



Red Hat Enterprise Linux 8

가상화 구성 및 관리

Red Hat Enterprise Linux 8의 호스트 설정, 가상 시스템 생성 및 관리, 가상화 기능 이해

Red Hat Enterprise Linux 8 가상화 구성 및 관리

Red Hat Enterprise Linux 8의 호스트 설정, 가상 시스템 생성 및 관리, 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

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Configuring_and_managing_virtualization.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

초록

이 문서에서는 RHEL 8(Red Hat Enterprise Linux 8)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	8
RED HAT 문서에 관한 피드백 제공	9
1장. RHEL의 가상화 소개	10
1.1. 가상화란 무엇입니까?	10
1.2. 가상화의 이점	10
1.3. 가상 머신 구성 요소 및 상호 작용	11
1.4. 가상화 관리를 위한 툴 및 인터페이스	12
1.5. RED HAT 가상화 솔루션	13
2장. 가상화 시작하기	15
2.1. 가상화 활성화	15
2.2. 가상 머신 생성	17
2.2.1. 명령줄 인터페이스를 사용하여 가상 머신 생성	17
2.2.2. 웹 콘솔을 사용하여 가상 머신 생성 및 게스트 운영 체제 설치	20
2.2.2.1. 웹 콘솔을 사용하여 가상 머신 생성	20
2.2.2.2. 웹 콘솔을 사용하여 디스크 이미지를 가져와 가상 머신 생성	22
2.2.2.3. 웹 콘솔을 사용하여 게스트 운영 체제 설치	23
2.2.3. 웹 콘솔을 사용하여 클라우드 이미지 인증으로 가상 머신 생성	24
2.3. 가상 머신 시작	27
2.3.1. 명령줄 인터페이스를 사용하여 가상 머신 시작	27
2.3.2. 웹 콘솔을 사용하여 가상 머신 시작	28
2.3.3. 호스트가 시작될 때 자동으로 가상 시스템 시작	28
2.4. 가상 머신에 연결	30
2.4.1. 웹 콘솔을 사용하여 가상 머신과 상호 작용	30
2.4.1.1. 웹 콘솔에서 가상 머신 그래픽 콘솔 보기	30
2.4.1.2. 웹 콘솔을 사용하여 원격 뷰어에서 그래픽 콘솔 보기	32
2.4.1.3. 웹 콘솔에서 가상 머신 직렬 콘솔 보기	34
2.4.2. Virt Viewer를 사용하여 가상 머신 그래픽 콘솔 열기	35
2.4.3. SSH를 사용하여 가상 머신에 연결	36
2.4.4. 가상 머신 직렬 콘솔 열기	38
2.4.5. 원격 가상화 호스트에 대한 손쉬운 액세스 설정	39
2.5. 가상 머신 종료	41
2.5.1. 명령줄 인터페이스를 사용하여 가상 머신 종료	41
2.5.2. 웹 콘솔을 사용하여 가상 머신 종료 및 재시작	42
2.5.2.1. 웹 콘솔에서 가상 머신 종료	42
2.5.2.2. 웹 콘솔을 사용하여 가상 머신 재시작	42
2.5.2.3. 웹 콘솔을 사용하여 VM에 마스킹할 수 없는 인터럽트 전송	43
2.6. 가상 머신 삭제	44
2.6.1. 명령줄 인터페이스를 사용하여 가상 머신 삭제	44
2.6.2. 웹 콘솔을 사용하여 가상 머신 삭제	44
2.7. 추가 리소스	45
3장. IBM POWER에서 가상화 시작하기	46
3.1. IBM POWER에서 가상화 활성화	46
3.2. IBM POWER의 가상화가 AMD64 및 INTEL 64와 다른 방법	47
4장. IBM Z에서 가상화 시작하기	50
4.1. IBM Z에서 가상화 활성화	50
4.2. IBM Z의 가상화 업데이트 RHEL 8.5에서 RHEL 8.6 이상으로	51
4.3. IBM Z의 가상화가 AMD64 및 INTEL 64와 어떻게 다른지	52

4.4. 다음 단계	54
4.5. 추가 리소스	55
5장. 웹 콘솔에서 가상 머신 관리	56
5.1. 웹 콘솔을 사용한 가상 머신 관리 개요	56
5.2. 가상 머신 관리를 위해 웹 콘솔 설정	56
5.3. 웹 콘솔을 사용하여 가상 머신 이름 변경	57
5.4. 웹 콘솔에서 사용할 수 있는 가상 머신 관리 기능	58
5.5. VIRTUAL MACHINE MANAGER와 웹 콘솔의 가상화 기능 간 차이점	59
6장. 가상 머신에 대한 정보 보기	61
6.1. 명령줄 인터페이스를 사용하여 가상 머신 정보 보기	61
6.2. 웹 콘솔을 사용하여 가상 머신 정보 보기	63
6.2.1. 웹 콘솔에서 가상화 개요 보기	63
6.2.2. 웹 콘솔을 사용하여 스토리지 풀 정보 보기	64
6.2.3. 웹 콘솔에서 기본 가상 머신 정보 보기	65
6.2.4. 웹 콘솔에서 가상 머신 리소스 사용량 보기	66
6.2.5. 웹 콘솔에서 가상 머신 디스크 정보 보기	67
6.2.6. 웹 콘솔에서 가상 네트워크 인터페이스 정보 보기 및 편집	68
6.3. 가상 머신 XML 구성 샘플	69
7장. 가상 머신 저장 및 복원	75
7.1. 가상 머신의 저장 및 복원 방법	75
7.2. 명령줄 인터페이스를 사용하여 가상 머신 저장	75
7.3. 명령줄 인터페이스를 사용하여 가상 머신 시작	76
7.4. 웹 콘솔을 사용하여 가상 머신 시작	77
8장. 가상 머신 복제	78
8.1. 가상 머신 복제의 작동 방식	78
8.2. 가상 머신 템플릿 생성	78
8.2.1. virt-sysprep을 사용하여 가상 머신 템플릿 생성	78
8.2.2. 수동으로 가상 머신 템플릿 생성	80
8.3. 명령줄 인터페이스를 사용하여 가상 머신 복제	82
8.4. 웹 콘솔을 사용하여 가상 머신 복제	83
9장. 가상 머신 마이그레이션	85
9.1. 가상 머신 마이그레이션 방법	85
9.2. 가상 머신 마이그레이션의 이점	85
9.3. 가상 머신 마이그레이션의 제한 사항	86
9.4. 다른 호스트와 가상 머신 디스크 이미지 공유	87
9.5. 명령줄 인터페이스를 사용하여 가상 머신 마이그레이션	88
9.6. 웹 콘솔을 사용하여 가상 머신 실시간 마이그레이션	91
9.7. 가상 머신 마이그레이션에 지원되는 호스트	93
9.8. 추가 리소스	94
10장. 가상 장치 관리	95
10.1. 가상 장치의 작동 방식	95
10.2. 가상 장치 유형	96
10.3. CLI를 사용하여 가상 머신에 연결된 장치 관리	99
10.3.1. 가상 머신에 장치 연결	99
10.3.2. 가상 머신에 연결된 장치 수정	101
10.3.3. 가상 머신에서 장치 제거	103
10.4. 웹 콘솔을 사용하여 호스트 장치 관리	104
10.4.1. 웹 콘솔을 사용하여 가상 머신에 연결된 장치 보기	105
10.4.2. 웹 콘솔을 사용하여 가상 머신에 장치 연결	106

10.4.3. 웹 콘솔을 사용하여 가상 머신에서 장치 제거	108
10.5. 가상 USB 장치 관리	110
10.5.1. 가상 머신에 USB 장치 연결	111
10.5.2. 가상 머신에서 USB 장치 제거	112
10.5.3. 스마트 카드 판독기를 가상 머신에 연결	113
10.6. 가상 광 드라이브 관리	115
10.6.1. 가상 머신에 광 드라이브 연결	115
10.6.2. 가상 광 드라이브에서 ISO 이미지 교체	116
10.6.3. 가상 광 드라이브에서 ISO 이미지 제거	117
10.6.4. 가상 머신에서 광 드라이브 제거	118
10.7. SR-IOV 장치 관리	119
10.7.1. SR-IOV란 무엇입니까?	119
10.7.2. 가상 머신에 SR-IOV 네트워킹 장치 연결	121
10.7.3. SR-IOV 할당에 지원되는 장치	126
10.8. IBM Z의 가상 머신에 DASD 장치 연결	126
11장. 가상 머신용 스토리지 관리	131
11.1. 가상 머신 스토리지 이해	131
11.1.1. 스토리지 풀 소개	131
11.1.2. 스토리지 볼륨 소개	132
11.1.3. libvirt를 사용한 스토리지 관리	133
11.1.4. 스토리지 관리 개요	133
11.1.5. 지원되지 않는 스토리지 풀 유형	134
11.2. CLI를 사용하여 가상 머신 스토리지 풀 관리	135
11.2.1. CLI를 사용하여 스토리지 풀 정보 보기	136
11.2.2. CLI를 사용하여 디렉터리 기반 스토리지 풀 생성	137
11.2.3. CLI를 사용하여 디스크 기반 스토리지 풀 생성	139
11.2.4. CLI를 사용하여 파일 시스템 기반 스토리지 풀 생성	142
11.2.5. CLI를 사용하여 GlusterFS 기반 스토리지 풀 생성	144
11.2.6. CLI를 사용하여 iSCSI 기반 스토리지 풀 생성	147
11.2.7. CLI를 사용하여 LVM 기반 스토리지 풀 생성	149
11.2.8. CLI를 사용하여 NFS 기반 스토리지 풀 생성	151
11.2.9. CLI를 사용하여 vHBA 장치가 있는 SCSI 기반 스토리지 풀 생성	153
11.2.10. CLI를 사용하여 스토리지 풀 삭제	155
11.3. 웹 콘솔을 사용하여 가상 머신 스토리지 풀 관리	156
11.3.1. 웹 콘솔을 사용하여 스토리지 풀 정보 보기	157
11.3.2. 웹 콘솔을 사용하여 디렉터리 기반 스토리지 풀 생성	159
11.3.3. 웹 콘솔을 사용하여 NFS 기반 스토리지 풀 생성	161
11.3.4. 웹 콘솔을 사용하여 iSCSI 기반 스토리지 풀 생성	163
11.3.5. 웹 콘솔을 사용하여 디스크 기반 스토리지 풀 생성	166
11.3.6. 웹 콘솔을 사용하여 LVM 기반 스토리지 풀 생성	168
11.3.7. 웹 콘솔을 사용하여 vHBA 장치로 SCSI 기반 스토리지 풀 생성	171
11.3.8. 웹 콘솔을 사용하여 스토리지 풀 제거	173
11.3.9. 웹 콘솔을 사용하여 스토리지 풀 비활성화	175
11.4. 스토리지 풀 생성을 위한 매개변수	176
11.4.1. 디렉터리 기반 스토리지 풀 매개변수	176
11.4.2. 디스크 기반 스토리지 풀 매개변수	177
11.4.3. 파일 시스템 기반 스토리지 풀 매개변수	178
11.4.4. GlusterFS 기반 스토리지 풀 매개변수	180
11.4.5. iSCSI 기반 스토리지 풀 매개변수	181
11.4.6. LVM 기반 스토리지 풀 매개변수	182
11.4.7. NFS 기반 스토리지 풀 매개변수	184
11.4.8. vHBA 장치가 있는 SCSI 기반 스토리지 풀의 매개변수	185

11.5. CLI를 사용하여 가상 머신 스토리지 볼륨 관리	187
11.5.1. CLI를 사용하여 스토리지 볼륨 정보 보기	188
11.5.2. CLI를 사용하여 스토리지 볼륨 생성 및 할당	188
11.5.3. CLI를 사용하여 스토리지 볼륨 삭제	191
11.6. 웹 콘솔을 사용하여 가상 머신 스토리지 볼륨 관리	192
11.6.1. 웹 콘솔을 사용하여 스토리지 볼륨 생성	192
11.6.2. 웹 콘솔을 사용하여 스토리지 볼륨 제거	194
11.7. 웹 콘솔을 사용하여 가상 머신 스토리지 디스크 관리	196
11.7.1. 웹 콘솔에서 가상 머신 디스크 정보 보기	197
11.7.2. 웹 콘솔을 사용하여 가상 머신에 새 디스크 추가	198
11.7.3. 웹 콘솔을 사용하여 기존 디스크를 가상 머신에 연결	201
11.7.4. 웹 콘솔을 사용하여 가상 머신에서 디스크 분리	203
11.8. LIBVIRT 보안을 사용하여 ISCSI 스토리지 풀 보안	204
11.9. VHBA 생성	206
12장. 가상 머신의 GPU 장치 관리	210
12.1. 가상 머신에 GPU 할당	210
12.2. NVIDIA vGPU 장치 관리	214
12.2.1. NVIDIA vGPU 장치 설정	214
12.2.2. NVIDIA vGPU 장치 제거	218
12.2.3. 시스템에 대한 NVIDIA vGPU 정보 얻기	220
12.2.4. NVIDIA vGPU 용 원격 데스크탑 스트리밍 서비스	221
12.2.5. 추가 리소스	222
13장. 가상 머신 네트워크 연결 구성	223
13.1. 가상 네트워킹 이해	223
13.1.1. 가상 네트워크의 작동 방식	223
13.1.2. 가상 네트워킹 기본 구성	225
13.2. 가상 머신 네트워크 인터페이스 관리에 웹 콘솔 사용	225
13.2.1. 웹 콘솔에서 가상 네트워크 인터페이스 정보 보기 및 편집	226
13.2.2. 웹 콘솔에서 가상 네트워크 인터페이스 추가 및 연결	228
13.2.3. 웹 콘솔에서 가상 네트워크 인터페이스 연결 해제 및 제거	229
13.3. 권장되는 가상 머신 네트워킹 구성	229
13.3.1. 명령줄 인터페이스를 사용하여 외부에 표시되는 가상 머신 구성	230
13.3.2. 웹 콘솔을 사용하여 외부에서 표시되는 가상 머신 구성	232
13.4. 가상 머신 네트워크 연결 유형	235
13.4.1. 네트워크 주소 변환을 통한 가상 네트워킹	235
13.4.2. 라우팅 모드의 가상 네트워킹	236
13.4.3. 브릿지 모드에서 가상 네트워킹	238
13.4.4. 분리된 모드의 가상 네트워킹	240
13.4.5. 오픈 모드에서 가상 네트워킹	240
13.4.6. 가상 머신 연결 유형 비교	240
13.5. PXE 서버에서 가상 머신 부팅	241
13.5.1. 가상 네트워크에서 PXE 부팅 서버 설정	241
13.5.2. PXE 및 가상 네트워크를 사용하여 가상 머신 부팅	243
13.5.3. PXE 및 브릿지 네트워크를 사용하여 가상 머신 부팅	244
13.6. 추가 리소스	245
14장. 호스트와 해당 가상 머신 간의 파일 공유	247
14.1. NFS를 사용하여 호스트와 LINUX 가상 시스템 간 파일 공유	247
14.2. SAMBA를 사용하여 호스트와 WINDOWS 가상 시스템 간에 파일 공유	250
14.3. VIRTIOFS를 사용하여 호스트와 가상 머신 간에 파일을 공유하는 웹 콘솔을 사용합니다.	255
14.4. 웹 콘솔을 사용하여 VIRTIOFS를 사용하여 호스트와 해당 가상 머신 간의 공유 파일을 제거	257

15장. 가상 머신 보안	260
15.1. 가상 머신에서 보안이 작동하는 방식	260
15.2. 가상 머신 보안을 위한 모범 사례	261
15.3. SECUREBOOT 가상 머신 생성	263
15.4. 가상 머신 사용자가 사용할 수 있는 작업 제한	264
15.5. 가상 머신 보안을 위한 자동 기능	266
15.6. 가상화를 위한 SELINUX 부울	267
15.7. IBM Z에서 IBM SECURE EXECUTION 설정	268
15.8. IBM Z의 가상 머신에 암호화 코프로세서 연결	273
15.9. WINDOWS 가상 머신에서 표준 하드웨어 보안 활성화	277
15.10. WINDOWS 가상 머신에서 향상된 하드웨어 보안 활성화	278
16장. 가상 머신 성능 최적화	281
16.1. 가상 머신 성능에 어떤 영향을 주는가	281
시스템 성능에 가상화가 미치는 영향	281
VM 성능 손실 감소	282
16.2. TUNED를 사용하여 가상 머신 성능 최적화	282
16.3. 가상 머신 메모리 구성	284
16.3.1. 웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거	284
16.3.2. 명령줄 인터페이스를 사용하여 가상 머신 메모리 추가 및 제거	286
16.3.3. 추가 리소스	288
16.4. 가상 머신 I/O 성능 최적화	289
16.4.1. 가상 머신의 블록 I/O 튜닝	289
16.4.2. 가상 머신의 디스크 I/O 제한	290
16.4.3. 다중 대기열 virtio-scsi 활성화	291
16.5. 가상 머신 CPU 성능 최적화	292
16.5.1. 명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거	292
16.5.2. 웹 콘솔을 사용하여 가상 CPU 관리	294
16.5.3. 가상 머신에서 NUMA 구성	296
16.5.4. 샘플 vCPU 성능 튜닝 시나리오	298
16.5.5. 커널 동일 페이지 병합 비활성화	305
16.6. 가상 머신 네트워크 성능 최적화	306
16.7. 가상 머신 성능 모니터링 툴	307
16.8. 추가 리소스	310
17장. WINDOWS 가상 머신 설치 및 관리	311
17.1. WINDOWS 가상 머신 설치	311
17.2. WINDOWS 가상 머신 최적화	314
17.2.1. Windows 가상 머신용 KVM 반가상화 드라이버 설치	314
17.2.1.1. Windows virtio 드라이버 작동 방식	315
17.2.1.2. 호스트 머신에서 virtio 드라이버 설치 미디어 준비	316
17.2.1.3. Windows 게스트에 virtio 드라이버 설치	318
17.2.2. Hyper-V 구현 활성화	321
17.2.2.1. Windows 가상 머신에서 Hyper-V 구현 활성화	321
17.2.2.2. 구성 가능한 Hyper-V 구현	322
17.2.3. NetKVM 드라이버 매개변수 구성	325
17.2.4. NetKVM 드라이버 매개변수	327
17.2.5. Windows 가상 머신에서 백그라운드 프로세스 최적화	328
17.3. WINDOWS 가상 머신에서 표준 하드웨어 보안 활성화	330
17.4. WINDOWS 가상 머신에서 향상된 하드웨어 보안 활성화	331
17.5. 다음 단계	333
18장. 중첩된 가상 머신 생성	334
18.1. INTEL에서 중첩된 가상 머신 생성	334

18.2. AMD에 중첩 가상 머신 생성	336
18.3. IBM Z에 중첩 가상 머신 생성	338
18.4. IBM POWER9에 중첩 가상 머신 생성	339
18.5. 중첩된 가상화에 대한 제한 사항 및 제한 사항	341
19장. 가상 머신 문제 진단	345
19.1. LIBVIRT 디버그 로그 생성	345
19.1.1. libvirt 디버그 로그 이해	345
19.1.2. libvirt 디버그 로그에 대한 영구 설정 활성화	345
19.1.3. 런타임 중 libvirt 디버그 로그 활성화	346
19.1.4. 요청을 지원하기 위해 libvirt 디버그 로그 연결	348
19.2. 가상 머신 코어 덤프	349
19.2.1. 가상 머신 코어 덤프 작동 방식	349
19.2.2. 가상 머신 코어 덤프 파일 생성	350
19.3. BACKTRACING 가상 머신 프로세스	351
20장. RHEL 8 가상화의 기능 지원 및 제한 사항	353
20.1. RHEL 가상화 지원의 작동 방식	353
20.2. RHEL 8 가상화의 권장 기능	353
20.3. RHEL 8 가상화에서 지원되지 않는 기능	356
20.4. RHEL 8 가상화의 리소스 할당 제한	361
20.5. RHEL 8의 가상화 기능 지원 개요	362

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. Red Hat이 어떻게 이를 개선할 수 있는지 알려 주십시오.

- 특정 문구에 대한 간단한 주석은 다음과 같습니다.
 1. 문서가 *Multi-page HTML* 형식으로 표시되는지 확인합니다. 또한 문서 오른쪽 상단에 **Feedback** (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. Component로 **Documentation**을 선택하십시오.
 3. **Description** 필드에 문서 개선을 위한 제안 사항을 기입하십시오. 관련된 문서의 해당 부분 링크를 알려주십시오.
 4. **Submit Bug**를 클릭하십시오.

1장. RHEL의 가상화 소개

가상화의 개념이나 Linux의 구현 개념에 익숙하지 않은 경우 다음 섹션에서는 RHEL 8의 가상화에 대한 개요를 제공합니다. 기본 사항, 이점, 구성 요소 및 Red Hat에서 제공하는 기타 가능한 가상화 솔루션.

1.1. 가상화란 무엇입니까?

RHEL 8에서는 RHEL 8을 실행하는 머신이 *게스트라고도 하는 여러 VM(가상 머신)*을 호스팅 할 수 있는 *가상화 기능을 제공합니다*. VM은 호스트의 물리적 하드웨어 및 컴퓨팅 리소스를 사용하여 호스트 운영 체제의 사용자 공간 프로세스로 별도의 가상화된 운영 체제(*게스트 OS*)를 실행합니다.

즉, 가상화를 통해 운영 체제 내에서 운영 체제를 구축할 수 있습니다.

VM을 사용하면 소프트웨어 구성 및 기능을 안전하게 테스트하거나 레거시 소프트웨어를 실행하거나 하드웨어의 워크로드 효율성을 최적화할 수 있습니다. 혜택에 대한 자세한 내용은 [가상화 Advantages](#)를 참조하십시오.

가상화란 무엇인지에 대한 자세한 내용은 [Red Hat 고객 포털](#)을 참조하십시오.

다음 단계

- Red Hat Enterprise Linux 8에서 가상화를 사용해 보려면 [가상화 시작하기](#)를 참조하십시오.
- Red Hat Enterprise Linux 8 가상화 외에도 Red Hat은 다양한 사용자 중심 및 기능을 갖춘 다양한 전문 가상화 솔루션을 제공합니다. 자세한 내용은 [Red Hat 가상화 솔루션](#)을 참조하십시오.

1.2. 가상화의 이점

VM(가상 머신)을 사용하면 물리적 머신 사용에 비해 다음과 같은 이점이 있습니다.

- **유연하고 세밀한 리소스 할당**
VM은 일반적으로 물리적인 호스트 시스템에서 실행되며 게스트 OS에서 사용할 물리적 하드웨어도 할당할 수 있습니다. 그러나 VM에 물리적 리소스를 할당하는 작업은 소프트웨어 수준에서 수행되므로 매우 유연합니다. VM은 호스트 메모리, CPU 또는 스토리지 공간의 구성 요소를 사용하며 매우 세밀한 리소스 요청을 지정할 수 있습니다.

예를 들어 게스트 OS가 호스트 파일 시스템의 디스크로 표시할 수 있는 것으로 표시되는 것은 물론, 해당 디스크의 크기는 물리적 디스크에 사용 가능한 크기보다 덜 제한적입니다.
- **소프트웨어 제어 구성**
VM의 전체 구성은 호스트에 데이터로 저장되며 소프트웨어 제어 하에 있습니다. 따라서 VM을 쉽게 생성, 제거, 복제, 마이그레이션, 작동 또는 원격 스토리지에 연결할 수 있습니다.
- **호스트에서 분리**
게스트 OS는 호스트 OS와는 달리 가상화된 커널에서 실행됩니다. 즉, VM에 모든 OS를 설치할 수 있으며 게스트 OS가 불안정해지거나 손상된 경우에도 호스트에 영향을 주지 않습니다.
- **공간 및 비용 효율성**
단일 물리적 시스템은 다수의 VM을 호스팅할 수 있습니다. 따라서 여러 물리적 시스템이 동일한 작업을 수행할 필요가 없으므로 물리적 하드웨어와 관련된 공간, 전력 및 유지 관리 요구 사항을 줄일 수 있습니다.
- **소프트웨어 호환성**

VM은 호스트와 다른 OS를 사용할 수 있기 때문에 가상화를 통해 호스트 OS에 대해 원래 릴리스되지 않은 애플리케이션을 실행할 수 있습니다. 예를 들어 RHEL 7 게스트 OS를 사용하면 RHEL 8 호스트 시스템에서 RHEL 7용으로 릴리스된 애플리케이션을 실행할 수 있습니다.



참고

RHEL 8 호스트에서 모든 운영 체제가 게스트 OS로 지원되는 것은 아닙니다. 자세한 내용은 [RHEL 8 가상화의 권장 기능](#)을 참조하십시오.

1.3. 가상 머신 구성 요소 및 상호 작용

RHEL 8의 가상화는 다음과 같은 주요 소프트웨어 구성 요소로 구성됩니다.

하이퍼바이저

RHEL 8에서 VM(가상 시스템)을 생성하는 것은 하드웨어를 제어하고 호스트 시스템에서 여러 운영 체제를 실행할 수 있는 소프트웨어 계층인 *하이퍼바이저*입니다.

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

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

또한 **libvirt** 소프트웨어 제품군은 관리 및 통신 계층 역할을 하며 QEMU를 보다 쉽게 상호 작용하고 보안 규칙을 적용하며 VM 구성 및 실행을 위한 여러 추가 툴을 제공합니다.

XML 구성

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

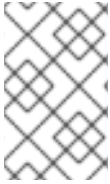
- VM 이름, 시간대 및 VM에 대한 기타 정보와 같은 메타데이터입니다.
- 가상 CPU(vCPU), 스토리지 장치, 입출력 장치, 네트워크 인터페이스 카드 및 기타 하드웨어, 실제 및 가상 하드웨어를 포함하여 VM의 장치에 대한 설명입니다.
- VM 설정(예: 사용할 수 있는 최대 메모리 양, 다시 시작 설정 및 VM 동작에 대한 기타 설정).

XML 구성의 콘텐츠에 대한 자세한 내용은 [샘플 가상 머신 XML 구성](#)을 참조하십시오.

구성 요소 상호 작용

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

이러한 가상화 도구를 사용하는 경우 libvirt는 입력을 QEMU의 지침으로 변환합니다. QEMU는 지침을 KVM에 전달하여 커널이 지침을 수행하는 데 필요한 리소스를 적절하게 할당할 수 있도록 합니다. 결과적으로 QEMU는 VM 생성 또는 수정이나 VM의 게스트 운영 체제에서 작업을 수행하는 등 해당 사용자 공간 변경 사항을 실행할 수 있습니다.

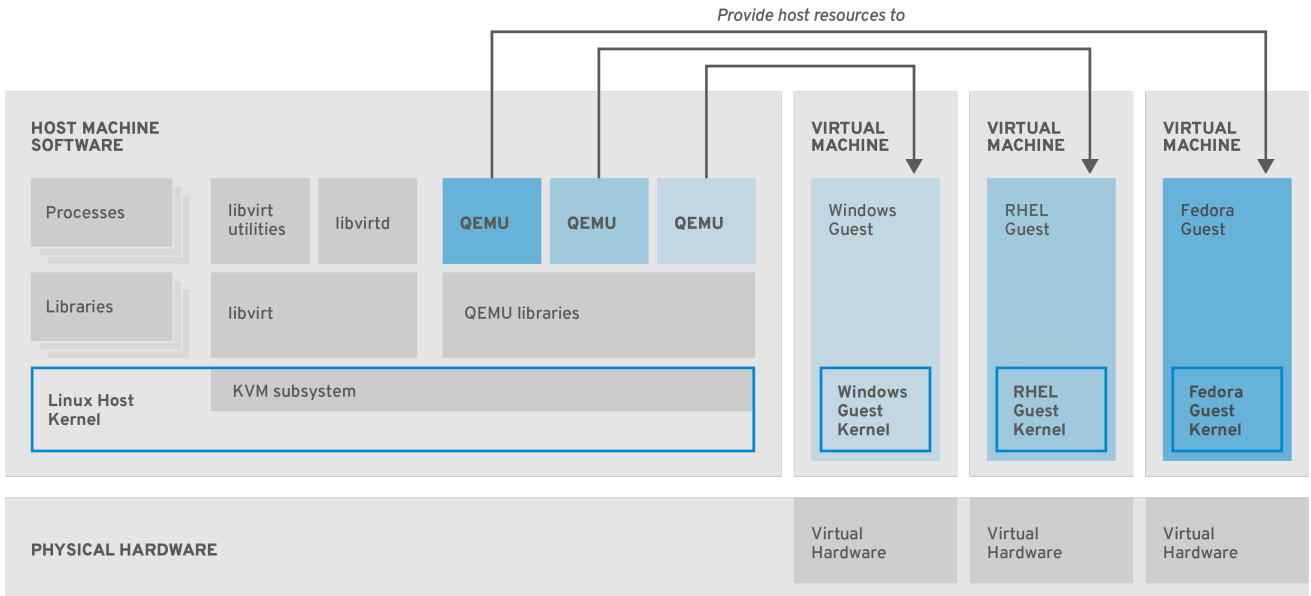


참고

QEMU는 아키텍처의 필수 구성 요소이지만 보안 문제로 인해 RHEL 8 시스템에서 직접 사용하지는 않습니다. 따라서 **qemu-*** 명령을 사용하는 것은 Red Hat에서 지원하지 않으므로 libvirt를 사용하여 QEMU와 상호 작용하는 것이 좋습니다.

호스트 기반 인터페이스에 대한 자세한 내용은 [가상화 관리를 위한 도구 및 인터페이스](#)를 참조하십시오.

그림 1.1. RHEL 8 가상화 아키텍처



RHEL_7_0319

1.4. 가상화 관리를 위한 툴 및 인터페이스

CLI(명령줄 인터페이스) 또는 여러 GUI(그래픽 사용자 인터페이스)를 사용하여 RHEL 8에서 가상화를 관리할 수 있습니다.

명령줄 인터페이스

CLI는 RHEL 8에서 가상화를 관리하는 가장 강력한 방법입니다. VM(가상 머신) 관리를 위한 주요 CLI 명령은 다음과 같습니다.

- **virsh** - 제공된 인수에 따라 다양한 목적을 가진 다재다능한 가상화 명령줄 유틸리티 및 셸입니다. 예를 들면 다음과 같습니다.
 - VM 시작 및 종료 - **virsh start** and **virsh shutdown**
 - 사용 가능한 VM 나열 - **virsh list**
 - 구성 파일에서 VM 생성 - **virsh create**
 - 가상화 셸 입력 - **virsh**

자세한 내용은 **virsh(1)** 도움말 페이지를 참조하십시오.

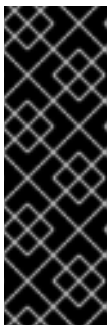
- **virt-install** - 새 VM 생성을 위한 CLI 유틸리티. 자세한 내용은 **virt-install(1)** 도움말 페이지를 참조하십시오.

- **virt-xml** - VM 구성 편집을 위한 유틸리티입니다.
- **guestfish** - VM 디스크 이미지를 검사하고 수정하기 위한 유틸리티입니다. 자세한 내용은 [guestfish\(1\)](#) 도움말 페이지를 참조하십시오.

그래픽 인터페이스

다음 GUI를 사용하여 RHEL 8의 가상화를 관리할 수 있습니다.

- **Cockpit** 이라고도 하는 **RHEL 8 웹 콘솔**은 VM 및 가상화 호스트를 관리하기 위한 원격으로 액세스 가능하고 사용하기 쉬운 그래픽 사용자 인터페이스를 제공합니다. 웹 콘솔을 사용한 기본 가상화 관리에 대한 자세한 내용은 웹 콘솔 [의 가상 머신 관리](#)를 참조하십시오.
- **virt-manager**(Virtual Machine Manager) 애플리케이션은 VM 및 가상화 호스트 관리를 위한 특수 GUI를 제공합니다.

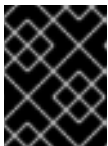


중요

아직 RHEL 8에서 지원되지만 **virt-manager**는 더 이상 사용되지 않습니다. 웹 콘솔은 후속 릴리스에서 교체될 예정입니다. 따라서 GUI에서 가상화를 관리하는 데 웹 콘솔에 익숙해지는 것이 좋습니다.

그러나 RHEL 8에서는 일부 기능은 **virt-manager** 또는 명령줄에서만 액세스할 수 있습니다. 자세한 내용은 [Virtual Machine Manager의 가상화 기능과 웹 콘솔 간의 차이](#)를 참조하십시오.

- **Gnome Boxes** 애플리케이션은 VM 및 원격 시스템을 보고 액세스할 수 있는 경량의 그래픽 인터페이스입니다. GNOME 박스는 주로 데스크탑 시스템에서 사용하도록 설계되었습니다.



중요

GNOME 박스는 GNOME 데스크탑 환경의 일부로 제공되며 RHEL 8에서 지원되지만 GUI에서 가상화를 관리하는 데 웹 콘솔을 사용하는 것이 좋습니다.

추가 리소스

- [가상화 시작하기](#)

1.5. RED HAT 가상화 솔루션

다음 Red Hat 제품은 RHEL 8 가상화 기능을 기반으로 구축되었으며 RHEL 8에서 사용할 수 있는 KVM 가상화 기능을 확장합니다. 또한 [RHEL 8 가상화의 많은 제한 사항](#)은 다음 제품에는 적용되지 않습니다.

RHV(Red Hat Virtualization)

RHV는 엔터프라이즈급 확장성과 성능을 위해 설계되었으며, 중앙 집중식 그래픽 인터페이스에서 호스트, 가상 시스템, 네트워크, 스토리지 및 사용자를 포함한 전체 가상 인프라를 관리할 수 있습니다. Red Hat Virtualization은 대규모 배포 또는 미션 크리티컬 애플리케이션을 실행하는 기업에서 사용할 수 있습니다. Red Hat Virtualization에 적합한 대규모 배포의 예로는 다운타임 없이 지속적으로 실행해야 하는 데이터베이스, 거래 플랫폼 및 메시징 시스템이 있습니다.

Red Hat Virtualization에 대한 자세한 내용은 [Red Hat 고객 포털](#) 또는 [Red Hat Virtualization 설명서 제품군](#)을 참조하십시오.

완전히 지원되는 60일 Red Hat Virtualization 평가판 버전을 다운로드하려면 [Red Hat 고객 포털](#)을 참조하십시오.

Red Hat OpenStack Platform (RHOSP)

Red Hat OpenStack Platform은 안전하고 안정적인 퍼블릭 또는 프라이빗 [OpenStack](#) 클라우드를 구축, 배포 및 확장할 수 있는 통합 기반을 제공합니다.

Red Hat OpenStack Platform에 대한 자세한 내용은 [Red Hat Customer Portal](#) 또는 [Red Hat OpenStack Platform 설명서 제품군](#)을 참조하십시오.

또한 특정 Red Hat 제품은 **컨테이너화**라고도 하는 운영 체제 수준의 **가상화**를 제공합니다.

- 컨테이너는 호스트 OS의 격리된 인스턴스이며 기존 OS 커널에서 작동합니다. 컨테이너에 대한 자세한 내용은 [Red Hat 고객 포털](#)을 참조하십시오.
- 컨테이너에는 KVM 가상화의 다양한 기능이 없지만 더 가볍고 유연하여 처리할 수 있습니다. 자세한 내용은 [Linux 컨테이너 소개](#)를 참조하십시오.



참고

RHEL에서 지원되지 않지만 다른 Red Hat 가상화 솔루션에서 지원되지 않는 가상화 기능에 대한 자세한 내용은 [RHEL 8 가상화의 지원되지 않는 기능](#)을 참조하십시오.

2장. 가상화 시작하기

RHEL 8에서 가상화를 사용하려면 아래 단계를 따르십시오. 기본 방법은 CLI(명령줄 인터페이스)를 사용하지만 사용자 편의를 위해 일부 단계는 웹 콘솔 GUI에서 완료할 수 있습니다.

1. 가상화 모듈을 활성화하고 가상화 패키지를 설치합니다. 가상화 패키지 [활성화](#)에서.
2. VM(가상 머신)을 생성합니다.
 - CLI의 경우 [명령줄 인터페이스를 사용하여 가상 머신 생성](#)을 참조하십시오.
 - GUI의 경우 [가상 머신 생성 및 웹 콘솔을 사용하여 게스트 운영 체제 설치](#)를 참조하십시오.
3. VM을 시작합니다.
 - CLI의 경우 [명령줄 인터페이스를 사용하여 가상 머신 시작](#)을 참조하십시오.
 - GUI에 대해서는 [웹 콘솔을 사용하여 가상 머신 시작](#)을 참조하십시오.
4. VM에 연결합니다.
 - CLI의 경우 [SSH를 사용하여 가상 머신에 연결](#)하거나 [Virt Viewer를 사용하여 가상 머신 그래픽 콘솔을 엽니다](#).
 - GUI에 대해서는 [웹 콘솔을 사용하여 가상 머신 상호 작용](#)을 참조하십시오.



참고

웹 콘솔은 현재 VM 관리 기능의 일부만 제공하므로 명령줄을 사용하면 RHEL 8의 고급 가상화를 사용하는 것이 좋습니다.

2.1. 가상화 활성화

RHEL 8에서 가상화를 사용하려면 가상화 모듈을 활성화하고, 가상화 패키지를 설치하고, 시스템이 VM(가상 시스템)을 호스팅하도록 구성되어 있는지 확인해야 합니다.

사전 요구 사항

- Red Hat Enterprise Linux 8은 호스트 시스템에 [설치 및 등록](#)됩니다.
- 시스템은 가상화 호스트로 작동하기 위한 다음 하드웨어 요구 사항을 충족합니다.
 - 사용 가능한 최소 시스템 리소스는 다음과 같습니다.
 - 호스트용 디스크 여유 공간이 6GB, 의도한 각 VM에 대해 다른 6GB가 추가됩니다.
 - 호스트용 2GB RAM 및 각 VM에 대해 다른 2GB.
 - 호스트의 CPU 4개. VM은 일반적으로 단일 할당된 vCPU로 실행할 수 있지만, 부하가 높은 동안 VM이 응답하지 않도록 Red Hat은 VM당 2개 이상의 vCPU를 할당하는 것이 좋습니다.
 - 호스트 시스템의 아키텍처는 [KVM 가상화를 지원](#)합니다.
 - 특히 RHEL 8은 64비트 ARM 아키텍처(ARM 64)에서 가상화를 지원하지 않습니다.

- 아래 절차는 AMD64 및 Intel 64 아키텍처(x86_64)에 적용됩니다. 다른 지원 아키텍처로 호스트에서 가상화를 활성화하려면 다음 섹션 중 하나를 참조하십시오.

- [IBM POWER에서 가상화 활성화](#)
- [IBM Z에서 가상화 활성화](#)

절차

1. RHEL 8 가상화 모듈에 패키지를 설치합니다.

```
# yum module install virt
```

2. **virt-install** 및 **virt-viewer** 패키지를 설치합니다.

```
# yum install virt-install virt-viewer
```

3. **libvirt** 서비스를 시작합니다.

```
# systemctl start libvirtd
```

검증

1. 시스템이 가상화 호스트가 될 준비가 되었는지 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 **PASS** 값을 반환하면 시스템에서 **VM을 생성할** 준비가 됩니다. 검사 중 **실패** 값을 반환하는 경우 표시된 지침에 따라 문제를 해결합니다.

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

문제 해결

- 호스트 CPU에서 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 VM은 성능 문제가 아닌 부팅에 실패합니다.

이 문제를 해결하려면 VM의 XML 구성에서 `<domain type='qemu'>` 값을 **qemu** 로 변경할 수 있습니다. 그러나 Red Hat은 **qemu** 도메인 유형을 사용하는 VM을 지원하지 않으며 프로덕션 환경에서는 권장되지 않습니다.

2.2. 가상 머신 생성

RHEL 8에서 VM(가상 머신)을 생성하려면 [명령줄 인터페이스](#) 또는 [RHEL 8 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 의도한 작업 및 워크로드에 따라 크게 달라질 수 있습니다.



주의

RHEL 8에서는 호스트 CD-ROM 또는 DVD-ROM 장치에서 설치할 수 없습니다. RHEL 8에서 사용 가능한 VM 설치 방법을 사용할 때 CD-ROM 또는 DVD-ROM을 설치 소스로 선택하면 설치에 실패합니다. 자세한 내용은 [Red Hat Knowledgebase](#) 를 참조하십시오.

2.2.1. 명령줄 인터페이스를 사용하여 가상 머신 생성

virt-install 유틸리티를 사용하여 RHEL 8 호스트에서 VM(가상 머신)을 만들려면 아래 지침을 따르십시오.

사전 요구 사항

- 가상화가 호스트 시스템에서 [활성화됩니다](#).
- 디스크 공간, RAM 또는 CPU와 같이 VM에 할당할 수 있는 충분한 시스템 리소스가 있습니다. 권장 값은 VM의 의도한 작업 및 워크로드에 따라 크게 달라질 수 있습니다.
- OS(운영 체제) 설치 소스는 로컬 또는 네트워크에서 사용할 수 있습니다. 다음 중 하나일 수 있습니다.
 - 설치 미디어의 ISO 이미지
 - 기존 VM 설치의 디스크 이미지



주의

RHEL 8에서는 호스트 CD-ROM 또는 DVD-ROM 장치에서 설치할 수 없습니다. RHEL 8에서 사용할 수 있는 VM 설치 방법을 사용할 때 CD-ROM 또는 DVD-ROM을 설치 소스로 선택하면 설치에 실패합니다. 자세한 내용은 [Red Hat Knowledgebase](#) 를 참조하십시오.

- 선택 사항: Kickstart 파일을 제공하여 설치 시 보다 빠르고 쉽게 구성할 수 있습니다.

절차

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

- 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 값은 다를 수 있습니다. 예를 보려면 다음을 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 RAM과 2개의 vCPU로 할당되며, 80GiB qcow2 가상 디스크도 자동으로 VM에 대해 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/rhel8.iso** 이미지를 사용하여 라이브 CD에서 RHEL 8 OS를 실행하는 **demo-guest2** 라는 VM을 생성합니다. 이 VM에 디스크 공간이 할당되지 않으므로 세션 중에 변경한 내용은 유지되지 않습니다. 또한 VM에는 4096MiB의 RAM과 4개의 vCPU가 할당됩니다.

```
# virt-install --name demo-guest2 --memory 4096 --vcpus 4 --disk none --livecd --os-variant rhel8.0 --cdrom /home/username/Downloads/rhel8.iso
```

- 다음은 기존 디스크 이미지 **/home/username/backup/disk.qcow2** 에 연결하는 **demo-guest3** 이라는 RHEL 8 VM을 생성합니다. 이것은 시스템 간에 하드 드라이브를 물리적으로 이동하는 것과 비슷하므로, **demo-guest3**에서 사용할 수 있는 OS와 데이터는 이전에 이미지를 처리한 방법에 따라 결정됩니다. 또한 이 VM에는 2048MiB RAM과 2개의 vCPU가 할당됩니다.

```
# virt-install --name demo-guest3 --memory 2048 --vcpus 2 --os-variant rhel8.0 --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 시작하려면 URL에 작동 중인 OS 설치 트리가 포함되어야 합니다. 또한 OS는 `/home/username/ks.cfg` kickstart 파일을 사용하여 자동으로 구성됩니다. 이 VM은 2048MiB RAM, vCPU 2개, 160GiB qcow2 가상 디스크도 함께 할당됩니다.

```
# virt-install --name demo-guest4 --memory 2048 --vcpus 2 --disk size=160 --os-variant
rhel8.0 --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 **RHEL8.iso** 이미지 파일에서 그래픽 없이 텍스트 전용 모드로 설치하는 **demo-guest5** 라는 VM을 생성합니다. 게스트 콘솔을 직렬 콘솔에 연결합니다. VM에는 16384MiB 메모리, 16 vCPU 및 280GiB 디스크가 있습니다. 이러한 종류의 설치에는 느린 네트워크 링크를 통해 호스트에 연결할 때 유용합니다.

```
# virt-install --name demo-guest5 --memory 16384 --vcpus 16 --disk size=280 --os-
variant rhel8.0 --location RHEL8.iso --graphics none --extra-args='console=ttyS0'
```

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

```
# virt-install --connect qemu+ssh://root@10.0.0.1/system --name demo-guest6 --
memory 16384 --vcpus 16 --disk size=280 --os-variant rhel8.0 --location RHEL8.iso --
graphics none --extra-args='console=ttyS0'
```

검증

- VM이 성공적으로 생성되면 VM의 그래픽 콘솔로 **virt-viewer** 창이 열리고 게스트 OS 설치를 시작합니다.

문제 해결

- **virt-install**이 실패하고 기본 네트워크 오류를 찾을 수 없는 경우:
 - a. **libvirt-daemon-config-network** 패키지가 설치되어 있는지 확인합니다.

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

- b. **libvirt** 기본 네트워크가 활성화 상태이며 자동으로 시작되도록 구성되어 있는지 확인합니다.

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

- c. 기본 네트워크를 활성화하여 자동 시작으로 설정합니다.

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

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

■

- i. 기본 네트워크를 활성화하는 데 실패하고 다음 오류가 발생하면 *libvirt-daemon-config-network* 패키지가 올바르게 설치되지 않았습니니다.

```
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.

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

- ii. 다음과 유사한 오류로 기본 네트워크를 활성화하지 못하면 기본 네트워크의 서브넷과 호스트의 기존 인터페이스 사이에 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
- [가상 머신 복제](#)

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

RHEL 8 호스트의 GUI에서 VM(가상 시스템)을 관리하려면 웹 콘솔을 사용합니다. 다음 섹션에서는 RHEL 8 웹 콘솔을 사용하여 VM을 생성하고 게스트 운영 체제를 설치하는 방법에 대한 정보를 제공합니다.

2.2.2.1. 웹 콘솔을 사용하여 가상 머신 생성

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

사전 요구 사항

- 가상화가 호스트 시스템에서 [활성화됩니다](#).
- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.
- 디스크 공간, RAM 또는 CPU와 같이 VM에 할당할 수 있는 충분한 시스템 리소스가 있습니다. 권장 값은 VM의 의도한 작업 및 워크로드에 따라 크게 달라질 수 있습니다.

절차

1. 웹 콘솔의 **Virtual Machines(가상 시스템)** 인터페이스에서 **Create VM(VM 만들기)**을 클릭합니다.
Create new virtual machine(새 가상 머신 만들기) 대화 상자가 표시됩니다.

Create new virtual machine

Name

Unique name

Installation type

Download an OS

Operating system

Choose an operating system

Storage

Create new volume

Size

10

GiB

Up to 42.2 GiB available on the default location

Memory

1

GiB

Up to 3.6 GiB available on the host

Run unattended installation

☐

Immediately start VM

☒

Create

Cancel

2. 생성할 VM의 기본 구성을 입력합니다.

- **Name** (이름) - VM의 이름입니다.
- **connection** - libvirt 연결, 시스템 또는 세션 유형입니다. 자세한 내용은 [시스템 및 세션 연결](#)을 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 세트](#)에 대해서만 지원을 제공합니다.
- **Storage** (스토리지) - VM을 구성할 스토리지 유형입니다.
- **size** - VM을 구성할 스토리지 공간의 양입니다.
- **memory** - VM을 구성할 메모리 양입니다.
- **무인 설치 실행** - 무인 설치 실행 여부. 이 옵션은 **설치 유형이 OS 다운로드인** 경우에만 사용할 수 있습니다.
- **즉시 VM 시작** - VM이 생성된 직후 VM이 시작되거나 있는지 여부.

3. 생성을 클릭합니다.

VM이 생성됩니다. **Immediately Start VM**(VM 즉시 시작) 확인란이 선택되면 VM이 즉시 시작되고 게스트 운영 체제 설치를 시작합니다.

추가 리소스

- [VM에 운영 체제 설치](#)

2.2.2.2. 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 의도한 작업 및 워크로드에 따라 크게 달라질 수 있습니다.
- 기존 VM 설치를 위한 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** (이름) - VM의 이름입니다.
- **connection** - libvirt 연결, 시스템 또는 세션 유형입니다. 자세한 내용은 [시스템 및 세션 연결](#)을 참조하십시오.
- **디스크 이미지** - 호스트 시스템에 있는 VM의 기존 디스크 이미지 경로입니다.
- **운영 체제** - VM의 운영 체제. Red Hat은 [제한된 게스트 운영 체제 세트](#)에 대해서만 지원을 제공합니다.
- **memory** - VM을 구성할 메모리 양입니다.
- **VM 즉시 시작** - VM이 생성된 직후에 VM이 시작되는지 여부입니다.

3. **Import(가져오기)**를 클릭합니다.

2.2.2.3. 웹 콘솔을 사용하여 게스트 운영 체제 설치

VM(가상 머신)이 처음 로드될 때 VM에 운영 체제를 설치해야 합니다.



참고

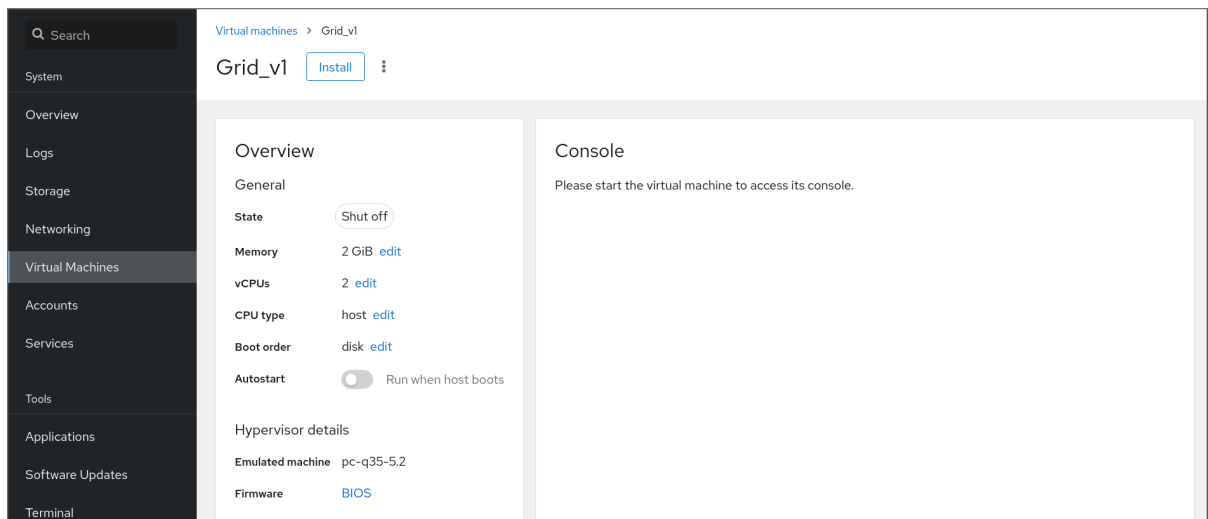
Create New Virtual Machine(새 가상 머신 생성) 대화 상자의 *Immediately Start VM* (VM 즉시 시작) 확인란이 선택되면 VM이 생성될 때 운영 체제의 설치 루틴이 자동으로 시작됩니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.
- 운영 체제를 설치할 VM을 사용할 수 있어야 합니다.

절차

1. **Virtual Machines** (가상 시스템) 인터페이스에서 게스트 OS를 설치할 VM을 클릭합니다.
선택한 VM에 대한 기본 정보와 VM의 다양한 측면을 관리하기 위한 컨트롤이 포함된 새 페이지가 열립니다.



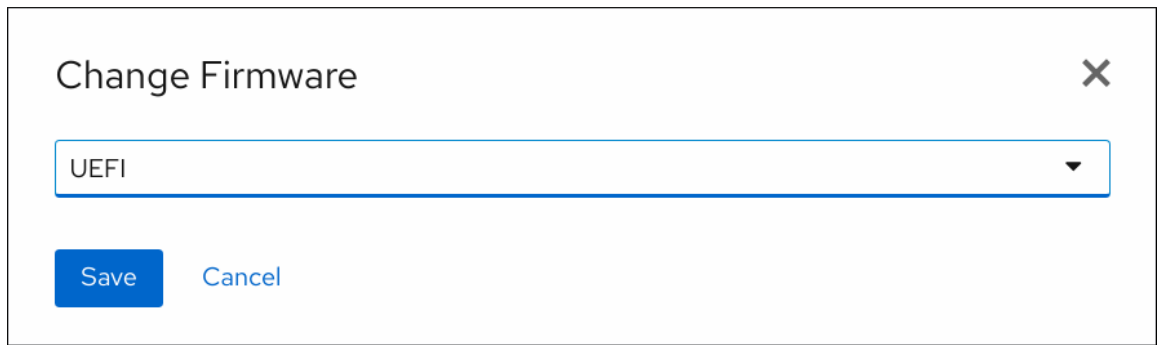
2. **선택 사항:** 펌웨어를 변경합니다.



참고

Create New Virtual Machine(새 가상 시스템 만들기) 대화 상자에서 *Immediately Start VM* (VM 즉시 시작) 확인란을 선택하지 않은 경우에만 펌웨어를 변경할 수 있으며, VM에 OS가 아직 설치되지 않은 경우에만 펌웨어를 변경할 수 있습니다.

- a. 펌웨어를 클릭합니다.
- b. Change 펌웨어(펌웨어 변경) 창에서 원하는 펌웨어를 선택합니다.



c. **저장**을 클릭합니다.

3. **설치**를 클릭합니다.

운영 체제의 설치 루틴은 VM 콘솔에서 실행됩니다.

문제 해결

- 설치 루틴이 실패하면 VM을 삭제하고 다시 생성해야 합니다.

2.2.3. 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.
- 가상화가 호스트 시스템에서 [활성화됩니다](#).
- 디스크 공간, RAM 또는 CPU와 같이 VM에 할당할 충분한 시스템 리소스가 있습니다. 권장 값은 VM의 의도한 작업 및 워크로드에 따라 크게 달라질 수 있습니다.

절차

1. 웹 콘솔의 **Virtual Machines(가상 시스템)** 인터페이스에서 **Create VM(VM 만들기)**을 클릭합니다.
Create new virtual machine(새 가상 머신 만들기) 대화 상자가 표시됩니다.

Create new virtual machine ✕

Name

Unique name

Installation type

Download an OS ▼

Operating system

Choose an operating system ▼

Storage

Create new volume ▼

Size

10

GiB ▼

Up to 42.2 GiB available on the default location

Memory

1

GiB ▼

Up to 3.6 GiB available on the host

Run unattended installation

☐

Immediately start VM

☒

Create

Cancel

- 이름 필드에 VM의 이름을 입력합니다.
- 설치 유형 필드에서 **Cloud base image**를 선택합니다.

Create new virtual machine

Name

VM-1

Installation type

Cloud base image

Installation source

/home/test

Operating system

Choose an operating system

Storage

Create new volume

Size

10

GiB

Up to 198.1 GiB available on the default location

Memory

1

GiB

Up to 15.2 GiB available on the host

☒ Set cloud init parameters

Root password

.....

Excellent password

User login

cloud-user

User password

.....

Excellent password

Create

Cancel

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 세트](#)에 대해서만 지원을 제공합니다.
- **Storage** (스토리지) - VM을 구성할 스토리지 유형입니다.
- **size** - VM을 구성할 스토리지 공간의 양입니다.
- **memory** - VM을 구성할 메모리 양입니다.

6. 클라우드 init 매개변수 설정을 선택합니다.

클라우드 인증 자격 증명을 설정합니다.

- **루트 암호** - VM의 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. **생성**을 클릭합니다.
VM이 생성됩니다.

추가 리소스

- [VM에 운영 체제 설치](#)

2.3. 가상 머신 시작

RHEL 8에서 VM(가상 머신)을 시작하려면 [명령줄 인터페이스](#) 또는 [웹 콘솔 GUI](#) 를 사용할 수 있습니다.

사전 요구 사항

- VM을 시작하기 전에 VM을 만들고, 이상적으로 OS를 사용하여 설치해야 합니다. 이를 수행하는 방법은 [가상 머신 생성](#)을 참조하십시오.

2.3.1. 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 권한.

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 호스트에 대한 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](#)

2.3.2. 웹 콘솔을 사용하여 가상 머신 시작

VM(가상 머신)이 종료된 상태인 경우 RHEL 8 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. **Run(실행)**을 클릭합니다.
VM이 시작되고 [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](#)

2.3.3. 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.

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

- a. **systemd** 구성 디렉터리 트리에서 **libvirtd.service.d** 디렉터리가 아직 없는 경우 생성합니다.

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

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

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

- c. **10-network-online.conf** 파일에 다음 행을 추가합니다. 이 구성 변경을 통해 **systemd**는 호스트의 네트워크가 준비된 후에만 **libvirtd** 서비스를 시작합니다.

```
[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. libvirt에서 관리하지 않는 네트워크 인터페이스를 사용하는 경우 **10-network-online.conf** 파일의 콘텐츠가 다음 출력과 일치하는지 확인합니다.

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

추가 리소스

- **virsh autostart --help** 명령
- [웹 콘솔을 사용하여 가상 머신 시작](#).

2.4. 가상 머신에 연결

RHEL 8의 VM(가상 머신)과 상호 작용하려면 다음 중 하나를 수행하여 연결해야 합니다.

- 웹 콘솔 인터페이스를 사용하는 경우 웹 콘솔 인터페이스의 Virtual Machines(가상 시스템) 창을 사용합니다. 자세한 내용은 [웹 콘솔을 사용하여 가상 머신 상호 작용](#)을 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 열기](#)를 참조하십시오.
- 그래픽 디스플레이를 사용할 수 없거나 필요하지 않은 경우 [SSH 터미널 연결](#)을 사용합니다.
- 네트워크를 사용하여 시스템에서 가상 시스템에 연결할 수 없는 경우 [virsh 콘솔](#)을 사용합니다.

연결하는 VM이 로컬 [호스트가 아닌 원격 호스트](#)에 있는 경우 선택적으로 원격 호스트에 대한 보다 편리한 [액세스](#)를 위해 시스템을 구성할 수 있습니다.

사전 요구 사항

- 상호 작용하려는 VM이 [설치 및 시작](#)됩니다.

2.4.1. 웹 콘솔을 사용하여 가상 머신과 상호 작용

RHEL 8 웹 콘솔에서 VM(가상 머신)과 상호 작용하려면 VM의 콘솔에 연결해야 합니다. 여기에는 그래픽 콘솔과 직렬 콘솔이 포함됩니다.

- 웹 콘솔에서 VM의 그래픽 인터페이스와 상호 작용하려면 [그래픽 콘솔](#)을 사용합니다.
- 원격 뷰어에서 VM의 그래픽 인터페이스와 상호 작용하려면 원격 뷰어 [에서 그래픽 콘솔](#)을 사용합니다.
- 웹 콘솔에서 VM의 CLI와 상호 작용하려면 [직렬 콘솔](#)을 사용합니다.

2.4.1.1. 웹 콘솔에서 가상 머신 그래픽 콘솔 보기

VM(가상 머신) 콘솔 인터페이스를 사용하여 RHEL 8 웹 콘솔에서 선택한 VM의 그래픽 출력을 볼 수 있습니다.

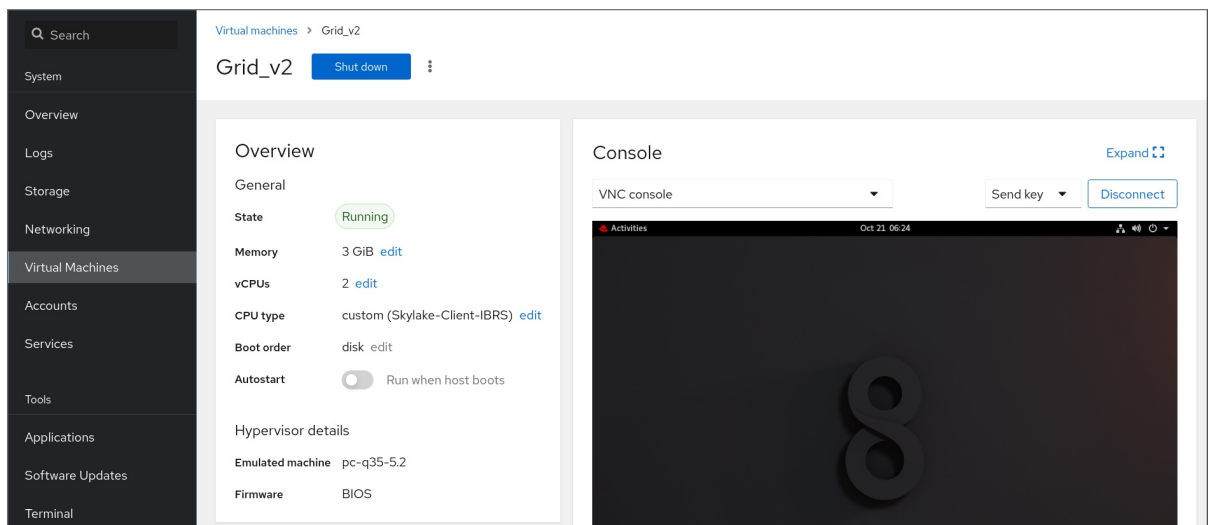
사전 요구 사항

- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.
- 호스트와 VM이 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)**를 클릭합니다.
이제 실제 시스템과 동일한 방식으로 마우스와 키보드를 사용하여 VM 콘솔과 상호 작용할 수 있습니다. VM 콘솔의 디스플레이는 VM에서 수행 중인 활동을 반영합니다.



참고

웹 콘솔이 실행 중인 호스트는 **Ctrl+Alt+Del** 과 같은 특정 키 조합을 가로채어 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

2.4.1.2. 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 콘솔 내에서 Virt Viewer를 시작할 수 있습니다. 기타 VNC 및 SPICE 원격 뷰어는 수동으로 시작할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.
- 호스트와 VM이 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.

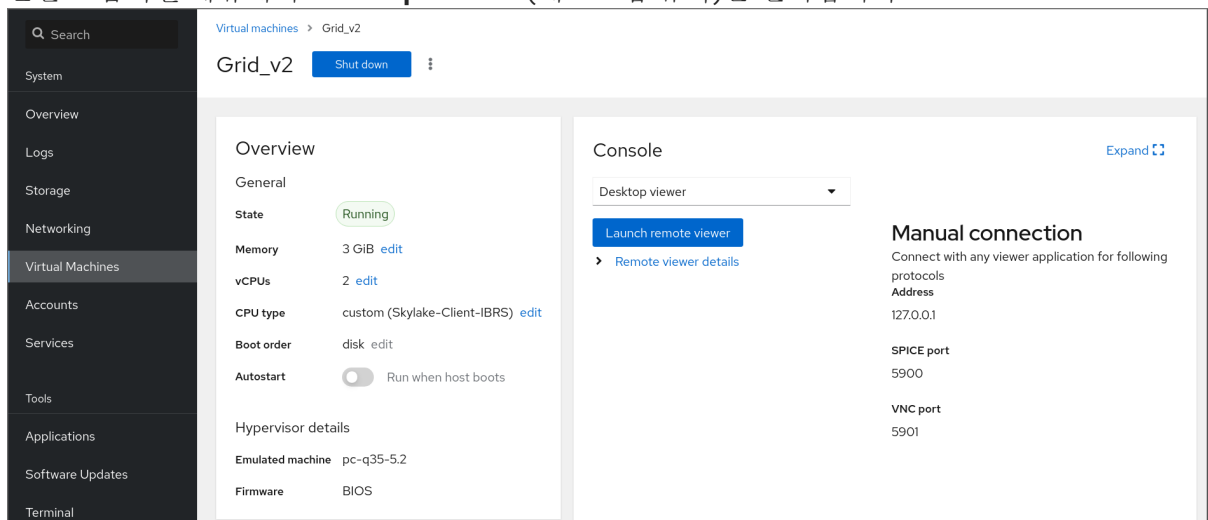


참고

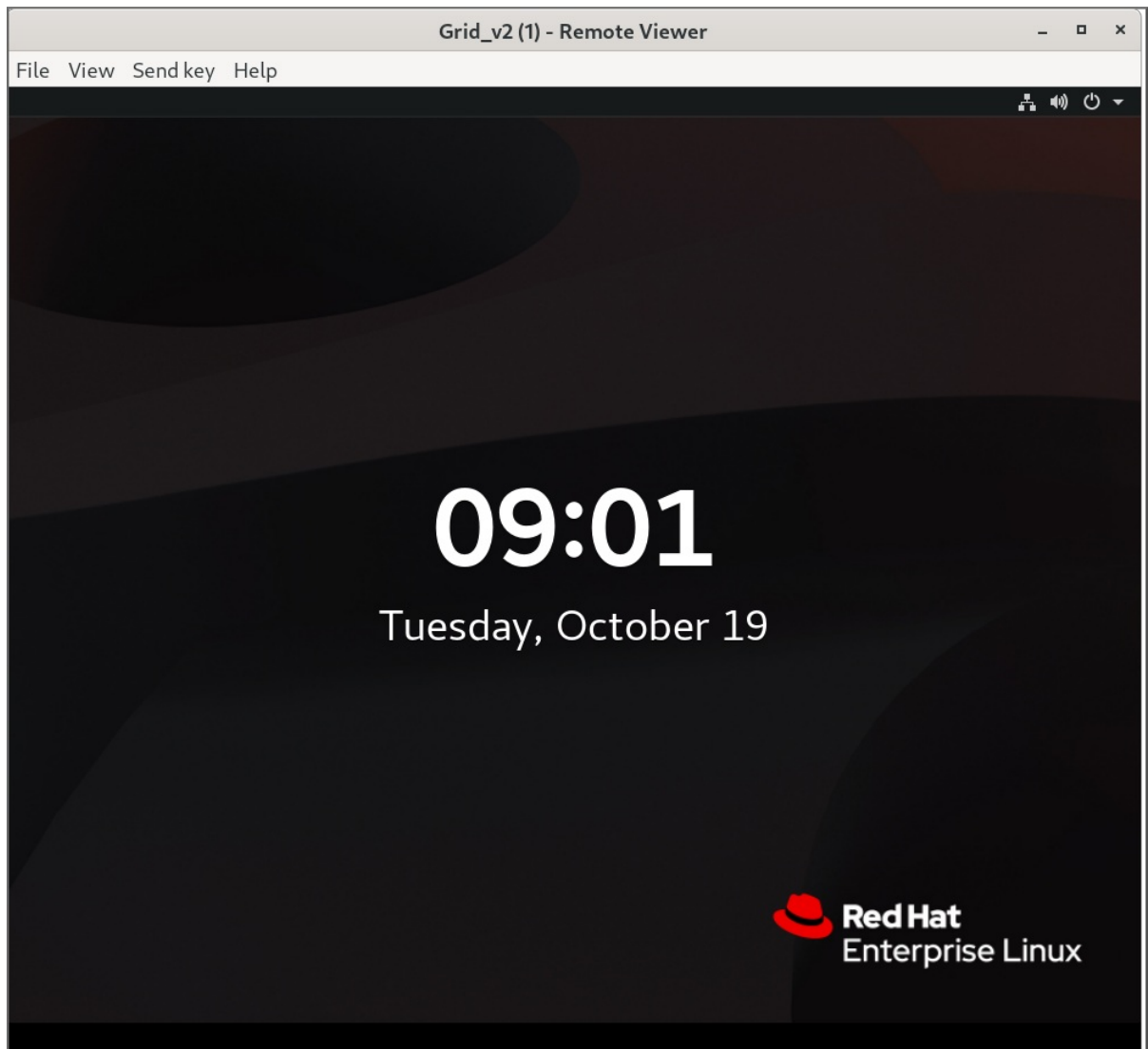
원격 뷰어는 대부분의 운영 체제에서 사용할 수 있습니다. 그러나 일부 브라우저 확장 및 플러그인에서는 웹 콘솔에서 Virt Viewer를 열 수 없습니다.

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 시작**을 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 VM에서 수행 중인 활동을 반영합니다.



참고

웹 콘솔이 실행 중인 서버는 **Ctrl+Alt+Del** 과 같은 특정 키 조합을 가로채어 VM으로 전송되지 않도록 할 수 있습니다.

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

예를 들어 **Ctrl+Alt+Del** 조합이 VM에 전송하려면 **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 Viewer를 시작하는 것이 작동하지 않거나 최적이지 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** 매개변수 또는 **spice_listen** 매개변수를 수정하여 호스트의 IP 주소로 변경할 수 있습니다.

- **SPICE 포트** - 5900
- **VNC 포트** - 5901

추가 리소스

- [웹 콘솔에서 가상 머신 그래픽 콘솔 보기](#)
- [웹 콘솔에서 가상 머신 직렬 콘솔 보기](#)

2.4.1.3. 웹 콘솔에서 가상 머신 직렬 콘솔 보기

RHEL 8 웹 콘솔에서 선택한 VM(가상 머신)의 직렬 콘솔을 볼 수 있습니다. 이 기능은 호스트 시스템 또는 VM이 그래픽 인터페이스로 구성되지 않은 경우 유용합니다.

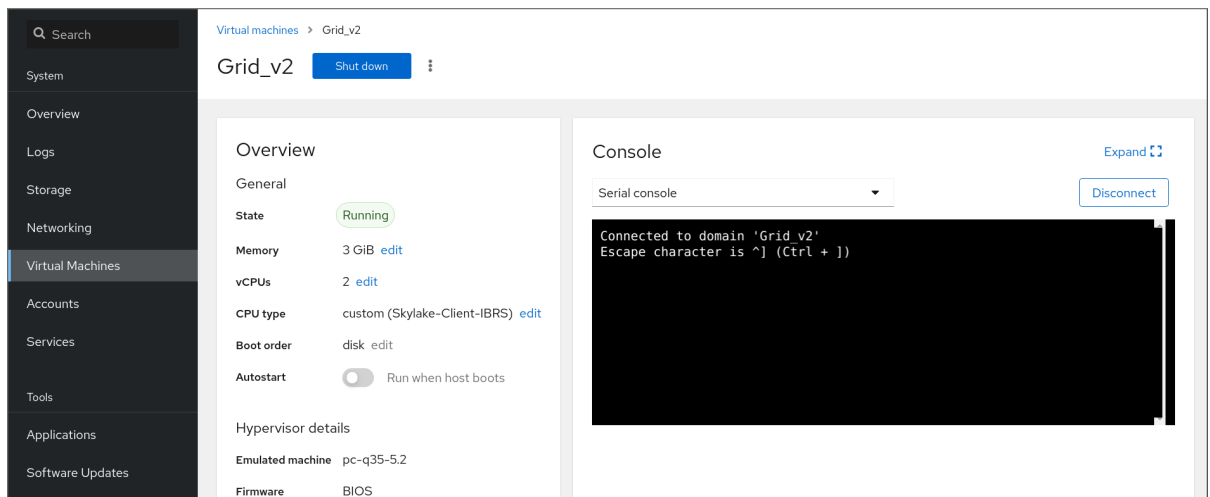
직렬 콘솔에 대한 자세한 내용은 [가상 머신 직렬 콘솔 열기](#)를 참조하십시오.

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 콘솔을 연결을 끊고 다시 연결할 수 있습니다.

- VM에서 직렬 콘솔을 연결 해제하려면 **Disconnect(연결 끊기)**를 클릭합니다.
- 직렬 콘솔을 VM에 다시 연결하려면 다시 연결을 클릭합니다.

추가 리소스

- [웹 콘솔에서 가상 머신 그래픽 콘솔 보기](#)
- [웹 콘솔을 사용하여 원격 뷰어에서 그래픽 콘솔 보기](#)

2.4.2. Virt Viewer를 사용하여 가상 머신 그래픽 콘솔 열기

KVM VM(가상 시스템)의 그래픽 콘솔에 연결하고 **Virt Viewer** 데스크탑 애플리케이션에서 열려면 아래 절차를 따르십시오.

사전 요구 사항

- 시스템과 연결 중인 VM은 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 연결하려는 VM의 이름으로 바꿉니다.

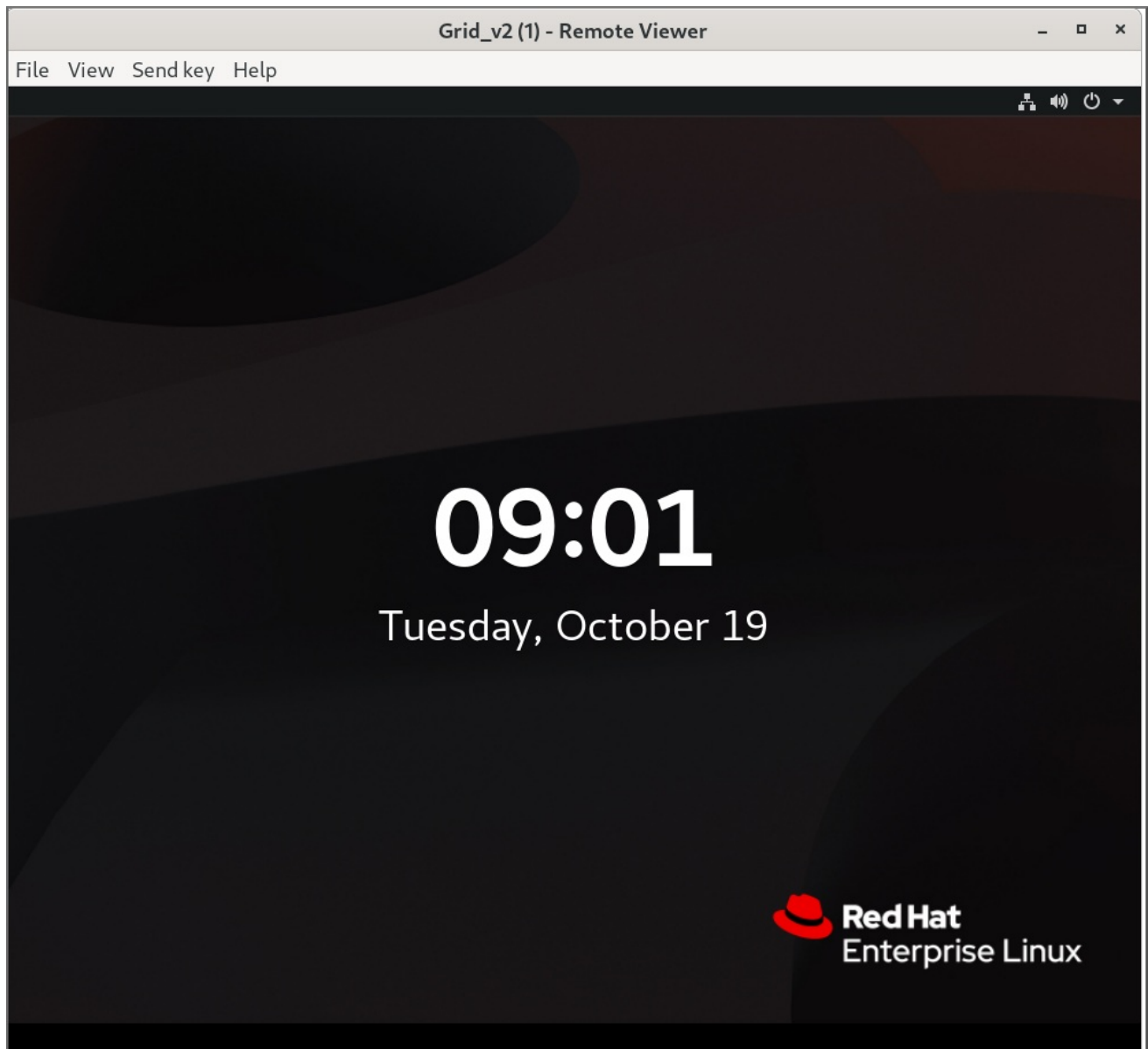
```
# virt-viewer guest-name
```

- 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. 연결에는 10.0.0.1의 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 작동하면 **Virt Viewer** (가상 뷰어) 창에 VM 디스플레이가 표시됩니다.



실제 시스템과 상호 작용하는 것과 동일한 방식으로 마우스와 키보드를 사용하여 VM 콘솔과 상호 작용할 수 있습니다. VM 콘솔의 디스플레이는 VM에서 수행 중인 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](#)

2.4.3. 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 있어야 합니다.
- 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 요소는 VM의 호스트에 설치 및 활성화됩니다. 그렇지 않은 경우 다음을 수행합니다.
 - a. **libvirt-nss** 패키지를 설치합니다.

```
# yum install libvirt-nss
```

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

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

절차

1. 원격 VM에 연결할 때 먼저 물리적 호스트에 SSH로 연결합니다. 다음 예제에서는 루트 자격 증명을 사용하여 호스트 머신 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 이름 및 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

- VM 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](#)

2.4.4. 가상 머신 직렬 콘솔 열기

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

이 기능은 VM에서 유용합니다.

- VNC 또는 SPICE 프로토콜을 제공하지 않으므로 GUI 도구에 대한 비디오 표시를 제공하지 않습니다.
- 예는 네트워크 연결이 없으므로 [SSH를 사용하여](#) 와 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>
```

- VM에는 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
```

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

- VM에 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 않도록 할 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* VM에 연결합니다.

```
# 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

2.4.5. 원격 가상화 호스트에 대한 손쉬운 액세스 설정

libvirt 유틸리티를 사용하여 원격 호스트 시스템에서 VM을 관리하는 경우 **-c** **qemu+ssh://root@hostname/system** 구문을 사용하는 것이 좋습니다. 예를 들어 10.0.0.1 호스트에서 **virsh list** 명령을 root로 사용하려면 다음을 수행합니다.

```
# 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

하지만 편의를 위해 SSH 및 libvirt 구성을 수정하여 연결 세부 정보를 완전히 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 사용하려면 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 다음은 `root@10.0.0.1`에 대해 `thetyrannosaurus` 별칭을 설정합니다.

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

2. `/etc/libvirt/libvirt.conf` 파일을 편집하거나 생성한 후 다음을 추가합니다. 여기서 `qemu-host-alias` 는 QEMU 및 libvirt 유틸리티가 의도한 호스트와 연관시킬 호스트 별칭입니다.

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

예를 들어, 다음은 이전 단계에서 구성한 `thetyrannosaurus` 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. 이렇게 하면 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 사용하려는 경우 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 관리하려는 경우에는 권장되지 않습니다.

- 원격 호스트에 연결할 때 원격 시스템에 루트 암호를 제공하지 않아도 됩니다. 이 작업을 수행하려면 다음 방법 중 하나 이상을 사용하십시오.
 - [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](#) 및 [virt-manager](#) 명령을 실행할 수 있습니다.

2.5. 가상 머신 종료

RHEL 8에서 호스팅되는 실행 중인 가상 머신 [을 종료하려면 명령줄 인터페이스](#) 또는 웹 콘솔 [GUI](#)를 사용합니다.

2.5.1. 명령줄 인터페이스를 사용하여 가상 머신 종료

응답하는 VM(가상 머신)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 예제에서는 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 실제로 VM 구성 또는 디스크 이미지를 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.

2.5.2. 웹 콘솔을 사용하여 가상 머신 종료 및 재시작

RHEL 8 웹 콘솔을 사용하여 실행 중인 가상 머신을 **종료** 하거나 **다시 시작** 할 수 있습니다. 또한 마스킹할 수 없는 인터럽트를 응답하지 않는 가상 머신으로 보낼 수도 있습니다.

2.5.2.1. 웹 콘솔에서 가상 머신 종료

VM(가상 머신)이 실행 중이면 RHEL 8 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 않으면 Shut Down(시스템 종료) 버튼 **옆에** 있는 Menu(메뉴) 버튼을 클릭하고 **Force Shut Down** (강제 종료)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](#)

2.5.2.2. 웹 콘솔을 사용하여 가상 머신 재시작

VM(가상 머신)이 실행 중이면 RHEL 8 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

- VM이 재시작되지 않는 경우 Restart 버튼 **옆에** 있는 Menu(메뉴) 버튼을 클릭하고 **Force Restart** (강제 다시 시작)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](#)

2.5.2.3. 웹 콘솔을 사용하여 VM에 마스킹할 수 없는 인터럽트 전송

마스킹할 수 없는 인터럽트(NMI)를 전송하면 VM(가상 시스템)이 응답하거나 종료될 수 있습니다. 예를 들어 표준 입력에 응답하지 않는 VM에 **Ctrl+Alt+Del** NMI를 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](#)

2.6. 가상 머신 삭제

RHEL 8에서 가상 머신을 삭제하려면 [명령줄 인터페이스](#) 또는 [웹 콘솔 GUI](#) 를 사용합니다.

2.6.1. 명령줄 인터페이스를 사용하여 가상 머신 삭제

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

사전 요구 사항

- VM에서 중요한 데이터를 백업합니다.
- VM을 종료합니다.
- 다른 VM이 동일한 관련 스토리지를 사용하지 않는지 확인합니다.

절차

- **virsh undefine** 유틸리티를 사용합니다.
예를 들어 다음 명령은 *guest1* VM, 관련 스토리지 볼륨 및 비발성 RAM(있는 경우)을 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

2.6.2. 웹 콘솔을 사용하여 가상 머신 삭제

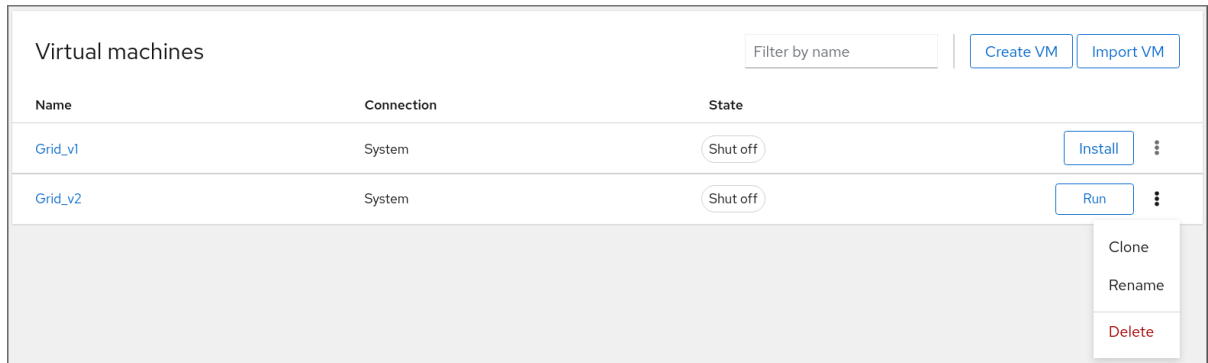
RHEL 8 웹 콘솔이 연결된 호스트에서 VM(가상 머신) 및 관련 스토리지 파일을 삭제하려면 다음 절차를 따르십시오.

사전 요구 사항

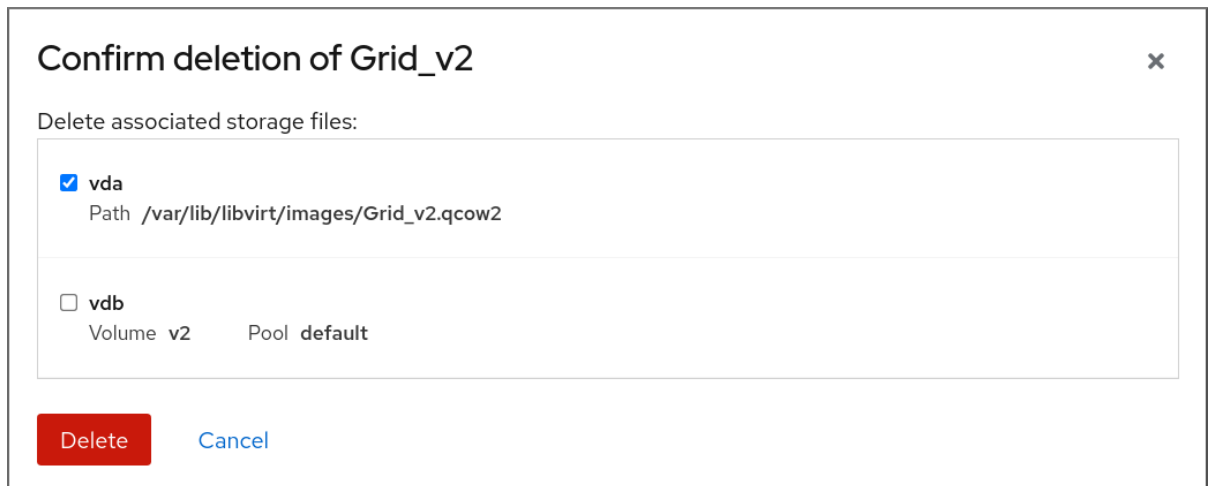
- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.
- VM에서 중요한 데이터를 백업합니다.
- 다른 VM이 동일한 연결된 스토리지를 사용하지 않는지 확인합니다.
- **선택 사항:** VM을 종료합니다.

절차

1. **Virtual Machines(가상 시스템)** 인터페이스에서 삭제할 VM 의 Menu(메뉴) 버튼 QCOW를 클릭합니다.
다양한 VM 작업에 대한 컨트롤이 포함된 드롭다운 메뉴가 나타납니다.



2. **삭제**를 클릭합니다.
확인 대화 상자가 나타납니다.



3. **선택 사항:** VM과 연결된 스토리지 파일 전체를 삭제하려면 삭제할 스토리지 파일 옆에 있는 확인란을 선택합니다.
4. **삭제**를 클릭합니다.
VM 및 선택한 스토리지 파일이 삭제됩니다.

2.7. 추가 리소스

- [IBM POWER에서 가상화 시작하기](#)
- [IBM Z에서 가상화 시작하기](#)

3장. IBM POWER에서 가상화 시작하기

IBM POWER8 또는 POWER9 하드웨어에서 RHEL 8을 사용할 때 KVM 가상화를 사용할 수 있습니다. 그러나 시스템에서 [KVM 하이퍼바이저를 활성화하려면](#) AMD64 및 Intel64 아키텍처의 가상화와 비교하여 추가 단계가 필요합니다. 특정 RHEL 8 가상화 기능은 IBM POWER에 대해 [다르거나 제한된 기능도](#) 있습니다.

다음 섹션의 정보 외에도 IBM POWER의 가상화를 사용하는 것은 AMD64 및 Intel 64와 동일하게 작동합니다. 따라서 IBM POWER에서 가상화를 사용할 때 자세한 내용은 다른 RHEL 8 가상화 설명서를 참조하십시오.

3.1. IBM POWER에서 가상화 활성화

RHEL 8을 실행하는 IBM POWER8 또는 IBM POWER9 시스템에 KVM 하이퍼바이저를 설정하고 VM(가상 시스템)을 생성하려면 아래 지침을 따르십시오.

사전 요구 사항

- RHEL 8은 호스트 시스템에 설치 및 등록되어 있습니다.
- 사용 가능한 최소 시스템 리소스는 다음과 같습니다.
 - 호스트용 디스크 여유 공간이 6GB, 의도한 각 VM에 대해 다른 6GB가 추가됩니다.
 - 호스트용 2GB RAM 및 각 VM에 대해 다른 2GB.
 - 호스트의 CPU 4개. VM은 일반적으로 단일 할당된 vCPU로 실행할 수 있지만, 부하가 높은 동안 VM이 응답하지 않도록 Red Hat은 VM당 2개 이상의 vCPU를 할당하는 것이 좋습니다.
- CPU 시스템 유형은 IBM POWER 가상화를 지원해야 합니다. 이를 확인하려면 `/proc/cpuinfo` 파일에서 플랫폼 정보를 쿼리합니다.

```
# grep ^platform /proc/cpuinfo/
platform      : PowerNV
```

이 명령의 출력에 **PowerNV** 항목이 포함된 경우 PowerNV 시스템 유형을 실행하고 IBM POWER에서 가상화를 사용할 수 있습니다.

절차

1. KVM-HV 커널 모듈 로드

```
# modprobe kvm_hv
```

2. KVM 커널 모듈이 로드되었는지 확인합니다.

```
# lsmod | grep kvm
```

KVM이 성공적으로 로드되면 이 명령의 출력에 **kvm_hv**가 포함됩니다.

3. 가상화 모듈에 패키지를 설치합니다.

```
# yum module install virt
```

4. **virt-install** 패키지를 설치합니다.

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

5. **libvirt** 서비스를 시작합니다.

```
# systemctl start libvirt
```

검증

1. 시스템이 가상화 호스트가 될 준비가 되었는지 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        : PASS
```

2. 모든 **virt-host-validate** 검사에서 **PASS** 값을 반환하면 시스템에서 **VM을 생성할** 준비가 됩니다. 검사 중 **실패** 값을 반환하는 경우 표시된 지침에 따라 문제를 해결합니다.

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

문제 해결

- 호스트 CPU에서 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 VM은 성능 문제가 아닌 부팅에 실패합니다.

이 문제를 해결하려면 VM의 XML 구성에서 **<domain type>** 값을 **qemu** 로 변경할 수 있습니다. 그러나 Red Hat은 **qemu** 도메인 유형을 사용하는 VM을 지원하지 않으며 프로덕션 환경에서는 권장되지 않습니다.

3.2. IBM POWER의 가상화가 AMD64 및 INTEL 64와 다른 방법

IBM POWER 시스템의 RHEL 8의 KVM 가상화는 AMD64 및 Intel 64 시스템의 KVM과 크게 다릅니다.

메모리 요구 사항

IBM POWER의 VM은 더 많은 메모리를 사용합니다. 따라서 IBM POWER 호스트의 VM(가상 시스템)에 권장되는 최소 메모리 할당은 2GB RAM입니다.

표시 프로토콜

SPICE 프로토콜은 IBM POWER 시스템에서 지원되지 않습니다. VM의 그래픽 출력을 표시하려면 **VNC** 프로토콜을 사용합니다. 또한 다음 가상 그래픽 카드 장치만 지원됩니다.

- **cd - -vga std** 모드에서만 지원되며 **-vga cirrus** 모드에서는 지원되지 않습니다.
- **virtio-vga**
- **virtio-gpu**

SMBIOS

SMBIOS 구성을 사용할 수 없습니다.

메모리 할당 오류

POWER8 VM(호환 모드 VM 포함)은 다음과 유사한 오류로 인해 실패할 수 있습니다.

```
qemu-kvm: Failed to allocate KVM HPT of order 33 (try smaller maxmem?): Cannot allocate memory
```

이는 RHEL 7.3 이전과 게스트 OS 이전의 VM에서 발생할 가능성이 훨씬 더 높습니다.

문제를 해결하려면 호스트의 커널 명령줄에 **kvm_cma_resv_ratio=** 메모리를 추가하여 게스트의 **해시된 페이지 테이블(HPT)**에 사용할 수 있는 **CMA메모리** 풀을 늘립니다. 여기서 **메모리**는 CMA 풀에 예약해야 하는 호스트 메모리의 백분율입니다(기본값: 5).

Huge Page

THP(Transparent Huge Page)는 IBM POWER8 VM에서 주목할 만한 성능 이점을 제공하지 않습니다. 그러나 IBM POWER9 VM은 THP를 예상대로 활용할 수 있습니다.

또한 IBM POWER8 시스템의 정적 대규모 페이지 크기는 AMD64, Intel 64 및 IBM POWER9의 경우 2MiB 및 1GiB와 달리 16MiB 및 16GiB입니다. 결과적으로 IBM POWER8 호스트에서 IBM POWER9 호스트로 정적 대규모 페이지로 구성된 VM을 마이그레이션하려면 먼저 VM에서 1GiB 대규모 페이지를 설정해야 합니다.

kvm-clock

kvm-clock 서비스는 IBM POWER9의 VM의 시간 관리를 위해 구성하지 않아도 됩니다.

pvpanic

IBM POWER9 시스템은 **pvpanic** 장치를 지원하지 않습니다. 그러나 이 아키텍처에서 기본적으로 동등한 기능을 사용할 수 있으며 활성화됩니다. VM에서 활성화하려면 **유지** 값과 함께 **<on_crash>** XML 구성 요소를 사용합니다.

또한 VM이 IBM POWER 시스템에서 부팅되지 않을 수 있으므로 **<devices>** 섹션에서 **<panic>** 요소를 제거해야 합니다.

단일 스레드 호스트

IBM POWER8 시스템에서 VM을 지원하려면 호스트 시스템을 **단일 스레드 모드**로 실행해야 합니다. **qemu-kvm** 패키지가 설치된 경우 자동으로 구성됩니다. 그러나 단일 스레드 호스트에서 실행되는 VM은 여전히 여러 스레드를 사용할 수 있습니다.

주변 장치

IBM POWER 시스템에서 AMD64 및 Intel 64 시스템에서 지원되는 여러 주변 장치가 지원되지 않거나 다른 장치가 대체로 지원됩니다.

- **ic3420** 및 **xio3130-downstream**을 포함한 **PCI-E** 계층 구조에 사용되는 장치는 지원되지 않습니다. 이 기능은 **spapr-pci-host-bridge** 장치에서 제공하는 여러 독립 PCI 루트 브리지로 교체됩니다.
- UHCI 및 EHCI PCI 컨트롤러는 지원되지 않습니다. 대신 OHCI 및 XHCI 컨트롤러를 사용합니다.

- 가상 IDE CD-ROM(**ide-cd**) 및 가상 IDE 디스크(**ide-hd**)를 포함하는 IDE 장치는 지원되지 않습니다. 대신 **virtio-scsi** 및 **virtio-blk** 장치를 사용합니다.
- 에뮬레이트된 PCI NIC(**rtl8139**)는 지원되지 않습니다. 대신 **virtio-net** 장치를 사용합니다.
- **intel-hda**, **hda-output**, **AC97** 을 포함한 사운드 장치는 지원되지 않습니다.
- **usb-redir** 및 **usb-tablet** 를 포함한 USB 리디렉션 장치는 지원되지 않습니다.

v2v 및 p2v

virt-v2v 및 **virt-p2v** 유틸리티는 AMD64 및 Intel 64 아키텍처에서만 지원되며 IBM POWER에서는 제공되지 않습니다.

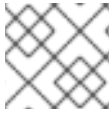
추가 소스

- Red Hat에서 지원하는 시스템 아키텍처에서 지원되는 일부 지원 및 지원되지 않는 가상화 기능을 비교하려면 [RHEL 8의 가상화 기능 개요](#) 를 참조하십시오.

4장. IBM Z에서 가상화 시작하기

IBM Z 하드웨어에서 RHEL 8을 사용할 때 KVM 가상화를 사용할 수 있습니다. 그러나 시스템에서 [KVM 하이퍼바이저를 활성화하려면](#) AMD64 및 Intel 64 아키텍처의 가상화와 비교하여 추가 단계가 필요합니다. 특정 RHEL 8 가상화 기능에는 IBM Z에서도 [다르거나 제한된 기능이](#) 있습니다.

다음 섹션의 정보 외에도 IBM Z의 가상화를 사용하는 것은 AMD64 및 Intel 64와 동일하게 작동합니다. 따라서 IBM Z에서 가상화를 사용할 때 자세한 내용은 다른 RHEL 8 가상화 설명서를 참조하십시오.



참고

z/VM OS에서 KVM을 실행하는 것은 지원되지 않습니다.

4.1. IBM Z에서 가상화 활성화

RHEL 8을 실행하는 IBM Z 시스템에 KVM 하이퍼바이저를 설정하고 VM(가상 머신)을 생성하려면 다음 지침을 따르십시오.

사전 요구 사항

- RHEL 8.6 이상이 호스트 머신에 설치되고 등록됩니다.



중요

RHEL 8.5 이하를 사용하여 IBM Z 시스템에서 가상화를 이미 활성화한 경우 대신 가상화 모듈을 재구성하고 시스템을 업데이트해야 합니다. 자세한 내용은 [IBM Z의 가상화가 AMD64 및 Intel 64와 어떻게 다른지](#) 참조하십시오.

- 사용 가능한 최소 시스템 리소스는 다음과 같습니다.
 - 호스트용 디스크 여유 공간이 6GB, 의도한 각 VM에 대해 다른 6GB가 추가됩니다.
 - 호스트용 2GB RAM 및 각 VM에 대해 다른 2GB.
 - 호스트의 CPU 4개. VM은 일반적으로 단일 할당된 vCPU로 실행할 수 있지만, 부하가 높은 동안 VM이 응답하지 않도록 Red Hat은 VM당 2개 이상의 vCPU를 할당하는 것이 좋습니다.
- IBM Z 호스트 시스템은 z13 CPU 이상을 사용하고 있습니다.
- RHEL 8은 LVR(논리 파티션)에 설치됩니다. 또한 LPAR은 *시작 인터프리티브 실행 (SIE)* 가상화 기능을 지원합니다.
이를 **확인하려면** `/proc/cpuinfo` 파일에서 `te sie`를 검색합니다.

```
# grep sie /proc/cpuinfo/
```

```
features      : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te sie
```

절차

1. KVM 커널 모듈을 로드합니다.

```
# modprobe kvm
```

2. KVM 커널 모듈이 로드되었는지 확인합니다.

```
# lsmod | grep kvm
```

KVM이 성공적으로 로드되면 이 명령의 출력에 **kvm** 이 포함됩니다.

3. **virt:rhel/common** 모듈에 패키지를 설치합니다.

```
# yum module install virt:rhel/common
```

4. 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 **PASS** 값을 반환하면 시스템에서 **VM을 생성할** 준비가 됩니다. 검사 중 **실패** 값을 반환하는 경우 표시된 지침에 따라 문제를 해결합니다.

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

문제 해결

- 호스트 CPU에서 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 VM은 성능 문제가 아닌 부팅에 실패합니다.

이 문제를 해결하려면 VM의 XML 구성에서 **<domain type>** 값을 **qemu** 로 변경할 수 있습니다. 그러나 Red Hat은 **qemu** 도메인 유형을 사용하는 VM을 지원하지 않으며 프로덕션 환경에서는 권장되지 않습니다.

4.2. IBM Z의 가상화 업데이트 RHEL 8.5에서 RHEL 8.6 이상으로

RHEL 8.6 이전의 IBM Z 하드웨어에 RHEL 8을 설치하는 경우 RHEL 8의 기본 RPM 스트림과는 별도로 AV 스트림에서 가상화 RPM을 가져와야 했습니다. RHEL 8.6부터 이전에는 AV 스트림에서 사용 가능한 가상화 RPM은 기본 RHEL 스트림에서만 사용할 수 있습니다. 또한 RHEL 8의 향후 마이너 릴리스에서 AV 스트림이 중단될 예정입니다. 따라서 AV 스트림을 사용하는 것은 더 이상 권장되지 않습니다.

아래 지침에 따라 AV 스트림을 비활성화하고 RHEL 8.6 이상 버전에서 사용 가능한 가상화 RPM에 대한 액세스를 활성화합니다.

사전 요구 사항

- **virt:av** 모듈이 설치된 IBM Z에서 RHEL 8.5를 사용하고 있습니다. 이 경우인지 확인하려면 다음을 수행합니다.

```
# hostnamectl | grep "Operating System"
Operating System: Red Hat Enterprise Linux 8.5 (Ootpa)
# yum module list --installed
[...]
Advanced Virtualization for RHEL 8 IBM Z Systems (RPMs)
Name           Stream          Profiles         Summary
virt           av [e]          common [i]       Virtualization module
```

절차

1. **virt:av** 모듈을 비활성화합니다.

```
# yum disable virt:av
```

2. 시스템에 이미 포함된 기존 가상화 패키지 및 모듈을 제거합니다.

```
# yum module reset virt -y
```

3. 패키지를 최신 RHEL 버전으로 업그레이드합니다.

```
# yum update
```

이렇게 하면 시스템에서 **virt:rhel** 모듈도 자동으로 활성화됩니다.

검증

- 시스템의 **virt** 모듈이 **rhel** 스트림에서 제공되는지 확인합니다.

```
# yum module info virt

Name           : virt
Stream          : rhel [d][e][a]
Version        : 8050020211203195115
[...]
```

추가 리소스

- [IBM Z의 가상화가 AMD64 및 Intel 64와 다른 방식](#)

4.3. IBM Z의 가상화가 AMD64 및 INTEL 64와 어떻게 다른지

IBM Z 시스템의 RHEL 8에서 KVM 가상화는 AMD64 및 Intel 64 시스템의 KVM과 다릅니다.

PCI 및 USB 장치

가상 PCI 및 USB 장치는 IBM Z에서 지원되지 않습니다. 또한 **virtio-**pci*** 장치가 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 첨부이 지원됩니다.

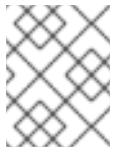
지원되는 게스트 OS

Red Hat은 RHEL 7, 8 또는 9를 게스트 운영 체제로 사용하는 경우에만 IBM Z에서 호스팅되는 VM을 지원합니다.

장치 부팅 순서

IBM Z는 **<boot dev=*장치*>** XML 구성 요소를 지원하지 않습니다. 장치 부팅 순서를 정의하려면 XML의 **<devices>** 섹션에 **<boot order=*번호*>** 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 것도 AMD64 및 Intel 64 호스트에서 선호됩니다.

메모리 핫 플러그

IBM Z에서는 실행 중인 VM에 메모리를 추가할 수 없습니다. 실행 중인 VM(메모리 핫 언플러그)에서 메모리를 제거하는 것은 IBM Z에서도 사용할 수 없으며 AMD64 및 Intel 64에서도 사용할 수 없습니다.

NUMA 토폴로지

CPU용 NUMA(Non-Uniform Memory Access) 토폴로지는 IBM Z의 **libvirt** 에서 지원하지 않습니다. 따라서 이러한 시스템에서 NUMA를 사용하여 vCPU 성능 튜닝을 수행할 수 없습니다.

vfio-ap

IBM Z 호스트의 VM은 다른 아키텍처에서 지원되지 않는 **vfio-ap** 암호화 장치 패스스루를 사용할 수 있습니다.

SMBIOS

IBM Z에서는 SMBIOS 구성을 사용할 수 없습니다.

위치독 장치

IBM Z 호스트에서 VM에서 위치독 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 고유하며 IBM Z의 VM 시간 관리를 위해 구성하지 않아도 됩니다.

v2v 및 p2v

virt-v2v 및 **virt-p2v** 유틸리티는 AMD64 및 Intel 64 아키텍처에서만 지원되며 IBM Z에서는 제공되지 않습니다.

중첩된 가상화

중첩된 VM을 생성하려면 AMD64 및 Intel 64와 IBM Z의 다른 설정이 필요합니다. 자세한 내용은 [중첩된 가상 머신 생성](#)을 참조하십시오.

이전 릴리스에서는 그래픽 출력이 없습니다

RHEL 8.3 또는 호스트의 이전 마이너 버전을 사용하는 경우 VNC 프로토콜을 사용하여 VM에 연결할 때 VM 그래픽 출력을 표시할 수 없습니다. 이는 IBM Z의 이전 RHEL 버전에서 **gnome-desktop** 유틸리티가 지원되지 않기 때문입니다. 또한 SPICE 디스플레이 프로토콜은 IBM Z에서 작동하지 않기 때문입니다.

마이그레이션

이후 호스트 모델로 마이그레이션(예: IBM z14에서 z15로 마이그레이션)하거나 하이퍼바이저를 업데이트하려면 **host-model** CPU 모드를 사용합니다. 일반적으로 마이그레이션 안전이 아니므로 **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) 또는 이전 버전의 QEMU, KVM 또는 RHEL 커널로 마이그레이션하려면 끝에 **-base** 없이 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](#)

4.4. 다음 단계

- IBM Z 시스템에 VM을 설정할 때는 게스트 OS가 "Spectre" 취약점으로부터 보호하는 것이 좋습니다. 이렇게 하려면 **virsh edit** 명령을 사용하여 VM의 XML 구성을 수정하고 다음 방법 중 하나로 CPU를 구성합니다.

- 호스트 CPU 모델을 사용합니다.

```
<cpu mode='host-model' check='partial'>
  <model fallback='allow'/>
</cpu>
```

이렇게 하면 호스트에서 지원하는 경우 **ppa15** 및 **bpb** 기능을 게스트에서 사용할 수 있습니다.

- 특정 호스트 모델을 사용하는 경우 **ppa15** 및 **pbb** 기능을 추가합니다. 다음 예제에서는 zEC12 CPU 모델을 사용합니다.

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>zEC12</model>
  <feature policy='force' name='ppa15'/>
```

```
<feature policy='force' name='bpb' />  
</cpu>
```

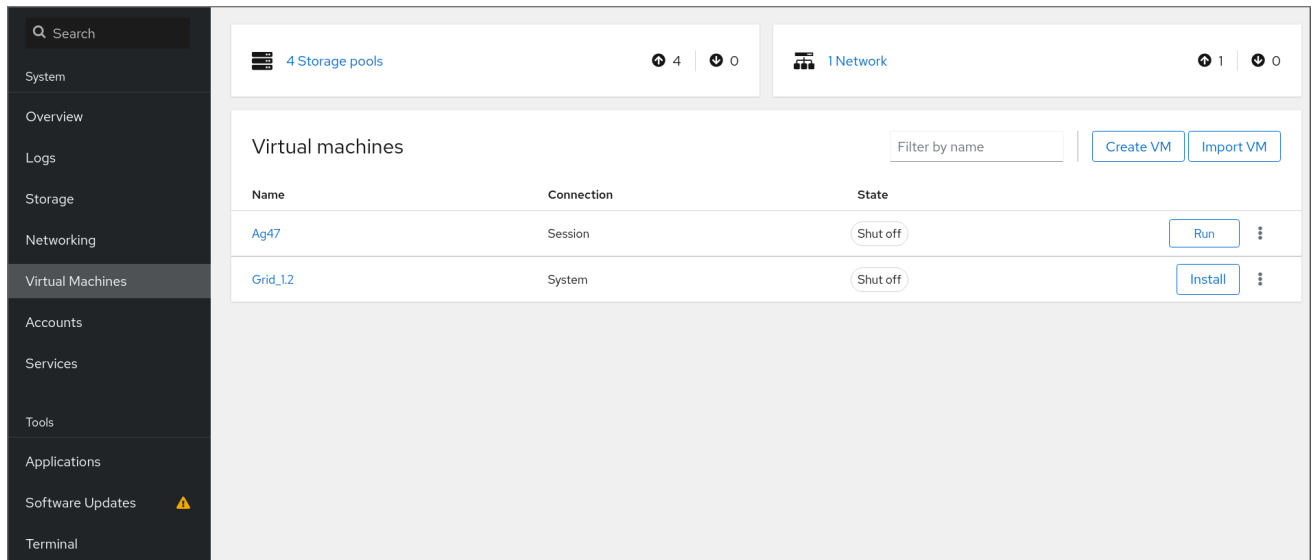
z12 CPU를 사용하는 호스트 시스템에서 **z114** 및 **z196** CPU 모델과 함께 **ppa15** 기능을 사용할 때는 최신 마이크로 코드 수준(bundle 95 이상)도 사용해야 합니다.

4.5. 추가 리소스

- IBM Z의 가상 머신에 DASD 장치 연결
- IBM Z의 가상 머신에 암호화 coprocessors 연결
- IBM Z에서 IBM Secure Execution 설정
- IBM Z에 중첩된 가상 머신 생성

5장. 웹 콘솔에서 가상 머신 관리

RHEL 8 호스트의 그래픽 인터페이스에서 가상 머신을 관리하려면 RHEL 8 웹 콘솔에서 **Virtual Machines** 창을 사용할 수 있습니다.



5.1. 웹 콘솔을 사용한 가상 머신 관리 개요

RHEL 8 웹 콘솔은 시스템 관리를 위한 웹 기반 인터페이스입니다. 웹 콘솔은 해당 기능 중 하나로 호스트 시스템에서 VM(가상 시스템)의 그래픽 보기를 제공하고 이러한 VM을 생성, 액세스 및 구성할 수 있도록 합니다.

웹 콘솔을 사용하여 RHEL 8에서 VM을 관리하려면 먼저 가상화 [를 위한 웹 콘솔 플러그인](#)을 설치해야 합니다.

다음 단계

- 웹 콘솔에서 VM 관리를 활성화하는 방법에 대한 지침은 [가상 시스템을 관리하기 위한 웹 콘솔 설정](#)을 참조하십시오.
- 웹 콘솔에서 제공하는 VM 관리 작업의 전체 목록은 웹 콘솔에서 [사용할 수 있는 가상 시스템 관리 기능을 참조하십시오](#).
- 현재 웹 콘솔에서는 사용할 수 없지만 **virt-manager** 애플리케이션에서 사용할 수 있는 기능 목록은 [Virtual Machine Manager의 가상화 기능과 웹 콘솔 간의 차이점](#)을 참조하십시오.

5.2. 가상 머신 관리를 위해 웹 콘솔 설정

RHEL 8 웹 콘솔을 사용하여 VM(가상 머신)을 관리하기 전에 호스트에 웹 콘솔 가상 머신 플러그인을 설치해야 합니다.

사전 요구 사항

- 웹 콘솔이 시스템에 설치 및 활성화되어 있는지 확인합니다.

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

이 명령이 **Unit cockpit.socket**을 찾을 수 없는 경우 [웹 콘솔 설치](#) 문서에 따라 웹 콘솔을 활성화합니다.

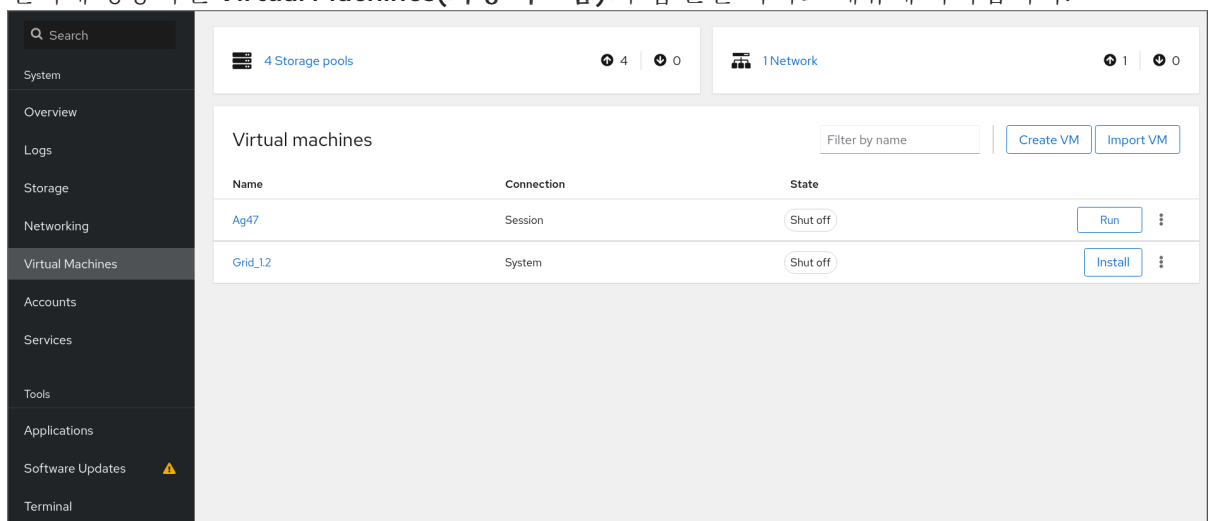
절차

- **cockpit-machines** 플러그인을 설치합니다.

```
# yum 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 8 웹 콘솔을 사용하여 시스템 관리](#)

5.3. 웹 콘솔을 사용하여 가상 머신 이름 변경

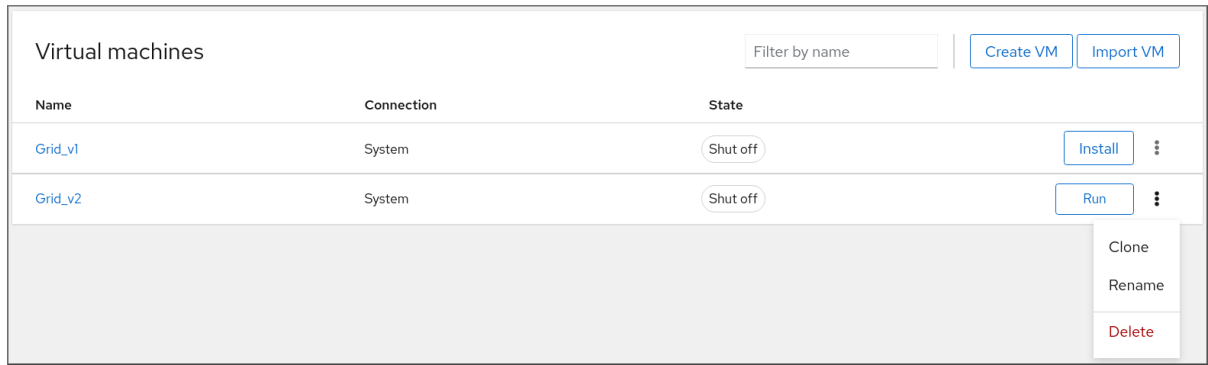
VM(가상 머신)을 생성한 후 충돌을 피하도록 VM의 이름을 바꾸거나 사용 사례에 따라 새로운 고유 이름을 할당할 수 있습니다. RHEL 웹 콘솔을 사용하여 VM의 이름을 변경할 수 있습니다.

사전 요구 사항

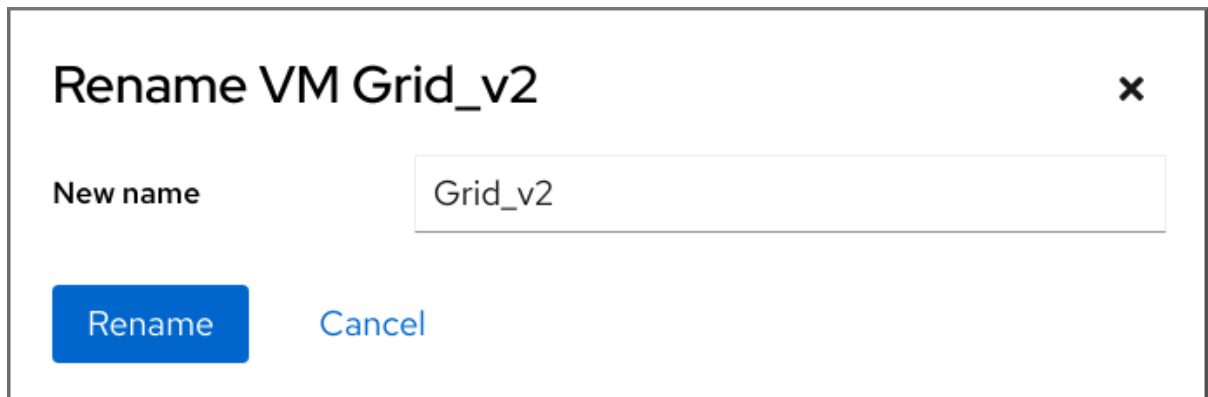
- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.
- VM이 종료되었는지 확인합니다.

절차

1. **Virtual Machines(가상 머신)** 인터페이스에서 이름을 변경할 VM의 메뉴 버튼을 클릭합니다. 다양한 VM 작업에 대한 컨트롤이 포함된 드롭다운 메뉴가 나타납니다.



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.

5.4. 웹 콘솔에서 사용할 수 있는 가상 머신 관리 기능

RHEL 8 웹 콘솔을 사용하면 다음 작업을 수행하여 시스템의 VM(가상 머신)을 관리할 수 있습니다.

표 5.1. RHEL 8 웹 콘솔에서 수행할 수 있는 VM 관리 작업

Task	자세한 내용은 를 참조하십시오.
VM을 생성하고 게스트 운영 체제로 설치합니다.	웹 콘솔을 사용하여 가상 머신 생성 및 게스트 운영 체제 설치
VM 삭제	웹 콘솔을 사용하여 가상 머신 삭제
VM을 시작, 종료, 다시 시작	웹 콘솔 을 사용하여 가상 머신 시작 및 웹 콘솔을 사용하여 가상 머신 종료 및 재시작
다양한 콘솔을 사용하여 VM과 연결 및 상호 작용	웹 콘솔을 사용하여 가상 머신과 상호 작용
VM에 대한 다양한 정보보기	웹 콘솔을 사용하여 가상 머신 정보 보기

Task	자세한 내용은 를 참조하십시오.
VM에 할당된 호스트 메모리 조정	웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거
VM의 네트워크 연결을 관리합니다	가상 머신 네트워크 인터페이스를 관리하기 위해 웹 콘솔 사용
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 virtual CPU 관리
VM 실시간 마이그레이션	웹 콘솔을 사용하여 가상 머신 실시간 마이그레이션
호스트 장치 관리	웹 콘솔을 사용하여 호스트 장치 관리

5.5. VIRTUAL MACHINE MANAGER와 웹 콘솔의 가상화 기능 간 차이점

virt-manager(가상 머신 관리자) 애플리케이션은 RHEL 8에서 지원되지만 더 이상 사용되지 않습니다. 웹 콘솔은 후속 주요 릴리스에서 교체될 예정입니다. 따라서 GUI에서 가상화를 관리하는 데 웹 콘솔에 익숙해지는 것이 좋습니다.

그러나 RHEL 8에서는 일부 VM 관리 작업은 **virt-manager** 또는 명령줄에서만 수행할 수 있습니다. 다음 표에서는 **virt-manager** 에서 사용할 수 있지만 RHEL 8.0 웹 콘솔에서 사용할 수 없는 기능을 강조 표시합니다.

이후 마이너 버전의 RHEL 8에서 기능을 사용할 수 있는 경우 웹 콘솔 도입 열에 최소 RHEL 8 버전이 표시됩니다.

표 5.2. RHEL 8.0에서 웹 콘솔을 사용하여 수행할 수 없는 VM management 작업

Task	웹 콘솔 지원 도입	CLI를 사용하는 대체 방법
호스트가 부팅될 때 시작하도록 가상 머신 설정	RHEL 8.1	virsh 자동 시작
가상 머신 일시 중단	RHEL 8.1	virsh 일시 중지
일시 중단된 가상 머신 재시작	RHEL 8.1	virsh 재개
파일 시스템 디렉터리 스토리지 풀 생성	RHEL 8.1	virsh pool-define-as
NFS 스토리지 풀 생성	RHEL 8.1	virsh pool-define-as
물리적 디스크 장치 스토리지 풀 생성	RHEL 8.1	virsh pool-define-as
LVM 볼륨 그룹 스토리지 풀 생성	RHEL 8.1	virsh pool-define-as

Task	웹 콘솔 지원 도입	CLI를 사용하는 대체 방법
파티션 기반 스토리지 풀 생성	현재 사용할 수 없습니다	virsh pool-define-as
GlusterFS 기반 스토리지 풀 생성	현재 사용할 수 없습니다	virsh pool-define-as
SCSI 장치를 사용하여 vHBA 기반 스토리지 풀 생성	현재 사용할 수 없습니다	virsh pool-define-as
다중 경로 기반 스토리지 풀 생성	현재 사용할 수 없습니다	virsh pool-define-as
RBD 기반 스토리지 풀 생성	현재 사용할 수 없습니다	virsh pool-define-as
새 스토리지 볼륨 생성	RHEL 8.1	virsh vol-create
새 가상 네트워크 추가	RHEL 8.1	virsh net-create 또는 virsh net-define
가상 네트워크 삭제	RHEL 8.1	virsh net-undefine
호스트 머신 인터페이스에서 가상 머신으로 브리지 만들기	현재 사용할 수 없습니다	virsh iface-bridge
스냅샷 생성	현재 사용할 수 없습니다	virsh snapshot-create-as
스냅샷으로 되돌리기	현재 사용할 수 없습니다	virsh snapshot-revert
스냅샷 삭제	현재 사용할 수 없습니다	virsh snapshot-delete
가상 머신 복제	RHEL 8.4	virt-clone
다른 호스트 머신으로 가상 머신 마이그레이션	RHEL 8.5	virsh 마이그레이션
VM에 호스트 장치 연결	RHEL 8.5	virt-xml --add-device
VM에서 호스트 장치 제거	RHEL 8.5	virt-xml --remove-device

추가 리소스

- [가상 머신 관리자 시작하기](#)

6장. 가상 머신에 대한 정보 보기

RHEL 8에서 가상화 배포의 모든 측면을 조정하거나 해결해야 할 경우 일반적으로 수행해야 하는 첫 번째 단계는 가상 시스템의 현재 상태 및 구성에 대한 정보를 확인하는 것입니다. 이를 위해 [명령줄 인터페이스](#) 또는 [웹 콘솔](#)을 사용할 수 있습니다. VM의 [XML 구성](#)에서 정보를 볼 수도 있습니다.

6.1. 명령줄 인터페이스를 사용하여 가상 머신 정보 보기

호스트 및 해당 구성의 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>
    [...]
  </metadata>
</domain>
```

- VM의 디스크 및 기타 블록 장치에 대한 자세한 내용은 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

VM 스토리지 관리에 대한 지침은 [가상 시스템의 스토리지](#) 관리를 참조하십시오.

- VM의 파일 시스템과 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 얻으려면 다음을 수행합니다.

```
# 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 [최적화](#)하려면 [가상 머신 CPU 성능 최적화](#)를 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 정보를 보려면 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 방법](#)에 대한 자세한 내용은 [가상 머신 네트워크 연결 구성](#)을 참조하십시오.

- 호스트의 스토리지 풀 및 스토리지 볼륨에 대한 정보를 확인하는 방법은 [CLI를 사용하여 가상 머신 스토리지 정보](#) 보기를 참조하십시오.

6.2. 웹 콘솔을 사용하여 가상 머신 정보 보기

RHEL 8 웹 콘솔을 사용하면 웹 콘솔 세션이 액세스할 수 있는 모든 [VM](#) 및 [스토리지 풀](#)에 대한 정보를 볼 수 있습니다.

웹 콘솔 세션이 [연결된 선택된 VM에 대한 정보](#)를 볼 수 있습니다. 여기에는 [디스크](#), [가상 네트워크 인터페이스](#) 및 [리소스 사용량](#)에 대한 정보가 포함됩니다.

6.2.1. 웹 콘솔에서 가상화 개요 보기

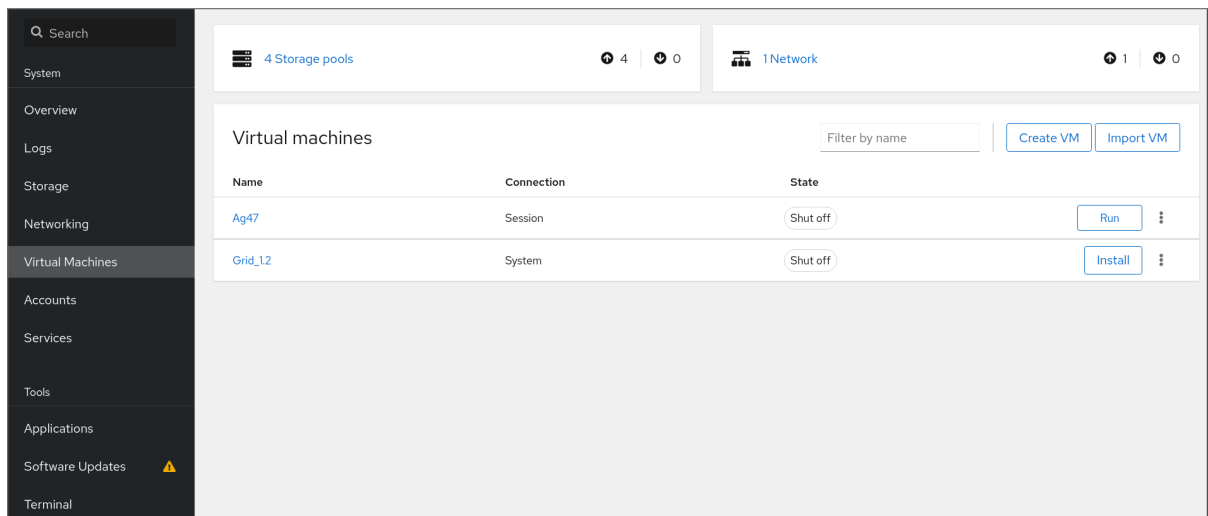
웹 콘솔을 사용하면 사용 가능한 VM(가상 시스템), 스토리지 풀 및 네트워크에 대한 요약 정보가 포함된 가상화 개요에 액세스할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.

절차

- 웹 콘솔의 사이드 메뉴에서 **Virtual Machines** (가상 시스템)를 클릭합니다.
사용 가능한 스토리지 풀, 사용 가능한 네트워크 및 웹 콘솔이 연결된 VM에 대한 정보가 포함된 대화 상자가 나타납니다.



이 정보에는 다음이 포함됩니다.

- **스토리지 풀** - 웹 콘솔 및 해당 상태에서 액세스할 수 있는 활성 또는 비활성 스토리지 풀 수입니다.
- **Networks(네트워크)** - 웹 콘솔 및 해당 상태에서 액세스할 수 있는 활성 또는 비활성 네트워크 수입니다.
- **Name (이름)** - VM의 이름입니다.
- **connection** - libvirt 연결, 시스템 또는 세션 유형입니다.
- **State** - VM 상태입니다.

추가 리소스

- [웹 콘솔을 사용하여 가상 머신 정보 보기](#)

6.2.2. 웹 콘솔을 사용하여 스토리지 풀 정보 보기

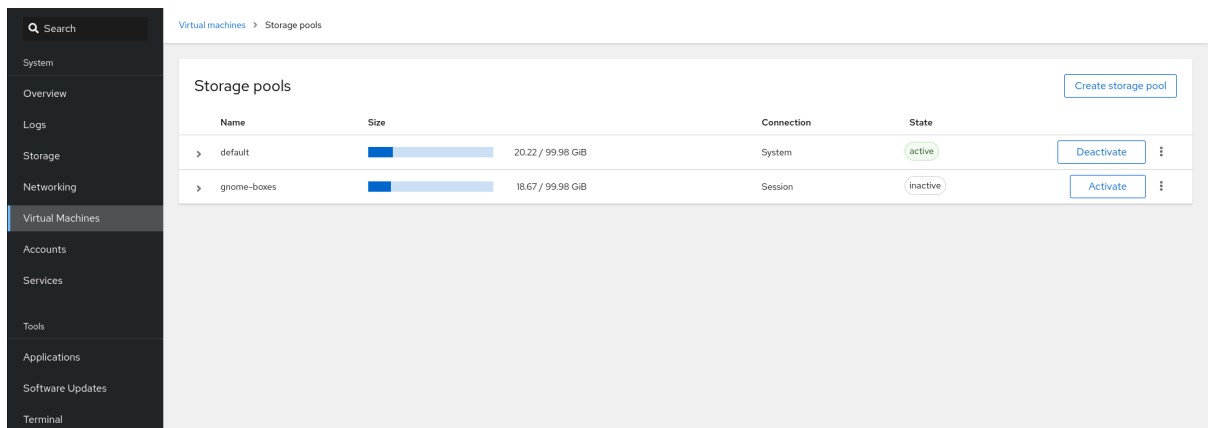
웹 콘솔을 사용하면 시스템에서 사용 가능한 스토리지 풀에 대한 자세한 정보를 볼 수 있습니다. 스토리지 풀을 사용하여 가상 시스템의 디스크 이미지를 생성할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.

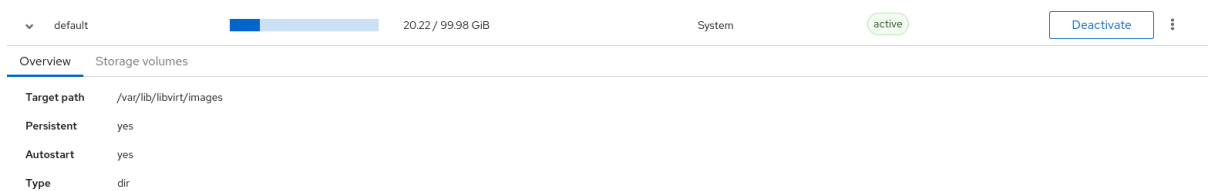
절차

1. 가상 머신 인터페이스 상단에 있는 **Storage Pools** (스토리지 풀)를 클릭합니다.
구성된 스토리지 풀 목록을 표시하는 Storage Pools(스토리지 풀) 창이 표시됩니다.



이 정보에는 다음이 포함됩니다.

- **name** - 스토리지 풀의 이름입니다.
 - **size** - 스토리지 풀의 현재 할당 및 총 용량입니다.
 - **연결** - 스토리지 풀 액세스에 사용되는 연결입니다.
 - **state** - 스토리지 풀의 상태입니다.
2. 정보를 보려는 스토리지 풀 옆에 있는 화살표를 클릭합니다.
행이 확장되어 선택한 스토리지 풀에 대한 자세한 정보가 포함된 Overview(개요) 창을 표시합니다.



정보에는 다음이 포함됩니다.

- **타겟 경로** - 디렉터리가 지원하는 스토리지 풀 유형의 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 **Storage(스토리지) Volumes (볼륨)** 를 클릭합니다.
- Storage Volumes(스토리지 볼륨) 창이 표시되고 구성된 스토리지 볼륨 목록이 표시됩니다.

▼ default	20.22 / 99.98 GiB	System	active	Deactivate	⋮
Overview	Storage volumes				
				Create volume	
Name	Used by	Size			
☐ volume1		0 / 1GB			
☐ volume2		0 / 1GB			

정보에는 다음이 포함됩니다.

- **name** - 스토리지 볼륨의 이름입니다.
- 현재 스토리지 볼륨을 사용하고 있는 **VM** 에서 사용합니다.
- **size** - 볼륨의 크기입니다.

추가 리소스

- [웹 콘솔을 사용하여 가상 머신 정보 보기](#)

6.2.3. 웹 콘솔에서 기본 가상 머신 정보 보기

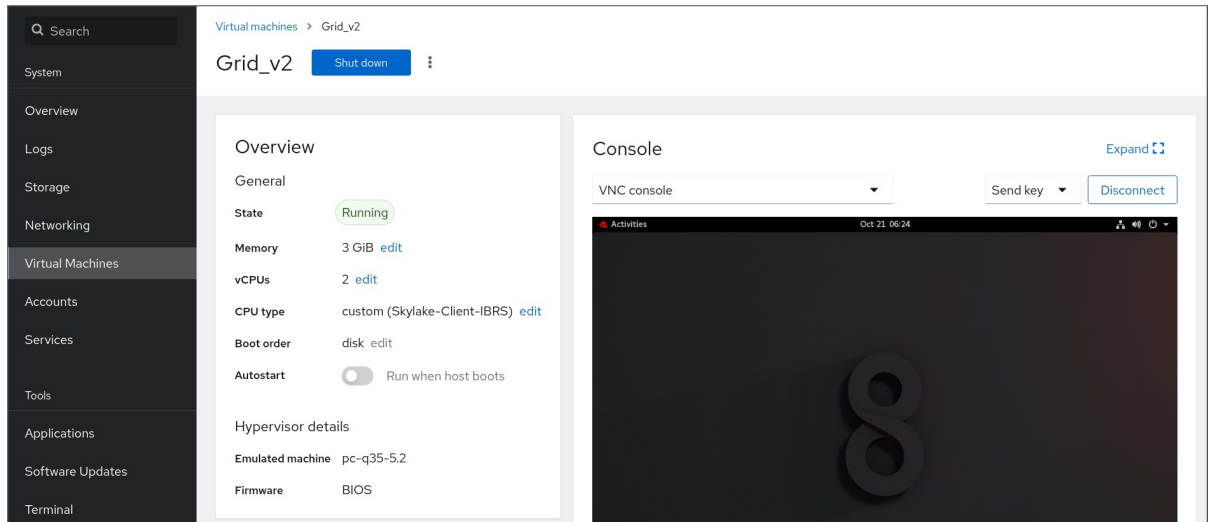
웹 콘솔을 사용하면 선택한 가상 시스템(VM)에 대해 할당된 리소스 또는 하이퍼바이저 세부 정보와 같은 기본 정보를 볼 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.

절차

1. 웹 콘솔 사이드 메뉴에서 **Virtual Machines** (가상 시스템)를 클릭합니다.
2. 보려는 정보를 보여주는 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 일반 VM 세부 정보가 포함되어 있습니다.

- **Status (상태)** - VM 상태, Running 또는 Shut off(실행 중 또는 종료)입니다.
- **memory** - VM에 할당된 메모리 양입니다.
- **vCPU** - VM에 대해 구성된 가상 CPU 수입니다.
- **CPU 유형** - VM에 대해 구성된 가상 CPU의 아키텍처입니다.
- **Boot Order(부팅 순서)** - VM에 대해 구성된 부팅 순서입니다.
- **auto start** - VM에 대해 자동시작 여부가 활성화됩니다.

이 정보에는 다음과 같은 하이퍼바이저 세부 정보도 포함됩니다.

- **에뮬레이트된 시스템** - VM에서 에뮬레이트된 시스템 유형입니다.
- **펌웨어** - VM 펌웨어입니다.

추가 리소스

- [웹 콘솔을 사용하여 가상 머신 정보 보기](#)
- [웹 콘솔을 사용하여 가상 CPU 관리](#)

6.2.4. 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

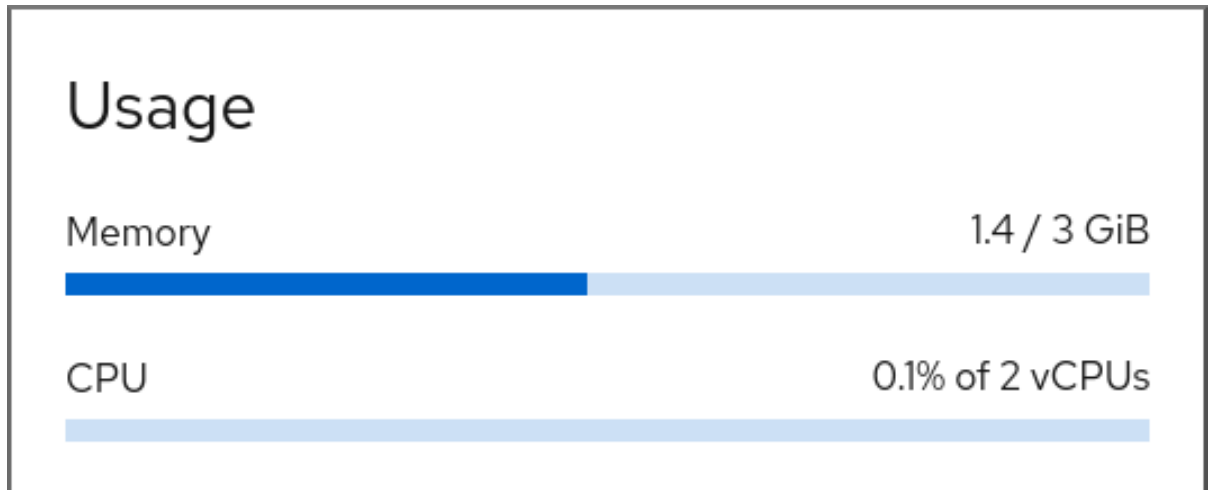
- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.

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. **Usage** (사용)로 스크롤합니다.

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



추가 리소스

- [웹 콘솔을 사용하여 가상 머신 정보 보기](#)

6.2.5. 웹 콘솔에서 가상 머신 디스크 정보 보기

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

사전 요구 사항

- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.

절차

1. 보려는 정보를 보여주는 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 스크롤.
Disks(디스크) 섹션에는 VM에 할당된 디스크 정보와 디스크 **추가**, **제거** 또는 **편집** 옵션이 표시됩니다.

Disks							Add disk
Device	Used	Capacity	Bus	Access	Source		
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove Edit
				Volume	v2		

이 정보에는 다음이 포함됩니다.

- **Device** - 디스크의 장치 유형입니다.
- **used** - 현재 할당된 디스크 양입니다.
- **capacity** - 스토리지 볼륨의 최대 크기입니다.

- **bus** - 에뮬레이트된 디스크 장치의 유형입니다.
- **액세스** - 디스크에 쓰기 가능 또는 읽기 전용 인지 여부. 원시 디스크의 경우 액세스를 쓰기 가능 및 공유 로 설정할 수도 있습니다.
- **Source** - 디스크 장치 또는 파일입니다.

추가 리소스

- [웹 콘솔을 사용하여 가상 머신 정보 보기](#)

6.2.6. 웹 콘솔에서 가상 네트워크 인터페이스 정보 보기 및 편집

RHEL 8 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 .

Network interfaces						Add network interface
Type	Model type	MAC address	IP address	Source	State	
network	virtio	52:54:00:b4:2a:62	inet 192.168.122.9/24	default	up	Delete Unplug Edit

+ 정보에는 다음이 포함됩니다.

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



참고

RHEL 8 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 **Edit(편집)**를 클릭합니다. Virtual Network Interface Settings(가상 네트워크 인터페이스 설정) 대화 상자가 열립니다.

52:54:00:b4:2a:62 virtual network interface settings

Interface type ②

Virtual network

Source

default

Model

(Linux, perf)

MAC address

52:64:00:b4:2a:63

Save

Cancel

4. 인터페이스 유형, 소스, 모델 또는 MAC 주소를 변경합니다.
5. **저장**을 클릭합니다. 네트워크 인터페이스가 수정되었습니다.



참고

가상 네트워크 인터페이스 설정에 대한 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](#)

6.3. 가상 머신 XML 구성 샘플

VM의 XML 구성(도메인 XML이라고도 함)은 VM의 설정 및 구성 요소를 결정합니다. 다음 표는 VM(가상 시스템)의 샘플 XML 구성 섹션과 내용을 설명합니다.

VM의 XML 구성을 얻기 위해 **virsh dumpxml** 명령 다음에 VM 이름을 사용할 수 있습니다.

```
# virsh dumpxml testguest1
```

표 6.1. 샘플 XML 구성

도메인 XML 섹션	설명
<pre><domain type='kvm'> <name>Testguest1</name> <uuid>ec6fbaa1-3eb4-49da-bf61-bb02fbec4967</uuid> <memory unit='KiB'>1048576</memory> <currentMemory unit='KiB'>1048576</currentMemory></pre>	<i>Testguest1</i>이라는 KVM 가상 머신으로, 1024MiB가 할당된 RAM이 할당됩니다.

도메인 XML 섹션	설명
<code><vcpu placement='static'>1</vcpu></code>	VM은 단일 가상 CPU(vCPU)로 할당됩니다. vCPU 구성에 대한 자세한 내용은 가상 머신 CPU 성능 최적화를 참조하십시오.
<pre> <os> <type arch='x86_64' machine='pc-q35-4.1'>hvm</type> <boot dev='hd'> </os> </pre>	머신 아키텍처는 AMD64 및 Intel 64 아키텍처로 설정되며 Intel Q35 시스템 유형을 사용하여 기능 호환성을 결정합니다. OS는 하드 드라이브에서 부팅되도록 설정됩니다. 설치된 OS를 사용하여 VM을 생성하는 방법에 대한 자세한 내용은 웹 콘솔을 사용하여 가상 머신 생성 및 게스트 운영 체제 설치를 참조하십시오.
<pre> <features> <acpi/> <apic/> </features> </pre>	acpi 및 apic 하이퍼바이저 기능은 비활성화되어 있습니다.
<code><cpu mode='host-model' check='partial'></code>	기능 XML의 호스트 CPU 정의(virsh 기능포함 가능)는 VM의 XML 구성으로 자동으로 복사됩니다. 따라서 VM을 부팅할 때 libvirt는 호스트 CPU와 유사한 CPU 모델을 선택한 다음 호스트 모델을 최대한 근접하게 만드는 추가 기능을 추가합니다.
<pre> <clock offset='utc'> <timer name='rtc' tickpolicy='catchup'> <timer name='pit' tickpolicy='delay'> <timer name='hpet' present='no'> </clock> </pre>	VM의 가상 하드웨어 클록은 UTC 시간대를 사용합니다. 또한 QEMU 하이퍼바이저와의 동기화를 위해 세 개의 다른 타이머가 설정됩니다.
<pre> <on_poweroff>destroy</on_poweroff> <on_reboot>restart</on_reboot> <on_crash>destroy</on_crash> </pre>	VM의 전원을 끄거나 OS가 예기치 않게 종료되면 libvirt는 VM을 종료하고 할당된 모든 리소스를 릴리스합니다. VM이 재부팅되면 libvirt가 동일한 구성으로 다시 시작합니다.

도메인 XML 섹션	설명
<pre> <pm> <suspend-to-mem enabled='no'/> <suspend-to-disk enabled='no'/> </pm> </pre>	이 VM에 대해 S3 및 S4 ACPI 절전 상태가 비활성화되어 있습니다.
<pre> <devices> <emulator>/usr/bin/qemu-kvm</emulator> <disk type='file' device='disk'> <driver name='qemu' type='qcow2'/> <source file='/var/lib/libvirt/images/Testguest.qcow2'/> <target dev='hda' bus='ide'/> </disk> <disk type='file' device='cdrom'> <driver name='qemu' type='raw'/> <target dev='hdb' bus='ide'/> <readonly/> </disk> </pre>	VM은 에뮬레이션에 /usr/bin/qemu-kvm 바이너리 파일을 사용하며 두 개의 디스크 장치가 연결되어 있습니다. 첫 번째 디스크는 호스트에 저장된 /var/libvirt/images/Testguest.qcow2를 기반으로 가상화된 하드 드라이브이며 해당 논리 장치 이름은 hda로 설정됩니다. 두 번째 디스크는 가상화된 CD-ROM이며 해당 논리적 장치 이름은 hdb로 설정됩니다.

도메인 XML 섹션	설명
<pre> <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'/> </pre>	VM은 USB 장치를 연결하는 데 단일 컨트롤러를 사용하고 PCI-Express(PCIe) 장치에 대한 루트 컨트롤러를 사용합니다. 또한 VM이 직렬 콘솔과 같은 다양한 방법으로 호스트와 상호 작용할 수 있는 virtio-serial 컨트롤러를 사용할 수 있습니다. 가상 장치에 대한 자세한 내용은 가상 장치 유형을 참조하십시오.
<pre> <interface type='network'> <mac address='52:54:00:65:29:21'/> <source network='default'/> <model type='rtl8139'/> </interface> </pre>	네트워크 인터페이스는 기본 가상 네트워크와 rtl8139 네트워크 장치 모델을 사용하는 VM에 설정됩니다. 네트워크 인터페이스 구성에 대한 자세한 내용은 가상 머신 네트워크 성능 최적화를 참조하십시오.

도메인 XML 섹션	설명
<pre> <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> <channel type='spicevmc'> <target type='virtio' name='com.redhat.spice.0'/> <address type='virtio-serial' controller='0' bus='0' port='2'/> </channel> </pre>	pty 직렬 콘솔이 VM에 설정되어 있으므로 호스트와의 기본 VM 통신이 가능합니다. 콘솔은 포트 1에서 UNIX 채널과 포트 2에서 반가상화 SPICE를 사용합니다. 이 설정은 자동으로 설정되며 이러한 설정을 변경하는 것은 권장되지 않습니다. VM과의 상호 작용에 대한 자세한 내용은 웹 콘솔을 사용하여 가상 머신과 상호 작용을 참조하십시오.
<pre> <input type='tablet' bus='usb'> <address type='usb' bus='0' port='1'/> </input> <input type='mouse' bus='ps2'> <input type='keyboard' bus='ps2'> </pre>	VM은 태블릿 입력을 수신하도록 설정된 가상 usb 포트와 마우스 및 키보드 입력을 수신하도록 설정된 가상 ps2 포트를 사용합니다. 이 설정은 자동으로 설정되며 이러한 설정을 변경하는 것은 권장되지 않습니다.
<pre> <graphics type='spice' autoport='yes' listen='127.0.0.1'> <listen type='address' address='127.0.0.1'/> <image compression='off'/> </graphics> <graphics type='vnc' port='-1' autoport='yes' listen='127.0.0.1'> <listen type='address' address='127.0.0.1'/> </graphics> </pre>	VM은 그래픽 출력을 렌더링하기 위해 VNC 및 SPICE 프로토콜을 사용하며 이미지 압축이 해제됩니다.
<pre> <sound model='ich6'> <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/> </sound> <video> <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1' primary='yes'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/> </video> </pre>	ICH6 HDA 사운드 장치가 VM에 대해 설정되고 QEMU QXL 반가상화 프레임버퍼 장치가 비디오 액셀러레이터로 설정됩니다. 이 설정은 자동으로 설정되며 이러한 설정을 변경하는 것은 권장되지 않습니다.

도메인 XML 섹션	설명
<pre> <redirdev bus='usb' type='spicevmc'> <address type='usb' bus='0' port='1' /> </redirdev> <redirdev bus='usb' type='spicevmc'> <address type='usb' bus='0' port='2' /> </redirdev> <memballoon model='virtio'> <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0' /> </memballoon> </devices> </domain> </pre>	VM에는 USB 장치를 원격으로 연결하는 두 개의 리디렉션이 있으며 메모리 증대가 커집니다. 이 설정은 자동으로 설정되며 이러한 설정을 변경하는 것은 권장되지 않습니다.

7장. 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 워크로드가 더 빨리 실행되도록 하려면 저장 및 복원 기능을 사용하여 OS 종료 및 부팅 시퀀스를 완전히 방지할 수 있습니다.

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

7.1. 가상 머신의 저장 및 복원 방법

VM(가상 시스템)을 저장하면 호스트의 디스크에 메모리 및 장치 상태가 저장되고 VM 프로세스가 즉시 중지됩니다. 실행 중 또는 일시 중지된 상태인 VM을 저장할 수 있으며 복원 시 VM이 해당 상태로 돌아갑니다.

이 프로세스는 디스크 공간을 대신하여 호스트 시스템의 RAM 및 CPU 리소스를 사용 가능하게 하므로 호스트 시스템 성능이 향상될 수 있습니다. 게스트 OS를 부팅할 필요가 없으므로 VM을 복원하면 부팅 기간이 길어집니다.

VM을 저장하려면 CLI(명령줄 인터페이스)를 사용할 수 있습니다. 자세한 내용은 [명령행 인터페이스를 사용하여 가상 시스템 절약을 참조](#)하십시오.

VM을 복원하려면 [CLI](#) 또는 [웹 콘솔 GUI](#)를 사용할 수 있습니다.

7.2. 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 수 있는 디스크 공간이 충분한지 확인합니다. VM에서 점유하는 공간은 해당 VM에 할당된 RAM의 양에 따라 다릅니다.
- VM이 영구적인지 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 **demo-guest1.save**로 **/var/lib/libvirt/qemu/save** 디렉터리에 있습니다.

다음에 VM이 **시작되면** 위의 파일에서 저장된 상태를 자동으로 복원합니다.

검증

- VM이 저장되었는지 확인하거나 **virsh list** 유틸리티를 사용하여 종료할 수 있습니다. 저장을 활성화한 관리 VM을 나열하려면 다음 명령을 사용합니다. 저장된 대로 나열된 VM은 관리 저장이 활성화됩니다.

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

관리되는 저장 이미지가 있는 VM을 나열하려면 다음을 수행합니다.

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

저장된 VM이 종료됨을 표시하려면 명령에 **--all** 또는 **--inactive** 옵션을 사용해야 합니다.

문제 해결

- 저장된 VM 파일이 손상되거나 읽을 수 없게 되면 VM을 복원하면 표준 VM 부팅이 시작됩니다.

추가 리소스

- **virsh managedsave --help** 명령
- [명령줄 인터페이스를 사용하여 저장된 VM 복원](#)
- [웹 콘솔을 사용하여 저장된 VM 복원](#)

7.3. 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 권한.

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 호스트에 대한 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

7.4. 웹 콘솔을 사용하여 가상 머신 시작

VM(가상 머신)이 종료된 상태인 경우 RHEL 8 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. **Run(실행)**을 클릭합니다.
VM이 시작되고 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 시작](#)을 참조하십시오.

추가 리소스

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

8장. 가상 머신 복제

특정 속성 집합을 사용하여 새 VM(가상 머신)을 신속하게 생성하려면 기존 VM을 복제할 수 있습니다.

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

8.1. 가상 머신 복제의 작동 방식

VM(가상 시스템)을 복제하면 소스 VM과 해당 디스크 이미지의 XML 구성을 복사하고 구성을 조정하여 새 VM의 고유성을 보장합니다. 여기에는 VM의 이름을 변경하고 디스크 이미지 복제본을 사용하는지 확인하는 작업이 포함됩니다. 하지만 복제본의 가상 디스크에 저장된 데이터는 소스 VM과 동일합니다.

이 프로세스는 새 VM을 만들고 게스트 운영 체제로 설치하는 것보다 빠르며 특정 구성 및 콘텐츠로 VM을 신속하게 생성하는 데 사용할 수 있습니다.

VM 복제본을 여러 개 생성하려는 경우 먼저 다음을 포함하지 않는 VM 템플릿을 생성합니다.

- 복제본이 올바르게 작동하지 않도록 할 수 있는 영구 네트워크 MAC 구성과 같은 고유한 설정입니다.
- SSH 키 및 암호 파일과 같은 중요한 데이터.

자세한 내용은 [가상 머신 템플릿 생성](#)을 참조하십시오.

추가 리소스

- [명령줄 인터페이스를 사용하여 가상 머신 복제](#)
- [웹 콘솔을 사용하여 가상 머신 복제](#)

8.2. 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 복제본을 쉽고 안전하게 생성하는 데 사용할 수 있는 VM 템플릿이 생성됩니다.

[virt-sysprep](#) 유틸리티를 사용하여 VM 템플릿을 생성하거나 요구 사항에 따라 수동으로 생성할 수 있습니다.

8.2.1. 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.

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

- 템플릿으로 의도된 VM이 종료됩니다.

- 소스 VM의 디스크 이미지가 있는 위치를 알고 있어야 하며 VM의 디스크 이미지 파일의 소유자입니다.
libvirt의 [시스템 연결](#)에 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 모두 백업되었습니다. 소스 VM을 그대로 유지하려면 먼저 [복제하고 복제본을](#) 편집하여 템플릿을 생성합니다.

절차

1. VM의 디스크 이미지 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 전환되었는지 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 이미지와 비교합니다. 다음 예제에서는 템플릿이 성공적으로 생성되었음을 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](#)

8.2.2. 수동으로 가상 머신 템플릿 생성

기존 VM(가상 시스템)에서 템플릿을 생성하려면 게스트 VM을 수동으로 재설정하거나 구성 해제하여 복제를 준비할 수 있습니다.

사전 요구 사항

- 소스 VM의 디스크 이미지 위치를 알고 있고 VM의 디스크 이미지 파일의 소유자임을 확인합니다. libvirt의 [시스템 연결](#)에 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 모두 백업되었습니다. 소스 VM을 그대로 유지하려면 먼저 [복제하고 복제본을](#) 편집하여 템플릿을 생성합니다.

절차

1. 복제할 VM을 구성합니다.
 - a. 복제본에 필요한 소프트웨어를 설치합니다.
 - b. 운영 체제에 대해 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 않은 경우 첫 번째 NIC 이름은 **eth 0** 대신 **eth 1** 이 될 수 있습니다.

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



참고

HWADDR이 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. DHCP를 구성하지만 HWADDR 또는 기타 고유한 정보는 포함하지 마십시오.

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

- c. 시스템에 있는 경우 다음 파일에 동일한 내용이 포함되어 있는지 확인합니다.

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



참고

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

3. 등록 세부 정보 삭제:

- 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 않으려는 경우:

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

- 원래 VM을 사용하려는 경우:

```
# subscription-manager clean
```



참고

원래 RHSM 프로파일은 ID 코드와 함께 포털에 유지됩니다. 다음 명령을 사용하여 VM에서 RHSM 등록을 복제한 후 다시 활성화합니다.

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

4. 다른 고유한 세부 정보 제거:

- a. ssh 공개/개인 키 쌍을 삭제합니다.

4. `ssh` 구성/패킷을 제거합니다.

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

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

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

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



참고

다음 부팅에서 실행되는 마법사는 VM에서 제거된 구성에 따라 다릅니다. 또한 복제가 처음 부팅되면 호스트 이름을 변경하는 것이 좋습니다.

8.3. 명령줄 인터페이스를 사용하여 가상 머신 복제

특정 속성 집합(예: 테스트용)을 사용하여 새 VM(가상 머신)을 신속하게 생성하려면 기존 VM을 복제할 수 있습니다. CLI를 사용하여 이를 수행하려면 다음 지침을 따르십시오.

사전 요구 사항

- source VM이 종료되었습니다.
- 복제된 디스크 이미지를 저장할 수 있는 디스크 공간이 충분한지 확인합니다.
- **선택 사항:** 여러 VM 복제본을 생성할 때 소스 VM에서 고유한 데이터 및 설정을 제거하여 복제된 VM이 제대로 작동하는지 확인합니다. 자세한 내용은 [가상 머신 템플릿 생성](#)을 참조하십시오.

절차

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

샘플 사용 사례

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

```
# 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, *geminus1*의 여러 디스크 중 2개만 사용하는 *geminus2* 라는 로컬 VM을 생성합니다.

```
# 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.
```

- 다른 호스트로 VM을 복제하려면 로컬 호스트에서 VM 정의를 해제하지 않고 VM을 마이그레이션합니다. 예를 들어 다음 명령은 이전에 만든 *geminus2* VM을 로컬 디스크를 포함하여 10.0.0.1 원격 시스템에 복제합니다. 이러한 명령을 사용하려면 10.0.0.1에 대한 루트 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. 복제본이 호스트의 VM 목록에 추가되었는지 확인합니다.

```
# virsh list --all
Id Name State
-----
- doppelganger shut off
- doppelganger-clone shut off
```

2. 복제본을 시작하고 부팅되는지 관찰합니다.

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

추가 리소스

- **virt-clone** 도움말 페이지
- [가상 머신 마이그레이션](#)

8.4. 웹 콘솔을 사용하여 가상 머신 복제

특정 속성 세트의 새 VM(가상 시스템)을 빠르게 생성하려면 이전에 구성한 VM을 복제할 수 있습니다. 다음 지침에서는 웹 콘솔을 사용하여 이를 수행하는 방법을 설명합니다.



참고

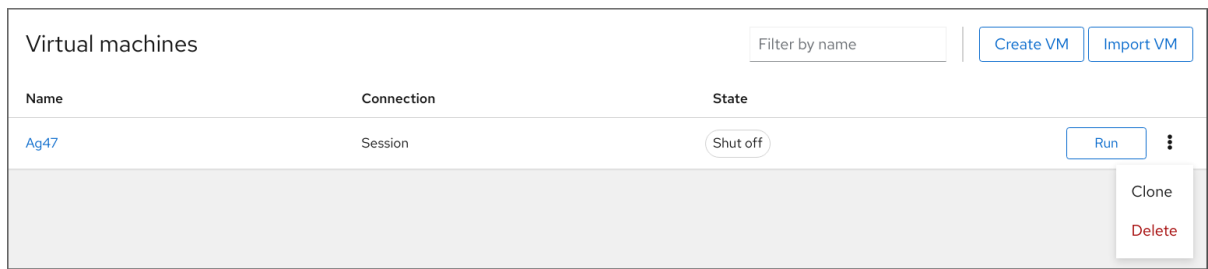
VM을 복제하면 해당 VM과 연결된 디스크도 복제됩니다.

사전 요구 사항

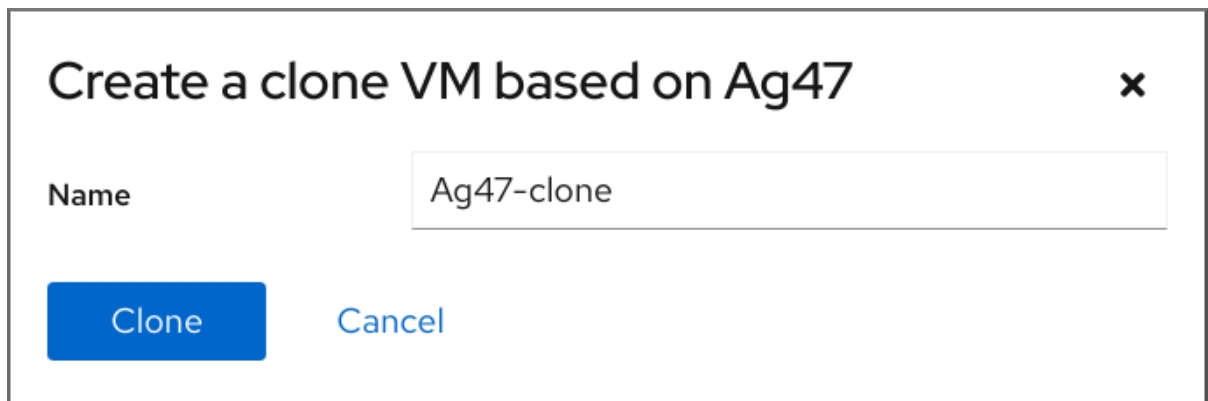
- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.
- 복제하려는 VM이 종료되었는지 확인합니다.

절차

1. 웹 콘솔의 Virtual Machines(가상 시스템) 인터페이스에서 복제할 **VM**의 Menu(메뉴) 단추를 클릭합니다.
다양한 VM 작업에 대한 컨트롤이 포함된 드롭다운 메뉴가 나타납니다.



2. **Clone(복제)**을 클릭합니다.
Create a clone VM(복제 VM 만들기) 대화 상자가 표시됩니다.



3. **선택 사항:** VM 복제의 새 이름을 입력합니다.
4. **Clone(복제)**을 클릭합니다.
소스 VM을 기반으로 새 VM이 생성됩니다.

검증

- 복제된 VM이 호스트에서 사용 가능한 VM 목록에 표시되는지 확인합니다.

9장. 가상 머신 마이그레이션

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

9.1. 가상 머신 마이그레이션 방법

VM(가상 시스템) 마이그레이션의 필수 부분은 VM의 XML 구성을 다른 호스트 시스템에 복사하는 것입니다. 마이그레이션된 VM이 종료되지 않은 경우 마이그레이션은 VM의 메모리 및 가상화된 장치의 상태도 대상 호스트 시스템으로 전송합니다. VM이 대상 호스트에서 작동 상태로 유지하려면 VM의 디스크 이미지를 계속 사용할 수 있어야 합니다.

기본적으로 마이그레이션된 VM은 대상 호스트에서 일시적이며 소스 호스트에도 정의되어 있습니다.

실시간 또는 실시간이 아닌 마이그레이션을 사용하여 실행 중인 VM을 마이그레이션할 수 있습니다. 가상 머신 종료를 마이그레이션하려면 오프라인 마이그레이션을 사용해야 합니다. 자세한 내용은 다음 표를 참조하십시오.

표 9.1. VM 마이그레이션 유형

마이그레이션 유형	설명	사용 사례	스토리지 요구사항
실시간 마이그레이션	KVM이 VM의 메모리 페이지를 대상 호스트로 전송하는 동안 VM은 소스 호스트 시스템에서 계속 실행됩니다. 마이그레이션이 거의 완료되면 KVM은 VM을 매우 간단하게 일시 중지하고 대상 호스트에서 다시 시작합니다.	지속적인 가동 시간이 필요한 VM에 유용합니다. 그러나 메모리 페이지를 수정하는 VM은 많은 I/O 부하가 있는 VM과 같이 메모리 페이지를 이전할 수 있으며 <i>실시간이 아닌 마이그레이션</i> 을 대신 사용해야 합니다.	VM의 디스크 이미지는 공유 네트워크 에 있어야 하며 소스 호스트와 대상 호스트에 모두 액세스할 수 있어야 합니다.
실시간이 아닌 마이그레이션	VM을 일시 중지하고 해당 구성과 해당 메모리를 대상 호스트에 복사한 다음 VM을 다시 시작합니다.	VM에 대한 다운타임을 생성하지만 실시간 마이그레이션보다 일반적으로 더 안정적입니다. 과도한 I/O 부하가 높은 VM에 권장.	VM의 디스크 이미지는 공유 네트워크 에 있어야 하며 소스 호스트와 대상 호스트에 모두 액세스할 수 있어야 합니다.
오프라인 마이그레이션	VM의 구성을 대상 호스트로 이동	VM 종료에 권장.	VM의 디스크 이미지는 공유 네트워크에서 사용할 수 없으며 대신 대상 호스트로 수동으로 복사하거나 이동할 수 있습니다.

추가 리소스

- [가상 머신 마이그레이션의 이점](#)
- [다른 호스트와 가상 머신 디스크 이미지 공유](#)

9.2. 가상 머신 마이그레이션의 이점

VM(가상 머신)을 마이그레이션하는 데 유용할 수 있습니다.

로드 밸런싱

VM은 호스트가 과부하가 발생하거나 다른 호스트가 활용도가 낮은 경우 사용량이 낮은 시스템을 호스트하도록 이동할 수 있습니다.

하드웨어 독립

호스트 시스템에서 하드웨어 장치를 업그레이드, 추가 또는 제거해야 하는 경우 VM을 다른 호스트로 안전하게 재배포할 수 있습니다. 즉, VM에서 하드웨어 개선에 다운타임이 발생하지 않습니다.

에너지 절약

VM을 다른 호스트에 재배포할 수 있으며, 로드되지 않은 호스트 시스템의 전원이 꺼져 사용 기간이 짧은 동안 에너지 절약 및 비용을 절감할 수 있습니다.

지리적 마이그레이션

VM은 대기 시간을 단축하거나 다른 이유로 필요한 경우 다른 물리적 위치로 이동할 수 있습니다.

9.3. 가상 머신 마이그레이션의 제한 사항

RHEL 8에서 가상 머신(VM)을 마이그레이션하기 전에 마이그레이션의 제한 사항을 알고 있어야 합니다.

- 실시간 스토리지 마이그레이션은 RHEL 8에서 수행할 수 없지만 VM의 전원이 꺼진 동안 스토리지를 마이그레이션할 수 있습니다. 실시간 스토리지 마이그레이션은 [Red Hat Virtualization](#)에서 사용할 수 있습니다.
- [libvirt](#)의 [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 통과
 - SR-IOV 장치 할당
 - vGPU와 같은 중재된 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 VM과 대상 VM의 에뮬레이션된 CPU는 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.

9.4. 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 소스 및 대상 호스트 모두 네트워크를 통해 연결할 수 있습니다. 이는 공유 스토리지에 가장 적합한 솔루션이며 Red Hat에서 권장합니다.
- NFS 파일 잠금이 KVM에서 지원되지 않으므로 사용되지 않는지 확인합니다.
- NFS는 소스 및 대상 호스트에 설치 및 활성화됩니다. 그렇지 않은 경우 다음을 수행합니다.

a. NFS 패키지를 설치합니다.

```
# yum install nfs-utils
```

b. NFS의 포트(예: 2049)가 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 제공하는 호스트에 연결합니다. 이 예제에서 호스트는 **satellite-hosts**입니다.

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

2. 디스크 이미지를 저장할 디렉터리를 만들고 마이그레이션 호스트와 공유할 디렉터리를 생성합니다.

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

3. VM의 디스크 이미지를 소스 호스트에서 새로 생성된 디렉터리로 복사합니다. 예를 들어 다음은 **wanderer1** VM의 디스크 이미지를 'cargo-' 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 **/etc/exports** 파일에 공유 디렉토리를 추가합니다. 다음 예제에서는 **/var/lib/libvirt/shared-images** 디렉터를 **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 호스트와 대상 호스트 모두에서 **/var/lib/libvirt/images** 디렉터리에 공유 디렉터를 마운트합니다.

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

검증

- 프로세스가 성공했는지 확인하려면 소스 호스트에서 VM을 시작하고 올바르게 부팅되는지 관찰합니다.

추가 리소스

- [NFS 공유 내보내기](#)

9.5. 명령줄 인터페이스를 사용하여 가상 머신 마이그레이션

VM(가상 머신)의 현재 호스트가 적합하지 않거나 더 이상 사용할 수 없는 경우 또는 호스팅 워크로드를 재배포하려는 경우 VM을 다른 KVM 호스트로 마이그레이션할 수 있습니다. 이 섹션에서는 이러한 마이그레이션의 다양한 시나리오에 대한 지침과 예를 설명합니다.

사전 요구 사항

- 소스 호스트와 대상 호스트 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 QEMU에 49152-49215 포트가 필요합니다.
- Red Hat에서 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 VM의 디스크 이미지는 소스 호스트와 대상 호스트에 모두 액세스할 수 있는 별도의 네트워크 위치에 있습니다. 이는 오프라인 마이그레이션에는 선택 사항이지만 실행 중인 VM을 마이그레이션하는 데 필요합니다.
이러한 공유 VM 스토리지를 설정하는 방법은 [다른 호스트와 가상 머신 디스크 이미지 공유를](#) 참조하십시오.
- 실행 중인 VM을 마이그레이션할 때 VM에서 더티티 메모리 페이지를 생성하는 속도보다 네트워크 대역폭이 커야 합니다.
실시간 마이그레이션을 시작하기 전에 VM의 더티 않은 페이지 비율을 얻으려면 다음을 수행하십시오.

- a. 짧은 기간 동안 VM의 더러운 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 더티티 메모리 페이지를 생성합니다. 대역폭이 2MB/s 이하인 네트워크에서 VM을 실시간 마이그레이션하려고 하면 VM을 일시 중지하지 않거나 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.

- 공용 브리지 탭 네트워크에서 기존 VM을 마이그레이션할 때 소스 및 대상 호스트가 동일한 네트워크에 있어야 합니다. 그렇지 않으면 마이그레이션 후 VM 네트워크가 작동하지 않습니다.
- **libvirt** 서비스가 활성화되어 실행 중인지 확인합니다.

```
# systemctl enable --now libvirtd.service
```

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. 마이그레이션하는 동안 VM은 계속 실행됩니다.

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

- 다음을 사용하면 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
```

이 절차는 예를 들어 대상 호스트에서 공유 VM 스토리지에 액세스하기 위해 다른 경로를 사용해야 하거나 대상 호스트와 관련된 기능을 구성할 때 유용할 수 있습니다.

- 다음은 **source-example** 호스트에서 **wanderer3** VM을 일시 중단하고, **dest-example** 호스트로 마이그레이션한 다음, **wanderer3-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 **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 부하 및 VM 크기에 따라 다소 시간이 걸릴 수 있습니다. **virsh** 마이그레이션에 **--verbose** 옵션을 사용하지 않으면 CLI에 오류를 제외한 진행 상태 표시기가 표시되지 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 아직 실행 중이면 이 명령에서 VM 상태를 **paused** 로 나열합니다.

문제 해결

- 대상 호스트는 네트워크 이름 또는 CPU 유형과 같은 마이그레이션된 VM의 XML 구성 값과 호환되지 않는 경우도 있습니다. 결과적으로 VM은 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 마이그레이션을 완료하는 데 시간이 오래 걸리는 경우 VM이 과부하 상태이고 메모리 페이지가 너무 많이 변경되어 실시간 마이그레이션이 가능하기 때문일 수 있습니다. 이 문제를 해결하려면 VM을 일시 중단하여 비실시간으로 마이그레이션을 변경합니다.

```
# virsh suspend wanderer1
```

추가 리소스

- **virsh migrate --help** 명령
- **virsh** man 페이지

9.6. 웹 콘솔을 사용하여 가상 머신 실시간 마이그레이션

지속적으로 실행해야 하는 작업을 수행하는 VM(가상 시스템)을 마이그레이션하려면 종료하지 않고 해당 VM을 다른 KVM 호스트로 마이그레이션할 수 있습니다. 이것을 실시간 마이그레이션이라고도 합니다. 다음 지침에서는 웹 콘솔을 사용하여 이를 수행하는 방법을 설명합니다.



주의

KVM이 많은 I/O 로드 작업과 같이 메모리 페이지를 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 호스트와 대상 호스트에서 액세스할 수 있는 [공유 스토리지](#)에 있습니다.
- 실행 중인 VM을 마이그레이션할 때 VM에서 더티티 메모리 페이지를 생성하는 속도보다 네트워크 대역폭이 커야 합니다.
실시간 마이그레이션을 시작하기 전에 VM의 더티티 페이지 비율을 얻으려면 명령줄 인터페이스에서 다음을 수행합니다.
 - a. 짧은 기간 동안 VM의 더러운 페이지 생성 속도를 모니터링합니다.

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

- b. 모니터링이 완료되면 결과를 가져옵니다.

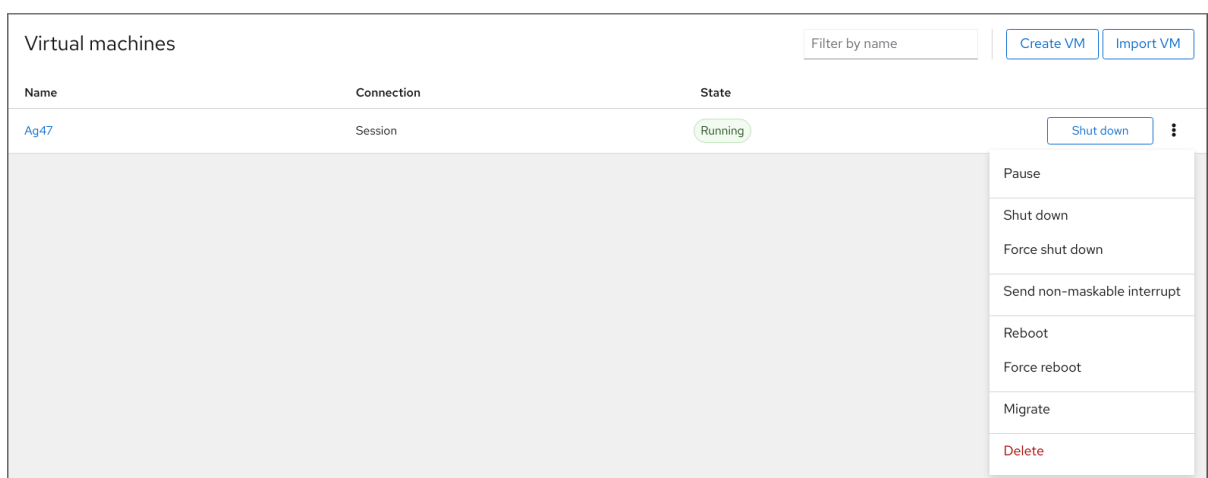
```
# virsh domstats vm-name --dirtyrate
Domain: 'vm-name'
dirtyrate.calc_status=2
dirtyrate.calc_start_time=200942
dirtyrate.calc_period=30
dirtyrate.megabytes_per_second=2
```

이 예에서 VM은 초당 2MB의 더티티 메모리 페이지를 생성합니다. 대역폭이 2MB/s 이하인 네트워크에서 VM을 실시간 마이그레이션하려고 하면 VM을 일시 중지하지 않거나 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(가상 시스템) 인터페이스에서 마이그레이션하려는 **VM**의 Menu(메뉴) 단추를 클릭합니다.
다양한 VM 작업에 대한 컨트롤이 포함된 드롭다운 메뉴가 나타납니다.



2. 마이그레이션을 클릭합니다.
Migrate VM to another host(VM을 다른 호스트로 마이그레이션) 대화 상자가 표시됩니다.

Migrate VM to another host
×

Storage volumes must be shared between this host and the destination host.

Destination URI

Duration ?
☐ Temporary migration

Migrate
Cancel

3. 대상 호스트의 URI를 입력합니다.

4. 마이그레이션 시간을 구성합니다.

- **Permanent** (영구) - VM을 영구적으로 마이그레이션하려는 경우 확인란을 선택하지 마십시오. 영구 마이그레이션은 소스 호스트에서 VM 구성을 완전히 제거합니다.
- **임시** - 임시 마이그레이션은 VM의 사본을 대상 호스트로 마이그레이션합니다. 이 복사본은 VM이 종료되면 대상 호스트에서 삭제됩니다. 원래 VM은 소스 호스트에 남아 있습니다.

5. 마이그레이션을 클릭합니다.

VM이 대상 호스트로 마이그레이션됩니다.

검증

VM이 성공적으로 마이그레이션되었으며 올바르게 작동하는지 확인하려면 다음을 수행하십시오.

- 대상 호스트에서 사용 가능한 VM 목록에 VM이 표시되는지 확인합니다.
- 마이그레이션된 VM을 시작하고 부팅되는지 관찰합니다.

9.7. 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 표는 지원되는 VM 마이그레이션 경로를 보여줍니다.

표 9.2. 실시간 마이그레이션 호환성

마이그레이션 방법	릴리스 유형	예제	지원 상태
앞으로	주요 릴리스	7.6+ → 8.1	지원되는 RHEL 7 시스템에서: 머신 유형 i440fx 및 q35
이전	주요 릴리스	8.1 → 7.6+	지원되는 RHEL 8 시스템에서 시스템 유형 i440fx 및 q35

마이그레이션 방법	릴리스 유형	예제	지원 상태
앞으로	마이너 릴리스	8.0.1+ → 8.1+	지원되는 RHEL 7 시스템에서: RHEL 7.6.0 이상에서 시스템 유형 i440fx 및 q35. 지원되는 RHEL 8 시스템에서 시스템 유형 q35.
이전	마이너 릴리스	8.1 → 8.0.1	지원하는 RHEL 7 시스템에서. 시스템 유형 i440fx 및 q35에 대해 완벽하게 지원됩니다. 지원되는 RHEL 8 시스템에서 시스템 유형 q35.

추가 리소스

- [Red Hat Enterprise Linux 라이프 사이클](#)

9.8. 추가 리소스

- RHEL 7 및 RHEL 8에서 **virt-v2v**를 사용하여 다른 하이퍼바이저에서 KVM으로 가상 머신을 변환

10장. 가상 장치 관리

VM(가상 시스템)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.

10.1. 가상 장치의 작동 방식

실제 시스템과 마찬가지로 VM(가상 시스템)에서는 성능, 메모리, 스토리지, 네트워킹 또는 그래픽과 같은 기능을 시스템에 제공하는 특수 장치가 필요합니다. 물리적 시스템은 일반적으로 이러한 목적으로 하드웨어 장치를 사용합니다. 그러나 VM은 소프트웨어 구현으로 작동하기 때문에 *가상 장치라고 하는 이러한 장치에 대한 소프트웨어 추상화를 사용해야 합니다.*

기본 사항

VM에 연결된 가상 장치는 VM [생성 시 구성할 수 있으며 기존 VM](#) 에서도 관리할 수 있습니다. 일반적으로 가상 장치는 VM이 종료된 경우에만 VM에서 연결하거나 분리할 수 있지만, VM이 실행 중인 경우 일부 장치를 추가하거나 제거할 수 있습니다. 이 기능을 장치 *핫 플러그 및 핫 언플러그* 라고 합니다.

새 VM을 만들 때 **libvirt** 는 사용자가 별도로 지정하지 않는 한, 필수 가상 장치 집합을 자동으로 생성하고 구성합니다. 이러한 항목은 호스트 시스템 아키텍처 및 시스템 유형을 기반으로 하며, 일반적으로 다음을 포함합니다.

- the CPU
- memory
- 키보드
- NIC(네트워크 인터페이스 컨트롤러)
- 다양한 장치 컨트롤러
- 비디오 카드
- 사운드 카드

VM이 생성된 후 가상 장치를 관리하려면 CLI(명령줄 인터페이스)를 사용합니다. 그러나 [가상 스토리지 장치](#) 및 [NIC](#) 를 관리하기 위해 RHEL 8 웹 콘솔을 사용할 수도 있습니다.

성능 또는 유연성

일부 유형의 장치의 경우 RHEL 8은 종종 성능과 유연성 간의 절충을 통해 여러 구현을 지원합니다.

예를 들어 가상 디스크에 사용되는 물리적 스토리지는 **qcow2** 또는 **raw** 와 같은 다양한 형식의 파일로 표시할 수 있으며 다양한 컨트롤러를 사용하여 VM에 제공할 수 있습니다.

- 에뮬레이트된 컨트롤러
- **virtio-scsi**
- **virtio-blk**

virtio 장치는 가상화를 위해 특별히 설계되었기 때문에 에뮬레이트된 컨트롤러는 **virtio** 컨트롤러보다 느립니다. 반면 에뮬레이트된 컨트롤러를 사용하면 **virtio** 장치의 드라이버가 없는 운영 체제를 실행할 수 있

습니다. 마찬가지로 **virtio-scsi**는 **SCSI** 명령에 대한 보다 완벽한 지원을 제공하며, **VM**에 다수의 디스크를 연결할 수 있습니다. 마지막으로 **virtio-blk**는 **virtio-scsi** 및 에뮬레이트된 컨트롤러보다 더 나은 성능을 제공하지만 사용 사례 범위가 더 제한적입니다. 예를 들어 **virtio-blk**를 사용하는 경우에는 물리적 디스크를 **VM**에 **LUN** 장치로 연결할 수 없습니다.

가상 장치 유형에 대한 자세한 내용은 가상 장치 [유형](#)을 참조하십시오.

추가 리소스

- [가상 머신용 스토리지 관리](#)
- [가상 머신 네트워크 인터페이스를 관리하기 위해 웹 콘솔 사용](#)
- [NVIDIA vGPU 장치 관리](#)

10.2. 가상 장치 유형

RHEL 8의 가상화는 가상 머신(**VM**)에 연결할 수 있는 여러 유형의 가상 장치를 제공할 수 있습니다.

에뮬레이트된 장치

에뮬레이트된 장치는 널리 사용되는 물리적 장치의 소프트웨어 구현입니다. 물리적 장치를 위해 설계된 드라이버도 에뮬레이트된 장치와 호환됩니다. 따라서 에뮬레이트된 장치를 매우 유연하게 사용할 수 있습니다.

그러나 특정 유형의 하드웨어를 강력하게 에뮬레이션해야 하므로 에뮬레이션된 장치는 해당 물리적 장치 또는 보다 최적화된 가상 장치에 비해 성능이 저하될 수 있습니다.

에뮬레이트된 장치는 다음과 같습니다.

- 다양한 **CPU** 모델을 선택할 수 있는 가상 **CPU(vCPU)**. 에뮬레이션의 성능 영향은 호스트 **CPU**와 에뮬레이트된 **vCPU**의 차이에 따라 크게 달라집니다.
- **PCI** 버스 컨트롤러와 같은 에뮬레이션된 시스템 구성 요소입니다.

- **SATA, SCSI 또는 IDE와 같은 에뮬레이션된 스토리지 컨트롤러.**
- **ICH9, ICH6 또는 AC97과 같은 에뮬레이션된 음향 장치.**
- **VGA 또는 QXL 카드와 같은 에뮬레이션된 그래픽 카드.**
- **rtl8139와 같은 에뮬레이션된 네트워크 장치.**

반가상화 장치

반가상화는 가상 장치를 VM에 노출하는 빠르고 효율적인 방법을 제공합니다. 반가상화 장치는 VM에서 사용하도록 특별히 설계된 인터페이스를 노출하므로 장치 성능이 크게 향상됩니다. **RHEL 8**은 하이퍼바이저와 VM 간의 계층으로 **virtio API**를 사용하여 VM에 반가상화 장치를 제공합니다. 이 접근 방식의 단점은 게스트 운영 체제에 특정 장치 드라이버가 필요하다는 것입니다.

가능한 한 VM용 에뮬레이트된 장치 대신 반가상화 장치를 사용하는 것이 좋습니다. 특히 I/O 집약적 애플리케이션을 실행하는 경우에도 마찬가지입니다. 반가상화 장치는 I/O 대기 시간을 줄이고 I/O 처리량을 높이며 경우에 따라 베어 메탈 성능에 매우 근접하게 합니다. 다른 반가상화 장치는 다른 방식으로 사용할 수 없는 VM에 기능을 추가합니다.

다음과 같은 유형의 반가상화 장치가 지원됩니다.

- 반가상화 네트워크 장치(**virtio-net**).
- 반가상화 스토리지 컨트롤러:
 - **Virtio-blk** - 블록 장치 에뮬레이션을 제공합니다.
 - **Virtio-scsi** - 보다 완전한 **SCSI** 에뮬레이션을 제공합니다.
- 반가상화 시계.

- 반가상화 직렬 장치(**virtio-serial**).
- **balloon** 장치(**virtio-balloon**)는 VM과 호스트 간에 메모리를 동적으로 배포하는 데 사용됩니다.
- 반가상화 난수 생성기(**virtio-rng**).
- 반가상화 그래픽 카드(**QXL**).

물리적 공유 장치

특정 하드웨어 플랫폼을 통해 VM은 다양한 하드웨어 장치 및 구성 요소에 직접 액세스할 수 있습니다. 이 프로세스를 **장치 할당** 또는 **패스스루**라고 합니다.

이러한 방식으로 연결된 경우 실제 시스템의 일부 측면을 VM에서 직접 사용할 수 있습니다. 이는 VM에서 사용할 때 장치에 뛰어난 성능을 제공합니다. 그러나 VM에 물리적으로 연결된 장치는 호스트에서 사용할 수 없게 되며 마이그레이션할 수도 없습니다.

하지만 일부 장치는 여러 VM에서 공유할 수 있습니다. 예를 들어 단일 물리적 장치는 특정 상황에서 여러 중재된 장치를 제공하여 개별 VM에 할당할 수 있습니다.

다음과 같은 유형의 패스스루 장치가 지원됩니다.

- **USB, PCI 및 SCSI 패스스루** - 게스트 소프트웨어에서 특정 기능을 사용할 수 있도록 일반적인 업계 표준 버스를 VM에 직접 노출합니다.
- **SR-IOV(Single-root I/O Virtualization) - PCI Express** 리소스의 하드웨어 강제 격리를 가능하게 하는 사양입니다. 따라서 단일 물리적 PCI 리소스를 가상 PCI 기능으로 분할하는 것이 안전하고 효율적입니다. 일반적으로 **NIC**(네트워크 인터페이스 카드)에 사용됩니다.
- **NPIV(N_Port ID Virtualization)** - 단일 **HBA**(호스트 버스 어댑터)를 여러 가상 포트와 공유하는 파이버 채널 기술입니다.
- **GPU 및 vGPU** - 특정 종류의 그래픽 또는 컴퓨팅 워크로드를 위한 액셀러레이터. 일부 GPU

는 VM에 직접 연결할 수 있지만 특정 유형은 기본 물리적 하드웨어를 공유하는 vGPU(가상 GPU)를 생성하는 기능도 제공합니다.

10.3. 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

10.3.1. 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 VM(가상 머신)을 생성하고 연결하는 방법을 보여줍니다. 일부 장치는 [RHEL 웹 콘솔을 사용하여 VM에 연결](#)할 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 8에서 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 생성하여 다음번 VM 시작 시 가상 디스크로 실행 중인 **testguest** VM에 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 **002**의 장치 **004**로 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 가져올 수 있습니다.

검증

장치가 추가되었는지 확인하려면 다음 중 하나를 수행하십시오.

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

예를 들어 다음 출력은 **testguest VM**의 구성을 보여주며 **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

10.3.2. 가상 머신에 연결된 장치 수정

연결된 가상 장치의 구성을 편집하여 **VM(가상 머신)**의 기능을 변경할 수 있습니다. 예를 들어 **VM**의 성능을 최적화하려면 호스트의 **CPU**에 맞게 가상 **CPU** 모델을 변경할 수 있습니다.

다음 절차에서는 **CLI(명령줄 인터페이스)**를 사용하여 가상 장치를 수정하기 위한 일반적인 지침을 제공합니다. **RHEL 8 웹 콘솔을 사용하여 VM에 연결된 일부 장치(예: 디스크 및 NIC)를 수정할 수도 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 출력을 파일로 전송합니다. 예를 들어, 다음은 **Motoko VM**의 구성을 **ko.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 사용하십시오.

예를 들어, 다음은 **shut-off testguest VM**의 **<cpu>** 구성을 지우고 이를 **host-model**로 설정합니다.

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

검증

장치가 수정되었는지 확인하려면 다음 중 하나를 수행하십시오.

-

VM을 실행하고 장치가 있는지 테스트하고 수정 사항을 반영합니다.

-

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

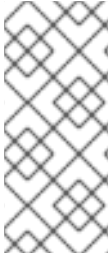
예를 들어 다음 출력은 **testguest VM**의 구성을 보여주고 CPU 모드가 **host-model**로 구성되었는지 확인합니다.

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

문제 해결

- 장치를 수정하면 VM이 부팅되지 않은 경우 **virsh define utility**를 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 구성을 중단할 가능성이 높기 때문에 이 방법을 더 광범위한 변경에는 사용하지 마십시오.

추가 리소스

- man virt-xml** 명령

10.3.3. 가상 머신에서 장치 제거

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

다음 절차에서는 CLI(명령줄 인터페이스)를 사용하여 VM(가상 머신)에서 가상 장치를 제거하는 방법을 보여줍니다. 디스크 또는 NIC와 같은 일부 장치도 **RHEL 8 웹 콘솔을 사용하여 VM에서 제거할 수 있습니다.**

사전 요구 사항

- 선택 사항: **virsh dumpxml vm-name** 을 사용하여 VM의 XML 구성을 백업하고 출력을 파일로 전송합니다. 예를 들어, 다음은 **Motoko VM**의 구성을 **ko.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 종료 후 실행 중인 **testguest** VM에서 **vdb** 로 표시된 스토리지 장치를 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** VM에서 **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 VM을 부팅할 수 없는 경우 **virsh define utility**를 사용하여 이전에 백업한 **XML** 구성 파일을 다시 로드하여 **XML** 구성을 복원합니다.

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

추가 리소스

•

man virt-xml 명령

10.4. 웹 콘솔을 사용하여 호스트 장치 관리

VM(가상 머신) 기능을 수정하려면 **RHEL 8** 웹 콘솔을 사용하여 **VM**에 연결된 호스트 장치를 관리할 수 있습니다.

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

웹 콘솔을 사용하여 다음을 수행할 수 있습니다.

- [장치 보기](#)
- [장치 연결](#)
- [장치 제거](#)

10.4.1. 웹 콘솔을 사용하여 가상 머신에 연결된 장치 보기

VM(가상 머신)에 연결된 장치를 추가하거나 수정하기 전에 **VM**에 이미 연결된 장치를 볼 수 있습니다. 다음 절차에서는 웹 콘솔을 사용하여 이러한 장치를 보는 방법을 설명합니다.

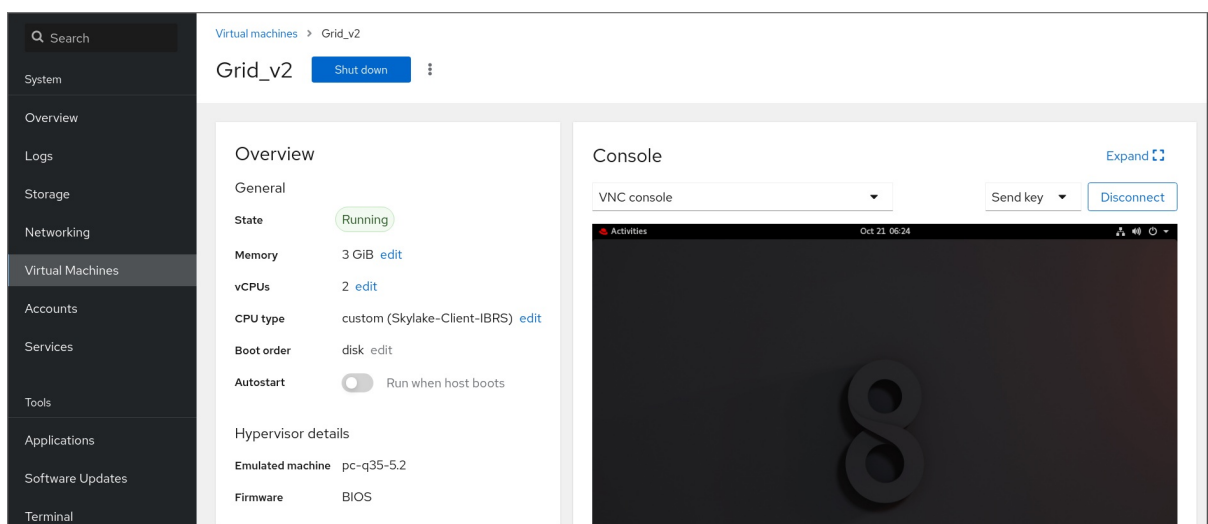
사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 **시스템에 설치되어** 있습니다.

절차

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

VM에 대한 자세한 정보가 포함된 새 페이지가 열립니다.



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

Host devices					Add host device
Type	Class	Model	Vendor	Source	
usb		Bluetooth wireless interface	Intel Corp.	Device 002	Remove
				Bus 001	

추가 리소스

-

[가상 장치 관리](#)

10.4.2. 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 8 Known Issues](#)를 참조하십시오.

사전 요구 사항

-

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



참고

PCI 장치를 **VM**에 연결할 때 **hostdev** 요소의 관리 속성을 생략하거나 **no**로 설정하지 마십시오. 이렇게 하면 **VM**에 전달할 때 **PCI** 장치는 호스트에서 자동으로 분리할 수 없습니다. 또한 **VM**을 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 VM**을 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.

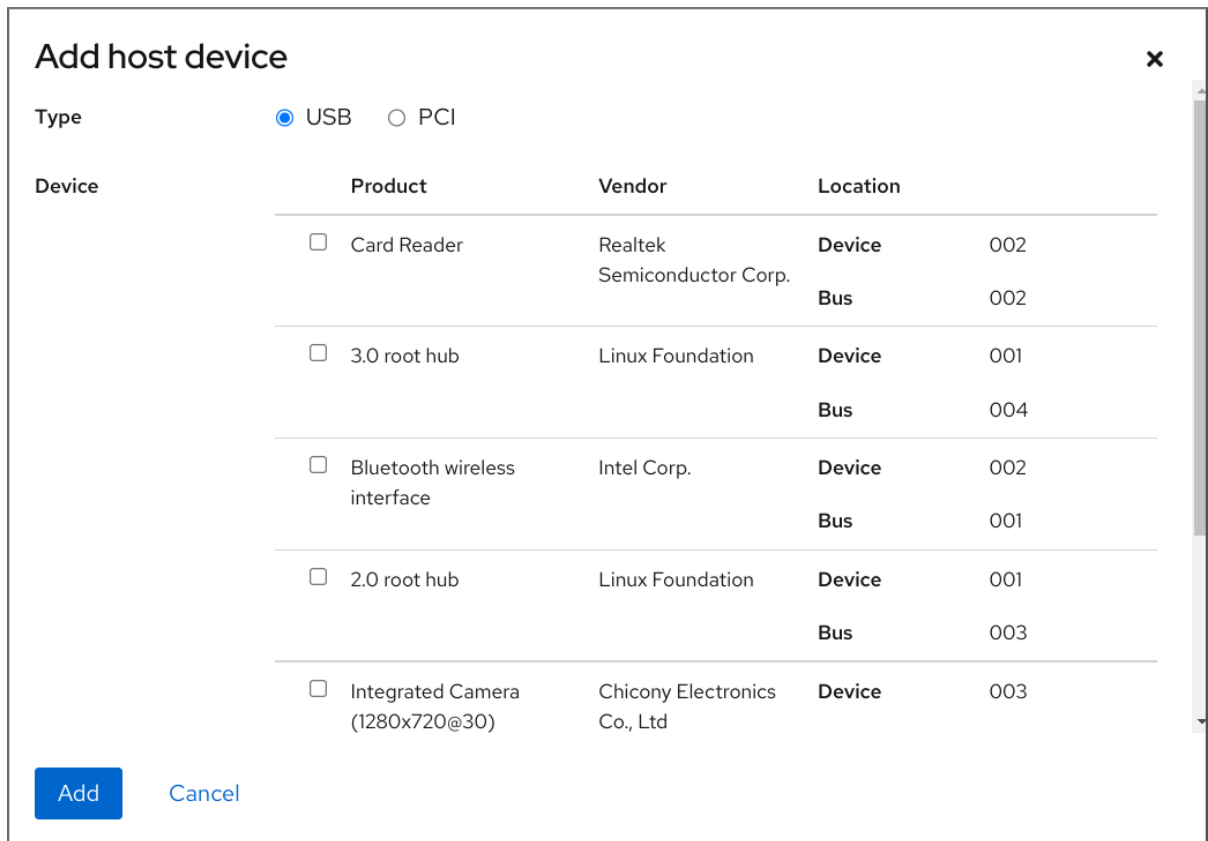
Host devices

Add host device

Type	Class	Model	Vendor	Source	
usb		Bluetooth wireless interface	Intel Corp.	Device	002
				Bus	001
					Remove

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.**

10.4.3. 웹 콘솔을 사용하여 가상 머신에서 장치 제거

리소스를 확보하거나 **VM**의 기능을 수정하려면, 웹 콘솔을 사용하여 **VM**을 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 8 Known Issues](#)를 참조하십시오.

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

```
# virsh edit Ag47
```

사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 **시스템에 설치되어** 있습니다.
- 선택 사항: **virsh dumpxml vm-name** 을 사용하여 **VM**의 **XML** 구성을 백업하고 출력을 파일로 전송합니다. 예를 들어, 다음은 **Motoko VM**의 구성을 **ko.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 **VM**을 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 **VM**에 연결된 장치와 장치 추가 또는 제거 옵션이 표시됩니다.

Host devices						Add host device
Type	Class	Model	Vendor	Source		
usb		Bluetooth wireless interface	Intel Corp.	Device	002	Remove
				Bus	001	

3.

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

제거 장치 확인 대화 상자가 나타납니다.

Remove usb host device 001.002

×

Confirm this action

Remove

Cancel

4.

제거를 클릭합니다.

장치가 **VM**에서 제거됩니다.

문제 해결

•

호스트 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 motoko.xml
```

10.5. 가상 USB 장치 관리

VM(가상 시스템)을 사용하는 경우 호스트 시스템에 연결된 플래시 드라이브 또는 웹 카메라와 같은 **USB** 장치에 액세스하고 제어할 수 있습니다. 이 시나리오에서 호스트 시스템은 장치 제어를 **VM**에 전달합니다. 이를 **USB-passthrough**라고도 합니다.

다음 섹션에서는 명령줄 사용에 대한 정보를 제공합니다.

- **VM에 USB 장치 연결**
- **VM에서 USB 장치 제거**

10.5.1. 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 통과하려는 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 장치 005로 버스 001에 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 VM**에 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 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](#)

10.5.2. 가상 머신에서 USB 장치 제거

VM(가상 시스템)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 장치 005로 버스 001에 연결됩니다.

```
# lsusb
[...]
Bus 001 Device 003: ID 2567:0a2b Intel Corp.
```

```
Bus 001 Device 005: ID 0407:6252 Kingston River 2.0
[...]
```

2.

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

예를 들어 다음 명령은 **Library VM**에서 버스 **001**의 장치 **005**로 호스트에 연결된 **USB** 플래시 드라이브를 제거합니다.

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



참고

실행 중인 **VM**에서 **USB** 장치를 제거하려면 이전 명령에 **--update** 인수를 추가합니다.

검증

- **VM**을 실행하고 장치가 장치 목록에서 제거되었는지 확인합니다.

추가 리소스

- **man virt-xml** 명령
- [가상 머신에 장치 연결](#)

10.5.3. 스마트 카드 판독기를 가상 머신에 연결

스마트 카드 판독기가 호스트에 연결된 경우 해당 호스트의 **VM**(가상 시스템)에서도 사용할 수 있습니다. **libvirt**는 게스트 **VM**에 스마트 카드 인터페이스를 제공하는 특수 가상 장치를 제공합니다. **SPICE** 원격 디스플레이 프로토콜을 사용하여 호스트에 대한 인증 요청을 터널링하는 **spicevmc** 장치 유형만 사용하는 것이 좋습니다.

스마트 카드 리더와 표준 장치 패스스루를 사용할 수는 있지만 이 방법은 호스트와 게스트 시스템 모두에서 장치를 사용할 수 없도록 합니다. 결과적으로 스마트 카드 리더를 **VM**에 연결할 때 호스트 시스템을 잠글 수 있습니다.

중요

RHEL 8에서 SPICE 원격 디스플레이 프로토콜이 더 이상 사용되지 않습니다. 스마트 카드 리더를 VM에 연결하는 데 권장되는 유일한 방법은 **SPICE** 프로토콜에 따라 달라지므로 게스트 VM에서 스마트 카드 사용도 **RHEL 8에서 더 이상 사용되지 않습니다.**

향후 주요 버전의 **RHEL**에서는 스마트 카드 리더를 VM에 연결하는 기능은 타사 원격 시각화 솔루션에서만 지원됩니다.

사전 요구 사항

- VM에 전달하려는 스마트 카드 리더가 호스트에 연결되어 있는지 확인합니다.
- **RHEL 8에서 스마트 카드 리더 유형이 지원되는 지** 확인합니다.

절차

- 가상 스마트 카드 리더 장치를 생성하고 VM에 연결합니다. 예를 들어 스마트 카드 리더를 **testguest** VM에 연결하려면 다음을 수행합니다.

```
# virt-xml testguest --add-device --smartcard mode=passthrough,type=spicevmc
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

참고

가상 스마트 카드 리더 장치를 실행 중인 VM에 연결하려면 이전 명령에 **--update** 인수를 추가합니다.

검증

1. VM의 XML 구성을 봅니다.

```
# virsh dumpxml testguest
```

2. XML 구성에 다음과 같은 스마트 카드 장치 정의가 포함되어 있는지 확인합니다.

```
<smartcard mode='passthrough' type='spicevmc'/>
```

10.6. 가상 광 드라이브 관리

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

다음 섹션에서는 명령줄 사용에 대한 정보를 제공합니다.

- [VM 에 드라이브 및 ISO 이미지 연결](#)
- [가상 광 드라이브에서 ISO 이미지 교체](#)
- [가상 광 드라이브에서 ISO 이미지 제거](#)
- [VM 에서 드라이브 제거](#)

10.6.1. 가상 머신에 광 드라이브 연결

가상 광 드라이브로 ISO 이미지를 연결하려면 가상 시스템(VM)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 VM**에 연결합니다.

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

검증

- VM을 실행하고 장치가 예상대로 작동하는지 테스트합니다.

추가 리소스

- **man virt-xml** 명령
- [가상 머신에 장치 연결](#)

10.6.2. 가상 광 드라이브에서 ISO 이미지 교체

가상 광 드라이브로 연결된 ISO 이미지를 VM(가상 머신)에 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 **targetda**의 **DN1 VM**에 연결된 **Doc10 ISO** 이미지를 **/Dvrs/current/** 디렉터리에 저장된 **Dr DN ISO** 이미지로 대체합니다.

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

검증

- **VM**을 실행하고 장치가 교체되고 예상대로 작동하는지 테스트합니다.

추가 리소스

- **man virt-xml** 명령

10.6.3. 가상 광 드라이브에서 ISO 이미지 제거

VM(가상 시스템)에 연결된 가상 광 드라이브에서 **ISO** 이미지를 제거하려면 **VM**의 **XML** 구성 파일을 편집합니다.

절차

1.

CD-ROM이 **VM**에 연결된 대상 장치를 찾습니다. 이 정보는 **VM**의 **XML** 구성 파일에서 확인할 수 있습니다.

예를 들어 다음 명령은 **CD-ROM**의 대상 장치가 **sda** 인 **DN1 VM**의 **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 VM**에 연결된 **CD** 드라이브에서 **DrDN ISO** 이미지를 제거합니다.

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

검증

- **VM**을 실행하고 이미지를 더 이상 사용할 수 있는지 확인합니다.

추가 리소스

- **man virt-xml** 명령

10.6.4. 가상 머신에서 광 드라이브 제거

VM(가상 시스템)에 연결된 광 드라이브를 제거하려면 **VM**의 **XML** 구성 파일을 편집합니다.

절차

1.

CD-ROM이 **VM**에 연결된 대상 장치를 찾습니다. 이 정보는 **VM**의 **XML** 구성 파일에서 확인할 수 있습니다.

예를 들어 다음 명령은 **CD-ROM**의 대상 장치가 **sda** 인 **DN1 VM**의 **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 VM** 에서 **targetda**로 연결된 광 드라이브를 제거합니다.

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

검증

- 장치가 **VM**의 **XML** 구성 파일에 더 이상 나열되지 않는지 확인합니다.

추가 리소스

- **man virt-xml** 명령

10.7. SR-IOV 장치 관리

에뮬레이트된 가상 장치는 하드웨어 네트워크 장치보다 많은 **CPU** 및 메모리를 사용하는 경우가 많습니다. 그러면 **VM**(가상 머신)의 성능을 제한할 수 있습니다. 그러나 가상화 호스트의 장치에서 **SR-IOV**(Single Root I/O Virtualization)를 지원하는 경우 이 기능을 사용하여 장치 성능을 개선하고 **VM**의 전반적인 성능도 향상시킬 수 있습니다.

10.7.1. SR-IOV란 무엇입니까?

SR-IOV(Single-root I/O Virtualization)는 단일 **PCI Express(PCIe)** 장치에서 **VF**(가상 기능)라는 여러 개의 개별 **PCI** 장치를 호스트 시스템에 제공할 수 있도록 하는 사양입니다. 이러한 각 장치:

- 은 원래 **PCIe** 장치와 동일한 또는 유사한 서비스를 제공할 수 있습니다.
- 호스트 **PCI** 버스의 다른 주소에 표시됩니다.
- **VFIO** 할당을 사용하여 다른 **VM**에 할당할 수 있습니다.

예를 들어 단일 **SR-IOV** 가능 네트워크 장치는 **VF**를 여러 **VM**에 제공할 수 있습니다. 모든 **VF**는 동일한 물리적 카드, 동일한 네트워크 연결 및 동일한 네트워크 케이블을 사용하지만 각 **VM**은 자체 하드웨어 네트워크 장치를 직접 제어하며 호스트에서 추가 리소스를 사용하지 않습니다.

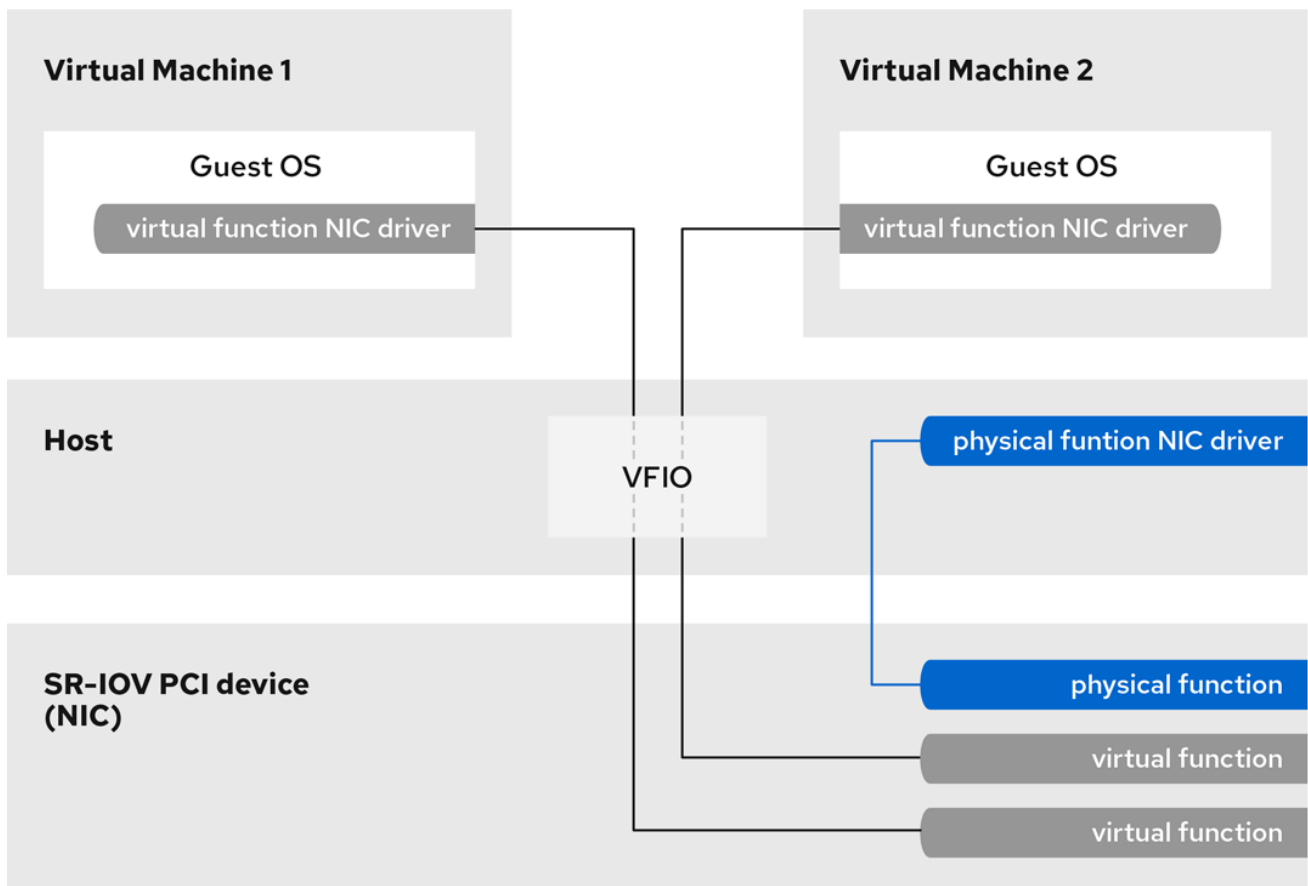
SR-IOV 작동 방식

SR-IOV 기능은 다음 PCIe 기능을 도입하여 수행할 수 있습니다.

- **PF(물리적 기능)** - 호스트에 장치(예: 네트워킹)의 기능을 제공하지만 **VF** 세트를 생성하고 관리할 수 있는 **PCIe** 기능입니다. 각 **SR-IOV** 지원 장치에는 하나 이상의 **PF**가 있습니다.
- **VF(가상 기능)** - 독립 장치 역할을 하는 경량 **PCIe** 기능입니다. 각 **VF**는 **PF**에서 파생됩니다. 장치에서 보유할 수 있는 최대 **VF** 수는 장치 하드웨어에 따라 다릅니다. 각 **VF**는 한 번에 하나의 **VM**에만 할당될 수 있지만 **VM**에는 여러 **VF**가 할당될 수 있습니다.

VM은 **VF**를 가상 장치로 인식합니다. 예를 들어 **SR-IOV** 네트워크 장치에서 생성한 **VF**는 실제 네트워크 카드가 호스트 시스템에 표시되는 것과 동일한 방식으로 할당된 **VM**의 네트워크 카드로 나타납니다.

그림 10.1. SR-IOV 아키텍처



해택

에뮬레이트된 장치가 아닌 **SR-IOV VF**를 사용하는 주된 이점은 다음과 같습니다.

- 성능 향상
- 호스트 CPU 및 메모리 리소스 사용 감소

예를 들어 vNIC가 VM에 연결된 VF는 물리적 NIC와 거의 동일한 수준에서 수행되며 반가상화 또는 에뮬레이트된 NIC보다 훨씬 효율적입니다. 특히 단일 호스트에서 여러 VF를 동시에 사용하는 경우 성능상의 이점이 매우 클 수 있습니다.

단점

- PF의 구성을 수정하려면 먼저 PF에 의해 노출된 VF 수를 0으로 변경해야 합니다. 따라서 이러한 VF가 할당된 VM에서 이러한 VF를 제공하는 장치도 제거해야 합니다.
- SR-IOV VF를 포함하여 VFIO로 할당된 장치가 연결된 VM은 다른 호스트로 마이그레이션할 수 없습니다. 할당된 장치를 에뮬레이트된 장치와 결합하여 이러한 제한 사항을 해결할 수 있는 경우도 있습니다. 예를 들어 할당된 네트워킹 VF를 에뮬레이트된 vNIC에 결합하고 마이그레이션 전에 VF를 제거할 수 있습니다.
- 또한 VFIO 할당 장치는 VM 메모리를 고정해야 하므로 VM의 메모리 사용량이 증가하고 VM에서 메모리 증대가 사용되지 않습니다.

추가 리소스

- [SR-IOV 할당에 지원되는 장치](#)

10.7.2. 가상 머신에 SR-IOV 네트워킹 장치 연결

Intel 또는 AMD 호스트의 VM(가상 머신)에 SR-IOV 네트워킹 장치를 연결하려면 호스트의 SR-IOV가 네트워킹 인터페이스에서 VF(가상 기능)를 생성하고 VF를 지정된 VM에 장치로 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 **ACS**(액세스 제어 서비스)를 사용하여 **PCIe** 토폴로지에 대한 **DMA**(직접 메모리 액세스) 격리를 제공합니다. 시스템 벤더를 사용하여 이를 확인합니다.

자세한 내용은 [SR-IOV 구현에 대한 하드웨어 고려 사항](#)을 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 fcb00000 (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 만드는 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. 이렇게 하려면 다음을 수행합니다.

○

Intel 호스트에서 VT-d를 활성화합니다.

■

Intel 호스트에서 여러 부팅 항목을 사용하는 경우:

A.

/etc/default/grub 파일을 편집하고 **GRUB_CMDLINE_LINUX** 행 끝에 **intel_iommu=on andiommu=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 호스트에서 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.

the **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. 이 작업을 수행하려면 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** 수의 **VF** 수입니다.

•

VF가 생성될 네트워크 인터페이스의 이름이 있는 **network -interface** 입니다.

다음 예제에서는 **eth1** 네트워크 인터페이스에서 2개의 VF를 생성합니다.

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

3.

VF가 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 만드는 데 사용한 네트워크 인터페이스에 대한 **udev** 규칙을 만들어 생성된 VF를 영구적으로 만듭니다. 예를 들어 **eth1** 인터페이스의 경우 **/etc/udev/rules.d/eth1.rules** 파일을 만들고 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** 인터페이스에 자동으로 사용할 수 있습니다. 영구 **SR-IOV** 장치가 필요하지 않은 경우 이 단계를 건너뛰니다.



주의

현재 **Broadcom NetXtreme II BCM57810** 어댑터에서 VF를 영구적으로 만들려고 할 때 위에서 설명한 설정이 제대로 작동하지 않습니다. 또한 이러한 어댑터를 기반으로 하는 VF를 **Windows VM**에 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 성공하면 게스트 운영 체제는 새 네트워크 인터페이스 카드를 감지합니다.

10.7.3. SR-IOV 할당에 지원되는 장치

SR-IOV에는 일부 장치를 사용할 수 없습니다. 다음 장치는 **RHEL 8**의 **SR-IOV**와 호환되는 것으로 테스트 및 검증되었습니다.

네트워킹 장치

- **Intel 82599ES 10 기가비트 이더넷 컨트롤러 - ixgbe** 드라이버 사용
- **Intel Ethernet Controller 710 Series - i40e** 드라이버 사용
- **Mellanox ConnectX-5 이더넷 어댑터 카드 - mlx5_core** 드라이버 사용
- **Intel Ethernet Network Adapter XXV710 - i40e** 드라이버를 사용합니다.
- **Intel 82576 기가비트 이더넷 컨트롤러 - igb** 드라이버를 사용합니다.
- **Broadcom NetXtreme II BCM57810 - bnx2x** 드라이버 사용

10.8. 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 **VM**이 **z/OS** 데이터 집합에 액세스하거나 할당된 **DASD**를 **z/OS** 머신에 제공할 수 있습니다.

사전 요구 사항

- 호스트 시스템은 **IBM Z** 하드웨어 아키텍처를 사용하며 **FICON** 프로토콜을 지원합니다.

- 대상 VM에서는 **Linux** 게스트 운영 체제를 사용합니다.

- **mdevctl** 패키지가 설치되어 있습니다.

```
# yum install mdevctl
```

- **driverctl** 패키지가 설치되어 있습니다.

```
# yum 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 있고 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 사용자의 장치의 검색된 하위 채널 식별자로 바꿉니다.

2.

이전 단계의 **lscss** 명령이 헤더 출력만 표시하고 장치 정보도 표시하지 않은 경우 다음 단계를 수행합니다.

a.

the **cio_ignore** 목록에서 장치를 제거합니다.

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

b.

게스트 OS에서 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

이렇게 하면 **0.0.29a8** 하위 채널을 **vfio_ccw** 에 영구적으로 바인딩합니다. 즉, 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 css unset-override 0.0.29a8
```

4.

UUID를 생성합니다.

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

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

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

6. 조정된 장치를 영구적으로 만듭니다.

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

7. 실행 중인 경우 **VM**을 종료합니다.

8. 조정된 장치를 **VM**에 연결합니다. 이 작업을 수행하려면 **virsh edit** 유틸리티를 사용하여 **VM**의 **XML** 구성을 편집하고 다음 섹션을 **XML**에 추가하고 **uuid** 값을 이전 단계에서 생성한 **UUID**로 바꿉니다.

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

검증

1. 조정된 **DASD** 장치에 **libvirt**가 할당된 식별자를 가져옵니다. 이를 위해 **VM**의 **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 **0.9**입니다.

2.

VM을 시작하고 게스트 **OS**에 로그인합니다.

3.

게스트 **OS**에서 **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 **OS**에서 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](#)

11장. 가상 머신용 스토리지 관리

실제 시스템과 마찬가지로 **VM**(가상 시스템)에는 데이터, 프로그램 및 시스템 파일을 위한 스토리지가 필요합니다. **VM** 관리자는 실제 또는 네트워크 기반 스토리지를 **VM**에 가상 스토리지로 할당할 수 있습니다. 기본 하드웨어와 관계없이 **VM**에 스토리지를 제공하는 방법도 수정할 수 있습니다.

다음 섹션에서는 다양한 유형의 **VM** 스토리지, 작동 방식 및 **CLI** 또는 웹 콘솔을 사용하여 관리하는 방법에 대한 정보를 제공합니다.

11.1. 가상 머신 스토리지 이해

VM(가상 시스템) 스토리지를 처음 사용하거나 작동 방식에 대해 잘 모르는 경우 다음 섹션에서는 **VM** 스토리지의 다양한 구성 요소, 작동 방식, 관리 기본 사항 및 **Red Hat**이 제공하는 지원 솔루션에 대한 일반적인 개요를 제공합니다.

다음에 대한 정보를 찾을 수 있습니다.

- [스토리지 풀](#)
- [스토리지 볼륨](#)
- [libvirt를 사용하여 스토리지 관리](#)
- [VM 스토리지 개요](#)
- [지원 및 지원되지 않는 스토리지 풀 유형](#)

11.1.1. 스토리지 풀 소개

스토리지 풀은 **libvirt** 에서 관리하는 파일, 디렉터리 또는 스토리지 장치로, **VM**(가상 시스템)에 스토리지를 제공합니다. 스토리지 풀을 스토리지 볼륨으로 분할하거나 **VM** 이미지를 추가 스토리지로 **VM**에 연결할 수 있습니다.

또한 여러 **VM**이 동일한 스토리지 풀을 공유하므로 스토리지 리소스를 더 효과적으로 할당할 수 있습니다.

니다.

- 스토리지 풀은 영구적이거나 일시적일 수 있습니다.
 - 영구 스토리지 풀은 호스트 시스템을 다시 시작해도 유지됩니다. **virsh pool-define** 을 사용하여 영구 스토리지 풀을 생성할 수 있습니다.
 - 임시 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. 로컬 디렉터리에는 로컬 디렉터리, 직접 연결된 디스크, 물리 파티션, 로컬 볼륨 관리(LVM) 볼륨 그룹이 포함됩니다.

로컬 스토리지 풀은 마이그레이션이 필요하지 않거나 다수의 VM이 있는 개발, 테스트 및 소규모 배포에 유용합니다.
- 네트워크(공유) 스토리지 풀

네트워크 스토리지 풀에는 표준 프로토콜을 사용하여 네트워크에서 공유되는 스토리지 장치가 포함됩니다.

11.1.2. 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 유형의 추상화입니다. 스토리지 볼륨은 기본 하드웨어에 관계없이 디스크와 같은 로컬 스토리지 장치로 VM에 제공됩니다.

호스트 시스템에서 스토리지 볼륨은 이름 및 스토리지 풀에서 파생되는 스토리지 풀의 식별자로 참조됩니다. **virsh** 명령줄에서 **--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
```

11.1.3. libvirt를 사용한 스토리지 관리

libvirt 원격 프로토콜을 사용하여 **VM** 스토리지의 모든 측면을 관리할 수 있습니다. 이러한 작업은 원격 호스트에서도 수행할 수 있습니다. 따라서 **libvirt** 를 사용하는 관리 애플리케이션(예: **RHEL** 웹 콘솔)을 사용하여 **VM** 스토리지를 구성하는 데 필요한 모든 작업을 수행할 수 있습니다.

libvirt API를 사용하여 스토리지 풀의 볼륨 목록을 쿼리하거나 해당 스토리지 풀에서 용량, 할당 및 사용 가능한 스토리지에 관한 정보를 가져올 수 있습니다. 이를 지원하는 스토리지 풀의 경우 **libvirt API**를 사용하여 스토리지 볼륨을 생성, 복제, 크기 조정 및 삭제할 수도 있습니다. 또한 **libvirt API**를 사용하여 스토리지 볼륨에 데이터를 업로드하고, 스토리지 볼륨에서 데이터를 다운로드하거나, 스토리지 볼륨에서 데이터를 초기화할 수 있습니다.

11.1.4. 스토리지 관리 개요

다음 예제에서는 스토리지 관리에 사용할 수 있는 옵션을 설명하기 위해 **mount -t nfs.example.com:/path/to/share /path/to/data** 를 사용하는 샘플 **NFS** 서버에 대해 이야기합니다.

스토리지 관리자:

- 가상화 호스트에서 **NFS** 스토리지 풀을 정의하여 내보낸 서버 경로 및 클라이언트 대상 경로를 설명할 수 있습니다. 따라서 **libvirt** 는 **libvirt**가 시작될 때 또는 **libvirt** 가 실행되는 동안 필요에 따라 스토리지를 자동으로 마운트할 수 있습니다.
- 이름별로 **VM**에 스토리지 풀과 스토리지 볼륨을 추가하기만 하면 됩니다. 대상 경로를 볼륨에 추가할 필요가 없습니다. 따라서 대상 클라이언트 경로가 변경되더라도 **VM**에 영향을 미치지 않습니다.
- 자동 시작되도록 스토리지 풀을 구성할 수 있습니다. 이 작업을 수행하면 **libvirt** 가 시작될 때 지정된 디렉터리에 **NFS** 공유 디스크를 자동으로 마운트합니다. **libvirt** 는 명령 **nfs.example.com:/path/to/share /vmdata**와 유사하게 지정된 디렉터리에 공유를 마운트합니다.

- **libvirt API**를 사용하여 스토리지 볼륨 경로를 쿼리할 수 있습니다. 이러한 스토리지 볼륨은 기본적으로 **NFS** 공유 디스크에 있는 파일입니다. 그런 다음 **VM**의 블록 장치에 대한 소스 스토리지를 설명하는 **VM**의 **XML** 정의의 섹션으로 이러한 경로를 복사할 수 있습니다.
- **NFS**의 경우 **libvirt API**를 사용하는 애플리케이션을 사용하여 스토리지 풀(**NFS** 공유의 파일)에서 크기(공유 스토리지 용량)의 한도까지 스토리지 볼륨을 생성하고 삭제할 수 있습니다.

일부 스토리지 풀 유형에서 볼륨 생성 및 삭제를 지원하는 것은 아닙니다.
- 더 이상 필요하지 않은 경우 스토리지 풀을 중지할 수 있습니다. 스토리지풀(풀-오버)을 중지하면 시작 작업이 취소됩니다. 이 경우 **NFS** 공유의 마운트를 해제합니다. 명령 이름이 제안하는 내용에도 불구하고 공유의 데이터는 삭제 작업에서 수정하지 않습니다. 자세한 내용은 **man virsh**를 참조하십시오.

11.1.5. 지원되지 않는 스토리지 풀 유형

지원되는 스토리지 풀 유형

다음은 **RHEL**에서 지원하는 스토리지 풀 유형 목록입니다.

- 디렉터리 기반 스토리지 풀
- 디스크 기반 스토리지 풀
- 파티션 기반 스토리지 풀
- **GlusterFS** 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 **RHEL**에서 지원하지 않는 **libvirt** 스토리지 풀 유형 목록입니다.

- **sheepdog** 기반 스토리지 풀
- **Vstorage** 기반 스토리지 풀
- **ZFS** 기반 스토리지 풀

11.2. 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 GlusterFS 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**

11.2.1. CLI를 사용하여 스토리지 풀 정보 보기

CLI를 사용하면 스토리지 풀에 대한 제한된 또는 전체 세부 정보가 있는 모든 스토리지 풀 목록을 볼 수 있습니다. 나열된 스토리지 풀도 필터링할 수 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

11.2.2. CLI를 사용하여 디렉터리 기반 스토리지 풀 생성

디렉터리 기반 스토리지 풀은 기존 마운트된 파일 시스템의 디렉터리를 기반으로 합니다. 이는 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'"
```

명령이 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** 디렉터를 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
```

생성하려는 스토리지 풀의 **XML** 구성이 이미 있는 경우 **XML**을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 디렉터리 [기반 스토리지 풀 매개 변수](#)를 참조하십시오.

2. 스토리지 풀 대상 경로 생성

virsh pool-build 명령을 사용하여 미리 포맷된 파일 시스템 스토리지 풀의 스토리지 풀 대상 경로를 생성하고, 스토리지 소스 장치를 초기화하며, 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.

폴이 생성되었는지 확인합니다.

virsh pool-list 명령을 사용하여 폴이 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 생성 시 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 자동 시작되도록 스토리지 풀을 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, 자동 시작이 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
```

11.2.3. CLI를 사용하여 디스크 기반 스토리지 풀 생성

디스크 기반 스토리지 풀에서 풀은 디스크 파티션을 기반으로 합니다. 예를 들어 **VM**(가상 머신) 스토리지로 전용 전체 디스크 파티션을 사용하려는 경우 유용합니다. **virsh** 유틸리티를 사용하여 디스크 기반 스토리지 풀을 만들 수 있습니다.

사전 요구 사항

- 하이퍼바이저가 디스크 기반 스토리지 풀을 지원하는지 확인하십시오.

```
# virsh pool-capabilities | grep "'disk' supported='yes'"
```

명령에서 출력을 표시하는 경우 디스크 기반 풀이 지원됩니다.

- 스토리지 풀의 기반이 되는 장치를 준비합니다. 이를 위해 파티션(예: **/dev/sdb1**) 또는 **LVM** 볼륨을 선호합니다. **VM**에 전체 디스크 또는 블록 장치(예: **/dev/sdb**)에 대한 쓰기 권한이 있는 경우 **VM**이 파티션을 지정하거나 자체 **LVM** 그룹을 만들 수 있습니다. 이로 인해 호스트에서 시스템 오류가 발생할 수 있습니다.

그러나 스토리지 풀에 전체 블록 장치를 사용해야 하는 경우 장치의 중요한 파티션을 **GRUB**의 **os-prober** 기능에서 보호하는 것이 좋습니다. 이를 위해 **/etc/default/grub** 파일을 편집하고 다음 구성 중 하나를 적용합니다.

- **os-prober**를 비활성화합니다.

```
GRUB_DISABLE_OS_PROBER=true
```

- **os-prober**가 특정 파티션을 찾지 못하게 합니다. 예를 들면 다음과 같습니다.

```
GRUB_OS_PROBER_SKIP_LIST="5ef6313a-257c-4d43@/dev/sdb1"
```

•

스토리지 풀을 생성하기 전에 선택한 스토리지 장치의 데이터를 백업하십시오. 사용 중인 **libvirt** 버전에 따라 스토리지 풀에 디스크를 지정하면 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
```

생성하려는 스토리지 풀의 **XML** 구성이 이미 있는 경우 **XML**을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 [디스크 기반 스토리지 풀 매개변수](#)를 참조하십시오.

2.

스토리지 풀 대상 경로 생성

virsh pool-build 명령을 사용하여 미리 포맷된 파일 시스템 스토리지 풀의 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 빌드는 디스크 기반, 파일 시스템 기반 및 논리적 스토리지 풀에만 필요합니다. **libvirt** 에서 소스 스토리지 장치의 데이터 형식이 선택한 스토리지 풀 유형과 다른 것을 감지하면 **덮어쓰기** 옵션이 지정되지 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 생성 시 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 자동 시작되도록 스토리지 풀을 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, 자동 시작이 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
```

11.2.4. 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 확인하십시오.

```
# virsh pool-capabilities | grep "'fs' supported='yes'"
```

명령에서 출력을 표시하는 경우 파일 기반 풀이 지원됩니다.

- 스토리지 풀의 기반이 되는 장치를 준비합니다. 이를 위해 파티션(예: **/dev/sdb1**) 또는 LVM 볼륨을 선호합니다. VM에 전체 디스크 또는 블록 장치(예: **/dev/sdb**)에 대한 쓰기 권한이 있는 경우 VM이 파티션을 지정하거나 자체 LVM 그룹을 만들 수 있습니다. 이로 인해 호스트에서 시스템 오류가 발생할 수 있습니다.

그러나 스토리지 풀에 전체 블록 장치를 사용해야 하는 경우 장치의 중요한 파티션을 **GRUB**의 **os-prober** 기능에서 보호하는 것이 좋습니다. 이를 위해 **/etc/default/grub** 파일을 편집하고 다음 구성 중 하나를 적용합니다.

- **os-prober**를 비활성화합니다.

```
GRUB_DISABLE_OS_PROBER=true
```

- **os-prober**가 특정 파티션을 찾지 못하게 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 마운트되는 **guest_images_fs**라는 스토리지 풀을 생성하려면 다음을 실행합니다.

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

생성하려는 스토리지 풀의 XML 구성이 이미 있는 경우 XML을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 [파일 시스템 기반 스토리지 풀 매개 변수](#)를 참조하십시오.

2.

스토리지 풀 대상 경로 정의

virsh pool-build 명령을 사용하여 미리 포맷된 파일 시스템 스토리지 풀의 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 생성 시 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 자동 시작되도록 스토리지 풀을 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, 자동 시작이 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 타겟 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
```

11.2.5. CLI를 사용하여 GlusterFS 기반 스토리지 풀 생성

GlusterFS는 **FUSE(User System in Userspace)** 소프트웨어 인터페이스를 사용하는 사용자 공간 파일 시스템입니다. **Gluster** 서버에 스토리지 풀을 사용하려면 **virsh** 유틸리티를 사용하여 **GlusterFS** 기반 스토리지 풀을 생성할 수 있습니다.

사전 요구 사항

- 호스트에서 **GlusterFS** 기반 스토리지 풀을 생성하려면 먼저 **Gluster**를 준비합니다.

- a. 다음 명령으로 상태를 나열하여 **Gluster** 서버의 **IP** 주소를 가져옵니다.

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

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

- b. 설치되지 않은 경우 **glusterfs-fuse** 패키지를 설치합니다.

- c. 활성화되지 않은 경우 **virt_use_fusefs** 부울을 활성화합니다. 활성화되어 있는지 확인합니다.

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

- 하이퍼바이저가 **GlusterFS** 기반 스토리지 풀을 지원하는지 확인합니다.

```
# virsh pool-capabilities | grep "'gluster' supported='yes'"
```

명령에서 출력을 표시하는 경우 **GlusterFS** 기반 풀이 지원됩니다.

절차

1. 스토리지 풀 생성

virsh pool-define-as 명령을 사용하여 **GlusterFS** 기반 스토리지 풀을 정의하고 생성합니다.

예를 들어 IP 111.222.111.222 와 함께 **gluster-vol1** 이라는 **Gluster** 서버를 사용하고 **Gluster** 서버의 루트 디렉터리에 마운트되는 **guest_images_glusterfs** 라는 스토리지 풀을 생성하려면 다음을 수행합니다.

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

생성하려는 스토리지 풀의 **XML** 구성이 이미 있는 경우 **XML**을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 [GlusterFS 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_glusterfs	inactive	no

3.

스토리지 풀 시작

virsh pool-start 명령을 사용하여 스토리지 풀을 마운트합니다.

```
# virsh pool-start guest_images_glusterfs
Pool guest_images_glusterfs started
```



참고

virsh pool-start 명령은 영구 스토리지 풀에만 필요합니다. 임시 스토리지 풀은 생성 시 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 자동 시작되도록 스토리지 풀을 구성합니다.

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

검증

- **virsh pool-info** 명령을 사용하여 스토리지 풀이 **running** 상태인지 확인합니다. 보고된 크기가 예상대로, 자동 시작이 올바르게 구성되었는지 확인합니다.

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

11.2.6. CLI를 사용하여 iSCSI 기반 스토리지 풀 생성

iSCSI(Internet Small Computer Systems Interface)는 데이터 스토리지 기능을 연결하기 위한 IP 기반 스토리지 네트워킹 표준입니다. iSCSI 서버에 스토리지 풀이 필요한 경우 **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'"
```

명령에서 출력을 표시하는 경우 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** IQN을 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
```

생성하려는 스토리지 풀의 XML 구성이 이미 있는 경우 XML을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 [iSCSI 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_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 생성 시 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 자동 시작되도록 스토리지 풀을 구성합니다.

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

•

virsh pool-info 명령을 사용하여 스토리지 풀이 **running** 상태인지 확인합니다. 보고된 크기가 예상대로, 자동 시작이 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
```

11.2.7. CLI를 사용하여 LVM 기반 스토리지 풀 생성

LVM 볼륨 그룹의 일부인 스토리지 풀이 필요한 경우 **virsh** 유틸리티를 사용하여 LVM 기반 스토리지 풀을 생성할 수 있습니다.

권장 사항

LVM 기반 스토리지 풀을 만들기 전에 다음 사항에 유의하십시오.

•

LVM 기반 스토리지 풀은 LVM의 완전한 유연성을 제공하지 않습니다.

•

libvirt 는 썬 논리 볼륨을 지원하지만 썬 스토리지 풀의 기능을 제공하지는 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 포함된 볼륨 그룹을 만들려면 LVM 유틸리티를 사용하는 대신 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 절차와 같이 호스트의 기존 볼륨 그룹을 사용하는 경우 아무것도 지워지지 않습니다.

사전 요구 사항

•

하이퍼바이저가 **LVM** 기반 스토리지 풀을 지원하는지 확인합니다.

```
# virsh pool-capabilities | grep "'logical' supported='yes'"
```

명령이 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 마운트되는 **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
```

생성하려는 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 생성 시 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 자동 시작되도록 스토리지 풀을 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, 자동 시작이 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
```

11.2.8. CLI를 사용하여 NFS 기반 스토리지 풀 생성

NFS(네트워크 파일 시스템) 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 디렉토리 **/home/net_mount**에 IP **111.222.111.222**가 마운트된 **NFS** 서버를 사용하는 **guest_images_net fs**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'
```

생성하려는 스토리지 풀의 **XML** 구성이 이미 있는 경우 **XML**을 기반으로 풀을 정의할 수도 있습니다. 자세한 내용은 **NFS 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_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 생성 시 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 자동 시작되도록 스토리지 풀을 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, 자동 시작이 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
```

11.2.9. CLI를 사용하여 vHBA 장치가 있는 SCSI 기반 스토리지 풀 생성

Small Computer System Interface(SCSI) 장치에 스토리지 풀을 사용하려면 호스트가 **vHBA**(가상 호스트 버스 어댑터)를 사용하여 **SCSI** 장치에 연결할 수 있어야 합니다. 그런 다음 **virsh** 유틸리티를 사용하여 **SCSI** 기반 스토리지 풀을 만들 수 있습니다.

사전 요구 사항

•

하이퍼바이저가 **SCSI** 기반 스토리지 풀을 지원하는지 확인합니다.

```
# virsh pool-capabilities | grep "'scsi' supported='yes'"
```

명령에서 출력을 표시하는 경우 **SCSI** 기반 풀이 지원됩니다.

-

vHBA 장치를 사용하여 **SCSI** 기반 스토리지 풀을 생성하기 전에 **vHBA**를 생성합니다. 자세한 내용은 [vHBA 생성](#) 을 참조하십시오.

절차

- 1.

스토리지 풀 생성

virsh pool-define-as 명령을 사용하여 **vHBA**를 사용하여 **SCSI** 스토리지 풀을 정의하고 생성합니다. 예를 들어 다음은 **scsi_host3** 상위 어댑터로 식별되는 **vHBA**, 전 세계 포트 번호 **5001a4ace3 ee047d** 및 세계 전체 노드 번호 **5001a4a93526d 0a1** 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
```

생성하려는 스토리지 풀의 **XML** 구성이 이미 있는 경우 **XML**을 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 생성 시 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 자동 시작되도록 스토리지 풀을 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, 자동 시작이 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
```

11.2.10. 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 삭제할 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 상주하는 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
```

11.3. 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.
 - **vHBA** 장치를 사용하여 **SCSI** 기반 스토리지 풀을 생성합니다.
- 스토리지 풀 제거.
- 스토리지 풀을 비활성화 합니다.

11.3.1. 웹 콘솔을 사용하여 스토리지 풀 정보 보기

웹 콘솔을 사용하면 시스템에서 사용 가능한 스토리지 풀에 대한 자세한 정보를 볼 수 있습니다. 스토리지 풀을 사용하여 가상 시스템의 디스크 이미지를 생성할 수 있습니다.

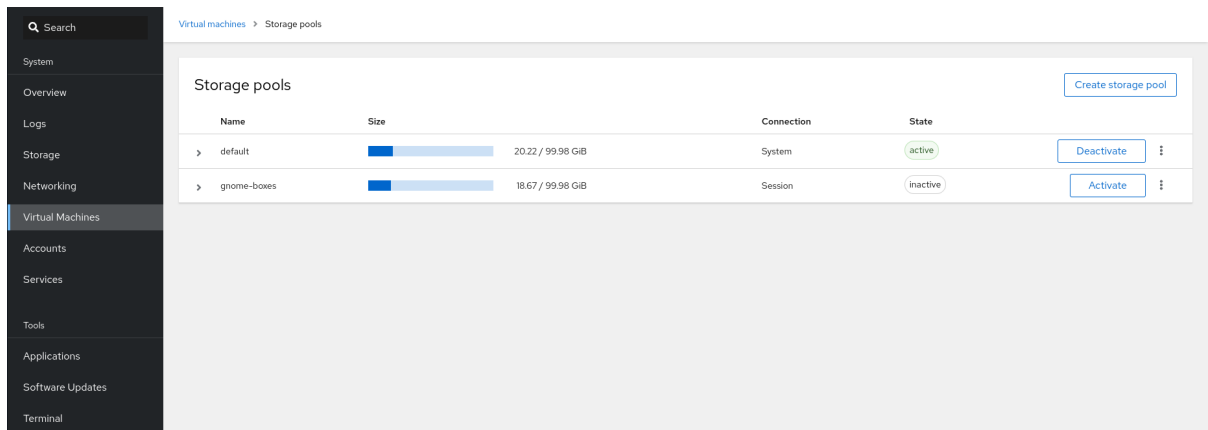
사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 **시스템에 설치되어** 있습니다.

절차

1. 가상 머신 인터페이스 상단에 있는 **Storage Pools** (스토리지 풀)를 클릭합니다.

구성된 스토리지 풀 목록을 표시하는 **Storage Pools**(스토리지 풀) 창이 표시됩니다.



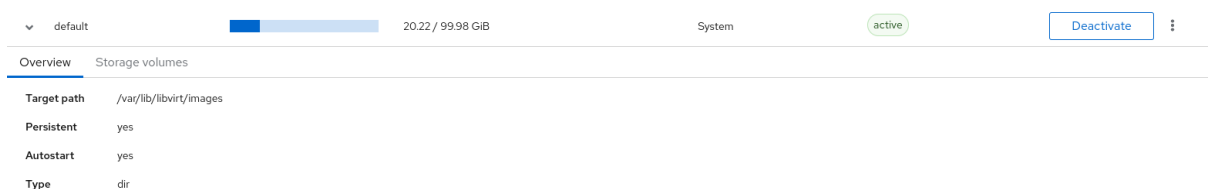
이 정보에는 다음이 포함됩니다.

- **name** - 스토리지 풀의 이름입니다.
- **size** - 스토리지 풀의 현재 할당 및 총 용량입니다.
- **연결** - 스토리지 풀 액세스에 사용되는 연결입니다.
- **state** - 스토리지 풀의 상태입니다.

2.

정보를 보려는 스토리지 풀 옆에 있는 화살표를 클릭합니다.

행이 확장되어 선택한 스토리지 풀에 대한 자세한 정보가 포함된 **Overview(개요)** 창을 표시합니다.



정보에는 다음이 포함됩니다.

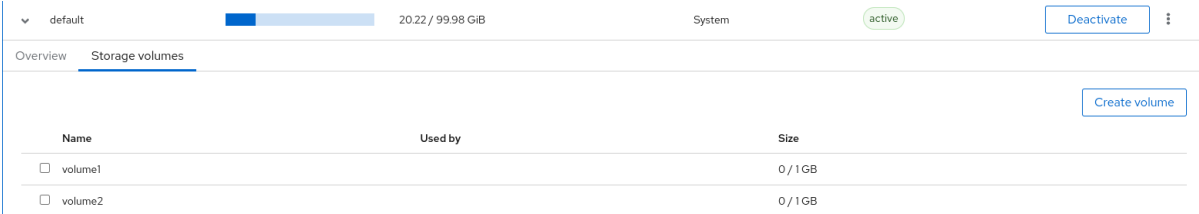
- **타겟 경로** - 디렉터리가 지원하는 스토리지 풀 유형의 소스(예: **dir** 또는 **netfs**)입니다.

- **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 **Storage(스토리지) Volumes (볼륨)** 를 클릭합니다.

Storage Volumes(스토리지 볼륨) 창이 표시되고 구성된 스토리지 볼륨 목록이 표시됩니다.



▼ default	20.22 / 99.98 GiB	System	active	Deactivate	⋮
Overview	Storage volumes				
				Create volume	
Name	Used by	Size			
☐ volume1		0 / 1GB			
☐ volume2		0 / 1GB			

정보에는 다음이 포함됩니다.

- **name** - 스토리지 볼륨의 이름입니다.
- 현재 스토리지 볼륨을 사용하고 있는 **VM** 에서 사용합니다.
- **size** - 볼륨의 크기입니다.

추가 리소스

- [웹 콘솔을 사용하여 가상 머신 정보 보기](#)

11.3.2. 웹 콘솔을 사용하여 디렉터리 기반 스토리지 풀 생성

디렉터리 기반 스토리지 풀은 기존 마운트된 파일 시스템의 디렉터리를 기반으로 합니다. 이는 예를 들어 파일 시스템의 나머지 공간을 다른 용도로 사용하려는 경우 유용합니다.

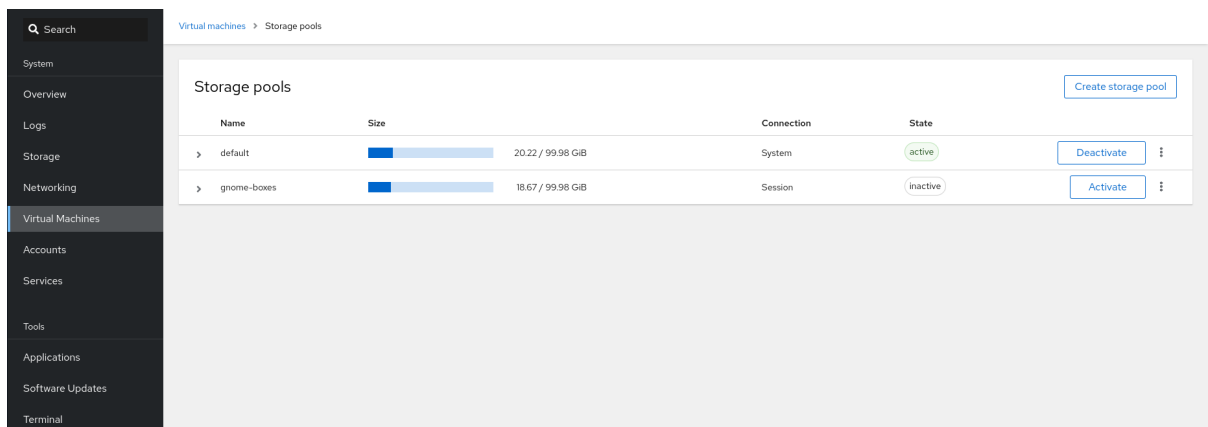
사전 요구 사항

- 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.

Create storage pool

×

Name

Storage pool name

Type

Filesystem directory ▼

Target path

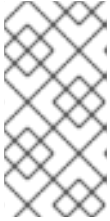
Path on host's filesystem ▼

Startup

☒ Start pool when host boots

Create

Cancel



참고

드롭다운 메뉴에 **Filesystem** 디렉터리 옵션이 표시되지 않으면 하이퍼바이저에서 디렉터리 기반 스토리지 풀을 지원하지 않습니다.

5.

다음 정보를 입력합니다.

- 타겟 경로 - 디렉터리가 지원하는 스토리지 풀 유형의 소스(예: **dir** 또는 **netfs**)입니다.
- **Startup** - 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](#)

11.3.3. 웹 콘솔을 사용하여 NFS 기반 스토리지 풀 생성

NFS 기반 스토리지 풀은 서버에서 호스팅되는 파일 시스템을 기반으로 합니다.

사전 요구 사항

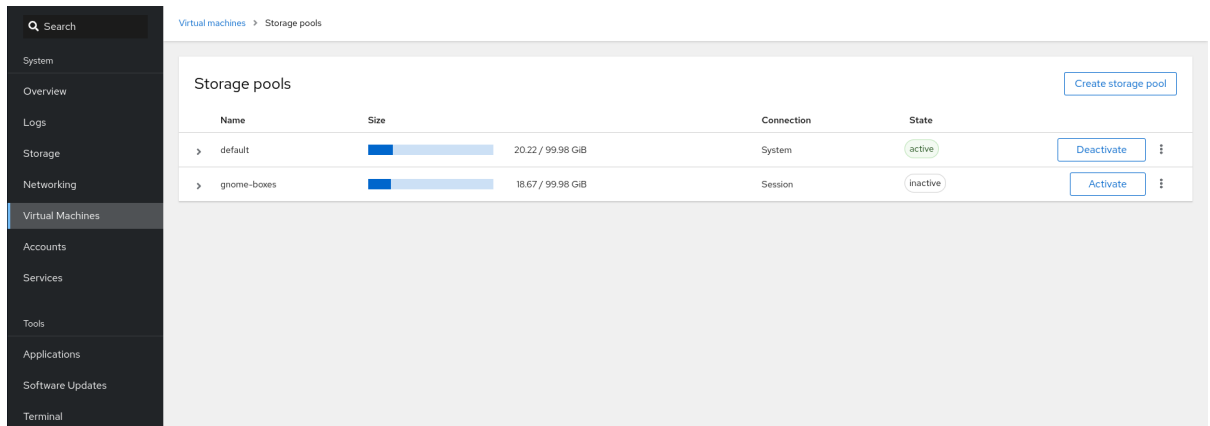
- 웹 콘솔 **VM** 플러그인이 [시스템에 설치되어](#) 있습니다.

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.

Create storage pool

×

Name

Storage pool name

Type

Network file system

Target path

Path on host's filesystem

Host

Host name

Source path

The directory on the server being exported

Startup

☒ Start pool when host boots

Create

Cancel



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. 이 경로는 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.
- **Startup** - 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](#)

11.3.4. 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

●

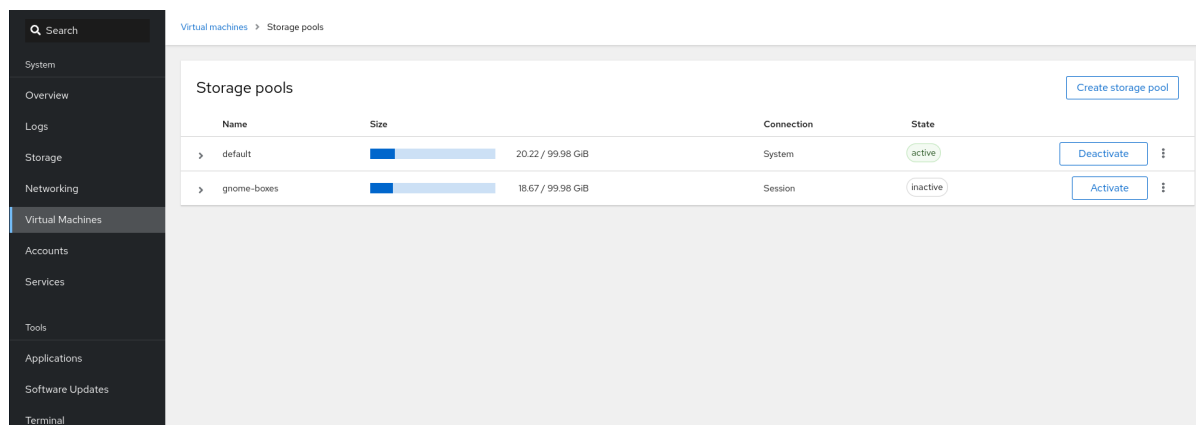
웹 콘솔 VM 플러그인이 **시스템에 설치되어** 있습니다.

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.

Create storage pool

×

Name

Storage pool name

Type

iSCSI target

▼

Target path

Path on host's filesystem

▼

Host

Host name

Source path

iSCSI target IQN

Startup

☒ Start pool when host boots

Create

Cancel

5.

정보의 나머지 부분을 입력하십시오:

- 대상 경로 - 대상을 지정하는 경로입니다. 이 경로는 스토리지 풀에 사용되는 경로입니다.
- host - iSCSI 서버의 호스트 이름 또는 IP 주소입니다.
- 소스 경로 - iSCSI 대상의 고유한 IQN(iSCSI Qualified Name)입니다.
- Startup - 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

11.3.5. 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

-

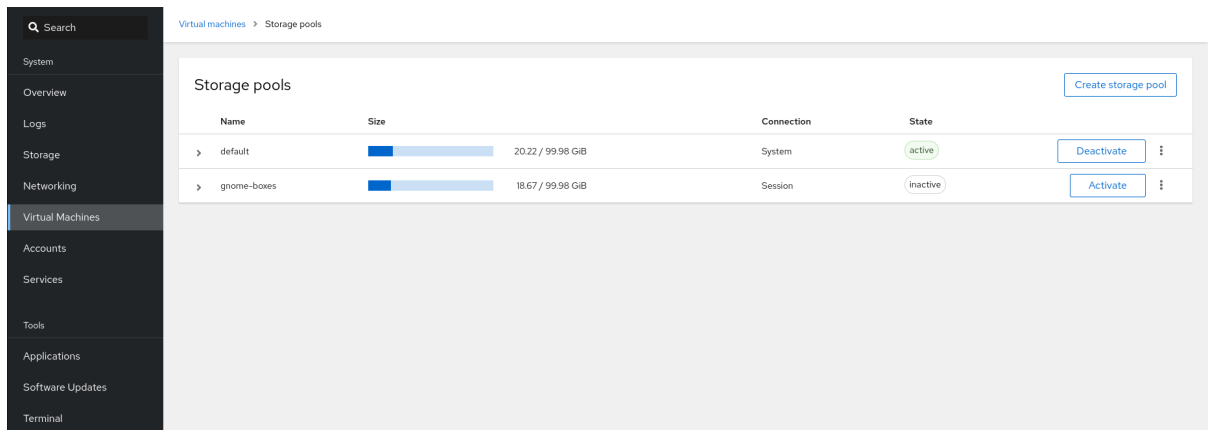
웹 콘솔 **VM** 플러그인이 **시스템에 설치되어** 있습니다.

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.

Create storage pool

×

Name

Storage pool name

Type

Physical disk device

▼

Target path

Path on host's filesystem

▼

Source path

Physical disk device on host

▼

Format

dos

▼

Startup

☒ Start pool when host boots

Create

Cancel



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. 이 경로는 스토리지 풀에 사용되는 경로입니다.
- 소스 경로 - 스토리지 장치를 지정하는 경로입니다. 예를 들면 `/dev/sdb` 입니다.
- Format - 파티션 테이블의 유형입니다.
- Startup - 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](#)

11.3.6. 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 쉘 논리 볼륨을 지원하지만 쉘 스토리지 풀의 기능을 제공하지는 않습니다.**
- **LVM 기반 스토리지 풀에는 전체 디스크 파티션이 필요합니다. virsh 명령을 사용하여 새 파티션 또는 장치를 활성화하면 파티션이 포맷되고 모든 데이터가 삭제됩니다. 이 절차와 같이 호스트의 기존 볼륨 그룹을 사용하는 경우 아무것도 지워지지 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

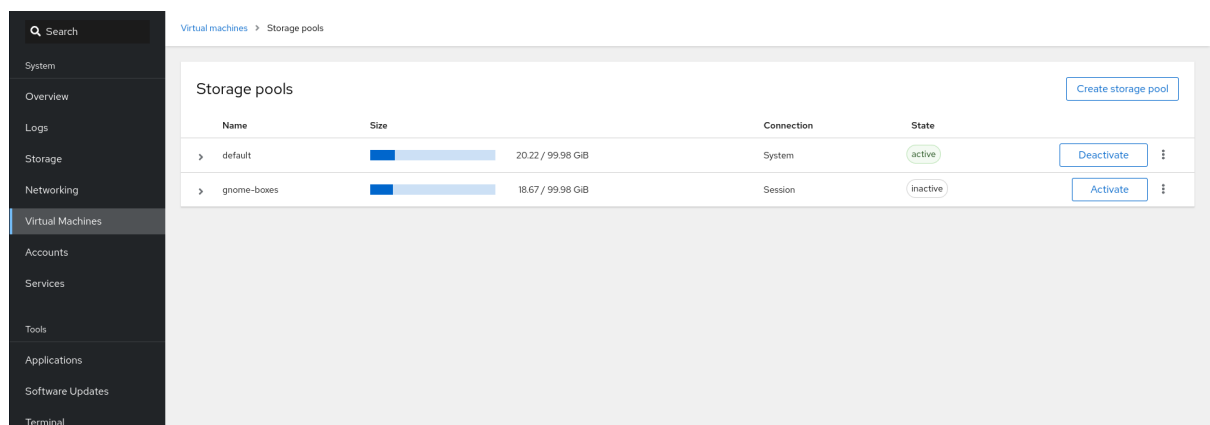
- 웹 콘솔 VM 플러그인이 **시스템에 설치되어** 있습니다.

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.

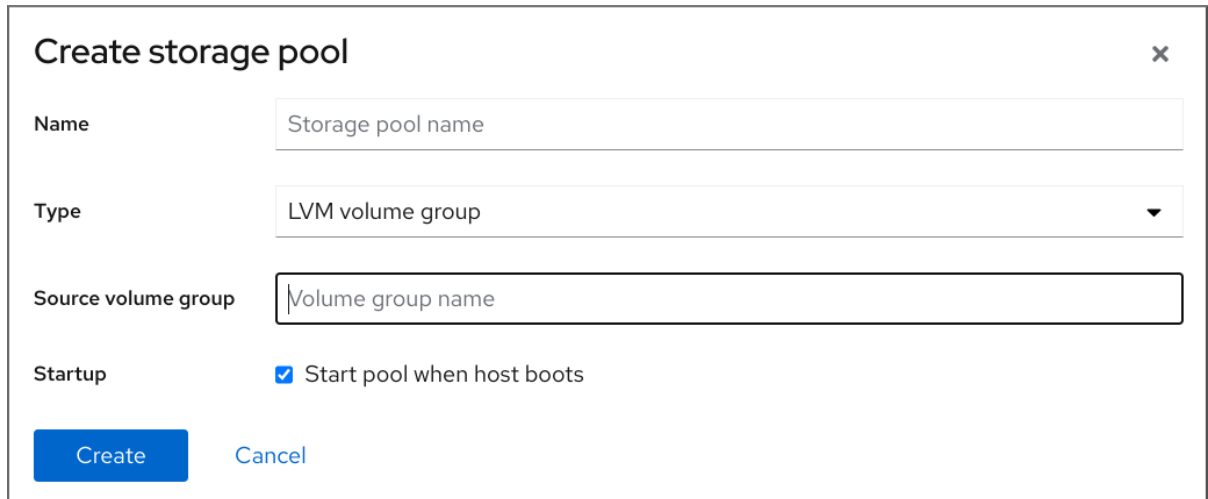
스토리지 풀 생성 대화 상자가 나타납니다.

3.

스토리지 풀의 이름을 입력합니다.

4.

유형 드롭다운 메뉴에서 **LVM** 볼륨 그룹 을 선택합니다.



The image shows a 'Create storage pool' dialog box with the following fields and options:

- Name:** A text input field containing 'Storage pool name'.
- Type:** A dropdown menu showing 'LVM volume group'.
- Source volume group:** A text input field containing 'Volume group name'.
- Startup:** A checkbox labeled 'Start pool when host boots' which is checked.
- Buttons:** 'Create' (blue) and 'Cancel' (light blue).



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.
- **Startup** - 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](#)

11.3.7. 웹 콘솔을 사용하여 vHBA 장치로 SCSI 기반 스토리지 풀 생성

SCSI 기반 스토리지 풀은 SCSI(Small Computer System Interface) 장치를 기반으로 합니다. 이 구성에서 호스트는 vHBA(가상 호스트 버스 어댑터)를 사용하여 SCSI 장치에 연결할 수 있어야 합니다.

사전 요구 사항

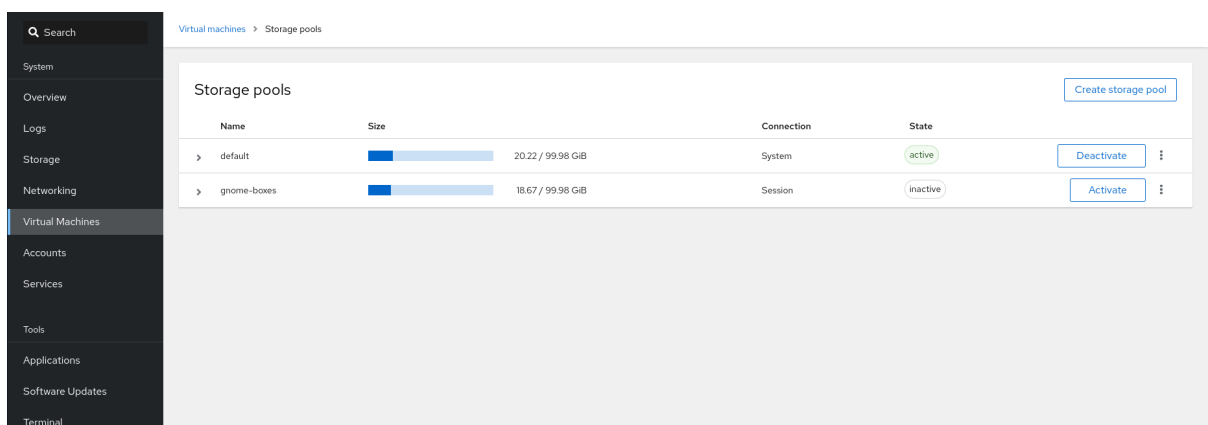
- vHBA를 만듭니다. 자세한 내용은 [vHBA 생성](#) 을 참조하십시오.
- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.

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** 직접 대상 을 선택합니다.

Create storage pool

×

Name

Storage pool name

Type

iSCSI direct target

▼

Host

Host name

Source path

iSCSI target IQN

Initiator

iSCSI initiator IQN

Startup

☒ Start pool when host boots

Create

Cancel



참고

드롭다운 메뉴에 **iSCSI** 직접 대상 옵션이 표시되지 않으면 하이퍼바이저에서 **SCSI** 기반 스토리지 풀을 지원하지 않습니다.

5. 정보의 나머지 부분을 입력하십시오:
 - **host** - 마운트 지점이 있는 네트워크 서버의 호스트 이름입니다. 호스트 이름 또는 IP 주소일 수 있습니다.
 - **소스 경로** - **iSCSI** 대상의 고유한 **IQN(iSCSI Qualified Name)**입니다.
 - **이니시에이터** - **iSCSI** 이니시에이터의 고유한 **IQN(iSCSI Qualified Name, vHBA)**입니다.
 - **Startup** - 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](#)

11.3.8. 웹 콘솔을 사용하여 스토리지 풀 제거

스토리지 풀을 제거하여 호스트 또는 네트워크의 리소스를 확보하여 시스템 성능을 향상시킬 수 있습니다. 스토리지 풀을 삭제하면 다른 가상 시스템(VM)에서 사용할 수 있는 리소스도 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 비활성화](#)를 참조하십시오.

사전 요구 사항

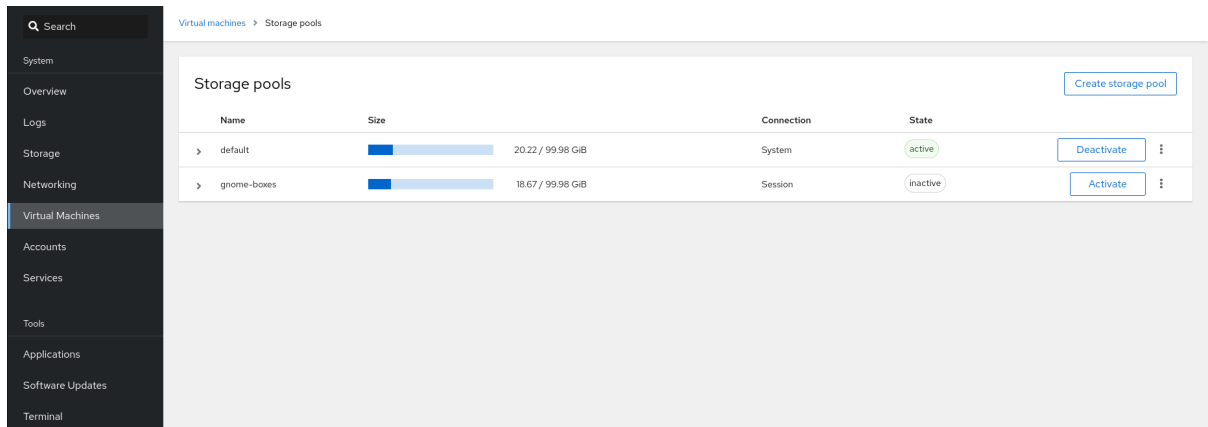
- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.
- 풀과 함께 스토리지 볼륨을 삭제하려면 먼저 VM에서 [디스크를 분리](#)해야 합니다.

절차

1.

가상 머신 탭에서 **Storage Pools** 를 클릭합니다.

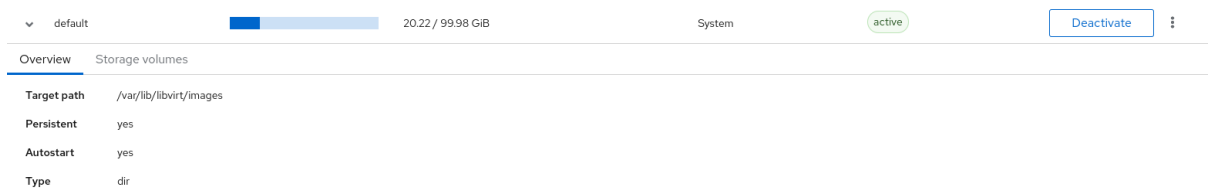
구성된 스토리지 풀 목록을 보여주는 스토리지 풀 창이 표시됩니다.



2.

Storage Pools (스토리지 풀) 창에서 삭제할 스토리지 풀을 클릭합니다.

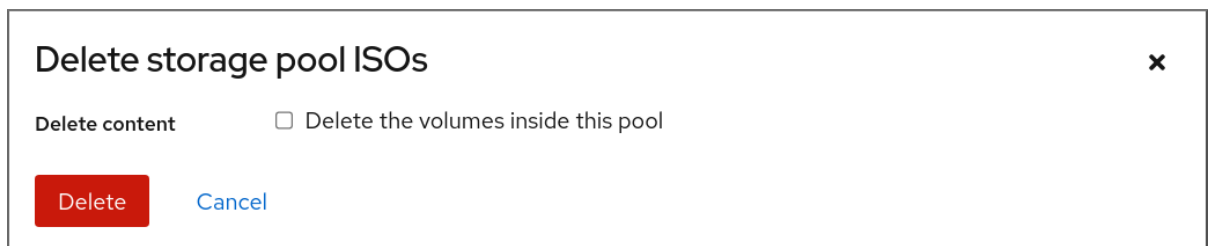
행이 확장되어 선택한 스토리지 풀에 대한 기본 정보와 스토리지 풀을 비활성화하거나 삭제하기 위한 제어에 대한 기본 정보가 포함된 개요 창이 표시됩니다.



3.

메뉴 버튼을 클릭하고 삭제를 클릭합니다.

확인 대화 상자가 나타납니다.



4.

선택 사항: 풀 내부의 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](#)

11.3.9. 웹 콘솔을 사용하여 스토리지 풀 비활성화

스토리지 풀을 영구적으로 삭제하지 않으려면 대신 일시적으로 비활성화할 수 있습니다.

스토리지 풀을 비활성화하면 해당 풀에 새 볼륨을 생성할 수 없습니다. 그러나 해당 풀에 볼륨이 있는 VM(가상 머신)은 계속 실행됩니다. 예를 들어, 시스템 성능을 높이기 위해 풀에 생성할 수 있는 볼륨 수를 제한할 수 있는 여러 가지 이유에 유용합니다.

RHEL 웹 콘솔을 사용하여 스토리지 풀을 비활성화하려면 다음 절차를 참조하십시오.

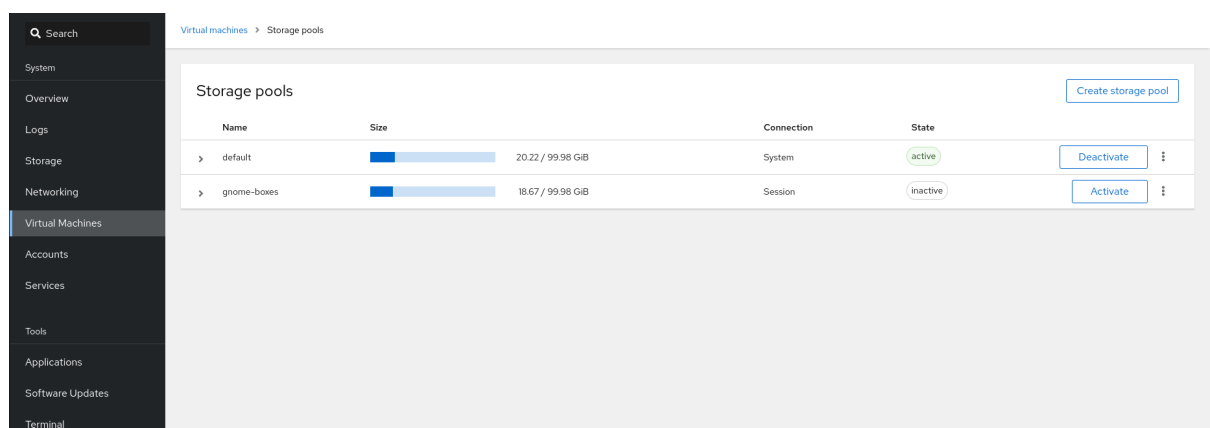
사전 요구 사항

- 웹 콘솔 VM 플러그인이 [시스템에 설치되어](#) 있습니다.

절차

1.

Virtual Machines(가상 머신) 탭의 상단에서 **Storage Pools** (스토리지 풀)를 클릭합니다. 구성된 스토리지 풀 목록을 표시하는 **Storage Pools**(스토리지 풀) 창이 표시됩니다.



2.

Storage Pools (스토리지 풀) 창에서 비활성화할 스토리지 풀을 클릭합니다.

행이 확장되어 선택한 스토리지 풀에 대한 기본 정보와 **VM**의 비활성화 및 삭제를 위한 제어 기능이 포함된 **Overview(개요)** 창이 표시됩니다.

▼ default		20.22 / 99.98 GiB	System	active	Deactivate ⋮
Overview Storage volumes					
Target path	/var/lib/libvirt/images				
Persistent	yes				
Autostart	yes				
Type	dir				

3.

비활성화 를 클릭합니다.

스토리지 풀이 비활성화되었습니다.

추가 리소스

•

[스토리지 풀 이해](#)

•

[웹 콘솔을 사용하여 스토리지 풀 정보 보기](#)

11.4. 스토리지 풀 생성을 위한 매개변수

필요한 스토리지 풀 유형에 따라 **XML** 구성 파일을 수정하고 특정 유형의 스토리지 풀을 정의할 수 있습니다. 이 섹션에서는 예제와 함께 다양한 스토리지 풀 유형을 생성하는 데 필요한 **XML** 매개 변수에 대한 정보를 제공합니다.

11.4.1. 디렉터리 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 디렉터리 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개 변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 **XML** 구성에 따라 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 파일의 필수 매개 변수 목록을 제공합니다.

표 11.1. 디렉터리 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀 유형	<code><pool type='dir'></code>
스토리지 풀의 이름	<code><name>name</name></code>
대상을 지정하는 경로입니다. 이 경로는 스토리지 풀에 사용되는 경로입니다.	<code><target></code> <code> <path>target_path</path></code> <code></target></code>

예제

다음은 `/guest_images` 디렉터를 기반으로 하는 스토리지 풀에 대한 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](#)

11.4.2. 디스크 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 디스크 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개 변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

`virsh pool-define` 명령을 사용하여 지정된 파일의 XML 구성에 따라 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 파일의 필수 매개 변수 목록을 제공합니다.

표 11.2. 디스크 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀 유형	<code><pool type='disk'></code>
스토리지 풀의 이름	<code><name>name</name></code>
스토리지 장치를 지정하는 경로입니다. 예를 들면 <code>/dev/sdb</code> 입니다.	<code><source></code> <code> <path>source_path</path></code> <code></source></code>
대상 장치를 지정하는 경로입니다. 이 경로는 스토리지 풀에 사용되는 경로입니다.	<code><target></code> <code> <path>target_path</path></code> <code></target></code>

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](#)

11.4.3. 파일 시스템 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 파일 시스템 기반 스토리지 풀을 생성하거나 수정하려면 필요한 특정 매개 변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 XML 구성에 따라 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 파일의 필수 매개 변수 목록을 제공합니다.

표 11.3. 파일 시스템 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀 유형	<code><pool type='fs'></code>
스토리지 풀의 이름	<code><name>name</name></code>
파티션을 지정하는 경로입니다. 예: <code>/dev/sdc1</code>	<code><source></code> <code><device path=device_path /></code>
파일 시스템 유형(예: <code>ext4</code>).	<code><format type=fs_type /></code> <code></source></code>
대상을 지정하는 경로입니다. 이 경로는 스토리지 풀에 사용되는 경로입니다.	<code><target></code> <code><path>path-to-pool</path></code> <code></target></code>

예제

다음은 `/dev/sdc1` 파티션을 기반으로 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](#)

11.4.4. GlusterFS 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 **GlusterFS** 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개 변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 **XML** 구성에 따라 스토리지 풀을 생성할 수 있습니다. 예를 들면 다음과 같습니다.

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

매개 변수

다음 표에서는 **GlusterFS** 기반 스토리지 풀에 대한 **XML** 파일의 필수 매개 변수 목록을 제공합니다.

표 11.4. GlusterFS 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀 유형	<code><pool type='gluster'></code>
스토리지 풀의 이름	<code><name>name</name></code>
Gluster 서버의 호스트 이름 또는 IP 주소	<code><source></code> <code><name=gluster-name /></code>
스토리지 풀에 사용되는 Gluster 서버의 경로입니다.	<code><dir path=gluster-path /></code> <code></source></code>

예제

다음은 111.222.111.222의 **Gluster** 파일 시스템을 기반으로 하는 스토리지 풀에 대한 **XML** 파일의 예입니다.

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

추가 리소스

CLI를 사용하여 GlusterFS 기반 스토리지 풀 생성

11.4.5. iSCSI 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 **iSCSI** 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개 변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 **XML** 구성에 따라 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** 파일의 필수 매개 변수 목록을 제공합니다.

표 11.5. iSCSI 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀 유형	<code><pool type='iscsi'></code>
스토리지 풀의 이름	<code><name>name</name></code>
호스트 이름	<code><source></code> <code><host name=hostname /></code>
iSCSI IQN	<code><device path= iSCSI_IQN /></code> <code></source></code>
대상을 지정하는 경로입니다. 이 경로는 스토리지 풀에 사용되는 경로입니다.	<code><target></code> <code><path>/dev/disk/by-path</path></code> <code></target></code>
[선택 사항] iSCSI 이니시에이터의 IQN입니다. 이는 ACL이 LUN을 특정 이니시에이터로 제한하는 경우에 필요합니다.	<code><initiator></code> <code><iqn name='initiator0' /></code> <code></initiator></code>



참고

iSCSI 이니시에이터의 IQN은 `virsh find-storage-pool-sources-as iscsi` 명령을 사용하여 결정할 수 있습니다.

예제

다음은 지정된 iSCSI 장치를 기반으로 하는 스토리지 풀에 대한 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](#)

11.4.6. LVM 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 LVM 기반 스토리지 풀을 생성하거나 수정하려면 필요한 특정 매개 변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

`virsh pool-define` 명령을 사용하여 지정된 파일의 XML 구성에 따라 스토리지 풀을 생성할 수 있습니다. 예를 들면 다음과 같습니다.

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

매개 변수

다음 표에는 LVM 기반 스토리지 풀의 XML 파일의 필수 매개 변수 목록이 나와 있습니다.

표 11.6. LVM 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀 유형	<code><pool type=' Logical'></code>
스토리지 풀의 이름	<code><name> name</name></code>
스토리지 풀의 장치 경로	<code><source> <device path=' device_path' /></code>
볼륨 그룹의 이름	<code><name> VG-name</name></code>
가상 그룹 형식	<code><format type='lvm2' /> </source></code>
대상 경로	<code><target> <path= target_path /> </target></code>

참고

논리 볼륨 그룹이 여러 디스크 파티션으로 구성된 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 LVM을 기반으로 하는 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

11.4.7. NFS 기반 스토리지 풀 매개변수

XML 구성 파일을 사용하여 **NFS** 기반 스토리지 풀을 생성하거나 수정하려면 특정 필수 매개 변수를 포함해야 합니다. 이러한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 **XML** 구성에 따라 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** 파일의 필수 매개 변수 목록을 제공합니다.

표 11.7. NFS 기반 스토리지 풀 매개변수

설명	XML
스토리지 풀 유형	<code><pool type='netfs'></code>
스토리지 풀의 이름	<code><name>name</name></code>
마운트 지점이 있는 네트워크 서버의 호스트 이름입니다. 호스트 이름 또는 IP 주소일 수 있습니다.	<code><source></code> <code><host name=hostname /></code>
스토리지 풀의 형식	다음 중 하나: <code><format type='nfs' /></code> <code><format type='glusterfs' /></code> <code><format type='cifs' /></code>
네트워크 서버에서 사용되는 디렉토리	<code><dir path=source_path/></code> <code></source></code>
대상을 지정하는 경로입니다. 이 경로는 스토리지 풀에 사용되는 경로입니다.	<code><target></code> <code><path>target_path</path></code> <code></target></code>

예제

다음은 **file_server** NFS 서버의 **/home/net_mount** 디렉터리에 기반한 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](#)

11.4.8. vHBA 장치가 있는 SCSI 기반 스토리지 풀의 매개변수

vHBA(가상 호스트 어댑터 버스) 장치를 사용하는 **SCSi** 기반 스토리지 풀의 **XML** 구성 파일을 생성하거나 수정하려면 **XML** 구성 파일에 필요한 특정 매개 변수를 포함해야 합니다. 필요한 매개변수에 대한 자세한 내용은 다음 표를 참조하십시오.

virsh pool-define 명령을 사용하여 지정된 파일의 **XML** 구성에 따라 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 **vHBA**가 있는 **SCSi** 기반 스토리지 풀의 **XML** 파일의 필수 매개변수 목록을 제공합니다.

표 11.8. vHBA 장치가 있는 SCSI 기반 스토리지 풀의 매개변수

설명	XML
스토리지 풀 유형	<pool type='scsi'>
스토리지 풀의 이름	<name>name</name>

설명	XML
vHBA의 식별자입니다. 상위 특성은 선택 사항입니다.	<pre><source> <adapter type='fc_host' [parent=parent_scsi_device wwnn='WWNN' wwpn='WWPN' /> </source></pre>
대상 경로입니다. 이 경로는 스토리지 풀에 사용되는 경로입니다.	<pre><target> <path=target_path /> </target></pre>

중요

<path> 필드가 **/dev/** 이면 **libvirt** 에서 볼륨 장치 경로에 대한 고유한 짧은 장치 경로를 생성합니다. 예를 들면 **/dev/sdc** 입니다. 그렇지 않으면 물리적 호스트 경로가 사용됩니다. 예를 들면 **/dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0** 입니다. 고유한 짧은 장치 경로를 사용하면 여러 스토리지 풀에서 동일한 볼륨을 여러 **VM**(가상 머신)에 나열할 수 있습니다. 여러 **VM**에서 물리적 호스트 경로를 사용하는 경우 중복 장치 유형 경고가 발생할 수 있습니다.

참고

상위 속성을 **<adapter>** 필드에서 사용하여 다양한 경로로 **NPIV LUN**을 사용할 수 있는 물리적 **HBA** 상위 항목을 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 재부팅된 후 동일한 **HBA**가 사용된다는 보장 정도를 제공합니다.

- 상위 항목이 지정되지 않은 경우 **libvirt** 는 **NPIV**를 지원하는 첫 번째 **scsi_hostN** 어댑터를 사용합니다.
- 상위 항목만 지정하면 추가 **SCSI** 호스트 어댑터가 구성에 추가되는 경우 문제가 발생할 수 있습니다.
- **parent_wwnn** 또는 **parent_wwpn** 이 지정된 경우 호스트가 재부팅된 후 동일한 **HBA**가 사용됩니다.
- **parent_fabric_wwn** 을 사용하는 경우, 사용된 **scsi_hostN** 과 관계없이 호스트가 재부팅된 후 동일한 페브릭에서 **HBA**를 선택합니다.

예

다음은 **vHBA**를 사용하여 **SCSI** 기반 스토리지 풀을 위한 **XML** 파일의 예입니다.

- **HBA**의 유일한 스토리지 풀인 스토리지 풀은 다음과 같습니다.

```
<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 상위 속성을 사용하여 **SCSI** 호스트 장치를 식별하는 여러 스토리지 풀 중 하나인 스토리지 풀입니다.

```
<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

- [CLI를 사용하여 vHBA 장치로 SCSI 기반 스토리지 풀 생성](#)

11.5. 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](#)

11.5.1. CLI를 사용하여 스토리지 볼륨 정보 보기

명령줄을 사용하여 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
```

11.5.2. CLI를 사용하여 스토리지 볼륨 생성 및 할당

디스크 이미지를 가져와 가상 디스크로 VM(가상 머신)에 연결하려면 스토리지 볼륨을 생성하고 해당 XML 구성을 VM에 할당합니다.

사전 요구 사항

- 호스트에 할당되지 않은 공간이 있는 스토리지 풀이 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 없으면 새로 생성합니다. 자세한 내용은 [가상 머신의 스토리지 관리](#)를 참조하십시오.

절차

1.

virsh vol-create-as 명령을 사용하여 스토리지 볼륨을 만듭니다. 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 생성하기 위해 특정 프로세스가 필요합니다.

- **GlusterFS 기반 - qemu-img** 명령을 사용하여 스토리지 볼륨을 생성합니다.
 - **iSCSI 기반 - iSCSI** 서버에서 미리 **iSCSI LUN**을 준비합니다.
 - **다중 경로 기반 - multipathd** 명령을 사용하여 다중 경로를 준비하거나 관리합니다.
 - **vHBA 기반 - 파이버 채널 카드**를 미리 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 예에서는 이전 단계에서 만든 **vm-disk1** 볼륨을 사용하는 가상 디스크를 지정하고 **an ide** 버스에서 디스크로 설정 할 볼륨을 설정합니다. 해당 매개변수를 환경에 적절하게 수정합니다.

중요: 특정 스토리지 풀 유형의 경우 다른 **XML** 형식을 사용하여 스토리지 볼륨 디스크를 설명해야 합니다.

-

GlusterFS 기반 풀의 경우:

```
<disk type='network' device='disk'>
  <driver name='qemu' type='raw' />
  <source protocol='gluster' name='Volume1/Image'>
    <host name='example.org' port='6000' />
  </source>
  <target dev='vda' bus='virtio' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0' />
</disk>
```

-

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 **testquest1** **VM**에 할당하려면 다음을 수행합니다.

```
# attach-device --config testguest1 ~/vm-disk1.xml
```

검증

- VM의 게스트 운영 체제에서 디스크 이미지를 포맷되지 않은 디스크로 사용할 수 있는지 확인합니다.

11.5.3. CLI를 사용하여 스토리지 볼륨 삭제

호스트 시스템에서 스토리지 볼륨을 제거하려면 풀을 중지하고 XML 정의를 제거해야 합니다.

사전 요구 사항

- 삭제하려는 스토리지 볼륨을 사용하는 가상 머신은 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 초기화하려면 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
```

11.6. 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.](#)

11.6.1. 웹 콘솔을 사용하여 스토리지 볼륨 생성

작동 중인 **VM**(가상 머신)을 생성하려면 **VM** 이미지 및 **VM** 관련 데이터를 저장할 수 있는 **VM**에 할당된 로컬 스토리지 장치가 필요합니다. 스토리지 풀에 스토리지 볼륨을 생성하고 **VM**에 스토리지 디스크로 할당할 수 있습니다.

웹 콘솔을 사용하여 스토리지 볼륨을 생성하려면 다음 절차를 참조하십시오.

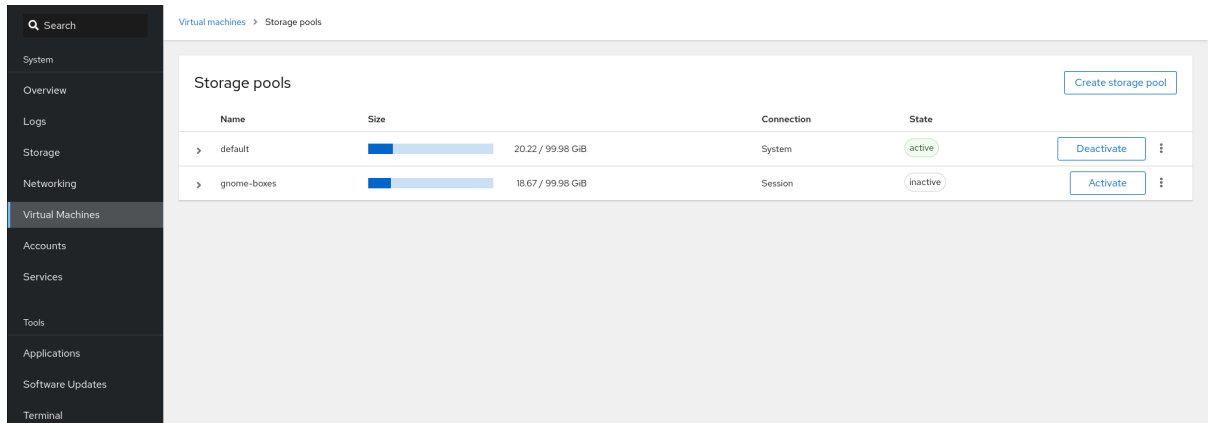
사전 요구 사항

웹 콘솔 VM 플러그인이 **시스템에 설치되어** 있습니다.

절차

1.

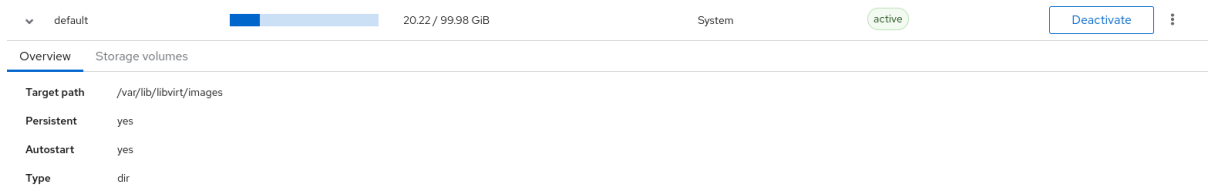
Virtual Machines(가상 머신) 탭의 상단에서 **Storage Pools** (스토리지 풀)를 클릭합니다. 구성된 스토리지 풀 목록을 표시하는 **Storage Pools**(스토리지 풀) 창이 표시됩니다.



2.

Storage Pools (스토리지 풀) 창에서 스토리지 볼륨을 생성할 스토리지 풀을 클릭합니다.

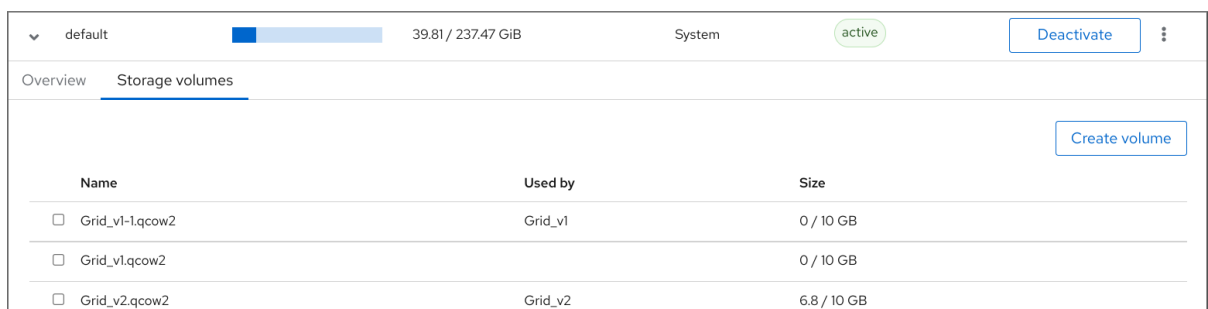
행이 확장되어 선택한 스토리지 풀에 대한 기본 정보가 포함된 **Overview**(개요) 창이 표시됩니다.



3.

확장된 행의 **Overview**(개요) 탭 옆에 있는 **Storage Volumes** (스토리지 볼륨)를 클릭합니다.

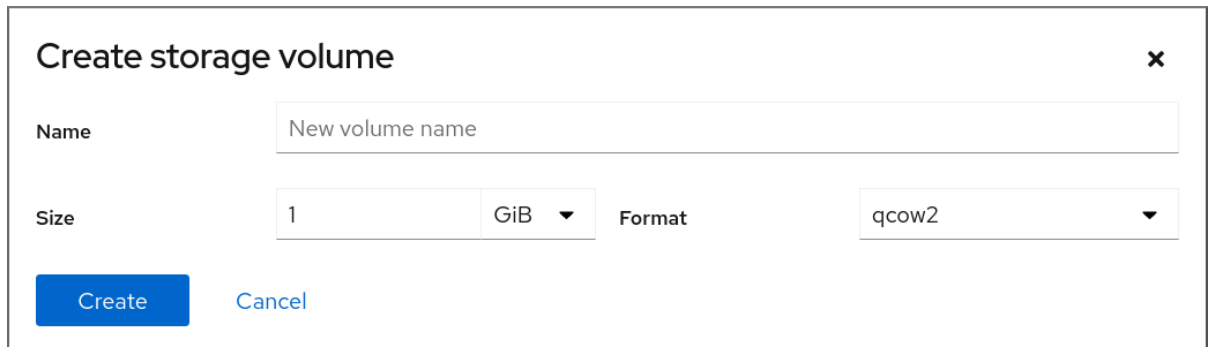
Storage Volume(스토리지 볼륨) 탭에 기존 스토리지 볼륨에 대한 기본 정보가 표시됩니다 (있는 경우).



4.

Create Volume (볼륨 만들기)을 클릭합니다.

Create Storage Volume(스토리지 볼륨 생성) 대화 상자가 표시됩니다.



The image shows a 'Create storage volume' dialog box with a close button (X) in the top right corner. It contains the following fields and controls:

- Name:** A text input field with the placeholder text 'New volume name'.
- Size:** A numeric input field containing the value '1'.
- Unit:** A dropdown menu currently set to 'GiB'.
- Format:** A dropdown menu currently set to 'qcow2'.
- Buttons:** A blue 'Create' button and a blue 'Cancel' button.

5.

Create Storage Volume(스토리지 볼륨 생성) 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 닫힙니다. 새 스토리지 볼륨이 스토리지 볼륨 목록에 표시됩니다.

추가 리소스

- [스토리지 볼륨 이해](#)
- [웹 콘솔을 사용하여 가상 머신에 새 디스크 추가](#)

11.6.2. 웹 콘솔을 사용하여 스토리지 볼륨 제거

스토리지 볼륨을 제거하여 스토리지 풀의 공간을 확보하거나 존재하지 않는 **VM**(가상 머신)과 관련된

스토리지 항목을 제거할 수 있습니다.

RHEL 웹 콘솔을 사용하여 스토리지 볼륨을 제거하려면 다음 절차를 참조하십시오.

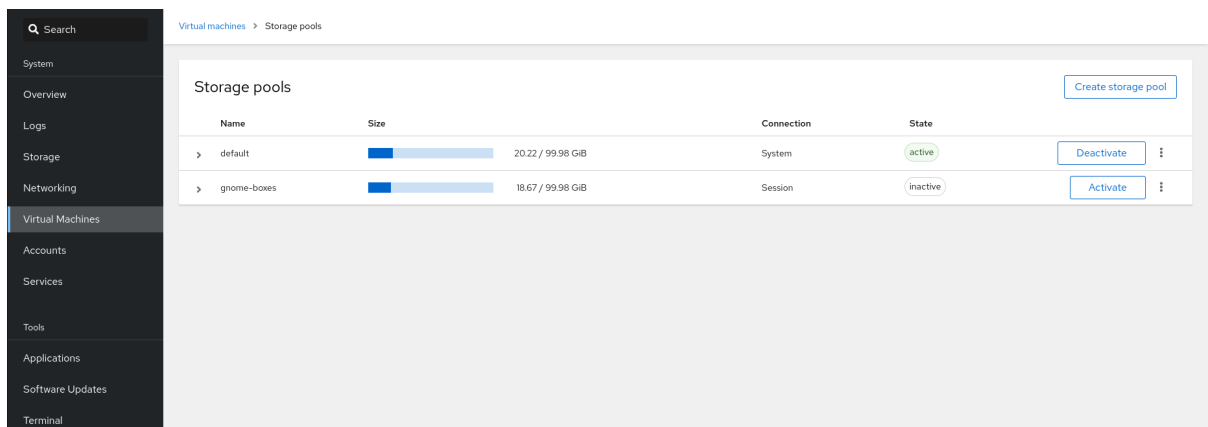
사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 **시스템에 설치되어** 있습니다.
- 삭제하려는 스토리지 볼륨을 사용하는 가상 머신은 종료됩니다.

절차

1.

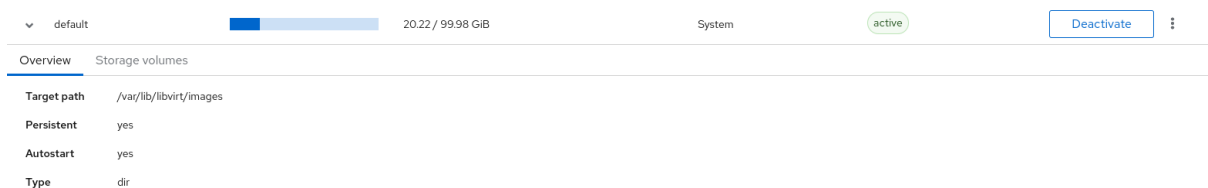
Virtual Machines(가상 머신) 탭의 상단에서 **Storage Pools** (스토리지 풀)를 클릭합니다. 구성된 스토리지 풀 목록을 표시하는 **Storage Pools**(스토리지 풀) 창이 표시됩니다.



2.

Storage Pools (스토리지 풀) 창에서 스토리지 볼륨을 제거할 스토리지 풀을 클릭합니다.

행이 확장되어 선택한 스토리지 풀에 대한 기본 정보가 포함된 **Overview**(개요) 창이 표시됩니다.



3.

확장된 행의 **Overview**(개요) 탭 옆에 있는 **Storage Volumes** (스토리지 볼륨)를 클릭합니다.

Storage Volume(스토리지 볼륨) 탭에 기존 스토리지 볼륨에 대한 기본 정보가 표시됩니다 (있는 경우).

▼ default		39.81 / 237.47 GiB	System	active	Deactivate	⋮
Overview <u>Storage volumes</u>						
Create volume						
Name	Used by	Size				
☐ Grid_v1-l.qcow2	Grid_v1	0 / 10 GB				
☐ Grid_v1.qcow2		0 / 10 GB				
☐ Grid_v2.qcow2	Grid_v2	6.8 / 10 GB				

4.

제거할 스토리지 볼륨을 선택합니다.

▼ default		39.81 / 237.47 GiB	System	active	Deactivate	⋮
Overview <u>Storage volumes</u>						
Delete 1 volume Create volume						
Name	Used by	Size				
☐ Grid_v1-l.qcow2	Grid_v1	0 / 10 GB				
☐ Grid_v1.qcow2		0 / 10 GB				
☐ Grid_v2.qcow2	Grid_v2	6.8 / 10 GB				
☒ v2		0 / 15 GB				

5.

Delete 1 Volume을 클릭합니다.

추가 리소스

- [스토리지 볼륨 이해](#)

11.7. 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.

11.7.1. 웹 콘솔에서 가상 머신 디스크 정보 보기

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

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. 보려는 정보를 보여주는 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 스크롤.

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

Disks							Add disk
Device	Used	Capacity	Bus	Access	Source		
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove Edit
					Volume	v2	

이 정보에는 다음이 포함됩니다.

- **Device** - 디스크의 장치 유형입니다.
- **used** - 현재 할당된 디스크 양입니다.
- **capacity** - 스토리지 볼륨의 최대 크기입니다.
- **bus** - 에뮬레이트된 디스크 장치의 유형입니다.
- **access** - 디스크에 쓰기 가능 또는 읽기 전용 인지 여부. 원시 디스크의 경우 액세스를 쓰기 가능 및 공유로 설정할 수도 있습니다.
- **Source** - 디스크 장치 또는 파일입니다.

추가 리소스

- [웹 콘솔을 사용하여 가상 머신 정보 보기](#)

11.7.2. 웹 콘솔을 사용하여 가상 머신에 새 디스크 추가

새 스토리지 볼륨을 생성하고 **RHEL 8** 웹 콘솔을 사용하여 **VM**(가상 머신)에 새 디스크를 추가할 수 있습니다.

사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 [시스템에 설치되어](#) 있습니다.

절차

1. **Virtual Machines** (가상 시스템) 인터페이스에서 새 디스크를 생성하고 연결할 **VM**을 클릭합니다.

VM의 그래픽 인터페이스에 액세스하기 위한 선택한 **VM** 및 콘솔 섹션에 대한 기본 정보가 포

함된 **Overview**(개요) 섹션이 포함된 새 페이지가 열립니다.

2.

디스크로 스크롤.

Disks(디스크) 섹션에는 **VM**에 할당된 디스크 정보와 디스크 추가,제거 또는 편집 옵션이 표시됩니다.

Disks							Add disk
Device	Used	Capacity	Bus	Access	Source		
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove Edit
				Volume	v2		

3.

디스크 추가를 클릭합니다.

디스크 추가 대화 상자가 나타납니다.

Add disk

Source

☒ Create new
 ☐ Use existing
 ☐ Custom path

Pool

default

Name

New volume name

Size

1

GiB

Format

qcow2

Persistence

☐ Always attach

Show additional options

Add

Cancel

4.

Create New (새로 만들기) 옵션을 선택합니다.

5.

새 디스크를 구성합니다.

- **Pool Pool (풀)** - 가상 디스크를 생성할 스토리지 풀을 선택합니다.
- **name** - 생성할 가상 디스크의 이름을 입력합니다.
- **size** - 크기를 입력하고 생성할 가상 디스크의 단위(**MiB** 또는 **GiB**)를 선택합니다.
- **Format (형식)** - 생성될 가상 디스크 형식을 선택합니다. 지원되는 유형은 **qcow2** 및 **raw** 입니다.
- **Persistent(영구)** - 선택하면 가상 디스크가 지속됩니다. 확인되지 않으면 가상 디스크가 일시적입니다.



참고

임시 디스크는 실행 중인 VM에만 추가할 수 있습니다.

- 추가 옵션 - 가상 디스크에 대한 추가 구성 설정.
 - **캐시** - 캐시 메커니즘을 선택합니다.
 - **bus** - 에뮬레이션할 디스크 장치 유형을 선택합니다.

6. **추가**를 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

11.7.3. 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 (가상 시스템) 인터페이스에서 새 디스크를 생성하고 연결할 **VM**을 클릭합니다.

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

2.

디스크로 스크롤.

Disks(디스크) 섹션에는 **VM**에 할당된 디스크 정보와 디스크 추가, 제거 또는 편집 옵션이 표시됩니다.

Disks							Add disk
Device	Used	Capacity	Bus	Access	Source		
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove Edit
				Volume	v2		

3.

디스크 추가를 클릭합니다.

디스크 추가 대화 상자가 나타납니다.

×

Add disk

Source

☒ Create new
 ☐ Use existing
 ☐ Custom path

Pool

default ▼

Name

New volume name

Size

1

GiB ▼

Format

qcow2 ▼

Persistence

☐ Always attach

▶ Show additional options

Add

Cancel

4.

Use Existing (기존 사용) 라디오 버튼을 클릭합니다.

적절한 구성 필드가 디스크 추가 대화 상자에 표시됩니다.

×

Add disk

Source

☐ Create new
 ☒ Use existing
 ☐ Custom path

Pool

default ▼

Volume

Grid_v1.qcow2 ▼

Persistence

☐ Always attach

▼ Hide additional options

Cache

default ▼

Bus

virtio ▼

Add

Cancel

5.

VM의 디스크를 구성합니다.

- **Pool (풀)** - 가상 디스크를 연결할 스토리지 풀을 선택합니다.
- **블룸** - 연결할 스토리지 블룸을 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.
 - 캐시 - 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](#)

11.7.4. 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 시스템) 인터페이스에서 디스크를 분리할 **VM**을 클릭합니다.

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

2.

디스크로 스크롤.

Disks(디스크) 섹션에는 **VM**에 할당된 디스크 정보와 디스크 추가, 제거 또는 편집 옵션이 표시됩니다.

Disks							Add disk	
Device	Used	Capacity	Bus	Access	Source			
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove	Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove	Edit
				Volume	v2			

3.

VM에서 분리할 디스크 옆에 있는 **Remove** (제거) 버튼을 클릭합니다. 디스크 제거 확인 대화 상자가 나타납니다.

4.

확인 대화 상자에서 제거를 클릭합니다.

가상 디스크는 **VM**에서 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](#)

11.8. LIBVIRT 보안을 사용하여 ISCSI 스토리지 풀 보안

사용자 이름 및 암호 매개 변수는 iSCSI 스토리지 풀을 보호하기 위해 **virsh** 를 사용하여 구성할 수 있습니다. 풀을 정의하기 전이나 후에 구성할 수 있지만 인증 설정을 적용하려면 풀을 시작해야 합니다.

다음은 **libvirt** 보안을 사용하여 **iSCSI** 기반 스토리지 풀을 보호하는 방법을 제공합니다.



참고

이 절차는 **iSCSI** 대상을 생성할 때 **user_ID** 및 **암호**가 정의된 경우 필요합니다.

사전 요구 사항

- **iSCSI** 기반 스토리지 풀을 생성했는지 확인합니다. 자세한 내용은 [CLI를 사용하여 iSCSI 기반 스토리지 풀 생성](#)을 참조하십시오.

절차

1. **CHAP**(챌린지 핸드셰이크 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** 보안을 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 단계의 출력에서 **UUID**에 시크릿 을 할당합니다. 이렇게 하면 **CHAP** 사용자 이름 및 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 **<auth>** 요소를 추가하여 인증 유형, 사용자 이름 및 시크릿 사용을 지정합니다.

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 다양한 위치에 있습니다. **<pool>**의 경우 **<source>** 요소 내에 **<auth>**가 지정됩니다. 인증은 일부 풀 소스(**iSCSI** 및 **RBD**)의 속성이므로 풀 소스를 찾을 위치를 설명합니다. 도메인의 하위 구현인 **<disk>**의 경우 **iSCSI** 또는 **RBD** 디스크에 대한 인증은 디스크의 속성입니다. 또한 디스크의 **<auth>** 하위 요소는 스토리지 풀의 하위 요소와 다릅니다.

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

6.

변경 사항을 활성화하려면 스토리지 풀을 활성화합니다. 풀이 이미 시작되면 스토리지 풀을 중지하고 다시 시작합니다.

```
# virsh pool-destroy iscsirhel7pool
# virsh pool-start iscsirhel7pool
```

11.9. VHBA 생성

vHBA(가상 호스트 버스 어댑터) 장치는 호스트 시스템을 **SCSI** 장치에 연결하고 **SCSI** 기반 스토리지 풀을 생성하는 데 필요합니다.

XML 구성 파일에서 **vHBA** 장치를 정의하여 생성할 수 있습니다.

절차

1.

virsh nodedev-list --cap vports 명령을 사용하여 호스트 시스템에서 **HBA**를 찾습니다.

다음 예제에서는 **vHBA**를 지원하는 **HBA**가 두 개 있는 호스트를 보여줍니다.

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

2.

virsh nodedev-dumpxml HBA _device 명령을 사용하여 **HBA**의 세부 정보를 확인합니다.

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

명령의 출력에는 **vHBA**를 생성하는 데 사용되는 **<name>**, **<wwnn>** 및 **<wwpn>** 필드가 나열됩니다. **<max_vports>**는 지원되는 최대 **vHBA** 수를 표시합니다. 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** 구성에 사용할 수 있는 총 127개의 가상 포트가 있음을 보여줍니다. **<vports>** 값은 현재 사용 중인 가상 포트 수를 보여줍니다. 이러한 값은 **vHBA**를 생

성한 후 업데이트됩니다.

3.

vHBA 호스트에 대해 다음 중 하나와 유사한 **XML** 파일을 만듭니다. 이 예에서 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 예에서는 **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 사항의 값과 일치해야 합니다.

<parent> 필드는 이 **vHBA** 장치와 연결할 **HBA** 장치를 지정합니다. 다음 단계에서 **<device>** 태그의 세부 정보를 사용하여 호스트에 대한 새 **vHBA** 장치를 만듭니다. **nodedev XML** 형식에 대한 자세한 내용은 [libvirt 업스트림 페이지](#)를 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 **VHBA**를 생성합니다.

```
# virsh nodedev-create vhba_host3
Node device scsi_host5 created from vhba_host3.xml
```

검증

- **virsh nodedev-dumpxml** 명령을 사용하여 새 **vHBA**의 세부 정보(**scsi_host5**)를 확인합니다.

```
# virsh nodedev-dumpxml scsi_host5
<device>
  <name>scsi_host5</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3/vport-3:0-0/host5</path>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <host>5</host>
    <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

- **CLI**를 사용하여 **vHBA** 장치로 **SCSI** 기반 스토리지 풀 생성

12장. 가상 머신의 GPU 장치 관리

RHEL 8 호스트에서 **VM**(가상 머신)의 그래픽 성능을 향상시키기 위해 **VM**에 호스트 **GPU**를 **VM**에 할당할 수 있습니다.

- 호스트에서 **GPU**를 분리하고 **GPU**의 전체 제어를 **VM**에 직접 전달할 수 있습니다.
- 물리적 **GPU**에서 여러 개의 중재된 장치를 생성하고 이러한 장치를 가상 **GPU(vGPU)**로 여러 게스트에 할당할 수 있습니다. 현재 이 기능은 선택한 **NVIDIA GPU**에서만 지원되며, 단일 게스트에 중재된 장치 하나만 할당할 수 있습니다.

12.1. 가상 머신에 GPU 할당

호스트 시스템에 연결된 **GPU**에 액세스하고 제어하려면 **GPU**의 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-d**를 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 있으며 `add iommu=pt` 를 추가하여 통과 모드로 전환할 수 있습니다.

1.

the `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 호스트 성능을 향상시킵니다. 그러나 일부 하드웨어가 옵션을 지원하지 않는 것은 아닙니다. 이 옵션이 활성화되어 있는지에 관계없이 장치를 계속 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 못하게 합니다. 이 작업을 수행하려면 `pci-stub` 드라이버와 함께 GPU의 PCI ID를 사용합니다.

예를 들어 다음 명령을 사용하면 `10de:11fa` 버스에 연결된 GPU에 드라이버가 바인딩되지 않습니다.

```
# grubby --args="pci-stub.ids=10de:11fa" --update-kernel DEFAULT
```

c.

호스트를 재부팅합니다.

2.

선택 사항: 제한 사항을 지원하여 오디오와 같은 특정 **GPU** 기능을 **VM**으로 전달할 수 없는 경우 **IOMMU** 그룹 내에서 끝점의 드라이버 바인딩을 수정하여 필요한 **GPU** 기능만 통과할 수 있습니다.

a.

GPU 설정을 **XML**로 변환하고 호스트 드라이버에 연결하지 못하려는 끝점의 **PCI** 주소를 기록해 둡니다.

이렇게 하려면 **pci_** 접두사를 주소에 추가하고 구분자를 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 연결되지 않도록 합니다.

이 예에서 VM에 GPU를 할당하려면 `<address domain='0x0000' bus='0x02' slot='0x00' function='0x00' function='0x1'/>` 오디오 기능에 해당하는 엔드포인트를 방지하고 대신 VFIO-PCI에 엔드포인트를 연결합니다.

```
# driverctl set-override 0000:02:00.1 vfio-pci
```

3.

GPU를 VM에 연결합니다

a.

PCI 버스 주소를 사용하여 GPU에 대한 XML 구성 파일을 생성합니다.

예를 들어 GPU 버스 주소에서 매개 변수를 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.

파일을 호스트 시스템에 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, Red Hat은 VM의 XML 구성에서 기본 에뮬레이트된 그래픽 장치를 제거하는 것을 권장하지 않습니다.

검증

•

장치가 VM의 XML 구성의 `< devices >` 섹션에 나타납니다. 자세한 내용은 [샘플 가상 머신 XML 구성에서](#) 참조하십시오.

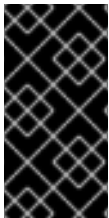
알려진 문제

- VM에 연결할 수 있는 GPU 수는 RHEL 8에서 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 8 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.

12.2. NVIDIA vGPU 장치 관리

vGPU 기능을 사용하면 실제 NVIDIA GPU 장치를 중재 장치라고 하는 여러 가상 장치로 나눌 수 있습니다. 그런 다음 이러한 중재된 장치를 여러 VM(가상 머신)에 가상 GPU로 할당할 수 있습니다. 결과적으로 이러한 VM은 단일 물리적 GPU의 성능을 공유할 수 있습니다.



중요

중재된 장치를 사용하거나 사용하지 않고 VM에 물리적 GPU를 할당하면 호스트에서 GPU를 사용할 수 없습니다.

12.2.1. NVIDIA vGPU 장치 설정

NVIDIA vGPU 기능을 설정하려면 GPU 장치용 NVIDIA vGPU 드라이버를 다운로드하고 중재된 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](#).

 - 호스트에서 사용 중인 GPU를 모르는 경우 *lshw* 패키지를 설치하고 *lshw -C 디스플레이* 명령을 사용합니다. 다음 예제에서는 시스템이 vGPU와 호환되는 NVIDIA Tesla P4 GPU를 사용하고 있는 것을 보여줍니다.

```
# 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 설명서](#)를 참조하십시오.

2.

NVIDIA 소프트웨어 설치 관리자에서 `/etc/modprobe.d/nvidia-installer-disable-nocin.conf` 파일을 생성하지 않은 경우 `/etc/modprobe.d/`에 모든 이름의 `conf` 파일을 만들고 파일에 다음 행을 추가합니다.

```
blacklist nouveau
options nouveau modeset=0
```

3.

현재 커널에 대한 초기 램디스크를 다시 생성한 다음 재부팅합니다.

```
# dracut --force
# reboot
```

4.

커널이 그 다음 `nvidia_vgpu_vfio` 모듈을 로드하고 `the 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 **vGPU**를 생성하는 경우, 물리 **GPU**에 가상 기능을 사용하도록 설정해야 합니다. 자세한 내용은 [NVIDIA 문서](#)를 참조하십시오.

5.

장치 **UUID** 생성.

```
# 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 **UUID**를 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 공유할 VM에 중재된 장치를 연결합니다. 이를 위해 이전에 **cryptoreated** 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 vGPU 중재 장치의 모든 기능을 사용하려면 VM에 **NVIDIA vGPU** 게스트 소프트웨어 라이선스를 설정합니다. 자세한 내용 및 지침은 [NVIDIA 가상 GPU 소프트웨어 라이선스 서버 사용자 가이드](#)를 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 GPU**로 감지하는지 확인합니다. 예를 들어 VM에서 **Linux**를 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

- RHEL 8** 게스트 운영 체제를 사용하는 VM에 **NVIDIA vGPU** 중재 장치를 할당하는 경우 현재 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

12.2.2. NVIDIA vGPU 장치 제거

할당된 vGPU 중재 장치의 구성을 변경하려면 할당된 VM에서 기존 장치를 제거해야 합니다. 자세한 내용은 다음을 참조하십시오.

사전 요구 사항

- 장치를 제거할 VM이 종료됩니다.

절차

1.

제거하려는 중재 장치의 **ID**를 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.

VM의 XML 구성에서 장치를 제거합니다. 이렇게 하려면 **virsh edit** 유틸리티를 사용하여 VM의 XML 구성을 편집하고 **mdev** 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 중지하고 분리하면 삭제되지 않고 정의된 대로 유지합니다. 따라서 장치를 다시 시작하고 다른 VM에 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

12.2.3. 시스템에 대한 NVIDIA vGPU 정보 얻기

사용 가능한 **vGPU** 기능의 기능을 평가하기 위해 시스템에서 중재된 장치에 대한 추가 정보를 얻을 수 있습니다(예:).

- 지정된 유형의 중재된 장치를 만들 수 있는 장치 수는 몇 개입니까?
- 시스템에 이미 구성된 중재된 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

12.2.4. NVIDIA vGPU 용 원격 데스크탑 스트리밍 서비스

다음 원격 데스크탑 스트리밍 서비스는 NVIDIA vGPU 또는 NVIDIA GPU 패스스루가 활성화된 RHEL 8 하이퍼바이저에서 지원됩니다.

- **HP ZCentral Remote Boost/Teradici**
- **NICE DCV**
- **Mechdyne TGX**

지원 세부 사항은 해당 벤더 지원 매트릭스를 참조하십시오.

12.2.5. 추가 리소스

- [NVIDIA vGPU 소프트웨어 문서](#)

13장. 가상 머신 네트워크 연결 구성

VM(가상 시스템)은 네트워크를 통해 호스트에 연결하고 호스트의 다른 **VM**에, 외부 네트워크의 위치로 연결하려면 그에 따라 **VM** 네트워킹을 구성해야 합니다. **VM** 네트워킹을 제공하기 위해 **RHEL 8** 하이퍼바이저와 새로 생성된 **VM**에는 기본 네트워크 구성이 있으며, 이 구성도 추가로 수정할 수 있습니다. 예를 들면 다음과 같습니다.

- **VM**이 호스트와 동일한 네트워크에 있는 것처럼 호스트 외부의 위치를 통해 호스트의 **VM**을 검색하고 연결할 수 있습니다.
- **VM**을 인바운드 네트워크 트래픽에서 부분적으로 또는 완전히 분리하여 보안을 강화하고 호스트에 영향을 미치는 **VM**과 관련된 문제의 위험을 최소화할 수 있습니다.

다음 섹션에서는 다양한 유형의 **VM** 네트워크 구성에 대해 설명하고 선택한 **VM** 네트워크 구성을 설정하는 지침을 제공합니다.

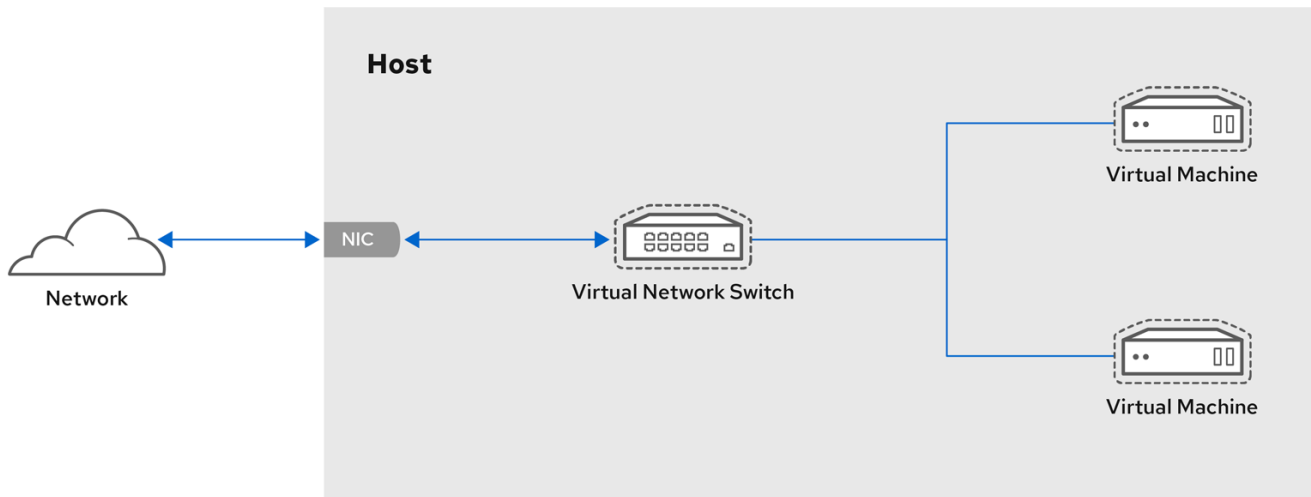
13.1. 가상 네트워킹 이해

VM(가상 시스템)을 호스트 하드웨어에서 다른 장치와 위치에 연결하려면 호스트 하드웨어에서 용이하게 해야 합니다. 다음 섹션에서는 **VM** 네트워크 연결 메커니즘에 대해 설명하고 기본 **VM** 네트워크 설정을 설명합니다.

13.1.1. 가상 네트워크의 작동 방식

가상 네트워킹은 가상 네트워크 스위치 개념을 사용합니다. 가상 네트워크 스위치는 호스트 시스템에서 작동하는 소프트웨어 구성입니다. **VM**은 가상 네트워크 스위치를 통해 네트워크에 연결합니다. 가상 스위치의 구성에 따라 **VM**은 하이퍼바이저 또는 다른 네트워크 연결 방법으로 관리되는 기존 가상 네트워크를 사용할 수 있습니다.

다음 그림은 두 개의 **VM**을 네트워크에 연결하는 가상 네트워크 스위치를 보여줍니다.



RHEL_52_1219

게스트 운영 체제의 관점에서 볼 때 가상 네트워크 연결은 실제 네트워크 연결과 동일합니다. 호스트 시스템은 가상 네트워크 스위치를 네트워크 인터페이스로 봅니다. **libvirtd** 서비스가 처음 설치 및 시작되면 **VM**의 기본 네트워크 인터페이스인 **virbr0** 을 만듭니다.

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** 유형 가상 네트워크에 연결됩니다. 자세한 내용은 [가상 네트워킹 기본 구성](#)을 참조하십시오.

VM에서 기본 아웃바운드 전용 네트워크 액세스의 경우 기본 네트워크가 **libvirt-daemon-config-network** 패키지와 함께 설치되고 **libvirtd** 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. 이러한 네트워크 및 인터페이스는 기본 **NAT** 외에도 다음 모드 중 하나를 사용하도록 구성할 수 있습니다.

- [라우팅 모드](#)
- [브릿지 모드](#)

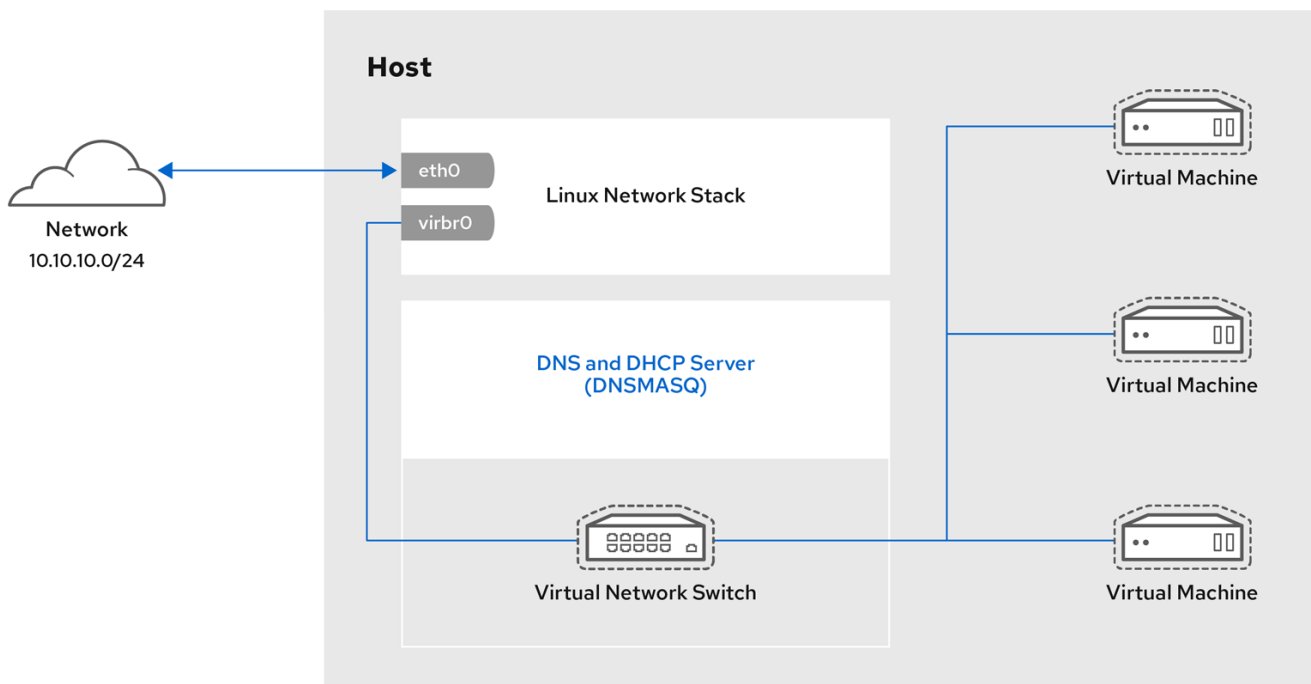
- 격리 모드
- 열기 모드

13.1.2. 가상 네트워킹 기본 구성

libvirt 서비스를 가상화 호스트에 처음 설치하면 **NAT**(네트워크 주소 변환) 모드의 초기 가상 네트워크 구성이 포함됩니다. 기본적으로 호스트의 모든 VM은 **default** 라는 동일한 **libvirt** 가상 네트워크에 연결됩니다. 이 네트워크의 VM은 호스트 이외의 호스트와 네트워크의 위치에 연결할 수 있지만 다음과 같은 제한 사항이 있습니다.

- 네트워크의 VM은 호스트 및 호스트의 다른 VM에 표시되지만 네트워크 트래픽은 게스트 운영 체제의 네트워크 스택의 방화벽과 게스트 인터페이스에 연결된 **libvirt** 네트워크 필터링 규칙에 의해 영향을 받습니다.
- 네트워크의 VM은 호스트 외부의 위치에 연결할 수 있지만 볼 수는 없습니다. 아웃바운드 트래픽은 호스트 시스템의 방화벽뿐만 아니라 **NAT** 규칙의 영향을 받습니다.

다음 다이어그램은 기본 VM 네트워크 구성을 보여줍니다.



RHEL_52_1219

13.2. 가상 머신 네트워크 인터페이스 관리에 웹 콘솔 사용

RHEL 8 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.](#)

13.2.1. 웹 콘솔에서 가상 네트워크 인터페이스 정보 보기 및 편집

RHEL 8 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 **VM**을 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 **VM**에 대해 구성된 가상 네트워크 인터페이스와 네트워크 인터페이스 추가,삭제,편집 또는 연결 해제 옵션이 표시됩니다 .

Network interfaces						Add network interface	
Type	Model type	MAC address	IP address	Source	State		
network	virtio	52:54:00:b4:2a:62	inet 192.168.122.9/24	default	up	Delete	Unplug Edit

+ 정보에는 다음이 포함됩니다.

- **type** - **VM**의 네트워크 인터페이스 유형입니다. 유형에는 가상 네트워크, **LAN**에 대한

브리지, 직접 연결이 포함됩니다.



참고

RHEL 8 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 **Edit(편집)**를 클릭합니다. **Virtual Network Interface Settings**(가상 네트워크 인터페이스 설정) 대화 상자가 열립니다.

52:54:00:b4:2a:62 virtual network interface settings
✕

Interface type ⓘ

Virtual network

▼

Source

default

▼

Model

(Linux, perf)

▼

MAC address

52:64:00:b4:2a:63

Save

Cancel

4. 인터페이스 유형, 소스, 모델 또는 **MAC** 주소를 변경합니다.
5. 저장을 클릭합니다. 네트워크 인터페이스가 수정되었습니다.



참고

가상 네트워크 인터페이스 설정에 대한 변경 사항은 **VM**을 다시 시작한 후에만 적용됩니다.

또한 **MAC** 주소는 **VM**이 종료된 경우에만 수정할 수 있습니다.

추가 리소스

- [웹 콘솔을 사용하여 가상 머신 정보 보기](#)

13.2.2. 웹 콘솔에서 가상 네트워크 인터페이스 추가 및 연결

RHEL 8 웹 콘솔을 사용하여 가상 네트워크 인터페이스를 만들고 **VM**(가상 머신)을 연결할 수 있습니다.

사전 요구 사항

- 웹 콘솔 **VM** 플러그인이 [시스템에 설치되어](#) 있습니다.

절차

- Virtual Machines** (가상 시스템) 인터페이스에서 표시할 정보가 있는 **VM**을 클릭합니다.

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

- 네트워크 인터페이스로 스크롤합니다.

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

Network interfaces						Add network interface
Type	Model type	MAC address	IP address	Source	State	
network	virtio	52:54:00:b4:2a:62	Unknown	default	down	Delete Plug Edit

3.

연결하려는 가상 네트워크 인터페이스 행에서 **Plug (연결)**를 클릭합니다.

선택한 가상 네트워크 인터페이스는 **VM**에 연결됩니다.

13.2.3. 웹 콘솔에서 가상 네트워크 인터페이스 연결 해제 및 제거

RHEL 8 웹 콘솔을 사용하면 선택한 **VM**(가상 머신)에 연결된 가상 네트워크 인터페이스의 연결을 해제할 수 있습니다.

사전 요구 사항

•

웹 콘솔 **VM** 플러그인이 **시스템에 설치되어** 있습니다.

절차

1.

Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 **VM**을 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 **VM**에 대해 구성된 가상 네트워크 인터페이스와 네트워크 인터페이스 추가,삭제,편집 또는 연결 해제 옵션이 표시됩니다 .

Network interfaces						Add network interface	
Type	Model type	MAC address	IP address	Source	State		
network	virtio	52:54:00:b4:2a:62	inet 192.168.122.9/24	default	up	Delete	Unplug Edit

3.

연결을 끊으려는 가상 네트워크 인터페이스 행에서 **Unplug (연결 해제)**를 클릭합니다.

선택한 가상 네트워크 인터페이스는 **VM**에서 연결을 끊습니다.

13.3. 권장되는 가상 머신 네트워킹 구성

많은 시나리오에서 기본 VM 네트워킹 구성만으로도 충분합니다. 그러나 구성을 조정해야 하는 경우 CLI(명령줄 인터페이스) 또는 RHEL 8 웹 콘솔을 사용하여 이를 수행할 수 있습니다. 다음 섹션에서는 이러한 상황에 대해 선택한 VM 네트워킹 설정에 대해 설명합니다.

13.3.1. 명령줄 인터페이스를 사용하여 외부에 표시되는 가상 머신 구성

기본적으로 새로 생성된 VM은 호스트에서 기본 가상 브리지인 **virbr0** 을 사용하는 NAT 유형 네트워크에 연결됩니다. 이렇게 하면 VM에서 호스트의 NIC(네트워크 인터페이스 컨트롤러)를 사용하여 외부 네트워크에 연결할 수 있지만 외부 시스템에서 VM에 연결할 수 없습니다.

VM을 하이퍼바이저와 동일한 외부 네트워크에 표시해야 하는 경우 대신 **bridged 모드**를 사용해야 합니다. 이렇게 하려면 하이퍼바이저의 물리적 네트워크 장치에 연결된 브리지 장치에 VM을 연결합니다. 이에 대한 명령줄 인터페이스를 사용하려면 아래 지침을 따르십시오.

사전 요구 사항

- 기본 NAT 설정을 사용하는 **기존 VM** 종료.
- 하이퍼바이저의 IP 구성. 이는 호스트의 네트워크 연결에 따라 다릅니다. 예를 들어 이 절차에서는 이더넷 케이블을 사용하여 호스트가 네트워크에 연결되어 있고 호스트의 물리적 **NIC MAC** 주소가 **DHCP** 서버의 고정 IP에 할당되는 시나리오를 사용합니다. 따라서 이더넷 인터페이스는 하이퍼바이저 IP로 처리됩니다.

이더넷 인터페이스의 IP 구성을 얻으려면 **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 실제 인터페이스에 대한 브리지 연결을 만들고 설정합니다. 자세한 내용은 **네트워크 브리지** 구성을 참조하십시오.

정적 IP 할당이 사용되는 시나리오에서는 실제 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 네트워크에 있는 다른 물리적 시스템인 것처럼 시스템 네트워크 인터페이스의 IP 및 DHCP 설정을 조정합니다.

이 작업의 특정 단계는 VM에서 사용하는 게스트 OS에 따라 다릅니다. 예를 들어 게스트 OS가 RHEL 8 인 경우 이더넷 연결 구성을 참조하십시오.

검증

1.

새로 생성된 브릿지가 실행 중이며 호스트의 실제 인터페이스와 VM의 인터페이스가 모두 포함되어 있는지 확인합니다.

```
# 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 구성되었습니다.

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 **nftables**를 사용하여 대상 NAT를 설정할 수 있습니다.

추가 리소스

- [웹 콘솔을 사용하여 외부에서 볼 수 있는 가상 머신 구성](#)
- [브리지 모드의 가상 네트워킹](#)

13.3.2. 웹 콘솔을 사용하여 외부에서 표시되는 가상 머신 구성

기본적으로 새로 생성된 VM은 호스트에서 기본 가상 브리지인 **virbr0** 을 사용하는 NAT 유형 네트워크에 연결됩니다. 이렇게 하면 VM에서 호스트의 NIC(네트워크 인터페이스 컨트롤러)를 사용하여 외부 네트워크에 연결할 수 있지만 외부 시스템에서 VM에 연결할 수 없습니다.

VM을 하이퍼바이저와 동일한 외부 네트워크에 표시해야 하는 경우 대신 **bridged 모드**를 사용해야 합니다. 이렇게 하려면 하이퍼바이저의 물리적 네트워크 장치에 연결된 브리지 장치에 VM을 연결합니다. RHEL 8 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** 구성. 이는 호스트의 네트워크 연결에 따라 다릅니다. 예를 들어 이 절차에서는 이더넷 케이블을 사용하여 호스트가 네트워크에 연결되어 있고 호스트의 물리적 **NIC MAC** 주소가 **DHCP** 서버의 고정 **IP**에 할당되는 시나리오를 사용합니다. 따라서 이더넷 인터페이스는 하이퍼바이저 **IP**로 처리됩니다.

이더넷 인터페이스의 **IP** 구성을 얻으려면 웹 콘솔의 **Networking(네트워킹)** 탭으로 이동하여 **Interfaces (인터페이스)** 섹션을 참조하십시오.

Interfaces		
Name	IP Address	Sending
enp0s25	10.0.0.148/24, 2a00:1028:83a4:1eda:91c7:667:8845:fa2e/64	2.29 Mbps

절차

1.

호스트에서 실제 인터페이스에 대한 브리지 연결을 만들고 설정합니다. 자세한 내용은 [웹 콘솔의 네트워크 브리지 구성](#)을 참조하십시오.

정적 **IP** 할당이 사용되는 시나리오에서는 실제 이더넷 인터페이스의 **IPv4** 설정을 브리지 인터페이스로 이동해야 합니다.

2.

브릿지 인터페이스를 사용하도록 **VM**의 네트워크를 수정합니다. **VM**의 [네트워크 인터페이스](#) 탭에서 다음을 수행합니다.

a.

네트워크 인터페이스 추가를 클릭합니다.

b.

가상 네트워크 인터페이스 추가 대화 상자에서 다음을 설정합니다.

-

인터페이스 유형에서 브리지에 **LAN**

-

새로 생성된 브리지를 소스로 지정합니다(예: **bridge0**).

c.

추가를 클릭합니다.

d.

선택 사항: **VM**에 연결된 다른 모든 인터페이스는 **Unplug** (연결 해제)를 클릭합니다.

3.

Run(실행)을 클릭하여 **VM**을 시작합니다.

4.

게스트 운영 체제에서 **VM**이 하이퍼바이저와 동일한 네트워크에 있는 다른 물리적 시스템인 것처럼 시스템 네트워크 인터페이스의 **IP** 및 **DHCP** 설정을 조정합니다.

이 작업의 특정 단계는 **VM**에서 사용하는 게스트 **OS**에 따라 다릅니다. 예를 들어 게스트 **OS**가 **RHEL 8**인 경우 **이더넷 연결** 구성을 참조하십시오.

검증

1.

호스트 웹 콘솔의 **Networking** (네트워킹) 탭에서 새로 생성된 브리지가 있는 행을 클릭하여 호스트가 실행 중이며 호스트의 실제 인터페이스와 **VM**의 인터페이스가 모두 포함되어 있는지 확인합니다.

bridge0	Bridge	54:EE:75:49:DC:46	Delete	✓
Status	10.0.0.148/24, 2a00:1028:83a4:1eda:2d00:bde0:db22:f24c/64, fe80:0:0:0:5c32:895b:51f8:7285/64			
Carrier	Yes			
General	☒ Connect automatically			
IPv4	Automatic (DHCP)			
IPv6	Automatic			
Bridge	Configure			
Ports	Sending	Receiving		
enp0s25	2.03 Kbps	2.09 Kbps	✓	-
vnet0	688 bps	624 bps	✓	-

2.

VM이 하이퍼바이저와 동일한 외부 네트워크에 표시되는지 확인합니다.

a.

게스트 운영 체제에서 시스템의 네트워크 **ID**를 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 구성되었습니다.

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 **nftables**를 사용하여 대상 NAT를 설정할 수 있습니다.

추가 리소스

•

명령줄 인터페이스를 사용하여 외부적으로 표시되는 가상 머신 구성

•

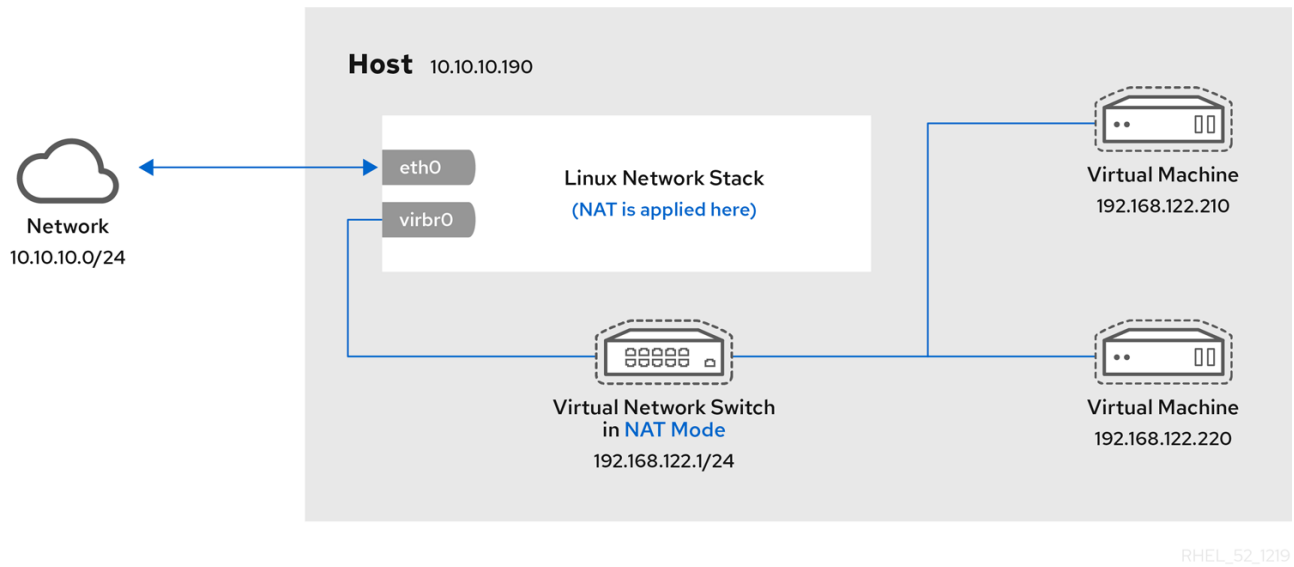
브리지 모드의 가상 네트워킹

13.4. 가상 머신 네트워크 연결 유형

VM의 네트워킹 속성과 동작을 수정하려면 VM에서 사용하는 가상 네트워크 유형 또는 인터페이스를 변경합니다. 다음 섹션에서는 RHEL 8의 VM에서 사용할 수 있는 연결 유형에 대해 설명합니다.

13.4.1. 네트워크 주소 변환을 통한 가상 네트워킹

기본적으로 가상 네트워크 스위치는 NAT(네트워크 주소 변환) 모드에서 작동합니다. SNAT(Source-NAT) 또는 Destination-NAT(DNAT) 대신 IP 마스커레이딩을 사용합니다. IP 마스커레이딩을 사용하면 연결된 VM에서 호스트 시스템의 IP 주소를 사용하여 외부 네트워크와 통신할 수 있습니다. 가상 네트워크 스위치가 NAT 모드에서 작동하면 호스트 외부 컴퓨터는 호스트 내의 VM과 통신할 수 없습니다.



RHEL_52_1219



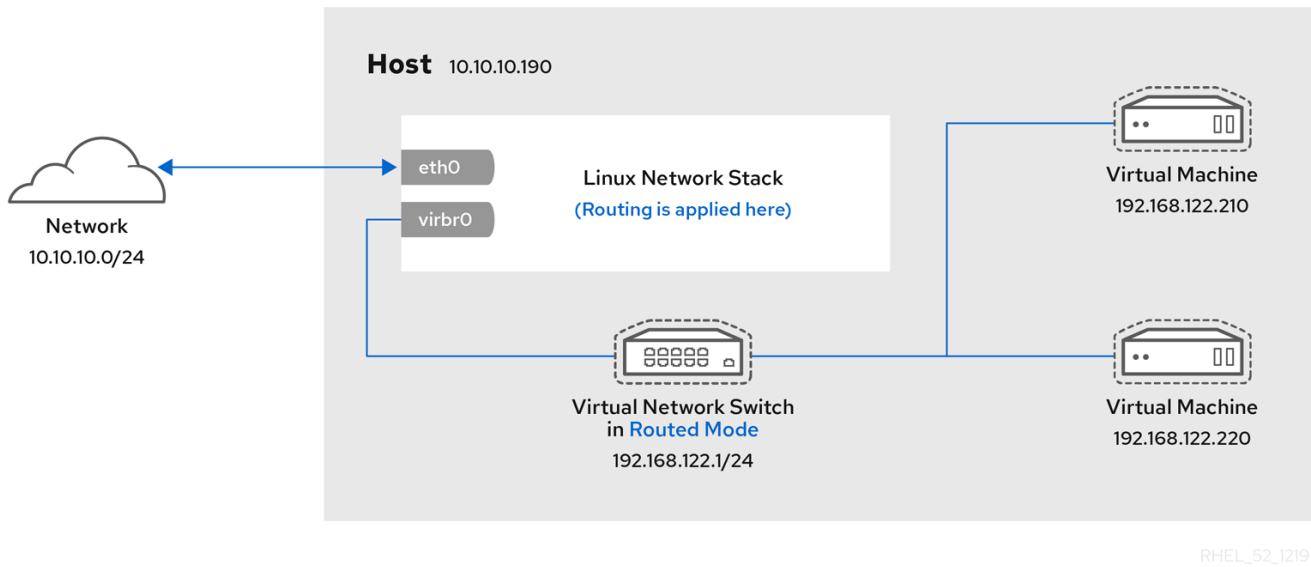
주의

가상 네트워크 스위치는 방화벽 규칙으로 구성된 **NAT**를 사용합니다. 스위치가 실행되는 동안 이러한 규칙을 편집하는 것은 권장되지 않습니다. 잘못된 규칙으로 인해 스위치가 통신할 수 없기 때문입니다.

13.4.2. 라우팅 모드의 가상 네트워킹

Routed 모드를 사용하는 경우 가상 스위치는 호스트 시스템에 연결된 실제 **LAN**에 연결하여 **NAT**를 사용하지 않고 트래픽을 앞뒤로 전달합니다. 가상 스위치는 모든 트래픽을 검사하고 네트워크 패킷에 포함된 정보를 사용하여 라우팅 결정을 내릴 수 있습니다. 이 모드를 사용하는 경우 **VM**(가상 시스템)은 모두 호스트 시스템과는 별도로 단일 서브넷에 있습니다. **VM** 서브넷은 호스트 시스템에 있는 가상 스위치를 통해 라우팅됩니다. 이렇게 하면 들어오는 연결을 활성화할 수 있지만 외부 네트워크의 시스템에는 추가 라우팅 테이블 항목이 필요합니다.

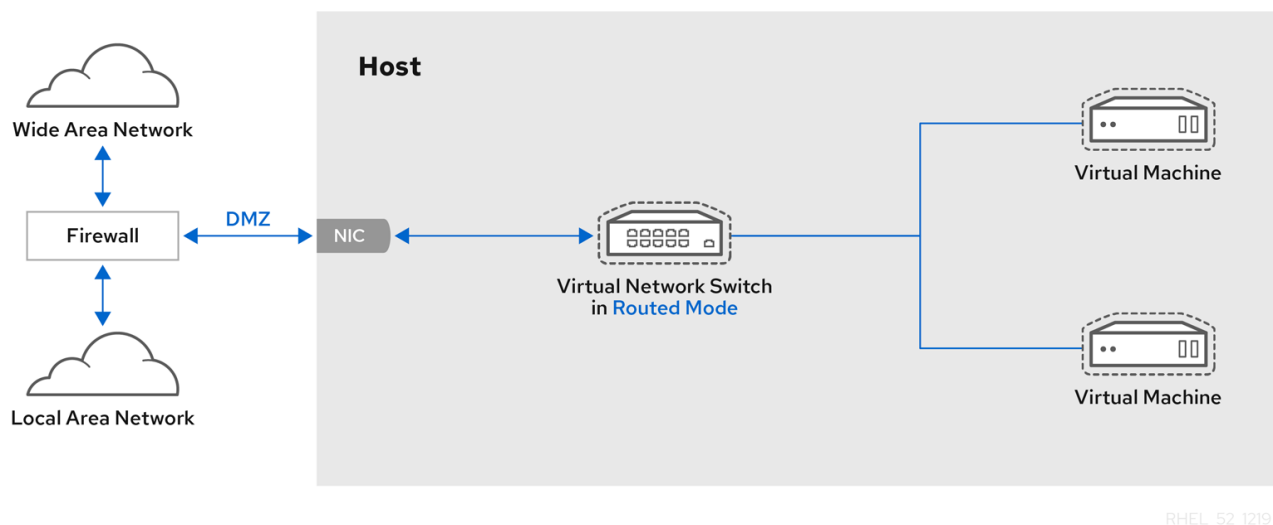
라우팅 모드는 **IP** 주소를 기반으로 라우팅을 사용합니다.



라우팅 모드를 사용하는 일반적인 토폴로지에는 **DMZ** 및 가상 서버 호스팅이 포함됩니다.

DMZ

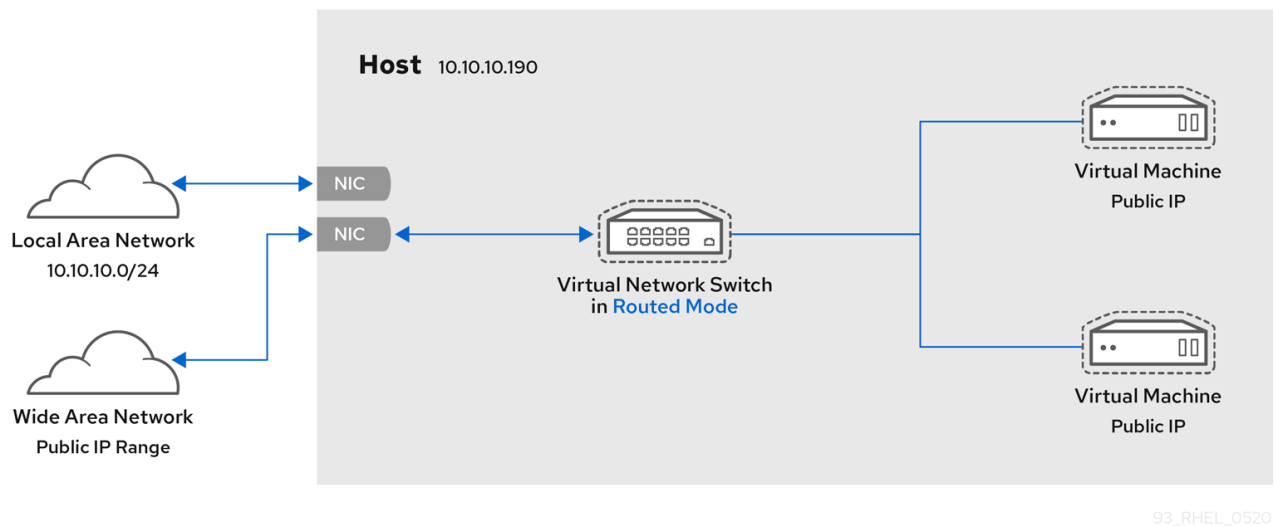
보안상의 이유로 하나 이상의 노드가 제어된 서브네트워크에 배치되는 네트워크를 만들 수 있습니다. 이러한 서브네트워크를 **DMZ(demilitarized zone)**라고 합니다.



DMZ의 호스트 시스템은 일반적으로 **WAN(외부)** 호스트 시스템과 **LAN(내부)** 호스트 시스템에 서비스를 제공합니다. 이 경우 여러 위치에서 액세스할 수 있어야 하고 이러한 위치는 보안 및 신뢰 수준에 따라 서로 다른 방식으로 제어 및 운영되므로 라우팅 모드는 이 환경에 가장 적합한 구성입니다.

가상 서버 호스팅

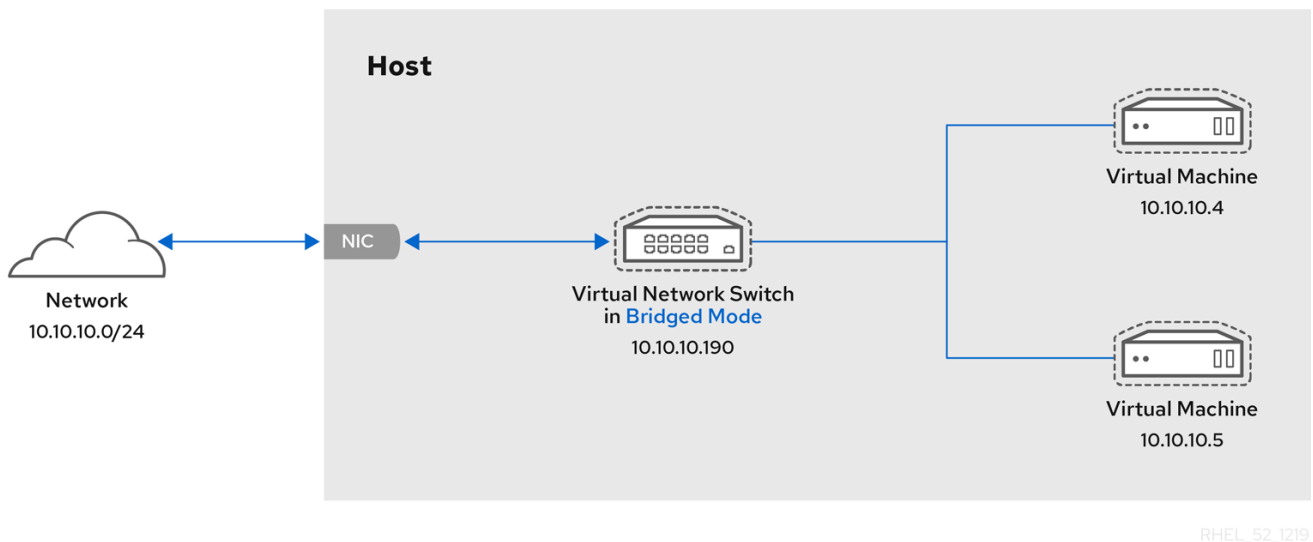
가상 서버 호스팅 공급업체에는 각각 두 개의 실제 네트워크 연결이 있는 여러 개의 호스트 시스템이 있을 수 있습니다. 인터페이스 한 개는 관리 및 회계에 사용되며 다른 인터페이스는 **VM**이 연결하는 데 사용됩니다. 각 **VM**에는 고유한 공용 **IP** 주소가 있지만, 호스트 시스템은 내부 관리자만 **VM**을 관리할 수 있도록 개인 **IP** 주소를 사용합니다.



13.4.3. 브릿지 모드에서 가상 네트워킹

대부분의 VM 네트워킹 모드에서 VM은 **virbr0** 가상 브리지를 자동으로 생성하고 연결합니다. 반대로, 브릿지 모드에서 VM은 호스트의 기존 **Linux** 브리지에 연결됩니다. 결과적으로 VM은 실제 네트워크에 직접 표시됩니다. 이 경우 들어오는 연결을 활성화하지만 추가 라우팅 테이블 항목이 필요하지 않습니다.

브리지 모드는 **MAC** 주소를 기반으로 연결 전환을 사용합니다.



브릿지 모드에서는 VM이 호스트 시스템과 동일한 서브넷 내에 표시됩니다. 동일한 물리적 네트워크의 다른 모든 실제 시스템은 VM을 감지하고 액세스할 수 있습니다.

브릿지 네트워크 본딩

본딩과 함께 결합하여 하이퍼바이저에서 여러 물리적 브리지 인터페이스를 사용할 수 있습니다. 그런 다음 본딩을 브리지에 추가할 수 있으며, 이후에 VM을 브리지에도 추가할 수 있습니다. 그러나 본딩 드라

이버에는 여러 가지 작업 모드가 있으며, 이러한 모드가 VM을 사용 중인 브리지와 함께 작동하는 것은 아닙니다.

다음과 같은 **본딩 모드**를 사용할 수 있습니다.

- 모드 1
- 모드 2
- 모드 4

반대로 모드 0, 3, 5 또는 6을 사용하면 연결이 실패할 수 있습니다. 또한 ARP(Address Resolution Protocol) 모니터링이 제대로 작동하지 않기 때문에 본딩 모드를 모니터링하는 데 미디어 독립 인터페이스(MII) 모니터링을 사용해야 합니다.

본딩 모드에 대한 자세한 내용은 [Red Hat 지식베이스](#)를 참조하십시오.

공통 시나리오

브릿지 모드의 가장 일반적인 사용 사례는 다음과 같습니다.

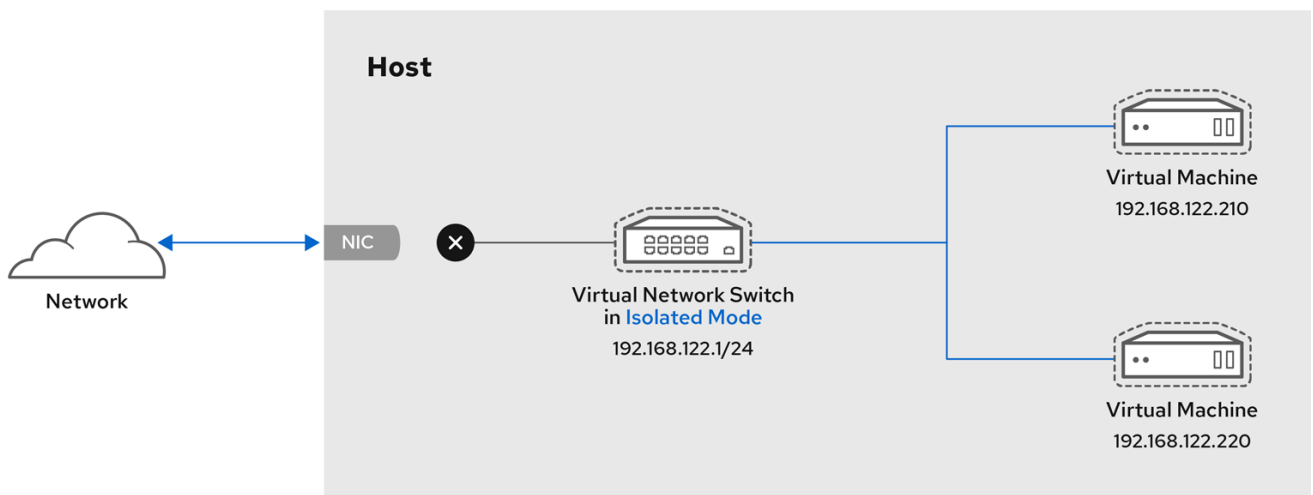
- 기존 네트워크에 VM을 호스트 시스템과 함께 배포하여 최종 사용자에게 가상 시스템과 실제 시스템을 구분합니다.
- 기존 물리적 네트워크 구성 설정을 변경하지 않고 VM 배포.
- 기존 물리적 네트워크에서 쉽게 액세스할 수 있어야 하는 VM 배포. DHCP 서비스에 액세스해야 하는 물리적 네트워크에 VM 배치.
- 가상 LAN(VLAN)이 사용되는 기존 네트워크에 VM 연결.

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

13.4.4. 분리된 모드의 가상 네트워킹

분리된 모드를 사용하는 경우 가상 스위치에 연결된 가상 시스템은 서로 통신하고 호스트 시스템과 통신할 수 있지만 트래픽은 호스트 시스템 외부에서 전달되지 않으며 호스트 시스템 외부에서 트래픽을 수신할 수 없습니다. 이 모드에서 **dnsmasq**를 사용하려면 **DHCP**와 같은 기본 기능에 필요합니다.



RHEL_52_1219

13.4.5. 오픈 모드에서 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.

13.4.6. 가상 머신 연결 유형 비교

다음 표에서는 선택한 유형의 **VM**(가상 시스템) 네트워크 구성이 연결할 수 있는 위치와 표시되는 위치에 대한 정보를 제공합니다.

표 13.1. 가상 머신 연결 유형

	호스트에 대한 연결	호스트의 다른 VM에 연결	외부 위치에 연결	외부 위치로 볼 수 있습니다
브릿지 모드	YES	YES	YES	YES
NAT	YES	YES	YES	no
라우팅 모드	YES	YES	YES	YES
격리 모드	YES	YES	no	no
열기 모드	호스트의 방화벽 규칙에 따라 달라집니다			

13.5. 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.

13.5.1. 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_leasehelper /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](#)

13.5.2. 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 **VM**을 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
```

o

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.

13.5.3. 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 **PXE**에서 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. **PXE**가 올바르게 구성된 경우 VM은 **PXE** 서버에서 사용 가능한 부팅 이미지에서 부팅됩니다.

추가 리소스

- [네트워크 브리지 구성](#)

13.6. 추가 리소스

- [네트워킹 구성 및 관리](#)

- 특정 네트워크 인터페이스 카드를 **SR-IOV** 장치로 연결하여 **VM** 성능을 향상시킵니다.

14장. 호스트와 해당 가상 머신 간의 파일 공유

호스트 시스템과 VM(가상 시스템) 간에 데이터를 공유해야 하는 경우가 많습니다. 이렇게 빠르고 효율적으로 수행하려면 시스템에 **NFS** 또는 **Samba** 파일 공유를 설정할 수 있습니다.

14.1. NFS를 사용하여 호스트와 LINUX 가상 시스템 간 파일 공유

RHEL 8 호스트 시스템과 연결된 **Linux VM** 간의 효율적인 파일 공유를 위해 VM이 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 생성합니다.

```
# mkdir shared-files
```

- 호스트는 VM의 네트워크를 통해 표시되고 연결할 수 있습니다. 일반적으로 가상 네트워크의 **NAT** 및 **브리지** 유형을 사용하여 VM이 연결된 경우입니다. 그러나 **macvtap** 연결의 경우 먼저 호스트에 **macvlan** 기능을 설정해야 합니다. 이렇게 하려면 다음을 수행합니다.

1. 호스트의 **/etc/systemd/network/** 디렉터리에 네트워크 장치 파일을 만듭니다(예: **vm-macvlan.netdev**).

```
# vim /etc/systemd/network/vm-macvlan.netdev
```

2. 네트워크 장치 파일을 편집하여 다음 내용을 포함합니다. **vm-macvlan** 을 네트워크 장치에 대해 선택한 이름으로 바꿀 수 있습니다.

```
[NetDev]
Name=vm-macvlan
Kind=macvlan
```

```
[MACVLAN]
Mode=bridge
```

3. **macvlan** 네트워크 장치에 대한 네트워크 구성 파일(예: **vm-macvlan.network**)을 만듭니다

니다.

```
# vim /etc/systemd/network/vm-macvlan.network
```

4.

네트워크 구성 파일을 편집하여 다음 내용을 포함합니다. **vm-macvlan** 을 네트워크 장치에 대해 선택한 이름으로 바꿀 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 이름을 확실하지 않은 경우 호스트에서 **ifconfig** 명령을 사용하여 활성 네트워크 인터페이스 목록을 가져올 수 있습니다.

6.

물리적 네트워크 구성 파일을 편집하여 물리적 네트워크를 **macvlan** 인터페이스의 일부로 만듭니다. 이 경우 **vm-macvlan**:

```
[Match]
Name=enp4s0

[Network]
MACVLAN=vm-macvlan
```

7.

호스트를 재부팅합니다.

•

선택 사항: 보안을 강화하기 위해 **VM**이 **NFS** 버전 4 이상과 호환되는지 확인하십시오.

절차

1.

호스트에서 **NFS**(네트워크 파일 시스템)로 공유할 파일과 함께 디렉터리를 내보냅니다.

a.

파일을 공유할 각 가상 머신의 **IP** 주소를 가져옵니다. 다음 예제에서는 **testguest1** 및 **testguest 2**의 **IP**를 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 디렉토리, 공유할 **VM**의 **IP** 및 공유 옵션을 포함하는 행을 추가합니다.

```
Shared directory VM1-IP(options) VM2-IP(options) [...]
```

예를 들어 다음은 **testguest1** 및 **testguest 2**를 사용하여 호스트의 **/usr/local/shared-files** 디렉터리를 공유하고 **VM**에서 디렉토리 내용을 편집할 수 있도록 합니다.

```
/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 공유할 VM에 의해 호스트에 연결하는 데 사용되는 네트워크입니다. 일반적으로 이것은 **virbr0** 입니다.

2.

/etc/exports 파일에 지정된 VM의 게스트 OS에서 내보낸 파일 시스템을 마운트합니다.

a.

공유 파일 시스템의 마운트 지점으로 사용할 디렉토리를 만듭니다(예: **/mnt/host-share**):

```
# mkdir /mnt/host-share
```

b.

호스트에서 내보낸 디렉토리를 마운트 지점에 마운트합니다. 이 예에서는 게스트의 **/mnt/host-share**에 **192.168.124.1** 호스트에서 내보낸 **/usr/local/shared-files** 디렉토리를 마운트합니다.

```
# mount 192.168.124.1:/usr/local/shared-files /mnt/host-share
```

검증

-

마운트에 성공했는지 확인하려면 마운트 지점의 공유 디렉터리에 액세스하고 살펴봅니다.

```
# cd /mnt/host-share
# ls
shared-file1 shared-file2 shared-file3
```

추가 리소스

-

[NFS 서버 내보내기 구성](#)

-

[방화벽 뒤에서 실행되도록 NFS 서버 구성](#)

14.2. SAMBA를 사용하여 호스트와 WINDOWS 가상 시스템 간에 파일 공유

RHEL 8 호스트 시스템과 연결된 **Windows VM** 간의 효율적인 파일 공유를 위해 VM이 액세스할 수 있는 **Samba** 서버를 준비할 수 있습니다.

사전 요구 사항

- **samba** 패키지가 호스트에 설치되어 있습니다. 그렇지 않은 경우 다음을 수행합니다.

```
# yum install samba
```

- 호스트는 VM의 네트워크를 통해 표시되고 연결할 수 있습니다. 일반적으로 가상 네트워크의 NAT 및 브리지 유형을 사용하여 VM이 연결된 경우입니다. 그러나 **macvtap** 연결의 경우 먼저 호스트에 **macvlan** 기능을 설정해야 합니다. 이렇게 하려면 다음을 수행합니다.

1. 네트워크 장치 파일(예: 호스트의 **/etc/systemd/network/** 디렉터리에 **vm-macvlan.netdev**)을 만듭니다.

```
# vim /etc/systemd/network/vm-macvlan.netdev
```

2. 네트워크 장치 파일을 편집하여 다음 내용을 포함합니다. **vm-macvlan** 을 네트워크 장치에 대해 선택한 이름으로 바꿀 수 있습니다.

```
[NetDev]
Name=vm-macvlan
Kind=macvlan
```

```
[MACVLAN]
Mode=bridge
```

3. **macvlan** 네트워크 장치에 대한 네트워크 구성 파일(예: **vm-macvlan.network**)을 만듭니다.

```
# vim /etc/systemd/network/vm-macvlan.network
```

4. 네트워크 구성 파일을 편집하여 다음 내용을 포함합니다. **vm-macvlan** 을 네트워크 장치에 대해 선택한 이름으로 바꿀 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** 인터페이스의 일부로 만듭니다. 이 경우 **vm-macvlan**:

```
[Match]
Name=enp4s0

[Network]
MACVLAN=vm-macvlan
```

7.

호스트를 재부팅합니다.

절차

1.

호스트에서 **Samba** 공유를 만들고 외부 시스템에서 액세스할 수 있도록 합니다.

a.

Samba에 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.

파일의 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
```

hosts allow 행은 VM 네트워크의 호스트에 대해서만 공유의 접근성을 제한합니다. 누구나 공유에 액세스할 수 있도록 하려면 행을 제거합니다.

c.

/samba/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 액세스하고 **VM**에 대해 수정할 수 있도록 허용합니다.

```
# chmod -R 0755 /samba/VM-share/
# chown -R nobody:nobody /samba/VM-share/
```

g.

SELinux Samba 공유 레이블을 **/etc/samba/VM-share/**에 추가합니다.

```
# chcon -t samba_share_t /samba/VM-share/
```

2.

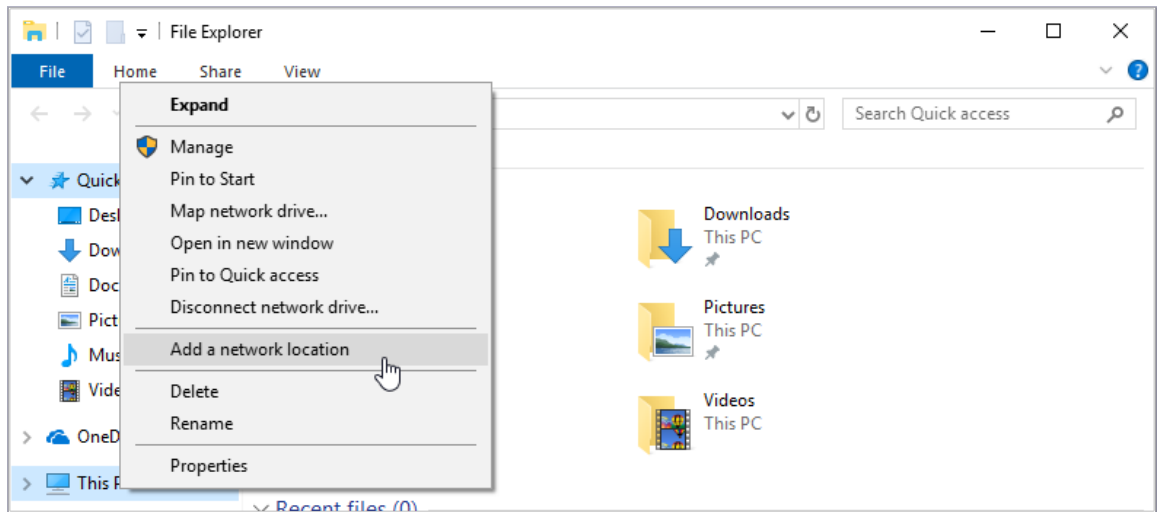
Windows 게스트 운영 체제에서 **Samba** 공유를 네트워크 위치로 연결합니다.

a.

File Explorer를 열고 **"This PC"**를 마우스 오른쪽 버튼으로 클릭합니다.

b.

컨텍스트 메뉴에서 네트워크 위치 추가를 클릭합니다.

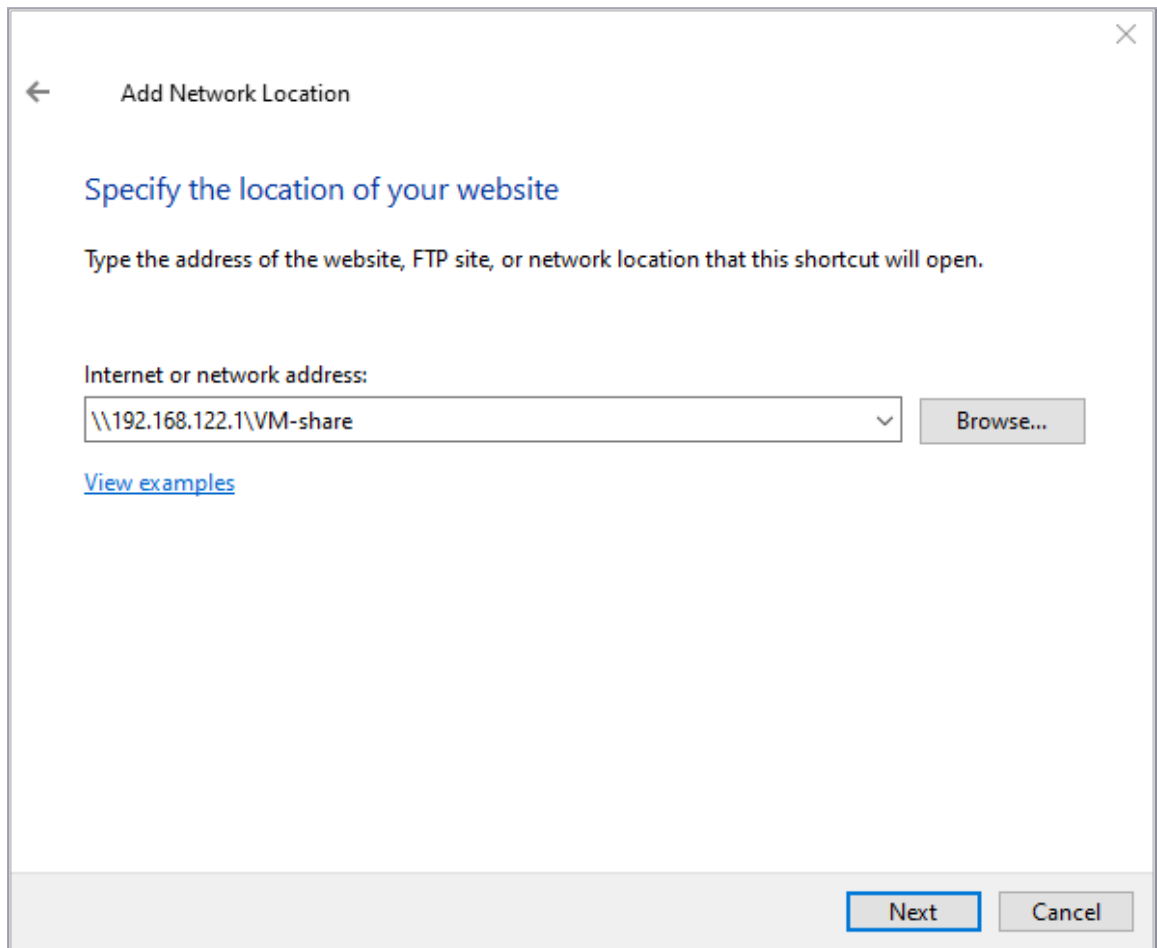


c.

열려 있는 **네트워크 위치 추가** 마법사에서 **"사용자 지정 네트워크 위치 선택"**을 선택하고 다음을 클릭합니다.

d.

"Internet or network address(인터넷 또는 네트워크 주소)" 필드에 **host-IP/VM-share**를 입력합니다. 여기서 **host-IP**는 호스트의 IP 주소입니다. 일반적으로 호스트 IP는 VM의 기본 게이트웨이입니다. 그런 다음 **Next (다음)**를 클릭합니다.



- e. 마법사에서 공유 디렉터리의 이름을 바꾸려는지 묻는 경우 기본 이름을 유지합니다. 이렇게 하면 VM과 게스트 전체에서 파일 공유 구성이 일관되게 유지됩니다. 다음을 클릭합니다.
- f. 네트워크 위치에 액세스하는 데 성공한 경우 이제 **Finish**(완료)를 클릭하고 공유 디렉터리를 열 수 있습니다.

추가 리소스

- 독립 실행형 서버로 **Samba** 설정
- **POSIX ACL**을 사용하는 **Samba** 파일 공유 설정
- 인증 없이 액세스를 허용하도록 공유 구성

14.3. 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**와 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.

Shared directories ⓘ		Add shared directory
Source path	Mount tag	
/home/centurion/	Ace	Remove

3. 공유 디렉터리 추가 를 클릭합니다.

게스트를 사용하여 호스트 디렉터리 공유 대화 상자가 나타납니다.

Share a host directory with the guest

Shared host directories need to be manually mounted inside the VM

Source path

Mount tag

Hide additional options

Extended attributes ☐ Enable/disable extended attributes (xattr) on files and directories

Share Cancel

4.

다음 정보를 입력합니다.

- **source path** - 공유할 호스트 디렉터리의 경로입니다.
- **Mount tag** - VM에서 디렉터를 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.

14.4. 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 공유되

는 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.

Shared directories ⓘ		Add shared directory
Source path	Mount tag	
/home/centurion/	Ace	Remove

3. VM과 공유할 디렉토리 옆에 있는 **Remove** (제거)를 클릭합니다.

파일 시스템 제거 대화 상자가 나타납니다.

Remove filesystem Ace

×

Confirm this action

Remove

Cancel

4. 제거를 클릭합니다.

선택한 디렉토리는 **VM**과 공유되지 않습니다.

검증

- 공유 디렉터리는 더 이상 사용할 수 없으며 **VM**에서 액세스할 수 있습니다.

15장. 가상 머신 보안

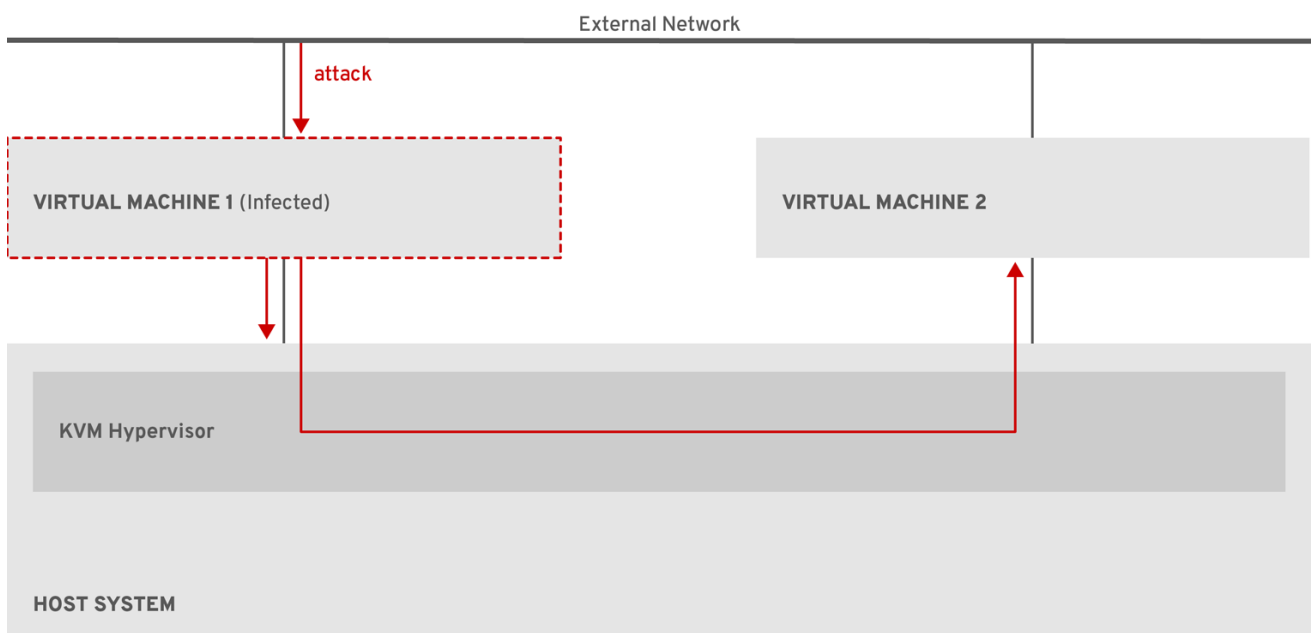
VM(가상 시스템)이 있는 **RHEL 8** 시스템의 관리자로서, 가상 머신(**VM**)이 악의적인 소프트웨어로 인해 게스트와 호스트 **OS**의 위험을 최대한 안전하게 유지할 수 있습니다.

이 문서에서는 **RHEL 8** 호스트에서 **VM**을 보호하는 메커니즘과 **VM**의 보안을 향상시키는 방법 목록을 간략하게 설명합니다.

15.1. 가상 머신에서 보안이 작동하는 방식

VM(가상 시스템)을 사용하는 경우 여러 운영 체제를 단일 호스트 시스템에 보유할 수 있습니다. 이러한 시스템은 하이퍼바이저를 통해 호스트와 연결되며 일반적으로 가상 네트워크를 통해 연결됩니다. 결과적으로 각 **VM**은 악의적인 소프트웨어를 사용하여 호스트를 공격하기 위한 벡터로 사용할 수 있으며, 호스트를 **VM**을 공격하기 위한 벡터로 사용할 수 있습니다.

그림 15.1. 가상화 호스트의 잠재적인 텔웨어 공격 벡터

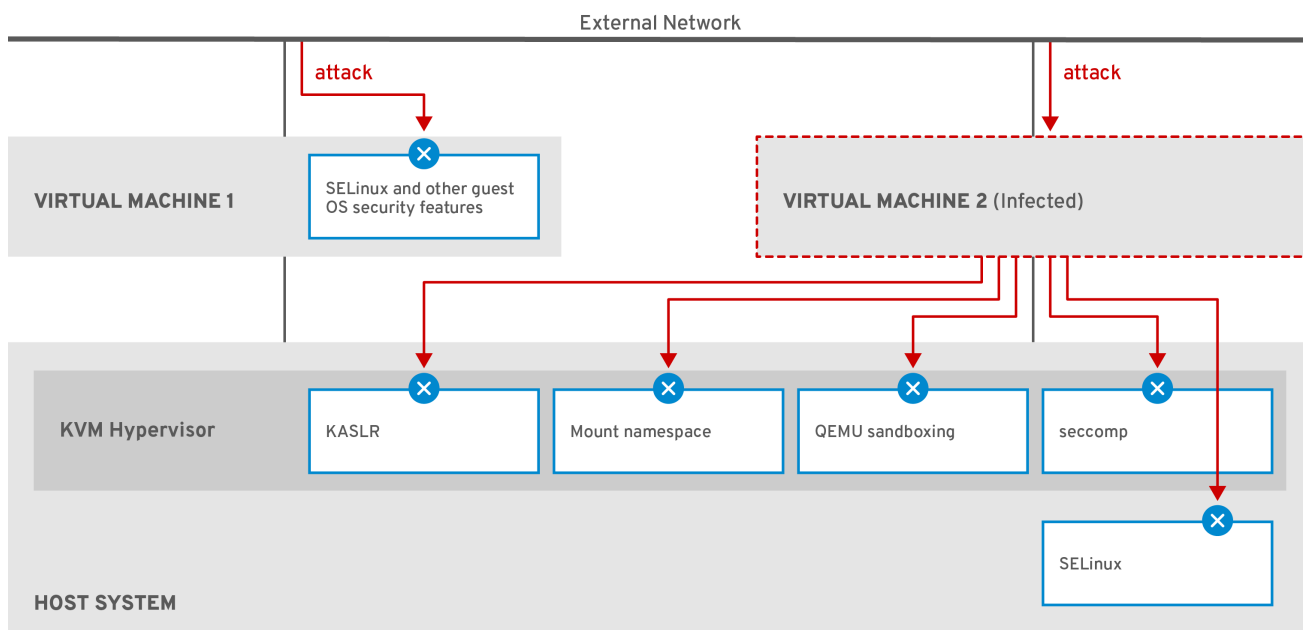


RHEL_7_0319

하이퍼바이저는 호스트 커널을 사용하여 **VM**을 관리하므로 **VM**의 운영 체제에서 실행되는 서비스는 악의적인 코드를 호스트 시스템에 삽입하는 데 자주 사용됩니다. 그러나 호스트 및 게스트 시스템의 여러 보안 기능을 사용하여 이러한 보안 위협으로부터 시스템을 보호할 수 있습니다.

SELinux 또는 **QEMU** 샌드박스나 같은 이러한 기능은 악의적인 코드가 하이퍼바이저를 공격하고 호스트와 **VM** 간 전송을 어렵게 만드는 다양한 조치를 제공합니다.

그림 15.2. 가상화 호스트에서 텔웨어 공격 방지



RHEL_7_0319

RHEL 8에서 VM 보안을 위해 제공하는 대부분의 기능은 항상 활성화되어 있으며 활성화 또는 구성하지 않아도 됩니다. 자세한 내용은 [가상 머신 보안의 자동 기능을 참조하십시오](#).

또한 다양한 모범 사례를 준수하여 VM과 하이퍼바이저의 취약점을 최소화할 수 있습니다. 자세한 내용은 [가상 머신 보안을 위한 모범 사례](#)를 참조하십시오.

15.2. 가상 머신 보안을 위한 모범 사례

아래 지침에 따라 가상 시스템이 악의적인 코드로 손상되고 공격 벡터로 사용되어 호스트 시스템을 노출하는 위험이 크게 줄어듭니다.

게스트 측에서 다음을 수행합니다.

-

가상 시스템이 실제 시스템인 것처럼 보호합니다. 보안을 강화하는 데 사용할 수 있는 특정 방법은 게스트 OS에 따라 다릅니다.

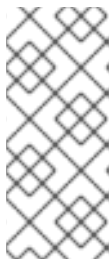
VM이 RHEL 8을 실행하는 경우 [Securing Red Hat Enterprise Linux 8](#)에서 게스트 시스템의 보안 향상에 대한 자세한 내용은 [Securing Red Hat Enterprise Linux 8](#)에서 참조하십시오.

호스트 측에서 다음을 수행합니다.

- VM을 원격으로 관리하는 경우 **SSH** 등의 암호화 유틸리티 및 VM 연결에 **SSL** 과 같은 네트워크 프로토콜을 사용합니다.
- SELinux가 강제 모드인지 확인합니다.

```
# getenforce
Enforcing
```

SELinux가 비활성화되었거나 허용 모드의 경우 강제 모드 활성화에 대한 지침은 [SELinux](#) 문서 사용을 참조하십시오.



참고

SELinux Enforcing 모드에서는 **sVirt RHEL 8** 기능도 활성화합니다. 가상화를 위한 특수 **SELinux** 부울 집합입니다. VM 보안 관리를 위해 [수동으로 조정](#)할 수 있습니다.

- **SecureBoot** 로 VM 사용 :
- SecureBoot**는 VM이 암호화 방식으로 서명된 **OS**를 실행할 수 있도록 하는 기능입니다. 이로 인해 텔웨어 공격으로 **OS**가 변경된 VM이 부팅되지 않습니다.
- SecureBoot**는 **OVMF** 펌웨어를 사용하는 **Linux VM**을 설치할 때만 적용할 수 있습니다. 자세한 내용은 [SecureBoot 가상 머신 생성](#)을 참조하십시오.
- **qemu-kvm** 과 같은 **qemu-*** 명령을 사용하지 마십시오.

QEMU는 **RHEL 8**의 가상화 아키텍처의 필수 구성 요소이지만 수동으로 관리하기 어렵고 **QEMU** 구성이 잘못되어 보안 취약점이 발생할 수 있습니다. 따라서 **qemu-*** 명령 사용은 **Red Hat**에서 지원하지 않습니다. 대신, 모범 사례에 따라 **QEMU**를 오케스트레이션하므로 **virsh**, **virt-install**, **virt-xml** 과 같은 **libvirt** 유틸리티를 사용하십시오.

추가 리소스

- [RHEL에서 가상화를 위한 SELinux 부울](#)

15.3. SECUREBOOT 가상 머신 생성

SecureBoot 기능을 사용하는 **Linux VM**(가상 머신)을 생성하여 **VM**이 암호화 서명 **OS**를 실행하고 있는지 확인할 수 있습니다. **VM**의 게스트 **OS**가 악성 시스템에 의해 변경된 경우 유용할 수 있습니다. 이러한 시나리오에서 **SecureBoot**는 **VM**이 부팅되지 않도록 하여 호스트 시스템에 대한 악성 프로그램의 잠재적인 확산을 방지합니다.

사전 요구 사항

- **VM**에서 **Q35** 시스템 유형을 사용하고 있습니다.

- **The edk2-OVMF** 패키지가 설치되어 있습니다.

```
# yum install edk2-ovmf
```

- **OS**(운영 체제) 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 8에서는 호스트 **CD-ROM** 또는 **DVD-ROM** 장치에서 설치할 수 없습니다. **RHEL 8**에서 사용할 수 있는 **VM** 설치 방법을 사용할 때 **CD-ROM** 또는 **DVD-ROM**을 설치 소스로 선택하면 설치에 실패합니다. 자세한 내용은 [Red Hat Knowledgebase](#) 를 참조하십시오.

- 선택 사항: **Kickstart** 파일을 제공하여 설치 시 보다 빠르고 쉽게 구성할 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 비발성 RAM(NVRAM) 설정에 대한 템플릿으로 사용하며 **SecureBoot** 기능을 활성화합니다.

예를 들면 다음과 같습니다.

```
# virt-install --name rhel8sb --memory 4096 --vcpus 4 --os-variant rhel8.0 --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-8.0-installation.iso
```

2.

화면 지침에 따라 OS 설치 절차를 따릅니다.

검증

1.

게스트 OS가 설치되면 그래픽 게스트 콘솔에서 터미널을 열거나 SSH를 사용하여 게스트 OS에 연결하여 VM의 명령줄에 액세스합니다.

2.

VM에서 **SecureBoot**가 활성화되었는지 확인하려면 **mokutil --sb-state** 명령을 사용합니다.

```
# mokutil --sb-state
SecureBoot enabled
```

추가 리소스

•

[AMD64, Intel 64 및 64비트 ARM에 RHEL 8 설치](#)

15.4. 가상 머신 사용자가 사용할 수 있는 작업 제한

경우에 따라 **RHEL 8**에서 호스팅되는 VM(가상 머신)의 사용자가 기본적으로 수행할 수 있는 작업은 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. 이렇게 하려면 다음을 수행합니다.

i.

`/etc/polkit-1/rules.d/` 디렉토리에 새 **.conf** 파일을 만듭니다.

ii.

사용자 지정 정책을 이 파일에 추가하고 저장합니다.

libvirt 제어 정책의 예제와 자세한 내용은 [libvirt 업스트림 문서를 참조하십시오](#).

2.

polkit 에 의해 결정된 액세스 정책을 사용하도록 **VM**을 구성합니다.

a.

`/etc/libvirt/libvirtd.conf` 파일을 엽니다.

b.

`access_drivers = [ "polkit" ]` 줄의 주석을 제거하고 파일을 저장합니다.

3.

libvirtd 서비스를 다시 시작합니다.

```
# systemctl restart libvirtd
```

검증

•

제한하려는 **VM** 작업의 사용자로 제한된 작업 중 하나를 수행합니다.

예를 들어 권한이 없는 사용자가 시스템 세션에서 생성된 **VM**을 볼 수 없는 경우:

```
$ virsh -c qemu:///system list --all
```

```
Id Name State
```

```
-----
```

시스템에 하나 이상의 VM이 있어도 이 명령에 VM이 나열되지 않은 경우 **polkit**은 권한이 없는 사용자에게 대한 작업을 성공적으로 제한합니다.

문제 해결

-

현재 **polkit**을 사용하도록 **libvirt**를 구성하면 **libvirt-dbus** 서비스와 호환되지 않아 **RHEL 8** 웹 콘솔을 사용하여 VM에 연결할 수 없습니다.

웹 콘솔에서 VM에 대한 세분화된 액세스 제어가 필요한 경우 사용자 정의 **D-Bus** 정책을 생성하는 것이 좋습니다. 자세한 내용은 **Red Hat** 지식베이스에서 **Cockpit**에서 가상 시스템을 세 부적으로 제어하는 방법을 참조하십시오.

추가 리소스

-

man polkit 명령

-

polkit 액세스 제어 정책의 **libvirt** 매뉴얼 페이지

15.5. 가상 머신 보안을 위한 자동 기능

가상 시스템 보안 **유지**에 나열된 가상 시스템의 보안을 수동으로 개선하는 방법 외에도 **libvirt** 소프트웨어 제품군에서 여러 보안 기능을 제공하며 **RHEL 8**에서 가상화를 사용할 때 자동으로 활성화됩니다. 여기에는 다음이 포함됩니다.

시스템 및 세션 연결

RHEL 8에서 가상 머신 관리에 사용 가능한 모든 유틸리티에 액세스하려면 **libvirt(qemu:///system)**의 시스템 연결을 사용해야 합니다. 이렇게 하려면 시스템에 대한 루트 권한이 있거나 **libvirt** 사용자 그룹의 일부여야 합니다.

libvirt 그룹에 없는 루트가 아닌 사용자는 리소스에 액세스할 때 로컬 사용자의 액세스 권한을 고려해야 하는 **libvirt(qemu:///session)**의 세션 연결만 액세스할 수 있습니다. 예를 들어 세션 연결을 사용하면 시스템 연결 또는 다른 사용자가 만든 VM을 감지하거나 액세스할 수 없습니다. 또한 사용 가능한 VM 네트워킹 구성 옵션은 크게 제한됩니다.



참고

RHEL 8 문서에서는 시스템 연결 권한이 있다고 가정합니다.

가상 머신 분리

개별 **VM**은 호스트에서 격리된 프로세스로 실행되며 호스트 커널에 의해 적용된 보안을 사용합니다. 따라서 **VM**은 동일한 호스트에 있는 다른 **VM**의 메모리 또는 스토리지를 읽거나 액세스할 수 없습니다.

QEMU 샌드박스

QEMU 코드가 호스트의 보안을 손상시킬 수 있는 시스템 호출을 실행하지 못하도록 하는 기능입니다.

KASLR(Kernel Address Space Randomization)

커널 이미지가 압축 해제되는 실제 및 가상 주소의 임의화를 활성화합니다. 따라서 커널 개체의 위치에 따라 게스트 보안 위협을 방지할 수 있습니다.

15.6. 가상화를 위한 SELINUX 부울

RHEL 8 시스템에서 가상 시스템 보안을 세밀하게 구성하려면 호스트에서 **SELinux** 부울을 구성하여 하이퍼바이저가 특정 방식으로 작동하도록 할 수 있습니다.

가상화 관련 부울과 해당 상태를 모두 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 명령에서 **setsebool -P *boolean_name*** 을 **root**로 사용합니다. 부울을 비활성화하려면 **setsebool -P *boolean_name* off** 를 사용합니다.

다음 표에는 **RHEL 8**에서 사용할 수 있는 가상화 관련 부울과 활성화된 경우 수행하는 작업이 나열되어 있습니다.

표 15.1. SELinux 가상화 부울

SELinux 부울	설명
staff_use_svirt	루트가 아닌 사용자가 VM을 sVirt로 만들고 전환할 수 있도록 합니다.
unprivuser_use_svirt	권한이 없는 사용자가 VM을 만들고 sVirt로 전환할 수 있도록 합니다.
virt_sandbox_use_audit	샌드박스 컨테이너에서 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 사용하도록 설정합니다.
virt_transition_userdomain	가상 프로세스가 사용자 도메인으로 실행될 수 있도록 합니다.
virt_use_comm	virt가 직렬/병렬 통신 포트를 사용할 수 있도록 합니다.
virt_use_execmem	제한된 가상 게스트가 실행 가능한 메모리 및 실행 가능한 스택을 사용할 수 있도록 합니다.
virt_use_fusefs	virt에서 FUSE 마운트된 파일을 읽을 수 있도록 합니다.
virt_use_nfs	virt를 통해 NFS 마운트 파일을 관리할 수 있습니다.
virt_use_rawip	virt가 rawip 소켓과 상호 작용할 수 있도록 합니다.
virt_use_samba	virt를 통해 CIFS 마운트 파일을 관리할 수 있습니다.
virt_use_sanlock	제한된 가상 게스트가 온락과 상호 작용할 수 있도록 합니다.
virt_use_usb	virt에서 USB 장치를 사용할 수 있도록 합니다.
virt_use_xserver	가상 시스템이 X Window 시스템과 상호 작용할 수 있도록 합니다.

15.7. IBM Z에서 IBM SECURE EXECUTION 설정

IBM Z 하드웨어를 사용하여 **RHEL 8** 호스트를 실행하는 경우 **VM**에 대한 **IBM Secure Execution**을 구성하여 **VM**(가상 시스템)의 보안을 향상시킬 수 있습니다.

IBM Secure Execution(보호된 가상화이라고도 함)을 사용하면 호스트 시스템이 **VM**의 상태 및 메모리 콘텐츠에 액세스할 수 없습니다. 따라서 호스트가 손상되어도 게스트 운영 체제를 공격하기 위한 벡터로 사용할 수 없습니다. 또한 **Secure Execution**을 사용하여 신뢰할 수 없는 호스트가 **VM**에서 중요한 정보를 얻지 못하도록 할 수 있습니다.

다음 절차에서는 **IBM Z** 호스트의 기존 **VM**을 보안 **VM**으로 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 **CPU**가 **Secure Execution**과 호환됩니다.

- 커널에는 보안 실행 지원이 포함됩니다. 확인하려면 다음을 사용합니다.

```
# ls /sys/firmware | grep uv
```

명령이 출력을 생성하는 경우 커널은 보안 실행을 지원합니다.

- 호스트 **CPU** 모델에는 압축 해제 기능이 포함되어 있습니다. 확인하려면 다음을 사용합니다.

```
# virsh domcapabilities | grep unpack
<feature policy='require' name='unpack'/>
```

명령이 위의 출력을 생성하는 경우 **CPU** 호스트 모델은 **Secure Execution**과 호환됩니다.

- VM의 CPU 모드는 **host-model** 로 설정되어 있습니다. 이를 확인하려면 다음을 사용하고 **vm-name** 을 VM 이름으로 바꿉니다.

```
# virsh dumpxml vm-name | grep "<cpu mode='host-model'/">"
```

명령에서 출력을 생성하는 경우 VM의 CPU 모드가 올바르게 설정됩니다.

- IBM Z 호스트 키 문서를 얻고 확인했습니다. 이 작업을 수행하는 방법은 IBM 문서 [의 호스트 키 문서 확인](#)을 참조하십시오.

절차

호스트에서 다음 단계를 수행합니다.

1. 호스트의 [부팅 구성](#)에 **prot_virt=1** 커널 매개 변수를 추가합니다.

```
# grubby --update-kernel=ALL --args="prot_virt=1"
```

2. 보호할 VM에서 공유 버퍼를 사용하도록 **virtio** 장치를 활성화합니다. 이렇게 하려면 **virsh edit** 를 사용하여 VM의 XML 구성을 수정하고 **add iommu='on'** 을 가진 모든 장치의 **<driver>** 행으로 수정합니다. 예를 들면 다음과 같습니다.

```
<interface type='network'>
  <source network='default'>
  <model type='virtio'>
  <driver name='vhost' iommu='on'>
</interface>
```

장치 구성에 **<driver>** 줄이 포함되어 있지 않으면 대신 **<driver iommu='on'>** 을 추가합니다.

3. 기능이 **Secure Execution**과 호환되지 않으므로 VM에서 메모리 증대를 비활성화합니다. 이렇게 하려면 VM의 XML 구성에 다음 행을 추가합니다.

```
<memballoon model='none'>
```

보호하려는 VM의 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
crashkernel=auto
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
```

4. 생성된 매개 변수 파일에 `options` 행의 내용을 추가하고 `swiotlb=262144` 를 추가합니다.

```
# echo "root=/dev/mapper/rhel-root crashkernel=auto rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap swiotlb=262144" > ~/secure-parameters
```

5. IBM 보안 실행 이미지를 생성합니다.

예를 들어 다음은 `secure-parameters` 파일, `/boot/initramfs-4.18.0-000201C048.crt` 호스트 키 문서를 사용하여 `/boot/vmlinuz-4.18.0- img` 초기 RAM 디스크 파일을 기반으로 `/boot/secure- image` 보안 이미지를 생성합니다.

```
# 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 `initrd` 및 옵션으로 시작하는 행을 필수가 아니므로 제거합니다.

예를 들어 **RHEL 8.3 VM**에서 **/boot/loader/entries/** 디렉터리에서 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 만들기

```
# 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 않은 경우 **Secure Execution**이 활성화된 **VM**은 여전히 해킹 시도 또는 민감한 데이터 수집에 취약해질 수 있습니다.

검증

•

호스트에서 **virsh dumpxml** 유틸리티를 사용하여 보안 **VM**의 **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](#)
- [genproting의 IBM 문서](#)

15.8. IBM Z의 가상 머신에 암호화 코프로세서 연결

IBM Z 호스트의 VM(가상 시스템)에서 하드웨어 암호화를 사용하려면 암호화 코프로세서 장치에서 중재된 장치를 생성하여 원하는 VM에 할당합니다. 자세한 지침은 아래를 참조하십시오.

사전 요구 사항

- 호스트가 IBM Z 하드웨어에서 실행되고 있습니다.
- 암호화 코프로세서는 장치 할당과 호환됩니다. 이를 확인하려면 귀하의 코프로세서 유형이 CEX4 이상으로 나열되어야 합니다.

```
# 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 설치되어 있습니다.
- `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 암호화 장치를 **vfio-ap** 드라이버에 다시 할당합니다. 다음 예제에서는 비트마스크 ID (0x05, 0x0004) 및 (0x05, 0x00ab) 가 있는 두 개의 암호화 장치를 **vfio-ap** 에 할당합니다.

```
# echo -0x05 > /sys/bus/ap/apmask
# echo -0x0004, -0x00ab > /sys/bus/ap/aqmask
```

비트마스크 ID 값 식별에 대한 자세한 내용은 IBM의 KVM 가상 서버 관리 문서의 [암호화 어댑터 리소스용 패스스루 장치 준비](#)를 참조하십시오.

2.

crypto 장치가 올바르게 다시 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 큐의 **DRIVER** 값이 **vfio_ap** 으로 변경된 경우 재할당에 성공했습니다.

3.

장치 **UUID** 생성.

```
# uuidgen
669d9b23-fe1b-4ecb-be08-a2fabca99b71
```

이 절차의 다음 단계에서는 669d9b23-fe1b-4ecb-be08-a2fabca99b71 을 생성된 **UUID**로 바꿉니다.

4.

UUID를 사용하여 새 **vfio_ap** 장치를 만듭니다.

다음 예제에서는 영구 조정 장치 생성 및 대기열 할당을 보여줍니다. 예를 들어 다음 명령은

도메인 어댑터 0x05 및 도메인 대기열 0x0004 및 0x00ab 장치를 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.

조정된 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 큐의 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 **<devices>** 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 **UUID**는 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.

조정된 장치를 할당한 VM을 시작합니다.

2.

게스트 운영 체제(OS)가 부팅된 후 할당된 암호화 장치를 탐지하는지 확인합니다.

```
# lszcrypt -V
```

CARD.DOMAIN	TYPE	MODE	STATUS	REQUESTS	PENDING	HWTYPE	QDEPTH
05	CEX5C	CCA-Coproc	online	1	0	11	08 S--D--N--
05.0004	CEX5C	CCA-Coproc	online	1	0	11	08 S--D--N--
05.00ab	CEX5C	CCA-Coproc	online	1	0	11	08 S--D--N--

```
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 동일한 암호화 코프로세서 장치가 있는 호스트 논리 파티션에서와 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 .....
-----
```

lszscrypt -d C 가 암호화 장치 매트릭스에 **B** 교집합을 표시하는 경우 컨트롤 도메인 할당에 성공했습니다.

15.9. WINDOWS 가상 머신에서 표준 하드웨어 보안 활성화

Windows VM(가상 머신)을 보호하려면 **Windows** 장치의 표준 하드웨어 기능을 사용하여 기본 수준의 보안을 활성화할 수 있습니다.

사전 요구 사항

- 최신 **WHQL** 인증 **VirtIO** 드라이버를 설치해야 합니다.
- **VM**의 펌웨어가 **UEFI** 부팅을 지원하는지 확인합니다.
- 호스트 시스템에 **the edk2-OVMF** 패키지를 설치합니다.

```
# yum install edk2-ovmf
```

- 호스트 시스템에 **vTPM** 패키지를 설치합니다.

```
# yum install swtpm libtpms
```

- **VM**에서 **Q35** 시스템 아키텍처를 사용하고 있는지 확인합니다.
- **Windows** 설치 미디어가 있는지 확인합니다.

절차

1. **VM**의 **XML** 구성의 **<devices>** 섹션에 다음 매개 변수를 추가하여 **TPM 2.0**을 활성화합니다.

```
<devices>
[...]
```

```
<tpm model='tpm-crb'>
  <backend type='emulator' version='2.0'>
</tpm>
[...]
</devices>
```

2.

UEFI 모드에서 **Windows**를 설치합니다. 이를 수행하는 방법에 대한 자세한 내용은 [SecureBoot 가상 머신 생성](#)을 참조하십시오.

3.

Windows VM에 **VirtIO** 드라이버를 설치합니다. 이를 수행하는 방법에 대한 자세한 내용은 [Windows 게스트에 virtio 드라이버 설치](#)를 참조하십시오.

4.

UEFI에서 **Secure Boot**를 활성화합니다. 이 작업을 수행하는 방법에 대한 자세한 내용은 [Secure Boot](#)를 참조하십시오.

검증

•

Windows 머신의 **Device Security** (장치 보안) 페이지에 다음 메시지가 표시되는지 확인합니다.

설정 > 업데이트 및 보안 > **Windows** 보안 > 장치 보안

Your device meets the requirements for standard hardware security.

15.10. WINDOWS 가상 머신에서 향상된 하드웨어 보안 활성화

Windows VM(가상 머신)의 보안을 강화하기 위해 **HVCI**(Hypervisor-Protected Code Integrity)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 VM의 XML 구성을 엽니다. 다음 예제에서는 **Example-L1 VM**의 구성을 엽니다.

```
# virsh edit Example-L1
```

2. `<cpu>` 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 > 코어 격리 세부 정보
5. 스위치를 토글하여 메모리 무결성을 활성화합니다.
6. VM을 재부팅합니다.



참고

HVCI를 활성화하는 다른 방법은 관련 **Microsoft** 문서를 참조하십시오.

검증

- **Windows VM의 Device Security (장치 보안) 페이지가 다음 메시지를 표시하는지 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 Guard, Hypervisor Enforced Code Integrity** 가 가상화 기반 보안 서비스 **Running** 아래에 나열되는지 확인합니다.

16장. 가상 머신 성능 최적화

VM(가상 시스템)은 항상 호스트와 비교하여 일정 수준의 성능 저하를 경험합니다. 다음 섹션에서는 이러한 문제가 발생하는 이유를 설명하고, 하드웨어 인프라 리소스를 최대한 효율적으로 사용할 수 있도록 **RHEL 8**의 가상화 성능 영향을 최소화하는 방법에 대한 지침을 제공합니다.

16.1. 가상 머신 성능에 어떤 영향을 주는가

VM은 호스트에서 사용자 공간 프로세스로 실행됩니다. 따라서 하이퍼바이저는 **VM**에서 사용할 수 있도록 호스트의 시스템 리소스를 변환해야 합니다. 결과적으로 변환에서 리소스의 일부를 소비하므로 **VM**은 호스트와 동일한 성능 효율성을 달성할 수 없습니다.

시스템 성능에 가상화가 미치는 영향

VM 성능 손실의 구체적인 이유는 다음과 같습니다.

- 가상 **CPU(vCPU)**는 **Linux** 스케줄러에서 처리하는 호스트의 스레드로 구현됩니다.
- **VM**은 호스트 커널에서 **NUMA** 또는 대규모 페이지와 같은 최적화 기능을 자동으로 상속하지 않습니다.
- 호스트의 디스크 및 네트워크 **I/O** 설정은 **VM**에 상당한 성능 영향을 줄 수 있습니다.
- 네트워크 트래픽은 일반적으로 소프트웨어 기반 브릿지를 통해 **VM**으로 이동합니다.
- 호스트 장치 및 해당 모델에 따라 특정 하드웨어의 에뮬레이션으로 인해 상당한 오버헤드가 발생할 수 있습니다.

VM 성능에 대한 가상화 영향의 심각도는 다음과 같은 다양한 요인의 영향을 받습니다.

- 동시에 실행 중인 **VM** 수입니다.
- 각 **VM**에서 사용하는 가상 장치의 양입니다.

- VM에서 사용하는 장치 유형입니다.

VM 성능 손실 감소

RHEL 8에서는 가상화의 부정적인 성능 효과를 줄이는 데 사용할 수 있는 여러 가지 기능을 제공합니다. 특히:

- **TuneD 서비스**는 VM의 리소스 배포 및 성능을 자동으로 최적화할 수 있습니다.
- **블록 I/O 튜닝**을 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 마이그레이션하기가 더 어려워질 수 있습니다.

16.2. 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 프로파일 전달 메커니즘입니다. 다수의 특정 사용 사례에서 성능을 향상시키고 전력 소비를 줄이기 위해 사전 구성된 여러 튜닝 프로파일을 제공합니다. 이러한 프로파일을 편집하거나 새 프로파일을 생성하여 가상화 환경을 포함한 환경에 맞는 성능 솔루션을 만들 수 있습니다.

가상화를 위해 RHEL 8을 최적화하려면 다음 프로파일을 사용하십시오.

- RHEL 8 가상 시스템의 경우 **virtual-guest** 프로파일을 사용합니다. 일반적으로 적용 가능한 **throughput-performance** 프로파일을 기반으로 하지만 가상 메모리의 스왑성도 줄어듭니다.
- RHEL 8 가상화 호스트의 경우 **virtual-host** 프로파일을 사용합니다. 이렇게 하면 더티 메모리 페이지의 보다 적극적인 쓰기가 가능하여 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

16.3. 가상 머신 메모리 구성

VM(가상 시스템)의 성능을 개선하기 위해 **VM**에 추가 호스트 **RAM**을 할당할 수 있습니다. 마찬가지로 호스트 메모리를 다른 **VM** 또는 작업에 할당할 수 있도록 **VM**에 할당된 메모리 양을 줄일 수 있습니다.

이러한 작업을 수행하려면 [웹 콘솔](#) 또는 [명령줄 인터페이스](#)를 사용할 수 있습니다.

16.3.1. 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. 이러한 경우인지 확인하려면 다음을 수행합니다.

- 1.

VM 구성에 **memballoon** 장치가 포함되어 있는지 확인합니다.

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

이 명령은 출력을 표시하며 모델을 **none** 으로 설정하지 않으면 **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 드라이버가 기본적으로 포함되며 **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. 그러면 변경 사항과 확인을 위한 기준선으로 사용할 수 있습니다.

```
# 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.

Memory Adjustment(메모리 조정) 대화 상자가 표시됩니다.

The dialog box titled "Grid_v2 memory adjustment" contains two sliders. The first slider, "Current allocation", ranges from 128 to 3072, with a blue dot at 3072. The second slider, "Maximum allocation", ranges from 128 to 15626, with a blue dot at 3072. To the right of each slider is an input field showing the current value (3072) and a unit dropdown menu set to "MiB". At the bottom left are "Save" and "Cancel" buttons.

4. 선택한 VM에 대해 가상 CPU를 구성합니다.

- **max allocation** (최대 할당) - VM이 프로세스에 사용할 수 있는 최대 호스트 메모리 양을 설정합니다. VM을 생성할 때 최대 메모리를 지정하거나 나중에 늘릴 수 있습니다. 메모리를 **MiB** 또는 **GiB**의 배수로 지정할 수 있습니다.

메모리 할당 최대 조정은 종료 VM에서만 가능합니다.

- current allocation** (현재 할당) - VM에 할당된 실제 메모리 양을 설정합니다. 이 값은 최대 할당보다 작을 수 있지만 초과할 수는 없습니다. 해당 프로세스에 대해 VM에서 사용할 수 있는 메모리를 제어하도록 값을 조정할 수 있습니다. 메모리를 **MiB** 또는 **GiB**의 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](#)

16.3.2. 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. 이러한 경우인지 확인하려면 다음을 수행합니다.

1.

VM 구성에 **memballoon** 장치가 포함되어 있는지 확인합니다.

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
</memballoon>
```

이 명령은 출력을 표시하며 모델을 **none** 으로 설정하지 않으면 **memballoon** 장치가 있습니다.

2.

balloon 드라이버가 게스트 OS에서 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 드라이버가 기본적으로 포함되며 **memballoon** 장치가 있는 경우 활성화됩니다.

절차

1.

선택 사항: 최대 메모리와 현재 VM에 사용된 메모리에 대한 정보를 가져옵니다. 그러면 변경 사항과 확인을 위한 기준으로 사용할 수 있습니다.

```
# virsh dominfo testguest
Max memory: 2097152 KiB
Used memory: 2097152 KiB
```

2.

VM에 할당된 최대 메모리를 조정합니다. 이 값을 늘리면 VM의 성능 잠재력이 향상되고 해당 값을 줄이면 VM이 호스트에 있는 성능 공간을 줄일 수 있습니다. 이러한 변경 사항은 종료 VM에서만 수행할 수 있으므로 실행 중인 VM을 조정하려면 재부팅을 적용해야 합니다.

예를 들어 **testguest** VM에서 사용할 수 있는 최대 메모리를 **4096MiB**로 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 메모리 수를 늘리려면 메모리 장치를 VM에 연결할 수 있습니다. 이를 메모리 핫 플러그 라고도 합니다. 자세한 내용은 [가상 머신에 장치 연결](#)을 참조하십시오.



주의

실행 중인 VM에서 메모리 장치 제거(메모리 핫 플러그라고도 함)은 지원되지 않으며 **Red Hat**에서 사용하지 않는 경우가 많습니다.

3.

선택 사항: VM에서 현재 사용 중인 메모리를 최대 할당까지 조정할 수도 있습니다. 이렇게 하면 최대 VM 할당을 변경하지 않고 VM이 호스트에 있는 메모리 부하를 다음 번 재부팅할 때까지 조정합니다.

```
# 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 증대 통계를 가져와 메모리 사용을 효과적으로 규제하는 방법을 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](#)

16.3.3. 추가 리소스

•

가상 머신에 장치 연결.

16.4. 가상 머신 I/O 성능 최적화

VM(가상 시스템)의 입력 및 출력(I/O) 기능은 VM의 전반적인 효율성을 크게 제한할 수 있습니다. 이 문제를 해결하기 위해 블록 I/O 매개변수를 구성하여 VM의 I/O를 최적화할 수 있습니다.

16.4.1. 가상 머신의 블록 I/O 튜닝

하나 이상의 VM에서 여러 블록 장치를 사용하는 경우 I/O 가중치를 수정하여 특정 가상 장치의 I/O 우선 순위를 조정하는 것이 중요할 수 있습니다.

장치의 I/O 가중치를 늘리면 I/O 대역폭의 우선 순위가 증가하므로 더 많은 호스트 리소스를 제공합니다. 마찬가지로 장치의 가중치를 줄이면 호스트 리소스를 더 적게 소비할 수 있습니다.



참고

각 장치의 가중치 값은 100~1000 범위 내에 있어야 합니다. 또는 장치별 목록에서 해당 장치를 제거하는 값이 0 일 수 있습니다.

절차

VM의 블록 I/O 매개변수를 표시하고 설정하려면 다음을 수행합니다.

1. VM의 현재 **<blkio>** 매개변수를 표시합니다.

```
# virsh dumpxml VM-name
```

```
<domain>
[...]
<blkio>
  <weight>800</weight>
  <device>
    <path>/dev/sda</path>
    <weight>1000</weight>
  </device>
  <device>
    <path>/dev/sdb</path>
    <weight>500</weight>
```

```

    </device>
  </blkio tune>
  [...]
</domain>

```

2.

지정된 장치의 I/O 가중치를 편집합니다.

```
# virsh blkio tune VM-name --device-weights device, I/O-weight
```

예를 들어 다음에서는 **riserul VM**의 **/dev/sda** 장치의 가중치를 **500**으로 변경합니다.

```
# virsh blkio tune liftbrul --device-weights /dev/sda, 500
```

16.4.2. 가상 머신의 디스크 I/O 제한

여러 개의 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 제한을 설정하는 기능을 제공합니다. 이렇게 하면 VM이 공유 리소스를 과도하게 활용하고 다른 VM의 성능에 영향을 미치지 않도록 할 수 있습니다.

디스크 I/O 제한을 활성화하려면 VM에 연결된 각 블록 장치에서 호스트 시스템으로 전송된 디스크 I/O 요청에 대한 제한을 설정합니다.

절차

1.

virsh domblklist 명령을 사용하여 지정된 VM의 모든 디스크 장치 이름을 나열합니다.

```
# virsh domblklist rollin-coal
Target    Source
-----
vda       /var/lib/libvirt/images/rollin-coal.qcow2
sda       -
sdb       /home/horridly-demanding-processes.iso
```

2.

제한할 가상 디스크가 마운트된 호스트 블록 장치를 찾습니다.

예를 들어 이전 단계에서 **sdb** 가상 디스크를 제한하려면 다음 출력에서 디스크가 **/dev/nvme0n1p3** 파티션에 마운트되어 있음을 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 장치의 I/O 제한을 설정합니다.

```
# virsh blkiotune VM-name --parameter device,limit
```

다음 예제에서는 **rollin-coal** VM의 **sdb** 디스크를 초당 읽기 및 쓰기 I/O 작업 수가 1000개, 초당 50MB의 읽기 및 쓰기 처리량으로 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 고객에게 속한 VM이 동일한 호스트에서 실행 중이거나 다른 VM에 대해 서비스 품질이 제공되는 경우와 같이 다양한 상황에서 유용할 수 있습니다. 디스크 I/O 제한도 느린 디스크를 시뮬레이션하는 데 사용할 수 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 제한 구성을 지원하지 않습니다. RHEL 8을 VM 호스트로 사용할 때 지원되지 않는 기능에 대한 자세한 내용은 [RHEL 8 가상화의 지원되지 않는 기능](#)을 참조하십시오.

16.4.3. 다중 대기열 virtio-scsi 활성화

VM(가상 시스템)에서 **virtio-scsi** 스토리지 장치를 사용할 때 **다중 대기열 virtio-scsi** 기능을 사용하면 스토리지 성능과 확장성이 향상됩니다. 이를 통해 각 가상 CPU(vCPU)에 다른 vCPU에 영향을 주지 않고 사용할 별도의 대기열과 인터럽트가 있을 수 있습니다.

절차

- 특정 VM에 대한 다중 대기열 **virtio-scsi** 지원을 활성화하려면 VM의 XML 구성에 다음을 추가합니다. 여기서 **N**은 총 vCPU 대기열 수입니다.

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

16.5. 가상 머신 CPU 성능 최적화

호스트 시스템의 물리적 CPU와 마찬가지로 vCPU는 VM(가상 시스템) 성능에 매우 중요합니다. 결과적으로 vCPU를 최적화하면 VM의 리소스 효율성에 큰 영향을 미칠 수 있습니다. vCPU를 최적화하려면 다음을 수행합니다.

1. VM에 할당된 호스트 CPU 수를 조정합니다. CLI 또는 웹 콘솔을 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. KSM(커널 동일 페이지 병합)을 비활성화합니다.
4. 호스트 시스템에서 NUMA(Non-Uniform Memory Access)를 사용하는 경우 해당 VM에 NUMA를 구성할 수도 있습니다. 이를 통해 호스트의 CPU 및 메모리 프로세스를 가능한 VM의 CPU 및 메모리 프로세스에 긴밀하게 매핑합니다. 실제로 NUMA 튜닝은 VM에 할당된 시스템 메모리에 대한 보다 능률적인 액세스를 제공하므로 vCPU 처리 효율성이 향상될 수 있습니다.

자세한 내용은 [가상 머신에서 NUMA 구성 및 샘플 vCPU 성능 튜닝 시나리오](#)를 참조하십시오.

16.5.1. 명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거

VM(가상 머신)의 CPU 성능을 늘리거나 최적화하려면 VM에 할당된 가상 CPU(vCPU)를 추가하거나 제거할 수 있습니다.

실행 중인 VM에서 수행되는 경우 vCPU 핫 플러그 및 핫 플러그 해제라고도 합니다. 그러나 vCPU 핫 언플러그는 RHEL 8에서 지원되지 않으며 Red Hat은 사용하지 않습니다.

사전 요구 사항

•

선택 사항: 대상 VM에서 vCPU의 현재 상태를 확인합니다. 예를 들어 *testquest* VM의 vCPU 수를 표시하려면 다음을 수행합니다.

```
# virsh vcpucount testquest
maximum    config    4
maximum    live      2
current     config    2
current     live      1
```

이 출력은 *testquest* 가 현재 1 vCPU를 사용 중이며, VM의 성능을 높이기 위해 1개의 vCPU를 핫 플러그로 연결할 수 있음을 나타냅니다. 그러나 재부팅 후 *testquest* 에서 사용하는 vCPU 수는 2개로 변경되며 2개의 vCPU를 핫 플러그로 연결할 수 있습니다.

절차

1.

VM에 연결할 수 있는 최대 vCPU 수를 조정하여 다음 부팅 시 적용됩니다.

예를 들어 *testquest* VM의 최대 vCPU 수를 8로 늘리려면 다음을 수행합니다.

```
# virsh setvcpus testquest 8 --maximum --config
```

CPU 토폴로지, 호스트 하드웨어, 하이퍼바이저 및 기타 요인에 의해 최대값이 제한될 수 있습니다.

2.

VM에 연결된 현재 vCPU 수를 이전 단계에서 구성한 최대값까지 조정합니다. 예를 들면 다음과 같습니다.

•

실행 중인 *testquest* VM에 연결된 vCPU 수를 4로 늘리려면 다음을 수행합니다.

```
# virsh setvcpus testquest 4 --live
```

이로 인해 VM의 다음 부팅까지 VM의 성능 및 호스트 부하 공간이 *testquest* 의 호스트 부하가 증가합니다.

•

testquest VM에 연결된 vCPU 수를 영구적으로 1로 줄입니다.

```
# virsh setvcpus testquest 1 --config
```

이를 통해 VM의 다음 부팅 후 *testguest*의 VM 성능 및 호스트 부하 공간이 줄어듭니다. 그러나 필요한 경우 VM에 추가 vCPU를 핫플러그하여 일시적으로 성능을 향상시킬 수 있습니다.

검증

- VM의 현재 vCPU 상태에 변경 사항이 반영되는지 확인합니다.

```
# virsh vcpucount testguest
maximum    config    8
maximum    live      4
current     config    1
current     live      4
```

추가 리소스

- [웹 콘솔을 사용하여 가상 CPU 관리](#)

16.5.2. 웹 콘솔을 사용하여 가상 CPU 관리

RHEL 8 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.

Grid_v2 vCPU details
✕

vCPU count ⓘ	2	Sockets ⓘ	1
vCPU maximum ⓘ	2	Cores per socket	1
		Threads per core	1

Apply
Cancel

1.

선택한 VM에 대해 가상 CPU를 구성합니다.

•

vCPU 개수 - 현재 사용 중인 vCPU 수입니다.



참고

vCPU 수는 **vCPU 최대**보다 클 수 없습니다.

•

vCPU 최대 - VM에 대해 구성할 수 있는 최대 가상 CPU 수입니다. 이 값이 vCPU 개수보다 크면 VM에 추가 vCPU를 연결할 수 있습니다.

•

sockets - VM에 표시할 소켓 수입니다.

•

Cores per socket (소켓당 코어 수) - 각 소켓이 VM에 노출될 코어 수입니다.

•

코어당 스레드 - 각 코어가 VM에 노출될 스레드 수입니다.

소켓, 소켓당 코어, 코어 옵션당 스레드는 VM의 CPU 토폴로지를 조정합니다. 이는 vCPU 성능에 도움이 될 수 있으며 게스트 OS의 특정 소프트웨어의 기능에 영향을 줄 수 있습니다. 배포에서 다른 설정이 필요하지 않은 경우 기본값을 유지합니다.

2.

Apply(적용)를 클릭합니다.

VM의 가상 CPU가 구성됩니다.



참고

가상 CPU 설정 변경 사항은 VM을 다시 시작한 후에만 적용됩니다.

추가 리소스

- [명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거](#)

16.5.3. 가상 머신에서 NUMA 구성

다음 방법을 사용하여 RHEL 8 호스트에서 가상 머신(VM)의 NUMA(Non-Uniform Memory Access) 설정을 구성할 수 있습니다.

사전 요구 사항

- 호스트는 NUMA 호환 시스템입니다. 이 경우 `virsh nodeinfo` 명령을 사용하고 NUMA 셀 행을 참조하십시오.

```
# 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
```

행의 값이 2 이상이면 호스트는 NUMA와 호환됩니다.

절차

사용하기 쉽도록 자동화된 유틸리티 및 서비스를 사용하여 VM의 NUMA 구성을 설정할 수 있습니다. 그러나 수동 NUMA 설정으로 성능이 크게 향상될 가능성이 높습니다.

자동 방법

- VM의 NUMA 정책을 Preferred 로 설정합니다. 예를 들어 `testquest5` VM에서 이를 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 CPU**를 메모리 리소스와 자동으로 정렬합니다.

```
# numad
```

수동 방법

1. 특정 **vCPU** 스레드를 특정 호스트 **CPU** 또는 **CPU** 범위에 고정합니다. 이는 **NUMA**가 아닌 호스트와 **VM**에서도 사용할 수 있으며 **vCPU** 성능을 안전하게 개선하는 방법으로 권장됩니다.

예를 들어 다음 명령은 각각 **CPU 1, 3, 5, 7, 9, 11**을 호스팅하도록 **testguest6 VM**의 **0~5 vCPU** 스레드를 고정합니다.

```
# 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
```

이후에 이 작업이 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 **VM**과 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 **15**에 고정하고 성공했는지 확인합니다.

```
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
-----
*: 13,15
```

3.

마지막으로 특정 VM에 특히 할당될 호스트 NUMA 노드를 지정할 수도 있습니다. 그러면 VM vCPU에 의해 호스트 메모리 사용량이 향상될 수 있습니다. 예를 들어 다음 명령은 호스트 NUMA 노드 3을 5로 사용하도록 *testguest6* 를 설정하고 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](#)

16.5.4. 샘플 vCPU 성능 튜닝 시나리오

최상의 vCPU 성능을 얻으려면 다음 시나리오와 같이 *manual vcpupin*, 에뮬레이터 *pin* 및 *numatune* 설정을 함께 사용하는 것이 좋습니다.

시나리오 시작

- 호스트에는 다음과 같은 하드웨어 세부 사항이 있습니다.
 - NUMA 노드 2개

- 각 노드의 **CPU 코어 3개**
- 각 코어에 두 개의 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 vCPU로 변경하려고 하므로 단일 NUMA 노드에 맞지 않습니다.

따라서 각 NUMA 노드에 4개의 vCPU를 배포하고 vCPU 토폴로지가 호스트 토폴로지와 최대한 긴밀하게 일치하도록 해야 합니다. 즉, 지정된 물리적 CPU의 스레드로 실행되는 vCPU는 동일한 코어의 호스트 스레드에 고정되어야 합니다. 자세한 내용은 다음 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 유틸리티**를 사용하여 **VM**의 성능을 테스트합니다.

3.

호스트에 **1GiB** 대규모 페이지를 설정하고 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
```

그러면 *node1*에서 **4GiB**의 대규모 페이지와 *node 2*에서 **4GiB**의 대규모 페이지가 예약됩니다.

d.

이전 단계에서 생성한 스크립트를 실행 파일로 만듭니다.

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

e.

부팅 시 대규모 페이지 예약을 활성화합니다.

```
# systemctl enable hugetlb-gigantic-pages
```

4.

`virsh edit` 명령을 사용하여 최적화하려는 VM의 XML 구성을 편집합니다. 이 예제 *super-VM*:

```
# virsh edit super-vm
```

5.

다음과 같은 방식으로 VM의 XML 구성을 조정합니다.

a.

8개의 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>` 섹션에서 `<vcpupin/>` 요소를 사용합니다.

위의 **virsh** 기능 유틸리티에 표시된 대로 호스트 **CPU** 스레드는 해당 코어에서 순차적으로 정렬되지 않습니다. 또한 **vCPU** 스레드는 동일한 **NUMA** 노드에서 사용 가능한 최고 수준의 호스트 코어 세트에 고정되어야 합니다. 테이블 그림에서는 아래의 샘플 토폴로지 섹션을 참조하십시오.

a 및 b. 단계를 위한 **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 **VM**을 설정합니다.

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB' />
  </hugepages>
</memoryBacking>
```

d.

호스트의 해당 **NUMA** 노드의 메모리를 사용하도록 **VM**의 **NUMA** 노드를 구성합니다. 이렇게 하려면 `<numatune/>` 섹션에서 `<memnode />` 요소를 사용합니다.

```
<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 **CPU**가 **passthrough** 모드에서 캐시

를 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 유틸리티**를 사용하여 **VM**의 성능을 테스트하여 **VM** 최적화의 영향을 평가합니다.

샘플 토폴로지

•

다음 표에서는 고정해야 하는 **vCPU**와 호스트 **CPU** 간의 연결을 보여줍니다.

표 16.1. 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 16.2. 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 16.3. 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 시나리오에는 **2개의 NUMA 노드**와 **8개의 vCPU**가 있습니다. 따라서 **4개의 vCPU 스레드**를 각 노드에 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 있으므로 노드 0의 세트는 다음과 같이 변환됩니다.

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

16.5.5. 커널 동일 페이지 병합 비활성화

커널 동일 페이지 병합(**KSM**)은 메모리 밀도를 향상하지만 **CPU** 사용률을 높이며 워크로드에 따라 전반적인 성능에 부정적 영향을 미칠 수 있습니다. 이러한 경우 **KSM**을 비활성화하여 **VM**(가상 머신) 성능을 향상시킬 수 있습니다.

요구 사항에 따라 단일 세션에서 **KSM**을 비활성화하거나 영구적으로 설정할 수 있습니다.

절차

- 단일 세션에서 **KSM**을 비활성화하려면 **systemctl** 유틸리티를 사용하여 **kvm** 및 **ksmtuned** 서비스를 중지합니다.

```
# systemctl stop ksm
# systemctl stop ksmtuned
```

- **KSM**을 영구적으로 비활성화하려면 **systemctl** 유틸리티를 사용하여 **kvm** 및 **ksmtuned** 서비스를 비활성화합니다.

```
# systemctl disable ksm
Removed /etc/systemd/system/multi-user.target.wants/kvm.service.
# systemctl disable ksmtuned
Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
```

참고

KSM을 비활성화하기 전에 **VM** 간에 공유되는 메모리 페이지는 공유됩니다. 공유를 중지하려면 다음 명령을 사용하여 시스템의 모든 **PageKSM** 페이지를 삭제합니다.

```
# echo 2 > /sys/kernel/mm/ksm/run
```

익명 페이지가 **KSM** 페이지를 교체한 후 **khugepaged** 커널 서비스는 **VM**의 물리적 메모리에서 투명한 대규모 페이지를 다시 빌드합니다.

16.6. 가상 머신 네트워크 성능 최적화

VM의 **NIC**(네트워크 인터페이스 카드)의 가상 특성으로 인해 **VM**은 할당된 호스트 네트워크 대역폭의 일부를 손실하여 **VM**의 전체 워크로드 효율성을 줄일 수 있습니다. 다음 팁은 가상 **NIC(vNIC)** 처리량에 가상화의 부정적인 영향을 최소화할 수 있습니다.

절차

다음 방법을 사용하여 **VM** 네트워크 성능에 도움이 되는지 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
```

다중 대기열 virtio-net 설정

VM에 대한 다중 대기열 **virtio-net** 기능을 설정하려면 **virsh edit** 명령을 사용하여 **VM**의 **XML** 구성으로 편집합니다. **XML**에서 **<devices>** 섹션에 다음을 추가하고 **N**을 **VM**의 **vCPU** 수로, 최대 16까지 교체합니다.

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

```
<model type='virtio'/>
<driver name='vhost' queues='N'/>
</interface>
```

VM이 실행 중인 경우 변경 사항이 적용되도록 VM을 다시 시작하십시오.

네트워크 패킷 배치

긴 전송 경로가 있는 Linux VM 구성에서는 커널에 제출하기 전에 패킷을 일괄 처리하면 캐시 사용률이 향상될 수 있습니다. 패킷 일괄 처리를 설정하려면 호스트에서 다음 명령을 사용하고, VM에서 사용하는 네트워크 인터페이스 이름으로 **tap0** 을 바꿉니다.

```
# ethtool -C tap0 rx-frames 64
```

SR-IOV

호스트 NIC에서 SR-IOV를 지원하는 경우 vNIC에 SR-IOV 장치 할당을 사용합니다. 자세한 내용은 [SR-IOV 장치 관리](#)를 참조하십시오.

추가 리소스

- [가상 네트워킹 이해](#)

16.7. 가상 머신 성능 모니터링 툴

가장 많은 VM 리소스를 사용하는 항목과 VM 성능에 최적화가 필요한 측면을 식별하기 위해 일반적인 성능 진단 툴과 VM별 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 8** 호스트에서 root로 **top** 유틸리티 또는 시스템 모니터 애플리케이션을 사용하고 출력에서 **qemu** 및 **virt** 을 찾습니다. 그러면 VM에서 사용하는 호스트 시스템 리소스의 양을 확인할 수 있습니다.
 - 모니터링 도구가 **qemu** 또는 **virt** 프로세스에서 호스트 **CPU** 또는 메모리 용량의 대부분을 사용하는 것을 표시하는 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 체제에서 시스템에서 사용할 수 있는 성능 유틸리티 및 애플리케이션을 사용하여 가장 많은 시스템 리소스를 사용하는 프로세스를 평가합니다.
 - Linux 시스템에서는 **top** 유틸리티를 사용할 수 있습니다.
 - Windows 시스템에서 **Task Manager** 응용 프로그램을 사용할 수 있습니다.

perf kvm

perf 유틸리티를 사용하여 **RHEL 8** 호스트의 성능에 대한 가상화 관련 통계를 수집하고 분석할 수 있습니다. 이렇게 하려면 다음을 수행합니다.

1. 호스트에서 **perf** 패키지를 설치합니다.


```
# yum install perf
```
2. 가상화 호스트에 대한 **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 데이터는 **perf kvm stat report** 명령을 사용하여 분석할 수 있는 **perf.data.guest** 파일에 저장됩니다.
3. **VM-EXIT** 이벤트 유형 및 해당 배포에 대한 **perf** 출력을 분석합니다. 예를 들어, **PAUSE_INSTRUCTION** 이벤트는 자주 일치하지 않지만 다음 출력에서는 호스트 CPU가 실행 중인 vCPU를 잘 처리하지 않는 것으로 제안합니다. 이러한 시나리오에서는 활성 VM 일부 종료, 이러한 VM에서 vCPU 제거 또는 **vCPU의 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 신호를 보낼 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 numa ctl 패키지를 설치하여 제공되는 numastat 유틸리티를 사용할 수 있습니다.

다음은 각각 여러 NUMA 노드에서 메모리를 가져오는 4개의 실행 중인 VM이 있는 호스트를 보여줍니다. 이는 vCPU 성능에 적합하지 않으며 [조정 보증](#):

```
# 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
```

반면 다음은 단일 노드에서 각 VM에 제공되는 메모리를 보여줍니다. 이 메모리는 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
```

16.8. 추가 리소스

- [Windows 가상 머신 최적화](#)

17장. WINDOWS 가상 머신 설치 및 관리

RHEL 8 호스트의 **VM**(가상 머신)의 게스트 운영 체제로 **Microsoft Windows**를 사용하려면 이러한 **VM**이 올바르게 실행되도록 추가 단계를 수행하는 것이 좋습니다.

이 목적을 위해 다음 섹션에서는 호스트에 **Windows VM**을 설치 및 최적화하고 이러한 **VM**에 드라이버를 설치하고 구성하는 방법에 대한 정보를 제공합니다.

17.1. WINDOWS 가상 머신 설치

RHEL 8 호스트에서 완전히 가상화된 **Windows** 머신을 생성하고, **VM**(가상 머신) 내부에서 그래픽 **Windows** 설치 프로그램을 시작하고, 설치된 **Windows** 게스트 운영 체제(**OS**)를 최적화할 수 있습니다.

VM을 생성하고 **Windows** 게스트 **OS**를 설치하려면 **virt-install** 명령 또는 **RHEL 8** 웹 콘솔을 사용합니다.

사전 요구 사항

- 다음 중 하나일 수 있으며 로컬 또는 네트워크에서 사용할 수 있는 **Windows OS** 설치 소스:
 - 설치 미디어의 **ISO** 이미지
 - 기존 **VM** 설치의 디스크 이미지
 - **KVM VirtIO** 드라이버가 있는 스토리지 미디어.
- 이 미디어를 [생성하려면 호스트 시스템의 virtio 드라이버 설치 미디어 준비](#)를 참조하십시오.
- **Windows 11**을 설치하는 경우 **edk2-ovmf,swtpm** 및 **libtpms** 패키지를 호스트에 설치해야 합니다.

절차

1. **VM**을 생성합니다. 자세한 내용은 [가상 머신 생성](#)을 참조하십시오. 그러나 다음과 같은 세부 사항을 고려하십시오.

- **virt-install** 유틸리티를 사용하여 VM을 생성하는 경우 다음 옵션을 명령에 추가합니다.

- **KVM VirtIO** 드라이버가 있는 스토리지 미디어. 예를 들면 다음과 같습니다.

```
--disk path=/usr/share/virtio-win/virtio-win.iso,device=cdrom
```

- 설치할 **Windows** 버전입니다. 예를 들어 **Windows 10** 및 **11**의 경우:

```
--os-variant win10
```

사용 가능한 **Windows** 버전 및 적절한 옵션 목록의 경우 다음 명령을 사용합니다.

```
# osinfo-query os
```

- **Windows 11**을 설치하는 경우 **UEFI(Unified Extensible Firmware Interface)** 및 **vTPM(가상 신뢰할 수 있는 플랫폼 모듈)**을 활성화합니다.

```
--boot uefi --tpm model=tpm-crb,backend.type=emulator,backend.version=2.0
```

- 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 11**을 설치하거나 추가 **Windows Server 2022** 기능을 사용하려면 **CLI**를 사용하여 **UEFI** 및 **vTPM**을 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 OS**를 설치합니다.

Windows 운영 체제 설치 방법에 대한 자세한 내용은 관련 **Microsoft** 설치 설명서를 참조하십시오.

3.

웹 콘솔을 사용하여 **VM**을 생성하는 경우 **Disks** 인터페이스를 사용하여 **virtio** 드라이버가 있는 스토리지 미디어를 **VM**에 연결합니다. 자세한 내용은 [웹 콘솔을 사용하여 기존 디스크를 가상 머신에 연결](#)을 참조하십시오.

4.

Windows 게스트 OS에서 **KVM virtio** 드라이버 구성. 자세한 내용은 [Windows 가상 머신용 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](#)

17.2. WINDOWS 가상 머신 최적화

RHEL 8에서 호스팅되는 **VM(가상 머신)**에서 게스트 운영 체제로 **Microsoft Windows**를 사용하는 경우 게스트의 성능에 부정적인 영향을 미칠 수 있습니다.

따라서 다음을 조합하여 **Windows VM**을 최적화하는 것이 좋습니다.

- 반가상화 드라이버 사용. 자세한 내용은 [Windows 가상 머신 용 KVM 반가상화 드라이버 설치](#)를 참조하십시오.
- **Hyper-V** 구현 활성화. 자세한 내용은 [Enabling Hyper-V Enlightenments](#)를 참조하십시오.
- **NetKVM** 드라이버 매개 변수 구성. 자세한 내용은 [NetKVM 드라이버 매개변수 구성](#)을 참조하십시오.
- **Windows** 백그라운드 프로세스 최적화 또는 비활성화. 자세한 내용은 [Windows 가상 머신에서 백그라운드 프로세스 최적화](#)를 참조하십시오.

17.2.1. Windows 가상 머신용 KVM 반가상화 드라이버 설치

Windows VM(가상 시스템)의 성능을 향상시키는 주요 방법은 **OS(게스트 운영 체제)**에서 **Windows**용 **KVM 반가상화(virtio)**드라이버를 설치하는 것입니다.

이렇게 하려면 다음을 수행합니다.

1. 호스트 시스템에 설치 미디어를 준비합니다. 자세한 내용은 [호스트 시스템에서 virtio 드라이버 설치 미디어 준비](#)를 참조하십시오.
2. 설치 미디어를 기존 **Windows VM**에 연결하거나 새 **Windows VM**을 생성할 때 연결합니다.
3. **Windows** 게스트 **OS**에 **virtio** 드라이버를 설치합니다. 자세한 내용은 [Windows 게스트에 virtio 드라이버 설치](#)를 참조하십시오.

17.2.1.1. Windows virtio 드라이버 작동 방식

반가상화 드라이버는 I/O 대기 시간을 줄이고 거의 베어 메탈 수준으로 처리량을 늘려 VM(가상 머신)의 성능을 향상시킵니다. I/O가 많은 작업 및 애플리케이션을 실행하는 VM에 반가상화 드라이버를 사용하는 것이 좋습니다.

VirtIO 드라이버는 KVM 호스트에서 실행되는 Windows VM에 사용할 수 있는 KVM의 반가상화 장치 드라이버입니다. 이러한 드라이버는 다음을 위한 드라이버를 포함하는 **virtio-win** 패키지에서 제공합니다.

- 블록(스토리지) 장치
- 네트워크 인터페이스 컨트롤러
- 비디오 컨트롤러
- 메모리 증대 장치
- 반가상 직렬 포트 장치
- 엔트로피 소스 장치
- 반가상 패닉 장치
- 마우스, 키보드 또는 태블릿과 같은 입력 장치
- 작은 에뮬레이트된 장치 세트



참고

에뮬레이션된 **virtio** 및 할당된 장치에 대한 자세한 내용은 [가상 장치 관리](#)를 참조하십시오.

KVM virtio 드라이버를 사용하면 다음과 같은 **Microsoft Windows** 버전이 물리적 시스템과 유사하게 실행될 것으로 예상됩니다.

- **Windows Server 버전:** Red Hat Knowledgebase의 [KVM이 포함된 Red Hat Enterprise Linux용 공인 게스트 운영 체제](#)를 참조하십시오.
- **Windows 데스크탑 (서버 이외의) 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)**

17.2.1.2. 호스트 머신에서 virtio 드라이버 설치 미디어 준비

Windows VM(가상 머신)에 **KVM virtio** 드라이버를 설치하려면 먼저 호스트 머신에서 **virtio** 드라이버용 설치 미디어를 준비해야 합니다. 이를 위해 호스트 시스템에 **virtio-win** 패키지를 설치하고 VM의 스토리지로 제공하는 **.iso** 파일을 사용합니다.

사전 요구 사항

- **RHEL 8** 호스트 시스템에서 가상화가 활성화되어 있는지 확인합니다. 자세한 내용은 [가상화 활성화](#)를 참조하십시오.

절차

1. 드라이버 다운로드
 - a. [Red Hat Enterprise Linux 다운로드](#) 로 이동.
 - b.

시스템 아키텍처와 관련된 제품 변형 을 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**(최신 항목 다운로드)를 클릭합니다.

RPM 파일이 다운로드됩니다.

2.

다운로드 디렉터리에서 **virtio-win** 패키지를 설치합니다. 예를 들면 다음과 같습니다.

```
# yum 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 준비되어 있습니다. 여기에는 각 아키텍처 및 지원되는 **Windows** 버전에 대해 하나씩 디렉터리 의 드라이버 파일이 있는 **ISO** 파일과 드라이버 디렉터리가 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. 이를 수행하려면 다음 중 하나를 수행합니다.

-

새 **Windows VM**을 생성할 때 파일을 디스크로 사용합니다.

-

파일을 기존 **Windows VM**에 **CD-ROM**으로 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.**

17.2.1.3. Windows 게스트에 virtio 드라이버 설치

Windows 게스트 운영 체제(OS)에 **KVM virtio** 드라이버를 설치하려면 가상 머신(VM) 이상을 만들거나 **Windows** 게스트 OS에 드라이버를 설치할 때 드라이버가 포함된 스토리지 장치를 추가해야 합니다.

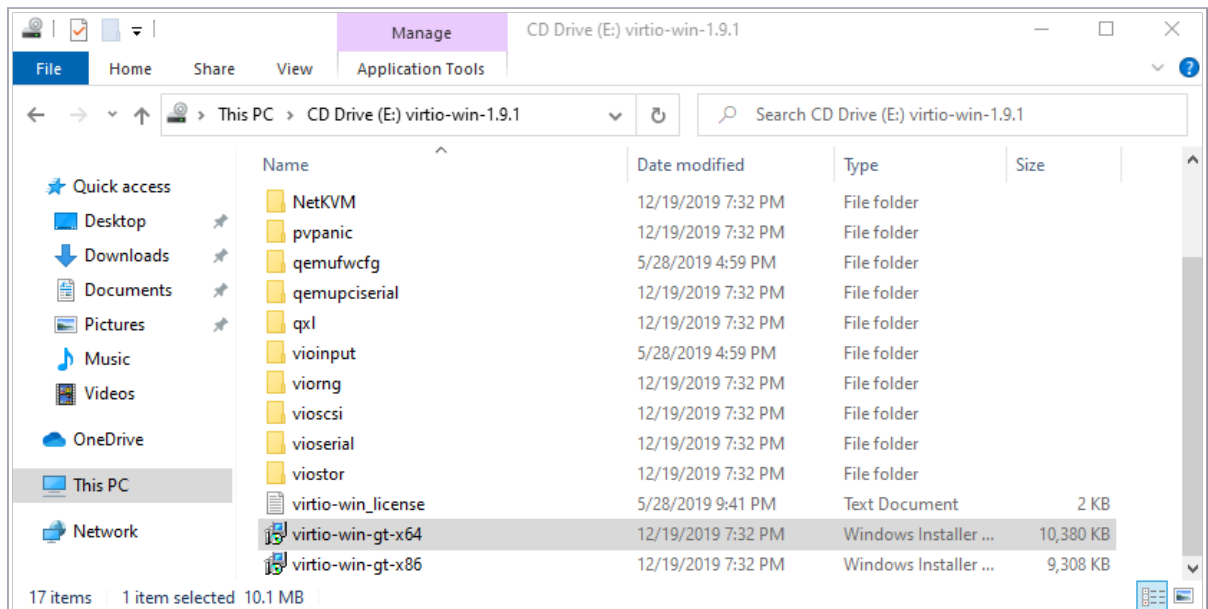
이 예에서는 그래픽 인터페이스를 사용하여 드라이버를 설치하는 방법을 보여줍니다. **Microsoft Windows Installer(MSI)** 명령줄 인터페이스도 사용할 수 있습니다.

사전 요구 사항

- **KVM virtio** 드라이버가 있는 설치 미디어를 VM에 연결해야 합니다. 중간 준비에 대한 지침은 **호스트 시스템의 virtio 드라이버 설치 미디어** 준비를 참조하십시오.

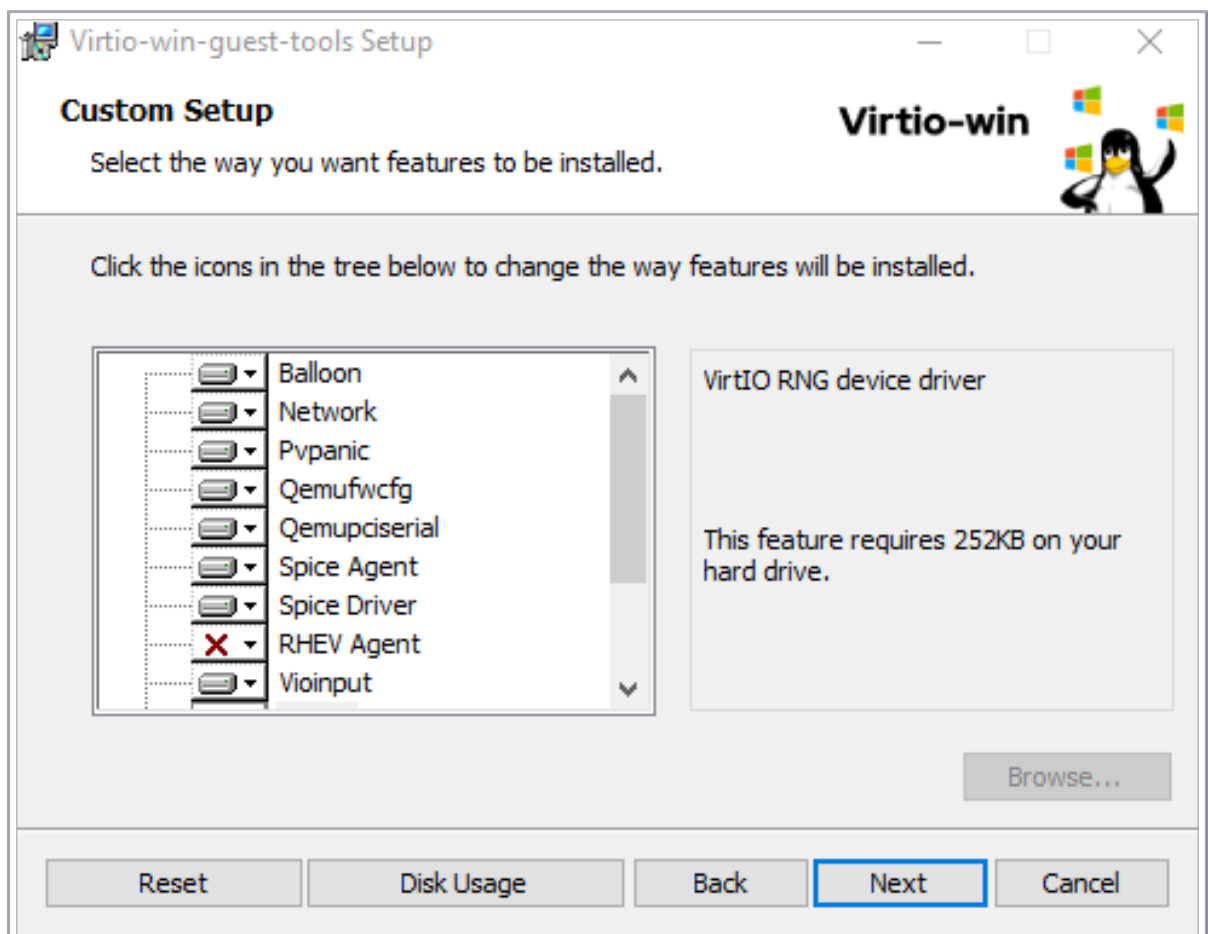
절차

1. **Windows** 게스트 OS에서 **File Explorer** 애플리케이션을 엽니다.
2. 이 **PC** 를 클릭합니다.
3. 장치 및 드라이브 창에서 **virtio-win** 미디어를 엽니다.
4. **VM vCPU**의 아키텍처에 따라 중간에서 설치 프로그램 중 하나를 실행합니다.
 - **32비트 vCPU**를 사용하는 경우 **virtio-win-gt-x86** 설치 프로그램을 실행합니다.
 - **64비트 vCPU**를 사용하는 경우 **virtio-win-gt-x64** 설치 프로그램을 실행합니다.



5.

열리는 **Virtio-win-guest-tools** 설정 마법사에서 **Custom Setup** 단계에 도달할 때까지 표시된 지침을 따릅니다.



6.

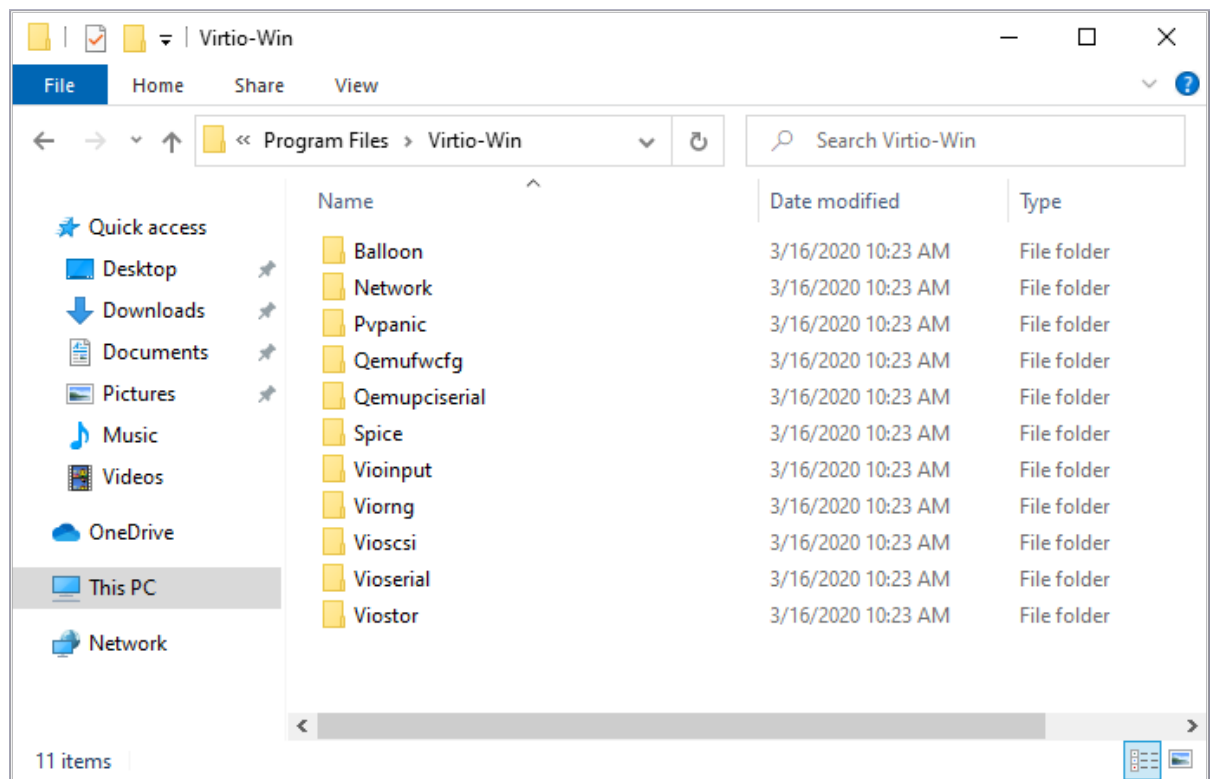
Custom Setup(사용자 지정 설정) 창에서 설치할 장치 드라이버를 선택합니다. 권장되는 드라이버 세트는 자동으로 선택되며, 드라이버의 설명이 목록 오른쪽에 표시됩니다.

7. 다음을 클릭한 다음 **Install(설치)**을 클릭합니다.
8. 설치가 완료되면 **Finish(완료)**를 클릭합니다 .
9. **VM**을 재부팅하여 드라이버 설치를 완료합니다.

검증

1. 이 **PC**에서 시스템 디스크를 엽니다. 일반적으로 **(C:)**입니다.
2. 프로그램 파일 디렉토리에서 **Virtio-Win** 디렉토리를 엽니다.

Virtio-Win 디렉토리가 존재하고 각 선택한 드라이버에 대한 하위 디렉터리가 포함된 경우 설치가 성공적으로 완료되었습니다.



다음 단계

- **NetKVM** 드라이버를 설치하는 경우 **Windows** 게스트의 네트워킹 매개 변수를 구성해야 할 수도 있습니다.

17.2.2. Hyper-V 구현 활성화

Hyper-V 구현은 KVM이 Microsoft Hyper-V 하이퍼바이저를 에뮬레이션할 수 있는 방법을 제공합니다. 이로 인해 Windows 가상 시스템의 성능이 향상됩니다.

다음 섹션에서는 지원되는 Hyper-V 구현 및 활성화 방법에 대한 정보를 제공합니다.

17.2.2.1. Windows 가상 머신에서 Hyper-V 구현 활성화

Hyper-V 시행은 RHEL 8 호스트에서 실행되는 Windows VM(가상 머신)에서 더 나은 성능을 제공합니다. 활성화 방법에 대한 지침은 다음을 참조하십시오.

절차

1. **virsh edit** 명령을 사용하여 VM의 XML 구성을 엽니다. 예를 들면 다음과 같습니다.

```
# virsh edit windows-vm
```

2. 다음 <hyperv> 하위 섹션을 XML의 <features> 섹션에 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' />
</hyperv>
[...]
```

```
</features>
```

XML에 이미 <hyperv> 하위 섹션이 포함된 경우 위와 같이 수정합니다.

3. 구성의 **clock** 섹션을 다음과 같이 변경합니다.

```
<clock offset='localtime'>
...
<timer name='hypervclock' present='yes' />
</clock>
```

4. **XML** 구성을 저장하고 종료합니다.

5. **VM**이 실행 중인 경우 다시 시작합니다.

검증

- **virsh dumpxml** 명령을 사용하여 실행 중인 **VM**의 **XML** 구성을 표시합니다. 다음 세그먼트가 포함된 경우 **VM**에서 **Hyper-V** 구현이 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'>
  <direct state='on' />
</stimer>
<frequencies state='on' />
</hyperv>

<clock offset='localtime'>
...
<timer name='hypervclock' present='yes' />
</clock>
```

17.2.2.2. 구성 가능한 **Hyper-V** 구현

특정 **Hyper-V** 기능을 구성하여 **Windows VM**을 최적화할 수 있습니다. 다음 표에서는 구성 가능한 **Hyper-V** 기능 및 해당 값에 대한 정보를 제공합니다.

표 17.1. 구성 가능한 **Hyper-V** 기능

Enlightenment	설명	값
---------------	----	---

Enlightenment	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.	켜짐, 꺼짐
evmcs	L0(KVM) 및 L1(Hyper-V) 하이퍼바이저 간에 반가상화 프로토콜을 구현하여 하이퍼바이저로 더 빨리 L2를 종료할 수 있습니다.  참고 이 기능은 Intel 프로세서에만 적용됩니다.	켜짐, 꺼짐
마케도니아	MSR(Hyper-V 빈도 Machine Specific Registers) 활성화.	켜짐, 꺼짐
ipi	반가상화 IPI(프로세서 간 인터럽트) 지원을 활성화합니다.	켜짐, 꺼짐
no-nonarch-coresharing	가상 프로세서가 SMT 스레드로 보고되지 않는 한 물리적 코어를 공유하지 않도록 게스트 OS에 알립니다. 이 정보는 동시 멀티스레딩(SMT) 관련 CPU 취약점을 적절하게 완화하기 위해 Windows 및 Hyper-V 게스트에 필요합니다.	켜짐, 끄기, 자동
Reenlightenment	마이그레이션 중에만 발생하는 타임스탬프 카운터(TSC) 빈도 변경이 있는 경우 알립니다. 또한 게스트가 새 빈도로 전환할 준비가 될 때까지 이전 빈도를 계속 사용할 수 있습니다.	켜짐, 꺼짐

Enlightenment	설명	값
완화됨	VM이 과도하게 로드된 호스트에서 실행 중일 때 일반적으로 발생하는 Windows 온전성 검사를 비활성화합니다. 이는 Linux 커널 옵션 <code>no_timer_check</code> 와 유사합니다. 이 옵션은 Linux가 KVM에서 실행될 때 자동으로 활성화됩니다.	켜짐, 꺼짐
runtime	게스트 코드 실행에 사용된 프로세서 시간을 게스트 코드를 대신 설정합니다.	켜짐, 꺼짐
spinlocks	- VM의 운영 체제에서 Hyper-V에 알릴 때 호출 가상 프로세서가 동일한 파티션 내의 다른 가상 프로세서가 보유하는 리소스를 확보하려고 한다고 알립니다. - Hyper-V가 사용하는 경우 가상 시스템의 운영 체제에 스핀 잠금이 시도되는 횟수를 나타낸 후 Hyper-V에 과도한 스핀 상황을 나타냅니다.	켜짐, 꺼짐
stimer	가상 프로세서의 컴퓨터 타이머 활성화. 이 구현이 제공되지 않는 경우 특정 Windows 버전은 HPET(또는 HPET을 사용할 수 없는 경우 RTC) 사용으로 되돌아갑니다. 이로 인해 가상 CPU가 유휴 상태인 경우에도 상당한 CPU 소비가 발생할 수 있습니다.	켜짐, 꺼짐
stimer-direct	만료 이벤트가 일반 인터럽트를 통해 전달되면 강제 타이머를 활성화합니다.	켜짐, 꺼짐.
Syncic	stimer와 함께, 온도 타이머를 활성화합니다. Windows 8은 이 기능을 주기 모드로 사용합니다.	켜짐, 꺼짐

Enlightenment	설명	값
time	VM에서 사용할 수 있는 다음과 같은 Hyper-V 관련 시계 소스를 활성화합니다. - MSR 기반 82 Hyper-V 클럭 소스 (HV_X64_MSR_TIME_REF_COUNT, 0x400000020) - MSR(HV_X64_MSR_REFERENCE_TSC, 0x400000021)을 통해 활성화되는 참조 TSC 83 페이지	켜짐, 꺼짐
tlbflush	가상 프로세서의 TLB를 플러시합니다.	켜짐, 꺼짐
vapic	사용량이 많은 메모리 매핑된 APIC(Advanced Programmable Interrupt Controller) 레지스터에 대한 MSR 액세스를 가속화하는 가상 APIC를 활성화합니다.	켜짐, 꺼짐
vendor_id	Hyper-V 벤더 ID 설정.	- 켜짐, 꺼짐 - ID 값 - 최대 12자의 문자열
vpindex	가상 프로세서 인덱스 활성화.	켜짐, 꺼짐

17.2.3. NetKVM 드라이버 매개변수 구성

NetKVM 드라이버를 설치한 후 환경에 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. 이 경우 기존 네트워크 활동이 중단됩니다.

사전 요구 사항

- **NetKVM** 드라이버는 가상 머신에 설치됩니다.

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. **Device Manager** 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 봅니다.

속성 창에서 고급 탭을 클릭합니다.**In the Properties window, click the Advanced tab.**
4. 장치 매개 변수를 수정합니다.
 - a. 수정할 매개변수를 클릭합니다.

해당 매개 변수에 대한 옵션이 표시됩니다.
 - b. 필요에 따라 옵션을 수정합니다.

NetKVM 매개변수 옵션에 대한 자세한 내용은 [NetKVM 드라이버 매개 변수](#) 를 참조하십시오.

c.

OK(확인)를 클릭하여 변경 사항을 저장합니다.

17.2.4. NetKVM 드라이버 매개변수

다음 표에서는 구성 가능한 **NetKVM** 드라이버 로깅 매개변수에 대한 정보를 제공합니다.

표 17.2. 로깅 매개변수

매개변수	설명 2
logging.Enable	로깅이 활성화되었는지 결정하는 부울 값입니다. 기본값은 Enabled입니다.
logging.Level	로깅 수준을 정의하는 정수입니다. 정수가 증가하면 로그의 상세 표시 여부도 마찬가지로 증가합니다. ● 기본값은 0(오류만)입니다. ● 1-2는 구성 메시지를 추가합니다. ● 4.5는 패킷 흐름 정보를 추가합니다. ● 5-6에는 인터럽트 및 DPC 수준 추적 정보가 추가됩니다.  참고 로깅 수준이 높으면 가상 머신 속도가 느려집니다.

다음 표에서는 구성 가능한 **NetKVM** 드라이버 초기 매개변수에 대한 정보를 제공합니다.

표 17.3. 초기 매개변수

매개변수	설명
MAC 할당	반가상화 NIC의 로컬에서 관리하는 MAC 주소를 정의하는 문자열입니다. 기본적으로 설정되지 않습니다.
Init.ConnectionRate(Mb)	초당 메가비트 단위의 연결 속도를 나타내는 정수입니다. Windows 2008 이상의 기본값은 10G(초당 메가비트 10,000개)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 65500의 값을 사용할 수 있습니다.
Init.MaxTxBuffers	할당할 TX 링 설명자 수를 나타내는 정수입니다. 기본값은 1024입니다. 유효한 값은 다음과 같습니다. 16, 32, 64, 128, 256, 512, 1024.
Init.MaxRxBuffers	할당할 RX 링 설명자 수를 나타내는 정수입니다. 기본값은 256입니다. 유효한 값은 다음과 같습니다. 16, 32, 64, 128, 256, 512, 1024.
Offload.Tx.Checksum	TX 체크섬 오프로딩 모드를 지정합니다. Red Hat Enterprise Linux 8에서 이 매개변수의 유효한 값은 다음과 같습니다. * IPv4 및 IPv6에 대해 IP, TCP 및 UDP 체크섬 오프로딩을 활성화하는 모두 (기본값) * IPv4 및 IPv6 모두에 대해 TCP 및 UDP 체크섬 오프로딩을 활성화하는 TCP/UDP(v4,v6) * IPv4에만 TCP 및 UDP 체크섬 오프로딩을 활성화하는 TCP/UDP(v4) * IPv4에서만 TCP 체크섬 오프로딩을 활성화하는 TCP(v4)

17.2.5. Windows 가상 머신에서 백그라운드 프로세스 최적화

Windows OS를 실행하는 **VM**(가상 머신)의 성능을 최적화하려면 다양한 **Windows** 프로세스를 구성하거나 비활성화할 수 있습니다.



주의

구성을 변경하면 특정 프로세스가 예상대로 작동하지 않을 수 있습니다.

절차

다음 조합을 수행하여 Windows VM을 최적화할 수 있습니다.

- 사용하지 않는 장치(예: USB 또는 CD-ROM)를 제거하고 포트를 비활성화합니다.
- SuperFetch 및 Windows Search와 같은 백그라운드 서비스를 비활성화합니다. 서비스를 중지하는 방법에 대한 자세한 내용은 [시스템 서비스 비활성화](#) 또는 [Stop-Service](#) 를 참조하십시오.
- useplatformclock 비활성화. 이를 수행하려면 다음 명령을 실행합니다.

bcdedit /set useplatformclock No
- 예약된 디스크 조각 모음과 같은 불필요한 예약된 작업을 검토하고 비활성화합니다. 이 수행 방법에 대한 자세한 내용은 [예약된 작업 비활성화](#)를 참조하십시오.
- 디스크가 암호화되지 않았는지 확인합니다.
- 서버 애플리케이션의 주기적인 활동을 줄입니다. 각 타이머를 편집하여 이를 수행할 수 있습니다. 자세한 내용은 [멀티미디어 타이머](#)를 참조하십시오.
- VM에서 Server Manager 애플리케이션을 종료합니다.
- 로켓 소프트웨어를 비활성화합니다. 로켓을 비활성화하면 VM의 보안이 손상될 수 있습니다.

- 화면 보호기를 비활성화합니다.
- 사용하지 않을 경우 **Windows OS**를 로그인 화면으로 유지합니다.

17.3. WINDOWS 가상 머신에서 표준 하드웨어 보안 활성화

Windows VM(가상 머신)을 보호하려면 **Windows** 장치의 표준 하드웨어 기능을 사용하여 기본 수준의 보안을 활성화할 수 있습니다.

사전 요구 사항

- 최신 **WHQL** 인증 **VirtIO** 드라이버를 설치해야 합니다.
- **VM**의 펌웨어가 **UEFI** 부팅을 지원하는지 확인합니다.
- 호스트 시스템에 **the edk2-OVMF** 패키지를 설치합니다.

```
# yum install edk2-ovmf
```

- 호스트 시스템에 **vTPM** 패키지를 설치합니다.

```
# yum install swtpm libtpms
```

- **VM**에서 **Q35** 시스템 아키텍처를 사용하고 있는지 확인합니다.
- **Windows** 설치 미디어가 있는지 확인합니다.

절차

1. **VM**의 **XML** 구성의 **<devices>** 섹션에 다음 매개 변수를 추가하여 **TPM 2.0**을 활성화합니다.

```
<devices>
[...]
<tpm model='tpm-crb'>
```

```
<backend type='emulator' version='2.0'/>
</tpm>
[...]
</devices>
```

2. UEFI 모드에서 **Windows**를 설치합니다. 이를 수행하는 방법에 대한 자세한 내용은 [SecureBoot 가상 머신 생성](#)을 참조하십시오.
3. **Windows VM**에 **VirtIO** 드라이버를 설치합니다. 이를 수행하는 방법에 대한 자세한 내용은 [Windows 게스트에 virtio 드라이버 설치](#)를 참조하십시오.
4. UEFI에서 **Secure Boot**를 활성화합니다. 이 작업을 수행하는 방법에 대한 자세한 내용은 [Secure Boot](#)를 참조하십시오.

검증

- **Windows** 머신의 **Device Security** (장치 보안) 페이지에 다음 메시지가 표시되는지 확인합니다.

설정 > 업데이트 및 보안 > **Windows** 보안 > 장치 보안

Your device meets the requirements for standard hardware security.

17.4. WINDOWS 가상 머신에서 향상된 하드웨어 보안 활성화

Windows VM(가상 머신)의 보안을 강화하기 위해 **HVCI**(Hypervisor-Protected Code Integrity)라고도 하는 코드 무결성의 가상화 기반 보호를 활성화할 수 있습니다.

사전 요구 사항

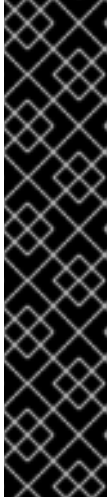
- 표준 하드웨어 보안이 활성화되어 있는지 확인합니다. 자세한 내용은 [Windows 가상 머신에서 표준 하드웨어 보안](#) 활성화를 참조하십시오.
- **Hyper-V** 시행을 활성화했는지 확인합니다. 자세한 내용은 [Enabling Hyper-V Enlightenments](#)를 참조하십시오.

절차

1. **Windows VM의 XML 구성을 엽니다.** 다음 예제에서는 **Example-L1 VM**의 구성을 엽니다.

```
# virsh edit Example-L1
```

2. **<cpu>** 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 > 코어 격리 세부 정보
5. 스위치를 토글하여 메모리 무결성 을 활성화합니다.
6. **VM을 재부팅합니다.**



참고

HVCI를 활성화하는 다른 방법은 관련 **Microsoft** 문서를 참조하십시오.

검증

- **Windows VM의 Device Security (장치 보안) 페이지가 다음 메시지를 표시하는지 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 Guard, Hypervisor Enforced Code Integrity** 가 가상화 기반 보안 서비스 **Running** 아래에 나열되는지 확인합니다.

17.5. 다음 단계

- **RHEL 8 호스트와 해당 Windows VM 간에 파일을 공유하려면 [Samba](#) 를 사용할 수 있습니다.**

18장. 중첩된 가상 머신 생성

RHEL 8 시스템 이상에서는 중첩된 **VM**(가상 머신)을 생성할 수 있습니다. 즉, **RHEL 8 VM**(레벨 1 또는 **L1**)에서 실행되는 **RHEL 8 VM**(레벨 0 또는 **L0**)은 하이퍼바이저 역할을 하며 자체 **VM**(레벨 2 또는 **L2**)을 생성할 수 있습니다.

즉, **L0** 호스트는 **L1 VM**(가상 시스템)을 실행할 수 있으며 각 **L1 VM**은 고유한 **L2 VM**을 호스팅할 수 있습니다. 이러한 경우 **L0** 및 **L1** 호스트는 **RHEL 8** 시스템이어야 하며 **L2** 게스트는 지원되는 모든 **RHEL** 또는 **Windows** 시스템일 수 있습니다.



주의

Red Hat은 현재 중첩된 가상화를 **기술 프리뷰**로만 제공하므로 지원되지 않습니다.

또한 다양한 **기능 제한**으로 인해 프로덕션 사용자 환경에서 중첩된 가상화를 사용하지 않는 것이 좋습니다. 대신 중첩된 가상화는 주로 다음과 같은 개발 및 테스트 시나리오를 위한 것입니다.

- 제한된 환경에서 하이퍼바이저 디버깅
- 제한된 양의 물리적 리소스에서 대규모 가상 배포 테스트

Intel, **AMD**, **IBM POWER9**, **IBM Z** 등 여러 아키텍처에서 중첩된 **VM**을 생성할 수도 있습니다. 사용된 아키텍처와 관계없이 중첩은 **기술 프리뷰** 이므로 **Red Hat**에서 지원하지 않습니다.

18.1. INTEL에서 중첩된 가상 머신 생성

다음 단계에 따라 **Intel** 호스트에서 중첩된 가상화를 활성화하고 구성합니다.



주의

중첩된 가상화는 현재 **Intel** 아키텍처에서 **기술 프리뷰**로만 제공되므로 지원되지 않습니다.

사전 요구 사항

- **L1 VM(가상 머신)**을 실행하는 **L0 RHEL 8 호스트**.
- 하이퍼바이저 **CPU**는 중첩된 가상화를 지원해야 합니다. 확인하려면 **L0** 하이퍼바이저에서 **cat /proc/cpuinfo** 명령을 사용합니다. 명령 출력에 **vmx** 및 **ept** 플래그가 포함된 경우 **L2 VM**을 생성할 수 있습니다. 일반적으로 이러한 사례는 **Intel Xeon v3** 코어 이상에서 이루어집니다.
- **L0 호스트**에서 중첩된 가상화가 활성화되어 있는지 확인합니다.

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

- 명령에서 **1** 또는 **Y**를 반환하면 기능이 활성화됩니다. 나머지 전제 조건 단계를 건너뛰고 **Procedure** 섹션을 계속합니다.
- 명령이 **0** 또는 **N**을 반환하지만 시스템에서 중첩된 가상화를 지원하는 경우 다음 단계를 사용하여 기능을 활성화합니다.
 - i. **kvm_intel** 모듈을 언로드합니다.


```
# modprobe -r kvm_intel
```
 - ii. 중첩 기능을 활성화합니다.


```
# modprobe kvm_intel nested=1
```
 - iii. 중첩 기능이 이제 활성화되지만 **L0 호스트**를 다음에 재부팅할 때까지만 중첩 기능이 활성화됩니다. 영구적으로 활성화하려면 **/etc/modprobe.d/kvm.conf** 파일에 다음 행

을 추가합니다.

```
options kvm_intel nested=1
```

절차

1. 중첩된 가상화를 위해 **L1 VM**을 구성합니다.

- a. **VM의 XML 구성**을 엽니다. 다음 예제에서는 **Intel-L1 VM**의 구성을 엽니다.

```
# virsh edit Intel-L1
```

- b. 구성에 다음 행을 추가합니다.

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

VM의 XML 구성 파일에 **<cpu>** 요소가 이미 포함되어 있는 경우 다시 작성합니다.

2. **L1 VM** 내에 **L2 VM** 생성. 이렇게 하려면 **L1 VM**을 **생성할** 때와 동일한 절차를 따릅니다.

18.2. AMD에 중첩 가상 머신 생성

AMD 호스트에서 중첩된 가상화를 활성화하고 구성하려면 아래 단계를 수행하십시오.



주의

중첩된 가상화는 현재 **AMD64** 아키텍처에서 **기술 프리뷰**로만 제공되므로 지원되지 않습니다.

사전 요구 사항

- **L1 VM(가상 머신)**을 실행하는 **L0 RHEL 8 호스트**.

- 하이퍼바이저 **CPU**는 중첩된 가상화를 지원해야 합니다. 확인하려면 **L0** 하이퍼바이저에서 **cat /proc/cpuinfo** 명령을 사용합니다. 명령 출력에 **svm** 및 **npt** 플래그가 포함된 경우 **L2 VM**을 생성할 수 있습니다. 일반적으로 **AMD EPYC** 코어 이상에서 사용되는 사례입니다.

- **L0** 호스트에서 중첩된 가상화가 활성화되어 있는지 확인합니다.

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

- 명령에서 **1** 또는 **Y**를 반환하면 기능이 활성화됩니다. 나머지 전제 조건 단계를 건너뛰고 **Procedure** 섹션을 계속합니다.

- 명령에서 **0** 또는 **N**을 반환하는 경우 다음 단계를 사용하여 기능을 활성화합니다.

- i. **L0** 호스트에서 실행 중인 모든 **VM**을 중지합니다.

- ii. **kvm_amd** 모듈을 언로드합니다.

```
# modprobe -r kvm_amd
```

- iii. 중첩 기능을 활성화합니다.

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

- iv. 중첩 기능이 이제 활성화되지만 **L0** 호스트를 다음에 재부팅할 때까지만 중첩 기능이 활성화됩니다. 영구적으로 활성화하려면 **/etc/modprobe.d/kvm.conf** 파일에 다음을 추가합니다.

```
options kvm_amd nested=1
```

절차

1. 중첩된 가상화를 위해 **L1 VM**을 구성합니다.
 - a. **VM**의 **XML** 구성을 엽니다. 다음 예제에서는 **AMD-L1 VM**의 구성을 엽니다.

virsh edit AMD-L1

b.

호스트 패스스루 모드를 사용하도록 VM의 CPU를 구성합니다.

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

host-passthrough 대신 특정 CPU를 사용해야 하는 경우 **<feature policy='require' name='vmx'/>** 행을 CPU 구성에 추가합니다. 예를 들면 다음과 같습니다.

```
<cpu mode='custom' match='exact' check='partial'>
<model fallback='allow'>Haswell-noTSX</model>
<feature policy='require' name='vmx'/>
```

2.

L1 VM 내에 **L2 VM** 생성. 이렇게 하려면 **L1 VM**을 생성할 때와 동일한 절차를 따릅니다.

18.3. IBM Z에 중첩 가상 머신 생성

아래 단계에 따라 **IBM Z** 호스트에서 중첩된 가상화를 활성화하고 구성합니다.

**주의**

중첩된 가상화는 현재 **IBM Z** 아키텍처에서 **기술 프리뷰**로만 제공되므로 지원되지 않습니다.

사전 요구 사항

- **L1 VM**(가상 머신)을 실행하는 **L0 RHEL 8** 호스트.
- 하이퍼바이저 CPU는 중첩된 가상화를 지원해야 합니다. 이 경우 **L0** 하이퍼바이저에서 **cat /proc/cpuinfo** 명령을 사용합니다. 명령 출력에 **the sie** 플래그가 포함된 경우 **L2 VM**을 생성할 수 있습니다.
- **L0** 호스트에서 중첩된 가상화가 활성화되어 있는지 확인합니다.

```
# cat /sys/module/kvm/parameters/nested
```

- 명령에서 1 또는 Y 를 반환하면 기능이 활성화됩니다. 나머지 전제 조건 단계를 건너뛰고 **Procedure** 섹션을 계속합니다.

- 명령에서 0 또는 N 을 반환하는 경우 다음 단계를 사용하여 기능을 활성화합니다.

- i. L0 호스트에서 실행 중인 모든 VM을 중지합니다.

- ii. kvm 모듈을 언로드합니다.

```
# modprobe -r kvm
```

- iii. 중첩 기능을 활성화합니다.

```
# modprobe kvm nested=1
```

- iv. 중첩 기능이 이제 활성화되지만 L0 호스트를 다음에 재부팅할 때까지만 중첩 기능이 활성화됩니다. 영구적으로 활성화하려면 `/etc/modprobe.d/kvm.conf` 파일에 다음 행을 추가합니다.

```
options kvm nested=1
```

절차

- L1 VM 내에 L2 VM 생성. 이렇게 하려면 **L1 VM을 생성할** 때와 동일한 절차를 따릅니다.

18.4. IBM POWER9에 중첩 가상 머신 생성

아래 단계에 따라 IBM POWER9 호스트에서 중첩된 가상화를 활성화하고 구성합니다.



주의

중첩된 가상화는 현재 **IBM POWER9** 아키텍처에서 **기술 프리뷰**로만 제공되므로 지원되지 않습니다. 또한 **IBM POWER8**과 같은 이전 버전의 **IBM POWER** 시스템에서 중첩된 **VM(가상 시스템)**을 생성할 수 없습니다.

사전 요구 사항

- **L0 RHEL 8** 호스트에서 **L1 VM**을 실행하고 있습니다. **L1 VM**은 **RHEL 8**을 게스트 운영 체제로 사용합니다.
- **L0** 호스트에서 중첩된 가상화가 활성화됩니다.

```
# cat /sys/module/kvm_hv/parameters/nested
```

- 명령에서 **1** 또는 **Y**를 반환하면 기능이 활성화됩니다. 나머지 전제 조건 단계를 건너뛰고 **Procedure** 섹션을 계속합니다.

- 명령에서 **0** 또는 **N**을 반환하는 경우 다음 단계를 사용하여 기능을 활성화합니다.

- i. **L0** 호스트에서 실행 중인 모든 **VM**을 중지합니다.

- ii. **kvm** 모듈을 인로드합니다.

```
# modprobe -r kvm_hv
```

- iii. 중첩 기능을 활성화합니다.

```
# modprobe kvm_hv nested=1
```

- iv. 중첩 기능이 이제 활성화되지만 **L0** 호스트를 다음에 재부팅할 때까지 중첩 기능이 활성화됩니다. 영구적으로 활성화하려면 **/etc/modprobe.d/kvm.conf** 파일에 다음 행을 추가합니다.

```
options kvm_hv nested=1
```

절차

1.

L1 VM에서 L2 VM을 생성할 수 있도록 하려면 **cap-nested-hv** 매개 변수를 L1 VM의 시스템 유형에 추가합니다. 이 작업을 수행하려면 **virsh edit** 명령을 사용하여 L1 VM의 XML 구성을 수정하고 다음 행을 <features> 섹션으로 수정합니다.

```
<nested-hv state='on'/'>
```

2.

L1 VM 내에 L2 VM 생성. 이렇게 하려면 **L1 VM을 생성할** 때와 동일한 절차를 따릅니다.

L2 VM의 성능을 크게 높이기 위해 Red Hat은 L2 VM의 XML 구성에 'cap-nested-hv' 매개 변수를 추가하는 것이 좋습니다. 자세한 내용은 이전 단계를 참조하십시오.

추가 정보

•

IBM POWER8을 L2 VM의 아키텍처로 사용하는 것은 현재 작동하지 않습니다.

18.5. 중첩된 가상화에 대한 제한 사항 및 제한 사항

중첩된 가상화를 사용할 때 다음과 같은 제한 사항을 염두에 두십시오.



주의

Red Hat은 현재 중첩된 가상화를 지원하지 않으며, **기술 프리뷰로** 중첩만 제공합니다.

지원되는 아키텍처

•

L0 호스트는 Intel, AMD, IBM POWER9 또는 IBM Z 시스템이어야 합니다. 중첩된 가상화는 현재 다른 아키텍처에서 작동하지 않습니다.

지원되는 게스트 운영 체제

- 중첩된 VM을 생성하려면 다음과 같은 게스트 운영 체제(OS)를 사용해야 합니다.

- L0 호스트에서 - RHEL 8.2 이상
 - L1 VM에서 - RHEL 7.8 이상 또는 RHEL 8.2 이상



참고

이 지원은 L1 VM에서 RHEL 7 및 RHEL 8을 기반으로 하는 가상화 제품 사용에 적용되지 않습니다. 여기에는 다음이 포함됩니다.

- Red Hat Virtualization
- Red Hat OpenStack Platform
- OpenShift Virtualization

- L2 VM에서 다음 OS 중 하나를 사용해야 합니다.

- RHEL 7.8 이상
 - RHEL 8.2 이상
 - Microsoft Windows Server 2016
 - Microsoft Windows Server 2019

- 또한 **IBM POWER9**에서 중첩된 가상화는 현재 다음과 같은 상황에서만 작동합니다.
 - **L0 호스트와 L1 VM 모두 RHEL 8**을 사용합니다.
 - **L2 VM은 RHEL 8 또는 RHEL 7을 rhel-alt 커널과 함께** 사용합니다.
 - **L1 VM 및 L2 VM은 POWER8 호환성 모드에서** 실행되지 않습니다.

하이퍼바이저 제한 사항

- 현재 **Red Hat**은 **RHEL-KVM**에서만 중첩을 지원합니다. **RHEL**을 **L0** 하이퍼바이저로 사용하는 경우 **WSL 2용 RHEL 8** 또는 **Windows**를 **L1** 하이퍼바이저로 사용할 수 있습니다.
- **VMware ESXi** 또는 **AWS(Amazon Web Services)**와 같은 비**KVM L0** 하이퍼바이저에서 **L1 RHEL 8 VM**을 사용하는 경우 **RHEL 8 게스트 OS**에서 **L2 VM**을 생성하는 것이 작동하지만 지원되지 않습니다.

기능 제한

- **L2 VM**을 하이퍼바이저로 사용하고 **L3** 게스트 생성은 제대로 테스트되지 않았으며 작동하지 않습니다.
- **L0 호스트**에서 중첩된 가상화가 활성화된 경우 현재 **AMD** 시스템에서 **VM 마이그레이션**이 작동하지 않습니다.
- **IBM Z** 시스템에서는 대규모 페이지 백업 스토리지와 중첩된 가상화를 동시에 사용할 수 없습니다.

```
# modprobe kvm hpage=1 nested=1
modprobe: ERROR: could not insert 'kvm': Invalid argument
# dmesg |tail -1
[90226.508366] kvm-s390: A KVM host that supports nesting cannot back its KVM
guests with huge pages
```

-

L0 호스트에서 사용할 수 있는 일부 기능은 **L1** 하이퍼바이저에서 사용할 수 없습니다.

예를 들어 **IBM POWER 9** 하드웨어에서는 외부 인터럽트 가상화 엔진(**XIVE**)이 작동하지 않습니다. 그러나 **L1 VM**은 에뮬레이트된 **XIVE** 인터럽트 컨트롤러를 사용하여 **L2 VM**을 시작할 수 있습니다.

19장. 가상 머신 문제 진단

VM(가상 머신)을 사용할 때 다양한 심각도 수준에 문제가 발생할 수 있습니다. 일부 문제는 빠르고 쉽게 수정할 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.

19.1. LIBVIRT 디버그 로그 생성

VM(가상 머신) 문제를 진단하려면 **libvirt** 디버그 로그를 생성하고 검토하는 것이 유용합니다. 디버그 로그 연결은 VM 관련 문제 해결을 요청하는 경우에도 유용합니다.

다음 섹션에서는 **디버그 로그가 무엇인지**, 이를 **영구적으로 설정하고**, **런타임 중에 활성화하고**, 문제를 보고할 때 **연결하는** 방법에 대해 설명합니다.

19.1.1. 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 **libvirt**가 시작될 때 활성화해야 합니다. 단일 세션 또는 **영구적으로** 로깅을 활성화할 수 있습니다. 데몬 **런타임 설정**을 수정하여 **libvirt** 데몬 세션이 이미 실행 중인 경우 로깅을 활성화할 수도 있습니다.

libvirt 디버그 로그 연결은 VM 문제와 관련된 지원을 요청할 때도 유용합니다.

19.1.2. libvirt 디버그 로그에 대한 영구 설정 활성화

libvirt를 시작할 때마다 **libvirt** 디버그 로깅이 자동으로 활성화되도록 구성할 수 있습니다. 기본적으로 **libvirtd**는 RHEL 8에서 유일한 **libvirt** 데몬입니다. **libvirt** 구성을 영구적으로 변경하려면 **/etc/libvirt** 디렉터리에 있는 **libvirtd.conf** 파일을 편집해야 합니다.

절차

1. 편집기에서 **libvirtd.conf** 파일을 엽니다.
2. 요구 사항에 따라 필터를 교체하거나 설정합니다.

표 19.1. 디버깅 필터 값

1	libvirt에서 생성된 모든 메시지를 기록합니다.
2	모든 디버깅되지 않은 정보를 기록합니다.
3	모든 경고 및 오류 메시지를 기록합니다. 이는 기본값입니다.
4	오류 메시지만 기록합니다.

예 19.1. 로깅 필터를 위한 데몬 설정 샘플

다음 설정:

- **remote,util.json** 및 **rpc** 계층의 모든 오류 및 경고 메시지를 기록합니다.
- 이벤트 계층의 오류 메시지만 기록합니다.
- 필터링된 로그를 **/var/log/libvirt/libvirt.log**에 저장합니다.

```
log_filters="3:remote 4:event 3:util.json 3:rpc"
log_outputs="1:file:/var/log/libvirt/libvirt.log"
```

3. 저장 후 종료합니다.
4. **libvirt** 데몬을 다시 시작합니다.

```
$ systemctl restart libvirtd.service
```

19.1.3. 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. 런타임 설정 수정은 구성 파일을 편집하거나 데몬을 다시 시작하지 않고 명령을 시도하려는 경우에도 유용합니다.

사전 요구 사항

- **libvirt-admin** 패키지가 설치되어 있는지 확인합니다.

절차

1. 선택 사항: 로그 필터의 활성 집합을 백업합니다.

```
# virt-admin 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 19.2. 디버깅 필터 값

1	libvirt에서 생성된 모든 메시지를 기록합니다.
2	모든 디버깅되지 않은 정보를 기록합니다.
3	모든 경고 및 오류 메시지를 기록합니다. 이는 기본값입니다.
4	오류 메시지만 기록합니다.

예 19.2. 로깅 필터를 위한 샘플 **virt-admin** 설정

다음 명령:

- **remote,util.json** 및 **rpc** 계층에서 모든 오류 및 경고 메시지를 기록합니다.
- 이벤트 계층에서 오류 메시지만 기록합니다.

```
# virt-admin 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 **/var/log/libvirt/** 디렉터리에 있는 **libvirt.log** 파일에 저장합니다.

```
# virt-admin 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 **libvirt**의 모듈에서 생성된 많은 양의 중복 정보가 포함될 수 있으므로 사용하지 않는 것이 좋습니다.

- **virt-admin** 유틸리티를 사용하여 빈 필터 세트를 지정합니다.

```
# virt-admin daemon-log-filters
Logging filters:
```

5.

선택 사항: 백업 파일을 사용하여 필터를 원래 상태로 복원합니다. 저장된 값을 사용하여 두 번째 단계를 수행하여 필터를 복원합니다.

19.1.4. 요청을 지원하기 위해 libvirt 디버그 로그 연결

VM(가상 머신) 문제를 진단하고 해결하려면 추가 지원을 요청해야 할 수 있습니다. 지원 요청에 디버그 로그를 연결하는 것이 좋습니다. 지원 팀이 VM 관련 문제를 신속하게 해결하는 데 필요한 모든 정보에 액세스할 수 있는지 확인하는 것이 좋습니다.

절차

- 문제를 보고하고 지원을 [요청하려면 지원 사례를 엽니다](#).

- 발생한 문제에 따라 다음 로그를 보고서와 함께 연결합니다.
 - **libvirt** 서비스에 문제가 있으면 호스트에서 `/var/log/libvirt/libvirt.log` 파일을 연결합니다.
 - 특정 VM에서의 문제의 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?](#)

19.2. 가상 머신 코어 덤프

VM(가상 시스템)이 충돌했거나 오작동한 이유를 분석하기 위해 나중에 분석 및 진단하기 위해 VM 코어를 디스크의 파일에 덤프할 수 있습니다.

이 섹션에서는 코어 덤프에 대한 간략한 소개를 제공하고 VM 코어를 특정 파일에 덤프 하는 방법을 설명합니다.

19.2.1. 가상 머신 코어 덤프 작동 방식

VM(가상 시스템)을 사용하려면 많은 실행 중인 프로세스가 정확하고 효율적으로 작동해야 합니다. 경우에 따라 실행 중인 VM을 사용하는 동안 예기치 않게 종료하거나 오작동할 수 있습니다. VM을 다시 시작하면 데이터가 재설정되거나 손실될 수 있으므로 VM 충돌로 인한 정확한 문제를 진단하기 어려울 수 있습니다.

이 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 물리적 메모리 이미지가 포함되어 있습니다. 이 정보는 수동으로 또는 **crash** 유틸리티와 같은 도구를 사용하여 VM 문제를 진단하는 데 사용할 수 있습니다.

추가 리소스

- [충돌 도움말 페이지](#)
- [크래시 Github 리포지토리](#)

19.2.2. 가상 머신 코어 덤프 파일 생성

VM(가상 시스템) 코어 덤프에는 지정된 시간에 VM 상태에 대한 자세한 정보가 포함되어 있습니다. 이 정보는 VM 스냅샷과 유사하게 VM의 오작동 또는 종료 시 문제를 감지하는 데 도움이 됩니다.

사전 요구 사항

- 파일을 저장할 디스크 공간이 충분한지 확인합니다. VM에서 점유하는 공간은 VM에 할당된 RAM의 양에 따라 다릅니다.

절차

- **virsh** 덤프 유틸리티를 사용합니다.

예를 들어 다음 명령은 **lander1** VM의 코어, 메모리 및 CPU 일반 레지스터 파일을 **/core/file** 디렉토리의 **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. 크래시를 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 메모

리를 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](#)

19.3. BACKTRACING 가상 머신 프로세스

VM(가상 시스템) 오작동과 관련된 프로세스가 발생하면 **gstack** 명령을 **PID**(프로세스 식별자)와 함께 사용하여 오작동 프로세스의 실행 스택 추적을 생성할 수 있습니다. 프로세스가 스레드 그룹의 일부인 경우 모든 스레드도 추적됩니다.

사전 요구 사항

- **GDB** 패키지가 설치되어 있는지 확인합니다.

GDB 및 사용 가능한 구성 요소 설치에 대한 자세한 내용은 [GNU Debugger 설치](#)를 참조하십시오.
- 역추적하려는 프로세스의 **PID**를 알고 있어야 합니다.

pgrep 명령 다음에 프로세스 이름을 사용하여 **PID**를 찾을 수 있습니다. 예를 들면 다음과 같습니다.

```
# pgrep libvirt
22014
22025
```

절차

- **gstack** 유틸리티를 사용하고 나면 **backtrace**하려는 프로세스의 **PID**를 추가합니다.

예를 들어 다음 명령은 **PID 22014**을 사용하여 **libvirt** 프로세스를 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 **FTP**를 사용하여 여러 가지 다른 방법을 사용하여 서비스 요청을 늘립니다.
 - 문제를 보고하고 지원을 요청하려면 [Open a Support Case](#) 를 참조하십시오.
- 서비스 요청을 제출할 때 **SOS** 보고서 및 로그 파일을 업로드합니다.

이를 통해 **Red Hat** 지원 엔지니어는 참조에 필요한 모든 진단 정보를 얻을 수 있습니다.

- **SOS** 보고서에 대한 자세한 내용은 [SOS 보고서란 무엇이며 Red Hat Enterprise Linux 에서 생성하는 방법은 무엇입니까?](#)
- 로그 파일을 연결하는 방법에 대한 자세한 내용은 [Red Hat 지원에 파일을 제공하는 방법을 참조하십시오.](#)

20장. RHEL 8 가상화의 기능 지원 및 제한 사항

이 문서에서는 **RHEL 8(Red Hat Enterprise Linux 8)** 가상화의 기능 지원 및 제한에 대한 정보를 제공합니다.

20.1. RHEL 가상화 지원의 작동 방식

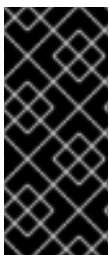
일련의 지원 제한 사항은 **RHEL 8(Red Hat Enterprise Linux 8)**의 가상화에 적용됩니다. 즉, **RHEL 8**에서 가상 머신을 사용할 때 특정 기능을 사용하거나 할당된 리소스를 초과할 때 특정 서브스크립션 계획이 없는 경우 **Red Hat**은 이러한 게스트를 지원하지 않습니다.

RHEL 8 가상화의 권장 기능에 나열된 기능은 **Red Hat**이 **RHEL 8** 시스템의 **KVM** 하이퍼바이저와 함께 작동하도록 테스트 및 인증되었습니다. 따라서 **RHEL 8**의 가상화에서 완전 지원 및 사용하는 것이 좋습니다.

RHEL 8 가상화의 Unsupported features에 나열된 기능은 작동할 수 있지만 **RHEL 8**에서는 지원되지 않으며 지원되지 않습니다. 따라서 **Red Hat**은 **KVM**의 **RHEL 8**에서 이러한 기능을 사용하지 않는 것이 좋습니다.

RHEL 8 가상화의 리소스 할당 제한은 **RHEL 8**의 **KVM** 게스트에서 지원되는 최대 특정 리소스 양입니다. 이러한 제한을 초과하는 게스트는 **Red Hat**에서 지원되지 않습니다.

별도로 명시하지 않는 한, **RHEL 8** 가상화에 대한 설명서에서 사용하는 모든 기능 및 솔루션이 지원됩니다. 그러나 이들 중 일부는 완전히 테스트되지 않았으므로 완전히 최적화되지 않을 수 있습니다.



중요

이러한 제한 사항 중 대부분은 **RHV(Red Hat Virtualization)**, **OpenShift Virtualization** 또는 **RHOSP(Red Hat OpenStack Platform)**와 같이 **Red Hat**에서 제공하는 다른 가상화 솔루션에는 적용되지 않습니다.

20.2. RHEL 8 가상화의 권장 기능

다음 기능은 **Red Hat Enterprise Linux 8 (RHEL 8)**에 포함된 **KVM** 하이퍼바이저와 함께 사용하는 것이 좋습니다.

호스트 시스템 아키텍처

KVM이 포함된 RHEL 8은 다음 호스트 아키텍처에서만 지원됩니다.

- **AMD64 및 Intel 64**
- **IBM Z - IBM z13 시스템 이상**
- **IBM POWER8**
- **IBM POWER9**

다른 하드웨어 아키텍처는 **RHEL 8을 KVM 가상화 호스트로 사용할 때 지원되지 않으며 Red Hat은 이러한 목표를 달성하는 것을 강력히 권장합니다.** 여기에는 **64비트 ARM 아키텍처(ARM 64)**가 포함됩니다.



참고

RHEL 8 설명서에서는 주로 AMD64 및 Intel 64 기능 및 사용에 대해 설명합니다. 다른 아키텍처에서 RHEL 8 가상화를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- [IBM POWER에서 가상화 시작하기](#)
- [IBM Z에서 가상화 시작하기](#)

게스트 운영 체제

Red Hat은 다음 OS(운영 체제)를 사용하는 KVM 가상 머신을 지원합니다.

- **Red Hat Enterprise Linux 6 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 라이선스 또는 서브스크립션을 활성화해야 합니다.

머신 유형

VM이 호스트 아키텍처와 호환되고 게스트 OS가 최적으로 실행되는지 확인하려면 VM에서 적절한 시스템 유형을 사용해야 합니다.

명령줄을 사용하여 VM을 생성할 때 **virt-install** 유틸리티는 머신 유형을 설정하는 여러 가지 방법을 제공합니다.

- **--os-variant** 옵션을 사용하면 **virt-install** 이 호스트 CPU에 권장하고 게스트 OS에서 지원하는 머신 유형을 자동으로 선택합니다.
- **--os-variant** 를 사용하지 않거나 다른 머신 유형이 필요한 경우 **--machine** 옵션을 사용하여 시스템 유형을 명시적으로 지정합니다.
- 호스트와 호환되지 않거나 호환되지 않는 **--machine** 값을 지정하면 **virt-install** 이 실패하고 오류 메시지가 표시됩니다.

지원되는 아키텍처에서 KVM 가상 머신에 권장되는 시스템 유형 및 **--machine** 옵션에 대한 해당 값은 다음과 같습니다. Y는 RHEL 8의 최신 마이너 버전을 나타냅니다.

- Intel 64 및 AMD64 (x86_64): PC-q35-rhel8.Y.0 → **--machine=q35**
- IBM Z (s390x): s390-ccw-virtio-rhel8.Y.0 → **--machine=s390-ccw-virtio**
- IBM POWER (PPC)에서 pseries-rhel8.Y.0 → **--machine=pseries**

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 8 가상화에서 지원되지 않는 기능](#)
- [RHEL 8 가상화의 리소스 할당 제한](#)
- [가상 머신 마이그레이션에 지원되는 호스트](#)

20.3. RHEL 8 가상화에서 지원되지 않는 기능

다음 기능은 **Red Hat Enterprise Linux 8 (RHEL 8)**에 포함된 **KVM** 하이퍼바이저에서 지원되지 않습니다.



중요

이러한 제한 사항은 **RHV(Red Hat Virtualization)**, **OpenShift Virtualization** 또는 **RHOSP(Red Hat OpenStack Platform)**와 같이 **Red Hat**에서 제공하는 다른 가상화 솔루션에는 적용되지 않을 수 있습니다.

RHV 4.2 이상 또는 **RHOSP 13** 이상에서 지원하는 기능은 다음 단락과 같이 설명되어 있습니다.

호스트 시스템 아키텍처

RHEL 8 KVM은 [RHEL 8 가상화의 권장 기능](#)에 나열되지 않은 호스트 아키텍처에서 지원되지 않습니다.

특히 **Red Hat**은 **RHEL 8**의 **KVM** 가상화를 위한 **64비트 ARM** 아키텍처(**ARM 64**)가 포함된 시스템 사용을 지원하지 않습니다.

게스트 운영 체제

RHEL 8 호스트에서 다음 게스트 운영 체제(OS)를 사용하는 KVM 가상 머신(VM)은 지원되지 않습니다.

- Microsoft Windows 8.1 이하
- Microsoft Windows Server 2012 이하
- macOS
- Solaris for x86 systems
- 2009년 이전에 출시된 모든 OS

RHEL 호스트에서 지원되는 게스트 OS 목록은 [Certified guest operating systems for Red Hat Enterprise Linux with KVM](#)을 참조하십시오.

Red Hat에서 제공하는 다른 가상화 솔루션에서 지원하는 게스트 OS 목록은 [Certified Guest Operating Systems in Red Hat OpenStack Platform and Red Hat Virtualization](#)에서 참조하십시오.

RHV에서 특별히 지원하는 게스트 OS 목록은 [RHV에서 지원되는 게스트 운영 체제](#)를 참조하십시오.

컨테이너에서 VM 생성

Red Hat은 RHEL 8 하이퍼바이저(예: QEMU 에뮬레이터 또는 libvirt 패키지)의 요소를 포함하는 모든 유형의 컨테이너에서 KVM 가상 머신 생성을 지원하지 않습니다.

컨테이너에 VM을 생성하려면 [OpenShift Virtualization](#) 오퍼링을 사용하는 것이 좋습니다.

QEMU 명령줄

QEMU는 RHEL 8의 가상화 아키텍처의 필수 구성 요소이지만 수동으로 관리하기 어렵고 QEMU 구성이 잘못되어 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 언플러그

실행 중인 VM에서 가상 CPU(vCPU)를 제거하는 것은 RHEL 8에서는 vCPU 핫 언플러그라고도 합니다.

RHV에서 vCPU 핫 언플러그가 지원됩니다. 자세한 내용은 [핫 플러그 VCPU](#)를 참조하십시오.

메모리 핫 언플러그

실행 중인 VM에 연결된 메모리 장치를 제거하면 메모리 핫 언플러그라고도 하며 RHEL 8에서는 지원되지 않습니다.

메모리 핫 언플러그는 RHV에서 지원되지만 RHEL을 실행하는 게스트 VM에만 특정 게스트 구성이 지원됩니다. 자세한 내용은 [핫 언플러그 가상 메모리에서 참조하십시오](#).

QEMU 측 I/O 제한

QEMU 측 I/O 제한이라고도 하는 가상 디스크 작업의 최대 입력 및 출력 수준을 RHEL 8에서는 **virsh blkdeviotune** 유틸리티를 사용하여 구성하지 않습니다.

RHEL 8에서 I/O 제한을 설정하려면 **virsh blkiotune** 을 사용합니다. **libvirt-side** I/O 제한이라고도 합니다. 자세한 내용은 [가상 머신의 디스크 I/O](#) 제한을 참조하십시오.

RHV에서 QEMU 측 I/O 제한이 지원됩니다. 자세한 내용은 [스토리지 서비스 품질](#)을 참조하십시오.

QEMU 측 I/O 제한도 RHOSP에서 지원됩니다. 자세한 내용은 [RHOSP 스토리지 가이드의 디스크에서 리소스 제한 설정 및 서비스 품질 사양 사용](#) 섹션을 참조하십시오.

또한 OpenShift Virtualization은 QEMU 측 I/O 제한도 지원합니다.

스토리지 실시간 마이그레이션

RHEL 8에서는 호스트 간에 실행 중인 VM의 디스크 이미지를 마이그레이션하는 것은 지원되지 않습니다.

RHV에서 스토리지 실시간 마이그레이션이 지원됩니다. 자세한 내용은 [라이브 스토리지 마이그레이션 개요](#)를 참조하십시오.

RHOSP에서도 스토리지 실시간 마이그레이션이 지원되지만 몇 가지 제한 사항이 있습니다. 자세한 내용은 [블록 마이그레이션](#)을 참조하십시오.

실시간 스냅샷

실행 중인 **VM**의 스냅샷을 생성하거나 로드하는 것은 실시간 스냅샷이라고도 하며 **RHEL 8**에서는 지원되지 않습니다.

또한 **RHEL 8**에서는 비실시간 **VM** 스냅샷이 더 이상 사용되지 않습니다. 따라서 종료 **VM**의 스냅샷을 생성하거나 로드하는 것은 지원되지만 **Red Hat**은 이를 사용하지 않는 것이 좋습니다.

RHV에서 실시간 스냅샷이 지원됩니다. 자세한 내용은 [Red Hat Virtualization의 라이브 스냅샷](#)을 참조하십시오.

RHOSP에서도 실시간 스냅샷이 지원됩니다. 자세한 내용은 [가상 머신을 오버클라우드로 가져오기](#)를 참조하십시오.

vhost-user

RHEL 8에서는 사용자 공간 **vHost** 인터페이스 구현을 지원하지 않습니다.

vhost-user는 **RHOSP**에서 지원되지만 **virtio-net** 인터페이스의 경우에만 지원됩니다. 자세한 내용은 [virtio-net 구현](#) 및 [vhost 사용자 포트](#)를 참조하십시오.

S3 및 S4 시스템 전원 상태

VM을 **RAM** 일시 중단(**S3**) 또는 **Suspend to disk (S4)** 시스템 전원 상태로 일시 중단하는 것은 지원되지 않습니다. 이러한 기능은 기본적으로 비활성화되어 있으며 이를 활성화하면 **Red Hat**에서 **VM**을 지원할 수 없습니다.

S3 및 **S4** 상태는 현재 **Red Hat**에서 제공하는 다른 가상화 솔루션에서도 지원되지 않습니다.

다중 경로 vDisk의 S3-PR

RHEL 8에서는 다중 경로 vDisk의 **SCSI3** 영구 예약(**S3-PR**)이 지원되지 않습니다. 따라서 **RHEL 8**에서는 **Windows** 클러스터가 지원되지 않습니다.

다중 경로 vDisk의 **S3-PR**은 **RHV**에서 지원됩니다. 따라서 **Windows** 클러스터 지원이 필요한 경우 **Red Hat**은 가상화 솔루션으로 **RHV**를 사용하는 것이 좋습니다. 자세한 내용은 [RHV 게스트에 대한 클러스터 지원](#)을 참조하십시오.

virtio-crypto

RHEL 8에서 **virtio-crypto** 장치 사용은 지원되지 않으므로 사용하지 않는 것이 좋습니다.

virtio-crypto 장치는 **Red Hat**에서 제공하는 다른 가상화 솔루션에서도 지원되지 않습니다.

증분 라이브 백업

증분 라이브 백업이라고도 하는 **VM** 백업에서만 **VM**을 구성하는 것은 **RHEL 8**에서 지원되지 않으며 **Red Hat**은 그 용도가 크게 저하됩니다.

증분 라이브 백업은 **RHV 4.4** 이상에서 기술 프리뷰로 제공됩니다.

net_failover

net_failover 드라이버를 사용하여 자동화된 네트워크 장치 장애 조치 메커니즘을 설정하는 것은 **RHEL 8**에서 지원되지 않습니다.

net_failover 는 현재 **Red Hat**에서 제공하는 다른 가상화 솔루션에서도 지원되지 않습니다.

멀티 FD 마이그레이션

RHEL 8에서는 다중 **FD**(파일 설명자)라고도 하는 다중 파일 설명자(**FD**)를 사용하여 **VM** 마이그레이션은 지원되지 않습니다.

다중 **FD** 마이그레이션은 현재 **Red Hat**에서 제공하는 다른 가상화 솔루션에서도 지원되지 않습니다.

virtiofs

virtiofs 파일 시스템을 사용하여 호스트와 **VM** 간 파일 공유는 **RHEL 8**에서 지원되지 않습니다.

virtiofs 사용은 현재 **Red Hat**에서 제공하는 다른 가상화 솔루션에서도 지원되지 않습니다.

NVMe 장치

PCI-passthrough를 사용하여 **PKCS** 장치로 **VM**에 **NVMe(Non-volatile Memory Express)** 장치를 연결하는 것은 지원되지 않습니다.

NVMe 장치를 **VM**에 연결하는 것도 현재 **Red Hat**에서 제공하는 다른 가상화 솔루션에서는 지원되지 않습니다.

TCG

QEMU 및 **libvirt**에는 **QEMU Tiny Code** 생성기(**TCG**)를 사용하는 동적 변환 모드가 포함되어 있습니다. 이 모드에서는 하드웨어 가상화 지원이 필요하지 않습니다. 그러나 **TCG**는 **Red Hat**에서 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 8 가상화의 권장 기능](#)
- [RHEL 8 가상화의 리소스 할당 제한](#)

20.4. RHEL 8 가상화의 리소스 할당 제한

다음 제한은 **RHEL 8(Red Hat Enterprise Linux 8)** 호스트의 단일 **KVM** 가상 머신(**VM**)에 할당할 수 있

는 가상화 리소스에 적용됩니다.



중요

이러한 제한 사항 중 다수는 **RHV(Red Hat Virtualization)**, **OpenShift Virtualization** 또는 **RHOSP(Red Hat OpenStack Platform)**와 같이 Red Hat에서 제공하는 다른 가상화 솔루션에는 적용되지 않습니다.

VM당 최대 vCPU

RHEL 8은 단일 VM에 할당된 최대 **384** 개의 vCPU를 지원합니다.

VM당 PCI 장치

RHEL 8은 VM 버스당 **64** PCI 장치 슬롯과 장치 슬롯당 **8** 개의 PCI 기능을 지원합니다. 이를 통해 VM에서 다기능 기능이 활성화되어 PCI 브리지가 사용되지 않는 경우, 기본적으로 버스당 최대 **512** PCI 함수가 제공됩니다.

각 PCI 브리지는 새 버스를 추가하여 다른 **512** 장치 주소를 활성화할 수 있습니다. 그러나 일부 버스는 사용자가 사용할 수 있는 **512** 장치 주소를 모두 만들지 않습니다. 예를 들어 루트 버스에는 슬롯을 점유하는 몇 개의 내장 장치가 있습니다.

가상화된 IDE 장치

KVM은 VM당 최대 **4** 개의 가상화 IDE 장치로 제한됩니다.

20.5. RHEL 8의 가상화 기능 지원 개요

다음 표에서는 지원되는 시스템 아키텍처에서 **RHEL 8**에서 선택한 가상화 기능의 지원 상태에 대한 비교 정보를 제공합니다.

표 20.1. 장치 핫 플러그 및 핫 언플러그

	Intel 64 및 AMD64	IBM Z	IBM POWER
CPU 핫 플러그	지원됨	지원됨	지원됨
CPU 핫플러그	지원되지 않음	지원되지 않음	지원되지 않음
메모리 핫 플러그	지원됨	지원되지 않음	지원됨

	Intel 64 및 AMD64	IBM Z	IBM POWER
메모리 핫 언플러그	지원되지 않음	지원되지 않음	지원되지 않음
PCI 핫 플러그	지원됨	지원됨 [a]	지원됨
PCI 핫플러그	지원됨	지원됨 [b]	지원됨
[a] virtio-* pci 대신 virtio-*ccw 장치를 사용해야 합니다			
[b] virtio-* pci 대신 virtio-*ccw 장치를 사용해야 합니다			

표 20.2. 기타 선택된 기능

	Intel 64 및 AMD64	IBM Z	IBM POWER
NUMA 튜닝	지원됨	지원되지 않음	지원됨
SR-IOV 장치	지원됨	지원되지 않음	지원됨
virt-v2v and p2v	지원됨	지원되지 않음	지원되지 않음

지원되지 않는 일부 기능은 **Red Hat Virtualization** 및 **Red Hat OpenStack** 플랫폼과 같은 다른 **Red Hat** 제품에서 지원됩니다. 자세한 내용은 [RHEL 8 가상화의 지원되지 않는 기능을](#) 참조하십시오.

추가 소스

- **RHEL 8**에서 지원되지 않는 가상 머신의 전체 목록은 [RHEL 8 가상화의 지원되지 않는 기능을](#) 참조하십시오.
- **IBM Z** 아키텍처에서의 가상화에 대한 자세한 내용은 [How virtualization on IBM Z and Intel 64에서](#) 참조하십시오.
- **IBM POWER** 아키텍처에서의 가상화 관련 세부 사항에 대한 자세한 내용은 [How virtualization on IBM POWER different from AMD64 및 Intel 64.](#)