하기 위해 추가 지원을 요청해야 할 수 있습니다. 지원 팀에 VM 관련 문제를 신속하게 해결하는 데 필요한 모든 정보에 액세스할 수 있도록 디버그 로그를 지원 요청에 연결하는 것이 좋습니다.

절차

- 문제를 보고하고 지원을 요청하려면 지원 케이스를 여십시오.
- 발생한 문제에 따라 보고서와 함께 다음 로그를 연결합니다.
  - libvirt 서비스에 문제가 있으면 호스트에서 /var/log/libvirt/libvirt.log 파일을 연결합니다.
  - 특정 VM에 문제가 있는 경우 해당 로그 파일을 연결합니다.

    예를 들어 testguest1 VM의 경우 /var/log/libvirt/qemu/ testguest1.log 에 있는 testguest1.log 파일을 연결합니다.

추가 리소스

- Red Hat 지원에 로그 파일을 제공하는 방법은 무엇입니까?

21.2. 가상 머신 코어 덤프

VM(가상 머신)이 충돌하거나 오작동한 이유를 분석하기 위해 나중에 분석 및 진단을 위해 VM 코어를 디스크의 파일로 덤프하면 됩니다.

이 섹션에서는 코어 덤프에 대한 간략한 소개 를 제공하고 VM 코어를 특정 파일에 덤프 하는 방법을 설명합니다.

21.2.1. 가상 머신 코어 덤프 작동 방식

VM(가상 머신)에는 정확하고 효율적으로 작동하려면 많은 실행 중인 프로세스가 필요합니다. 경우에 따라 VM을 사용하는 동안 VM이 예기치 않게 종료되거나 오작동이 중단될 수 있습니다. VM을 다시 시작하면 데이터가 재설정되거나 손실될 수 있으므로 VM이 충돌한 정확한 문제를 진단하기가 어렵습니다.

이러한 경우 VM을 재부팅하기 전에 virsh dump 유틸리티를 사용하여 VM의 코어를 파일에 저장(또는 덤프)할 수 있습니다. 코어 덤프 파일에는 VM에 대한 자세한 정보가 포함된 VM의 원시 물리 메모리 이미지가 포함되어 있습니다. 이 정보는 수동으로 또는 crash 유틸리티와 같은 툴을 사용하여 VM 문제를 진단하는 데 사용할 수 있습니다.

추가 리소스

- 충돌 도움말 페이지
- 크래시 Github 리포지토리

21.2.2. 가상 머신 코어 덤프 파일 생성

VM(가상 머신) 코어 덤프에는 언제든지 VM 상태에 대한 자세한 정보가 포함되어 있습니다. VM 스냅샷과 유사한 이 정보는 VM의 장애 발생 또는 갑자기 종료되는 경우 문제를 감지하는 데 도움이 될 수 있습니다.

사전 요구 사항

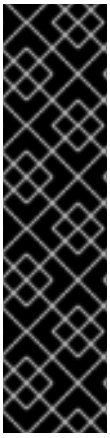
- 파일을 저장할 충분한 디스크 공간이 있는지 확인합니다. VM에 사용되는 공간은 VM에 할당된 RAM 용량에 따라 다릅니다.

절차

- virsh dump 유틸리티를 사용합니다.

  예를 들어 다음 명령은 lander1 VM의 코어, 메모리 및 CPU 공통 레지스터 파일을 /core/file 디렉터리의 gargantua.file 에 덤프합니다.

```
# virsh dump lander1 /core/file/gargantua.file --memory-only
Domain 'lander1' dumped to /core/file/gargantua.file
```



중요

crash 유틸리티는 더 이상 **virsh dump** 명령의 기본 파일 형식을 지원하지 않습니다. 크래시 를 사용하여 코어 덤프 파일을 분석하려면 **--memory-only** 옵션을 사용하여 파일을 생성해야 합니다.

또한 **Red Hat** 지원 사례에 첨부하려면 코어 덤프 파일을 생성할 때 **--memory-only** 옵션을 사용해야 합니다.

문제 해결

virsh dump 명령이 시스템 오류에서 교착 상태에 있는 경우 코어 덤프 파일에 대해 충분한 메모리를 할당했는지 확인하십시오. 이렇게 하려면 다음 **crashkernel** 옵션 값을 사용합니다. 또는 코어 덤프 메모리를 자동으로 할당하는 **crashkernel** 을 사용하지 마십시오.

```
crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M
```

추가 리소스

- **virsh dump --help** 명령
- **virsh man** 페이지
- 지원 케이스 만들기

21.3. BACKTRACING 가상 머신 프로세스

VM(가상 머신) 오작동과 관련된 프로세스가 프로세스 식별자(PID)와 함께 **gstack** 명령을 사용하여 오작동 프로세스의 실행 스택 추적을 생성할 수 있습니다. 프로세스가 스레드 그룹의 일부이면 모든 스레드도 추적됩니다.

사전 요구 사항

- **GDB** 패키지가 설치되어 있는지 확인합니다.

  GDB 및 사용 가능한 구성 요소 설치에 대한 자세한 내용은 **GNU Debugger** 설치를 참조하십시오.

- *trace할 프로세스의 PID를 알고 있는지 확인합니다.*

  pgrep 명령 뒤에 프로세스 이름을 사용하여 PID를 찾을 수 있습니다. 예를 들면 다음과 같습니다.

```
# pgrep libvirt
22014
22025
```

절차

- *gstack 유틸리티와 backtrace하려는 프로세스의 PID를 사용합니다.*

  예를 들어 다음 명령은 libvirt 프로세스를 PID 22014로 역추적합니다.

```
# gstack 22014
Thread 3 (Thread 0x7f33edaf7700 (LWP 22017)):
#0  0x00007f33f81aef21 in poll () from /lib64/libc.so.6
#1  0x00007f33f89059b6 in g_main_context_iterate.isra () from /lib64/libglib-2.0.so.0
#2  0x00007f33f8905d72 in g_main_loop_run () from /lib64/libglib-2.0.so.0
...
```

추가 리소스

- *gstack 도움말 페이지*
- *GNU Debugger (GDB)*

가상 머신 문제 보고 및 로그 제공을 위한 추가 리소스

추가 도움말 및 지원을 요청하려면 다음을 수행할 수 있습니다.

- *redhat-support-tool 명령줄 옵션, Red Hat Portal UI 또는 FTP를 사용하여 여러 가지 다른 방법을 사용하여 서비스 요청을 늘립니다.*
  - *문제를 보고하고 지원을 요청하려면 지원 케이스 열기를 참조하십시오.*

- *서비스 요청을 제출할 때 **SOS Report** 및 로그 파일을 업로드합니다.*

  *이를 통해 **Red Hat** 지원 엔지니어는 참조에 필요한 모든 진단 정보를 얻을 수 있습니다.*

  - ***SOS** 보고서에 대한 자세한 내용은 **What is an SOS Report and How to create one in Red Hat Enterprise Linux?***

  - *로그 파일을 연결하는 방법에 대한 자세한 내용은 **Red Hat** 지원에 파일을 제공하는 방법을참조하십시오.*

22장. RHEL 9 가상화의 기능 지원 및 제한 사항

이 문서에서는 Red Hat Enterprise Linux 9(RHEL 9) 가상화의 기능 지원 및 제한 사항에 대한 정보를 제공합니다.

22.1. RHEL 가상화 지원의 작동 방식

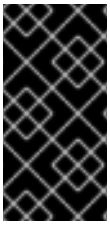
일련의 지원 제한 사항은 RHEL 9(Red Hat Enterprise Linux 9)의 가상화에 적용됩니다. 즉, RHEL 9에서 가상 머신을 사용할 때 특정 기능을 사용하거나 할당된 리소스를 초과할 때 특정 서브스크립션 계획이 없는 경우 Red Hat은 이러한 게스트를 지원하지 않습니다.

RHEL 9 가상화의 권장 기능에 나열된 기능은 Red Hat이 RHEL 9 시스템의 KVM 하이퍼바이저와 함께 작동하도록 테스트 및 인증되었습니다. 따라서 RHEL 9의 가상화에서 완전 지원 및 사용하는 것이 좋습니다.

RHEL 9 가상화의 Unsupported features에 나열된 기능은 작동할 수 있지만 RHEL 9에서는 지원되지 않으며 지원되지 않습니다. 따라서 Red Hat은 KVM에서 RHEL 9에서 이러한 기능을 사용하지 않는 것이 좋습니다.

RHEL 9 가상화의 리소스 할당 제한 은 RHEL 9의 KVM 게스트에서 지원되는 최대 특정 리소스 양을 나열합니다. 이러한 제한을 초과하는 게스트는 Red Hat에서 지원되지 않습니다.

별도로 명시하지 않는 한, RHEL 9 가상화 설명서에 사용된 모든 기능 및 솔루션이 지원됩니다. 그러나 이들 중 일부는 완전히 테스트되지 않았으므로 최적화되지 않을 수 있습니다.



중요

이러한 제한 사항 중 다수는 Red Hat에서 제공하는 다른 가상화 솔루션에는 적용되지 않습니다(예: OpenShift Virtualization 또는 RHOSP(Red Hat OpenStack Platform).

22.2. RHEL 9 가상화 권장 기능

다음 기능은 RHEL 9(Red Hat Enterprise Linux 9)에 포함된 KVM 하이퍼바이저와 함께 사용하는 것이 좋습니다.

호스트 시스템 아키텍처

KVM이 있는 RHEL 9는 다음 호스트 아키텍처에서만 지원됩니다.

- AMD64 및 Intel 64
- IBM Z - IBM z13 시스템 이상

다른 하드웨어 아키텍처는 RHEL 9를 KVM 가상화 호스트로 사용할 수 없으며 Red Hat은 이러한 목표를 달성하는 것을 강력히 권장합니다. 특히 기술 프리뷰로만 제공되는 64비트 ARM 아키텍처(ARM 64)가 포함됩니다.

게스트 운영 체제

Red Hat은 다음 OS(운영 체제)를 사용하는 KVM 가상 머신을 지원합니다.

- Red Hat Enterprise Linux 7 이상
- Microsoft Windows 10 이상
- Microsoft Windows Server 2016 이상

그러나 기본적으로 게스트 OS는 호스트와 동일한 서브스크립션을 사용하지 않습니다. 따라서 게스트 OS가 제대로 작동하려면 별도의 라이센스 또는 서브스크립션을 활성화해야 합니다.

머신 유형

VM이 호스트 아키텍처와 호환되고 게스트 OS가 최적으로 실행되도록 하려면 VM에서 적절한 시스템 유형을 사용해야 합니다.

명령줄을 사용하여 VM을 생성할 때 virt-install 유틸리티에서는 시스템 유형을 설정하는 여러 방법을 제공합니다.

- --os-variant 옵션을 사용하면 virt-install 에서 호스트 CPU에 권장되는 머신 유형을 자동으로 선택하고 게스트 OS에서 지원합니다.

- --os-variant 을 사용하지 않거나 다른 머신 유형이 필요한 경우 --machine 옵션을 사용하여 머신 유형을 명시적으로 지정합니다.

- 호스트에서 지원되지 않거나 호환되지 않는 --machine 값을 지정하면 virt-install 이 실패하고 오류 메시지가 표시됩니다.

지원되는 아키텍처에서 KVM 가상 머신에 권장되는 머신 유형과 --machine 옵션에 해당하는 값은 다음과 같습니다. Y 는 RHEL 9의 최신 마이너 버전을 나타냅니다.

- Intel 64 및 AMD64 (x86_64): pc-q35-rhel9.Y.0 → --machine=q35

- IBM Z (s390x): s390-ccw-virtio-rhel9.Y.0 → --machine=s390-ccw-virtio

기존 VM의 머신 유형을 가져오려면 다음을 수행합니다.

```
# virsh dumpxml VM-name | grep machine=
```

호스트에서 지원되는 전체 머신 유형 목록을 보려면 다음을 수행합니다.

```
# /usr/libexec/qemu-kvm -M help
```

추가 리소스

- RHEL 9 가상화에서 지원되지 않는 기능

- RHEL 9 가상화의 리소스 할당 제한

22.3. RHEL 9 가상화에서 지원되지 않는 기능

RHEL 9(Red Hat Enterprise Linux 9)에 포함된 KVM 하이퍼바이저에서는 다음 기능을 지원하지 않습니다.

중요

이러한 제한 사항 중 다수는 Red Hat에서 제공하는 다른 가상화 솔루션에는 적용되지 않을 수 있습니다(예: OpenShift Virtualization 또는 RHOSP(Red Hat OpenStack Platform).

다른 가상화 솔루션에서 지원하는 기능은 다음 단락에서 설명합니다.

호스트 시스템 아키텍처

KVM이 있는 RHEL 9는 RHEL 9 가상화의 권장 기능에 나열되지 않은 호스트 아키텍처에서 지원되지 않습니다.

특히 64비트 ARM 아키텍처(ARM 64)는 RHEL 9에서 KVM 가상화를 위한 기술 프리뷰로만 제공되므로 프로덕션 환경에서의 사용을 권장하지 않습니다.

게스트 운영 체제

RHEL 9 호스트에서 다음 게스트 운영 체제(OS)를 사용하는 KVM 가상 머신(VM)은 지원되지 않습니다.

- Microsoft Windows 8.1 및 이전 버전
- Microsoft Windows Server 2012 및 이전 버전
- macOS
- Solaris for x86 시스템
- 2009년 이전에 출시된 OS

RHEL 호스트에서 지원되는 게스트 OS 목록은 Certified guest operating systems for Red Hat Enterprise Linux with KVM 에서 참조하십시오.

기타 해결책:

- *Red Hat에서 제공하는 다른 가상화 솔루션에서 지원하는 게스트 OS 목록은 Certified Guest Operating Systems in Red Hat OpenStack Platform, Red Hat Virtualization 및 OpenShift Virtualization 에서 참조하십시오.*

컨테이너에서 VM 생성

Red Hat은 RHEL 9 하이퍼바이저(예: QEMU 에뮬레이터 또는 libvirt 패키지)의 요소를 포함하는 모든 유형의 컨테이너에서 KVM 가상 머신 생성을 지원하지 않습니다.

기타 해결책:

- *컨테이너에 VM을 생성하려면 OpenShift Virtualization 오퍼링을 사용하는 것이 좋습니다.*

문서화되지 않은 virsh 명령 및 옵션

Red Hat 설명서에서 명시적으로 권장되지 않는 virsh 명령과 옵션은 제대로 작동하지 않을 수 있으며, 프로덕션 환경에서 사용하지 않는 것이 좋습니다.

QEMU 명령줄

QEMU는 RHEL 9의 가상화 아키텍처의 필수 구성 요소이지만 수동으로 관리하기가 어렵고 부적절한 QEMU 구성으로 인해 보안 취약점이 발생할 수 있습니다. 따라서 qemu- 명령줄 유틸리티(예: qemu-kvm) 사용은 Red Hat에서 지원되지 않습니다.*

대신, 모범 사례에 따라 QEMU를 오케스트레이션하므로 virsh,virt-install, virt-xml 과 같은 libvirt 유틸리티를 사용하십시오.

vCPU 핫 플러그 해제

실행 중인 VM에서 가상 CPU(vCPU)를 vCPU 핫 플러그라고도 하는 가상 CPU를 제거하는 것은 RHEL 9에서 지원되지 않습니다.

메모리 핫 플러그 연결

실행 중인 VM에 연결된 메모리 장치(메모리 핫 플러그라고도 함)는 RHEL 9에서 지원되지 않습니다.

QEMU 측 I/O 제한

virsh blkdeviotune 유틸리티를 사용하여 QEMU 측 I/O 제한이라고도 하는 가상 디스크의 작업에 대한 최대 입력 및 출력 수준을 RHEL 9에서는 지원되지 않습니다.

RHEL 9에서 I/O 제한을 설정하려면 virsh blkiotune 을 사용합니다. 이를 libvirt 측 I/O 제한이라고도 합니다. 자세한 내용은 가상 머신의 디스크 I/O 제한을 참조하십시오.

기타 해결책:

- *RHOSP에서 QEMU 측 I/O 제한도 지원됩니다. 자세한 내용은 RHOSP 스토리지 가이드 의 디스크 리소스 제한 설정 및 서비스 품질 사양 사용 섹션을 참조하십시오.*
- *또한 OpenShift Virtualizaton에서는 QEMU 측 I/O 제한도 지원합니다.*

스토리지 실시간 마이그레이션

RHEL 9에서는 호스트 간에 실행 중인 VM의 디스크 이미지 마이그레이션이 지원되지 않습니다.

기타 해결책:

- *RHOSP에서는 스토리지 실시간 마이그레이션도 지원되지만 몇 가지 제한 사항이 있습니다. 자세한 내용은 볼륨 마이그레이션을 참조하십시오.*
- *OpenShift Virtualization을 사용할 때 실시간 마이그레이션 VM 스토리지도 가능합니다. 자세한 내용은 가상 머신 실시간 마이그레이션 을 참조하십시오.*

실시간 스냅샷

RHEL 9에서는 실행 중인 VM의 스냅샷을 라이브 스냅샷이라고도 하는 생성 또는 로드는 지원되지 않습니다.

또한 RHEL 9에서는 라이브 이외의 VM 스냅샷이 더 이상 사용되지 않습니다. 따라서 종료 VM의 스냅샷을 생성하거나 로드하는 것은 지원되지만 Red Hat은 이를 사용하지 않는 것이 좋습니다.

기타 해결책:

- RHOSP에서는 실시간 스냅샷도 지원합니다. 자세한 내용은 가상 머신을 오버클라우드로 가져오기를 참조하십시오.

vhost 데이터 경로 4.6.1

RHEL 9 호스트에서 virtio 장치에 대해 vHost Data Pathpwd(vDPA)를 구성할 수 있지만 Red Hat은 현재 이 기능을 지원하지 않으며 프로덕션 환경에서의 사용을 강력하게 권장하지 않습니다.

vhost-user

RHEL 9는 사용자 공간 vHost 인터페이스의 구현을 지원하지 않습니다.

기타 해결책:

- vhost-user 는 RHOSP에서 지원되지만 virtio-net 인터페이스에서만 지원됩니다. 자세한 내용은 virtio-net 구현 및 vhost 사용자 포트를 참조하십시오.
- OpenShift Virtualization에서는 vhost-user 도 지원합니다.

S3 및 S4 시스템 전원 상태

VM을 Suspend to RAM (S3)으로 일시 중지하거나 디스크에 대한 Suspend to disk (S4) 시스템 전원 상태는 지원되지 않습니다. 이러한 기능은 기본적으로 비활성화되어 있으며 이를 활성화하면 Red Hat에서 VM을 지원할 수 없습니다.

S3 및 S4 상태는 현재 Red Hat에서 제공하는 다른 가상화 솔루션에서도 지원되지 않습니다.

다중 경로 vDisk의 S3-PR

다중 경로 vDisk의 SCSI3 S3-PR(S3-PR)은 RHEL 9에서 지원되지 않습니다. 따라서 RHEL 9에서는 Windows 클러스터가 지원되지 않습니다.

virtio-crypto

RHEL 9에서 virtio-crypto 장치 사용은 지원되지 않으므로 사용하지 않는 것이 좋습니다.

virtio-crypto 장치는 Red Hat에서 제공하는 다른 가상화 솔루션에서도 지원되지 않습니다.

증분 라이브 백업

증분 라이브 백업이라고도 하는 VM 백업만 VM 백업을 구성하는 것은 RHEL 9에서 지원되지 않으며 Red Hat은 그 용도가 좋지 않습니다.

net_failover

net_failover 드라이버를 사용하여 자동화된 네트워크 장치 장애 조치 메커니즘을 설정하는 것은 RHEL 9에서 지원되지 않습니다.

net_failover 는 현재 Red Hat에서 제공하는 다른 가상화 솔루션에서도 지원되지 않습니다.

다중 FD 마이그레이션

다중 FD 마이그레이션이라고도 하는 FD(여러 파일 설명자)를 사용하여 VM 마이그레이션은 RHEL 9에서 지원되지 않습니다.

다중 FD 마이그레이션은 현재 Red Hat에서 제공하는 다른 가상화 솔루션에서도 지원되지 않습니다.

NVMe 장치

PCI-passthrough를 사용하여 PKCS 장치로 VM에 NVMe(Non-volatile Memory Express) 장치를 연결하는 것은 지원되지 않습니다.

NVMe 장치를 VM에 연결하는 것도 현재 Red Hat에서 제공하는 다른 가상화 솔루션에서는 지원되지 않습니다.

TCG

QEMU 및 libvirt에는 QEMU Tiny Code Generator(TCG)를 사용하는 동적 번역 모드가 포함됩니다. 이 모드에서는 하드웨어 가상화 지원이 필요하지 않습니다. 그러나 TCG는 Red Hat에서 지원하지 않습니다.

TCG 기반 게스트는 예를 들어 virsh dumpxml 명령을 사용하여 XML 구성을 검사하여 인식할 수 있습니다.

- TCG 게스트의 구성 파일에는 다음 행이 포함되어 있습니다.

```
<domain type='qemu'>
```

- KVM 게스트의 구성 파일에는 다음 행이 포함되어 있습니다.

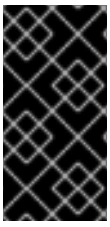
```
<domain type='kvm'>
```

추가 리소스

- RHEL 9 가상화 권장 기능
- RHEL 9 가상화의 리소스 할당 제한

22.4. RHEL 9 가상화의 리소스 할당 제한

다음 제한은 RHEL 9(Red Hat Enterprise Linux 9) 호스트의 단일 KVM 가상 머신(VM)에 할당할 수 있는 가상화된 리소스에 적용됩니다.



중요

이러한 제한 사항 중 다수는 Red Hat에서 제공하는 다른 가상화 솔루션에는 적용되지 않습니다(예: OpenShift Virtualization 또는 RHOSP(Red Hat OpenStack Platform).

VM당 최대 vCPU

RHEL 9는 단일 VM에 할당된 최대 384 개의 vCPU를 지원합니다.

VM당 PCI 장치

RHEL 9는 VM 버스당 32 PCI 장치 슬롯과 장치 슬롯당 8 PCI 기능을 지원합니다. 그러면 VM에서 다중 기능 기능이 활성화되어 있고 PCI 브리지가 사용되지 않는 경우 버스당 최대 256 PCI 기능을 사용할 수 있습니다.

각 PCI 브릿지는 새 버스를 추가하여 잠재적으로 또 다른 256 장치 주소를 활성화합니다. 그러나 일부 버스는 모든 256 장치 주소를 사용할 수있는 것은 아닙니다. 예를 들어 루트 버스에는 슬롯을 사용하는 여러 개의 내장 장치가 있습니다.

가상화된 IDE 장치

KVM은 VM당 최대 4 개의 가상화된 IDE 장치로 제한됩니다.

22.5. IBM Z의 가상화가 AMD64 및 INTEL 64와 다른 방법

IBM Z 시스템의 RHEL 9의 KVM 가상화는 AMD64 및 Intel 64 시스템의 KVM과 다릅니다.

PCI 및 USB 장치

IBM Z에서 가상 PCI 및 USB 장치는 지원되지 않습니다. 즉 virtio-*-pci 장치는 지원되지 않으며 virtio-*-ccw 장치를 대신 사용해야 합니다. 예를 들어 virtio-net-pci 대신 virtio-net-ccw 를 사용합니다.

PCI 패스스루라고도 하는 PCI 장치의 직접 연결이 지원됩니다.

지원되는 게스트 OS

Red Hat은 RHEL 7, 8 또는 9를 게스트 운영 체제로 사용하는 경우에만 IBM Z에서 호스팅되는 VM을 지원합니다.

장치 부팅 순서

IBM Z는 < boot dev='device'> XML 구성 요소를 지원하지 않습니다. 장치 부팅 순서를 정의하려면 XML의 < devices > 섹션에 있는 < boot order='항목' > 요소를 사용합니다. 예를 들면 다음과 같습니다.

```
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2'/>
  <source file='/path/to/qcow2'/>
  <target dev='vda' bus='virtio'/>
  <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000'/>
  <boot order='2'>
</disk>
```

참고

부팅 순서 관리에 **<boot order='번호' >**를 사용하는 것이 **AMD64** 및 **Intel 64** 호스트에서도 선호됩니다.

메모리 핫 플러그

실행 중인 **VM**에 메모리를 추가하는 것은 **IBM Z**에서 사용할 수 없습니다. 실행 중인 **VM**에서메모리를제거하는 것은 **AMD64** 및 **Intel 64**에서는 **IBM Z**에서도 사용할 수 없습니다.

NUMA 토폴로지

CPU의 **NUMA(Non-Uniform Memory Access)** 토폴로지는 **IBM Z**의 **libvirt** 에서 지원되지 않습니다. 따라서 **NUMA**를 사용한 **vCPU** 성능을 조정할 수 없습니다.

vfio-ap

IBM Z 호스트의 **VM**은 다른 아키텍처에서 지원되지 않는 **vfio-ap** 암호화 장치 패스스루를 사용할 수 있습니다.

SMBIOS

IBM Z에서는 **SMBIOS** 구성을 사용할 수 없습니다.

워치독 장치

IBM Z 호스트의 **VM**에서 워치독 장치를 사용하는 경우 **diag288** 모델을 사용합니다. 예를 들면 다음과 같습니다.

```
<devices>
  <watchdog model='diag288' action='poweroff'/>
</devices>
```

kvm-clock

kvm-clock 서비스는 **AMD64** 및 **Intel 64** 시스템에 한정되며 **IBM Z**에서 **VM** 시간 관리를 위해 구성할 필요가 없습니다.

v2v 및 **p2v**

virt-v2v 및 **virt-p2v** 유틸리티는 **AMD64** 및 **Intel 64** 아키텍처에서만 지원되며 **IBM Z**에서는 제공되지 않습니다.

마이그레이션

(예: **IBM z14**에서 **z15**로) 이후 호스트 모델로 마이그레이션하거나 하이퍼바이저를 업데이트하려면 호스트 모델 **CPU** 모드를 사용합니다. 일반적으로 마이그레이션되지 않으므로 **host-passthrough**

및 최대 CPU 모드는 권장되지 않습니다.

사용자 정의 CPU 모드에서 명시적 CPU 모델을 지정하려면 다음 지침을 따르십시오.

- *-base 로 끝나는 CPU 모델을 사용하지 마십시오.*
- *qemu,max 또는 host CPU 모델을 사용하지 마십시오.*

이전 호스트 모델(예: z15에서 z14)로 마이그레이션하거나 이전 버전의 QEMU, KVM 또는 RHEL 커널로 마이그레이션하려면 끝에 -base 없이 사용 가능한 가장 오래된 호스트 모델의 CPU 유형을 사용합니다.

- *소스 호스트와 대상 호스트가 모두 실행 중인 경우 대상 호스트에서 virsh cpu-baseline 명령을 사용하여 적절한 CPU 모델을 가져올 수 있습니다.*

추가 리소스

- *아키텍처 전체에서 가상화 기능 지원 개요*

22.6. ARM 64의 가상화가 AMD64 및 INTEL 64와 다른 방법

ARM 64 시스템의 RHEL 9의 KVM 가상화는 여러 측면에서 AMD64 및 Intel 64 시스템의 KVM과 다릅니다. 여기에는 다음이 포함되지만 이에 국한되지는 않습니다.

지원

ARM 64의 가상화는 RHEL 9에서 기술 프리뷰 로만 제공되므로 지원되지 않습니다.

게스트 운영 체제

현재 ARM 64 VM(가상 머신)에서 작동하는 유일한 게스트 운영 체제는 RHEL 9입니다.

웹 콘솔 관리

RHEL 9 웹 콘솔에서 VM 관리 의 일부 기능은 ARM 64 하드웨어에서 제대로 작동하지 않을 수 있습니다.

vCPU 핫 플러그 및 핫 언플러그

실행 중인 VM에 가상 CPU(vCPU)를 연결하면 ARM 64 호스트에서는 vCPU 핫 플러그라고도 합니다. 또한 AMD64 및 Intel 64 호스트와 마찬가지로 실행 중인 VM(vCPU 핫 플러그)에서 vCPU를 제거하는 것은 ARM 64에서 지원되지 않습니다.

SecureBoot

SecureBoot 기능은 ARM 64 시스템에서 사용할 수 없습니다.

PXE

PXE(Preboot Execution Environment)에서 부팅하는 것은 virtio-net-pci 네트워크 인터페이스 컨트롤러(NIC)에서만 가능합니다. 또한 가상 머신 UEFI 플랫폼 펌웨어(edk2-aarch64 패키지와 함께 설치됨)의 기본 제공 VirtioNetDxe 드라이버를 PXE 부팅에 사용해야 합니다. iPXE 옵션 Rom은 지원되지 않습니다.

장치 메모리

듀얼 인라인 메모리 모듈(DIMM) 및 NVDIMM(Non-volatile DIMM)과 같은 장치 메모리 기능은 ARM 64에서 작동하지 않습니다.

pvpanic

pvpanic 장치는 현재 ARM 64에서 작동하지 않습니다. VM이 부팅되지 않을 수 있으므로 ARM 64에서 게스트 XML 구성의 < devices > 섹션에서 < panic > 요소를 제거해야 합니다.

OVMF

ARM 64 호스트의 VM은 edk2-ovmf 패키지에 포함된 AMD64 및 Intel 64에서 사용된 OVMF UEFI 펌웨어를 사용할 수 없습니다. 대신 이러한 VM은 유사한 인터페이스를 제공하고 유사한 기능 세트를 구현하는 edk2-aarch64 패키지에 포함된 UEFI 펌웨어를 사용합니다.

특히 edk2-aarch64 는 내장 UEFI 쉘을 제공하지만 다음 기능은 지원하지 않습니다.

- SecureBoot
- 관리 모드
- TPM-1.2 지원

kvm-clock

kvm-clock 서비스는 ARM 64의 VM에서 시간 관리를 위해 구성할 필요가 없습니다.

주변 장치

ARM 64 시스템은 AMD64 및 Intel 64 시스템에서 지원되는 모든 주변 장치를 지원하지 않습니다. 경우에 따라 장치 기능이 전혀 지원되지 않으며 다른 경우에는 동일한 기능에 대해 다른 장치가 지원됩니다.

직렬 콘솔 구성

VM에 직렬 콘솔을 설정하는 경우 /etc/default/grub 파일에서 console=ttyS0 대신 console=ttyAMA0 매개변수를 사용합니다.

마스킹할 수 없는 인터럽트

현재 NMI(Non-maskable 인터럽트)를 ARM 64 VM으로 전송할 수 없습니다.

중첩 가상화

현재 ARM 64 호스트에서 중첩된 VM을 생성할 수 없습니다.

v2v 및 p2v

virt-v2v 및 virt-p2v 유틸리티는 AMD64 및 Intel 64 아키텍처에서만 지원되므로 ARM 64에서는 제공되지 않습니다.

22.7. RHEL 9에서 지원되는 가상화 기능 개요

다음 표에서는 사용 가능한 시스템 아키텍처에서 RHEL 9에서 선택한 가상화 기능의 지원 상태에 대한 비교 정보를 제공합니다.

표 22.1. 일반 지원

Intel 64 및 AMD64	IBM Z	ARM 64
지원됨	지원됨	*지원되지 않음* (기술 프리뷰)

표 22.2. 장치 핫 플러그 및 핫 언플러그

	Intel 64 및 AMD64	IBM Z	ARM 64
CPU 핫 플러그	지원됨	지원됨	*사용할 수 없음*
CPU 핫 플러그 연결	*지원되지 않음*	*지원되지 않음*	*사용할 수 없음*
메모리 핫 플러그	지원됨	*지원되지 않음*	*사용할 수 없음*

	Intel 64 및 AMD64	IBM Z	ARM 64
메모리 핫 플러그 연결	*지원되지 않음*	*지원되지 않음*	*사용할 수 없음*
주변 장치 핫 플러그	지원됨	지원됨 [a]	Available but *UNSUPPORTED*
주변 장치 핫 플러그 연결	지원됨	지원됨 [b]	Available but *UNSUPPORTED*

[a] virtio-*- **pci 대신 virtio- * - ccw** 장치를 사용해야 합니다.

[b] virtio-*- **pci 대신 virtio- * - ccw** 장치를 사용해야 합니다.

*표 **22.3.** 기타 선택된 기능*

	Intel 64 및 AMD64	IBM Z	ARM 64
NUMA 튜닝	지원됨	*지원되지 않음*	*사용할 수 없음*
SR-IOV 장치	지원됨	*지원되지 않음*	*사용할 수 없음*
virt-v2v 및 p2v	지원됨	*지원되지 않음*	*사용할 수 없음*

*지원되지 않는 일부 기능은 **Red Hat Virtualization** 및 **Red Hat OpenStack** 플랫폼과 같은 다른 **Red Hat** 제품에서 지원됩니다. 자세한 내용은 **RHEL 9** 가상화의 지원되지 않는 기능을 참조하십시오.*

추가 소스

- ***RHEL 9** 가상화에서 지원되지 않는 기능*