OpenShift Container Platform 4.16

가상화

OpenShift Virtualization 설치, 사용법, 릴리스 정보

Last Updated: 2024-09-02

OpenShift Container Platform 4.16 가상화

OpenShift Virtualization 설치, 사용법, 릴리스 정보

법적 공지

초록

이 문서에서는 OpenShift Container Platform에서 OpenShift Virtualization을 사용하는 방법에 대한 정보를 제공합니다.

차례

1장. 정보

1.1. OPENSHIFT VIRTUALIZATION 정보

OpenShift Virtualization의 기능 및 지원 범위에 대해 알아보십시오.

1.1.1. OpenShift Virtualization으로 수행할 수 있는 작업

OpenShift Virtualization은 컨테이너 워크로드와 함께 가상 머신 워크로드를 실행하고 관리할 수 있는 OpenShift Container Platform의 애드온입니다.

OpenShift Virtualization은 Kubernetes 사용자 지정 리소스를 사용하여 가상화 작업을 활성화하여 OpenShift Container Platform 클러스터에 새 개체를 추가합니다. 다음과 같은 가상화 작업이 지원됩니다.

- Linux 및 Windows VM(가상 머신) 생성 및 관리
- 클러스터에서 서로 함께 Pod 및 VM 워크로드 실행
- 다양한 콘솔 및 CLI 툴을 통해 가상 머신에 연결
- 기존 가상 머신 가져오기 및 복제
- 가상 머신에 연결된 네트워크 인터페이스 컨트롤러 및 스토리지 디스크 관리
- 노드 간 실시간 가상 머신 마이그레이션

향상된 웹 콘솔에서 제공되는 그래픽 포털을 통해 OpenShift Container Platform 클러스터 컨테이너 및 인프라와 함께 가상화 리소스를 관리할 수 있습니다.

OpenShift Virtualization은 Red Hat OpenShift Data Foundation 기능과 원활하게 작동하도록 설계 및 테스트되었습니다.

중요

OpenShift Data Foundation을 사용하여 OpenShift Virtualization을 배포할 때 Windows 가상 머신 디스크용 전용 스토리지 클래스를 생성해야 합니다. 자세한 내용은 Windows VM용 ODF PersistentVolume 최적화 를 참조하십시오.

OVN-Kubernetes,OpenShift SDN 또는 인증된 OpenShift CNI 플러그인에 나열된 다른 인증 네트워크 플러그인 중 하나와 함께 OpenShift Virtualization을 사용할 수 있습니다.

Compliance Operator 를 설치하고 **ocp4-moderate** 및 **ocp4-moderate-node** 프로필 을 사용하여 검사를 실행하여 OpenShift Virtualization 클러스터에서 규정 준수 문제를 확인할 수 있습니다. Compliance Operator는 NIST 인증 툴 인 OpenSCAP을 사용하여 보안 정책을 검사하고 적용합니다.

1.1.1.1. OpenShift Virtualization 지원 클러스터 버전

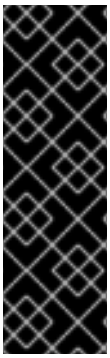
OpenShift Container Platform 4.16 클러스터에서 사용할 수 있도록 OpenShift Virtualization 4.16이 지원됩니다. OpenShift Virtualization의 최신 z-stream 릴리스를 사용하려면 먼저 OpenShift Container Platform의 최신 버전으로 업그레이드해야 합니다.

1.1.2. 가상 머신 디스크의 볼륨 및 액세스 모드 정보

알려진 스토리지 공급자와 스토리지 API를 사용하는 경우 볼륨 및 액세스 모드가 자동으로 선택됩니다. 그러나 스토리지 프로필이 없는 스토리지 클래스를 사용하는 경우 볼륨 및 액세스 모드를 구성해야 합니다.

최상의 결과를 얻으려면 RWX(**ReadWriteMany**) 액세스 모드와 **Block** 볼륨 모드를 사용합니다. 이는 다음과 같은 이유로 중요합니다.

- 실시간 마이그레이션에는 RWX(**ReadWriteMany**) 액세스 모드가 필요합니다.
- **블록** 볼륨 모드는 **Filesystem** 볼륨 모드보다 훨씬 더 잘 작동합니다. 이는 **Filesystem** 볼륨 모드가 파일 시스템 계층 및 디스크 이미지 파일을 포함하여 더 많은 스토리지 계층을 사용하기 때문입니다. 이러한 계층은 VM 디스크 스토리지에 필요하지 않습니다.
  예를 들어 Red Hat OpenShift Data Foundation을 사용하는 경우 CephFS 볼륨에 Ceph RBD 볼륨을 사용하는 것이 좋습니다.



중요

다음 구성으로 가상 머신을 실시간 마이그레이션할 수 없습니다.

- RWO(**ReadWriteOnce**) 액세스 모드가 있는 스토리지 볼륨
- GPU와 같은 패스스루 기능

이러한 가상 머신에 대해 **evictionStrategy** 필드를 **None** 으로 설정합니다. **None** 전략은 노드를 재부팅하는 동안 VM의 전원을 끕니다.

1.1.3. 단일 노드 OpenShift 차이점

단일 노드 OpenShift에 OpenShift Virtualization을 설치할 수 있습니다.

그러나 Single-node OpenShift는 다음 기능을 지원하지 않습니다.

- 고가용성
- Pod 중단
- 실시간 마이그레이션
- 제거 전략이 구성된 가상 머신 또는 템플릿

1.1.4. 추가 리소스

- OpenShift Container Platform 스토리지의 일반 용어집
- 단일 노드 OpenShift 정보
- 지원되는 설치 관리자
- Pod 중단 예산
- 실시간 마이그레이션 정보
- 제거 전략
- 튜닝 및 확장 가이드
- OpenShift Virtualization 4.x에 지원되는 제한 사항

1.2. 보안 정책

OpenShift Virtualization 보안 및 권한 부여에 대해 알아보십시오.

키 포인트

- OpenShift Virtualization은 Pod 보안을 위해 현재 모범 사례를 적용하는 것을 목표로 **제한된** Kubernetes Pod 보안 표준 프로필을 준수합니다.
- VM(가상 머신) 워크로드는 권한이 없는 Pod로 실행됩니다.
- SCC(보안 컨텍스트 제약 조건)는 **kubevirt-controller** 서비스 계정에 대해 정의됩니다.
- OpenShift Virtualization 구성 요소의 TLS 인증서는 자동으로 갱신되고 순환됩니다.

1.2.1. 워크로드 보안 정보

기본적으로 VM(가상 머신) 워크로드는 OpenShift Virtualization에서 root 권한으로 실행되지 않으며 루트 권한이 필요한 OpenShift Virtualization 기능이 없습니다.

각 VM에 대해 **virt-launcher** Pod는 *세션 모드에서* **libvirt** 인스턴스를 실행하여 VM 프로세스를 관리합니다. 세션 모드에서 **libvirt** 데몬은 루트가 아닌 사용자 계정으로 실행되며 동일한 사용자 식별자(UID)에서 실행 중인 클라이언트의 연결만 허용합니다. 따라서 VM은 권한이 없는 포드로 실행되며 최소 권한의 보안 원칙을 따릅니다.

1.2.2. TLS 인증서

OpenShift Virtualization 구성 요소의 TLS 인증서는 자동으로 갱신되고 순환됩니다. 수동으로 새로 고치지 않아도 됩니다.

자동 갱신 일정

TLS 인증서는 다음 일정에 따라 자동으로 삭제되고 교체됩니다.

- KubeVirt 인증서는 매일 갱신됩니다.
- CDI(Containerized Data Importer) 컨트롤러 인증서는 15일마다 갱신됩니다.
- MAC 풀 인증서는 매년 갱신됩니다.

자동 TLS 인증서 순환이 수행되어도 작업이 중단되지 않습니다. 예를 들면 다음 작업이 중단되지 않고 계속 수행됩니다.

- 마이그레이션
- 이미지 업로드
- VNC 및 콘솔 연결

1.2.3. 권한 부여

OpenShift Virtualization에서는 역할 기반 액세스 제어 (RBAC)를 사용하여 사용자 및 서비스 계정에 대한 권한을 정의합니다. 서비스 계정에 정의된 권한은 OpenShift Virtualization 구성 요소가 수행할 수 있는 작업을 제어합니다.

RBAC 역할을 사용하여 가상화 기능에 대한 사용자 액세스를 관리할 수도 있습니다. 예를 들어 관리자는 가상 머신을 시작하는 데 필요한 권한을 제공하는 RBAC 역할을 생성할 수 있습니다. 그러면 관리자가 특정 사용자에게 역할을 바인딩하여 액세스를 제한할 수 있습니다.

1.2.3.1. OpenShift Virtualization의 기본 클러스터 역할

클러스터 역할 집계를 사용하면 OpenShift Virtualization에서 기본 OpenShift Container Platform 클러스터 역할을 확장하여 가상화 오브젝트에 대한 액세스 권한을 포함합니다.

표 1.1. OpenShift Virtualization 클러스터 역할

기본 클러스터 역할	OpenShift Virtualization 클러스터 역할	OpenShift Virtualization 클러스터 역할 설명
view	**kubevirt.io:view**	클러스터의 모든 OpenShift Virtualization 리소스를 볼 수 있지만 생성, 삭제, 수정 또는 액세스할 수 없는 사용자입니다. 예를 들어 사용자는 VM(가상 머신)이 실행 중이지만 이를 종료하거나 콘솔에 액세스할 수 없음을 확인할 수 있습니다.
edit	**kubevirt.io:edit**	클러스터의 모든 OpenShift Virtualization 리소스를 수정할 수 있는 사용자입니다. 예를 들어 사용자가 VM을 생성하고 VM 콘솔에 액세스한 후 VM을 삭제할 수 있습니다.
admin	**kubevirt.io:admin**	리소스 컬렉션 삭제 기능을 포함하여 모든 OpenShift Virtualization 리소스에 대한 전체 권한이 있는 사용자입니다. 사용자는 **openshift-cnv** 네임스페이스의 **HyperConverged** 사용자 지정 리소스에 있는 OpenShift Virtualization 런타임 구성을 보고 수정할 수도 있습니다.

1.2.3.2. OpenShift Virtualization의 스토리지 기능에 대한 RBAC 역할

cdi-operator 및 **cdi-controller** 서비스 계정을 포함하여 CDI(Containerized Data Importer)에 다음 권한이 부여됩니다.

1.2.3.2.1. 클러스터 전체 RBAC 역할

표 1.2. cdi.kubevirt.io API 그룹에 대해 집계된 클러스터 역할

CDI 클러스터 역할	Resources	verbs
cdi.kubevirt.io:admin	**DataVolumes,uploadtokenrequests**	* (모두)
	DataVolumes/source	**create**
cdi.kubevirt.io:edit	**DataVolumes,uploadtokenrequests**	*
	DataVolumes/source	**create**

CDI 클러스터 역할	Resources	verbs
cdi.kubevirt.io:view	**cdiconfigs,dataimportcrons,데이터 소스,datavolumes,objecttransfers,storageprofiles,volumeimportsources,volumeuploadsources,volumeclonesources**	**get,list,watch**
	DataVolumes/source	**create**
cdi.kubevirt.io:config-reader	**cdiconfigs,storageprofiles**	**get,list,watch**

표 1.3. **cdi-operator** 서비스 계정의 클러스터 전체 역할

API 그룹	Resources	verbs
rbac.authorization.k8s.io	**clusterrolebindings,clusterroles**	**get,list,watch,create,update,delete**
security.openshift.io	**securitycontextconstraints**	**get,list,watch,update,create**
apiextensions.k8s.io	**customresourcedefinitions, customresourcedefinitions/status**	**get,list,watch,create,update,delete**
cdi.kubevirt.io	*	*
upload.cdi.kubevirt.io	*	*
admissionregistration.k8s.io	**ValidatingWebhookConfigurations,mutatingwebhookconfigurations**	생성,목록,감시
admissionregistration.k8s.io	**ValidatingWebhookConfigurations** 허용 목록: **cdi-api-dataimportcron-validate, cdi-api-populator-validate, cdi-api-datavolume-validate, cdi-api-validate, objecttransfer-api-validate**	**get,update,delete**

API 그룹	Resources	verbs
admissionregistration.k8s.io	**mutatingwebhookconfigurations** 허용 목록: **cdi-api-datavolume-mutate**	**get,update,delete**
apiregistration.k8s.io	**APIServices**	**get,list,watch,create,update,delete**

표 1.4. **cdi-controller** 서비스 계정의 클러스터 전체 역할

API 그룹	Resources	verbs
"" (코어)	이벤트	**create,patch**
"" (코어)	**persistentvolumeclaims**	**get,list,watch,create,update,delete,deletecollection,patch**
"" (코어)	**persistentVolumes**	**get,list,watch,update**
"" (코어)	**PersistentVolumeClaims/finalizers,pods/finalizers**	업데이트
"" (코어)	**Pod**,서비스	**get,list,watch,create,delete**
"" (코어)	**configmaps**	**get,create**
storage.k8s.io	**storageclasses**, **csidrivers**	**get,list,watch**
config.openshift.io	프록시	**get,list,watch**
cdi.kubevirt.io	*	*
snapshot.storage.k8s.io	**VolumeSnapshots,volumesnapshotclasses,volumesnapshotcontents**	**get,list,watch,create,delete**
snapshot.storage.k8s.io	**VolumeSnapshots**	업데이트,삭제 수집
apiextensions.k8s.io	**CustomResourceDefinitions**	**get,list,watch**

API 그룹	Resources	verbs
scheduling.k8s.io	**priorityclasses**	**get,list,watch**
image.openshift.io	이미지 스트림	**get,list,watch**
"" (코어)	**secrets**	**create**
kubevirt.io	**virtualmachines/finalizers**	업데이트

1.2.3.2.2. 네임스페이스가 지정된 RBAC 역할

표 1.5. **cdi-operator** 서비스 계정에 대한 네임스페이스 지정 역할

API 그룹	Resources	verbs
rbac.authorization.k8s.io	**rolebindings**,역할	**get,list,watch,create,update,delete**
"" (코어)	**serviceaccounts,configmaps,events,secrets,services**	**get,list,watch,create,update,patch,delete**
앱	**deployments, deployments/finalizers**	**get,list,watch,create,update,delete**
route.openshift.io	**routes, routes/custom-host**	**get,list,watch,create,update**
config.openshift.io	프록시	**get,list,watch**
monitoring.coreos.com	**ServiceMonitors,prometheusrules**	**get,list,watch,create,delete,update,patch**
coordination.k8s.io	리스	**get,create,update**

표 1.6. **cdi-controller** 서비스 계정에 네임스페이스가 지정된 역할

API 그룹	Resources	verbs
"" (코어)	**configmaps**	**get,list,watch,create,update,delete**
"" (코어)	**secrets**	**get,list,watch**
batch	**cronjobs**	**get,list,watch,create,update,delete**

API 그룹	Resources	verbs
batch	작업	생성,삭제,목록,감시
coordination.k8s.io	리스	**get,create,update**
networking.k8s.io	**ingresses**	**get,list,watch**
route.openshift.io	**routes**	**get,list,watch**

1.2.3.3. kubevirt-controller 서비스 계정에 대한 추가 SCC 및 권한

SCC(보안 컨텍스트 제약 조건)는 Pod에 대한 권한을 제어합니다. 이러한 권한에는 컨테이너 모음인 Pod에서 수행할 수 있는 작업과 액세스할 수 있는 리소스가 포함됩니다. Pod가 시스템에 수용되려면 일련의 조건을 함께 실행해야 하는데, SCC를 사용하여 이러한 조건을 정의할 수 있습니다.

virt-controller 는 클러스터의 가상 머신에 대해 **virt-launcher** Pod를 생성하는 클러스터 컨트롤러입니다. 이러한 Pod에는 **kubevirt-controller** 서비스 계정에서 권한을 부여합니다.

kubevirt-controller 서비스 계정에는 적절한 권한으로 **virt-launcher** Pod를 생성할 수 있도록 추가 SCC 및 Linux 기능이 부여됩니다. 이러한 확장된 권한을 통해 가상 머신은 일반적인 Pod 범위를 벗어나는 OpenShift Virtualization 기능을 사용할 수 있습니다.

kubevirt-controller 서비스 계정에는 다음 SCC가 부여됩니다.

- **scc.AllowHostDirVolumePlugin = true**
  가상 머신에서 hostpath 볼륨 플러그인을 사용할 수 있습니다.
- **scc.AllowPrivilegedContainer = false**
  virt-launcher Pod가 권한 있는 컨테이너로 실행되지 않습니다.
- **scc.AllowedCapabilities = []corev1.Capability{"SYS_NICE", "NET_BIND_SERVICE"}**
  - **SYS_NICE** 를 사용하면 CPU 선호도를 설정할 수 있습니다.
  - **NET_BIND_SERVICE** 는 DHCP 및 Slirp 작업을 허용합니다.

kubevirt-controller에 대한 SCC 및 RBAC 정의 보기

oc 툴을 사용하여 **kubevirt-controller**에 대한 **SecurityContextConstraints** 정의를 볼 수 있습니다.

```
$ oc get scc kubevirt-controller -o yaml
```

oc 툴을 사용하여 **kubevirt-controller** clusterrole에 대한 RBAC 정의를 볼 수 있습니다.

```
$ oc get clusterrole kubevirt-controller -o yaml
```

1.2.4. 추가 리소스

- 보안 컨텍스트 제약 조건 관리
- RBAC를 사용하여 권한 정의 및 적용

- 클러스터 역할 생성
- 클러스터 역할 바인딩 명령
- 네임스페이스 간에 데이터 볼륨을 복제할 수 있는 사용자 권한 활성화

1.3. OPENSHIFT VIRTUALIZATION ARCHITECTURE

OLM(Operator Lifecycle Manager)은 OpenShift Virtualization의 각 구성 요소에 대해 Operator Pod를 배포합니다.

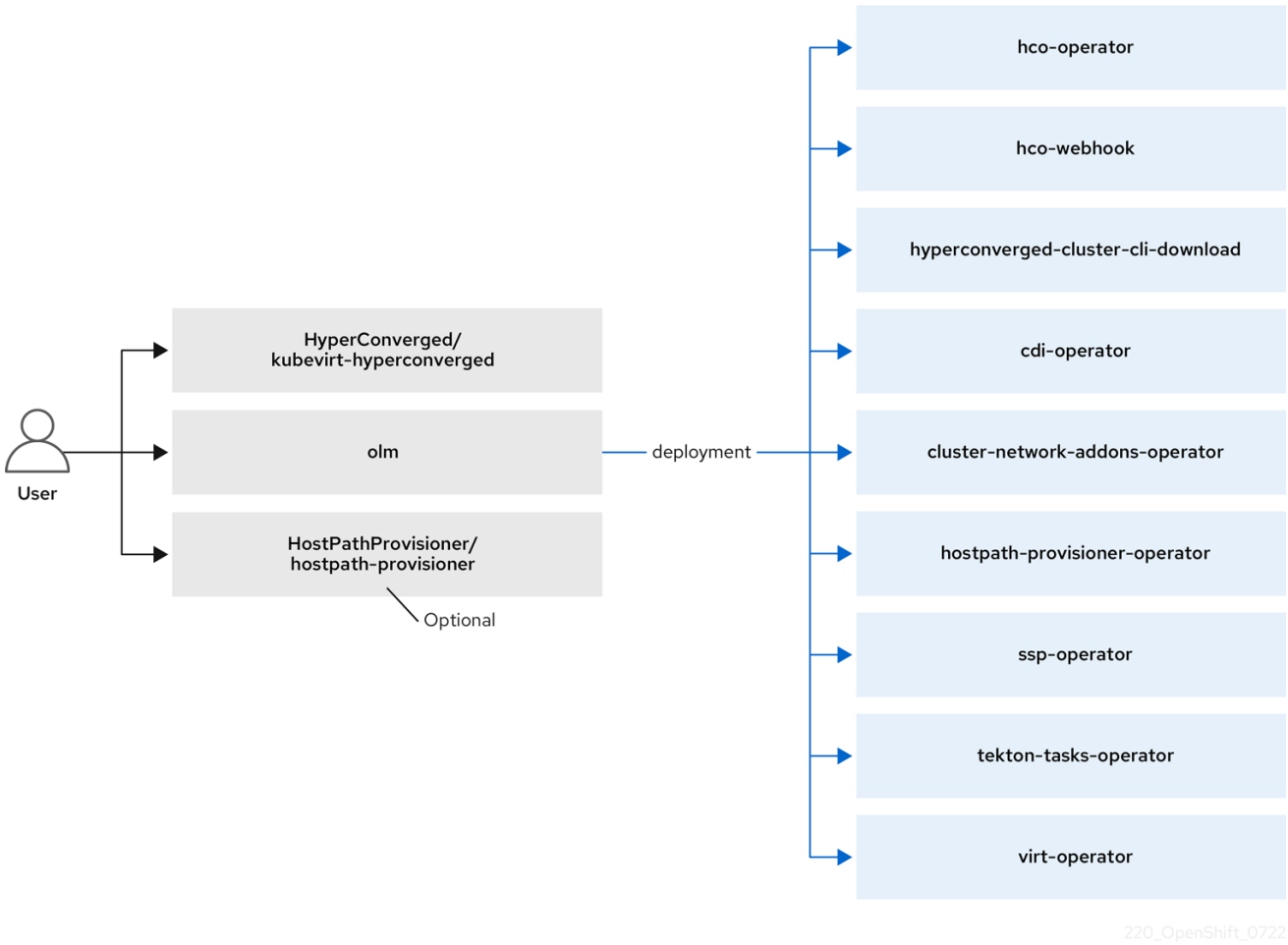
- Compute: **virt-operator**
- 스토리지: **cdi-operator**
- 네트워크: **cluster-network-addons-operator**
- 스케일링: **ssp-operator**

또한 OLM은 다른 구성 요소의 배포, 구성 및 라이프 사이클과 **hco-webhook** 및 하이퍼 컨버지드-cluster- **cli-download 라는 여러 도우미 Pod를 담당하는 hyperconverged-cluster- operator** Pod를 배포합니다.

모든 Operator Pod가 성공적으로 배포된 후 **HyperConverged** CR(사용자 정의 리소스)을 생성해야 합니다. **HyperConverged** CR에 설정된 구성은 단일 정보 소스 및 OpenShift Virtualization의 진입점 역할을 하며 CR의 동작을 안내합니다.

HyperConverged CR은 조정 루프 내에서 다른 모든 구성 요소의 Operator에 대한 해당 CR을 생성합니다. 각 Operator는 OpenShift Virtualization 컨트롤 플레인에 대한 데몬 세트, 구성 맵 및 추가 구성 요소와 같은 리소스를 생성합니다. 예를 들어 HyperConverged Operator(HCO)가 **KubeVirt** CR을 생성하면 OpenShift Virtualization Operator가 이를 조정하고 **virt-controller**,**virt-handler**, **virt-api** 와 같은 추가 리소스를 생성합니다.

OLM은 HPP(Hostpath Provisioner) Operator를 배포하지만 **hostpath-provisioner** CR을 생성할 때까지 작동하지 않습니다.



- Virtctl 클라이언트 명령

1.3.1. HyperConverged Operator(HCO) 정보

HCO인 **hco-operator** 는 OpenShift Virtualization과 의견이 지정된 기본값을 사용하여 여러 도우미 운영자를 배포 및 관리하기 위한 단일 진입점을 제공합니다. 또한 해당 Operator에 대한 CR(사용자 정의 리소스)을 생성합니다.

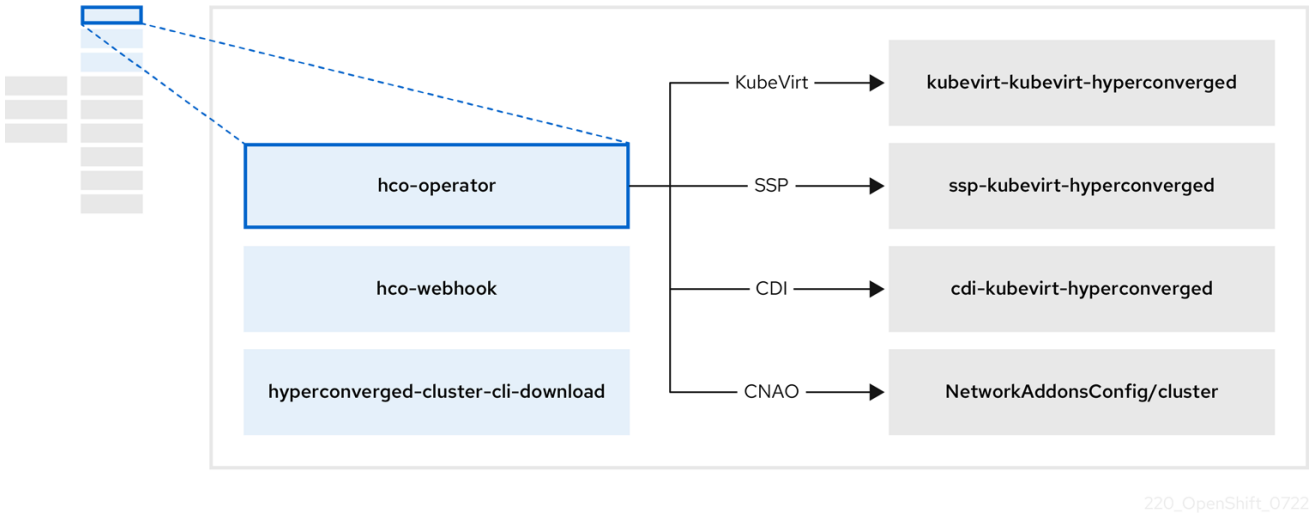


표 1.7. HyperConverged Operator 구성 요소

Component	설명
deployment/hco-webhook	**HyperConverged** 사용자 정의 리소스 콘텐츠를 검증합니다.
deployment/hyperconverged-cluster-cli-download	클러스터에서 직접 다운로드할 수 있도록 **virtctl** 툴 바이너리를 클러스터에 제공합니다.
KubeVirt/kubevirt-kubevirt-hyperconverged	OpenShift Virtualization에 필요한 모든 Operator, CR 및 개체를 포함합니다.
SSP/ssp-kubevirt-hyperconverged	스케줄링, 스케일 및 성능(SSP) CR. 이는 HCO에 의해 자동으로 생성됩니다.
CDI/cdi-kubevirt-hyperconverged	CDI(Containerized Data Importer) CR. 이는 HCO에 의해 자동으로 생성됩니다.
NetworkAddonsConfig/cluster	**cluster-network-addons-operator** 에 의해 지시 및 관리되는 CR입니다.

1.3.2. CDI(Containerized Data Importer) Operator 정보

CDI Operator인 **cdi-operator** 는 데이터 볼륨을 사용하여 CDI 및 해당 관련 리소스를 관리하여 VM(가상 머신) 이미지를 PVC(영구 볼륨 클레임)로 가져옵니다.

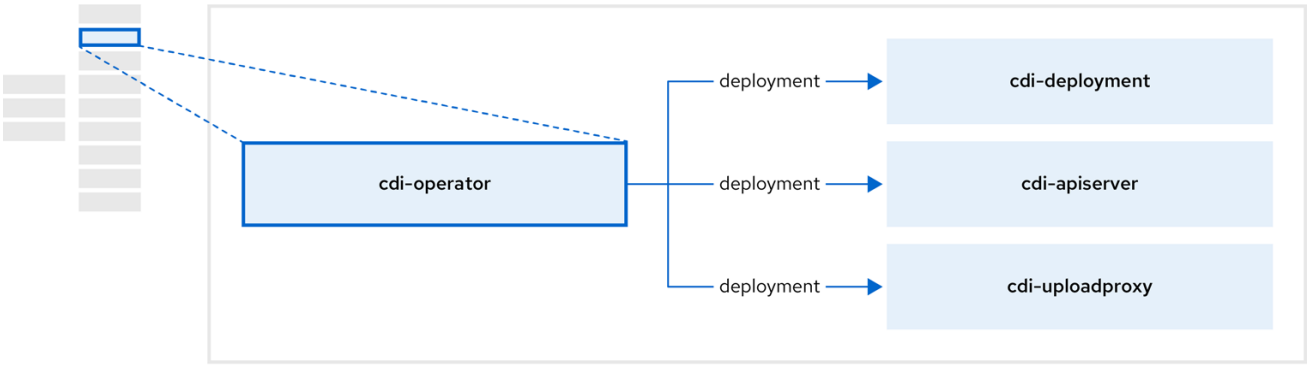


표 1.8. CDI Operator 구성 요소

Component	설명
deployment/cdi-apiserver	보안 업로드 토큰을 발행하여 VM 디스크를 PVC에 업로드하는 권한을 관리합니다.
deployment/cdi-uploadproxy	올바른 PVC에 쓸 수 있도록 외부 디스크 업로드 트래픽을 적절한 업로드 서버 pod로 전달합니다. 유효한 업로드 토큰이 필요합니다.

Component	설명
pod/cdi-importer	데이터 볼륨을 생성할 때 가상 머신 이미지를 PVC로 가져오는 도우미 Pod입니다.

1.3.3. CNO(Cluster Network Addons) Operator 정보

Cluster Network Addons Operator인 **cluster-network-addons-operator** 는 클러스터에 네트워킹 구성 요소를 배포하고 확장된 네트워크 기능에 대한 관련 리소스를 관리합니다.

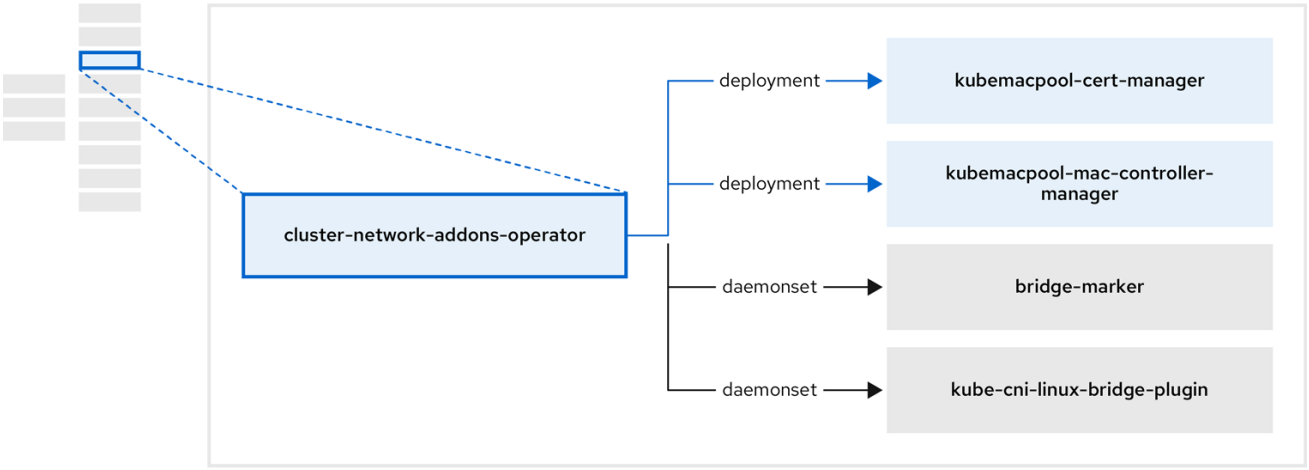


표 1.9. CNO(Cluster Network Addons) Operator 구성 요소

Component	설명
deployment/kubemacpool-cert-manager	Kubemacpool의 Webhook TLS 인증서를 관리합니다.
deployment/kubemacpool-mac-controller-manager	VM(가상 머신) 네트워크 인터페이스 카드(NIC)에 대한 MAC 주소 풀링 서비스를 제공합니다.
daemonset/bridge-marker	노드에서 사용 가능한 네트워크 브릿지를 노드 리소스로 표시합니다.
daemonset/kube-cni-linux-bridge-plugin	클러스터 노드에 CNI(Container Network Interface) 플러그인을 설치하여 네트워크 연결 정의를 통해 Linux 브리지에 VM을 연결할 수 있습니다.

1.3.4. HPP(Hostpath Provisioner) Operator 정보

HPP Operator인 **hostpath-provisioner-operator** 는 다중 노드 HPP 및 관련 리소스를 배포 및 관리합니다.

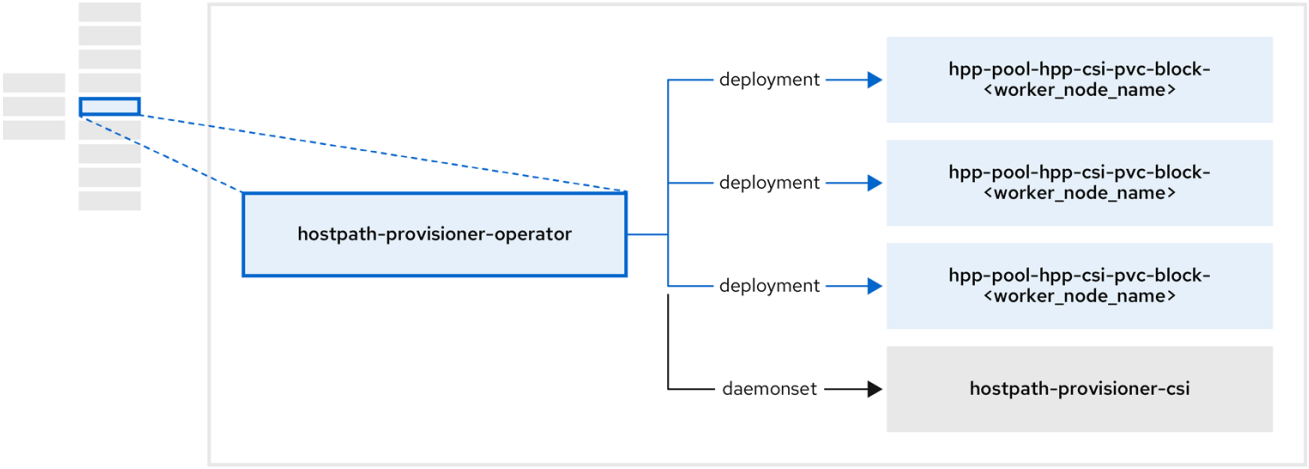


표 1.10. HPP Operator 구성 요소

Component	설명
deployment/hpp-pool-hpp-csi-pvc-block-<worker_node_name>	HPP가 실행되도록 지정된 각 노드에 대해 작업자를 제공합니다. Pod는 지정된 백업 스토리지를 노드에 마운트합니다.
daemonset/hostpath-provisioner-csi	HPP의 CSI(Container Storage Interface) 드라이버 인터페이스를 구현합니다.
daemonset/hostpath-provisioner	HPP의 레거시 드라이버 인터페이스를 구현합니다.

1.3.5. SSP(Scheduling, Scale, Performance) Operator 정보

SSP Operator, **ssp-operator** 는 공통 템플릿, 관련 기본 부팅 소스, 파이프라인 작업, 템플릿 검증기를 배포합니다.

1.3.6. OpenShift Virtualization Operator 정보

OpenShift Virtualization Operator인 **virt-operator** 는 현재 VM(가상 머신) 워크로드를 중단하지 않고 OpenShift Virtualization을 배포, 업그레이드 및 관리합니다. 또한 OpenShift Virtualization Operator는 공통 인스턴스 유형 및 일반적인 기본 설정을 배포합니다.

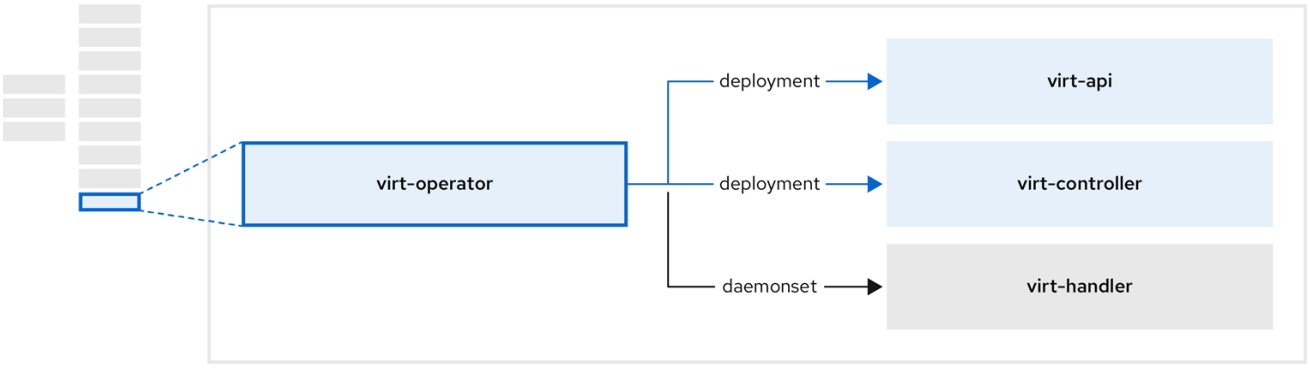


표 1.11. virt-operator 구성 요소

Component	설명
deployment/virt-api	모든 가상화 관련 흐름의 진입점 역할을 하는 HTTP API 서버입니다.
deployment/virt-controller	새 VM 인스턴스 오브젝트 생성을 관찰하고 해당 Pod를 생성합니다. 노드에 Pod가 예약되면 **virt-controller** 는 VM을 노드 이름으로 업데이트합니다.
daemonset/virt-handler	VM에 대한 모든 변경 사항을 모니터링하고 **virt-launcher** 에 필요한 작업을 수행하도록 지시합니다. 이 구성 요소는 노드에 따라 다릅니다.
pod/virt-launcher	**libvirt** 및 **qemu** 에서 구현한 사용자가 생성한 VM을 포함합니다.

2장. 릴리스 노트

2.1. OPENSHIFT VIRTUALIZATION 릴리스 정보

2.1.1. 문서 피드백 제공

오류를 보고하거나 문서를 개선하기 위해 Red Hat Jira 계정 에 로그인하여 Jira 문제를 제출하십시오.

2.1.2. Red Hat OpenShift Virtualization 정보

Red Hat OpenShift Virtualization을 사용하면 기존 VM(가상 머신)을 OpenShift Container Platform에 가져와서 컨테이너와 함께 실행할 수 있습니다. OpenShift Virtualization에서 VM은 OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 관리할 수 있는 기본 Kubernetes 오브젝트입니다.

OpenShift Virtualization은  아이콘으로 표시됩니다.

OVN-Kubernetes 또는 OpenShiftSDN 기본 CNI(Container Network Interface) 네트워크 공급자와 함께 OpenShift Virtualization을 사용할 수 있습니다.

OpenShift Virtualization으로 수행할 수 있는 작업에 대해 자세히 알아보십시오.

OpenShift Virtualization 아키텍처 및 배포에 대해 자세히 알아보십시오.

OpenShift Virtualization을 위한 클러스터를 준비합니다.

2.1.2.1. OpenShift Virtualization 지원 클러스터 버전

OpenShift Container Platform 4.16 클러스터에서 사용할 수 있도록 OpenShift Virtualization 4.16이 지원됩니다. OpenShift Virtualization의 최신 z-stream 릴리스를 사용하려면 먼저 OpenShift Container Platform의 최신 버전으로 업그레이드해야 합니다.

2.1.2.2. 지원되는 게스트 운영 체제

OpenShift Virtualization에서 지원되는 게스트 운영 체제를 보려면 Certified Guest Operating Systems in Red Hat OpenStack Platform, Red Hat Virtualization, OpenShift Virtualization 및 Red Hat Enterprise Linux with KVM 을 참조하십시오.

2.1.2.3. Microsoft Windows SVVP 인증

OpenShift Virtualization은 Microsoft의 Windows SVVP(서버 가상화 유효성 검사 프로그램)에서 Windows Server 워크로드를 실행하도록 인증되었습니다.

SVVP 인증은 다음에 적용됩니다.

- Red Hat Enterprise Linux CoreOS 작업자. SVVP 카탈로그에서는 *RHEL CoreOS 9의 Red Hat OpenShift Container Platform 4*로 이름이 지정됩니다.
- Intel 및 AMD CPU

2.1.3. 퀵스타트

여러 OpenShift Virtualization 기능에 대한 퀵스타트 둘러보기를 사용할 수 있습니다. 둘러보기를 보려면 OpenShift Container Platform 웹 콘솔의 헤더에 있는 메뉴 표시줄에서 **도움말** 아이콘 **?** 을 클릭한 다음 **빠른 시작을 선택합니다**. 필터 필드에 키워드 **가상화를** 입력하여 사용 가능한 둘러보기를 **필터링** 할 수 있습니다.

2.1.4. 새로운 기능 및 변경된 기능

이 릴리스에는 다음 구성 요소 및 개념과 관련된 새로운 기능 및 개선 사항이 추가되었습니다.

2.1.4.1. 설치 및 업데이트

- OpenShift Virtualization 4.16으로 업그레이드한 후 이전에 가비지 컬렉션을 통해 제거된 데이터 볼륨이 다시 생성될 수 있습니다. 이 동작이 예상됩니다. 데이터 볼륨 가비지 컬렉션이 비활성화되었으므로 다시 생성된 데이터 볼륨을 무시할 수 있습니다.

2.1.4.2. 가상화

- Windows 10 VM은 이제 TPM과 함께 UEFI를 사용하여 부팅됩니다.
- **AutoResourceLimits** 기능 게이트를 활성화하면 VM의 CPU 및 메모리 제한을 자동으로 관리합니다.
- KubeVirt Tekton 작업이 OpenShift Container Platform Pipelines 카탈로그 의 일부로 제공됩니다.

2.1.4.3. 네트워킹

- 이제 헤드리스 서비스를 사용하여 FQDN(정규화된 도메인 이름)의 기본 내부 pod 네트워크에 연결된 VM에 액세스 할 수 있습니다.

2.1.4.4. 웹 콘솔

- 이제 가상 머신을 핫플러그할 수 있습니다. 가상 머신을 핫 플러그로 연결할 수 없는 경우 VM에 **RestartRequired** 조건이 적용됩니다. 이 조건은 웹 콘솔의 **탭에서** 볼 수 있습니다.
- 이제 인스턴스 유형에서 Microsoft Windows VM을 생성할 때 **sysprep** 옵션을 선택할 수 있습니다. 이전에는 VM 생성 후 사용자 지정으로 **sysprep** 옵션을 설정해야 했습니다.

2.1.4.5. 모니터링

- 관리자는 **downwardMetrics** 기능 게이트를 활성화하고 Downward **Metrics** 장치를 구성하여 OpenShift Virtualization 의 virtio-serial 포트를 통해 제한된 호스트 및 VM(가상 머신) 메트릭을 게스트 VM에 노출 할 수 있습니다. 사용자는 **vm-dump-metrics** 툴을 사용하거나 명령줄에서 지표를 검색합니다.
  RHEL(Red Hat Enterprise Linux) 9에서 명령줄을 사용하여 하향 메트릭을 확인합니다. **vm-dump-metrics** 툴은 RHEL(Red Hat Enterprise Linux) 9 플랫폼에서 지원되지 않습니다.

2.1.4.6. 주요 기술 변경 사항

- VM에는 메모리 핫플러그를 활성화하기 위해 최소 1GiB의 할당된 메모리가 필요합니다. VM에 1GiB 미만의 할당된 메모리가 있는 경우 메모리 핫플러그가 비활성화됩니다.
- OpenShift Virtualization 경고용 Runbook은 이제 **openshift/runbooks** git 리포지토리에서 만 유지됩니다. 이제 제거된 runbooks 대신 runbook 소스 파일에 대한 링크를 사용할 수 있습니다.

2.1.5. 사용되지 않거나 삭제된 기능

2.1.5.1. 더 이상 사용되지 않는 기능

더 이상 사용되지 않는 기능은 현재 릴리스에 포함되어 있으며 계속 지원됩니다. 그러나 더 이상 사용되지 않는 기능은 향후 릴리스에서 제거되며 새 배포에는 권장되지 않습니다.

- **tekton-tasks-operator** 는 더 이상 사용되지 않으며 Tekton 작업과 예제 파이프라인은 이제 **ssp-operator** 에서 배포합니다.
- **copy-template,modify-vm-template** 및 **create-vm-from-template** 작업은 더 이상 사용되지 않습니다.
- Windows Server 2012 R2 템플릿에 대한 지원은 더 이상 사용되지 않습니다.
- 경고 **KubeVirtComponentExceedsRequestedMemory** 및 **KubeVirtComponentExceedsRequestedCPU** 는 더 이상 사용되지 않습니다. 안전하게 음소거 할 수 있습니다.

2.1.5.2. 삭제된 기능

제거된 기능은 현재 릴리스에서 지원되지 않습니다.

2.1.6. 기술 프리뷰 기능

이 릴리스의 일부 기능은 현재 기술 프리뷰 단계에 있습니다. 이러한 실험적 기능은 프로덕션용이 아닙니다. 해당 기능은 Red Hat Customer Portal의 지원 범위를 참조하십시오.

기술 프리뷰 기능 지원 범위

- 이제 전체 클러스터에 대한 VM 제거 전략을 구성할 수 있습니다.
- 이제 OpenShift Virtualization 호스트에서 중첩된 가상화를 활성화할 수 있습니다.
- 클러스터 관리자는 이제 **개요 → 설정 → 프리뷰 기능에 있는 OpenShift Container Platform 웹 콘솔의 네임스페이스에서 CPU 리소스 제한을 활성화할 수 있습니다**.
- 클러스터 관리자는 이제 **wasp-agent** 툴을 사용하여 메모리 양을 과다 할당, RAM에서 스왑 리소스를 VM 워크로드에 할당하여 클러스터에서 더 높은 VM 워크로드 밀도를 구성할 수 있습니다.
- OpenShift Virtualization은 이제 Red Hat OpenShift Data Foundation (ODF) Regional Disaster Recovery와의 호환성을 지원합니다.

2.1.7. 확인된 문제

모니터링

- Pod 중단 예산(PDB)을 사용하면 migratable 가상 머신 이미지에 대한 Pod 중단을 방지할 수 있습니다. PDB에서 Pod 중단을 감지하면 **openshift-monitoring** 에서 **LiveMigrate** 제거 전략을 사용하는 가상 머신 이미지에 대해 60분마다 **PodDisruptionBudgetAtLimit** 경고를 보냅니다. (CNV-33834)
  - 해결 방법으로 경고는 음소거 합니다.

노드

- OpenShift Virtualization을 설치 제거해도 OpenShift Virtualization에서 생성한 **feature.node.kubevirt.io** 노드 레이블이 제거되지 않습니다. 레이블을 수동으로 제거해야 합니다. (CNV-38543)

스토리지

- AWS에서 Portworx를 스토리지 솔루션으로 사용하고 VM 디스크 이미지를 생성하는 경우 두 번 고려되는 파일 시스템 오버헤드로 인해 생성된 이미지가 예상보다 작을 수 있습니다. (CNV-40217)
  - 이 문제를 해결하려면 초기 프로비저닝 프로세스가 완료된 후 PVC(영구 볼륨 클레임)를 수동으로 확장하여 사용 가능한 공간을 늘릴 수 있습니다.
- 경우에 따라 여러 가상 머신이 읽기-쓰기 모드로 동일한 PVC를 마운트할 수 있으므로 데이터가 손상될 수 있습니다. (CNV-13500)
  - 이 문제를 해결하려면 여러 VM이 있는 읽기-쓰기 모드에서 단일 PVC를 사용하지 마십시오.
- **csi-clone** 복제 전략을 사용하여 100개 이상의 VM을 복제하면 Ceph CSI에서 복제본을 제거하지 못할 수 있습니다. 복제를 수동으로 삭제하는 경우에도 실패할 수 있습니다. (CNV-23501)
  - 이 문제를 해결하려면 **ceph-mgr** 을 다시 시작하여 VM 복제를 제거할 수 있습니다.

가상화

- CPU 유형이 혼합된 클러스터에서 VM 마이그레이션이 실패할 수 있습니다. (CNV-43195)
  - 이 문제를 해결하려면 VM 사양 수준 또는 클러스터 수준에서 CPU 모델을 설정할 수 있습니다.
- Windows VM에 vTPM(가상 신뢰할 수 있는 플랫폼 모듈) 장치를 추가하면 vTPM 장치가 영구적이지 않은 경우에도 BitLocker 드라이브 암호화 시스템 검사가 통과됩니다. 이는 영구 저장소가 아닌 vTPM 장치는 **virt-launcher** Pod의 수명 동안 임시 스토리지를 사용하여 암호화 키를 복구하기 때문입니다. VM이 마이그레이션되거나 종료되면 vTPM 데이터가 손실됩니다. (CNV-36448)
- OpenShift Virtualization은 Pod에서 사용하는 서비스 계정 토큰을 해당 특정 Pod에 연결합니다. OpenShift Virtualization은 토큰이 포함된 디스크 이미지를 생성하여 서비스 계정 볼륨을 구현합니다. VM을 마이그레이션하면 서비스 계정 볼륨이 유효하지 않습니다. (CNV-33835)
  - 이 문제를 해결하려면 사용자 계정 토큰이 특정 Pod에 바인딩되지 않으므로 서비스 계정 대신 사용자 계정을 사용합니다.
- RHSA-2023:3722 권고가 릴리스되면서 TLS **확장 마스터 시크릿** (ECDSA) 확장(RFC 7627)은 FIPS 지원 Red Hat Enterprise Linux (RHEL) 9 시스템에서 TLS 1.2 연결에 필요합니다. 이는 FIPS-140-3 요구 사항에 따라 수행됩니다. TLS 1.3은 영향을 받지 않습니다.
  ECDSA 또는 TLS 1.3을 지원하지 않는 기존 OpenSSL 클라이언트는 이제 RHEL 9에서 실행되는 FIPS 서버에 연결할 수 없습니다. 마찬가지로 FIPS 모드의 RHEL 9 클라이언트는 ECDSA 없이 TLS 1.2만 지원하는 서버에 연결할 수 없습니다. 실제로 이러한 클라이언트는 RHEL 6, RHEL 7 및 비 RHEL 레거시 운영 체제의 서버에 연결할 수 없습니다. 이는 OpenSSL의 기존 1.0.x 버전이 ECDSA 또는 TLS 1.3을 지원하지 않기 때문입니다. 자세한 내용은 Red Hat Enterprise Linux 9.2에서 적용된 TLS 확장 "확장 마스터 시크릿" 을 참조하십시오.
  - 이 문제를 해결하려면 기존 OpenSSL 클라이언트를 TLS 1.3을 지원하는 버전으로 업데이트하고 FIPS 모드의 경우 **Modern** TLS 보안 프로필 유형으로 TLS 1.3을 사용하도록 OpenShift Virtualization을 구성합니다.

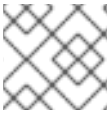
웹 콘솔

- OpenShift Container Platform 클러스터를 처음 배포할 때 웹 콘솔을 사용하여 템플릿 또는 인스턴스 유형에서 VM을 생성하면 **cluster-admin** 권한이 없는 경우 실패합니다.
  - 이 문제를 해결하려면 클러스터 관리자가 먼저 다른 사용자가 템플릿과 인스턴스 유형을 사용하여 VM을 생성할 수 있도록 구성 맵 을 생성해야 합니다. (CNV-38284)
- 웹 콘솔의 영구 볼륨 클레임 **생성 목록에서 데이터 업로드 양식 사용을 선택하여 PVC(영구볼륨 클레임)를** 생성하면 **Upload Data** (데이터 업로드) 필드를 사용하여 PVC에 데이터를 업로드하지 못합니다. (CNV-37607)

3장. 시작하기

3.1. OPENSHIFT VIRTUALIZATION 시작하기

기본 환경을 설치하고 구성하여 OpenShift Virtualization의 기능 및 기능을 탐색할 수 있습니다.



참고

클러스터 구성 절차에는 **cluster-admin** 권한이 필요합니다.

3.1.1. OpenShift Virtualization 계획 및 설치

OpenShift Container Platform 클러스터에 OpenShift Virtualization을 계획하고 설치합니다.

- OpenShift Virtualization용 베어 메탈 클러스터를 계획합니다.
- OpenShift Virtualization을 위한 클러스터를 준비합니다.
- OpenShift Virtualization Operator를 설치합니다.
- **virtctl** CLI(명령줄 인터페이스) 툴을 설치합니다.

계획 및 설치 리소스

- 가상 머신 디스크의 스토리지 볼륨 정보.
- CSI 지원 스토리지 공급자 사용
- 가상 머신의 로컬 스토리지 구성.
- Kubernetes NMState Operator 설치
- 가상 머신의 노드 지정.
- **virtctl** 명령.

3.1.2. 가상 머신 생성 및 관리

VM(가상 머신)을 생성합니다.

- Red Hat 이미지에서 VM을 생성합니다.
  Red Hat 템플릿 또는 인스턴스 유형을 사용하여 VM을 생성할 수 있습니다.
- 사용자 지정 이미지에서 VM을 생성합니다.
  컨테이너 레지스트리 또는 웹 페이지에서 사용자 정의 이미지를 가져오거나, 로컬 머신에서 이미지를 업로드하거나 PVC(영구 볼륨 클레임)를 복제하여 VM을 생성할 수 있습니다.

VM을 보조 네트워크에 연결합니다.

- Linux 브리지 네트워크.
- OVN(가상 네트워크)-Kubernetes 보조 네트워크.
- SR-IOV(Single Root I/O Virtualization) 네트워크.

참고

VM은 기본적으로 Pod 네트워크에 연결됩니다.

VM에 연결합니다.

- VM의 직렬 콘솔 또는 VNC 콘솔에 연결합니다.
- SSH를 사용하여 VM에 연결합니다.
- Windows VM의 데스크탑 뷰어에 연결합니다.

VM을 관리합니다.

- 웹 콘솔을 사용하여 VM을 관리합니다.
- **virtctl** CLI 툴을 사용하여 VM을 관리합니다.
- VM 내보내기.

3.1.3. 다음 단계

- 설치 후 구성 옵션을 검토합니다.
- 스토리지 옵션 및 자동 부팅 소스 업데이트를 구성합니다.
- 모니터링 및 상태 점검에 대해 알아보십시오.
- 실시간 마이그레이션에 대해 알아보기
- OADP(OpenShift API for Data Protection)를 사용하여 VM을 백업하고 복원합니다.
- 클러스터를 튜닝하고 확장합니다.

3.2. CLI 툴 사용

virtctl 명령줄 툴을 사용하여 OpenShift Virtualization 리소스를 관리할 수 있습니다.

libguestfs 명령줄 툴을 사용하여 VM(가상 머신) 디스크 이미지에 액세스하고 수정할 수 있습니다. **virtctl libguestfs** 명령을 사용하여 **libguestfs**를 배포합니다.

3.2.1. virtctl 설치

RHEL(Red Hat Enterprise Linux) 9, Linux, Windows 및 MacOS 운영 체제에 **virtctl** 을 설치하려면 **virtctl** 바이너리 파일을 다운로드하여 설치합니다.

RHEL 8에 **virtctl** 을 설치하려면 OpenShift Virtualization 리포지토리를 활성화한 다음 **kubevirt-virtctl** 패키지를 설치합니다.

3.2.1.1. RHEL 9, Linux, Windows 또는 macOS에 virtctl 바이너리 설치

OpenShift Container Platform 웹 콘솔에서 운영 체제의 **virtctl** 바이너리를 다운로드한 다음 설치할 수 있습니다.

프로세스

1. 웹 콘솔의 **가상화 → 개요** 페이지로 이동합니다.
2. **Download virtctl** 링크를 클릭하여 운영 체제의 **virtctl** 바이너리를 다운로드합니다.
3. install **virtctl**:
   - RHEL 9 및 기타 Linux 운영 체제의 경우:
     a. 아카이브 파일의 압축을 풉니다.

     ```
     $ tar -xvf <virtctl-version-distribution.arch>.tar.gz
     ```

     b. 다음 명령을 실행하여 **virtctl** 바이너리를 실행할 수 있도록 합니다.

     ```
     $ chmod +x <path/virtctl-file-name>
     ```

     c. **virtctl** 바이너리를 **PATH** 환경 변수의 디렉터리로 이동합니다.
     다음 명령을 실행하여 경로를 확인할 수 있습니다.

     ```
     $ echo $PATH
     ```

     d. **KUBECONFIG** 환경 변수를 설정합니다.

     ```
     $ export KUBECONFIG=/home/<user>/clusters/current/auth/kubeconfig
     ```

   - Windows의 경우:
     a. 아카이브 파일의 압축을 풉니다.
     b. 추출된 폴더 계층 구조로 이동하고 **virtctl** 실행 파일을 두 번 클릭하여 클라이언트를 설치합니다.
     c. **virtctl** 바이너리를 **PATH** 환경 변수의 디렉터리로 이동합니다.
     다음 명령을 실행하여 경로를 확인할 수 있습니다.

     ```
     C:\> path
     ```

   - macOS의 경우:
     a. 아카이브 파일의 압축을 풉니다.
     b. **virtctl** 바이너리를 **PATH** 환경 변수의 디렉터리로 이동합니다.
     다음 명령을 실행하여 경로를 확인할 수 있습니다.

     ```
     echo $PATH
     ```

3.2.1.2. RHEL 8에 virtctl RPM 설치

OpenShift Virtualization 리포지토리를 활성화하고 **kubevirt-virtctl** 패키지를 설치하여 RHEL(Red Hat Enterprise Linux) 8에 **virtctl** RPM 패키지를 설치할 수 있습니다.

사전 요구 사항

- 클러스터의 각 호스트는 RHSM(Red Hat Subscription Manager)에 등록되어 있어야 하며 유효한 OpenShift Container Platform 서브스크립션이 있어야 합니다.

프로세스

1. **subscription-manager** CLI 툴을 사용하여 다음 명령을 실행하여 OpenShift Virtualization 리포지토리를 활성화합니다.

```
# subscription-manager repos --enable cnv-4.16-for-rhel-8-x86_64-rpms
```

2. 다음 명령을 실행하여 **kubevirt-virtctl** 패키지를 설치합니다.

```
# yum install kubevirt-virtctl
```

3.2.2. virtctl 명령

virtctl 클라이언트는 OpenShift Virtualization 리소스를 관리하는 명령줄 유틸리티입니다.

참고

VM(가상 머신) 명령은 별도로 지정하지 않는 한 VMI(가상 머신 인스턴스)에도 적용됩니다.

3.2.2.1. virtctl information 명령

virtctl information 명령을 사용하여 **virtctl** 클라이언트에 대한 정보를 봅니다.

표 3.1. 정보 명령

명령	설명
virtctl version	**virtctl** 클라이언트 및 서버 버전을 확인합니다.
virtctl help	**virtctl** 명령 목록을 확인합니다.
virtctl <command> -h|--help	특정 명령의 옵션 목록을 확인합니다.
virtctl 옵션	**virtctl** 명령에 대한 글로벌 명령 옵션 목록을 확인합니다.

3.2.2.2. VM 정보 명령

virtctl 을 사용하여 VM(가상 머신) 및 VMI(가상 머신 인스턴스)에 대한 정보를 볼 수 있습니다.

표 3.2. VM 정보 명령

명령	설명
virtctl fslist <vm_name>	게스트 머신에서 사용 가능한 파일 시스템을 확인합니다.
virtctl guestosinfo <vm_name>	게스트 머신의 운영 체제에 대한 정보를 봅니다.

명령	설명
virtctl userlist <vm_name>	게스트 머신에서 로그인한 사용자를 확인합니다.

3.2.2.3. VM 매니페스트 생성 명령

virtctl create 명령을 사용하여 가상 머신, 인스턴스 유형 및 기본 설정에 대한 매니페스트를 생성할 수 있습니다.

표 3.3. VM 매니페스트 생성 명령

명령	설명
virtctl create vm	**VirtualMachine** (VM) 매니페스트를 생성합니다.
virtctl create vm --name <vm_name>	VM 매니페스트를 생성하여 VM의 이름을 지정합니다.
virtctl create vm --instancetype <instancetype_name>	기존 클러스터 전체 인스턴스 유형을 사용하는 VM 매니페스트를 생성합니다.
virtctl create vm --instancetype=virtualmachineinstancetype/<instancetype_name>	기존 네임스페이스 인스턴스 유형을 사용하는 VM 매니페스트를 생성합니다.
virtctl create instancetype --cpu_value> --memory <memory_value> --name <instancetype_name>	클러스터 전체 인스턴스 유형에 대한 매니페스트를 생성합니다.
virtctl create instancetype --cpu_value> --memory <memory_value> --name <instancetype_name> --namespace <namespace_value>	네임스페이스가 지정된 인스턴스 유형에 대한 매니페스트를 생성합니다.
virtctl create preference --name <preference_name>	클러스터 전체 VM 기본 설정에 대한 매니페스트를 생성하여 기본 설정 이름을 지정합니다.
virtctl create preference --namespace <namespace_value>	네임스페이스가 지정된 VM 기본 설정에 대한 매니페스트를 생성합니다.

3.2.2.4. VM 관리 명령

virtctl VM(가상 머신) 관리 명령을 사용하여 VM(가상 머신) 및 VMI(가상 머신 인스턴스)를 관리하고 마이그레이션합니다.

표 3.4. VM 관리 명령

명령	설명
virtctl start <vm_name>	VM을 시작합니다.
virtctl start --paused <vm_name>	일시 중지된 상태에서 VM을 시작합니다. 이 옵션을 사용하면 VNC 콘솔에서 부팅 프로세스를 중단할 수 있습니다.
virtctl stop <vm_name>	VM을 중지합니다.
virtctl stop <vm_name> --grace-period 0 --force	VM을 강제로 중지합니다. 이 옵션을 사용하면 데이터 불일치 또는 데이터 손실이 발생할 수 있습니다.
virtctl pause vm <vm_name>	VM 일시 중지. 머신 상태는 메모리에 유지됩니다.
virtctl unpause vm <vm_name>	VM 일시 중지를 해제합니다.
virtctl migrate <vm_name>	VM 마이그레이션.
virtctl migrate-cancel <vm_name>	VM 마이그레이션을 취소합니다.
virtctl restart <vm_name>	VM을 다시 시작합니다.

3.2.2.5. VM 연결 명령

virtctl 연결 명령을 사용하여 포트를 노출하고 VM(가상 머신) 및 VMI(가상 머신 인스턴스)에 연결합니다.

표 3.5. VM 연결 명령

명령	설명
virtctl console <vm_name>	VM의 직렬 콘솔에 연결합니다.
virtctl expose vm <vm_name> --name <service_name> --type <ClusterIP|NodePort|LoadBalancer> --port <port>	VM의 지정된 포트를 전달하고 서비스를 노드의 지정된 포트에 노출하는 서비스를 생성합니다. 예: **virtctl expose vm rhel9_vm --name rhel9-ssh --type NodePort --port 22**
virtctl scp -i <ssh_key> <file_name> <user_name>@<vm_name>	시스템에서 VM으로 파일을 복사합니다. 이 명령은 SSH 키 쌍의 개인 키를 사용합니다. VM은 공개 키를 사용하여 구성해야 합니다.
virtctl scp -i <ssh_key> <user_name@<vm_name>: <file_name> .	VM에서 시스템으로 파일을 복사합니다. 이 명령은 SSH 키 쌍의 개인 키를 사용합니다. VM은 공개 키를 사용하여 구성해야 합니다.

명령	설명
virtctl ssh -i <ssh_key> <user_name>@<vm_name>	VM을 사용하여 SSH 연결을 엽니다. 이 명령은 SSH 키 쌍의 개인 키를 사용합니다. VM은 공개 키를 사용하여 구성해야 합니다.
virtctl vnc <vm_name>	VM의 VNC 콘솔에 연결합니다. **virt-viewer** 가 설치되어 있어야 합니다.
virtctl vnc --proxy-only=true <vm_name>	포트 번호를 표시하고 VNC 연결을 통해 뷰어를 사용하여 VM에 수동으로 연결합니다.
virtctl vnc --port=<port-number> <vm_name>	해당 포트를 사용할 수 있는 경우 지정된 포트에서 프록시를 실행할 포트 번호를 지정합니다. 포트 번호를 지정하지 않으면 프록시는 임의의 포트에서 실행됩니다.

3.2.2.6. VM 내보내기 명령

virtctl vmexport 명령을 사용하여 VM, VM 스냅샷 또는 PVC(영구 볼륨 클레임)에서 내보낸 볼륨을 생성, 다운로드 또는 삭제합니다. 특정 매니페스트에는 OpenShift Virtualization에서 사용할 수 있는 형식으로 디스크 이미지를 가져오기 위해 엔드포인트에 대한 액세스 권한을 부여하는 헤더 시크릿도 포함되어 있습니다.

표 3.6. VM 내보내기 명령

명령	설명
virtctl vmexport create <vmexport_name> --vm|snapshot|pvc= <object_name>	**VirtualMachineExport** CR(사용자 정의 리소스)을 생성하여 VM, VM 스냅샷 또는 PVC에서 볼륨을 내보냅니다. ● **--** VM : VM의 PVC를 내보냅니다. ● **--snapshot**: **VirtualMachineSnapshot** CR에 포함된 PVC를 내보냅니다. ● **--PVC**: PVC를 내보냅니다. ● 선택 사항: **--ttl=1h** 는 실시간 시간을 지정합니다. 기본 기간은 2시간입니다.
virtctl vmexport delete <vmexport_name>	**VirtualMachineExport** CR을 수동으로 삭제합니다.

명령	설명
virtctl vmexport download <vmexport_name> --output= <output_file> --volume= <volume_name>	**VirtualMachineExport** CR에 정의된 볼륨을 다운로드합니다. • **--output** 은 파일 형식을 지정합니다. 예:**disk.img.gz**. • **--volume** 은 다운로드할 볼륨을 지정합니다. 하나의 볼륨만 사용할 수 있는 경우 이 플래그는 선택 사항입니다. 선택 사항: • **--keep-vme** 는 다운로드 후 **VirtualMachineExport** CR을 유지합니다. 기본 동작은 다운로드 후 **VirtualMachineExport** CR을 삭제하는 것입니다. • **--insecure** 는 비보안 HTTP 연결을 활성화합니다.
virtctl vmexport download <vmexport_name> -- <vm|snapshot|pvc>= <object_name> --output= <output_file> --volume= <volume_name>	**VirtualMachineExport** CR을 생성한 다음 CR에 정의된 볼륨을 다운로드합니다.
virtctl vmexport 다운로드 내보내기 --manifest	기존 내보내기에 대한 매니페스트를 검색합니다. 매니페스트에는 헤더 보안이 포함되지 않습니다.
virtctl vmexport download export --manifest -- vm=example	VM 예제에 대한 VM 내보내기를 생성하고 매니페스트를 검색합니다. 매니페스트에는 헤더 보안이 포함되지 않습니다.
virtctl vmexport download export --manifest -- snap=example	VM 스냅샷 예에 대한 VM 내보내기를 생성하고 매니페스트를 검색합니다. 매니페스트에는 헤더 보안이 포함되지 않습니다.
virtctl vmexport download export --manifest --include-secret	기존 내보내기에 대한 매니페스트를 검색합니다. 매니페스트에는 헤더 보안이 포함됩니다.
virtctl vmexport download export --manifest --manifest-output-format=json	기존 내보내기에 대한 매니페스트를 json 형식으로 검색합니다. 매니페스트에는 헤더 보안이 포함되지 않습니다.
virtctl vmexport download export --manifest --include-secret -- output=manifest.yaml	기존 내보내기에 대한 매니페스트를 검색합니다. 매니페스트에는 헤더 보안이 포함되어 있으며 지정된 파일에 씁니다.

3.2.2.7. VM 메모리 덤프 명령

virtctl memory-dump 명령을 사용하여 PVC에 VM 메모리 덤프를 출력할 수 있습니다. 기존 PVC를 지정하거나 **--create-claim** 플래그를 사용하여 새 PVC를 생성할 수 있습니다.

사전 요구 사항

- PVC 볼륨 모드는 **FileSystem** 이어야 합니다.
- PVC는 메모리 덤프를 포함할 수 있을 만큼 커야 합니다.
  PVC 크기를 계산하는 공식은 **(VMMemorySize + 100Mi) * FileSystemOverhead** 입니다. 여기서 **100Mi** 는 메모리 덤프 오버헤드입니다.
- 다음 명령을 실행하여 **HyperConverged** 사용자 정의 리소스에서 핫 플러그 기능 게이트를 활성화해야 합니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "add", "path": "/spec/featureGates", \
  "value": "HotplugVolumes"}]'
```

메모리 덤프 다운로드

virtctl vmexport download 명령을 사용하여 메모리 덤프를 다운로드해야 합니다.

```
$ virtctl vmexport download <vmexport_name> --vm|pvc=<object_name> \
  --volume=<volume_name> --output=<output_file>
```

표 3.7. VM 메모리 덤프 명령

명령	설명

virtctl memory-dump get <vm_name> --claim-name= <pvc_name>	VM의 메모리 덤프를 PVC에 저장합니다. 메모리 덤프 상태가 **VirtualMachine** 리소스의 **status** 섹션에 표시됩니다.
	
	선택 사항:
	• **--create-claim** 은 적절한 크기로 새 PVC를 생성합니다. 이 플래그에는 다음과 같은 옵션이 있습니다.
	  ◦ **--storage-class=<storage_class&** gt; : PVC의 스토리지 클래스를 지정합니다.
	  ◦ **--access-mode=<access_mode&gt**; : **ReadWriteOnce** 또는 **ReadWriteMany** 를 지정합니다.

virtctl memory-dump get <vm_name>	동일한 PVC를 사용하여 **virtctl memory-dump** 명령을 재실행합니다. 이 명령은 이전 메모리 덤프를 덮어씁니다.
virtctl memory-dump remove <vm_name>	메모리 덤프 제거. 대상 PVC를 변경하려면 메모리 덤프를 수동으로 제거해야 합니다. 이 명령은 VM과 PVC 간의 연결을 제거하여 메모리 덤프가 **VirtualMachine** 리소스의 **status** 섹션에 표시되지 않습니다. PVC에는 영향을 미치지 않습니다.

3.2.2.8. 핫플러그 및 핫 플러그 해제 명령

virtctl 을 사용하여 실행 중인 VM(가상 머신) 및 VMI(가상 머신 인스턴스)에서 리소스를 추가하거나 제거합니다.

표 3.8. 핫플러그 및 핫 플러그 해제 명령

명령	설명
virtctl addvolume <vm_name> --volume-name= <datavolume_or_PVC> [--persist] [--serial=<label>]	데이터 볼륨 또는 PVC(영구 볼륨 클레임)를 핫플러그합니다. 선택 사항: ● **--persist** 는 VM에 가상 디스크를 영구적으로 마운트합니다.**이 플래그는 VMI에는 적용되지 않습니다.** ● **--serial=<label&** gt;은 VM에 레이블을 추가합니다. 라벨을 지정하지 않으면 default 레이블은 데이터 볼륨 또는 PVC 이름입니다.
virtctl removevolume <vm_name> --volume-name=<virtual_disk>	가상 디스크 핫 플러그를 해제합니다.
virtctl addinterface <vm_name> --network-attachment-definition-name <net_attach_def_name> --name <interface_name>	Linux 브리지 네트워크 인터페이스를 핫 플러그로 연결합니다.
virtctl removeinterface <vm_name> --name <interface_name>	Linux 브리지 네트워크 인터페이스를 핫 플러그 해제합니다.

3.2.2.9. 이미지 업로드 명령

virtctl image-upload 명령을 사용하여 VM 이미지를 데이터 볼륨에 업로드합니다.

표 3.9. 이미지 업로드 명령

명령	설명
virtctl image-upload dv <datavolume_name> --image-path= </path/to/image> --no-create	VM 이미지를 이미 존재하는 데이터 볼륨에 업로드합니다.
virtctl image-upload dv <datavolume_name> --size= <datavolume_size> --image-path=</path/to/image>	요청된 크기의 새 데이터 볼륨에 VM 이미지를 업로드합니다.

3.2.3. virtctl을 사용하여 libguestfs 배포

virtctl guestfs 명령을 사용하여 **libguestfs-tools** 및 연결된 PVC(영구 볼륨 클레임)를 사용하여 대화형 컨테이너를 배포할 수 있습니다.

절차

- **libguestfs-tools**를 사용하여 컨테이너를 배포하려면 PVC를 마운트하고 쉘을 연결하려면 다음 명령을 실행합니다.

```
$ virtctl guestfs -n <namespace> <pvc_name> 1
```

1 PVC 이름은 필수 인수입니다. 이를 포함하지 않으면 오류 메시지가 표시됩니다.

3.2.3.1. libguestfs 및 virtctl guestfs 명령

Libguestfs 툴을 사용하면 VM(가상 머신) 디스크 이미지에 액세스하고 수정할 수 있습니다. **libguestfs** 툴을 사용하여 게스트의 파일을 보고 편집하고, 가상 시스템을 복제 및 빌드하며, 디스크를 포맷하고 크기를 조정할 수 있습니다.

virtctl guestfs 명령과 해당 하위 명령을 사용하여 PVC에서 VM 디스크를 수정, 검사 및 디버깅할 수도 있습니다. 가능한 하위 명령의 전체 목록을 보려면 명령줄에 **virt-** 을 입력하고 Tab 키를 누릅니다. 예를 들면 다음과 같습니다.

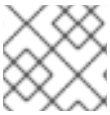
명령	설명
virt-edit -a /dev/vda /etc/motd	터미널에서 파일을 대화식으로 편집합니다.
virt-customize -a /dev/vda --ssh-inject root:string:<public key example>	ssh 키를 게스트에 삽입하고 로그인을 만듭니다.
virt-df -a /dev/vda -h	VM에서 사용하는 디스크 공간 크기를 확인하십시오.
virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'	전체 목록이 포함된 출력 파일을 생성하여 게스트에 설치된 모든 RPM의 전체 목록을 확인하십시오.
virt-cat -a /dev/vda /rpm-list	터미널에서 **virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'** 명령을 사용하여 생성된 모든 RPM의 출력 파일 목록을 표시합니다.
virt-sysprep -a /dev/vda	템플릿으로 사용할 가상 시스템 디스크 이미지를 봉인합니다.

기본적으로 **virtctl guestfs** 는 VM 디스크를 관리하는 데 필요한 모든 내용으로 세션을 생성합니다. 그러나 이 명령은 동작을 사용자 지정하려는 경우 여러 플래그 옵션도 지원합니다.

플래그 옵션	설명
--h 또는 **--help**	**guestfs**에 대한 도움말을 제공합니다.

플래그 옵션	설명
<pvc_name> 인수가 있는 **-n <namespace>** 옵션	특정 네임스페이스에서 PVC를 사용하려면 다음을 수행합니다. **-n <namespace>** 옵션을 사용하지 않는 경우 현재 프로젝트가 사용됩니다. 프로젝트를 변경하려면 **oc project <namespace>**를 사용합니다. **<pvc_name>** 인수를 포함하지 않으면 오류 메시지가 표시됩니다.
--image string	**libguestfs-tools** 컨테이너 이미지를 나열합니다. **--image** 옵션을 사용하여 사용자 지정 이미지를 사용하도록 컨테이너를 구성할 수 있습니다.
--kvm	**libguestfs-tools** 컨테이너에서 **kvm**이 사용됨을 나타냅니다. 기본적으로 **virtctl guestfs**는 대화형 컨테이너에 대해 **kvm**을 설정하므로 QEMU를 사용하기 때문에 **libguest-tools** 실행 속도가 훨씬 빨라집니다. 클러스터에 **kvm** 지원 노드가 없는 경우 **--kvm=false** 옵션을 설정하여 **kvm**을 비활성화해야 합니다. 설정되지 않은 경우 **libguestfs-tools** Pod는 모든 노드에서 예약할 수 없으므로 보류 중으로 유지됩니다.
--pull-policy string	**libguestfs** 이미지의 가져오기 정책을 표시합니다. **pull-policy** 옵션을 설정하여 이미지의 가져오기 정책을 덮어쓸 수도 있습니다.

또한 명령은 다른 pod에서 PVC를 사용 중인지 확인합니다. 이 경우 오류 메시지가 표시됩니다. 그러나 **libguestfs-tools** 프로세스가 시작되면 동일한 PVC를 사용하는 새 Pod를 방지할 수 없습니다. 동일한 PVC에 액세스하는 VM을 시작하기 전에 활성 **virtctl guestfs** Pod가 없는지 확인해야 합니다.



참고

virtctl guestfs 명령은 대화형 Pod에 연결된 단일 PVC만 허용합니다.

3.2.4. Ansible 사용

OpenShift Virtualization에 Ansible 컬렉션을 사용하려면 Red Hat Ansible Automation Hub (Red Hat Hybrid Cloud Console)를 참조하십시오.

4장. 설치

4.1. OPENSHIFT VIRTUALIZATION을 위한 클러스터 준비

OpenShift Virtualization을 설치하기 전에 이 섹션을 검토하여 클러스터가 요구 사항을 충족하는지 확인합니다.

중요

설치 방법 고려 사항

사용자 프로비저닝, 설치 관리자 프로비저닝 또는 지원 설치 프로그램을 포함하여 모든 설치 방법을 사용하여 OpenShift Container Platform을 배포할 수 있습니다. 그러나 설치 방법과 클러스터 토폴로지는 스냅샷 또는 실시간 마이그레이션 과 같은 OpenShift Virtualization 기능에 영향을 미칠 수 있습니다.

Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation을 사용하여 OpenShift Virtualization을 배포하는 경우 Windows 가상 머신 디스크용 전용 스토리지 클래스를 생성해야 합니다. 자세한 내용은 Windows VM용 ODF PersistentVolume 최적화 를 참조하십시오.

IPv6

단일 스택 IPv6 클러스터에서 OpenShift Virtualization을 실행할 수 없습니다.

FIPS 모드

FIPS 모드에서 클러스터를 설치하는 경우 OpenShift Virtualization에 추가 설정이 필요하지 않습니다.

4.1.1. 지원되는 플랫폼

OpenShift Virtualization에서 다음 플랫폼을 사용할 수 있습니다.

- 온프레미스 베어 메탈 서버. OpenShift Virtualization용 베어 메탈 클러스터 계획을 참조하십시오.
- Amazon Web Services 베어 메탈 인스턴스. 사용자 지정을 사용하여 AWS에 클러스터 설치를 참조하십시오.
- IBM Cloud® 베어 메탈 서버. Deploy OpenShift Virtualization on IBM Cloud® Bare Metal 노드를 참조하십시오.

중요

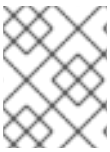
IBM Cloud® Bare Metal Server에 OpenShift Virtualization을 설치하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

다른 클라우드 공급자가 제공하는 베어 메탈 인스턴스 또는 서버는 지원되지 않습니다.

4.1.1.1. AWS 베어 메탈의 OpenShift Virtualization

AWS(Amazon Web Services) 베어 메탈 OpenShift Container Platform 클러스터에서 OpenShift Virtualization을 실행할 수 있습니다.



참고

OpenShift Virtualization은 AWS 베어 메탈 클러스터와 동일한 구성 요구 사항이 있는 AWS(ROSA) 클래식 클러스터에서도 지원됩니다.

클러스터를 설정하기 전에 지원되는 기능 및 제한 사항에 대한 다음 요약을 검토하십시오.

설치

- **install-config.yaml** 파일을 편집하여 작업자 노드의 베어 메탈 인스턴스 유형을 지정하도록 설치 관리자 프로비저닝 인프라를 사용하여 클러스터를 설치할 수 있습니다. 예를 들어 x86_64 아키텍처를 기반으로 하는 머신에 **c5n.metal** 유형 값을 사용할 수 있습니다.
  자세한 내용은 AWS에 설치하는 방법에 대한 OpenShift Container Platform 설명서를 참조하십시오.

VM(가상 머신) 액세스

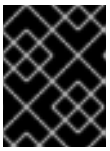
- **virtctl** CLI 도구 또는 OpenShift Container Platform 웹 콘솔을 사용하여 VM에 액세스하는 방법은 변경되지 않습니다.
- **NodePort** 또는 **LoadBalancer** 서비스를 사용하여 VM을 노출할 수 있습니다.
  - OpenShift Container Platform이 AWS에서 로드 밸런서를 자동으로 생성하고 라이프사이클을 관리하므로 로드 밸런서 접근 방식을 사용하는 것이 좋습니다. 로드 밸런서에 대한 보안 그룹도 생성되며, 주석을 사용하여 기존 보안 그룹을 연결할 수 있습니다. 서비스를 제거하면 OpenShift Container Platform에서 로드 밸런서 및 관련 리소스를 제거합니다.

네트워킹

- VLAN(Virtual LAN)을 포함하여 SR-IOV(Single Root I/O Virtualization) 또는 브리지 CNI(Container Network Interface) 네트워크를 사용할 수 없습니다. 애플리케이션에 플랫 계층 2 네트워크 또는 IP 풀을 제어해야 하는 경우 OVN-Kubernetes 보조 오버레이 네트워크를 사용하는 것이 좋습니다.

스토리지

- 스토리지 벤더가 인증된 모든 스토리지 솔루션을 사용하여 기본 플랫폼으로 작업할 수 있습니다.



중요

AWS 베어 메탈 및 ROSA 클러스터에는 다른 지원되는 스토리지 솔루션이 있을 수 있습니다. 스토리지 벤더에 대한 지원을 확인하십시오.

- OpenShift Virtualization과 함께 EBS(Amazon Elastic File System) 또는 Amazon Elastic Block Store(EBS)를 사용하면 성능 및 기능 제한이 발생할 수 있습니다. 실시간 마이그레이션, 빠른 VM 생성 및 VM 스냅샷 기능을 활성화하려면 RWX(ReadWriteMany), 복제 및 스냅샷을 지원하는 CSI 스토리지를 사용하는 것이 좋습니다.

호스트된 컨트롤 플레인(HCP)

- OpenShift Virtualization의 HCP는 현재 AWS 인프라에서 지원되지 않습니다.

추가 리소스

- 가상 머신을 OVN-Kubernetes 보조 네트워크에 연결
- 서비스를 사용하여 가상 머신 노출

4.1.2. 하드웨어 및 운영 체제 요구사항

OpenShift Virtualization에 대한 다음 하드웨어 및 운영 체제 요구 사항을 검토하십시오.

4.1.2.1. CPU 요구 사항

- RHEL(Red Hat Enterprise Linux) 9에서 지원합니다.
  지원되는 CPU는 Red Hat Ecosystem Catalog 에서 참조하십시오.

참고

작업자 노드에 CPU가 다른 경우 CPU마다 기능이 다르기 때문에 실시간 마이그레이션 오류가 발생할 수 있습니다. 작업자 노드에 적절한 용량이 있고 가상 머신에 대한 노드 유사성 규칙을 구성하여 이 문제를 완화할 수 있습니다.

자세한 내용은 필수 노드 유사성 규칙 구성 을 참조하십시오.

- AMD 및 Intel 64비트 아키텍처(x86-64-v2) 지원
- Intel 64 또는 AMD64 CPU 확장 지원
- Intel VT 또는 AMD-V 하드웨어 가상화 확장 기능이 활성화되어 있습니다.
- NX(실행 없음) 플래그가 활성화되어 있습니다.

4.1.2.2. 운영 체제 요구 사항

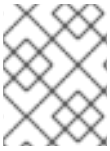
- 작업자 노드에 RHCOS(Red Hat Enterprise Linux CoreOS)가 설치되어 있습니다.
  자세한 내용은 RHCOS 정보를 참조하십시오.

참고

RHEL 작업자 노드는 지원되지 않습니다.

4.1.2.3. 스토리지 요구사항

- OpenShift Container Platform에서 지원합니다. 스토리지 최적화를 참조하십시오.
- 기본 OpenShift Virtualization 또는 OpenShift Container Platform 스토리지 클래스를 생성해야 합니다. 이 문제의 목적은 VM 워크로드의 고유한 스토리지 요구 사항을 해결하고 최적화된 성능, 안정성 및 사용자 환경을 제공하는 것입니다. OpenShift Virtualization 및 OpenShift Container Platform 기본 스토리지 클래스가 둘 다 있는 경우 VM 디스크를 생성할 때 OpenShift Virtualization 클래스가 우선합니다.



참고

스토리지 클래스를 가상화 워크로드의 기본값으로 표시하려면 **storageclass.kubevirt.io/is-default-virt-class** 를 **"true"** 로 설정합니다.

- 스토리지 프로비저너가 스냅샷을 지원하는 경우 **VolumeSnapshotClass** 오브젝트를 기본 스토리지 클래스와 연결해야 합니다.

4.1.2.3.1. 가상 머신 디스크의 볼륨 및 액세스 모드 정보

알려진 스토리지 공급자와 스토리지 API를 사용하는 경우 볼륨 및 액세스 모드가 자동으로 선택됩니다. 그러나 스토리지 프로필이 없는 스토리지 클래스를 사용하는 경우 볼륨 및 액세스 모드를 구성해야 합니다.

최상의 결과를 얻으려면 RWX(**ReadWriteMany**) 액세스 모드와 **Block** 볼륨 모드를 사용합니다. 이는 다음과 같은 이유로 중요합니다.

- 실시간 마이그레이션에는 RWX(**ReadWriteMany**) 액세스 모드가 필요합니다.
- **블록** 볼륨 모드는 **Filesystem** 볼륨 모드보다 훨씬 더 잘 작동합니다. 이는 **Filesystem** 볼륨 모드가 파일 시스템 계층 및 디스크 이미지 파일을 포함하여 더 많은 스토리지 계층을 사용하기 때문입니다. 이러한 계층은 VM 디스크 스토리지에 필요하지 않습니다.
  예를 들어 Red Hat OpenShift Data Foundation을 사용하는 경우 CephFS 볼륨에 Ceph RBD 볼륨을 사용하는 것이 좋습니다.

중요

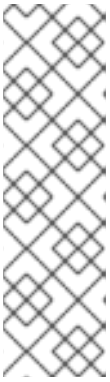
다음 구성으로 가상 머신을 실시간 마이그레이션할 수 없습니다.

- RWO(**ReadWriteOnce**) 액세스 모드가 있는 스토리지 볼륨
- GPU와 같은 패스스루 기능

이러한 가상 머신에 대해 **evictionStrategy** 필드를 **None** 으로 설정합니다. **None** 전략은 노드를 재부팅하는 동안 VM의 전원을 끕니다.

4.1.3. 실시간 마이그레이션 요구 사항

- RWX(**ReadWriteMany**) 액세스 모드를 사용한 공유 스토리지.
- 충분한 RAM 및 네트워크 대역폭.



참고

노드 드레이닝을 지원하기 위해 클러스터에 메모리 요청 용량이 충분한지 확인하여 실시간 마이그레이션을 수행해야 합니다. 다음 계산을 사용하여 필요한 예비 메모리를 확인할 수 있습니다.

> Product of (Maximum number of nodes that can drain in parallel) and (Highest total VM memory request allocations across nodes)

클러스터에서 병렬로 실행할 수 있는 기본 마이그레이션 수는 5입니다.

- 가상 머신에서 호스트 모델 CPU를 사용하는 경우 노드는 가상 시스템의 호스트 모델 CPU를 지원해야 합니다.

- 실시간 마이그레이션을 위한 전용 Multus 네트워크를 사용하는 것이 좋습니다. 전용 네트워크는 마이그레이션 중에 테넌트 워크로드에 대한 네트워크 포화 상태로 인한 영향을 최소화합니다.

4.1.4. 물리적 리소스 오버헤드 요구사항

OpenShift Virtualization은 OpenShift Container Platform의 추가 기능이며 클러스터를 계획할 때 고려해야 하는 추가 오버헤드를 적용합니다. 각 클러스터 머신은 OpenShift Container Platform 요구 사항 이외에도 다음과 같은 오버헤드 요구 사항을 충족해야 합니다. 클러스터에서 물리적 리소스를 초과 구독하면 성능에 영향을 미칠 수 있습니다.

중요

이 문서에 명시된 수치는 Red Hat의 테스트 방법론 및 설정을 기반으로 한 것입니다. 고유한 개별 설정 및 환경에 따라 수치가 달라질 수 있습니다.

메모리 오버헤드
아래 식을 사용하여 OpenShift Virtualization의 메모리 오버헤드 값을 계산합니다.

클러스터 메모리 오버헤드

```
Memory overhead per infrastructure node ≈ 150 MiB
```

```
Memory overhead per worker node ≈ 360 MiB
```

또한, OpenShift Virtualization 환경 리소스에는 모든 인프라 노드에 분산된 총 2179MiB의 RAM이 필요합니다.

가상 머신 메모리 오버헤드

```
Memory overhead per virtual machine ≈ (1.002 × requested memory) \
            + 218 MiB \ 1
            + 8 MiB × (number of vCPUs) \ 2
            + 16 MiB × (number of graphics devices) \ 3
            + (additional memory overhead) 4
```

1. **virt-launcher** Pod에서 실행되는 프로세스에 필요합니다.
2. 가상 머신에서 요청한 가상 CPU 수입니다.
3. 가상 머신에서 요청한 가상 그래픽 카드 수입니다.
4. 추가 메모리 오버헤드:
    - 환경에 SR-IOV(Single Root I/O Virtualization) 네트워크 장치 또는 GPU(Graphics Processing Unit)가 포함된 경우 각 장치에 대해 1GiB의 추가 메모리 오버헤드를 할당합니다.
    - SEV(Secure Encrypted Virtualization)가 활성화된 경우 256MiB를 추가합니다.
    - 신뢰할 수 있는 플랫폼 모듈(TPM)이 활성화된 경우 53MiB를 추가합니다.

CPU 오버헤드

아래 식을 사용하여 OpenShift Virtualization에 대한 클러스터 프로세서 오버헤드 요구 사항을 계산합니다. 가상 머신당 CPU 오버헤드는 개별 설정에 따라 다릅니다.

클러스터 CPU 오버헤드

```
CPU overhead for infrastructure nodes ≈ 4 cores
```

OpenShift Virtualization은 로깅, 라우팅 및 모니터링과 같은 클러스터 수준 서비스의 전반적인 사용률을 높입니다. 이 워크로드를 처리하려면 인프라 구성 요소를 호스팅하는 노드에 4 개의 추가 코어 (4000밀리코어)가 해당 노드에 분산되어 있는지 확인합니다.

```
CPU overhead for worker nodes ≈ 2 cores + CPU overhead per virtual machine
```

가상 머신을 호스팅하는 각 작업자 노드는 가상 머신 워크로드에 필요한 CPU 외에도 OpenShift Virtualization 관리 워크로드에 대한 2개의 추가 코어(2000밀리코어)용 용량이 있어야 합니다.

가상 머신 CPU 오버헤드

전용 CPU가 요청되면 클러스터 CPU 오버헤드 요구 사항에 대한 1:1 영향이 있습니다. 그러지 않으면 가상 머신에 필요한 CPU 수에 대한 구체적인 규칙이 없습니다.

스토리지 오버헤드
아래 지침을 사용하여 OpenShift Virtualization 환경에 대한 스토리지 오버헤드 요구 사항을 추정할 수 있습니다.

클러스터 스토리지 오버헤드

```
Aggregated storage overhead per node ≈ 10 GiB
```

10GiB는 OpenShift Virtualization을 설치할 때 클러스터의 각 노드에 대해 예상되는 온디스크 스토리지 영향입니다.

가상 머신 스토리지 오버헤드

가상 머신당 스토리지 오버헤드는 가상 머신 내의 리소스 할당 요청에 따라 다릅니다. 이 요청은 클러스터의 다른 위치에서 호스팅되는 노드 또는 스토리지 리소스의 임시 스토리지에 대한 요청일 수 있습니다. 현재 OpenShift Virtualization은 실행 중인 컨테이너 자체에 대한 추가 임시 스토리지를 할당하지 않습니다.

예

클러스터 관리자가 클러스터에서 10개의 가상 머신을 호스팅하는 경우 1GiB RAM과 2개의 vCPU가 장착된 메모리의 클러스터 전체에 대한 영향은 11.68GiB입니다. 클러스터의 각 노드에 대한 디스크 스토리지 영향은 10GiB이며 호스트 가상 머신 워크로드가 최소 2개 코어인 작업자 노드에 대한 CPU 영향은 최소 2개입니다.

4.1.5. 단일 노드 OpenShift 차이점

단일 노드 OpenShift에 OpenShift Virtualization을 설치할 수 있습니다.

그러나 Single-node OpenShift는 다음 기능을 지원하지 않습니다.

- 고가용성
- Pod 중단
- 실시간 마이그레이션

- 제거 전략이 구성된 가상 머신 또는 템플릿

추가 리소스

- OpenShift Container Platform 스토리지의 일반 용어집

4.1.6. 오브젝트 최대값

클러스터를 계획할 때 다음과 같은 테스트된 오브젝트 최대값을 고려해야 합니다.

- OpenShift Container Platform 오브젝트 최대값.
- OpenShift Virtualization 오브젝트 최대값.

4.1.7. 클러스터 고가용성 옵션

클러스터에 대해 다음과 같은 HA(고가용성) 옵션 중 하나를 구성할 수 있습니다.

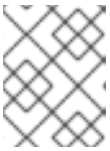
- 머신 상태 점검을 배포하여 설치 관리자 프로비저닝 인프라 (IPI)의 자동 고가용성을 사용할 수 있습니다.

참고

설치 관리자 프로비저닝 인프라를 사용하여 설치된 OpenShift Container Platform 클러스터에서는 올바르게 구성된 **MachineHealthCheck** 리소스를 사용하여 노드가 머신 상태 점검에 실패하고 클러스터에서 사용할 수 없게 되는 경우 재활용됩니다. 실패한 노드에서 실행된 VM에서 다음에 수행되는 작업은 일련의 조건에 따라 다릅니다. 잠재적 결과 및 실행 전략이 이러한 결과에 미치는 영향에 대한 자세한 내용은 전략 실행을 참조하십시오.

- OpenShift Container Platform 클러스터에서 **Node Health Check Operator**를 사용하여 **NodeHealthCheck** 컨트롤러를 배포하면 IPI 및 비 IPI에 대한 자동 고가용성을 사용할 수 있습니다. 컨트롤러는 비정상 노드를 식별하고 Self Node Remediation Operator 또는 Fence Agents Remediation Operator와 같은 수정 공급자를 사용하여 비정상 노드를 수정합니다. 노드 수정, 펜싱 및 유지 관리에 대한 자세한 내용은 Workload Availability for Red Hat OpenShift 설명서를 참조하십시오.

- 모든 플랫폼의 고가용성은 모니터링 시스템 또는 자격을 갖춘 사람을 사용하여 노드 가용성을 모니터링합니다. 노드가 손실되면 노드를 종료하고 **oc delete node <lost_node>**를 실행합니다.

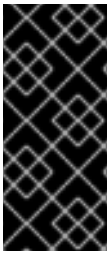


참고

외부 모니터링 시스템 또는 인증된 사용자 모니터링 노드 상태가 없으면 가상 머신의 가용성이 저하됩니다.

4.2. OPENSHIFT VIRTUALIZATION 설치

OpenShift Virtualization을 설치하여 OpenShift Container Platform 클러스터에 가상화 기능을 추가합니다.



중요

인터넷 연결이 없는 제한된 환경에 OpenShift Virtualization을 설치하는 경우 제한된 네트워크에 대해 OLM(Operator Lifecycle Manager)을 구성해야 합니다.

인터넷 연결이 제한된 경우 OperatorHub에 액세스하도록 OLM에서 프록시 지원을 구성할 수 있습니다.

4.2.1. OpenShift Virtualization Operator 설치

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 OpenShift Virtualization Operator를 설치합니다.

4.2.1.1. 웹 콘솔을 사용하여 OpenShift Virtualization Operator 설치

OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift Virtualization Operator를 배포할 수 있습니다.

사전 요구 사항

- 클러스터에 OpenShift Container Platform 4.16을 설치합니다.
- OpenShift Container Platform 웹 콘솔에 **cluster-admin** 권한이 있는 사용자로 로그인합니다.

프로세스

1. **관리자**로 **Operator → OperatorHub**를 클릭합니다.
2. **키워드로 필터링** 필드에 가상화를 **입력합니다**.
3. **Red Hat** 소스 레이블을 사용하여 **OpenShift Virtualization Operator** 타일을 선택합니다.
4. Operator에 대한 정보를 확인하고 **Install**을 클릭합니다.
5. **Operator 설치** 페이지에서 다음을 수행합니다.
   a. 사용 가능한 **업데이트 채널** 옵션 목록에서 **stable**을 선택합니다. 이렇게 하면 OpenShift Container Platform 버전과 호환되는 OpenShift Virtualization 버전을 설치할 수 있습니다.
   b. **설치된 네임스페이스**의 경우 **Operator 권장 네임스페이스** 옵션이 선택되어 있는지 확인합니다. 그러면 필수 **openshift-cnv** 네임스페이스에 Operator가 설치되고, 해당 네임스페이스가 존재하지 않는 경우 자동으로 생성됩니다.

주의

openshift-cnv 이외의 네임스페이스에 OpenShift Virtualization Operator를 설치하려고 하면 설치가 실패합니다.

   c. **승인 전략**의 경우 기본값인 **자동**을 선택하여 OpenShift Virtualization이 **안정적**인 업데이트 채널에서 새 버전을 사용할 수 있을 때 자동으로 업데이트되도록 하는 것이 좋습니다.

수동 승인 전략을 선택할 수 있지만 클러스터의 지원 가능성 및 기능에 미칠 위험이 높기 때문에 이 방법은 권장할 수 없습니다. 이러한 위험을 완전히 이해하고 **자동**을 사용할 수 없는 경우에만 **수동**을 선택합니다.

주의

해당 OpenShift Container Platform 버전과 함께 사용할 때만 OpenShift Virtualization을 지원하므로 누락된 OpenShift Virtualization 업데이트가 없으면 클러스터가 지원되지 않을 수 있습니다.

6. **openshift-cnv** 네임스페이스에서 Operator를 사용할 수 있도록 **설치**를 클릭합니다.
7. Operator가 설치되면 **HyperConverged 생성**을 클릭합니다.
8. 선택 사항: OpenShift Virtualization 구성 요소에 대한 **Infra** 및 **워크로드** 노드 배치 옵션을 구성합니다.
9. **생성**을 클릭하여 OpenShift Virtualization을 시작합니다.

검증

- **워크로드 → Pods** 페이지로 이동하여 모두 **실행 중** 상태가 될 때까지 OpenShift Virtualization Pod를 모니터링합니다. 모든 Pod에 **실행 중** 상태가 표시되면 OpenShift Virtualization을 사용할 수 있습니다.

4.2.1.2. 명령줄을 사용하여 OpenShift Virtualization Operator 설치

OpenShift Virtualization 카탈로그를 구독하고 클러스터에 매니페스트를 적용하여 OpenShift Virtualization Operator를 설치합니다.

4.2.1.2.1. CLI를 사용하여 OpenShift Virtualization 카탈로그 구독

OpenShift Virtualization을 설치하기 전에 OpenShift Virtualization 카탈로그를 구독해야 합니다. 구독하면 **openshift-cnv** 네임스페이스에서 OpenShift Virtualization Operator에 액세스할 수 있습니다.

구독하려면 클러스터에 단일 매니페스트를 적용하여 **Namespace**, **OperatorGroup**, **Subscription** 오브젝트를 구성합니다.

사전 요구 사항

- 클러스터에 OpenShift Container Platform 4.16을 설치합니다.
- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

프로세스

1. 다음 매니페스트를 포함하는 YAML 파일을 만듭니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-cnv
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: kubevirt-hyperconverged-group
  namespace: openshift-cnv
spec:
  targetNamespaces:
    - openshift-cnv
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.16.0
  channel: "stable" (1)
```

(1) **stable** 채널을 사용하면 OpenShift Container Platform 버전과 호환되는 OpenShift Virtualization 버전을 설치할 수 있습니다.

2. 다음 명령을 실행하여 OpenShift Virtualization에 필요한 **Namespace**, **OperatorGroup** 및 **Subscription** 오브젝트를 생성합니다.

```
$ oc apply -f <file name>.yaml
```

참고

YAML 파일에서 인증서 교체 매개변수를 구성할 수 있습니다.

4.2.1.2.2. CLI를 사용하여 OpenShift Virtualization Operator 배포

oc CLI를 사용하여 OpenShift Virtualization Operator를 배포할 수 있습니다.

사전 요구 사항

- **openshift-cnv** 네임스페이스에서 OpenShift Virtualization 카탈로그를 구독합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

프로세스

1. 다음 매니페스트를 포함하는 YAML 파일을 만듭니다.

```
apiVersion: hco.kubevirt.io/v1beta1
```

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
```

2. 다음 명령을 실행하여 OpenShift Virtualization Operator를 배포합니다.

```
$ oc apply -f <file_name>.yaml
```

검증

- **openshift-cnv** 네임스페이스에서 CSV(클러스터 서비스 버전)의 **PHASE**를 확인하여 OpenShift Virtualization이 성공적으로 배포되었는지 확인합니다. 다음 명령을 실행합니다.

```
$ watch oc get csv -n openshift-cnv
```

배포에 성공하면 다음 출력이 표시됩니다.

출력 예

```
NAME                                      DISPLAY                    VERSION   REPLACES   PHASE
kubevirt-hyperconverged-operator.v4.16.0   OpenShift Virtualization   4.16.0
Succeeded
```

4.2.2. 다음 단계

- hostpath 프로비전 프로그램은 OpenShift Virtualization용으로 설계된 로컬 스토리지 프로비전 프로그램입니다. 가상 머신의 로컬 스토리지를 구성하려면 먼저 hostpath 프로비전 프로그램을 활성화해야 합니다.

4.3. OPENSHIFT VIRTUALIZATION 설치 제거

웹 콘솔 또는 CLI(명령줄 인터페이스)를 사용하여 OpenShift Virtualization 워크로드, Operator 및 해당 리소스를 삭제하여 OpenShift Virtualization을 설치 제거합니다.

4.3.1. 웹 콘솔을 사용하여 OpenShift Virtualization 설치 제거

웹 콘솔 을 사용하여 다음 작업을 수행하여 OpenShift Virtualization을 설치 제거합니다.

1. **HyperConverged** CR을 삭제합니다.
2. OpenShift Virtualization Operator를 삭제합니다.
3. **openshift-cnv** 네임스페이스를 삭제합니다.
4. OpenShift Virtualization CRD(사용자 정의 리소스 정의)를 삭제합니다.

중요

먼저 모든 가상 머신 및 가상 머신 인스턴스를 삭제해야 합니다.

워크로드가 클러스터에 남아 있는 동안 OpenShift Virtualization을 설치 제거할 수 없습니다.

4.3.1.1. HyperConverged 사용자 정의 리소스 삭제

OpenShift Virtualization을 설치 제거하려면 먼저 **HyperConverged** CR(사용자 정의 리소스)을 삭제합니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.

프로세스

1. **Operator → 설치된 Operator** 페이지로 이동합니다.
2. OpenShift Virtualization Operator를 선택합니다.
3. **OpenShift Virtualization 배포** 탭을 클릭합니다.
4. **kubevirt-hyperconverged** 옆에 있는 옵션 메뉴 를 클릭하고 **HyperConverged 삭제** 를 선택합니다.
5. 확인 창에서 **삭제**를 클릭합니다.

4.3.1.2. 웹 콘솔을 사용하여 클러스터에서 Operator 삭제

클러스터 관리자는 웹 콘솔을 사용하여 선택한 네임스페이스에서 설치된 Operator를 삭제할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터 웹 콘솔에 액세스할 수 있습니다.

프로세스

1. **Operator → 설치된 Operator** 페이지로 이동합니다.
2. 제거하려는 Operator를 찾으려면 **이름으로 필터링** 필드에 키워드를 스크롤하거나 입력합니다. 그런 다음 해당 Operator를 클릭합니다.
3. **Operator 세부 정보** 페이지 오른쪽에 있는 **작업** 목록에서 **Operator 제거**를 선택합니다. **Operator를 설치 제거하시겠습니까?** 대화 상자가 표시됩니다.
4. **설치 제거**를 선택하여 Operator, Operator 배포 및 Pod를 제거합니다. 이 작업 후에 Operator는 실행을 중지하고 더 이상 업데이트가 수신되지 않습니다.

참고

이 작업은 CRD(사용자 정의 리소스 정의) 및 CR(사용자 정의 리소스)을 포함하여 Operator에서 관리하는 리소스를 제거하지 않습니다. 웹 콘솔에서 활성화된 대시보드 및 탐색 항목과 계속 실행되는 클러스터 외부 리소스는 수동 정리가 필요할 수 있습니다. Operator를 설치 제거한 후 해당 항목을 제거하려면 Operator CRD를 수동으로 삭제해야 할 수 있습니다.

4.3.1.3. 웹 콘솔을 사용하여 네임스페이스 삭제

OpenShift Container Platform 웹 콘솔을 사용하여 네임스페이스를 삭제할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.

프로세스

1. **관리 → 네임스페이스**로 이동합니다.
2. 네임스페이스 목록에서 삭제하려는 네임스페이스를 찾습니다.
3. 네임스페이스 목록 맨 오른쪽에 있는 옵션 메뉴  에서 **네임스페이스 삭제**를 선택합니다.
4. **네임스페이스 삭제** 창이 열리면 삭제할 네임스페이스 이름을 필드에 입력합니다.
5. **삭제**를 클릭합니다.

4.3.1.4. OpenShift Virtualization 사용자 정의 리소스 정의 삭제

웹 콘솔을 사용하여 OpenShift Virtualization CRD(사용자 정의 리소스 정의)를 삭제할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.

프로세스

1. **관리 → 클러스터 리소스 정의**로 이동합니다.
2. **라벨** 필터를 선택하고 **검색** 필드에 **operators.coreos.com/kubevirt-hyperconverged.openshift-cnv**를 입력하여 OpenShift Virtualization CRD를 표시합니다.
3. 각 CRD 옆에 있는 옵션 메뉴  를 클릭하고 **CustomResourceDefinition 삭제**를 선택합니다.

4.3.2. CLI를 사용하여 OpenShift Virtualization 설치 제거

OpenShift CLI(**oc**)를 사용하여 OpenShift Virtualization을 설치 제거할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.
- 모든 가상 머신 및 가상 머신 인스턴스를 삭제했습니다. 워크로드가 클러스터에 남아 있는 동안 OpenShift Virtualization을 설치 제거할 수 없습니다.

프로세스

1. **HyperConverged** 사용자 정의 리소스를 삭제합니다.

```
$ oc delete HyperConverged kubevirt-hyperconverged -n openshift-cnv
```

2. OpenShift Virtualization Operator 서브스크립션을 삭제합니다.

```
$ oc delete subscription kubevirt-hyperconverged -n openshift-cnv
```

3. OpenShift Virtualization **ClusterServiceVersion** 리소스를 삭제합니다.

```
$ oc delete csv -n openshift-cnv -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv
```

4. OpenShift Virtualization 네임스페이스를 삭제합니다.

```
$ oc delete namespace openshift-cnv
```

5. **시험 실행 옵션으로 oc delete crd 명령을 실행하여 OpenShift Virtualization CRD(사용자 정의 리소스 정의)**를 나열합니다.

```
$ oc delete crd --dry-run=client -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv
```

출력 예

```
customresourcedefinition.apiextensions.k8s.io "cdis.cdi.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io
"hostpathprovisioners.hostpathprovisioner.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "hyperconvergeds.hco.kubevirt.io" deleted
(dry run)
customresourcedefinition.apiextensions.k8s.io "kubevirts.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io
"networkaddonsconfigs.networkaddonsoperator.network.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "ssps.ssp.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "tektontasks.tektontasks.kubevirt.io" deleted
(dry run)
```

6. **dry-run** 옵션 없이 **oc delete crd** 명령을 실행하여 CRD를 삭제합니다.

```
$ oc delete crd -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv
```

추가 리소스

- 가상 머신 삭제
- 가상 머신 인스턴스 삭제

5장. 설치 후 구성

5.1. 설치 후 구성

일반적으로 OpenShift Virtualization을 설치한 후 다음 절차를 수행합니다. 사용자 환경과 관련된 구성 요소를 구성할 수 있습니다.

- OpenShift Virtualization Operator, 워크로드 및 컨트롤러에 대한 노드 배치 규칙
- 네트워크 구성:
  - Kubernetes NMState 및 SR-IOV Operator 설치
  - VM(가상 머신)에 대한 외부 액세스를 위한 Linux 브리지 네트워크 구성
  - 실시간 마이그레이션을 위한 전용 보조 네트워크 구성
  - SR-IOV 네트워크 구성
  - OpenShift Container Platform 웹 콘솔을 사용하여 로드 밸런서 서비스 생성 활성화
- 스토리지 구성:
  - CSI(Container Storage Interface)의 기본 스토리지 클래스 정의
  - HPP(Hostpath Provisioner)를 사용하여 로컬 스토리지 구성

5.2. OPENSHIFT VIRTUALIZATION 구성 요소를 위한 노드 지정

베어 메탈 노드의 VM(가상 머신)의 기본 예약이 적합합니다. 선택적으로 노드 배치 규칙을 구성하여 OpenShift Virtualization Operator, 워크로드 및 컨트롤러를 배포할 노드를 지정할 수 있습니다.

참고

OpenShift Virtualization을 설치한 후 일부 구성 요소에 대한 노드 배치 규칙을 구성할 수 있지만 워크로드에 대한 노드 배치 규칙을 구성하려면 가상 머신이 존재할 수 없습니다.

5.2.1. OpenShift Virtualization 구성 요소의 노드 배치 규칙 정보

다음 작업에 노드 배치 규칙을 사용할 수 있습니다.

- 가상화 워크로드를 위한 노드에만 가상 머신을 배포합니다.
- 인프라 노드에만 Operator를 배포합니다.
- 워크로드 분리를 유지 관리합니다.

오브젝트에 따라, 다음 규칙 유형 중 하나 이상을 사용할 수 있습니다.

nodeSelector

이 필드에서 지정하는 키-값 쌍으로 라벨이 지정된 노드에 Pod를 예약할 수 있습니다. 노드에는 나열된 모든 쌍과 정확히 일치하는 라벨이 있어야 합니다.

유사성

더 많은 표현 구문을 사용하여 노드와 Pod의 일치 규칙을 설정할 수 있습니다. 유사성을 사용하면 규칙 적용 방법을 보다 자세하게 설정할 수 있습니다. 예를 들어 규칙은 요구 사항이 아닌 기본 설정임을 지정할 수 있습니다. 규칙이 기본 설정인 경우 규칙이 충족되지 않으면 Pod가 계속 예약됩니다.

허용 오차

일치하는 테인트가 있는 노드에 Pod를 예약할 수 있습니다. 테인트가 노드에 적용되는 경우, 해당 노드는 테인트를 허용하는 Pod만 허용합니다.

5.2.2. 노드 배치 규칙 적용

명령줄을 사용하여 **Subscription**,**HyperConverged** 또는 **HostPathProvisioner** 오브젝트를 편집하여 노드 배치 규칙을 적용할 수 있습니다.

사전 요구 사항

- **oc** CLI 툴이 설치되어 있습니다.
- 클러스터 관리자 권한으로 로그인되어 있습니다.

프로세스

1. 다음 명령을 실행하여 기본 편집기에서 오브젝트를 편집합니다.

```
$ oc edit <resource_type> <resource_name> -n {CNVNamespace}
```

2. 파일을 저장하여 변경 사항을 적용합니다.

5.2.3. 노드 배치 규칙 예

Subscription,**HyperConverged** 또는 **HostPathProvisioner** 오브젝트를 편집하여 OpenShift Virtualization 구성 요소에 대한 노드 배치 규칙을 지정할 수 있습니다.

5.2.3.1. 서브스크립션 오브젝트 노드 배치 규칙 예

OLM이 OpenShift Virtualization Operator를 배포하는 노드를 지정하려면, OpenShift Virtualization 설치 중에 **서브스크립션** 오브젝트를 편집합니다.

현재는 웹 콘솔을 사용하여 **서브스크립션** 오브젝트에 대한 노드 배치 규칙을 구성할 수 없습니다.

Subscription 오브젝트는 **유사성** 노드 pplacement 규칙을 지원하지 않습니다.

nodeSelector 규칙이 있는 Subscription 오브젝트의 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.16.0
  channel: "stable"
```

```
  config:
    nodeSelector:
      example.io/example-infra-key: example-infra-value 1
```

1 OLM은 **example.io/example-infra-key = example-infra-value** 레이블이 지정된 노드에 OpenShift Virtualization Operator를 배포합니다.

허용 오차 규칙이 있는 Subscription 오브젝트의 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source:  redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.16.0
  channel: "stable"
  config:
    tolerations:
    - key: "key"
      operator: "Equal"
      value: "virtualization" 1
      effect: "NoSchedule"
```

1 OLM은 **key = virtualization:NoSchedule** 테인트에 레이블이 지정된 노드에 OpenShift Virtualization Operator를 배포합니다. 허용 오차가 일치하는 Pod만 이러한 노드에 예약됩니다.

5.2.3.2. HyperConverged 오브젝트 노드 배치 규칙 예

OpenShift Virtualization이 구성 요소를 배포하는 노드를 지정하려면 OpenShift Virtualization을 설치하는 동안 생성한 HyperConverged CR(사용자 정의 리소스) 파일에서 **nodePlacement** 오브젝트를 편집할 수 있습니다.

nodeSelector 규칙이 있는 HyperConverged 오브젝트의 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  infra:
    nodePlacement:
      nodeSelector:
        example.io/example-infra-key: example-infra-value 1
  workloads:
    nodePlacement:
      nodeSelector:
        example.io/example-workloads-key: example-workloads-value 2
```

1. 인프라 리소스는 **example.io/example-infra-key = example-infra-value** 레이블이 지정된 노드에 배치됩니다.

2. 워크로드는 **example.io/example-workloads-key = example-workloads-value** 라벨이 지정된 노드에 배치됩니다.

유사성 규칙이 있는 HyperConverged 오브젝트의 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  infra:
    nodePlacement:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-infra-key
                operator: In
                values:
                - example-infra-value 1
  workloads:
    nodePlacement:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-workloads-key 2
                operator: In
                values:
                - example-workloads-value
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: example.io/num-cpus
                operator: Gt
                values:
                - 8 3
```

1. 인프라 리소스는 **example.io/example-infra-key = example-value** 레이블이 지정된 노드에 배치됩니다.

2. 워크로드는 **example.io/example-workloads-key = example-workloads-value** 라벨이 지정된 노드에 배치됩니다.

3. 워크로드에 9개 이상의 CPU를 사용하는 것이 좋지만, 사용할 수 없는 경우에도 Pod가 예약됩니다.

허용 오차 규칙이 있는 HyperConverged 오브젝트의 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  workloads:
    nodePlacement:
      tolerations: 1
      - key: "key"
        operator: "Equal"
        value: "virtualization"
        effect: "NoSchedule"
```

1 OpenShift Virtualization 구성 요소로 예약된 노드는 **key = virtualization:NoSchedule** 테인트로 레이블이 지정됩니다. 허용 오차가 일치하는 Pod만 예약된 노드에 예약됩니다.

5.2.3.3. HostPathProvisioner 오브젝트 노드 배치 규칙의 예

직접 또는 웹 콘솔을 사용하여 **HostPathProvisioner** 오브젝트를 편집할 수 있습니다.

주의

동일한 노드에 hostpath 프로비전 프로그램 및 OpenShift Virtualization 구성 요소를 예약해야 합니다. 예약하지 않으면 hostpath 프로비전 프로그램을 사용하는 가상화 Pod를 실행할 수 없습니다. 가상 머신을 실행할 수 없습니다.

HPP(Hostpath provisioner) 스토리지 클래스를 사용하여 VM(가상 머신)을 배포한 후 노드 선택기를 사용하여 동일한 노드에서 hostpath 프로비저너 Pod를 제거할 수 있습니다. 그러나 VM을 삭제하기 전에 먼저 특정 노드의 경우 해당 변경 사항을 되돌리고 Pod가 실행될 때까지 기다려야 합니다.

hostpath 프로비전 프로그램을 설치할 때 생성하는 **HostPathProvisioner** 오브젝트의 **spec.workload** 필드에 **nodeSelector**,**affinity** 또는 **tolerations** 를 지정하여 노드 배치 규칙을 구성할 수 있습니다.

nodeSelector 규칙이 있는 HostPathProvisioner 오브젝트의 예

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  pathConfig:
    path: "</path/to/backing/directory>"
    useNamingPrefix: false
```

```
workload:
  nodeSelector:
    example.io/example-workloads-key: example-workloads-value (1)
```

(1) 워크로드는 **example.io/example-workloads-key = example-workloads-value** 라벨이 지정된 노드에 배치됩니다.

5.2.4. 추가 리소스

- 가상 머신용 노드 지정
- 노드 선택기를 사용하여 특정 노드에 Pod 배치
- 노드 유사성 규칙을 사용하여 노드에 Pod 배치 제어
- 노드 테인트를 사용하여 Pod 배치 제어

5.3. 설치 후 네트워크 구성

기본적으로 OpenShift Virtualization은 단일 내부 pod 네트워크와 함께 설치됩니다.

OpenShift Virtualization을 설치한 후 네트워킹 Operator를 설치하고 추가 네트워크를 구성할 수 있습니다.

5.3.1. 네트워킹 Operator 설치

실시간 마이그레이션 또는 VM(가상 머신)에 대한 외부 액세스를 위해 Linux 브리지 네트워크를 구성하려면 Kubernetes NMState Operator 를 설치해야 합니다. 설치 지침은 웹 콘솔을 사용하여 Kubernetes NMState Operator 설치를 참조하십시오.

SR-IOV Operator 를 설치하여 SR-IOV 네트워크 장치 및 네트워크 연결을 관리할 수 있습니다. 설치 지침은 SR-IOV Network Operator 설치를 참조하십시오.

MetalLB Operator 를 추가하여 클러스터에서 MetalLB 인스턴스의 라이프사이클을 관리할 수 있습니다. 설치 지침은 웹 콘솔을 사용하여 OperatorHub에서 MetalLB Operator 설치를 참조하십시오.

5.3.2. Linux 브리지 네트워크 구성

Kubernetes NMState Operator를 설치한 후 실시간 마이그레이션 또는 VM(가상 머신)에 대한 외부 액세스를 위해 Linux 브리지 네트워크를 구성할 수 있습니다.

5.3.2.1. Linux 브리지 NNCP 생성

Linux 브리지 네트워크에 대한 **NodeNetworkConfigurationPolicy** (NNCP) 매니페스트를 생성할 수 있습니다.

사전 요구 사항

- Kubernetes NMState Operator가 설치되어 있습니다.

프로세스

- **NodeNetworkConfigurationPolicy** 매니페스트를 생성합니다. 이 예제에는 고유한 정보로 교체해야 하는 샘플 값이 포함되어 있습니다.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy (1)
spec:
  desiredState:
    interfaces:
      - name: br1 (2)
        description: Linux bridge with eth1 as a port (3)
        type: linux-bridge (4)
        state: up (5)
        ipv4:
          enabled: false (6)
        bridge:
          options:
            stp:
              enabled: false (7)
          port:
            - name: eth1 (8)
```

(1) 정책 이름입니다.

(2) 인터페이스 이름입니다.

(3) 선택 사항: 사람이 읽을 수 있는 인터페이스 설명입니다.

(4) 인터페이스 유형입니다. 이 예제에서는 브리지를 만듭니다.

(5) 생성 후 인터페이스에 요청되는 상태입니다.

(6) 이 예에서는 IPv4를 비활성화합니다.

(7) 이 예에서는 STP를 비활성화합니다.

(8) 브리지가 연결된 노드 NIC입니다.

5.3.2.2. 웹 콘솔을 사용하여 Linux 브리지 생성

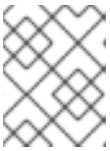
OpenShift Container Platform 웹 콘솔을 사용하여 네트워크 연결 정의(NAD)를 생성하여 Pod 및 가상 머신에 계층 2 네트워킹을 제공할 수 있습니다.

Linux 브리지 네트워크 연결 정의는 가상 머신을 VLAN에 연결하는 가장 효율적인 방법입니다.

> **주의**
>
> 가상 머신의 네트워크 연결 정의에서 IP 주소 관리(IPAM) 구성은 지원되지 않습니다.

프로세스

1. 웹 콘솔에서 **네트워킹 → NetworkAttachmentDefinitions** 를 클릭합니다.
2. **네트워크 연결 정의 생성** 을 클릭합니다.



참고

네트워크 연결 정의는 Pod 또는 가상 머신과 동일한 네임스페이스에 있어야 합니다.

3. 고유한 **이름**과 선택적 **설명**을 입력합니다.
4. **네트워크 유형** 목록에서 **CNV Linux 브리지** 를 선택합니다.
5. **브리지 이름** 필드에 브리지 이름을 입력합니다.
6. 선택 사항: 리소스에 VLAN ID가 구성된 경우 **VLAN 태그 번호** 필드에 ID 번호를 입력합니다.
7. 선택 사항: **MAC 스푸핑 검사를** 선택하여 MAC 스푸핑 필터링을 활성화합니다. 이 기능은 단일 MAC 주소만 Pod를 종료할 수 있도록 허용하여 MAC 스푸핑 공격에 대한 보안을 제공합니다.
8. **생성**을 클릭합니다.

다음 단계

- Linux 브리지 네트워크에 VM(가상 머신) 연결

5.3.3. 실시간 마이그레이션을 위한 네트워크 구성

Linux 브리지 네트워크를 구성한 후 실시간 마이그레이션을 위한 전용 네트워크를 구성할 수 있습니다. 전용 네트워크는 실시간 마이그레이션 중에 테넌트 워크로드에 대한 네트워크 포화 상태로 인한 영향을 최소화합니다.

5.3.3.1. 실시간 마이그레이션을 위한 전용 보조 네트워크 구성

실시간 마이그레이션을 위해 전용 보조 네트워크를 구성하려면 먼저 CLI를 사용하여 브리지 네트워크 연결 정의(NAD)를 생성해야 합니다. 그런 다음 **NetworkAttachmentDefinition** 오브젝트의 이름을 **HyperConverged** CR(사용자 정의 리소스)에 추가합니다.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 역할의 사용자로 클러스터에 로그인했습니다.
- 각 노드에는 NIC(네트워크 인터페이스 카드)가 두 개 이상 있습니다.
- 실시간 마이그레이션의 NIC는 동일한 VLAN에 연결됩니다.

프로세스

1. 다음 예에 따라 **NetworkAttachmentDefinition** 매니페스트를 생성합니다.

설정 파일 예

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network 1
  namespace: openshift-cnv 2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", 3
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", 4
      "range": "10.200.5.0/24" 5
    }
  }'
```

1. **NetworkAttachmentDefinition** 오브젝트의 이름을 지정합니다.
2. 3. 실시간 마이그레이션에 사용할 NIC의 이름을 지정합니다.
4. CryostatD에 대한 네트워크를 제공하는 CNI 플러그인의 이름을 지정합니다.
5. 보조 네트워크의 IP 주소 범위를 지정합니다. 이 범위는 기본 네트워크의 IP 주소를 겹치지 않아야 합니다.

2. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged** CR을 엽니다.

```
oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

3. **HyperConverged** CR의 **spec.liveMigrationConfig** 스탠자에 **NetworkAttachmentDefinition** 오브젝트의 이름을 추가합니다.

HyperConverged 매니페스트의 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    network: <network> 1
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
# ...
```

1. 실시간 마이그레이션에 사용할 Multus **NetworkAttachmentDefinition** 오브젝트의 이름을 지정합니다.

4. 변경 사항을 저장하고 편집기를 종료합니다. **virt-handler** Pod가 다시 시작되고 보조 네트워크에 연결합니다.

검증

- 가상 머신이 실행되는 노드가 유지보수 모드로 배치되면 VM이 클러스터의 다른 노드로 자동으로 마이그레이션됩니다. VMI(가상 머신 인스턴스) 메타데이터에서 대상 IP 주소를 확인하여 마이그레이션이 기본 pod 네트워크가 아닌 보조 네트워크를 통해 발생했는지 확인할 수 있습니다.

```
$ oc get vmi <vmi_name> -o jsonpath='{.status.migrationState.targetNodeAddress}'
```

5.3.3.2. 웹 콘솔을 사용하여 전용 네트워크 선택

OpenShift Container Platform 웹 콘솔을 사용하여 실시간 마이그레이션 전용 네트워크를 선택할 수 있습니다.

사전 요구 사항

- 실시간 마이그레이션을 위해 Multus 네트워크를 구성했습니다.

프로세스

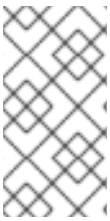
1. OpenShift Container Platform 웹 콘솔에서 **가상화 > 개요**로 이동합니다.
2. **설정** 탭을 클릭한 다음 **실시간 마이그레이션** 을 클릭합니다.
3. **실시간 마이그레이션 네트워크 목록에서 네트워크를** 선택합니다.

5.3.4. SR-IOV 네트워크 구성

SR-IOV Operator를 설치한 후 SR-IOV 네트워크를 구성할 수 있습니다.

5.3.4.1. SR-IOV 네트워크 장치 구성

SR-IOV Network Operator는 **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition을 OpenShift Container Platform에 추가합니다. SriovNetworkNodePolicy CR(사용자 정의 리소스)을 만들어 SR-IOV 네트워크 장치를 구성할 수 있습니다.



참고

SriovNetworkNodePolicy 오브젝트에 지정된 구성을 적용하면 SR-IOV Operator가 노드를 비우고 경우에 따라 노드를 재부팅할 수 있습니다.

구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- SR-IOV Network Operator가 설치되어 있습니다.
- 비운 노드에서 제거된 워크로드를 처리하기 위해 클러스터에 사용 가능한 노드가 충분합니다.

- SR-IOV 네트워크 장치 구성에 대한 컨트롤 플레인 노드를 선택하지 않았습니다.

절차

1. **SriovNetworkNodePolicy** 오브젝트를 생성한 후 YAML을 **<name>-sriov-node-network.yaml** 파일에 저장합니다. **<name>**을 이 구성의 이름으로 바꿉니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  numVfs: <num> 7
  nicSelector: 8
    vendor: "<vendor_code>" 9
    deviceID: "<device_id>" 10
    pfNames: ["<pf_name>", ...] 11
    rootDevices: ["<pci_bus_id>", "..."] 12
  deviceType: vfio-pci 13
  isRdma: false 14
```

1. CR 오브젝트의 이름을 지정합니다.
2. SR-IOV Operator가 설치된 네임스페이스를 지정합니다.
3. SR-IOV 장치 플러그인의 리소스 이름을 지정합니다. 리소스 이름에 대해 여러 **SriovNetworkNodePolicy** 오브젝트를 생성할 수 있습니다.
4. 구성할 노드를 선택하려면 노드 선택기를 지정합니다. 선택한 노드의 SR-IOV 네트워크 장치만 구성됩니다. SR-IOV CNI(Container Network Interface) 플러그인 및 장치 플러그인은 선택한 노드에만 배포됩니다.
5. 선택 사항: **0**에서 **99** 사이의 정수 값을 지정합니다. 숫자가 작을수록 우선 순위가 높아지므로 우선 순위 **10**은 우선 순위 **99**보다 높습니다. 기본값은 **99**입니다.
6. 선택 사항: 가상 기능의 최대 전송 단위(MTU) 값을 지정합니다. 최대 MTU 값은 NIC 모델마다 다를 수 있습니다.
7. SR-IOV 물리적 네트워크 장치에 생성할 가상 기능(VF) 수를 지정합니다. Intel NIC(Network Interface Controller)의 경우 VF 수는 장치에서 지원하는 총 VF보다 클 수 없습니다. Mellanox NIC의 경우 VF 수는 **128**보다 클 수 없습니다.
8. **nicSelector** 매핑은 Operator가 구성할 이더넷 장치를 선택합니다. 모든 매개변수에 값을 지정할 필요는 없습니다. 의도하지 않게 이더넷 장치를 선택할 가능성을 최소화하기 위해 이더넷 어댑터를 충분히 정밀하게 식별하는 것이 좋습니다. **rootDevices**를 지정하면 **vendor**, **deviceID** 또는 **pfNames**의 값도 지정해야 합니다. **pfNames**와 **rootDevices**를 동시에 지정하는 경우 동일한 장치를 가리키는지 확인하십시오.

9 선택 사항: SR-IOV 네트워크 장치의 공급업체 16진 코드를 지정합니다. 허용되는 유일한 값은 **8086** 또는 **15b3**입니다.

10 선택 사항: SR-IOV 네트워크 장치의 장치 16진수 코드를 지정합니다. 허용되는 값은 **158b**, **1015**, **1017**입니다.

11 선택 사항:이 매개변수는 이더넷 장치에 대한 하나 이상의 PF(물리적 기능) 이름 배열을 허용합니다.

12 이 매개변수는 이더넷 장치의 물리적 기능을 위해 하나 이상의 PCI 버스 주소 배열을 허용합니다. 주소를 **0000:02: 00.1** 형식으로 입력합니다.

13 **vfio-pci** 드라이버 유형은 OpenShift Virtualization의 가상 기능에 필요합니다.

14 선택 사항: 원격 직접 메모리 액세스(RDMA) 모드 사용 여부를 지정합니다. Mellanox 카드의 경우 **isRdma**를 **false**로 설정합니다. 기본값은 **false**입니다.

참고

isRDMA 플래그가 **true**로 설정된 경우 RDMA 가능 VF를 일반 네트워크 장치로 계속 사용할 수 있습니다. 어느 모드에서나 장치를 사용할 수 있습니다.

2. 선택 사항: **SriovNetworkNodePolicy.Spec.NodeSelector** 를 사용하여 SR-IOV 가능 클러스터 노드에 레이블을 지정하지 않은 경우 레이블을 지정합니다. 노드 레이블 지정에 대한 자세한 내용은 "노드에서 라벨을 업데이트하는 방법 이해"를 참조하십시오.

3. **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f <name>-sriov-node-network.yaml
```

<name>은 이 구성의 이름을 지정합니다.

구성 업데이트를 적용하면 **sriov-network-operator** 네임스페이스의 모든 Pod가 **Running** 상태로 전환됩니다.

4. SR-IOV 네트워크 장치가 구성되어 있는지 확인하려면 다음 명령을 입력합니다. **<node_name>**을 방금 구성한 SR-IOV 네트워크 장치가 있는 노드 이름으로 바꿉니다.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

다음 단계

- SR-IOV 네트워크에 VM(가상 머신) 연결

5.3.5. 웹 콘솔을 사용하여 로드 밸런서 서비스 생성 활성화

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)에 대한 로드 밸런서 서비스 생성을 활성화할 수 있습니다.

사전 요구 사항

- 클러스터에 대한 로드 밸런서를 구성했습니다.

- **cluster-admin** 역할의 사용자로 로그인되어 있습니다.

프로세스

1. **가상화 → 개요** 로 이동합니다.
2. **설정** 탭에서 클러스터를 **클릭합니다**.
3. **일반 설정** 및 **SSH 구성** 을 확장합니다.
4. **LoadBalancer 서비스를 통한 SSH**를 on으로 설정합니다.

5.4. 설치 후 스토리지 구성

다음 스토리지 구성 작업은 필수입니다.

- 클러스터의 기본 스토리지 클래스 를 구성해야 합니다. 그렇지 않으면 클러스터에서 자동 부팅 소스 업데이트를 수신할 수 없습니다.
- CDI에서 스토리지 공급자를 인식하지 못하는 경우 스토리지 프로필을 구성해야 합니다. 스토리지 프로필은 관련 스토리지 클래스를 기반으로 권장 스토리지 설정을 제공합니다.

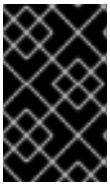
선택 사항: HPP(Hostpath provisioner)를 사용하여 로컬 스토리지를 구성할 수 있습니다.

CDI(Containerized Data Importer), 데이터 볼륨 및 자동 부팅 소스 업데이트를 구성하는 등 자세한 옵션은 스토리지 구성 개요 를 참조하십시오.

5.4.1. HPP를 사용하여 로컬 스토리지 구성

OpenShift Virtualization Operator를 설치하면 HPP(Hostpath Provisioner) Operator가 자동으로 설치됩니다. HPP Operator는 HPP 프로비저너를 생성합니다.

HPP는 OpenShift Virtualization을 위해 설계된 로컬 스토리지 프로비전 프로그램입니다. HPP를 사용하려면 HPP CR(사용자 정의 리소스)을 생성해야 합니다.



중요

HPP 스토리지 풀은 운영 체제와 동일한 파티션에 있으면 안 됩니다. 그렇지 않으면 스토리지 풀이 운영 체제 파티션을 채울 수 있습니다. 운영 체제 파티션이 가득 차면 성능이 영향을 미치거나 노드가 불안정하거나 사용할 수 없게 될 수 있습니다.

5.4.1.1. storagePools 스탠자를 사용하여 CSI 드라이버의 스토리지 클래스 생성

HPP(Hostpath provisioner)를 사용하려면 CSI(Container Storage Interface) 드라이버에 연결된 스토리지 클래스를 생성해야 합니다.

스토리지 클래스를 생성할 때 해당 스토리지 클래스에 속하는 PV(영구 볼륨)의 동적 프로비저닝에 영향을 주는 매개변수를 설정합니다. **StorageClass** 오브젝트를 생성한 후에는 이 오브젝트의 매개변수를 업데이트할 수 없습니다.

참고

가상 머신은 로컬 PV를 기반으로 하는 데이터 볼륨을 사용합니다. 로컬 PV는 특정 노드에 바인딩됩니다. 디스크 이미지는 가상 머신에서 사용할 수 있지만 이전에 로컬 스토리지 PV가 고정된 노드에 가상 머신을 예약할 수 없습니다.

이 문제를 해결하려면 Kubernetes Pod 스케줄러를 사용하여 PVC(영구 볼륨 클레임)를 올바른 노드의 PV에 바인딩합니다. **volumeBindingMode** 매개변수가 **WaitForFirstConsumer** 로 설정된 **StorageClass** 값을 사용하면 PVC를 사용하여 Pod가 생성될 때까지 PV의 바인딩 및 프로비저닝이 지연됩니다.

프로세스

1. **storageclass_csi.yaml** 파일을 생성하여 스토리지 클래스를 정의합니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hostpath-csi
provisioner: kubevirt.io.hostpath-provisioner
reclaimPolicy: Delete (1)
volumeBindingMode: WaitForFirstConsumer (2)
parameters:
  storagePool: my-storage-pool (3)
```

(1) **reclaimPolicy**에 사용할 수 있는 값은 **Delete** 및 **Retain** 두 가지입니다. 값을 지정하지 않으면 기본값은 **Delete** 입니다.

(2) **volumeBindingMode** 매개변수는 동적 프로비저닝 및 볼륨 바인딩이 발생하는 시기를 결정합니다. PVC(영구 볼륨 클레임)를 사용하는 Pod가 생성될 때까지 PV(영구 볼륨)의 바인딩 및 프로비저닝을 지연하려면 **WaitForFirstConsumer** 를 지정합니다. 이렇게 하면 PV에서 Pod의 스케줄링 요구 사항을 충족할 수 있습니다.

(3) HPP CR에 정의된 스토리지 풀의 이름을 지정합니다.

2. 파일을 저장하고 종료합니다.

3. 다음 명령을 실행하여 **StorageClass** 오브젝트를 만듭니다.

```
$ oc create -f storageclass_csi.yaml
```

6장. 업데이트

6.1. OPENSHIFT VIRTUALIZATION 업데이트

OLM(Operator Lifecycle Manager)에서 OpenShift Virtualization에 z-stream 및 마이너 버전 업데이트를 제공하는 방법을 알아봅니다.

6.1.1. RHEL 9의 OpenShift Virtualization

OpenShift Virtualization 4.16은 RHEL(Red Hat Enterprise Linux) 9를 기반으로 합니다. 표준 OpenShift Virtualization 업데이트 절차에 따라 RHEL 8을 기반으로 하는 버전에서 OpenShift Virtualization 4.16으로 업데이트할 수 있습니다. 추가 단계는 필요하지 않습니다.

이전 버전과 마찬가지로 실행 중인 워크로드를 중단하지 않고 업데이트를 수행할 수 있습니다. OpenShift Virtualization 4.16은 RHEL 8 노드에서 RHEL 9 노드로의 실시간 마이그레이션을 지원합니다.

6.1.1.1. RHEL 9 머신 유형

OpenShift Virtualization에 포함된 모든 VM 템플릿은 이제 RHEL 9 머신 유형을 기본적으로 사용합니다. **machineType: pc-q35-rhel9.<y>.0 .0.** 여기서 < **y** >는 RHEL 9의 최신 마이너 버전에 해당하는 단일 숫자입니다. 예를 들어, **pc-q35-rhel9.2.0** 값은 RHEL 9.2에 사용됩니다.

OpenShift Virtualization을 업데이트해도 기존 VM의 **machineType** 값은 변경되지 않습니다. 이러한 VM은 업데이트 전과 마찬가지로 계속 작동합니다. RHEL 9의 개선 사항을 활용할 수 있도록 VM의 머신 유형을 선택적으로 변경할 수 있습니다.

중요

VM의 **machineType** 값을 변경하기 전에 VM을 종료해야 합니다.

6.1.2. OpenShift Virtualization 업데이트 정보

- OLM(Operator Lifecycle Manager)은 OpenShift Virtualization Operator의 라이프사이클을 관리합니다. OpenShift Container Platform 설치 중에 배포되는 Marketplace Operator는 클러스터에서 외부 Operator를 사용할 수 있도록 합니다.

- OLM은 OpenShift Virtualization에 z-stream 및 마이너 버전 업데이트를 제공합니다. OpenShift Container Platform을 다음 마이너 버전으로 업데이트할 때 마이너 버전 업데이트를 사용할 수 있습니다. 먼저 OpenShift Container Platform을 업데이트하지 않고 OpenShift Virtualization을 다음 마이너 버전으로 업데이트할 수 없습니다.

- OpenShift Virtualization 서브스크립션은 **stable** 이라는 단일 업데이트 채널을 사용합니다. **stable** 채널을 사용하면 OpenShift Virtualization 및 OpenShift Container Platform 버전이 호환됩니다.

- 서브스크립션의 승인 전략이 자동으로 설정된 경우 **stable** 채널에서 새 버전의 Operator를 사용할 수 있는 즉시 업데이트 프로세스가 시작됩니다. **자동 승인 전략을 사용하여 지원 가능한 환경을 유지하는 것이 좋습니다. OpenShift Virtualization의 각 부 버전은 해당 OpenShift Container Platform 버전을 실행하는 경우에만 지원됩니다. 예를 들어 OpenShift Container Platform 4.16에서 OpenShift Virtualization 4.16을 실행해야 합니다.**

    - **수동 승인 전략을 선택할 수 있지만 클러스터의 지원 가능성과 기능에 미칠 위험이 높기 때문에 이 방법은 권장되지 않는 것이 좋습니다. 수동 승인 전략을 사용하면 보류 중인 모든 업데이트를 수동으로 승인해야 합니다. OpenShift Container Platform 및 OpenShift Virtualization 업**

데이트가 동기화되지 않으면 클러스터가 지원되지 않습니다.

- 업데이트를 완료하는 데 걸리는 시간은 네트워크 연결에 따라 달라집니다. 대부분의 자동 업데이트는 15분 이내에 완료됩니다.
- OpenShift Virtualization을 업데이트해도 네트워크 연결이 중단되지 않습니다.
- 데이터 볼륨 및 관련 영구 볼륨 클레임은 업데이트 중에 유지됩니다.

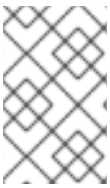
중요

hostpath 프로비전 프로그램 스토리지를 사용하는 가상 머신이 실행 중인 경우 실시간 마이그레이션할 수 없으며 OpenShift Container Platform 클러스터 업데이트를 차단할 수 있습니다.

이 문제를 해결하려면 클러스터 업데이트 중에 전원이 자동으로 꺼지도록 가상 머신을 재구성할 수 있습니다. **evictionStrategy** 필드를 **None** 으로 설정하고 **runStrategy** 필드를 **Always** 로 설정합니다.

6.1.2.1. 워크로드 업데이트 정보

OpenShift Virtualization을 업데이트하면 **libvirt**, **virt-launcher**, **qemu** 를 포함한 가상 머신 워크로드가 실시간 마이그레이션을 지원하는 경우 자동으로 업데이트됩니다.



참고

각 가상 머신에는 VMI(가상 머신 인스턴스)를 실행하는 **virt-launcher** Pod가 있습니다. **virt-launcher** Pod는 VM(가상 머신) 프로세스를 관리하는 데 사용되는 **libvirt** 인스턴스를 실행합니다.

HyperConverged CR (사용자 정의 리소스)의 **spec.workloadUpdateStrategy** 스탠자를 편집하여 워크로드가 업데이트되는 방법을 구성할 수 있습니다. 사용 가능한 워크로드 업데이트 방법은 **LiveMigrate** 및 **Evict** 입니다.

Evict 메서드는 VMI Pod를 종료하므로 **LiveMigrate** 업데이트 전략만 기본적으로 활성화됩니다.

LiveMigrate 가 유일하게 활성화된 업데이트 전략인 경우:

- 실시간 마이그레이션을 지원하는 VMI는 업데이트 프로세스 중에 마이그레이션됩니다. VM 게스트는 업데이트된 구성 요소가 활성화된 새 Pod로 이동합니다.
- 실시간 마이그레이션을 지원하지 않는 VMI는 중단되거나 업데이트되지 않습니다.
    - VMI에 **LiveMigrate** 제거 전략이 있지만 실시간 마이그레이션을 지원하지 않는 경우 업데이트되지 않습니다.

LiveMigrate 및 **Evict** 를 모두 활성화하는 경우:

- 실시간 마이그레이션을 지원하는 VMI는 **LiveMigrate** 업데이트 전략을 사용합니다.
- 실시간 마이그레이션을 지원하지 않는 VMI는 **Evict** 업데이트 전략을 사용합니다. **runStrategy: Always** 가 설정된 **VirtualMachine** 오브젝트에서 VMI를 제어하는 경우 업데이트된 구성 요소가 있는 새 VMI가 새 Pod에 생성됩니다.

마이그레이션 시도 및 타임아웃
워크로드를 업데이트할 때 Pod가 다음 기간에 **Pending** 상태인 경우 실시간 마이그레이션이 실패합니다.

5분

Pod가 예약 불가 때문에 보류 중인 경우 .

15분

어떤 이유로든 Pod가 보류 중 상태에 있는 경우입니다.

VMI가 마이그레이션에 실패하면 **virt-controller** 에서 다시 마이그레이션하려고 합니다. 새**virt-launcher** Pod에서 모든 편중 가능한 VMI가 실행될 때까지 이 프로세스를 반복합니다. VMI가 잘못 구성된 경우 이러한 시도는 무기한 반복될 수 있습니다.

참고

각 시도는 마이그레이션 오브젝트에 해당합니다. 가장 최근의 5개의 시도만 버퍼에 저장됩니다. 이렇게 하면 마이그레이션 오브젝트가 디버깅에 대한 정보를 유지하면서 시스템에서 누적되지 않습니다.

6.1.2.2. EUS에서 EUS로의 업데이트 정보

4.10 및 4.12를 포함한 모든 OpenShift Container Platform 마이너 버전은 EUS (Extended Update Support) 버전입니다. 그러나 Kubernetes 설계에는 마이너 버전 업데이트가 필요하므로 하나의 EUS 버전에서 다음 EUS 버전으로 직접 업데이트할 수 없습니다.

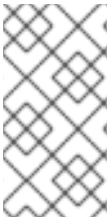
소스 EUS 버전에서 다음 홀수의 마이너 버전으로 업데이트한 후 OpenShift Virtualization을 업데이트 경로에 있는 해당 마이너 버전의 모든 z-stream 릴리스로 순차적으로 업데이트해야 합니다. 최신 적용 가능한 z-stream 버전으로 업그레이드하면 OpenShift Container Platform을 대상 EUS 마이너 버전으로 업데이트할 수 있습니다.

OpenShift Container Platform 업데이트가 성공하면 OpenShift Virtualization에 대한 해당 업데이트를 사용할 수 있습니다. OpenShift Virtualization을 대상 EUS 버전으로 업데이트할 수 있습니다.

6.1.2.2.1. 업데이트 준비

EUS에서 EUS로의 업데이트를 시작하기 전에 다음을 수행해야 합니다.

- EUS-to-EUS 업데이트를 시작하기 전에 작업자 노드의 머신 구성 풀을 일시 중지하여 작업자가 두 번 재부팅되지 않습니다.
- 업데이트 프로세스를 시작하기 전에 자동 워크로드 업데이트를 비활성화합니다. 이는 대상 EUS 버전으로 업데이트할 때까지 OpenShift Virtualization이 VM(가상 머신)을 마이그레이션하거나 제거하지 않도록 하기 위한 것입니다.



참고

기본적으로 OpenShift Virtualization은 OpenShift Virtualization Operator를 업데이트할 때 **virt-launcher** Pod와 같은 워크로드를 자동으로 업데이트합니다.**HyperConverged** 사용자 정의 리소스의 **spec.workloadUpdateStrategy** 스탠자에서 이 동작을 구성할 수 있습니다.

EUS에서 EUS로의 업데이트 수행에 대해 자세히 알아보십시오.

6.1.3. EUS에서 EUS로 업데이트 중 워크로드 업데이트 방지

하나의 EUS (Extended Update Support) 버전에서 다음으로 업데이트되는 경우 OpenShift Virtualization이 업데이트 프로세스 중에 워크로드를 마이그레이션하거나 제거하지 못하도록 자동 워크로드 업데이트를 수동으로 비활성화해야 합니다.

사전 요구 사항

- OpenShift Container Platform의 EUS 버전을 실행 중이며 다음 EUS 버전으로 업데이트하려고 합니다. 둘 사이의 홀수 버전으로 아직 업데이트되지 않았습니다.
- "EUS에서 EUS로의 업데이트를 수행하기 위한 준비"를 읽고 OpenShift Container Platform 클러스터와 관련된 경고 및 요구 사항을 알아봅니다.
- OpenShift Container Platform 설명서에서 지시한 대로 작업자 노드의 머신 구성 풀을 일시 중지했습니다.
- 기본 자동 승인 전략을 사용하는 것이 좋습니다.수동 승인 전략을 사용하는 경우 웹 콘솔에서 보류 중인 모든 업데이트를 승인해야 합니다. 자세한 내용은 "수동 보류 중인 Operator 업데이트" 섹션을 참조하십시오.

프로세스

1. 다음 명령을 실행하여 현재 **workloadUpdateMethods** 구성을 백업합니다.

```
$ WORKLOAD_UPDATE_METHODS=$(oc get kv kubevirt-kubevirt-hyperconverged \
  -n openshift-cnv -o
jsonpath='{.spec.workloadUpdateStrategy.workloadUpdateMethods}')
```

2. 다음 명령을 실행하여 모든 워크로드 업데이트 방법을 끕니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p
'[{"op":"replace","path":"/spec/workloadUpdateStrategy/workloadUpdateMethods",
"value":[]}]'
```

출력 예

```
hyperconverged.hco.kubevirt.io/kubevirt-hyperconverged patched
```

3. 계속하기 전에 **HyperConverged** Operator를 업그레이드할 수 있는지 확인합니다. 다음 명령을 입력하고 출력을 모니터링합니다.

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o json | jq
".status.conditions"
```

예 6.1. 출력 예

```
[
  {
    "lastTransitionTime": "2022-12-09T16:29:11Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
    "reason": "ReconcileCompleted",
    "status": "True",
    "type": "ReconcileComplete"
```

```
    },
    {
      "lastTransitionTime": "2022-12-09T20:30:10Z",
      "message": "Reconcile completed successfully",
      "observedGeneration": 3,
      "reason": "ReconcileCompleted",
      "status": "True",
      "type": "Available"
    },
    {
      "lastTransitionTime": "2022-12-09T20:30:10Z",
      "message": "Reconcile completed successfully",
      "observedGeneration": 3,
      "reason": "ReconcileCompleted",
      "status": "False",
      "type": "Progressing"
    },
    {
      "lastTransitionTime": "2022-12-09T16:39:11Z",
      "message": "Reconcile completed successfully",
      "observedGeneration": 3,
      "reason": "ReconcileCompleted",
      "status": "False",
      "type": "Degraded"
    },
    {
      "lastTransitionTime": "2022-12-09T20:30:10Z",
      "message": "Reconcile completed successfully",
      "observedGeneration": 3,
      "reason": "ReconcileCompleted",
      "status": "True",
      "type": "Upgradeable" (1)
    }
  ]
```


(1) OpenShift Virtualization Operator에는 **Upgradeable** 상태가 있습니다.

4. 소스 EUS 버전에서 OpenShift Container Platform의 다음 마이너 버전으로 클러스터를 수동으로 업데이트합니다.

```
$ oc adm upgrade
```

검증

- 다음 명령을 실행하여 현재 버전을 확인합니다.

```
$ oc get clusterversion
```

참고

OpenShift Container Platform을 다음 버전으로 업데이트하는 것은 OpenShift Virtualization을 업데이트하기 위한 사전 요구 사항입니다. 자세한 내용은 OpenShift Container Platform 설명서의 "클러스터 업그레이드" 섹션을 참조하십시오.

5. OpenShift Virtualization을 업데이트합니다.
   - 기본 자동 승인 전략을 사용하면 OpenShift Container Platform을 업데이트한 후 OpenShift Virtualization이 해당 버전으로 자동으로 업데이트됩니다.
   - 수동 승인 전략을 사용하는 경우 웹 콘솔을 사용하여 보류 중인 업데이트를 승인합니다.
6. 다음 명령을 실행하여 OpenShift Virtualization 업데이트를 모니터링합니다.

```
$ oc get csv -n openshift-cnv
```

7. OpenShift Virtualization을 EUS 마이너 버전이 아닌 모든 z-stream 버전으로 업데이트하고 이전 단계에 표시된 명령을 실행하여 각 업데이트를 모니터링합니다.
8. 다음 명령을 실행하여 OpenShift Virtualization이 EUS가 아닌 버전의 최신 z-stream 릴리스로 성공적으로 업데이트되었는지 확인합니다.

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o json | jq
".status.versions"
```

출력 예

```
[
  {
    "name": "operator",
    "version": "4.16.0"
  }
]
```

9. 다음 업데이트를 수행하기 전에 **HyperConverged** Operator가 **Upgradeable** 상태가 될 때까지 기다립니다. 다음 명령을 입력하고 출력을 모니터링합니다.

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o json | jq
".status.conditions"
```

10. OpenShift Container Platform을 대상 EUS 버전으로 업데이트합니다.
11. 클러스터 버전을 확인하여 업데이트가 성공했는지 확인합니다.

```
$ oc get clusterversion
```

12. OpenShift Virtualization을 대상 EUS 버전으로 업데이트합니다.
    - 기본 자동 승인 전략을 사용하면 OpenShift Container Platform을 업데이트한 후 OpenShift Virtualization이 해당 버전으로 자동으로 업데이트됩니다.
    - 수동 승인 전략을 사용하는 경우 웹 콘솔을 사용하여 보류 중인 업데이트를 승인합니다.

13. 다음 명령을 실행하여 OpenShift Virtualization 업데이트를 모니터링합니다.

```
$ oc get csv -n openshift-cnv
```

VERSION 필드가 대상 EUS 버전과 일치하고**PHASE** 필드가 **Succeeded** 인 경우 업데이트가 완료됩니다.

14. 백업한 워크로드 업데이트 방법 구성을 복원합니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv --type json -p \
  "[{\"op\":\"add\",\"path\":\"/spec/workloadUpdateStrategy/workloadUpdateMethods\",
\"value\":$WORKLOAD_UPDATE_METHODS}]"
```

출력 예

```
hyperconverged.hco.kubevirt.io/kubevirt-hyperconverged patched
```

검증

- 다음 명령을 실행하여 VM 마이그레이션의 상태를 확인합니다.

```
$ oc get vmim -A
```

다음 단계

- 이제 작업자 노드의 머신 구성 풀 일시 중지를 해제할 수 있습니다.

6.1.4. 워크로드 업데이트 방법 구성

HyperConverged CR(사용자 정의 리소스)을 편집하여 워크로드 업데이트 방법을 구성할 수 있습니다.

사전 요구 사항

- 실시간 마이그레이션을 업데이트 방법으로 사용하려면 먼저 클러스터에서 실시간 마이그레이션을 활성화해야 합니다.

참고

VirtualMachineInstance CR에 **evictionStrategy: LiveMigrate** 가 포함되어 있고 VMI(가상 머신 인스턴스)가 실시간 마이그레이션을 지원하지 않는 경우 VMI가 업데이트되지 않습니다.

프로세스

1. 기본 편집기에서 **HyperConverged** CR을 열려면 다음 명령을 실행합니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **HyperConverged** CR의 **workloadUpdateStrategy** 스탠자를 편집합니다. 예를 들면 다음과 같습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
```

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  workloadUpdateStrategy:
    workloadUpdateMethods: 1
    - LiveMigrate 2
    - Evict 3
    batchEvictionSize: 10 4
    batchEvictionInterval: "1m0s" 5
# ...
```

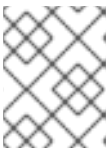
1. 자동화된 워크로드 업데이트를 수행하는 데 사용할 수 있는 방법입니다. 사용 가능한 값은 **LiveMigrate** 및 **Evict** 입니다. 이 예에 표시된 대로 두 옵션을 모두 활성화하면 업데이트에서 실시간 마이그레이션을 지원하지 않는 VMI에 실시간 마이그레이션 및 **Evict**를 지원하는 VMI에 **LiveMigrate**를 사용합니다. 자동 워크로드 업데이트를 비활성화하려면 **workloadUpdateStrategy** 스탠자를 제거하거나 **workloadUpdateMethods: []** 를 설정하여 배열을 비워 둘 수 있습니다.

2. 중단이 적은 업데이트 방법입니다. VMI(가상 머신) 게스트를 업데이트된 구성 요소가 활성화된 새 Pod로 마이그레이션하여 실시간 마이그레이션을 지원하는 VMI가 업데이트됩니다. **LiveMigrate**가 나열된 유일한 워크로드 업데이트 방법인 경우 실시간 마이그레이션을 지원하지 않는 VMI는 중단되거나 업데이트되지 않습니다.

3. 업그레이드 중 VMI Pod를 종료하는 중단 방법입니다. **Evict**는 클러스터에서 실시간 마이그레이션이 활성화되지 않은 경우 사용 가능한 유일한 업데이트 방법입니다. **runStrategy: Always** 가 구성된 **VirtualMachine** 오브젝트에서 VMI를 제어하는 경우 업데이트된 구성 요소가 있는 새 VMI가 새 Pod에 생성됩니다.

4. **Evict** 방법을 사용하여 한 번에 강제로 업데이트할 수 있는 VMI 수입니다. 이는 **LiveMigrate** 방법에는 적용되지 않습니다.

5. 다음 워크로드 배치를 제거하기 전에 대기하는 간격입니다. 이는 **LiveMigrate** 방법에는 적용되지 않습니다.



참고

HyperConverged CR의 **spec.liveMigrationConfig** 스탠자를 편집하여 실시간 마이그레이션 제한 및 타임아웃을 구성할 수 있습니다.

3. 변경 사항을 적용하려면 편집기를 저장하고 종료합니다.

6.1.5. 보류 중인 Operator 업데이트 승인

6.1.5.1. 보류 중인 Operator 업데이트 수동 승인

설치된 Operator의 서브스크립션에 있는 승인 전략이 수동으로 설정된 경우 새 업데이트가 현재 업데이트 채널에 릴리스될 때 업데이트를 수동으로 승인해야 설치가 시작됩니다.

사전 요구 사항

- OLM(Operator Lifecycle Manager)을 사용하여 이전에 설치한 Operator입니다.

절차

1. OpenShift Container Platform 웹 콘솔의 관리자 관점에서 Operator → 설치된 Operator로 이동합니다.
2. 보류 중인 업데이트가 있는 Operator에 업그레이드 사용 가능 상태가 표시됩니다. 업데이트할 Operator 이름을 클릭합니다.
3. 서브스크립션 탭을 클릭합니다. 승인이 필요한 업데이트는업그레이드 상태 옆에 표시됩니다. 예를 들어 1 승인 필요가 표시될 수 있습니다.
4. 1 승인 필요를 클릭한 다음 설치 계획 프리뷰를 클릭합니다.
5. 업데이트에 사용 가능한 것으로 나열된 리소스를 검토합니다. 문제가 없는 경우 승인을 클릭합니다.
6. Operator → 설치된 Operator 페이지로 이동하여업데이트 진행 상황을 모니터링합니다. 완료되면 상태가 성공 및 최신으로 변경됩니다.

6.1.6. 업데이트 상태 모니터링

6.1.6.1. OpenShift Virtualization 업그레이드 상태 모니터링

OpenShift Virtualization Operator 업그레이드 상태를 모니터링하려면 CSV(클러스터 서비스 버전) **PHASE**를 확인합니다. 웹 콘솔에서 또는 여기에 제공된 명령을 실행하여 CSV 조건을 모니터링할 수도 있습니다.

참고

PHASE 및 조건 값은 사용 가능한 정보를 기반으로 한 근사치입니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 로그인합니다.
- OpenShift CLI(**oc**)를 설치합니다.

절차

1. 다음 명령을 실행합니다.

```
$ oc get csv -n openshift-cnv
```

2. **PHASE** 필드를 확인하여 출력을 검토합니다. 예를 들면 다음과 같습니다.

출력 예

```
VERSION  REPLACES                                   PHASE
4.9.0    kubevirt-hyperconverged-operator.v4.8.2    Installing
4.9.0    kubevirt-hyperconverged-operator.v4.9.0    Replacing
```

3. 선택 사항: 다음 명령을 실행하여 모든 OpenShift Virtualization 구성 요소 조건을 집계한 상태를 모니터링합니다.

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  -o=jsonpath='{range .status.conditions[*]}{.type}{"\t"}{.status}{"\t"}{.message}{"\n"}
{end}'
```

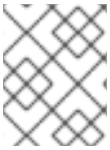
업그레이드가 완료되면 다음과 같은 결과가 나타납니다.

출력 예

```
ReconcileComplete  True  Reconcile completed successfully
Available          True  Reconcile completed successfully
Progressing        False Reconcile completed successfully
Degraded           False Reconcile completed successfully
Upgradeable        True  Reconcile completed successfully
```

6.1.6.2. 오래된 OpenShift Virtualization 워크로드 보기

CLI를 사용하여 오래된 워크로드 목록을 볼 수 있습니다.



참고

클러스터에 오래된 가상화 Pod가 있는 경우 **OutdatedVirtualMachineInstanceWorkloads** 경고가 실행됩니다.

절차

- 오래된 VMI(가상 머신 인스턴스) 목록을 보려면 다음 명령을 실행합니다.

```
$ oc get vmi -l kubevirt.io/outdatedLauncherImage --all-namespaces
```

참고

VMI가 자동으로 업데이트되도록 워크로드 업데이트를 구성합니다.

6.1.7. 추가 리소스

- EUS에서 EUS로 업데이트 수행
- Operator란 무엇인가?
- Operator Lifecycle Manager 개념 및 리소스
- CSV(클러스터 서비스 버전)
- 실시간 마이그레이션 정보
- 제거 전략 구성
- 실시간 마이그레이션 제한 및 타임아웃 구성

7장. 가상 머신

7.1. RED HAT 이미지에서 VM 생성

7.1.1. Red Hat 이미지에서 가상 머신 생성 개요

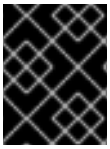
Red Hat 이미지는 골든 이미지입니다. 보안 레지스트리에 컨테이너 디스크로 게시됩니다. CDI(Containerized Data Importer)는 컨테이너 디스크를 폴링하고 클러스터로 가져와 **openshift-virtualization-os-images** 프로젝트에 스냅샷 또는 PVC(영구 볼륨 클레임)로 저장합니다.

Red Hat 이미지가 자동으로 업데이트됩니다. 이러한 이미지에 대한 자동 업데이트를 비활성화하고 다시 활성화할 수 있습니다. Red Hat 부팅 소스 업데이트 관리를참조하십시오.

클러스터 관리자는 OpenShift Virtualization 웹 콘솔 의 RHEL(Red Hat Enterprise Linux) 가상 머신에 대한 자동 서브스크립션을 활성화할 수 있습니다.

다음 방법 중 하나를 사용하여 Red Hat에서 제공하는 운영 체제 이미지에서 VM(가상 머신)을 생성할 수 있습니다.

- 웹 콘솔을 사용하여 템플릿에서 VM 생성
- 웹 콘솔을 사용하여 인스턴스 유형에서 VM 생성
- 명령줄을 사용하여 VirtualMachine 매니페스트에서 VM 생성



중요

기본 **openshift-*** 네임스페이스에 VM을 생성하지 마십시오. 대신 새 네임스페이스를 만들거나 **openshift** 접두사 없이 기존 네임스페이스를 사용하십시오.

7.1.1.1. 골든 이미지 정보

골든 이미지는 새 VM을 배포하는 리소스로 사용할 수 있는 VM(가상 머신)의 사전 구성된 스냅샷입니다. 예를 들어 golden 이미지를 사용하여 동일한 시스템 환경을 일관되게 프로비저닝하고 시스템을 보다 신속하고 효율적으로 배포할 수 있습니다.

7.1.1.1.1. 골든 이미지는 어떻게 작동합니까?

Cryostat 이미지는 참조 머신 또는 가상 머신에 운영 체제 및 소프트웨어 애플리케이션을 설치하고 구성하여 생성됩니다. 여기에는 시스템 설정, 필요한 드라이버 설치, 패치 및 업데이트 적용, 특정 옵션 및 기본 설정 구성이 포함됩니다.

golden 이미지가 생성되면 여러 클러스터에 복제 및 배포할 수 있는 템플릿 또는 이미지 파일로 저장됩니다. golden 이미지는 필요한 소프트웨어 업데이트 및 패치를 통합하기 위해 유지 관리자에 의해 주기적으로 업데이트할 수 있으므로 이미지가 최신 상태로 유지되고 새로 생성된 VM은 업데이트된 이미지를 기반으로 합니다.

7.1.1.1.2. Red Hat의 골든 이미지 구현

Red Hat은 RHEL(Red Hat Enterprise Linux) 버전의 레지스트리에 골든 이미지를 컨테이너 디스크로 게시합니다. 컨테이너 디스크는 컨테이너 이미지 레지스트리에 컨테이너 이미지로 저장된 가상 머신 이미지입니다. OpenShift Virtualization을 설치한 후 게시된 이미지는 연결된 클러스터에서 자동으로 사용할 수 있습니다. 클러스터에서 이미지를 사용할 수 있게 되면 VM을 생성하는 데 사용할 수 있습니다.

7.1.1.2. VM 부팅 소스 정보

VM(가상 머신)은 VM 정의와 데이터 볼륨에서 지원하는 하나 이상의 디스크로 구성됩니다. VM 템플릿을 사용하면 사전 정의된 사양을 사용하여 VM을 생성할 수 있습니다.

모든 템플릿에는 구성된 드라이버를 포함하여 완전히 구성된 디스크 이미지인 부팅 소스가 필요합니다. 각 템플릿에는 부팅 소스에 대한 포인터가 있는 VM 정의가 포함되어 있습니다. 각 부팅 소스에는 사전 정의된 이름과 네임스페이스가 있습니다. 일부 운영 체제의 경우 부팅 소스가 자동으로 제공됩니다. 제공되지 않는 경우 관리자는 사용자 지정 부팅 소스를 준비해야 합니다.

제공된 부팅 소스는 운영 체제의 최신 버전으로 자동으로 업데이트됩니다. 자동 업데이트된 부팅 소스의 경우 PVC(영구 볼륨 클레임) 및 볼륨 스냅샷은 클러스터의 기본 스토리지 클래스를 사용하여 생성됩니다. 구성 후 다른 기본 스토리지 클래스를 선택하는 경우 이전 기본 스토리지 클래스로 구성된 클러스터 네임스페이스의 기존 부팅 소스를 삭제해야 합니다.

7.1.2. 인스턴스 유형에서 가상 머신 생성

OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 VM을 생성하는지 여부에 관계없이 인스턴스 유형을 사용하여 VM(가상 머신) 생성을 단순화할 수 있습니다.

7.1.2.1. 인스턴스 유형 정보

인스턴스 유형은 재사용 가능한 오브젝트로, 새 VM에 적용할 리소스와 특성을 정의할 수 있습니다. 사용자 지정 인스턴스 유형을 정의하거나 OpenShift Virtualization을 설치할 때 포함된 다양한 유형을 사용할 수 있습니다.

새 인스턴스 유형을 생성하려면 먼저 수동으로 또는 **virtctl** CLI 툴을 사용하여 매니페스트를 생성해야 합니다. 그런 다음 매니페스트를 클러스터에 적용하여 인스턴스 유형 오브젝트를 생성합니다.

OpenShift Virtualization에서는 인스턴스 유형을 구성하기 위한 두 가지 CRD를 제공합니다.

- 네임스페이스가 지정된 오브젝트: **VirtualMachineInstancetype**
- 클러스터 전체 오브젝트: **VirtualMachineClusterInstancetype**

이러한 오브젝트는 동일한 **VirtualMachineInstancetypeSpec** 을 사용합니다.

7.1.2.1.1. 필수 속성

인스턴스 유형을 구성할 때 **cpu** 및 **memory** 속성을 정의해야 합니다. 기타 속성은 선택 사항입니다.

참고

인스턴스 유형에서 VM을 생성할 때 인스턴스 유형에 정의된 매개 변수를 재정의할 수 없습니다.

인스턴스 유형에는 정의된 CPU 및 메모리 속성이 필요하므로 OpenShift Virtualization은 인스턴스 유형에서 VM을 생성할 때 이러한 리소스에 대한 추가 요청을 항상 거부합니다.

인스턴스 유형 매니페스트를 수동으로 생성할 수 있습니다. 예를 들면 다음과 같습니다.

필수 필드가 있는 YAML 파일의 예

```
apiVersion: instancetype.kubevirt.io/v1beta1
kind: VirtualMachineInstancetype
```

```
metadata:
  name: example-instancetype
spec:
  cpu:
    guest: 1 (1)
  memory:
    guest: 128Mi (2)
```

(1) 필수 항목입니다. 게스트에 할당할 vCPU 수를 지정합니다.

(2) 필수 항목입니다. 게스트에 할당할 메모리 양을 지정합니다.

virtctl CLI 유틸리티를 사용하여 인스턴스 유형 매니페스트를 생성할 수 있습니다. 예를 들면 다음과 같습니다.

필수 필드가 있는 **virtctl** 명령의 예

```
$ virtctl create instancetype --cpu 2 --memory 256Mi
```

다음과 같습니다.

--cpu <value>
: 게스트에 할당할 vCPU 수를 지정합니다. 필수 항목입니다.

--memory <value>
: 게스트에 할당할 메모리 양을 지정합니다. 필수 항목입니다.

작은 정보

다음 명령을 실행하여 새 매니페스트에서 오브젝트를 즉시 생성할 수 있습니다.

```
$ virtctl create instancetype --cpu 2 --memory 256Mi | oc apply -f -
```

7.1.2.1.2. 선택적 속성

필수 **cpu** 및 **memory** 속성 외에도 **VirtualMachineInstancetypeSpec** 에 다음과 같은 선택적 속성을 포함할 수 있습니다.

annotations
: VM에 적용할 주석을 나열합니다.

gpus
: 패스스루의 vGPU를 나열합니다.

hostDevices
: 패스스루의 호스트 장치를 나열합니다.

ioThreadsPolicy
: 전용 디스크 액세스를 관리하기 위한 IO 스레드 정책을 정의합니다.

launchSecurity
: SEV(Secure Encrypted Virtualization) 구성.

nodeSelector

이 VM이 예약된 노드를 제어하려면 노드 선택기를 지정합니다.

schedulerName

기본 스케줄러 대신 이 VM에 사용할 사용자 정의 스케줄러를 정의합니다.

7.1.2.2. 사전 정의된 인스턴스 유형

OpenShift Virtualization에는 **common-instancetypes** 라는 사전 정의된 인스턴스 유형 세트가 포함되어 있습니다. 일부는 특정 워크로드에 대해 전문적이며 다른 워크로드는 워크로드와 무관합니다.

이러한 인스턴스 유형 리소스의 이름은 시리즈, 버전 및 크기에 따라 지정됩니다. **size** 값은 **.** 구분 기호 및 **nano** 에서 **8xlarge** 까지의 범위를 따릅니다.

표 7.1. **common-instancetypes** 시리즈 비교

사용 사례	시리즈	특성	vCPU 대 메모리 비율	리소스 예
Universal	U	• Burstable CPU 성능	1:4	**u1.medium** • vCPU 1개 • 4GI 메모리
오버 커밋됨	O	• 오버 커밋된 메모리 • Burstable CPU 성능	1:4	**o1.small** • 1 vCPU • 2GI 메모리
compute-exclusive	CX	• hugepages • 전용 CPU • 격리된 에뮬레이터 스레드 • vNUMA	1:2	**cx1.2xlarge** • 8개의 vCPU • 16GI 메모리
NVIDIA GPU	GN	• NVIDIA GPU Operator에서 제공하는 GPU를 사용하는 VM의 경우 • 사전 정의된 GPU • Burstable CPU 성능	1:4	**gn1.8xlarge** • 32 vCPU • 128GI 메모리

사용 사례	시리즈	특성	vCPU 대 메모리 비율	리소스 예
메모리 집약적	M	• hugepages • Burstable CPU 성능	1:8	**m1.large** • vCPU 2개 • 16GI 메모리
네트워크 집약적	N	• hugepages • 전용 CPU • 격리된 에뮬레이터 스레드 • DPDK 워크로드를 실행할 수 있는 노드 필요	1:2	**n1.medium** • 4개의 vCPU • 4Gi 메모리

7.1.2.3. virtctl 툴을 사용하여 매니페스트 생성

virtctl CLI 유틸리티를 사용하여 VM, VM 인스턴스 유형 및 VM 기본 설정에 대한 매니페스트 생성을 간소화할 수 있습니다. 자세한 내용은 VM 매니페스트 생성 명령을 참조하십시오.

VirtualMachine 매니페스트가 있는 경우 명령줄 에서 VM을 생성할 수 있습니다.

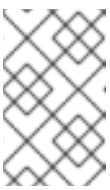
7.1.2.4. 웹 콘솔을 사용하여 인스턴스 유형에서 VM 생성

OpenShift Container Platform 웹 콘솔을 사용하여 인스턴스 유형에서 VM(가상 머신)을 생성할 수 있습니다. 기존 스냅샷을 복사하거나 VM을 복제하여 웹 콘솔을 사용하여 VM을 생성할 수도 있습니다.

사용 가능한 부팅 가능한 볼륨 목록에서 VM을 생성할 수 있습니다. 목록에 Linux 또는 Windows 기반 볼륨을 추가할 수 있습니다.

프로세스

1. 웹 콘솔에서 가상화 → 카탈로그 로 이동합니다.
   InstanceTypes 탭이 기본적으로 열립니다.
2. 다음 옵션 중 하나를 선택합니다.
   - 목록에서 적절한 부팅 가능한 볼륨을 선택합니다. 목록이 잘린 경우 모두 표시 버튼을 클릭하여 전체 목록을 표시합니다.



참고

부팅 가능한 볼륨 테이블에는 **instancetype.kubevirt.io/default-preference** 레이블이 있는 **openshift-virtualization-os-images** 네임스페이스의 볼륨만 나열됩니다.

- 선택 사항: 별 아이콘을 클릭하여 부팅 가능한 볼륨을 즐겨 찾기로 지정합니다. 지정된 부팅 가능한 볼륨이 볼륨 목록에 먼저 표시됩니다.

- 볼륨 추가 를 클릭하여 새 볼륨을 업로드하거나 기존 PVC(영구 볼륨 클레임), 볼륨 스냅샷 또는 **containerDisk** 볼륨을 사용합니다. 저장을 클릭합니다.
  클러스터에서 사용할 수 없는 운영 체제의 로고는 목록 하단에 표시됩니다. 볼륨 추가 링크를 클릭하여 필요한 운영 체제에 볼륨을 추가할 수 있습니다.

  또한 Windows 부팅 소스 생성 빠른 시작에 대한링크가 있습니다. 동일한 링크가 *줄에서 부팅할 볼륨 선택* 옆에 있는 물음표 아이콘 위에 포인터를 가져가면 팝업 창에 나타납니다.

  환경을 설치한 직후 또는 환경의 연결이 끊어지면 부팅할 볼륨 목록이 비어 있습니다. 이 경우 Windows, RHEL 및 Linux의 세 가지 운영 체제 로고가 표시됩니다. 볼륨 추가 버튼을 클릭하여 요구 사항을 충족하는 새 볼륨을 추가할 수 있습니다.

3. 인스턴스 유형 타일을 클릭하고 워크로드에 적합한 리소스 크기를 선택합니다.

4. 선택 사항: 부팅 중인 볼륨에 적용되는 VM 이름을 포함하여 가상 머신 세부 정보를 선택합니다.

   - Linux 기반 볼륨의 경우 다음 단계에 따라 SSH를 구성합니다.

     a. 프로젝트에 공개 SSH 키를 아직 추가하지 않은 경우 VirtualMachine details 섹션에서 인증된 SSH 키 옆에 있는 편집 아이콘을 클릭합니다.

     b. 다음 옵션 중 하나를 선택합니다.

        - Use existing: 시크릿 목록에서 시크릿을 선택합니다.
        - 새 추가: 다음 단계를 따르십시오.
          i. 공개 SSH 키 파일을 찾아보거나 키 필드에 파일을 붙여넣습니다.
          ii. 시크릿 이름을 입력합니다.
          iii. 선택 사항: 이 프로젝트에서 생성한 새 VirtualMachine에 자동으로 이 키 적용을선택합니다.

     c. 저장을 클릭합니다.

   - Windows 볼륨의 경우 다음 단계 세트 중 하나를 수행하여 sysprep 옵션을 구성합니다.
     - Windows 볼륨에 sysprep 옵션을 아직 추가하지 않은 경우 다음 단계를 따르십시오.
       i. VirtualMachine details 섹션에서 edit 아이콘 besideSysprep 을 클릭합니다.
       ii. Autoattend.xml 응답 파일을 추가합니다.
       iii. Unattend.xml 응답 파일을 추가합니다.
       iv. 저장을 클릭합니다.
     - Windows 볼륨에 기존 sysprep 옵션을 사용하려면 다음 단계를 따르십시오.
       i. Attach existing sysprep 을 클릭합니다.
       ii. 기존 sysprep Unattend.xml 응답 파일의 이름을 입력합니다.
       iii. 저장을 클릭합니다.

5. 선택 사항: Windows VM을 생성하는 경우 Windows 드라이버 디스크를 마운트할 수 있습니다.
   a. Customize VirtualMachine 버튼을 클릭합니다.
   b. VirtualMachine 세부 정보 페이지에서 스토리지를 클릭합니다.
   c. Mount Windows drivers disk 확인란을 선택합니다.
6. 선택 사항: YAML 및 CLI 보기를 클릭하여 YAML 파일을 확인합니다. CLI 를 클릭하여 CLI 명령을 확인합니다. YAML 파일 콘텐츠 또는 CLI 명령을 다운로드하거나 복사할 수도 있습니다.
7. VirtualMachine 생성을 클릭합니다.

VM이 생성되면 VirtualMachine 세부 정보 페이지에서 상태를 모니터링할 수 있습니다.

7.1.3. 템플릿에서 가상 머신 생성

OpenShift Container Platform 웹 콘솔을 사용하여 Red Hat 템플릿에서 VM(가상 머신)을 생성할 수 있습니다.

7.1.3.1. VM 템플릿 정보

부팅 소스

사용 가능한 부팅 소스가 있는 템플릿을 사용하여 VM 생성을 신속하게 수행할 수 있습니다. 부팅 소스가 있는 템플릿에는 사용자 지정 레이블이 없는 경우 사용 가능한 부팅 소스에레이블이 지정됩니다.
부팅 소스가 없는 템플릿에는 부팅 소스 필수라고 레이블이 지정됩니다. 사용자 지정 이미지에서 가상 머신 생성 을 참조하십시오.

사용자 정의

VM을 시작하기 전에 디스크 소스 및 VM 매개변수를 사용자 지정할 수 있습니다.

디스크 소스 설정에 대한 자세한 내용은 스토리지 볼륨 유형 및 스토리지 필드를 참조하십시오.

참고

모든 레이블 및 주석이 있는 VM 템플릿을 복사하는 경우 새 버전의 Scheduling, Scale, Performance (SSP) Operator가 배포될 때 템플릿의 버전이 더 이상 사용되지 않는 것으로 표시됩니다. 이 지정을 제거할 수 있습니다. 웹 콘솔을 사용하여 VM 템플릿 사용자 지정을참조하십시오.

단일 노드 OpenShift

스토리지 동작의 차이로 인해 일부 템플릿은 단일 노드 OpenShift와 호환되지 않습니다. 호환성을 유지하려면 데이터 볼륨 또는 스토리지 프로필을 사용하는 템플릿 또는 VM에 대해 **evictionStrategy** 필드를 설정하지 마십시오.

7.1.3.2. 템플릿에서 VM 생성

OpenShift Container Platform 웹 콘솔을 사용하여 사용 가능한 부팅 소스가 있는 템플릿에서 VM(가상 머신)을 생성할 수 있습니다.

선택 사항: VM을 시작하기 전에 템플릿 또는 VM 매개변수(예: 데이터 소스, cloud-init 또는 SSH 키)를 사용자 지정할 수 있습니다.

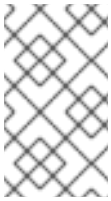
프로세스

1. 웹 콘솔에서 가상화 → 카탈로그 로 이동합니다.
2. 사용 가능한 부팅 소스를 클릭하여 부팅 소스로 템플릿을 필터링합니다.
   카탈로그에 기본 템플릿이 표시됩니다. 모든 항목을 클릭하여 필터에 사용 가능한 모든 템플릿을 확인합니다.
3. 템플릿 타일을 클릭하여 세부 정보를 봅니다.
4. 선택 사항: Windows 템플릿을 사용하는 경우 마운트 Windows 드라이버 디스크 확인란을 선택하여 Windows 드라이버 디스크를 마운트할 수 있습니다.
5. template 또는 VM 매개변수를 사용자 정의할 필요가 없는 경우 빠른 create VirtualMachine를 클릭하여 템플릿에서 VM을 생성합니다.
   template 또는 VM 매개변수를 사용자 지정해야 하는 경우 다음을 수행합니다.
   a. VirtualMachine 사용자 지정을 클릭합니다.
   b. 스토리지 또는 선택적 매개변수를 확장하여 데이터 소스 설정을 편집합니다.
   c. VirtualMachine 매개변수 사용자 지정을 클릭합니다.
      Customize and create VirtualMachine 창에는 Overview,YAML,Scheduling,Environment,Network interfaces,Disks,Scripts, Metadata 탭이 표시됩니다.
   d. VM이 부팅되기 전에 설정해야 하는 매개변수(예: cloud-init 또는 정적 SSH 키)를 편집합니다.
   e. VirtualMachine 생성을 클릭합니다.
      VirtualMachine 세부 정보 페이지에 프로비저닝 상태가 표시됩니다.

7.1.3.2.1. 스토리지 볼륨 유형

표 7.2. 스토리지 볼륨 유형

유형	설명
임시	네트워크 볼륨을 읽기 전용 백업 저장소로 사용하는 로컬 COW(기록 중 복사) 이미지입니다. 백업 볼륨은 PersistentVolumeClaim이어야 합니다. 임시 이미지는 가상 머신이 시작되고 모든 쓰기를 로컬로 저장할 때 생성됩니다. 임시 이미지는 가상 머신이 중지, 재시작 또는 삭제될 때 삭제됩니다. 백업 볼륨(PVC)은 어떤 식으로든 변경되지 않습니다.
persistentVolumeClaim	사용 가능한 PV를 가상 머신에 연결합니다. PV를 연결하면 세션이 바뀌어도 가상 머신 데이터가 지속됩니다. 기존 가상 머신을 OpenShift Container Platform으로 가져올 때는 CDI를 사용하여 기존 가상 머신 디스크를 PVC로 가져와서 PVC를 가상 머신 인스턴스에 연결하는 것이 좋습니다. PVC 내에서 디스크를 사용하려면 몇 가지 요구 사항이 있습니다.

유형	설명
dataVolume	데이터 볼륨은 가져오기, 복제 또는 업로드 작업을 통해 가상 머신 디스크를 준비하는 프로세스를 관리하여 **PersistentVolumeClaim** 디스크 유형에 빌드합니다. 이 볼륨 유형을 사용하는 VM은 볼륨이 준비된 후 시작할 수 있습니다. **type: dataVolume** 또는 **type: ""**로 지정합니다. **type**에 다른 값(예: **persistentVolumeClaim**)을 지정하면 경고가 표시되고 가상 머신이 시작되지 않습니다.
cloudInitNoCloud	참조된 cloud-init NoCloud 데이터 소스가 포함된 디스크를 연결하여 가상 머신에 사용자 데이터 및 메타데이터를 제공합니다. 가상 머신 디스크 내부에 cloud-init을 설치해야 합니다.
containerDisk	컨테이너 이미지 레지스트리에 저장된 가상 머신 디스크와 같은 이미지를 참조합니다. 이 이미지는 가상 머신이 시작될 때 레지스트리에서 가져와서 가상 머신에 디스크로 연결됩니다. **containerDisk** 볼륨은 단일 가상 머신에 국한되지 않으며, 영구 스토리지가 필요하지 않은 다수의 가상 머신 클론을 생성하는 데 유용합니다. 컨테이너 이미지 레지스트리에는 RAW 및 QCOW2 형식의 디스크 유형만 지원됩니다. 이미지 크기를 줄이기 위해 QCOW2를 사용하는 것이 좋습니다.  **참고** **containerDisk**는 임시 볼륨입니다. 이 볼륨은 가상 머신이 중지, 재시작 또는 삭제될 때 삭제됩니다. **containerDisk** 볼륨은 CD-ROM과 같은 읽기 전용 파일 시스템이나 일회용 가상 머신에 유용합니다.
emptyDisk	가상 머신 인터페이스의 라이프사이클에 연결된 추가 스파스(sparse) QCOW2 디스크를 생성합니다. 해당 데이터는 가상 머신에서 게스트가 시작한 재부팅 후에는 유지되지만 가상 머신이 중지되거나 웹 콘솔에서 재시작되면 삭제됩니다. 빈 디스크는 임시 디스크의 제한된 임시 파일 시스템 크기를 초과하지 않도록 애플리케이션 종속성 및 데이터를 저장하는 데 사용됩니다. 디스크 **용량** 크기도 제공해야 합니다.

7.1.3.2.2. 스토리지 필드

필드	설명
비어있음 (PVC 생성)	빈 디스크를 만듭니다.
URL을 통해 가져오기 (PVC 생성)	URL(HTTP 또는 HTTPS 끝점)을 통해 콘텐츠를 가져옵니다.
기존 PVC 사용	클러스터에서 이미 사용 가능한 PVC를 사용합니다.

필드	설명
기존 PVC 복제 (PVC 생성)	클러스터에서 사용 가능한 기존 PVC를 선택하고 복제합니다.
레지스트리를 통해 가져오기(PVC 생성)	컨테이너 레지스트리를 통해 콘텐츠를 가져옵니다.
컨테이너 (임시)	클러스터에서 액세스할 수 있는 레지스트리에 있는 컨테이너에서 콘텐츠를 업로드합니다. 컨테이너 디스크는 CD-ROM 또는 임시 가상 머신과 같은 읽기 전용 파일 시스템에만 사용해야 합니다.
이름	디스크 이름입니다. 이름에는 소문자(**a-z**), 숫자(**0-9**), 하이픈(**-**), 마침표(**.**)가 최대 253자까지 포함될 수 있습니다. 첫 문자와 마지막 문자는 영숫자여야 합니다. 이름에는 대문자, 공백 또는 특수 문자가 포함되어서는 안 됩니다.
크기	디스크 크기(GiB)입니다.
유형	디스크의 유형입니다. 예: 디스크 또는 CD-ROM
인터페이스	디스크 장치의 유형입니다. 지원되는 인터페이스는 **virtIO**, **SATA**, **SCSI**입니다.
스토리지 클래스	디스크를 만드는 데 사용되는 스토리지 클래스입니다.

고급 스토리지 설정
다음 고급 스토리지 설정은 선택 사항이며 Blank 에서 사용할 수 있으며URL을 통해 가져오기, 기존 PVC 복제 디스크에 사용할 수 있습니다.

이러한 매개변수를 지정하지 않으면 시스템은 기본 스토리지 프로필 값을 사용합니다.

매개변수	옵션	매개변수 설명
볼륨 모드	파일 시스템	파일 시스템 기반 볼륨에 가상 디스크를 저장합니다.
	블록	가상 디스크를 블록 볼륨에 직접 저장합니다. 기본 스토리지에서 지원하는 경우에만 **Block**을 사용하십시오.
액세스 모드	ReadWriteOnce (RWO)	볼륨은 단일 노드에서 읽기-쓰기로 마운트할 수 있습니다.
	ReadWriteMany (RWX)	볼륨은 한 번에 여러 노드에서 읽기-쓰기로 마운트할 수 있습니다.  **참고** 이 모드는 실시간 마이그레이션에 필요합니다.

7.1.3.2.3. 웹 콘솔을 사용하여 VM 템플릿 사용자 정의

VM을 시작하기 전에 데이터 소스, cloud-init 또는 SSH 키와 같은 VM 또는 템플릿 매개 변수를 수정하여 기존 VM(가상 머신) 템플릿을 사용자 지정할 수 있습니다. 복사하여 템플릿을 사용자 지정하고 모든 레이블 및 주석을 포함하여 템플릿을 사용자 지정하면 새 버전의 스케줄링, 스케일 및 성능(SSP) Operator가 배포될 때 사용자 지정 템플릿이 더 이상 사용되지 않는 것으로 표시됩니다.

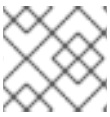
사용자 지정 템플릿에서 더 이상 사용되지 않는 지정을 제거할 수 있습니다.

프로세스

1. 웹 콘솔에서 가상화 → 템플릿 으로 이동합니다.
2. VM 템플릿 목록에서 더 이상 사용되지 않는 것으로 표시된 템플릿을 클릭합니다.
3. 레이블 옆에 있는 연필 아이콘 옆에 있는편집을 클릭합니다.
4. 다음 두 레이블을 제거합니다.
   - **template.kubevirt.io/type: "base"**
   - **template.kubevirt.io/version: "version"**
5. 저장을 클릭합니다.
6. 기존 주석 수 옆에 있는 연필 아이콘을 클릭합니다.
7. 다음 주석을 제거합니다.
   - **template.kubevirt.io/deprecated**
8. 저장을 클릭합니다.

7.1.4. 명령줄에서 가상 머신 생성

VirtualMachine 매니페스트를 편집하거나 생성하여 명령줄에서 VM(가상 머신)을 생성할 수 있습니다. VM 매니페스트에서 인스턴스 유형을 사용하여 VM 구성을 단순화할 수 있습니다.



참고

웹 콘솔을 사용하여 인스턴스 유형에서 VM을 생성할수도 있습니다.

7.1.4.1. virtctl 툴을 사용하여 매니페스트 생성

virtctl CLI 유틸리티를 사용하여 VM, VM 인스턴스 유형 및 VM 기본 설정에 대한 매니페스트 생성을 간소화할 수 있습니다. 자세한 내용은 VM 매니페스트 생성 명령을참조하십시오.

7.1.4.2. VirtualMachine 매니페스트에서 VM 생성

VirtualMachine 매니페스트에서 VM(가상 머신)을 생성할 수 있습니다.

프로세스

1. VM의 **VirtualMachine** 매니페스트를 편집합니다. 다음 예제에서는 RHEL(Red Hat Enterprise Linux) VM을 구성합니다.

참고

이 예제 매니페스트에서는 VM 인증을 구성하지 않습니다.

RHEL VM의 매니페스트 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
 name: rhel-9-minimal
spec:
 dataVolumeTemplates:
  - metadata:
      name: rhel-9-minimal-volume
    spec:
      sourceRef:
        kind: DataSource
        name: rhel9 1
        namespace: openshift-virtualization-os-images 2
      storage: {}
 instancetype:
  name: u1.medium 3
 preference:
  name: rhel.9 4
 running: true
 template:
  spec:
    domain:
      devices: {}
    volumes:
      - dataVolume:
          name: rhel-9-minimal-volume
        name: rootdisk
```

1. rhel9 골든 이미지는 RHEL 9를 게스트 운영 체제로 설치하는 데 사용됩니다.
2. Cryostat 이미지는 openshift-virtualization-os-images 네임스페이스에 저장됩니다.
3. u1.medium 인스턴스 유형은 VM에 대해 1 vCPU 및 4Gi 메모리를 요청합니다. 이러한 리소스 값은 VM 내에서 재정의할 수 없습니다.
4. rhel.9 기본 설정은 RHEL 9 게스트 운영 체제를 지원하는 추가 속성을 지정합니다.

2. 매니페스트 파일을 사용하여 가상 머신을 생성합니다.

```
$ oc create -f <vm_manifest_file>.yaml
```

3. 선택 사항: 가상 머신을 시작합니다.

```
$ virtctl start <vm_name> -n <namespace>
```

다음 단계

- 가상 머신에 대한 SSH 액세스 구성

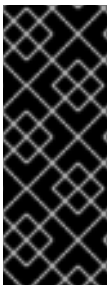
7.2. 사용자 정의 이미지에서 VM 생성

7.2.1. 사용자 정의 이미지에서 가상 머신 생성 개요

다음 방법 중 하나를 사용하여 사용자 정의 운영 체제 이미지에서 VM(가상 머신)을 생성할 수 있습니다.

- 레지스트리에서 이미지를 컨테이너 디스크로 가져옵니다.
  선택 사항: 컨테이너 디스크에 대한 자동 업데이트를 활성화할 수 있습니다. 자세한 내용은 자동 부팅 소스 업데이트 관리를 참조하십시오.
- 웹 페이지에서 이미지를 가져옵니다.
- 로컬 머신에서 이미지 업로드.
- 이미지가 포함된 PVC(영구 볼륨 클레임)를 복제 합니다.

CDI(Containerized Data Importer)는 데이터 볼륨을 사용하여 이미지를 PVC로 가져옵니다. OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 VM에 PVC를 추가합니다.



중요

Red Hat에서 제공하지 않는 운영 체제 이미지에서 생성된 VM에 QEMU 게스트 에이전트를 설치해야 합니다.

Windows VM에 VirtIO 드라이버도 설치해야 합니다.

QEMU 게스트 에이전트는 Red Hat 이미지에 포함되어 있습니다.

7.2.2. 컨테이너 디스크를 사용하여 VM 생성

운영 체제 이미지에서 빌드된 컨테이너 디스크를 사용하여 VM(가상 머신)을 생성할 수 있습니다.

컨테이너 디스크에 대한 자동 업데이트를 활성화할 수 있습니다. 자세한 내용은 자동 부팅 소스 업데이트 관리를 참조하십시오.

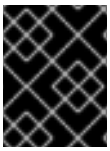
중요

컨테이너 디스크가 크면 I/O 트래픽이 증가할 수 있으므로 작업자 노드를 사용할 수 없게 됩니다. 다음 작업을 수행하여 이 문제를 해결할 수 있습니다.

- DeploymentConfig 오브젝트 정리
- 가비지 컬렉션 구성.

다음 단계를 수행하여 컨테이너 디스크에서 VM을 생성합니다.

1. 운영 체제 이미지를 컨테이너 디스크에 빌드하고 컨테이너 레지스트리에 업로드합니다.
2. 컨테이너 레지스트리에 TLS가 없는 경우 레지스트리에 TLS를 비활성화하도록 환경을 구성합니다.
3. 웹 콘솔 또는 명령줄 을 사용하여 컨테이너 디스크를 디스크 소스로 사용하여 VM을 생성합니다.



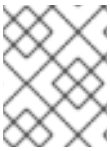
중요

Red Hat에서 제공하지 않는 운영 체제 이미지에서 생성된 VM에 QEMU 게스트 에이전트를 설치해야 합니다.

7.2.2.1. 컨테이너 디스크 빌드 및 업로드

VM(가상 머신) 이미지를 컨테이너 디스크에 빌드하고 레지스트리에 업로드할 수 있습니다.

컨테이너 디스크의 크기는 컨테이너 디스크가 호스팅되는 레지스트리의 최대 계층 크기로 제한됩니다.



참고

Red Hat Quay의 경우 Red Hat Quay를 처음 배포할 때 생성된 YAML 구성 파일을 편집하여 최대 계층 크기를 변경할 수 있습니다.

사전 요구 사항

- **podman**이 설치되어 있어야 합니다.
- QCOW2 또는 RAW 이미지 파일이 있어야 합니다.

프로세스

1. Dockerfile을 생성하여 VM 이미지를 컨테이너 이미지에 빌드합니다. VM 이미지는 UID가 **107** 인 QEMU에 속하고 컨테이너 내부의 **/disk/** 디렉터리에 배치해야 합니다. 그런 다음 **/disk/** 디렉터리에 대한 권한을 **0440**으로 설정해야 합니다.
   다음 예제에서는 Red Hat UBI(Universal Base Image)를 사용하여 첫 번째 단계에서 이러한 구성 변경을 처리하고, 두 번째 단계에서 최소 **scratch** 이미지를 사용하여 결과를 저장합니다.

```
$ cat > Dockerfile << EOF
FROM registry.access.redhat.com/ubi8/ubi:latest AS builder
ADD --chown=107:107 <vm_image>.qcow2 /disk/ \ 1
RUN chmod 0440 /disk/*

FROM scratch
COPY --from=builder /disk/* /disk/
EOF
```

   1 여기서 **<vm_image** >는 QCOW2 또는 RAW 형식의 이미지입니다. 원격 이미지를 사용하는 경우 < **vm_image>.qcow2** 를 전체 URL로 바꿉니다.

2. 컨테이너를 빌드하고 태그를 지정합니다.

```
$ podman build -t <registry>/<container_disk_name>:latest .
```

3. 컨테이너 이미지를 레지스트리에 푸시합니다.

```
$ podman push <registry>/<container_disk_name>:latest
```

7.2.2.2. 컨테이너 레지스트리의 TLS 비활성화

HyperConverged 사용자 정의 리소스의 insecureRegistries 필드를 편집하여 하나 이상의 컨테이너 레지스트리에 대해 TLS(전송 계층 보안)를 비활성화할 수 있습니다.

사전 요구 사항

1. 다음 명령을 실행하여 기본 편집기에서 HyperConverged CR을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 비보안 레지스트리 목록을 spec.storageImport.insecureRegistries 필드에 추가합니다.

HyperConverged 사용자 정의 리소스의 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  storageImport:
    insecureRegistries: 1
      - "private-registry-example-1:5000"
      - "private-registry-example-2:5000"
```

1 이 목록의 예제를 유효한 레지스트리 호스트 이름으로 바꿉니다.

7.2.2.3. 웹 콘솔을 사용하여 컨테이너 디스크에서 VM 생성

OpenShift Container Platform 웹 콘솔을 사용하여 컨테이너 레지스트리에서 컨테이너 디스크를 가져와 VM(가상 머신)을 생성할 수 있습니다.

프로세스

1. 웹 콘솔에서 가상화 → 카탈로그 로 이동합니다.
2. 사용 가능한 부팅 소스 없이 템플릿 타일을 클릭합니다.
3. VirtualMachine 사용자 지정을 클릭합니다.
4. 템플릿 매개변수 사용자 지정 페이지의 스토리지를 확장하고 디스크 소스 목록에서 레지스트리 (PVC 생성) 를 선택합니다.
5. 컨테이너 이미지 URL을 입력합니다. 예: **https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2**
6. 디스크 크기를 설정합니다.
7. 다음을 클릭합니다.
8. VirtualMachine 생성을 클릭합니다.

7.2.2.4. 명령줄을 사용하여 컨테이너 디스크에서 VM 생성

명령줄을 사용하여 컨테이너 디스크에서 VM(가상 머신)을 생성할 수 있습니다.

VM(가상 머신)이 생성되면 컨테이너 디스크가 있는 데이터 볼륨을 영구 스토리지로 가져옵니다.

사전 요구 사항

- 컨테이너 디스크가 포함된 컨테이너 레지스트리에 대한 액세스 인증 정보가 있어야 합니다.

프로세스

1. 컨테이너 레지스트리에 인증이 필요한 경우 인증 정보를 지정하여 시크릿 매니페스트를 생성하고 이를 **data-source-secret.yaml** 파일로 저장합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: data-source-secret
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" 1
  secretKey:   "" 2
```

1. Base64로 인코딩된 키 ID 또는 사용자 이름을 지정합니다.
2. Base64로 인코딩된 시크릿 키 또는 암호를 지정합니다.

2. 다음 명령을 실행하여 시크릿 매니페스트를 적용합니다.

```
$ oc apply -f data-source-secret.yaml
```

3. VM이 시스템 CA 번들에서 서명하지 않은 자체 서명 인증서 또는 인증서를 사용하는 서버와 통신해야 하는 경우 VM과 동일한 네임스페이스에 구성 맵을 생성합니다.

```
$ oc create configmap tls-certs 1
  --from-file=</path/to/file/ca.pem> 2
```

1. 구성 맵 이름을 지정합니다.
2. CA 인증서의 경로를 지정합니다.

4. **VirtualMachine** 매니페스트를 편집하고 **vm-fedora-datavolume.yaml** 파일로 저장합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    kubevirt.io/vm: vm-fedora-datavolume
  name: vm-fedora-datavolume 1
spec:
```

```
  dataVolumeTemplates:
  - metadata:
      creationTimestamp: null
      name: fedora-dv (2)
    spec:
      storage:
        resources:
          requests:
            storage: 10Gi (3)
        storageClassName: <storage_class> (4)
      source:
        registry:
          url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest" (5)
          secretRef: data-source-secret (6)
          certConfigMap: tls-certs (7)
    status: {}
  running: true
  template:
    metadata:
      creationTimestamp: null
      labels:
        kubevirt.io/vm: vm-fedora-datavolume
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: datavolumedisk1
        machine:
          type: ""
        resources:
          requests:
            memory: 1.5Gi
      terminationGracePeriodSeconds: 180
      volumes:
      - dataVolume:
          name: fedora-dv
        name: datavolumedisk1
status: {}
```

(1) VM의 이름을 지정합니다.

(2) 데이터 볼륨의 이름을 지정합니다.

(3) 데이터 볼륨에 요청된 스토리지의 크기를 지정합니다.

(4) 선택 사항: 스토리지 클래스를 지정하지 않으면 기본 스토리지 클래스가 사용됩니다.

(5) 컨테이너 레지스트리의 URL을 지정합니다.

(6) 선택 사항: 컨테이너 레지스트리 액세스 인증 정보의 시크릿을 생성한 경우 시크릿 이름을 지정합니다.

(7) 선택 사항: CA 인증서 구성 맵을 지정합니다.

5. 다음 명령을 실행하여 VM을 생성합니다.

```
$ oc create -f vm-fedora-datavolume.yaml
```

oc create 명령은 데이터 볼륨과 VM을 생성합니다. CDI 컨트롤러는 올바른 주석을 사용하여 기본 PVC를 생성하고 가져오기 프로세스가 시작됩니다. 가져오기가 완료되면 데이터 볼륨 상태가 성공으로 변경됩니다. VM을 시작할 수 있습니다.

데이터 볼륨 프로비저닝은 백그라운드에서 수행되므로 프로세스를 모니터링할 필요가 없습니다.

검증

1. 가져오기 Pod는 지정된 URL에서 컨테이너 디스크를 다운로드하여 프로비저닝된 영구 볼륨에 저장합니다. 다음 명령을 실행하여 가져오기 Pod의 상태를 확인합니다.

```
$ oc get pods
```

2. 다음 명령을 실행하여 상태가 성공 될 때까지 데이터 볼륨을 모니터링합니다.

```
$ oc describe dv fedora-dv 1
```

1 **VirtualMachine** 매니페스트에 정의된 데이터 볼륨 이름을 지정합니다.

3. 직렬 콘솔에 액세스하여 프로비저닝이 완료되었고 VM이 시작되었는지 확인합니다.

```
$ virtctl console vm-fedora-datavolume
```

7.2.3. 웹 페이지에서 이미지를 가져와 VM 생성

웹 페이지에서 운영 체제 이미지를 가져와 VM(가상 머신)을 생성할 수 있습니다.

중요

Red Hat에서 제공하지 않는 운영 체제 이미지에서 생성된 VM에 QEMU 게스트 에이전트를 설치해야 합니다.

7.2.3.1. 웹 콘솔을 사용하여 웹 페이지의 이미지에서 VM 생성

OpenShift Container Platform 웹 콘솔을 사용하여 웹 페이지에서 이미지를 가져와 VM(가상 머신)을 생성할 수 있습니다.

사전 요구 사항

- 이미지가 포함된 웹 페이지에 액세스할 수 있어야 합니다.

프로세스

1. 웹 콘솔에서 가상화 → 카탈로그 로 이동합니다.
2. 사용 가능한 부팅 소스 없이 템플릿 타일을 클릭합니다.
3. VirtualMachine 사용자 지정을 클릭합니다.

4. 템플릿 매개변수 사용자 지정 페이지의 스토리지를 확장하고 디스크 소스 목록에서 URL(PVC 생성) 을 선택합니다.

5. 이미지 URL을 입력합니다. 예: **https://access.redhat.com/downloads/content/69/ver=/rhel---7/7.9/x86_64/product-software**

6. 컨테이너 이미지 URL을 입력합니다. 예: **https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2**

7. 디스크 크기를 설정합니다.

8. 다음을 클릭합니다.

9. VirtualMachine 생성을 클릭합니다.

7.2.3.2. 명령줄을 사용하여 웹 페이지의 이미지에서 VM 생성

명령줄을 사용하여 웹 페이지의 이미지에서 VM(가상 머신)을 생성할 수 있습니다.

VM(가상 머신)이 생성되면 이미지가 포함된 데이터 볼륨을 영구 스토리지로 가져옵니다.

사전 요구 사항

- 이미지가 포함된 웹 페이지에 대한 액세스 인증 정보가 있어야 합니다.

프로세스

1. 웹 페이지에 인증이 필요한 경우 인증 정보를 지정하여 시크릿 매니페스트를 생성하고 이를 **data-source-secret.yaml** 파일로 저장합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: data-source-secret
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" 1
  secretKey:   "" 2
```

1 Base64로 인코딩된 키 ID 또는 사용자 이름을 지정합니다.

2 Base64로 인코딩된 시크릿 키 또는 암호를 지정합니다.

2. 다음 명령을 실행하여 시크릿 매니페스트를 적용합니다.

```
$ oc apply -f data-source-secret.yaml
```

3. VM이 시스템 CA 번들에서 서명하지 않은 자체 서명 인증서 또는 인증서를 사용하는 서버와 통신해야 하는 경우 VM과 동일한 네임스페이스에 구성 맵을 생성합니다.

```
$ oc create configmap tls-certs (1)
  --from-file=</path/to/file/ca.pem> (2)
```

(1) 구성 맵 이름을 지정합니다.

(2) CA 인증서의 경로를 지정합니다.

4. **VirtualMachine** 매니페스트를 편집하고 **vm-fedora-datavolume.yaml** 파일로 저장합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    kubevirt.io/vm: vm-fedora-datavolume
  name: vm-fedora-datavolume (1)
spec:
  dataVolumeTemplates:
  - metadata:
      creationTimestamp: null
      name: fedora-dv (2)
    spec:
      storage:
        resources:
          requests:
            storage: 10Gi (3)
        storageClassName: <storage_class> (4)
      source:
        http:
          url:
"https://mirror.arizona.edu/fedora/linux/releases/35/Cloud/x86_64/images/Fedora-
Cloud-Base-35-1.2.x86_64.qcow2" (5)
        registry:
          url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest" (6)
          secretRef: data-source-secret (7)
          certConfigMap: tls-certs (8)
    status: {}
  running: true
  template:
    metadata:
      creationTimestamp: null
      labels:
        kubevirt.io/vm: vm-fedora-datavolume
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: datavolumedisk1
        machine:
          type: ""
        resources:
```

```
        requests:
          memory: 1.5Gi
      terminationGracePeriodSeconds: 180
      volumes:
      - dataVolume:
          name: fedora-dv
        name: datavolumedisk1
status: {}
```

1. VM의 이름을 지정합니다.
2. 데이터 볼륨의 이름을 지정합니다.
3. 데이터 볼륨에 요청된 스토리지의 크기를 지정합니다.
4. 선택 사항: 스토리지 클래스를 지정하지 않으면 기본 스토리지 클래스가 사용됩니다.
5. 6 웹 페이지의 URL을 지정합니다.
7. 선택 사항: 웹 페이지 액세스 인증 정보에 대한 시크릿을 생성한 경우 시크릿 이름을 지정합니다.
8. 선택 사항: CA 인증서 구성 맵을 지정합니다.

5. 다음 명령을 실행하여 VM을 생성합니다.

```
$ oc create -f vm-fedora-datavolume.yaml
```

oc create 명령은 데이터 볼륨과 VM을 생성합니다. CDI 컨트롤러는 올바른 주석을 사용하여 기본 PVC를 생성하고 가져오기 프로세스가 시작됩니다. 가져오기가 완료되면 데이터 볼륨 상태가 성공으로 변경됩니다. VM을 시작할 수 있습니다.

데이터 볼륨 프로비저닝은 백그라운드에서 수행되므로 프로세스를 모니터링할 필요가 없습니다.

검증

1. 가져오기 Pod는 지정된 URL에서 이미지를 다운로드하여 프로비저닝된 영구 볼륨에 저장합니다. 다음 명령을 실행하여 가져오기 Pod의 상태를 확인합니다.

```
$ oc get pods
```

2. 다음 명령을 실행하여 상태가 성공 될 때까지 데이터 볼륨을 모니터링합니다.

```
$ oc describe dv fedora-dv 1
```

1. **VirtualMachine** 매니페스트에 정의된 데이터 볼륨 이름을 지정합니다.

3. 직렬 콘솔에 액세스하여 프로비저닝이 완료되었고 VM이 시작되었는지 확인합니다.

```
$ virtctl console vm-fedora-datavolume
```

7.2.4. 이미지 업로드를 통해 VM 생성

로컬 머신에서 운영 체제 이미지를 업로드하여 VM(가상 머신)을 생성할 수 있습니다.

PVC에 Windows 이미지를 업로드하여 Windows VM을 생성할 수 있습니다. 그런 다음 VM을 생성할 때 PVC를 복제합니다.

중요

Red Hat에서 제공하지 않는 운영 체제 이미지에서 생성된 VM에 QEMU 게스트 에이전트를 설치해야 합니다.

Windows VM에 VirtIO 드라이버도 설치해야 합니다.

7.2.4.1. 웹 콘솔을 사용하여 업로드된 이미지에서 VM 생성

OpenShift Container Platform 웹 콘솔을 사용하여 업로드된 운영 체제 이미지에서 VM(가상 머신)을 생성할 수 있습니다.

사전 요구 사항

- **IMG,ISO** 또는 **QCOW2** 이미지 파일이 있어야 합니다.

프로세스

1. 웹 콘솔에서 가상화 → 카탈로그로 이동합니다.
2. 사용 가능한 부팅 소스 없이 템플릿 타일을 클릭합니다.
3. VirtualMachine 사용자 지정을 클릭합니다.
4. Customize template parameters 페이지에서 Storage를 확장하고 디스크 소스 목록에서 새 파일을 PVC에 업로드(업로드)를 선택합니다.
5. 로컬 시스템의 이미지를 검색하여 디스크 크기를 설정합니다.
6. VirtualMachine 사용자 지정을 클릭합니다.
7. VirtualMachine 생성을 클릭합니다.

7.2.4.2. Windows VM 생성

PVC(영구 볼륨 클레임)에 Windows 이미지를 업로드한 다음 OpenShift Container Platform 웹 콘솔을 사용하여 VM을 생성할 때 PVC를 복제하여 Windows 가상 머신(VM)을 생성할 수 있습니다.

사전 요구 사항

- Windows 미디어 생성 도구를 사용하여 Windows 설치 DVD 또는 USB를 생성했습니다. Microsoft 문서의 Windows 10 설치 미디어 만들기를 참조하십시오.
- **autounattend.xml** 응답 파일을 생성했습니다. Microsoft 문서의 Answer 파일(unattend.xml)을 참조하십시오.

프로세스

1. Windows 이미지를 새 PVC로 업로드합니다.

a. 웹 콘솔에서 스토리지 → PersistentVolumeClaims 로 이동합니다.

b. 영구 볼륨 클레임 생성 → 데이터 업로드 양식 을 클릭합니다.

c. Windows 이미지로 이동하여 선택합니다.

d. PVC 이름을 입력하고 스토리지 클래스와 크기를 선택한 다음 업로드 를 클릭합니다. Windows 이미지가 PVC에 업로드됩니다.

2. 업로드된 PVC를 복제하여 새 VM을 구성합니다.

a. 가상화 → 카탈로그 로 이동합니다.

b. Windows 템플릿 타일을 선택하고 Customize VirtualMachine 을 클릭합니다.

c. 디스크 소스 목록에서 Clone (clone PVC) 을 선택합니다.

d. PVC 프로젝트, Windows 이미지 PVC, 디스크 크기를 선택합니다.

3. 응답 파일을 VM에 적용합니다.

a. VirtualMachine 매개변수 사용자 지정을 클릭합니다.

b. 스크립트 탭의 Sysprep 섹션에서 편집 을 클릭합니다.

c. **autounattend.xml** 응답 파일을 찾아저장을 클릭합니다.

4. VM의 실행 전략을 설정합니다.

a. VM 이 즉시 시작되지 않도록 생성 후 이 VirtualMachine시작을 지웁니다.

b. VirtualMachine 생성을 클릭합니다.

c. YAML 탭에서 **running:false** 를 **runStrategy: RerunOnFailure** 로 바꾸고저장을 클릭합니다.

5. 옵션 메뉴  를 클릭하고 시작을 선택합니다.
VM은 **autounattend.xml** 응답 파일이 포함된 **sysprep** 디스크에서 부팅됩니다.

7.2.4.2.1. Windows VM 이미지 일반화

이미지를 사용하여 새 VM(가상 머신)을 생성하기 전에 Windows 운영 체제 이미지를 일반화하여 모든 시스템 관련 구성 데이터를 제거할 수 있습니다.

VM을 일반화하기 전에는 무인 Windows 설치 후 **sysprep** 툴에서 응답 파일을 감지할 수 없는지 확인해야 합니다.

사전 요구 사항

- QEMU 게스트 에이전트가 설치된 실행 중인 Windows VM.

프로세스

1. OpenShift Container Platform 콘솔에서 가상화 → VirtualMachines 를 클릭합니다.

2. Windows VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.

3. 구성 → 디스크 를 클릭합니다.
4. **sysprep** 디스크 옆에 있는 옵션 메뉴  를 클릭하고 Detach 를 선택합니다.
5. Detach 를 클릭합니다.
6. **sysprep** 툴의 탐지를 방지하기 위해 **C:\Windows\Panther\unattend.xml** 의 이름을 변경합니다.
7. 다음 명령을 실행하여 **sysprep** 프로그램을 시작합니다.

```
%WINDIR%\System32\Sysprep\sysprep.exe /generalize /shutdown /oobe /mode:vm
```

8. **sysprep** 툴이 완료되면 Windows VM이 종료됩니다. 이제 VM의 디스크 이미지를 Windows VM의 설치 이미지로 사용할 수 있습니다.

이제 VM을 전문으로 지정할 수 있습니다.

7.2.4.2.2. Windows VM 이미지 특수화

Windows VM(가상 머신)을 전문으로 지정하면 일반화된 Windows 이미지에서 VM에 대한 컴퓨터 관련 정보가 구성됩니다.

사전 요구 사항

- 일반화된 Windows 디스크 이미지가 있어야 합니다.
- **unattend.xml** 응답 파일을 생성해야 합니다. 자세한 내용은Microsoft 설명서 를 참조하십시오.

프로세스

1. OpenShift Container Platform 콘솔에서 가상화 → 카탈로그 를 클릭합니다.
2. Windows 템플릿을 선택하고 Customize VirtualMachine 을 클릭합니다.
3. 디스크 소스 목록에서 PVC(복제 PVC) 를 선택합니다.
4. 일반 Windows 이미지의 PVC 프로젝트 및 PVC 이름을 선택합니다.
5. VirtualMachine 매개변수 사용자 지정을 클릭합니다.
6. 스크립트 탭을 클릭합니다.
7. Sysprep 섹션에서 편집 을 클릭하고 **unattend.xml** 응답 파일로 이동한 다음 저장을 클릭합니다.
8. VirtualMachine 생성을 클릭합니다.

초기 부팅 중에 Windows는 **unattend.xml** 응답 파일을 사용하여 VM을 전문으로 지정합니다. 이제 VM을 사용할 준비가 되었습니다.

Windows VM 생성을 위한 추가 리소스

- Microsoft, Sysprep (Generalize) Windows 설치
- Microsoft, generalize

- Microsoft, specialize

7.2.4.3. 명령줄을 사용하여 업로드된 이미지에서 VM 생성

virtctl 명령줄 툴을 사용하여 운영 체제 이미지를 업로드할 수 있습니다. 기존 데이터 볼륨을 사용하거나 이미지에 대한 새 데이터 볼륨을 생성할 수 있습니다.

사전 요구 사항

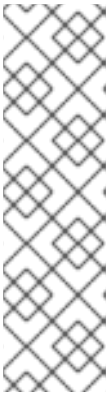
- **ISO,IMG** 또는 **QCOW2** 운영 체제 이미지 파일이 있어야 합니다.
- 최상의 성능을 위해 virt-sparsify 도구 또는 **xz** 또는 **gzip** 유틸리티를 사용하여 이미지 파일을 압축합니다.
- **virtctl** 이 설치되어 있어야 합니다.
- 클라이언트 머신이 OpenShift Container Platform 라우터의 인증서를 신뢰하도록 구성되어 있어야 합니다.

프로세스

1. **virtctl image-upload** 명령을 실행하여 이미지를 업로드합니다.

```
$ virtctl image-upload dv <datavolume_name> \ 1
  --size=<datavolume_size> \ 2
  --image-path=</path/to/image> \ 3
```

1. 데이터 볼륨의 이름입니다.
2. 데이터 볼륨의 크기입니다. 예를 들면 **--size=500Mi, --size=1G**와 같습니다.
3. 이미지의 파일 경로입니다.



참고

- 새 데이터 볼륨을 생성하지 않으려면 **--size** 매개변수를 생략하고 **--no-create** 플래그를 포함합니다.
- 디스크 이미지를 PVC에 업로드할 때 PVC 크기는 압축되지 않은 가상 디스크의 크기보다 커야 합니다.
- HTTPS를 사용할 때 비보안 서버 연결을 허용하려면 **--insecure** 매개변수를 사용하십시오. **--insecure** 플래그를 사용하면 업로드 끝점의 진위 여부를 확인하지 않습니다.

2. 선택 사항: 데이터 볼륨이 생성되었는지 확인하려면 다음 명령을 실행하여 모든 데이터 볼륨을 확인합니다.

```
$ oc get dvs
```

7.2.5. QEMU 게스트 에이전트 및 VirtIO 드라이버 설치

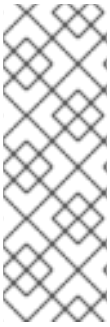
QEMU 게스트 에이전트는 VM(가상 머신)에서 실행되고 VM, 사용자, 파일 시스템 및 보조 네트워크에 대한 정보를 호스트에 전달하는 데몬입니다.

Red Hat에서 제공하지 않는 운영 체제 이미지에서 생성된 VM에 QEMU 게스트 에이전트를 설치해야 합니다.

7.2.5.1. QEMU 게스트 에이전트 설치

7.2.5.1.1. Linux VM에 QEMU 게스트 에이전트 설치

qemu-guest-agent 는 널리 사용 가능하며 RHEL(Red Hat Enterprise Linux) 가상 머신(VM)에서 기본적으로 사용할 수 있습니다. 에이전트를 설치하고 서비스를 시작합니다.



참고

가장 높은 무결성을 가진 온라인(실행 상태) VM의 스냅샷을 생성하려면 QEMU 게스트 에이전트를 설치합니다.

QEMU 게스트 에이전트는 시스템 워크로드에 따라 VM 파일 시스템을 가능한 한 많이 정지하여 일관된 스냅샷을 사용합니다. 이렇게 하면 스냅샷을 생성하기 전에 진행 중인 I/O가 디스크에 기록됩니다. 게스트 에이전트가 없으면 정지를 수행할 수 없으며 최상의 스냅샷을 생성합니다. 스냅샷이 수행된 조건은 웹 콘솔 또는 CLI에 표시되는 스냅샷 표시에 반영됩니다.

절차

1. 콘솔 또는 SSH를 사용하여 VM에 로그인합니다.
2. 다음 명령을 실행하여 QEMU 게스트 에이전트를 설치합니다.

```
$ yum install -y qemu-guest-agent
```

3. 서비스가 지속되는지 확인하고 다음을 시작합니다.

```
$ systemctl enable --now qemu-guest-agent
```

검증

- 다음 명령을 실행하여 **AgentConnected** 가 VM 사양에 나열되어 있는지 확인합니다.

```
$ oc get vm <vm_name>
```

7.2.5.1.2. Windows VM에 QEMU 게스트 에이전트 설치

Windows 가상 머신(VM)의 경우 QEMU 게스트 에이전트는 VirtIO 드라이버에 포함됩니다. Windows 설치 중 또는 기존 Windows VM에 드라이버를 설치할 수 있습니다.

참고

가장 높은 무결성을 가진 온라인(실행 상태) VM의 스냅샷을 생성하려면 QEMU 게스트 에이전트를 설치합니다.

QEMU 게스트 에이전트는 시스템 워크로드에 따라 VM 파일 시스템을 가능한 한 많이 정지하여 일관된 스냅샷을 사용합니다. 이렇게 하면 스냅샷을 생성하기 전에 진행 중인 I/O가 디스크에 기록됩니다. 게스트 에이전트가 없으면 정지를 수행할 수 없으며 최상의 스냅샷을 생성합니다. 스냅샷이 수행된 조건은 웹 콘솔 또는 CLI에 표시되는 스냅샷 표시에 반영됩니다.

프로세스

1. Windows 게스트 운영 체제에서 File Explorer 를 사용하여 **virtio-win** CD 드라이브의 **guest-agent** 디렉터리로 이동합니다.
2. **qemu-ga-x86_64.msi** 설치 프로그램을 실행합니다.

검증

1. 다음 명령을 실행하여 네트워크 서비스 목록을 가져옵니다.

```
$ net start
```

2. 출력에 **QEMU** 게스트 에이전트 가 포함되어 있는지 확인합니다.

7.2.5.2. Windows VM에 VirtIO 드라이버 설치

virtio 드라이버는 Microsoft Windows VM(가상 머신)을 OpenShift Virtualization에서 실행하는 데 필요한 반가상화 장치 드라이버입니다. 드라이버는 나머지 이미지와 함께 제공되며 별도의 다운로드가 필요하지 않습니다.

드라이버 설치를 활성화하려면 **container-native-virtualization/virtio-win** 컨테이너 디스크를 VM에 SATA CD 드라이브로 연결해야 합니다. VirtIO 드라이버는 Windows 설치 중에 설치하거나 기존 Windows 설치에 추가할 수 있습니다.

드라이버가 설치되면 **container-native-virtualization/virtio-win** 컨테이너 디스크를 VM에서 제거할 수 있습니다.

표 7.3. 지원되는 드라이버

드라이버 이름	하드웨어 ID	설명
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	블록 드라이버입니다. **기타 장치** 그룹에서 **SCSI 컨트롤러로** 레이블이 지정되는 경우가 있습니다.
viorng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	엔트로피 소스 드라이버입니다. **기타 장치 그룹에서 PCI 장치로** 레이블이 지정되는 경우도 있습니다.

드라이버 이름	하드웨어 ID	설명
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	네트워크 드라이버입니다. **기타 장치** 그룹에서 **이더넷 컨트롤러로** 레이블이 지정되기도 합니다. VirtIO NIC가 구성된 경우에만 사용할 수 있습니다.

7.2.5.2.1. 설치 중에 VirtIO 컨테이너 디스크를 Windows VM에 연결

필요한 Windows 드라이버를 설치하려면 VirtIO 컨테이너 디스크를 Windows VM에 연결해야 합니다. 이 작업은 VM 생성 중에 수행할 수 있습니다.

프로세스

1. 템플릿에서 Windows VM을 생성할 때 VirtualMachine 사용자 지정을 클릭합니다.
2. Windows 드라이버 디스크 마운트를 선택합니다.
3. Customize VirtualMachine 매개변수를 클릭합니다.
4. VirtualMachine 생성을 클릭합니다.

VM이 생성되면 **virtio-win** SATA CD 디스크가 VM에 연결됩니다.

7.2.5.2.2. 기존 Windows VM에 VirtIO 컨테이너 디스크 연결

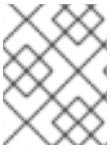
필요한 Windows 드라이버를 설치하려면 VirtIO 컨테이너 디스크를 Windows VM에 연결해야 합니다. 이 작업은 기존 VM에 수행할 수 있습니다.

프로세스

1. 기존 Windows VM으로 이동하여 작업 → 중지 를 클릭합니다.
2. VM 세부 정보 → 구성 → 디스크로 이동하여 디스크 추가 를 클릭합니다.
3. 컨테이너 소스에서 **windows-driver-disk** 를 추가하고 유형을 CD-ROM 으로 설정한 다음 인터페이스를 SATA 로 설정합니다.
4. 저장을 클릭합니다.
5. VM을 시작하고 그래픽 콘솔에 연결합니다.

7.2.5.2.3. Windows 설치 중 VirtIO 드라이버 설치

VM(가상 머신)에 Windows를 설치하는 동안 VirtIO 드라이버를 설치할 수 있습니다.



참고

이 절차에서는 일반적인 Windows 설치 방법을 사용하며, 설치 방법은 Windows 버전마다 다를 수 있습니다. 설치 중인 Windows 버전에 대한 설명서를 참조하십시오.

사전 요구 사항

- virtio 드라이버가 포함된 스토리지 장치를 VM에 연결해야 합니다.

프로세스

1. Windows 운영 체제에서 File Explorer 를 사용하여 virtio-win CD 드라이브로 이동합니다.
2. 드라이브를 두 번 클릭하여 VM에 적절한 설치 프로그램을 실행합니다.
   64비트 vCPU의 경우 virtio-win-gt-x64 설치 프로그램을 선택합니다. 32비트 vCPU는 더 이상 지원되지 않습니다.
3. 선택 사항: 설치 프로그램의 사용자 지정 설정 단계에서 설치할 장치 드라이버를 선택합니다. 권장 드라이버 세트는 기본적으로 선택됩니다.
4. 설치가 완료되면 완료 를 선택합니다.
5. VM을 재부팅합니다.

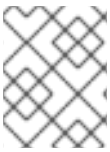
검증

1. PC에서 시스템 디스크를 엽니다. 일반적으로 이 값은 C: 입니다.
2. 프로그램 파일 → Virtio-Win 으로 이동합니다.

Virtio-Win 디렉터리가 있고 각 드라이버의 하위 디렉터리가 포함된 경우 설치에 성공했습니다.

7.2.5.2.4. 기존 Windows VM의 SATA CD 드라이브에서 VirtIO 드라이버 설치

기존 Windows VM(가상 머신)의 SATA CD 드라이브에서 VirtIO 드라이버를 설치할 수 있습니다.



참고

다음 절차에서는 일반적인 방법을 사용하여 Windows에 드라이버를 추가합니다. 특정 설치 단계는 사용 중인 Windows 버전의 설치 설명서를 참조하십시오.

사전 요구 사항

- virtio 드라이버가 포함된 스토리지 장치를 VM에 SATA CD 드라이브로 연결해야 합니다.

절차

1. VM을 시작하고 그래픽 콘솔에 연결합니다.
2. Windows 사용자 세션에 로그인합니다.
3. 장치 관리자를 열고 기타 장치를 확장하여 알 수 없는 장치를 나열합니다.
   a. 장치 속성을 열어 알 수 없는 장치를 식별합니다.
   b. 장치를 마우스 오른쪽 버튼으로 클릭하고 속성을 선택합니다.
   c. 세부 정보 탭을 클릭하고 속성 목록에서 하드웨어 ID를 선택합니다.
   d. 하드웨어 ID의 값을 지원되는 VirtIO 드라이버와 비교합니다.
4. 장치를 마우스 오른쪽 단추로 클릭하고 드라이버 소프트웨어 업데이트를 선택합니다.

5. 컴퓨터에서 드라이버 소프트웨어 찾아보기를 클릭하고 VirtIO 드라이버가 있는 연결된 SATA CD 드라이브를 찾습니다. 드라이버는 드라이버 유형, 운영 체제, CPU 아키텍처에 따라 계층적으로 정렬됩니다.

6. 다음을 클릭하여 드라이버를 설치합니다.

7. 필요한 모든 VirtIO 드라이버에 대해 이 과정을 반복합니다.

8. 드라이버 설치 후 닫기를 클릭하여 창을 닫습니다.

9. VM을 재부팅하여 드라이버 설치를 완료합니다.

7.2.5.2.5. 컨테이너 디스크에서 VirtIO 드라이버 설치 SATA CD 드라이브로 추가됨

Windows VM(가상 머신)에 SATA CD 드라이브로 추가하는 컨테이너 디스크에서 VirtIO 드라이버를 설치할 수 있습니다.

작은 정보

Red Hat Ecosystem Catalog에서 **container-native-virtualization/virtio-win** 컨테이너 디스크를 다운로드하는 것은 클러스터에 없는 경우 Red Hat 레지스트리에서 컨테이너 디스크를 다운로드하기 때문에 필수가 아닙니다. 그러나 다운로드하면 설치 시간이 줄어듭니다.

사전 요구 사항

- 제한된 환경에서 Red Hat 레지스트리 또는 다운로드한 **container-native-virtualization/virtio-win** 컨테이너 디스크에 액세스할 수 있어야 합니다.

프로세스

1. **VirtualMachine** 매니페스트를 편집하여 **container-native-virtualization/virtio-win** 컨테이너 디스크를 CD 드라이브로 추가합니다.

```
# ...
spec:
  domain:
    devices:
      disks:
        - name: virtiocontainerdisk
          bootOrder: 2 (1)
          cdrom:
            bus: sata
volumes:
  - containerDisk:
      image: container-native-virtualization/virtio-win
    name: virtiocontainerdisk
```

(1) OpenShift Virtualization은 **VirtualMachine** 매니페스트에 정의된 순서대로 VM 디스크를 부팅합니다. container **-native-virtualization/virtio-win** 컨테이너 디스크 전에 부팅되는 다른 VM 디스크를 정의하거나 선택적 **bootOrder** 매개변수를 사용하여 VM이 올바른 디스크에서 부팅되도록 할 수 있습니다. 디스크의 부팅 순서를 구성하는 경우 다른 디스크에 대한 부팅 순서를 구성해야 합니다.

2. 변경 사항을 적용합니다.

- VM이 실행 중이 아닌 경우 다음 명령을 실행합니다.

```
$ virtctl start <vm> -n <namespace>
```

- VM이 실행 중인 경우 VM을 재부팅하거나 다음 명령을 실행합니다.

```
$ oc apply -f <vm.yaml>
```

3. VM을 시작한 후 SATA CD 드라이브에서 VirtIO 드라이버를 설치합니다.

7.2.5.3. VirtIO 드라이버 업데이트

7.2.5.3.1. Windows VM에서 VirtIO 드라이버 업데이트

Windows Update 서비스를 사용하여 Windows VM(가상 머신)에서 **virtio** 드라이버를 업데이트합니다.

사전 요구 사항

- 클러스터가 인터넷에 연결되어 있어야 합니다. 연결이 끊긴 클러스터는 Windows Update 서비스에 연결할 수 없습니다.

프로세스

1. Windows 게스트 운영 체제에서 Windows 키를 클릭하고 설정을선택합니다.
2. Windows 업데이트 → 고급 옵션 → 선택적 업데이트로 이동합니다.
3. Red Hat, Inc.의 모든 업데이트를 설치합니다.
4. VM을 재부팅합니다.

검증

1. Windows VM에서 장치 관리자로 이동합니다.
2. 장치를 선택합니다.
3. 드라이버 탭을 선택합니다.
4. 드라이버 세부 정보를 클릭하고 **virtio** 드라이버 세부 정보에 올바른 버전이 표시되는지 확인합니다.

7.2.6. VM 복제

VM(가상 머신)을 복제하거나 스냅샷에서 새 VM을 생성할 수 있습니다.

7.2.6.1. 웹 콘솔을 사용하여 VM 복제

웹 콘솔을 사용하여 기존 VM을 복제할 수 있습니다.

프로세스

1. 웹 콘솔에서 가상화 → VirtualMachines 로 이동합니다.

2. VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 작업을 클릭합니다.
4. Clone 을 선택합니다.
5. Clone VirtualMachine 페이지에서 새 VM의 이름을 입력합니다.
6. (선택 사항) 복제된 VM 시작 확인란을 선택하여 복제된 VM을 시작합니다.
7. Clone 을 클릭합니다.

7.2.6.2. 웹 콘솔을 사용하여 기존 스냅샷에서 VM 생성

기존 스냅샷을 복사하여 새 VM을 생성할 수 있습니다.

프로세스

1. 웹 콘솔에서 가상화 → VirtualMachines 로 이동합니다.
2. VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 스냅샷 탭을 클릭합니다.
4. 복사할 스냅샷의 작업 메뉴  를 클릭합니다.
5. Create VirtualMachine 를 선택합니다.
6. 가상 머신의 이름을 입력합니다.
7. (선택 사항) 생성 후 이 VirtualMachine 시작 확인란을 선택하여 새 가상 머신을 시작합니다.
8. 생성을 클릭합니다.

7.2.6.3. 추가 리소스

- PVC 복제를 통해 VM 생성

7.2.7. PVC 복제를 통해 VM 생성

사용자 정의 이미지로 기존 PVC(영구 볼륨 클레임)를 복제하여 VM(가상 머신)을 생성할 수 있습니다.

Red Hat에서 제공하지 않는 운영 체제 이미지에서 생성된 VM에 QEMU 게스트 에이전트 를 설치해야 합니다.

소스 PVC를 참조하는 데이터 볼륨을 생성하여 PVC를 복제합니다.

7.2.7.1. 복제 정보

데이터 볼륨을 복제할 때 CDI(Containerized Data Importer)는 다음 CSI(Container Storage Interface) 복제 방법 중 하나를 선택합니다.

- CSI 볼륨 복제

- 스마트 복제

CSI 볼륨 복제 및 스마트 복제 방법 모두 효율적이지만 사용하기 위한 특정 요구 사항이 있습니다. 요구 사항이 충족되지 않으면 CDI에서 호스트 지원 복제를 사용합니다. 호스트 지원 복제는 가장 느리고 효율적인 복제 방법이지만 다른 두 복제 방법보다 요구 사항이 적습니다.

7.2.7.1.1. CSI 볼륨 복제

CSI(Container Storage Interface) 복제는 CSI 드라이버 기능을 사용하여 소스 데이터 볼륨을 보다 효율적으로 복제합니다.

CSI 볼륨 복제에는 다음과 같은 요구 사항이 있습니다.

- PVC(영구 볼륨 클레임)의 스토리지 클래스를 지원하는 CSI 드라이버는 볼륨 복제를 지원해야 합니다.
- CDI에서 인식하지 못하는 프로비저너의 경우 해당 스토리지 프로파일에 **cloneStrategy** 가 CSI Volume Cloning으로 설정되어 있어야 합니다.
- 소스 및 대상 PVC에는 동일한 스토리지 클래스와 볼륨 모드가 있어야 합니다.
- 데이터 볼륨을 생성하는 경우 소스 네임스페이스에 **datavolumes/source** 리소스를 생성할 수 있는 권한이 있어야 합니다.
- 소스 볼륨이 사용 중이 아니어야 합니다.

7.2.7.1.2. 스마트 복제

스냅샷 기능이 있는 CSI(Container Storage Interface) 플러그인을 사용할 수 있는 경우 CDI(Containerized Data Importer)는 스냅샷에서 PVC(영구 볼륨 클레임)를 생성하여 추가 PVC를 효율적으로 복제할 수 있습니다.

스마트 복제에는 다음과 같은 요구 사항이 있습니다.

- 스토리지 클래스와 연결된 스냅샷 클래스가 있어야 합니다.
- 소스 및 대상 PVC에는 동일한 스토리지 클래스와 볼륨 모드가 있어야 합니다.
- 데이터 볼륨을 생성하는 경우 소스 네임스페이스에 **datavolumes/source** 리소스를 생성할 수 있는 권한이 있어야 합니다.
- 소스 볼륨이 사용 중이 아니어야 합니다.

7.2.7.1.3. 호스트 지원 복제

CSI(Container Storage Interface) 볼륨 복제 또는 스마트 복제에 대한 요구 사항이 충족되지 않으면 호스트 지원 복제가 대체 방법으로 사용됩니다. 호스트 지원 복제는 다른 두 복제 방법 중 하나보다 효율적입니다.

호스트 지원 복제는 소스 Pod 및 대상 Pod를 사용하여 소스 볼륨의 데이터를 대상 볼륨으로 복사합니다. 대상 PVC(영구 볼륨 클레임)에는 호스트 지원 복제가 사용된 이유를 설명하고 이벤트가 생성되는 대체 이유가 포함되어 있습니다.

PVC 대상 주석의 예

```
apiVersion: v1
kind: PersistentVolumeClaim
```

```
metadata:
  annotations:
    cdi.kubevirt.io/cloneFallbackReason: The volume modes of source and target are
incompatible
    cdi.kubevirt.io/clonePhase: Succeeded
    cdi.kubevirt.io/cloneType: copy
```

이벤트 예

```
NAMESPACE   LAST SEEN   TYPE      REASON                    OBJECT
MESSAGE
test-ns     0s          Warning   IncompatibleVolumeModes   persistentvolumeclaim/test-target
The volume modes of source and target are incompatible
```

7.2.7.2. 웹 콘솔을 사용하여 PVC에서 VM 생성

OpenShift Container Platform 웹 콘솔을 사용하여 웹 페이지에서 이미지를 가져와 VM(가상 머신)을 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하여 PVC(영구 볼륨 클레임)를 복제하여 VM(가상 머신)을 생성할 수 있습니다.

사전 요구 사항

- 이미지가 포함된 웹 페이지에 액세스할 수 있어야 합니다.
- 소스 PVC가 포함된 네임스페이스에 액세스할 수 있어야 합니다.

프로세스

1. 웹 콘솔에서 가상화 → 카탈로그 로 이동합니다.
2. 사용 가능한 부팅 소스 없이 템플릿 타일을 클릭합니다.
3. VirtualMachine 사용자 지정을 클릭합니다.
4. 템플릿 매개변수 사용자 지정 페이지의 스토리지를 확장하고 디스크 소스 목록에서 PVC(복제 PVC) 를 선택합니다.
5. 이미지 URL을 입력합니다. 예: **https://access.redhat.com/downloads/content/69/ver=/rhel---7/7.9/x86_64/product-software**
6. 컨테이너 이미지 URL을 입력합니다. 예: **https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2**
7. PVC 프로젝트 및 PVC 이름을 선택합니다.
8. 디스크 크기를 설정합니다.
9. 다음을 클릭합니다.
10. VirtualMachine 생성을 클릭합니다.

7.2.7.3. 명령줄을 사용하여 PVC에서 VM 생성

명령줄을 사용하여 기존 VM의 PVC(영구 볼륨 클레임)를 복제하여 VM(가상 머신)을 생성할 수 있습니다.

다음 옵션 중 하나를 사용하여 PVC를 복제할 수 있습니다.

- 새 데이터 볼륨에 PVC 복제.
  이 방법은 라이프사이클이 원래 VM과 독립적인 데이터 볼륨을 생성합니다. 원래 VM을 삭제해도 새 데이터 볼륨 또는 연결된 PVC에 영향을 미치지 않습니다.

- **dataVolumeTemplates** 스탠자를 사용하여 **VirtualMachine** 매니페스트를 생성하여 PVC를 복제합니다.
  이 방법은 라이프사이클이 원래 VM에 종속된 데이터 볼륨을 생성합니다. 원래 VM을 삭제하면 복제된 데이터 볼륨 및 연결된 PVC가 삭제됩니다.

7.2.7.3.1. 데이터 볼륨에 PVC 복제

명령줄을 사용하여 기존 VM(가상 머신) 디스크의 PVC(영구 볼륨 클레임)를 데이터 볼륨에 복제할 수 있습니다.

원래 소스 PVC를 참조하는 데이터 볼륨을 생성합니다. 새 데이터 볼륨의 라이프사이클은 원래 VM과 독립적입니다. 원래 VM을 삭제해도 새 데이터 볼륨 또는 연결된 PVC에 영향을 미치지 않습니다.

소스 및 대상 PV가 **kubevirt** 콘텐츠 유형에 속하는 경우 PV(블록 영구 볼륨)에서 파일 시스템 PV로의 호스트 지원 복제와 같은 호스트 지원 복제가 지원됩니다.

참고

스마트 복제는 스냅샷을 사용하여 PVC를 복제하기 때문에 호스트 지원 복제보다 빠르고 효율적입니다. 스마트 복제는 Red Hat OpenShift Data Foundation과 같은 스냅샷을 지원하는 스토리지 공급자가 지원합니다.

다양한 볼륨 모드 간 복제는 스마트 복제에 지원되지 않습니다.

사전 요구 사항

- 소스 PVC가 있는 VM의 전원을 꺼야 합니다.
- PVC를 다른 네임스페이스에 복제하는 경우 대상 네임스페이스에 리소스를 생성할 수 있는 권한이 있어야 합니다.
- 스마트 복제를 위한 추가 사전 요구 사항:
  - 스토리지 공급자에서 스냅샷을 지원해야 합니다.
  - 소스 및 대상 PVC에는 동일한 스토리지 공급자 및 볼륨 모드가 있어야 합니다.
  - **VolumeSnapshotClass** 오브젝트의 드라이버 키 값은 다음 예와 같이**StorageClass** 오브젝트의 **provisioner** 키 값과 일치해야 합니다.

    VolumeSnapshotClass 오브젝트의 예

    ```
    kind: VolumeSnapshotClass
    apiVersion: snapshot.storage.k8s.io/v1
    driver: openshift-storage.rbd.csi.ceph.com
    # ...
    ```

    StorageClass 오브젝트의 예

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
# ...
provisioner: openshift-storage.rbd.csi.ceph.com
```

프로세스

1. 다음 예와 같이 **DataVolume** 매니페스트를 생성합니다.

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <datavolume> 1
spec:
  source:
    pvc:
      namespace: "<source_namespace>" 2
      name: "<my_vm_disk>" 3
  storage: {}
```

1. 새 데이터 볼륨의 이름을 지정합니다.
2. 소스 PVC의 네임스페이스를 지정합니다.
3. 소스 PVC의 이름을 지정합니다.

2. 다음 명령을 실행하여 데이터 볼륨을 생성합니다.

```
$ oc create -f <datavolume>.yaml
```

참고

데이터 볼륨은 PVC를 준비하기 전에 VM이 시작되지 않습니다. PVC가 복제되는 동안 새 데이터 볼륨을 참조하는 VM을 생성할 수 있습니다.

7.2.7.3.2. 데이터 볼륨 템플릿을 사용하여 복제된 PVC에서 VM 생성

데이터 볼륨 템플릿을 사용하여 기존 VM의 PVC(영구 볼륨 클레임)를 복제하는 VM(가상 머신)을 생성할 수 있습니다.

이 방법은 라이프사이클이 원래 VM에 종속된 데이터 볼륨을 생성합니다. 원래 VM을 삭제하면 복제된 데이터 볼륨 및 연결된 PVC가 삭제됩니다.

사전 요구 사항

- 소스 PVC가 있는 VM의 전원을 꺼야 합니다.

프로세스

1. 다음 예에 표시된 대로 **VirtualMachine** 매니페스트를 생성합니다.

```
apiVersion: kubevirt.io/v1
```

```
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-dv-clone
  name: vm-dv-clone (1)
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-dv-clone
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: root-disk
        resources:
          requests:
            memory: 64M
      volumes:
      - dataVolume:
          name: favorite-clone
        name: root-disk
  dataVolumeTemplates:
  - metadata:
      name: favorite-clone
    spec:
      storage:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 2Gi
      source:
        pvc:
          namespace: <source_namespace> (2)
          name: "<source_pvc>" (3)
```

(1) VM의 이름을 지정합니다.

(2) 소스 PVC의 네임스페이스를 지정합니다.

(3) 소스 PVC의 이름을 지정합니다.

2. PVC 복제 데이터 볼륨으로 가상 머신을 생성합니다.

```
$ oc create -f <vm-clone-datavolumetemplate>.yaml
```

7.3. 가상 머신 콘솔에 연결

다음 콘솔에 연결하여 실행 중인 VM(가상 머신)에 액세스할 수 있습니다.

- VNC 콘솔
- 직렬 콘솔
- Windows VM용 데스크탑 뷰어

7.3.1. VNC 콘솔에 연결

OpenShift Container Platform 웹 콘솔 또는 **virtctl** 명령줄 툴을 사용하여 가상 머신의 VNC 콘솔에 연결할 수 있습니다.

7.3.1.1. 웹 콘솔을 사용하여 VNC 콘솔에 연결

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)의 VNC 콘솔에 연결할 수 있습니다.

참고

중재된 장치로 할당된 vGPU를 사용하여 Windows VM에 연결하는 경우 기본 디스플레이와 vGPU 디스플레이 간에 전환할 수 있습니다.

프로세스

1. 가상화 → VirtualMachines 페이지에서 VM을 클릭하여 VirtualMachine 세부 정보 페이지를 엽니다.
2. 콘솔 탭을 클릭합니다. VNC 콘솔 세션이 자동으로 시작됩니다.
3. 선택 사항: Windows VM의 vGPU 디스플레이로 전환하려면 Send 키 목록에서 Ctl + Alt + 2를 선택합니다.
   - 기본 디스플레이를 복원하려면 Send 키 목록에서 Ctl + Alt + 1을 선택합니다.
4. 콘솔 세션을 종료하려면 콘솔 창 외부를 클릭한 다음 연결 끊기 를 클릭합니다.

7.3.1.2. virtctl을 사용하여 VNC 콘솔에 연결

virtctl 명령줄 툴을 사용하여 실행 중인 가상 머신의 VNC 콘솔에 연결할 수 있습니다.

참고

SSH 연결을 통해 원격 머신에서 **virtctl vnc** 명령을 실행하는 경우 **-X** 또는 **-Y** 플래그를 사용하여 **ssh** 명령을 실행하여 X 세션을 로컬 머신에 전달해야 합니다.

사전 요구 사항

- **virt-viewer** 패키지를 설치해야 합니다.

프로세스

1. 다음 명령을 실행하여 콘솔 세션을 시작합니다.

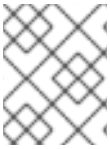
```
$ virtctl vnc <vm_name>
```

2. 연결에 실패하면 다음 명령을 실행하여 문제 해결 정보를 수집합니다.

```
$ virtctl vnc <vm_name> -v 4
```

7.3.1.3. VNC 콘솔의 임시 토큰 생성

VM(가상 머신)의 VNC에 액세스하려면 Kubernetes API에 대한 임시 인증 전달자 토큰을 생성합니다.



참고

Kubernetes에서는 curl 명령을 수정하여 전달자 토큰 대신 클라이언트 인증서를 사용한 인증도 지원합니다.

사전 요구 사항

- OpenShift Virtualization 4.14 이상 및 ssp-operator 4.14 이상이 있는 실행 중인 VM

프로세스

1. HyperConverged (HCO) 사용자 정의 리소스(CR)에서 기능 게이트를 활성화합니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv --type json -p
'[{"op": "replace", "path": "/spec/featureGates/deployVmConsoleProxy", "value":
true}]'
```

2. 다음 명령을 입력하여 토큰을 생성합니다.

```
$ curl --header "Authorization: Bearer ${TOKEN}" \
    "https://api.
<cluster_fqdn>/apis/token.kubevirt.io/v1alpha1/namespaces/<namespace>/virtualmac
hines/<vm_name>/vnc?duration=<duration>"
```

& lt;duration > 매개변수는 시간 및 분 단위로 설정할 수 있으며 최소 기간은 10분입니다. 예: 5h30m. 이 매개변수를 설정하지 않으면 토큰이 기본적으로 10분 동안 유효합니다.

샘플 출력:

```
{ "token": "eyJhb..." }
```

3. 선택 사항: 출력에 제공된 토큰을 사용하여 변수를 생성합니다.

```
$ export VNC_TOKEN="<token>"
```

이제 토큰을 사용하여 VM의 VNC 콘솔에 액세스할 수 있습니다.

검증

1. 다음 명령을 입력하여 클러스터에 로그인합니다.

```
$ oc login --token ${VNC_TOKEN}
```

2. virtctl 명령을 사용하여 VM의 VNC 콘솔에 대한 액세스를 테스트합니다.

```
$ virtctl vnc <vm_name> -n <namespace>
```

주의

현재 특정 토큰을 취소할 수 없습니다.

토큰을 취소하려면 해당 토큰을 생성하는 데 사용된 서비스 계정을 삭제해야 합니다. 그러나 서비스 계정을 사용하여 생성된 다른 모든 토큰도 취소합니다. 다음 명령을 주의해서 사용하십시오.

```
$ virtctl delete serviceaccount --namespace "<namespace>" "<vm_name>-vnc-access"
```

7.3.1.3.1. 클러스터 역할을 사용하여 VNC 콘솔에 대한 토큰 생성 권한 부여

클러스터 관리자는 클러스터 역할을 설치하고 사용자 또는 서비스 계정에 바인딩하여 VNC 콘솔의 토큰을 생성하는 엔드포인트에 액세스할 수 있습니다.

프로세스

- 클러스터 역할을 사용자 또는 서비스 계정에 바인딩하도록 선택합니다.
  - 다음 명령을 실행하여 클러스터 역할을 사용자에게 바인딩합니다.

```
$ kubectl create rolebinding "${ROLE_BINDING_NAME}" --
clusterrole="token.kubevirt.io:generate" --user="${USER_NAME}"
```

  - 다음 명령을 실행하여 클러스터 역할을 서비스 계정에 바인딩합니다.

```
$ kubectl create rolebinding "${ROLE_BINDING_NAME}" --
clusterrole="token.kubevirt.io:generate" --
serviceaccount="${SERVICE_ACCOUNT_NAME}"
```

7.3.2. 직렬 콘솔 연결

OpenShift Container Platform 웹 콘솔 또는 **virtctl** 명령줄 툴을 사용하여 가상 머신의 직렬 콘솔에 연결할 수 있습니다.

참고

단일 가상 머신에 대한 동시 VNC 연결 실행은 현재 지원되지 않습니다.

7.3.2.1. 웹 콘솔을 사용하여 직렬 콘솔에 연결

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)의 직렬 콘솔에 연결할 수 있습니다.

프로세스

1. 가상화 → VirtualMachines 페이지에서 VM을 클릭하여 VirtualMachine 세부 정보 페이지를 엽니다.

2. 콘솔 탭을 클릭합니다. VNC 콘솔 세션이 자동으로 시작됩니다.
3. 연결 끊기 를 클릭하여 VNC 콘솔 세션을 종료합니다. 그렇지 않으면 VNC 콘솔 세션이 백그라운드에서 계속 실행됩니다.
4. 콘솔 목록에서 직렬 콘솔 을 선택합니다.
5. 콘솔 세션을 종료하려면 콘솔 창 외부를 클릭한 다음 연결 끊기 를 클릭합니다.

7.3.2.2. virtctl을 사용하여 직렬 콘솔에 연결

virtctl 명령줄 툴을 사용하여 실행 중인 가상 머신의 직렬 콘솔에 연결할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 콘솔 세션을 시작합니다.

```
$ virtctl console <vm_name>
```

2. **Ctrl Cryostat**를 눌러 콘솔 세션을 종료합니다.

7.3.3. 데스크탑 뷰어에 연결

데스크탑 뷰어 및 RDP(Remote Desktop Protocol)를 사용하여 Windows VM(가상 머신)에 연결할 수 있습니다.

7.3.3.1. 웹 콘솔을 사용하여 데스크탑 뷰어에 연결

OpenShift Container Platform 웹 콘솔을 사용하여 Windows VM(가상 머신)의 데스크탑 뷰어에 연결할 수 있습니다.

사전 요구 사항

- Windows VM에 QEMU 게스트 에이전트가 설치되어 있어야 합니다.
- RDP 클라이언트가 설치되어 있어야 합니다.

프로세스

1. 가상화 → VirtualMachines 페이지에서 VM을 클릭하여VirtualMachine 세부 정보 페이지를 엽니다.
2. 콘솔 탭을 클릭합니다. VNC 콘솔 세션이 자동으로 시작됩니다.
3. 연결 끊기 를 클릭하여 VNC 콘솔 세션을 종료합니다. 그렇지 않으면 VNC 콘솔 세션이 백그라운드에서 계속 실행됩니다.
4. 콘솔 목록에서 데스크탑 뷰어를 선택합니다.
5. RDP 서비스 생성 을 클릭하여 RDP 서비스 대화 상자를 엽니다.
6. Expose RDP Service 를 선택하고 저장을 클릭하여 노드 포트 서비스를 생성합니다.
7. 원격 데스크탑 시작을 클릭하여 **.rdp** 파일을 다운로드하고 데스크탑 뷰어를 시작합니다.

7.4. 가상 머신에 대한 SSH 액세스 구성

다음 방법을 사용하여 VM(가상 머신)에 대한 SSH 액세스를 구성할 수 있습니다.

- **virtctl ssh 명령**
  SSH 키 쌍을 생성하고 VM에 공개 키를 추가하고 개인 키로 **virtctl ssh** 명령을 실행하여 VM에 연결합니다.

  런타임 시 RHEL(Red Hat Enterprise Linux) 9 VM에 공개 SSH 키를 추가하거나 먼저 cloud-init 데이터 소스를 사용하여 구성할 수 있는 게스트 운영 체제를 사용하여 VM에 부팅할 수 있습니다.

- **virtctl port-forward 명령**
  virtctl port-foward 명령을 **.ssh/config** 파일에 추가하고 OpenSSH를 사용하여 VM에 연결합니다.

- **Service**
  서비스를 생성하고 서비스를 VM과 연결하고 서비스에서 노출하는 IP 주소 및 포트에 연결합니다.

- **보조 네트워크**
  보조 네트워크를 구성하고 VM(가상 머신)을 보조 네트워크 인터페이스에 연결한 다음 DHCP 할당된 IP 주소에 연결합니다.

7.4.1. 액세스 구성 고려 사항

VM(가상 머신)에 대한 액세스를 구성하는 각 방법에는 트래픽 로드 및 클라이언트 요구 사항에 따라 장단점이 있습니다.

서비스는 우수한 성능을 제공하며 클러스터 외부에서 액세스하는 애플리케이션에 권장됩니다.

내부 클러스터 네트워크가 트래픽 로드를 처리할 수 없는 경우 보조 네트워크를 구성할 수 있습니다.

virtctl ssh 및 virtctl port-forwarding 명령

- 간편한 구성.
- VM 문제 해결에 권장됩니다.
- **virtctl port-forwarding** 은 Ansible을 사용한 VM의 자동 구성에 권장됩니다.
- 동적 공용 SSH 키를 사용하여 Ansible을 사용하여 VM을 프로비저닝할 수 있습니다.
- API 서버의 부하 때문에 Rsync 또는 Remote Desktop Protocol과 같은 트래픽이 많은 애플리케이션에는 권장되지 않습니다.
- API 서버는 트래픽 로드를 처리할 수 있어야 합니다.
- 클라이언트는 API 서버에 액세스할 수 있어야 합니다.
- 클라이언트에는 클러스터에 대한 액세스 자격 증명이 있어야 합니다.

클러스터 IP 서비스

- 내부 클러스터 네트워크는 트래픽 부하를 처리할 수 있어야 합니다.
- 클라이언트는 내부 클러스터 IP 주소에 액세스할 수 있어야 합니다.

노드 포트 서비스

- 내부 클러스터 네트워크는 트래픽 부하를 처리할 수 있어야 합니다.
- 클라이언트는 하나 이상의 노드에 액세스할 수 있어야 합니다.

로드 밸런서 서비스

- 로드 밸런서를 구성해야 합니다.
- 각 노드는 하나 이상의 로드 밸런서 서비스의 트래픽 로드를 처리할 수 있어야 합니다.

보조 네트워크

- 트래픽이 내부 클러스터 네트워크를 통과하지 않기 때문에 우수한 성능.
- 네트워크 토폴로지에 대한 유연한 접근 방식을 허용합니다.
- VM이 보조 네트워크에 직접 노출되므로 게스트 운영 체제를 적절한 보안으로 구성해야 합니다. VM이 손상되면 침입자가 보조 네트워크에 액세스할 수 있습니다.

7.4.2. virtctl ssh 사용

공용 SSH 키를 VM(가상 머신)에 추가하고 **virtctl ssh** 명령을 실행하여 VM에 연결할 수 있습니다.

이 방법은 쉽게 구성할 수 있습니다. 그러나 API 서버에 부담을 주기 때문에 트래픽이 많은 로드에는 권장되지 않습니다.

7.4.2.1. 정적 및 동적 SSH 키 관리 정보

처음 부팅 시 또는 런타임 시 동적으로 공용 SSH 키를 VM(가상 머신)에 정적으로 추가할 수 있습니다.

참고

RHEL(Red Hat Enterprise Linux) 9만 동적 키 삽입을 지원합니다.

정적 SSH 키 관리
cloud-init 데이터 소스를 사용하여 구성을 지원하는 게스트 운영 체제를 사용하여 VM에 정적으로 관리되는 SSH 키를 추가할 수 있습니다. 키는 처음 부팅할 때 VM(가상 머신)에 추가됩니다.

다음 방법 중 하나를 사용하여 키를 추가할 수 있습니다.

- 웹 콘솔 또는 명령줄을 사용하여 단일 VM에 키를 추가합니다.
- 웹 콘솔을 사용하여 프로젝트에 키를 추가합니다. 이후 이 프로젝트에서 생성한 VM에 키가 자동으로 추가됩니다.

사용 사례

- VM 소유자는 새로 생성된 모든 VM을 단일 키로 프로비저닝할 수 있습니다.

동적 SSH 키 관리

RHEL(Red Hat Enterprise Linux) 9가 설치된 VM에 대해 동적 SSH 키 관리를 활성화할 수 있습니다. 나중에 런타임 중에 키를 업데이트할 수 있습니다. 키는 Red Hat 부팅 소스와 함께 설치된 QEMU 게스트 에이전트에 의해 추가됩니다.

보안상의 이유로 동적 키 관리를 비활성화할 수 있습니다. 그런 다음 VM은 생성된 이미지의 키 관리 설정을 상속합니다.

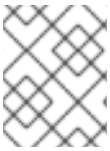
사용 사례

- VM에 대한 액세스 권한 부여 또는 취소: 클러스터 관리자는 네임스페이스의 모든 VM에 적용되는 **Secret** 오브젝트에서 개별 사용자의 키를 추가하거나 제거하여 원격 VM 액세스 권한을 부여하거나 취소할 수 있습니다.
- 사용자 액세스: 생성 및 관리하는 모든 VM에 액세스 자격 증명을 추가할 수 있습니다.
- Ansible 프로비저닝:
  - 작업 팀 구성원으로 Ansible 프로비저닝에 사용되는 모든 키가 포함된 단일 시크릿을 생성할 수 있습니다.
  - VM 소유자는 VM을 생성하고 Ansible 프로비저닝에 사용되는 키를 연결할 수 있습니다.
- 키 교체:
  - 클러스터 관리자는 네임스페이스의 VM에서 사용하는 Ansible 프로비저너 키를 순환할 수 있습니다.
  - 워크로드 소유자는 관리하는 VM의 키를 회전할 수 있습니다.

7.4.2.2. 정적 키 관리

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 VM(가상 머신)을 생성할 때 정적으로 관리되는 공용 SSH 키를 추가할 수 있습니다. VM이 처음 부팅되면 키가 cloud-init 데이터 소스로 추가됩니다.

웹 콘솔을 사용하여 VM을 생성할 때 프로젝트에 공개 SSH 키를 추가할 수도 있습니다. 키는 시크릿으로 저장되며 사용자가 생성한 모든 VM에 자동으로 추가됩니다.



참고

프로젝트에 보안을 추가한 다음 VM을 삭제하면 네임스페이스 리소스이므로 보안이 유지됩니다. 시크릿을 수동으로 삭제해야 합니다.

7.4.2.2.1. 템플릿에서 VM을 생성할 때 키 추가

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)을 생성할 때 정적으로 관리되는 공개 SSH 키를 추가할 수 있습니다. 키는 처음 부팅할 때 VM에 cloud-init 데이터 소스로 추가됩니다. 이 방법은 cloud-init 사용자 데이터에 영향을 미치지 않습니다.

선택 사항: 프로젝트에 키를 추가할 수 있습니다. 나중에 이 키는 프로젝트에서 생성하는 VM에 자동으로 추가됩니다.

사전 요구 사항

- **ssh-keygen** 명령을 실행하여 SSH 키 쌍을 생성했습니다.

프로세스

1. 웹 콘솔에서 가상화 → 카탈로그 로 이동합니다.
2. 템플릿 타일을 클릭합니다.
   게스트 운영 체제는 cloud-init 데이터 소스의 구성을 지원해야 합니다.
3. VirtualMachine 사용자 지정을 클릭합니다.
4. 다음을 클릭합니다.
5. 스크립트 탭을 클릭합니다.
6. 프로젝트에 공개 SSH 키를 아직 추가하지 않은 경우 인증된 SSH 키 옆에 있는 편집 아이콘을 클릭하고 다음 옵션 중 하나를 선택합니다.
   - Use existing: 시크릿 목록에서 시크릿을 선택합니다.
   - 새 추가:
     a. SSH 키 파일을 찾아보거나 키 필드에 파일을 붙여넣습니다.
     b. 시크릿 이름을 입력합니다.
     c. 선택 사항: 이 프로젝트에서 생성한 새 VirtualMachine에 자동으로 이 키 적용을선택합니다.
7. 저장을 클릭합니다.
8. VirtualMachine 생성을 클릭합니다.
   VirtualMachine 세부 정보 페이지에는 VM 생성 진행률이 표시됩니다.

검증

- 구성 탭에서 스크립트 탭을 클릭합니다.
  시크릿 이름은 인증된 SSH 키 섹션에 표시됩니다.

7.4.2.2.2. 웹 콘솔을 사용하여 인스턴스 유형에서 VM을 생성할 때 키 추가

OpenShift Container Platform 웹 콘솔을 사용하여 인스턴스 유형에서 VM(가상 머신)을 생성할 수 있습니다. 기존 스냅샷을 복사하거나 VM을 복제하여 웹 콘솔을 사용하여 VM을 생성할 수도 있습니다.

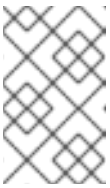
사용 가능한 부팅 가능한 볼륨 목록에서 VM을 생성할 수 있습니다. 목록에 Linux 또는 Windows 기반 볼륨을 추가할 수 있습니다.

OpenShift Container Platform 웹 콘솔을 사용하여 인스턴스 유형에서 VM(가상 머신)을 생성할 때 정적으로 관리되는 SSH 키를 추가할 수 있습니다. 키는 처음 부팅할 때 VM에 cloud-init 데이터 소스로 추가됩니다. 이 방법은 cloud-init 사용자 데이터에 영향을 미치지 않습니다.

프로세스

1. 웹 콘솔에서 가상화 → 카탈로그 로 이동합니다.
   InstanceTypes 탭이 기본적으로 열립니다.
2. 다음 옵션 중 하나를 선택합니다.

- 목록에서 적절한 부팅 가능한 볼륨을 선택합니다. 목록이 잘린 경우 모두 표시 버튼을 클릭하여 전체 목록을 표시합니다.



참고

부팅 가능한 볼륨 테이블에는 **instancetype.kubevirt.io/default-preference** 레이블이 있는 **openshift-virtualization-os-images** 네임스페이스의 볼륨만 나열됩니다.

  - 선택 사항: 별 아이콘을 클릭하여 부팅 가능한 볼륨을 즐겨 찾기로 지정합니다. 지정된 부팅 가능한 볼륨이 볼륨 목록에 먼저 표시됩니다.

- 볼륨 추가 를 클릭하여 새 볼륨을 업로드하거나 기존 PVC(영구 볼륨 클레임), 볼륨 스냅샷 또는 **containerDisk** 볼륨을 사용합니다. 저장을 클릭합니다.
  클러스터에서 사용할 수 없는 운영 체제의 로고는 목록 하단에 표시됩니다. 볼륨 추가 링크를 클릭하여 필요한 운영 체제에 볼륨을 추가할 수 있습니다.

  또한 Windows 부팅 소스 생성 빠른 시작에 대한링크가 있습니다. 동일한 링크가 *줄에서 부팅할 볼륨 선택* 옆에 있는 물음표 아이콘 위에 포인터를 가져가면 팝업 창에 나타납니다.

  환경을 설치한 직후 또는 환경의 연결이 끊어지면 부팅할 볼륨 목록이 비어 있습니다. 이 경우 Windows, RHEL 및 Linux의 세 가지 운영 체제 로고가 표시됩니다. 볼륨 추가 버튼을 클릭하여 요구 사항을 충족하는 새 볼륨을 추가할 수 있습니다.

3. 인스턴스 유형 타일을 클릭하고 워크로드에 적합한 리소스 크기를 선택합니다.

4. 선택 사항: 부팅 중인 볼륨에 적용되는 VM 이름을 포함하여 가상 머신 세부 정보를 선택합니다.

   - Linux 기반 볼륨의 경우 다음 단계에 따라 SSH를 구성합니다.

     a. 프로젝트에 공개 SSH 키를 아직 추가하지 않은 경우 VirtualMachine details 섹션에서 인증된 SSH 키 옆에 있는 편집 아이콘을 클릭합니다.

     b. 다음 옵션 중 하나를 선택합니다.

        - Use existing: 시크릿 목록에서 시크릿을 선택합니다.
        - 새 추가: 다음 단계를 따르십시오.

          i. 공개 SSH 키 파일을 찾아보거나 키 필드에 파일을 붙여넣습니다.

          ii. 시크릿 이름을 입력합니다.

          iii. 선택 사항: 이 프로젝트에서 생성한 새 VirtualMachine에 자동으로 이 키 적용을선택합니다.

     c. 저장을 클릭합니다.

   - Windows 볼륨의 경우 다음 단계 세트 중 하나를 수행하여 sysprep 옵션을 구성합니다.

     - Windows 볼륨에 sysprep 옵션을 아직 추가하지 않은 경우 다음 단계를 따르십시오.

       i. VirtualMachine details 섹션에서 edit 아이콘 besideSysprep 을 클릭합니다.

       ii. Autoattend.xml 응답 파일을 추가합니다.

       iii. Unattend.xml 응답 파일을 추가합니다.

iv. 저장을 클릭합니다.

- Windows 볼륨에 기존 sysprep 옵션을 사용하려면 다음 단계를 따르십시오.

i. Attach existing sysprep 을 클릭합니다.

ii. 기존 sysprep Unattend.xml 응답 파일의 이름을 입력합니다.

iii. 저장을 클릭합니다.

5. 선택 사항: Windows VM을 생성하는 경우 Windows 드라이버 디스크를 마운트할 수 있습니다.

a. Customize VirtualMachine 버튼을 클릭합니다.

b. VirtualMachine 세부 정보 페이지에서 스토리지를 클릭합니다.

c. Mount Windows drivers disk 확인란을 선택합니다.

6. 선택 사항: YAML 및 CLI 보기를 클릭하여 YAML 파일을 확인합니다.CLI 를 클릭하여 CLI 명령을 확인합니다. YAML 파일 콘텐츠 또는 CLI 명령을 다운로드하거나 복사할 수도 있습니다.

7. VirtualMachine 생성을 클릭합니다.

VM이 생성되면 VirtualMachine 세부 정보 페이지에서 상태를 모니터링할 수 있습니다.

7.4.2.2.3. 명령줄을 사용하여 VM을 생성할 때 키 추가

명령줄을 사용하여 VM(가상 머신)을 생성할 때 정적으로 관리되는 공용 SSH 키를 추가할 수 있습니다. 키는 처음 부팅할 때 VM에 추가됩니다.

키가 VM에 cloud-init 데이터 소스로 추가됩니다. 이 방법은 cloud-init 사용자 데이터의 애플리케이션 데이터와 액세스 자격 증명을 구분합니다. 이 방법은 cloud-init 사용자 데이터에 영향을 미치지 않습니다.

사전 요구 사항

- **ssh-keygen** 명령을 실행하여 SSH 키 쌍을 생성했습니다.

프로세스

1. **VirtualMachine** 오브젝트 및 **Secret** 오브젝트에 대한 매니페스트 파일을 생성합니다.

매니페스트 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  dataVolumeTemplates:
    - metadata:
        name: example-vm-volume
      spec:
        sourceRef:
          kind: DataSource
          name: rhel9
```

```
        namespace: openshift-virtualization-os-images
      storage:
        resources: {}
  instancetype:
    name: u1.medium
  preference:
    name: rhel.9
  running: true
  template:
    spec:
      domain:
        devices: {}
      volumes:
        - dataVolume:
            name: example-vm-volume
          name: rootdisk
        - cloudInitNoCloud: 1
            userData: |-
              #cloud-config
              user: cloud-user
          name: cloudinitdisk
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              noCloud: {}
            source:
              secret:
                secretName: authorized-keys 2
---
apiVersion: v1
kind: Secret
metadata:
  name: authorized-keys
data:
  key: c3NoLXJzYSB... 3
```

1 cloudInitNoCloud 데이터 소스를 지정합니다.

2 Secret 오브젝트 이름을 지정합니다.

3 공개 SSH 키를 붙여넣습니다.

2. 다음 명령을 실행하여 VirtualMachine 및 Secret 오브젝트를 생성합니다.

```
$ oc create -f <manifest_file>.yaml
```

3. 다음 명령을 실행하여 VM을 시작합니다.

```
$ virtctl start vm example-vm -n example-namespace
```

검증

- VM 구성을 가져옵니다.

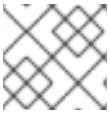
```
$ oc describe vm example-vm -n example-namespace
```

출력 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  template:
    spec:
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              noCloud: {}
            source:
              secret:
                secretName: authorized-keys
# ...
```

7.4.2.3. 동적 키 관리

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 VM(가상 머신)에 동적 키 삽입을 활성화할 수 있습니다. 그런 다음 런타임 시 키를 업데이트할 수 있습니다.



참고

RHEL(Red Hat Enterprise Linux) 9만 동적 키 삽입을 지원합니다.

동적 키 삽입을 비활성화하면 VM은 생성된 이미지의 키 관리 방법을 상속합니다.

7.4.2.3.1. 템플릿에서 VM을 생성할 때 동적 키 삽입 활성화

OpenShift Container Platform 웹 콘솔을 사용하여 템플릿에서 VM(가상 머신)을 생성할 때 동적 공개 SSH 키 삽입을 활성화할 수 있습니다. 그런 다음 런타임 시 키를 업데이트할 수 있습니다.

참고

RHEL(Red Hat Enterprise Linux) 9만 동적 키 삽입을 지원합니다.

키는 RHEL 9와 함께 설치된 QEMU 게스트 에이전트에 의해 VM에 추가됩니다.

사전 요구 사항

- **ssh-keygen** 명령을 실행하여 SSH 키 쌍을 생성했습니다.

프로세스

1. 웹 콘솔에서 가상화 → 카탈로그 로 이동합니다.
2. Red Hat Enterprise Linux 9 VM타일을 클릭합니다.

3. VirtualMachine 사용자 지정을 클릭합니다.
4. 다음을 클릭합니다.
5. 스크립트 탭을 클릭합니다.
6. 프로젝트에 공개 SSH 키를 아직 추가하지 않은 경우 인증된 SSH 키 옆에 있는 편집 아이콘을 클릭하고 다음 옵션 중 하나를 선택합니다.
   - Use existing: 시크릿 목록에서 시크릿을 선택합니다.
   - 새 추가:
     a. SSH 키 파일을 찾아보거나 키 필드에 파일을 붙여넣습니다.
     b. 시크릿 이름을 입력합니다.
     c. 선택 사항: 이 프로젝트에서 생성한 새 VirtualMachine에 자동으로 이 키 적용을선택합니다.
7. 동적 SSH 키 삽입을의 로 설정합니다.
8. 저장을 클릭합니다.
9. VirtualMachine 생성을 클릭합니다.
   VirtualMachine 세부 정보 페이지에는 VM 생성 진행률이 표시됩니다.

검증

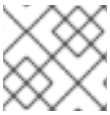
- 구성 탭에서 스크립트 탭을 클릭합니다.
  시크릿 이름은 인증된 SSH 키 섹션에 표시됩니다.

7.4.2.3.2. 웹 콘솔을 사용하여 인스턴스 유형에서 VM을 생성할 때 동적 키 삽입 활성화

OpenShift Container Platform 웹 콘솔을 사용하여 인스턴스 유형에서 VM(가상 머신)을 생성할 수 있습니다. 기존 스냅샷을 복사하거나 VM을 복제하여 웹 콘솔을 사용하여 VM을 생성할 수도 있습니다.

사용 가능한 부팅 가능한 볼륨 목록에서 VM을 생성할 수 있습니다. 목록에 Linux 또는 Windows 기반 볼륨을 추가할 수 있습니다.

OpenShift Container Platform 웹 콘솔을 사용하여 인스턴스 유형에서 VM(가상 머신)을 생성할 때 동적 SSH 키 삽입을 활성화할 수 있습니다. 그런 다음 런타임에 키를 추가하거나 취소할 수 있습니다.



참고

RHEL(Red Hat Enterprise Linux) 9만 동적 키 삽입을 지원합니다.

키는 RHEL 9와 함께 설치된 QEMU 게스트 에이전트에 의해 VM에 추가됩니다.

프로세스

1. 웹 콘솔에서 가상화 → 카탈로그 로 이동합니다.
   InstanceTypes 탭이 기본적으로 열립니다.
2. 다음 옵션 중 하나를 선택합니다.

- 목록에서 적절한 부팅 가능한 볼륨을 선택합니다. 목록이 잘린 경우 모두 표시 버튼을 클릭하여 전체 목록을 표시합니다.

참고

부팅 가능한 볼륨 테이블에는 **instancetype.kubevirt.io/default-preference** 레이블이 있는 **openshift-virtualization-os-images** 네임스페이스의 볼륨만 나열됩니다.

  - 선택 사항: 별 아이콘을 클릭하여 부팅 가능한 볼륨을 즐겨 찾기로 지정합니다. 지정된 부팅 가능한 볼륨이 볼륨 목록에 먼저 표시됩니다.

- 볼륨 추가 를 클릭하여 새 볼륨을 업로드하거나 기존 PVC(영구 볼륨 클레임), 볼륨 스냅샷 또는 **containerDisk** 볼륨을 사용합니다. 저장을 클릭합니다.
클러스터에서 사용할 수 없는 운영 체제의 로고는 목록 하단에 표시됩니다. 볼륨 추가 링크를 클릭하여 필요한 운영 체제에 볼륨을 추가할 수 있습니다.

  또한 Windows 부팅 소스 생성 빠른 시작에 대한링크가 있습니다. 동일한 링크가 *줄에서 부팅할 볼륨 선택* 옆에 있는 물음표 아이콘 위에 포인터를 가져가면 팝업 창에 나타납니다.

  환경을 설치한 직후 또는 환경의 연결이 끊어지면 부팅할 볼륨 목록이 비어 있습니다. 이 경우 Windows, RHEL 및 Linux의 세 가지 운영 체제 로고가 표시됩니다. 볼륨 추가 버튼을 클릭하여 요구 사항을 충족하는 새 볼륨을 추가할 수 있습니다.

3. 인스턴스 유형 타일을 클릭하고 워크로드에 적합한 리소스 크기를 선택합니다.
4. Red Hat Enterprise Linux 9 VM타일을 클릭합니다.
5. 선택 사항: 부팅 중인 볼륨에 적용되는 VM 이름을 포함하여 가상 머신 세부 정보를 선택합니다.
   - Linux 기반 볼륨의 경우 다음 단계에 따라 SSH를 구성합니다.
     a. 프로젝트에 공개 SSH 키를 아직 추가하지 않은 경우 VirtualMachine details 섹션에서 인증된 SSH 키 옆에 있는 편집 아이콘을 클릭합니다.
     b. 다음 옵션 중 하나를 선택합니다.
        - Use existing: 시크릿 목록에서 시크릿을 선택합니다.
        - 새 추가: 다음 단계를 따르십시오.
          i. 공개 SSH 키 파일을 찾아보거나 키 필드에 파일을 붙여넣습니다.
          ii. 시크릿 이름을 입력합니다.
          iii. 선택 사항: 이 프로젝트에서 생성한 새 VirtualMachine에 자동으로 이 키 적용을선택합니다.
     c. 저장을 클릭합니다.
   - Windows 볼륨의 경우 다음 단계 세트 중 하나를 수행하여 sysprep 옵션을 구성합니다.
     - Windows 볼륨에 sysprep 옵션을 아직 추가하지 않은 경우 다음 단계를 따르십시오.
       i. VirtualMachine details 섹션에서 edit 아이콘 besideSysprep 을 클릭합니다.
       ii. Autoattend.xml 응답 파일을 추가합니다.

iii. Unattend.xml 응답 파일을 추가합니다.

iv. 저장을 클릭합니다.

- Windows 볼륨에 기존 sysprep 옵션을 사용하려면 다음 단계를 따르십시오.

i. Attach existing sysprep을 클릭합니다.

ii. 기존 sysprep Unattend.xml 응답 파일의 이름을 입력합니다.

iii. 저장을 클릭합니다.

6. VirtualMachine details 섹션에서 동적 SSH 키 삽입을 의 로 설정합니다.
7. 선택 사항: Windows VM을 생성하는 경우 Windows 드라이버 디스크를 마운트할 수 있습니다.
   a. Customize VirtualMachine 버튼을 클릭합니다.
   b. VirtualMachine 세부 정보 페이지에서 스토리지를 클릭합니다.
   c. Mount Windows drivers disk 확인란을 선택합니다.
8. 선택 사항: YAML 및 CLI 보기를 클릭하여 YAML 파일을 확인합니다.CLI 를 클릭하여 CLI 명령을 확인합니다. YAML 파일 콘텐츠 또는 CLI 명령을 다운로드하거나 복사할 수도 있습니다.
9. VirtualMachine 생성을 클릭합니다.

VM이 생성되면 VirtualMachine 세부 정보 페이지에서 상태를 모니터링할 수 있습니다.

7.4.2.3.3. 웹 콘솔을 사용하여 동적 SSH 키 삽입 활성화

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)에 동적 키 삽입을 활성화할 수 있습니다. 그런 다음 런타임 시 공용 SSH 키를 업데이트할 수 있습니다.

키는 RHEL(Red Hat Enterprise Linux) 9와 함께 설치된 QEMU 게스트 에이전트에 의해 VM에 추가됩니다.

사전 요구 사항

- 게스트 운영 체제는 RHEL 9입니다.

프로세스

1. 웹 콘솔에서 가상화 → VirtualMachines 로 이동합니다.
2. VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 구성 탭에서 스크립트를 클릭합니다.
4. 프로젝트에 공개 SSH 키를 아직 추가하지 않은 경우 인증된 SSH 키 옆에 있는 편집 아이콘을 클릭하고 다음 옵션 중 하나를 선택합니다.
   - Use existing: 시크릿 목록에서 시크릿을 선택합니다.
   - 새 추가:
     a. SSH 키 파일을 찾아보거나 키 필드에 파일을 붙여넣습니다.

b. 시크릿 이름을 입력합니다.

c. 선택 사항: 이 프로젝트에서 생성한 새 VirtualMachine에 자동으로 이 키 적용을선택합니다.

5. 동적 SSH 키 삽입을의 로 설정합니다.

6. 저장을 클릭합니다.

7.4.2.3.4. 명령줄을 사용하여 동적 키 삽입 활성화

명령줄을 사용하여 VM(가상 머신)에 동적 키 삽입을 활성화할 수 있습니다. 그런 다음 런타임 시 공용 SSH 키를 업데이트할 수 있습니다.

참고

RHEL(Red Hat Enterprise Linux) 9만 동적 키 삽입을 지원합니다.

키는 RHEL 9와 함께 자동으로 설치되는 QEMU 게스트 에이전트에 의해 VM에 추가됩니다.

사전 요구 사항

- **ssh-keygen** 명령을 실행하여 SSH 키 쌍을 생성했습니다.

프로세스

1. **VirtualMachine** 오브젝트 및 **Secret** 오브젝트에 대한 매니페스트 파일을 생성합니다.

매니페스트 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  dataVolumeTemplates:
    - metadata:
        name: example-vm-volume
      spec:
        sourceRef:
          kind: DataSource
          name: rhel9
          namespace: openshift-virtualization-os-images
        storage:
          resources: {}
  instancetype:
    name: u1.medium
  preference:
    name: rhel.9
  running: true
  template:
    spec:
      domain:
        devices: {}
```

```
      volumes:
        - dataVolume:
            name: example-vm-volume
          name: rootdisk
        - cloudInitNoCloud: 1
            userData: |-
              #cloud-config
              runcmd:
              - [ setsebool, -P, virt_qemu_ga_manage_ssh, on ]
          name: cloudinitdisk
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              qemuGuestAgent:
                users: ["cloud-user"]
            source:
              secret:
                secretName: authorized-keys 2
---
apiVersion: v1
kind: Secret
metadata:
  name: authorized-keys
data:
  key: c3NoLXJzYSB... 3
```

1 cloudInitNoCloud 데이터 소스를 지정합니다.

2 Secret 오브젝트 이름을 지정합니다.

3 공개 SSH 키를 붙여넣습니다.

2. 다음 명령을 실행하여 VirtualMachine 및 Secret 오브젝트를 생성합니다.

```
$ oc create -f <manifest_file>.yaml
```

3. 다음 명령을 실행하여 VM을 시작합니다.

```
$ virtctl start vm example-vm -n example-namespace
```

검증

- VM 구성을 가져옵니다.

```
$ oc describe vm example-vm -n example-namespace
```

출력 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
```

```
spec:
  template:
    spec:
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              qemuGuestAgent:
                users: ["cloud-user"]
            source:
              secret:
                secretName: authorized-keys
# ...
```

7.4.2.4. virtctl ssh 명령 사용

virtcl ssh 명령을 사용하여 실행 중인 VM(가상 머신)에 액세스할 수 있습니다.

사전 요구 사항

- **virtctl** 명령줄 툴을 설치했습니다.
- VM에 공개 SSH 키를 추가했습니다.
- SSH 클라이언트가 설치되어 있어야 합니다.
- **virtctl** 툴을 설치한 환경에는 VM에 액세스하는 데 필요한 클러스터 권한이 있습니다. 예를 들어 **oc login** 을 실행하거나 **KUBECONFIG** 환경 변수를 설정합니다.

프로세스

- **virtctl ssh** 명령을 실행합니다.

```
$ virtctl -n <namespace> ssh <username>@example-vm -i <ssh_key> 1
```

1 네임스페이스, 사용자 이름 및 SSH 개인 키를 지정합니다. 기본 SSH 키 위치는 **/home/user/.ssh** 입니다. 키를 다른 위치에 저장하는 경우 경로를 지정해야 합니다.

예제

```
$ virtctl -n my-namespace ssh cloud-user@example-vm -i my-key
```

작은 정보

VirtualMachines 페이지의 VM 옆에 있는 옵션 메뉴에서 SSH 명령 복사를 선택하여 웹 콘솔에서 **virtctl ssh** 명령을 복사할 수 있습니다.

7.4.3. virtctl port-forward 명령 사용

로컬 OpenSSH 클라이언트와 **virtctl port-forward** 명령을 사용하여 실행 중인 VM(가상 머신)에 연결할 수 있습니다. 이 방법을 Ansible과 함께 사용하여 VM 구성을 자동화할 수 있습니다.

이 방법은 컨트롤 플레인을 통해 포트 전달 트래픽이 전송되므로 트래픽이 적은 애플리케이션에 권장됩니다. 이 방법은 API 서버에 많은 부담을 주기 때문에 Rsync 또는 원격 데스크탑 프로토콜과 같은 트래픽이 많은 애플리케이션에 권장되지 않습니다.

사전 요구 사항

- **virtctl** 클라이언트를 설치했습니다.
- 액세스하려는 가상 머신이 실행 중입니다.
- **virtctl** 툴을 설치한 환경에는 VM에 액세스하는 데 필요한 클러스터 권한이 있습니다. 예를 들어 **oc login** 을 실행하거나 **KUBECONFIG** 환경 변수를 설정합니다.

프로세스

1. 클라이언트 머신의 **~/.ssh/config** 파일에 다음 텍스트를 추가합니다.

```
Host vm/*
  ProxyCommand virtctl port-forward --stdio=true %h %p
```

2. 다음 명령을 실행하여 VM에 연결합니다.

```
$ ssh <user>@vm/<vm_name>.<namespace>
```

7.4.4. SSH 액세스에 서비스 사용

VM(가상 머신)에 대한 서비스를 생성하고 서비스에서 노출하는 IP 주소 및 포트에 연결할 수 있습니다.

서비스는 우수한 성능을 제공하며 클러스터 외부에서 또는 클러스터 내에서 액세스하는 애플리케이션에 권장됩니다. Ingress 트래픽은 방화벽에 의해 보호됩니다.

클러스터 네트워크가 트래픽 로드를 처리할 수 없는 경우 VM 액세스를 위해 보조 네트워크를 사용하는 것이 좋습니다.

7.4.4.1. 서비스 정보

Kubernetes 서비스는 클라이언트의 네트워크 액세스를 포드 세트에서 실행되는 애플리케이션에 노출합니다. 서비스는 추상화, 로드 밸런싱 및 **NodePort** 및 **LoadBalancer** 유형의 경우 외부 세계에 노출을 제공합니다.

ClusterIP

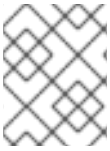
내부 IP 주소에 서비스를 노출하고 클러스터 내의 다른 애플리케이션에 DNS 이름으로 노출합니다. 단일 서비스는 여러 가상 머신에 매핑할 수 있습니다. 클라이언트가 서비스에 연결하려고 하면 사용 가능한 백엔드 간에 클라이언트 요청이 부하 분산됩니다. **ClusterIP** 는 기본 서비스 유형입니다.

NodePort

클러스터에서 선택한 각 노드의 동일한 포트에 서비스를 노출합니다. **NodePort** 를 사용하면 노드 자체에 클라이언트에서 외부에서 액세스할 수 있는 한 클러스터 외부에서 포트에 액세스할 수 있습니다.

LoadBalancer

현재 클라우드에 외부 로드 밸런서를 생성하고(지원되는 경우) 고정 외부 IP 주소를 서비스에 할당합니다.



참고

온프레미스 클러스터의 경우 MetalLB Operator를 배포하여 로드 밸런싱 서비스를 구성할 수 있습니다.

7.4.4.2. 서비스 생성

OpenShift Container Platform 웹 콘솔, **virtctl** 명령줄 툴 또는 YAML 파일을 사용하여 VM(가상 머신)을 노출하는 서비스를 생성할 수 있습니다.

7.4.4.2.1. 웹 콘솔을 사용하여 로드 밸런서 서비스 생성 활성화

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)에 대한 로드 밸런서 서비스 생성을 활성화할 수 있습니다.

사전 요구 사항

- 클러스터에 대한 로드 밸런서를 구성했습니다.
- **cluster-admin** 역할의 사용자로 로그인되어 있습니다.

프로세스

1. 가상화 → 개요 로 이동합니다.
2. 설정 탭에서 클러스터를 클릭합니다.
3. 일반 설정 및 SSH 구성 을 확장합니다.
4. LoadBalancer 서비스를 통한 SSH를 on으로 설정합니다.

7.4.4.2.2. 웹 콘솔을 사용하여 서비스 생성

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)에 대한 노드 포트 또는 로드 밸런서 서비스를 생성할 수 있습니다.

사전 요구 사항

- 로드 밸런서 또는 노드 포트를 지원하도록 클러스터 네트워크를 구성했습니다.
- 로드 밸런서 서비스를 생성하려면 로드 밸런서 서비스 생성을 활성화했습니다.

프로세스

1. VirtualMachines 로 이동하여 가상 머신을 선택하여VirtualMachine 세부 정보 페이지를 확인합니다.
2. 세부 정보 탭의 SSH 서비스 유형 목록에서 LoadBalancer를 통한SSH 를 선택합니다.
3. 선택 사항: 복사 아이콘을 클릭하여 **SSH** 명령을 클립보드에 복사합니다.

검증

- 세부 정보 탭에서 서비스 창을 선택하여 새 서비스를 확인합니다.

7.4.4.2.3. virtctl을 사용하여 서비스 생성

virtctl 명령줄 툴을 사용하여 VM(가상 머신)에 대한 서비스를 생성할 수 있습니다.

사전 요구 사항

- **virtctl** 명령줄 툴을 설치했습니다.
- 서비스를 지원하도록 클러스터 네트워크를 구성했습니다.
- **virtctl** 을 설치한 환경에는 VM에 액세스하는 데 필요한 클러스터 권한이 있습니다. 예를 들어**oc login** 을 실행하거나**KUBECONFIG** 환경 변수를 설정합니다.

프로세스

- 다음 명령을 실행하여 서비스를 생성합니다.

```
$ virtctl expose vm <vm_name> --name <service_name> --type <service_type> --port
<port> 1
```

1 **ClusterIP,NodePort** 또는 **LoadBalancer** 서비스 유형을 지정합니다.

예제

```
$ virtctl expose vm example-vm --name example-service --type NodePort --port 22
```

검증

- 다음 명령을 실행하여 서비스를 확인합니다.

```
$ oc get service
```

다음 단계

virtctl 을 사용하여 서비스를 생성한 후**VirtualMachine** 매니페스트의 **spec.template.metadata.labels** 스탠자에 **special: key** 를 추가해야 합니다. 명령줄을 사용하여 서비스 생성을 참조하십시오.

7.4.4.2.4. 명령줄을 사용하여 서비스 생성

명령줄을 사용하여 서비스를 생성하고 VM(가상 머신)과 연결할 수 있습니다.

사전 요구 사항

- 서비스를 지원하도록 클러스터 네트워크를 구성했습니다.

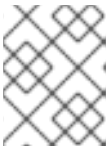
프로세스

1. **VirtualMachine** 매니페스트를 편집하여 서비스 생성 레이블을 추가합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
```

```
  namespace: example-namespace
spec:
  running: false
  template:
    metadata:
      labels:
        special: key (1)
# ...
```

(1) spec.template.metadata.labels 스탠자에 special: key 를 추가합니다.



참고

가상 머신의 라벨은 Pod로 전달됩니다. special: 키 레이블은서비스 매니페스트의 spec.selector 속성의 레이블과 일치해야 합니다.

2. VirtualMachine 매니페스트 파일을 저장하여 변경 사항을 적용합니다.

3. VM을 노출하는 서비스 매니페스트를 생성합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  namespace: example-namespace
spec:
# ...
  selector:
    special: key (1)
  type: NodePort (2)
  ports: (3)
    protocol: TCP
    port: 80
    targetPort: 9376
    nodePort: 30000
```

(1) VirtualMachine 매니페스트의 spec.template.metadata.labels 스탠자에 추가한 라벨을 지정합니다.

(2) ClusterIP,NodePort 또는 LoadBalancer 를 지정합니다.

(3) 가상 머신에서 노출하려는 네트워크 포트 및 프로토콜 컬렉션을 지정합니다.

4. 서비스 매니페스트 파일을 저장합니다.

5. 다음 명령을 실행하여 서비스를 생성합니다.

```
$ oc create -f example-service.yaml
```

6. VM을 다시 시작하여 변경 사항을 적용합니다.

검증

- Service 오브젝트를 쿼리하여 사용 가능한지 확인합니다.

```
$ oc get service -n example-namespace
```

7.4.4.3. SSH를 사용하여 서비스에서 노출하는 VM에 연결

SSH를 사용하여 서비스에서 노출하는 VM(가상 머신)에 연결할 수 있습니다.

사전 요구 사항

- VM을 노출하는 서비스를 생성하셨습니다.
- SSH 클라이언트가 설치되어 있어야 합니다.
- 클러스터에 로그인되어 있습니다.

프로세스

- 다음 명령을 실행하여 VM에 액세스합니다.

```
$ ssh <user_name>@<ip_address> -p <port> 1
```

1 클러스터 IP 서비스의 클러스터 IP, 노드 포트 서비스의 노드 IP 또는 로드 밸런서 서비스의 외부 IP 주소를 지정합니다.

7.4.5. SSH 액세스에 보조 네트워크 사용

보조 네트워크를 구성하고 VM(가상 머신)을 보조 네트워크 인터페이스에 연결한 다음 SSH를 사용하여 DHCP 할당된 IP 주소에 연결할 수 있습니다.

중요

보조 네트워크는 클러스터 네트워크 스택에서 트래픽을 처리하지 않기 때문에 우수한 성능을 제공합니다. 그러나 VM은 보조 네트워크에 직접 노출되며 방화벽에 의해 보호되지 않습니다. VM이 손상되면 침입자가 보조 네트워크에 액세스할 수 있습니다. 이 방법을 사용하는 경우 VM의 운영 체제 내에서 적절한 보안을 구성해야 합니다.

네트워킹 옵션에 대한 자세한 내용은 OpenShift Virtualization 튜닝 및 확장 가이드의 Multus 및 SR-IOV 설명서를 참조하십시오.

사전 요구 사항

- Linux 브리지 또는 SR-IOV 와 같은 보조 네트워크를 구성했습니다.
- Linux 브리지 네트워크 또는 SR-IOV Network Operator에 대한 네트워크연결 정의를 생성한 경우 SriovNetwork 오브젝트를 생성할 때 네트워크 연결 정의가 생성되었습니다.

7.4.5.1. 웹 콘솔을 사용하여 VM 네트워크 인터페이스 구성

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)의 네트워크 인터페이스를 구성할 수 있습니다.

사전 요구 사항

- 네트워크에 대한 네트워크 연결 정의를 생성했습니다.

프로세스

1. 가상화 → VirtualMachines 로 이동합니다.
2. VM을 클릭하여 VirtualMachine 세부 정보 페이지를 확인합니다.
3. 구성 탭에서 네트워크 인터페이스 탭을 클릭합니다.
4. 네트워크 인터페이스 추가를 클릭합니다.
5. 인터페이스 이름을 입력하고 네트워크 목록에서 네트워크 연결 정의를 선택합니다.
6. 저장을 클릭합니다.
7. VM을 다시 시작하여 변경 사항을 적용합니다.

7.4.5.2. SSH를 사용하여 보조 네트워크에 연결된 VM에 연결

SSH를 사용하여 보조 네트워크에 연결된 VM(가상 머신)에 연결할 수 있습니다.

사전 요구 사항

- DHCP 서버를 사용하여 보조 네트워크에 VM을 연결했습니다.
- SSH 클라이언트가 설치되어 있어야 합니다.

프로세스

1. 다음 명령을 실행하여 VM의 IP 주소를 가져옵니다.

```
$ oc describe vm <vm_name> -n <namespace>
```

출력 예

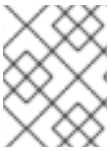
```
# ...
Interfaces:
  Interface Name:  eth0
  Ip Address:      10.244.0.37/24
  Ip Addresses:
    10.244.0.37/24
    fe80::858:aff:fef4:25/64
  Mac:             0a:58:0a:f4:00:25
  Name:            default
# ...
```

2. 다음 명령을 실행하여 VM에 연결합니다.

```
$ ssh <user_name>@<ip_address> -i <ssh_key>
```

예제

```
$ ssh cloud-user@10.244.0.37 -i ~/.ssh/id_rsa_cloud-user
```



참고

클러스터 FQDN을 사용하여 보조 네트워크 인터페이스에 연결된 VM에 액세스할 수도있습니다.

7.5. 가상 머신 편집

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신) 구성을 업데이트할 수 있습니다. YAML 파일 또는 VirtualMachine 세부 정보 페이지를 업데이트할 수 있습니다.

명령줄을 사용하여 VM을 편집할 수도 있습니다.

가상 디스크 또는 LUN을 사용하여 디스크 공유를 구성하도록 VM을 편집하려면 가상 머신의 공유 볼륨 구성 을 참조하십시오.

7.5.1. 명령줄을 사용하여 가상 머신 편집

명령줄을 사용하여 VM(가상 머신)을 편집할 수 있습니다.

사전 요구 사항

- oc CLI를 설치했습니다.

프로세스

1. 다음 명령을 실행하여 가상 머신 구성을 가져옵니다.

```
$ oc edit vm <vm_name>
```

2. YAML 구성을 편집합니다.
3. 실행 중인 가상 머신을 편집하는 경우 다음 중 하나를 수행해야 합니다.
   - 가상 머신을 재시작합니다.
   - 새 구성을 적용하려면 다음 명령을 실행합니다.

```
$ oc apply vm <vm_name> -n <namespace>
```

7.5.2. 가상 머신에 디스크 추가

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)에 가상 디스크를 추가할 수 있습니다.

프로세스

1. 웹 콘솔에서 가상화 → VirtualMachines 로 이동합니다.
2. VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 디스크 탭에서 디스크 추가 를 클릭합니다.

4. **소스 , 이름,크기,유형,인터페이스 및 스토리지 클래스를 지정합니다.**

    a. **선택 사항: 빈 디스크 소스를 사용하고 데이터 볼륨을 생성할 때 최대 쓰기 성능이 필요한 경우 사전 할당을 활성화할 수 있습니다. 이를 수행하려면 사전 할당 활성화 확인란을 선택합니다.**

    b. **선택 사항: 최적화된 StorageProfile 설정 적용을지워 가상 디스크의볼륨 모드 및 액세스 모드를 변경할 수 있습니다. 이러한 매개변수를 지정하지 않으면kubevirt-storage-class-defaults 구성 맵의 기본값이 사용됩니다.**

5. **추가를 클릭합니다.**

참고

VM이 실행 중인 경우 변경 사항을 적용하려면 VM을 다시 시작해야 합니다.

7.5.2.1. 스토리지 필드

필드	설명
비어있음 (PVC 생성)	빈 디스크를 만듭니다.
URL을 통해 가져오기 (PVC 생성)	URL(HTTP 또는 HTTPS 끝점)을 통해 콘텐츠를 가져옵니다.
기존 PVC 사용	클러스터에서 이미 사용 가능한 PVC를 사용합니다.
기존 PVC 복제 (PVC 생성)	클러스터에서 사용 가능한 기존 PVC를 선택하고 복제합니다.
레지스트리를 통해 가져오기(PVC 생성)	컨테이너 레지스트리를 통해 콘텐츠를 가져옵니다.
컨테이너 (임시)	클러스터에서 액세스할 수 있는 레지스트리에 있는 컨테이너에서 콘텐츠를 업로드합니다. 컨테이너 디스크는 CD-ROM 또는 임시 가상 머신과 같은 읽기 전용 파일 시스템에만 사용해야 합니다.
이름	디스크 이름입니다. 이름에는 소문자(**a-z**), 숫자(**0-9**), 하이픈(**-**), 마침표(**.**)가 최대 253자까지 포함될 수 있습니다. 첫 문자와 마지막 문자는 영숫자여야 합니다. 이름에는 대문자, 공백 또는 특수 문자가 포함되어서는 안 됩니다.
크기	디스크 크기(GiB)입니다.
유형	디스크의 유형입니다. 예: 디스크 또는 CD-ROM
인터페이스	디스크 장치의 유형입니다. 지원되는 인터페이스는 **virtIO**, **SATA**, **SCSI**입니다.
스토리지 클래스	디스크를 만드는 데 사용되는 스토리지 클래스입니다.

고급 스토리지 설정

다음 고급 스토리지 설정은 선택 사항이며 Blank 에서 사용할 수 있으며URL을 통해 가져오기, 기존 PVC 복제 디스크에 사용할 수 있습니다.

이러한 매개변수를 지정하지 않으면 시스템은 기본 스토리지 프로필 값을 사용합니다.

매개변수	옵션	매개변수 설명
볼륨 모드	파일 시스템	파일 시스템 기반 볼륨에 가상 디스크를 저장합니다.
	블록	가상 디스크를 블록 볼륨에 직접 저장합니다. 기본 스토리지에서 지원하는 경우에만 **Block**을 사용하십시오.
액세스 모드	ReadWriteOnce (RWO)	볼륨은 단일 노드에서 읽기-쓰기로 마운트할 수 있습니다.
	ReadWriteMany (RWX)	볼륨은 한 번에 여러 노드에서 읽기-쓰기로 마운트할 수 있습니다.  **참고** 이 모드는 실시간 마이그레이션에 필요합니다.

7.5.3. 가상 머신에 Windows 드라이버 디스크 마운트

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)에 Windows 드라이버 디스크를 마운트할 수 있습니다.

프로세스

1. 가상화 → VirtualMachines 로 이동합니다.
2. 필요한 VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 구성 탭에서 스토리지를 클릭합니다.
4. Mount Windows drivers disk 확인란을 선택합니다.
   Windows 드라이버 디스크가 마운트된 디스크 목록에 표시됩니다.

7.5.4. 가상 머신에 시크릿, 구성 맵 또는 서비스 계정 추가

OpenShift Container Platform 웹 콘솔을 사용하여 가상 머신에 시크릿, 구성 맵 또는 서비스 계정을 추가합니다.

이러한 리소스는 가상 머신에 디스크로 추가됩니다. 그런 다음 다른 디스크를 마운트하는 것처럼 시크릿, 구성 맵 또는 서비스 계정을 마운트합니다.

가상 머신이 실행 중인 경우 가상 머신을 재시작해야 변경 사항이 적용됩니다. 새로 추가된 리소스는 페이지 상단에 보류 중인 변경 사항으로 표시됩니다.

사전 요구 사항

- 추가할 시크릿, 구성 맵 또는 서비스 계정은 대상 가상 머신과 동일한 네임스페이스에 있어야 합니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 가상 머신을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 구성 → 환경을 클릭합니다.
4. 구성 맵, 시크릿 또는 서비스 계정 추가를 클릭합니다.
5. 리소스 선택을 클릭하고 목록에서 리소스를 선택합니다. 선택한 리소스에 대해 6자리 일련 번호가 자동으로 생성됩니다.
6. 선택 사항: 다시 로드 를 클릭하여 환경을 마지막 저장된 상태로 되돌립니다.
7. 저장을 클릭합니다.

검증

1. VirtualMachine 세부 정보 페이지에서 구성 → 디스크 를 클릭하고 리소스가 디스크 목록에 표시되는지 확인합니다.
2. 작업 → 재시작을 클릭하여 가상 머신을재시작 합니다.

이제 다른 디스크를 마운트할 때와 같이 시크릿, 구성 맵 또는 서비스 계정을 마운트할 수 있습니다.

구성 맵, 시크릿 및 서비스 계정에 대한 추가 리소스

- 구성 맵 이해
- Pod에 민감한 데이터 제공
- 서비스 계정 이해 및 생성

7.6. 부팅 순서 편집

웹 콘솔 또는 CLI를 사용하여 부팅 순서 목록 값을 업데이트할 수 있습니다.

가상 머신 개요 페이지의 부팅 순서를 사용하여 다음을 수행할 수 있습니다.

- 디스크 또는 NIC(네트워크 인터페이스 컨트롤러)를 선택하고 부팅 순서 목록에 추가합니다.
- 부팅 순서 목록에서 디스크 또는 NIC 순서를 편집합니다.
- 부팅 순서 목록에서 디스크 또는 NIC를 제거하고 부팅 가능 소스 인벤토리로 반환합니다.

7.6.1. 웹 콘솔에서 부팅 순서 목록에 항목 추가

웹 콘솔을 사용하여 부팅 순서 목록에 항목을 추가합니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 가상 머신을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.

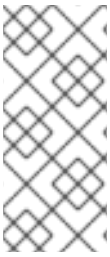
3. 세부 정보 탭을 클릭합니다.

4. 부팅 순서 오른쪽에 있는 연필 아이콘을 클릭합니다. YAML 구성이 존재하지 않거나 처음으로 부팅 순서 목록을 생성하는 경우 다음 메시지가 표시됩니다. 선택된 리소스가 없습니다. VM은 YAML 파일에 나타나는 순서에 따라 디스크에서 부팅하려고 합니다.

5. 소스 추가를 클릭하고 가상 시스템의 부팅 가능한 디스크 또는 NIC(네트워크 인터페이스 컨트롤러)를 선택합니다.

6. 부팅 순서 목록에 추가 디스크 또는 NIC를 추가합니다.

7. 저장을 클릭합니다.



참고

가상 머신이 실행 중인 경우 가상 머신을 재시작해야 Boot Order 변경 사항이 적용됩니다.

부팅 순서 필드 오른쪽에 있는보류 중인 변경 사항 보기를 클릭하면 보류 중인 변경 사항을 볼 수 있습니다. 페이지 상단의 보류 중인 변경 사항배너에는 가상 머신이 재시작될 때 적용되는 모든 변경 사항 목록이 표시됩니다.

7.6.2. 웹 콘솔에서 부팅 순서 목록 편집

웹 콘솔에서 부팅 순서 목록을 편집합니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.

2. 가상 머신을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.

3. 세부 정보 탭을 클릭합니다.

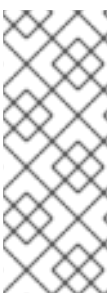
4. 부팅 순서 오른쪽에 있는 연필 아이콘을 클릭합니다.

5. 부팅 순서 목록에서 항목을 이동하는 적절한 방법을 선택합니다.

   - 화면 판독기를 사용하지 않는 경우 이동할 항목 옆에 있는 화살표 아이콘 위로 마우스를 가져가서 항목을 위 또는 아래로 끌어 원하는 위치에 놓습니다.

   - 화면 판독기를 사용하는 경우 위쪽 화살표 키 또는 아래쪽 화살표 키를 눌러 부팅 순서 목록에서 항목을 이동합니다. 그런 다음 Tab 키를 눌러 원하는 위치에 항목을 놓습니다.

6. 저장을 클릭합니다.



참고

가상 머신이 실행중인 경우 가상 머신을 재시작해야 부팅 순서 목록의 변경 사항이 적용됩니다.

부팅 순서 필드 오른쪽에 있는보류 중인 변경 사항 보기를 클릭하면 보류 중인 변경 사항을 볼 수 있습니다. 페이지 상단의 보류 중인 변경 사항배너에는 가상 머신이 재시작될 때 적용되는 모든 변경 사항 목록이 표시됩니다.

7.6.3. YAML 구성 파일의 부팅 순서 목록 편집

CLI를 사용하여 YAML 구성 파일의 부팅 순서 목록을 편집합니다.

절차

1. 다음 명령을 실행하여 가상 머신의 YAML 구성 파일을 엽니다.

```
$ oc edit vm <vm_name> -n <namespace>
```

2. YAML 파일을 편집하고 디스크 또는 NIC(네트워크 인터페이스 컨트롤러)와 연결된 부팅 순서 값을 수정합니다. 예를 들면 다음과 같습니다.

```
disks:
  - bootOrder: 1 1
    disk:
      bus: virtio
    name: containerdisk
  - disk:
      bus: virtio
    name: cloudinitdisk
  - cdrom:
      bus: virtio
    name: cd-drive-1
interfaces:
  - boot Order: 2 2
    macAddress: '02:96:c4:00:00'
    masquerade: {}
    name: default
```

1 디스크에 지정된 부팅 순서 값입니다.

2 네트워크 인터페이스 컨트롤러에 지정된 부팅 순서 값입니다.

3. YAML 파일을 저장합니다.

7.6.4. 웹 콘솔의 부팅 순서 목록에서 항목 제거

웹 콘솔을 사용하여 부팅 순서 목록에서 항목을 제거합니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 가상 머신을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 세부 정보 탭을 클릭합니다.
4. 부팅 순서 오른쪽에 있는 연필 아이콘을 클릭합니다.
5. 해당 항목 옆에 있는 제거 아이콘 ⛔ 을 클릭합니다. 항목이 부팅 순서 목록에서 제거되고 사용 가능한 부팅 소스 목록에 저장됩니다. 부팅 순서 목록의 모든 항목을 제거하면 다음 메시지가 표시됩니다. 선택된 리소스가 없습니다. VM은 YAML 파일에 나타나는 순서에 따라 디스크에서 부팅하려고 합니다.

참고

가상 머신이 실행 중인 경우 가상 머신을 재시작해야 Boot Order 변경 사항이 적용됩니다.

부팅 순서 필드 오른쪽에 있는보류 중인 변경 사항 보기를 클릭하면 보류 중인 변경 사항을 볼 수 있습니다. 페이지 상단의 보류 중인 변경 사항 배너에는 가상 머신이 재시작될 때 적용되는 모든 변경 사항 목록이 표시됩니다.

7.7. 가상 머신 삭제

웹 콘솔에서 또는 oc 명령줄 인터페이스를 사용하여 가상 머신을 삭제할 수 있습니다.

7.7.1. 웹 콘솔을 사용하여 가상 머신 삭제

가상 머신을 삭제하면 클러스터에서 가상 머신이 영구적으로 제거됩니다.

프로세스

1. OpenShift Container Platform 콘솔의 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 가상 머신 옆에 있는 옵션 메뉴 를 클릭하고 삭제 를 선택합니다.
   또는 가상 머신 이름을 클릭하여 VirtualMachine 세부 정보 페이지를 열고 작업 → 삭제 를 클릭합니다.
3. 선택 사항: 유예 기간 동안 디스크를 선택하거나 삭제 디스크를 지웁니다.
4. 삭제 를 클릭하여 가상 머신을 영구적으로 삭제합니다.

7.7.2. CLI를 사용하여 가상 머신 삭제

oc CLI(명령줄 인터페이스)를 사용하여 가상 머신을 삭제할 수 있습니다.oc 클라이언트를 사용하면 여러 가상 머신에서 작업을 수행할 수 있습니다.

사전 요구 사항

- 삭제할 가상 머신의 이름을 확인합니다.

절차

- 다음 명령을 실행하여 가상 머신을 삭제합니다.

```
$ oc delete vm <vm_name>
```

참고

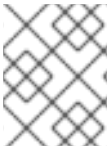
이 명령은 현재 프로젝트의 VM만 삭제합니다. 삭제하려는 VM이 다른 프로젝트 또는 네임스페이스에 있는 경우 -n <project_name > 옵션을 지정합니다.

7.8. 가상 머신 내보내기

VM(가상 머신) 및 관련 디스크를 내보내 VM을 다른 클러스터로 가져오거나 법의 목적으로 볼륨을 분석할 수 있습니다.

명령줄 인터페이스를 사용하여 **VirtualMachineExport** CR(사용자 정의 리소스)을 생성합니다.

또는 **virtctl vmexport 명령을 사용하여** **VirtualMachineExport** CR을 생성하고 내보낸 볼륨을 다운로드할 수 있습니다.



참고

Migration Toolkit for Virtualization을 사용하여 OpenShift Virtualization 클러스터 간에 가상 머신을 마이그레이션할 수 있습니다.

7.8.1. VirtualMachineExport 사용자 정의 리소스 생성

VirtualMachineExport CR(사용자 정의 리소스)을 생성하여 다음 오브젝트를 내보낼 수 있습니다.

- 가상 머신(VM): 지정된 VM의 PVC(영구 볼륨 클레임)를 내보냅니다.
- VM 스냅샷: **VirtualMachineSnapshot** CR에 포함된 PVC를 내보냅니다.
- PVC: PVC를 내보냅니다. **virt-launcher** Pod와 같은 다른 Pod에서 PVC를 사용하는 경우 PVC가 더 이상 사용되지 않을 때까지 내보내기는 **Pending** 상태로 유지됩니다.

VirtualMachineExport CR은 내보낸 볼륨에 대한 내부 및 외부 링크를 생성합니다. 내부 링크는 클러스터 내에서 유효합니다. 외부 링크는 **Ingress** 또는 경로를 사용하여 액세스할 수있습니다.

내보내기 서버는 다음 파일 형식을 지원합니다.

- **raw**: 원시 디스크 이미지 파일입니다.
- **gzip**: 압축 디스크 이미지 파일.
- **dir**: PVC 디렉토리 및 파일
- **tar.gz**: 압축 된 PVC 파일.

사전 요구 사항

- VM 내보내기에는 VM을 종료해야 합니다.

프로세스

1. 다음 예에 따라 VirtualMachine , **VirtualMachine Snapshot** 또는 **PersistentVolumeClaim** CR에서 볼륨을 내보내는 **VirtualMachineExport** 매니페스트를 생성하고 **example-export.yaml** 로 저장합니다.

   VirtualMachineExport 예

```
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
```

```
    apiGroup: "kubevirt.io" 1
    kind: VirtualMachine 2
    name: example-vm
  ttlDuration: 1h 3
```

1 적절한 API 그룹을 지정합니다.

- VirtualMachine 의 경우 "kubevirt.io ".
- VirtualMachineSnapshot 의 경우 "snapshot.kubevirt.io ".
- PersistentVolumeClaim 의 경우 "" 입니다.

2 VirtualMachine,VirtualMachineSnapshot 또는 PersistentVolumeClaim 을 지정합니다.

3 선택 사항: 기본 기간은 2시간입니다.

2. VirtualMachineExport CR을 생성합니다.

```
$ oc create -f example-export.yaml
```

3. VirtualMachineExport CR을 가져옵니다.

```
$ oc get vmexport example-export -o yaml
```

내보낸 볼륨의 내부 및 외부 링크가 status 스탠자에 표시됩니다.

출력 예

```
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export
  namespace: example
spec:
  source:
    apiGroup: ""
    kind: PersistentVolumeClaim
    name: example-pvc
  tokenSecretRef: example-token
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2022-06-21T14:10:09Z"
    reason: podReady
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2022-06-21T14:09:02Z"
    reason: pvcBound
    status: "True"
    type: PVCReady
  links:
    external: 1
```

```
    cert: |-
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
    volumes:
    - formats:
      - format: raw
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexp
orts/example-export/volumes/example-disk/disk.img
      - format: gzip
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexp
orts/example-export/volumes/example-disk/disk.img.gz
      name: example-disk
  internal:  2
    cert: |-
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
    volumes:
    - formats:
      - format: raw
        url: https://virt-export-example-export.example.svc/volumes/example-
disk/disk.img
      - format: gzip
        url: https://virt-export-example-export.example.svc/volumes/example-
disk/disk.img.gz
      name: example-disk
 phase: Ready
 serviceName: virt-export-example-export
```

1 외부 링크는 Ingress 또는 경로를 사용하여 클러스터 외부에서 액세스할 수있습니다.

2 내부 링크는 클러스터 내에서만 유효합니다.

7.8.2. 내보낸 가상 머신 매니페스트에 액세스

VM(가상 머신) 또는 스냅샷을 내보낸 후 내보내기 서버에서 VirtualMachine 매니페스트 및 관련 정보를 가져올 수 있습니다.

사전 요구 사항

- VirtualMachineExport CR(사용자 정의 리소스)을 생성하여 가상 머신 또는 VM 스냅샷을 내보냈습니다.

참고

spec.source.kind: PersistentVolumeClaim 매개변수가 있는 VirtualMachineExport 오브젝트는 가상 머신 매니페스트를 생성하지 않습니다.

프로세스

1. 매니페스트에 액세스하려면 먼저 소스 클러스터에서 대상 클러스터로 인증서를 복사해야 합니다.

a. 소스 클러스터에 로그인합니다.

b. 다음 명령을 실행하여 **cacert.crt** 파일에 인증서를 저장합니다.

```
$ oc get vmexport <export_name> -o jsonpath={.status.links.external.cert} >
cacert.crt (1)
```

(1) **VirtualMachineExport** 오브젝트의 **metadata.name** 값으로 바꿉니다 **<export_name>**.

c. **cacert.crt** 파일을 대상 클러스터에 복사합니다.

2. 다음 명령을 실행하여 소스 클러스터에서 토큰을 디코딩하고 **token_decode** 파일에 저장합니다.

```
$ oc get secret export-token-<export_name> -o jsonpath={.data.token} | base64 --
decode > token_decode (1)
```

(1) **VirtualMachineExport** 오브젝트의 **metadata.name** 값으로 바꿉니다 **<export_name>**.

3. **token_decode** 파일을 대상 클러스터에 복사합니다.

4. 다음 명령을 실행하여 **VirtualMachineExport** 사용자 정의 리소스를 가져옵니다.

```
$ oc get vmexport <export_name> -o yaml
```

5. 외부 및 내부 섹션으로 나누어진 **status.links** 스탠자를 검토합니다. 각 섹션 내의 **manifests.url** 필드를 확인합니다.

출력 예

```
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
    apiGroup: "kubevirt.io"
    kind: VirtualMachine
    name: example-vm
  tokenSecretRef: example-token
status:
#...
  links:
    external:
#...
      manifests:
      - type: all
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexp
orts/example-export/external/manifests/all (1)
      - type: auth-header-secret
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexp
orts/example-export/external/manifests/secret (2)
```

```
    internal:
#...
      manifests:
      - type: all
        url: https://virt-export-export-pvc.default.svc/internal/manifests/all (3)
      - type: auth-header-secret
        url: https://virt-export-export-pvc.default.svc/internal/manifests/secret
  phase: Ready
  serviceName: virt-export-example-export
```

(1) VirtualMachine 매니페스트, DataVolume 매니페스트(있는 경우) 및 외부 URL 수신 또는 경로에 대한 공용 인증서가 포함된 ConfigMap 매니페스트가 포함되어 있습니다.

(2) CDI(Containerized Data Importer)와 호환되는 헤더가 포함된 보안이 포함되어 있습니다. 헤더에는 내보내기 토큰의 텍스트 버전이 포함되어 있습니다.

(3) VirtualMachine 매니페스트, DataVolume 매니페스트(있는 경우) 및 내부 URL 내보내기 서버의 인증서가 포함된 ConfigMap 매니페스트가 포함되어 있습니다.

6. 대상 클러스터에 로그인합니다.
7. 다음 명령을 실행하여 시크릿 매니페스트를 가져옵니다.

```
$ curl --cacert cacert.crt <secret_manifest_url> -H \ (1)
"x-kubevirt-export-token:token_decode" -H \ (2)
"Accept:application/yaml"
```

(1) & lt;secret_manifest_url >을 VirtualMachineExport YAML 출력의 auth-header-secret URL로 바꿉니다.

(2) 이전에 생성한 token_decode 파일을 참조합니다.

예를 들면 다음과 같습니다.

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexp
orts/example-export/external/manifests/secret -H "x-kubevirt-export-
token:token_decode" -H "Accept:application/yaml"
```

8. 다음 명령을 실행하여 type: all (예: ConfigMap 및 VirtualMachine 매니페스트)의 매니페스트를 가져옵니다.

```
$ curl --cacert cacert.crt <all_manifest_url> -H \ (1)
"x-kubevirt-export-token:token_decode" -H \ (2)
"Accept:application/yaml"
```

(1) & lt;all_manifest_url& gt;을 VirtualMachineExport YAML 출력의 URL로 바꿉니다.

(2) 이전에 생성한 token_decode 파일을 참조합니다.

예를 들면 다음과 같습니다.

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexp
orts/example-export/external/manifests/all -H "x-kubevirt-export-token:token_decode"
-H "Accept:application/yaml"
```

다음 단계

- 내보낸 매니페스트를 사용하여 대상 클러스터에서 **ConfigMap** 및 **VirtualMachine** 오브젝트를 생성할 수 있습니다.

7.9. 가상 머신 인스턴스 관리

OpenShift Virtualization 환경 외부에서 독립적으로 생성된 독립 실행형 VMI(가상 머신 인스턴스)가 있는 경우 웹 콘솔을 사용하거나 CLI(명령줄 인터페이스)에서 **oc** 또는 **virtctl** 명령을 사용하여 관리할 수 있습니다.

virtctl 명령은 **oc** 명령보다 더 많은 가상화 옵션을 제공합니다. 예를 들어**virtctl** 을 사용하여 VM을 일시 중지하거나 포트를 노출할 수 있습니다.

7.9.1. 가상 머신 인스턴스 정보

VMI(가상 머신 인스턴스)는 실행 중인 VM(가상 머신)을 나타냅니다. VMI가 VM 또는 다른 오브젝트에 속하는 경우, 웹 콘솔의 해당 소유자를 통해 또는 **oc** CLI(명령줄 인터페이스)를 사용하여 VMI를 관리합니다.

독립 실행형 VMI는 스크립트, 자동화 또는 CLI의 다른 방법을 통해 독립적으로 생성 및 시작됩니다. OpenShift Virtualization 환경 외부에서 개발 및 시작된 독립 실행형 VMI가 사용자 환경에 있을 수 있습니다. CLI를 사용하여 이러한 독립 실행형 VMI를 계속 관리할 수 있습니다. 다음과 같이 독립 실행형 VMI와 관련된 특정 작업에 웹 콘솔을 사용할 수도 있습니다.

- 독립 실행형 VMI 및 세부 정보를 나열합니다.
- 독립 실행형 VMI의 라벨 및 주석을 편집합니다.
- 독립 실행형 VMI를 삭제합니다.

VM을 삭제하면 관련 VMI가 자동으로 삭제됩니다. 독립 실행형 VMI는 VM 또는 다른 오브젝트에 속하지 않기 때문에 직접 삭제합니다.

참고

OpenShift Virtualization을 설치 제거하기 전에 CLI 또는 웹 콘솔을 사용하여 독립 실행형 VMI를 나열하고 확인하십시오. 그런 다음 처리 중인 VMI를 삭제합니다.

VM을 편집하면 일부 설정이 VMI에 동적으로 적용될 수 있습니다. VMI에 동적으로 적용할 수 없는 VM 오브젝트를 변경하면 필수 **VM** 조건이 재시작 됩니다. 다음 재부팅 시 변경 사항이 적용되며 조건이 제거됩니다.

7.9.2. CLI를 사용하여 모든 가상 머신 인스턴스 나열

oc CLI(명령줄 인터페이스)를 사용하면 독립 실행형 VMI(가상 머신 인스턴스) 및 가상 머신에 속하는 VMI를 포함하여 클러스터의 모든 VMI를 나열할 수 있습니다.

절차

- 다음 명령을 실행하여 VMI를 모두 나열합니다.

```
$ oc get vmis -A
```

7.9.3. 웹 콘솔을 사용하여 독립 실행형 가상 머신 인스턴스 나열

웹 콘솔을 사용하면 VM(가상 머신)에 속하지 않는 클러스터의 독립 실행형 VMI(가상 머신 인스턴스)를 나열하고 확인할 수 있습니다.

참고

VM 또는 다른 오브젝트에 속하는 VMI는 웹 콘솔에 표시되지 않습니다. 웹 콘솔에는 독립 실행형 VMI만 표시됩니다. 클러스터의 모든 VMI를 나열하려면 CLI를 사용해야 합니다.

프로세스

- 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
  독립 실행형 VMI는 이름 옆에 어두운 색상의 배지로 식별할 수 있습니다.

7.9.4. 웹 콘솔을 사용하여 독립 실행형 가상 머신 인스턴스 편집

웹 콘솔을 사용하여 독립 실행형 VMI(가상 머신 인스턴스)의 주석 및 레이블을 편집할 수 있습니다. 다른 필드는 편집할 수 없습니다.

프로세스

1. OpenShift Container Platform 콘솔의 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 독립 실행형 VMI를 선택하여 VirtualMachineInstance 세부 정보 페이지를 엽니다.
3. 세부 정보 탭에서 주석 또는 라벨 옆에 있는 연필 아이콘을 클릭합니다.
4. 관련 사항을 변경하고 저장을 클릭합니다.

7.9.5. CLI를 사용하여 독립 실행형 가상 머신 인스턴스 삭제

oc CLI(명령줄 인터페이스)를 사용하여 독립 실행형 VMI(가상 머신 인스턴스)를 삭제할 수 있습니다.

사전 요구 사항

- 삭제할 VMI의 이름을 확인합니다.

절차

- 다음 명령을 실행하여 VMI를 삭제합니다.

```
$ oc delete vmi <vmi_name>
```

7.9.6. 웹 콘솔을 사용하여 독립 실행형 가상 머신 인스턴스 삭제

웹 콘솔에서 독립 실행형 VMI(가상 머신 인스턴스)를 삭제합니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 작업 → VirtualMachineInstance 삭제를 클릭합니다.
3. 확인 팝업 창에서 삭제를 클릭하여 독립 실행형 VMI를 영구적으로 삭제합니다.

7.10. 가상 머신 상태 제어

웹 콘솔에서 가상 머신을 중지, 시작, 재시작, 일시 정지 해제할 수 있습니다.

virtctl 을 사용하여 가상 머신 상태를 관리하고 CLI에서 다른 작업을 수행할 수 있습니다. 예를 들어virtctl 을 사용하여 VM을 강제 중지하거나 포트를 노출할 수 있습니다.

7.10.1. 가상 머신 시작

웹 콘솔에서 가상 머신을 시작할 수 있습니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 시작할 가상 머신이 포함된 행을 찾습니다.
3. 사용 사례에 적합한 메뉴로 이동합니다.
   - 여러 가상 머신에서 작업을 수행할 수 있는 이 페이지를 유지하려면 다음을 수행하십시오.
     a. 행의 맨 오른쪽 끝에 있는 옵션 메뉴  를 클릭하고 VirtualMachine 시작을 클릭합니다.
   - 시작하기 전에 선택한 가상 머신에 대한 포괄적인 정보를 보려면 다음을 수행하십시오.
     a. 가상 머신 이름을 클릭하여 VirtualMachine 세부 정보 페이지에 액세스합니다.
     b. 작업 → 시작을 클릭합니다.

참고

URL 소스에서 프로비저닝된 가상 머신을 처음 시작하면 가상 머신의 상태가가져오는 중이 되고 OpenShift Virtualization은 URL 끝점에서 컨테이너를 가져옵니다. 이미지 크기에 따라 이 프로세스에 몇 분이 걸릴 수 있습니다.

7.10.2. 가상 머신 중지

웹 콘솔에서 가상 머신을 중지할 수 있습니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 중지할 가상 머신이 포함된 행을 찾습니다.

3. 사용 사례에 적합한 메뉴로 이동합니다.
   - 여러 가상 머신에서 작업을 수행할 수 있는 이 페이지를 유지하려면 다음을 수행하십시오.
     a. 행의 맨 오른쪽에 있는 옵션 메뉴 를 클릭하고 VirtualMachine 중지 를 클릭합니다.
   - 중지하기 전에 선택한 가상 머신에 대한 포괄적인 정보를 보려면 다음을 수행하십시오.
     a. 가상 머신 이름을 클릭하여 VirtualMachine 세부 정보 페이지에 액세스합니다.
     b. 작업 → 중지 를 클릭합니다.

7.10.3. 가상 머신 재시작

웹 콘솔에서 실행 중인 가상 머신을 재시작할 수 있습니다.

중요

오류를 방지하려면 가져오는 중 상태의 가상 머신을 재시작하지 마십시오.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 재시작할 가상 머신이 포함된 행을 찾습니다.
3. 사용 사례에 적합한 메뉴로 이동합니다.
   - 여러 가상 머신에서 작업을 수행할 수 있는 이 페이지를 유지하려면 다음을 수행하십시오.
     a. 행의 맨 오른쪽 끝에 있는 옵션 메뉴 를 클릭하고 다시 시작을 클릭합니다.
   - 재시작하기 전에 선택한 가상 머신에 대한 포괄적인 정보를 보려면 다음을 수행하십시오.
     a. 가상 머신 이름을 클릭하여 VirtualMachine 세부 정보 페이지에 액세스합니다.
     b. 작업 → 재시작 을 클릭합니다.

7.10.4. 가상 머신 일시 중지

웹 콘솔에서 가상 머신을 일시 중지할 수 있습니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 일시 중지하려는 가상 머신이 포함된 행을 찾습니다.
3. 사용 사례에 적합한 메뉴로 이동합니다.
   - 여러 가상 머신에서 작업을 수행할 수 있는 이 페이지를 유지하려면 다음을 수행하십시오.

a. 행의 맨 오른쪽 끝에 있는 옵션 메뉴 를 클릭하고 Pause VirtualMachine 를 클릭합니다.

- 일시 정지하기 전에 선택한 가상 머신에 대한 포괄적인 정보를 보려면 다음을 수행하십시오.
  a. 가상 머신 이름을 클릭하여 VirtualMachine 세부 정보 페이지에 액세스합니다.
  b. 작업 → 일시 중지 를 클릭합니다.

7.10.5. 가상 머신 정지 해제

웹 콘솔에서 일시 정지된 가상 머신의 일시 정지를 해제할 수 있습니다.

사전 요구 사항

- 가상 머신 중 하나 이상의 상태가 일시 정지됨이어야 합니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 일시 정지를 해제할 가상 머신이 포함된 행을 찾습니다.
3. 사용 사례에 적합한 메뉴로 이동합니다.
   - 여러 가상 머신에서 작업을 수행할 수 있는 이 페이지를 유지하려면 다음을 수행하십시오.
     a. 행의 맨 오른쪽에 있는 옵션 메뉴 를 클릭하고 VirtualMachine 일시 중지 해제를 클릭합니다.
   - 일시 정지하기 전에 선택한 가상 머신에 대한 포괄적인 정보를 보려면 다음을 수행하십시오.
     a. 가상 머신 이름을 클릭하여 VirtualMachine 세부 정보 페이지에 액세스합니다.
     b. 작업 → 일시 중지를 클릭합니다.

7.11. 가상 신뢰할 수 있는 플랫폼 모듈 장치 사용

VirtualMachine(VM) 또는 **VirtualMachine Instance** (VM) 매니페스트를 편집하여 새 또는 기존 가상 머신에 vTPM(가상 신뢰할 수 있는 플랫폼 모듈) 장치를 추가합니다.

7.11.1. vTPM 장치 정보

가상 신뢰할 수 있는 플랫폼 모듈(vTPM) 장치는 물리적 신뢰할 수 있는 플랫폼 모듈(TPM) 하드웨어 칩과 같은 기능을 합니다.

운영 체제와 함께 vTPM 장치를 사용할 수 있지만 Windows 11에서는 설치 또는 부팅을 위해 TPM 칩이 있어야 합니다. vTPM 장치를 사용하면 Windows 11 이미지에서 생성된 VM이 물리적 TPM 칩 없이 작동할 수 있습니다.

vTPM을 활성화하지 않으면 노드에 있는 경우에도 VM에서 TPM 장치를 인식하지 못합니다.

또한 vTPM 장치는 물리적 하드웨어 없이 시크릿을 저장하여 가상 머신을 보호합니다. OpenShift Virtualization은 VM에 PVC(영구 볼륨 클레임)를 사용하여 vTPM 장치 상태 유지를 지원합니다. **HyperConverged** CR(사용자 정의 리소스)에서 **vmStateStorageClass** 속성을 설정하여 PVC에서 사용할

스토리지 클래스를 지정해야 합니다.

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  vmStateStorageClass: <storage_class_name>

# ...
```

참고

스토리지 클래스는 **Filesystem** 유형이어야 하며 RWX(**ReadWriteMany**) 액세스 모드를 지원해야 합니다.

7.11.2. 가상 머신에 vTPM 장치 추가

가상 머신(VM)에 vTPM(가상 신뢰할 수 있는 플랫폼 모듈) 장치를 추가하면 물리적 TPM 장치 없이 Windows 11 이미지에서 생성된 VM을 실행할 수 있습니다. vTPM 장치는 해당 VM의 시크릿도 저장합니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- RWX(**ReadWriteMany**) 액세스 모드를 지원하는 **Filesystem** 유형의 스토리지 클래스를 사용하도록 PVC(영구 볼륨 클레임)를 구성했습니다. 이 작업은 VM 재부팅 시 vTPM 장치 데이터를 유지해야 합니다.

프로세스

1. 다음 명령을 실행하여 VM 구성을 업데이트합니다.

```
$ oc edit vm <vm_name> -n <namespace>
```

2. VM 사양을 편집하여 vTPM 장치를 추가합니다. 예를 들면 다음과 같습니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
   name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          tpm: 1
            persistent: true 2
# ...
```

1. vTPM 장치를 VM에 추가합니다.
2. VM이 종료된 후 vTPM 장치 상태가 유지되도록 지정합니다. 기본값은 **false**입니다.

3. 변경 사항을 적용하려면 편집기를 저장하고 종료합니다.

4. 선택 사항: 실행 중인 가상 머신을 편집한 경우 변경 사항을 적용하려면 가상 머신을 다시 시작해야 합니다.

7.12. OPENSHIFT PIPELINES를 사용하여 가상 머신 관리

Red Hat OpenShift Pipelines는 Kubernetes 네이티브 CI/CD 프레임워크로, 개발자가 자체 컨테이너에서 CI/CD 파이프라인의 각 단계를 설계 및 실행할 수 있습니다.

SSP(Scheduling, Scale, Performance) Operator는 OpenShift Virtualization과 OpenShift Pipelines를 통합합니다. SSP Operator에는 다음을 수행할 수 있는 작업 및 예제 파이프라인이 포함되어 있습니다.

- 가상 머신(VM), PVC(영구 볼륨 클레임) 및 데이터 볼륨 생성 및 관리
- VM에서 명령 실행
- **libguestfs** 툴을 사용하여 디스크 이미지 조작

7.12.1. 사전 요구 사항

- **cluster-admin** 권한이 있는 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.
- OpenShift Pipelines를 설치했습니다.

7.12.2. SSP Operator에서 지원하는 가상 머신 작업

다음 표에서는 SSP Operator의 일부로 포함된 작업을 보여줍니다.

표 7.4. SSP Operator에서 지원하는 가상 머신 작업

Task	설명
create-vm-from-manifest	제공된 매니페스트 또는 **virtctl** 을 사용하여 가상 머신을 생성합니다.
create-vm-from-template	템플릿에서 가상 머신을 생성합니다.
copy-template	가상 머신 템플릿을 복사합니다.
modify-vm-template	가상 머신 템플릿을 수정합니다.
modify-data-object	데이터 볼륨 또는 데이터 소스를 생성하거나 삭제합니다.
cleanup-vm	가상 머신에서 스크립트 또는 명령을 실행하고 나중에 가상 머신을 중지하거나 삭제합니다.
disk-virt-customize	**virt-customize** 툴을 사용하여 대상 PVC에서 사용자 지정 스크립트를 실행합니다.

Task	설명
disk-virt-sysprep	**virt-sysprep** 툴을 사용하여 대상 PVC에서 sysprep 스크립트를 실행합니다.
wait-for-vmi-status	가상 머신 인스턴스의 특정 상태를 기다린 후 상태에 따라 실패하거나 성공합니다.

참고

이제 파이프라인에서 가상 머신 생성에서 더 이상 사용되지 않는 템플릿 기반 작업 대신 **ClusterInstanceType** 및 **ClusterPreference** 를 사용합니다. **create-vm-from-template,copy-template, modify-vm-template** 명령은 계속 사용할 수 있지만 기본 파이프라인 작업에는 사용되지 않습니다.

7.12.3. Windows EFI 설치 프로그램 파이프라인

웹 콘솔 또는 CLI를 사용하여 Windows EFI 설치 프로그램 파이프라인을 실행할 수 있습니다.

Windows EFI 설치 프로그램 파이프라인은 Windows 10, Windows 11 또는 Windows Server 2022를 Windows 설치 이미지(ISO 파일)의 새 데이터 볼륨에 설치합니다. 사용자 지정 응답 파일은 설치 프로세스를 실행하는 데 사용됩니다.

참고

Windows EFI 설치 프로그램 파이프라인은 OpenShift Container Platform에서 사전 정의하고 Microsoft ISO 파일에 적합한 **sysprep** 이 사전 정의된 구성 맵 파일을 사용합니다. 다른 Windows Edition과 관련된 ISO 파일의 경우 시스템별 **sysprep** 정의를 사용하여 새 구성 맵 파일을 생성해야 할 수 있습니다.

7.12.3.1. 웹 콘솔을 사용하여 예제 파이프라인 실행

웹 콘솔의 파이프라인 메뉴에서 예제 파이프라인 을 실행할 수 있습니다.

프로세스

1. 사이드 메뉴에서 파이프라인 → 파이프라인 을 클릭합니다.
2. 파이프라인 세부 정보 페이지를 열려면 파이프라인 을 선택합니다.
3. 작업 목록에서 시작을 선택합니다. Start Pipeline 대화 상자가 표시됩니다.
4. 매개변수의 기본값을 유지하고 시작을 클릭하여 파이프라인을 실행합니다. 세부 정보 탭에서는 각 작업의 진행 상황을 추적하고 파이프라인 상태를 표시합니다.

7.12.3.2. CLI를 사용하여 예제 파이프라인 실행

PipelineRun 리소스를 사용하여 예제 파이프라인을 실행합니다. **PipelineRun** 오브젝트는 파이프라인의 실행 중인 인스턴스입니다. 클러스터에서 특정 입력, 출력 및 실행 매개변수를 사용하여 실행할 파이프라인을 인스턴스화합니다. 또한 파이프라인의 각 작업에 대해 **TaskRun** 오브젝트를 생성합니다.

프로세스

1. Windows 10 설치 프로그램 파이프라인을 실행하려면 다음 **PipelineRun** 매니페스트를 생성합니다.

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-installer-run-
  labels:
    pipelinerun: windows10-installer-run
spec:
  params:
  - name: winImageDownloadURL
    value: <link_to_windows_10_iso> (1)
  pipelineRef:
    name: windows10-installer
  taskRunSpecs:
    - pipelineTaskName: copy-template
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template
      taskServiceAccountName: modify-vm-template-task
    - pipelineTaskName: create-vm-from-template
      taskServiceAccountName: create-vm-from-template-task
    - pipelineTaskName: wait-for-vmi-status
      taskServiceAccountName: wait-for-vmi-status-task
    - pipelineTaskName: create-base-dv
      taskServiceAccountName: modify-data-object-task
    - pipelineTaskName: cleanup-vm
      taskServiceAccountName: cleanup-vm-task
  status: {}
```

(1) Windows 10 64비트 ISO 파일의 URL을 지정합니다. 제품 언어는 영어(미국)여야 합니다.

2. **PipelineRun** 매니페스트를 적용합니다.

```
$ oc apply -f windows10-installer-run.yaml
```

3. Windows 10 사용자 지정 파이프라인을 실행하려면 다음 **PipelineRun** 매니페스트를 생성합니다.

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-customize-run-
  labels:
    pipelinerun: windows10-customize-run
spec:
  params:
    - name: allowReplaceGoldenTemplate
      value: true
    - name: allowReplaceCustomizationTemplate
      value: true
  pipelineRef:
    name: windows10-customize
```

```
  taskRunSpecs:
    - pipelineTaskName: copy-template-customize
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template-customize
      taskServiceAccountName: modify-vm-template-task
    - pipelineTaskName: create-vm-from-template
      taskServiceAccountName: create-vm-from-template-task
    - pipelineTaskName: wait-for-vmi-status
      taskServiceAccountName: wait-for-vmi-status-task
    - pipelineTaskName: create-base-dv
      taskServiceAccountName: modify-data-object-task
    - pipelineTaskName: cleanup-vm
      taskServiceAccountName: cleanup-vm-task
    - pipelineTaskName: copy-template-golden
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template-golden
      taskServiceAccountName: modify-vm-template-task
status: {}
```

4. **PipelineRun** 매니페스트를 적용합니다.

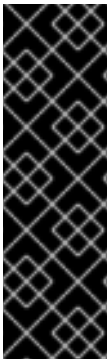
```
$ oc apply -f windows10-customize-run.yaml
```

7.12.4. 추가 리소스

- Red Hat OpenShift Pipelines를 사용하여 애플리케이션용 CI/CD 솔루션 생성
- Windows VM 생성

7.13. 더 높은 VM 워크로드 밀도 구성

VM(가상 머신) 수를 늘리려면 RAM(메모리) 양을 과다 할당하여 클러스터에서 더 높은 VM 워크로드 밀도를 구성할 수 있습니다.



중요

높은 워크로드 밀도를 구성하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

다음 워크로드는 특히 높은 워크로드 밀도에 적합합니다.

- 많은 유사한 워크로드
- 사용되지 않는 워크로드

참고

오버 커밋된 메모리는 워크로드 밀도가 높을 수 있지만 고도로 사용되는 시스템의 워크로드 성능을 저하시킬 수 있습니다.

7.13.1. wasp-agent 를 사용하여 더 높은 VM 워크로드 밀도 구성

wasp-agent 구성 요소를 사용하면 OpenShift Container Platform 클러스터에서 스왑 리소스를 VM(가상 머신) 워크로드에 할당할 수 있습니다. 스왑 사용은 작업자 노드에서만 지원됩니다.

중요

스왑 리소스는 **Burstable** Quality of Service(QoS) 클래스의 VM(가상 머신 워크로드)에만 할당할 수 있습니다. 보장된 QoS 클래스의 VM Pod 및 VM에 속하지 않는 QoS 클래스의 Pod는 리소스를 스왑할 수 없습니다.

QoS 클래스에 대한 설명은 Pod용 서비스 품질 구성 (Kubernetes 문서)을 참조하십시오.

사전 요구 사항

- **oc** 도구를 사용할 수 있습니다.
- cluster-admin 역할을 사용하여 클러스터에 로그인되어 있습니다.
- 메모리 초과 커밋 비율이 정의됩니다.
- 노드는 작업자 풀에 속합니다.

프로세스

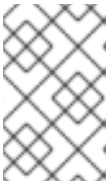
1. 다음 명령을 입력하여 권한 있는 서비스 계정을 생성합니다.

```
$ oc adm new-project wasp
```

```
$ oc create sa -n wasp wasp
```

```
$ oc adm policy add-cluster-role-to-user cluster-admin -n wasp -z wasp
```

```
$ oc adm policy add-scc-to-user -n wasp privileged -z wasp
```



참고

wasp-agent 구성 요소는 OCI 후크를 배포하여 노드 수준에서 컨테이너의 스왑 사용을 활성화합니다. 낮은 수준의 특성을 사용하려면 **DaemonSet** 오브젝트에 권한을 부여해야 합니다.

2. deploy **wasp-agent**:
   a. 다음과 같이 **DaemonSet** 오브젝트를 생성합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: DaemonSet
apiVersion: apps/v1
```

```
metadata:
  name: wasp-agent
  namespace: wasp
  labels:
    app: wasp
    tier: node
spec:
  selector:
    matchLabels:
      name: wasp
  template:
    metadata:
      annotations:
        description: >-
          Configures swap for workloads
      labels:
        name: wasp
    spec:
      serviceAccountName: wasp
      hostPID: true
      hostUsers: true
      terminationGracePeriodSeconds: 5
      containers:
        - name: wasp-agent
          image: >-
            registry.redhat.io/container-native-virtualization/wasp-agent-rhel9:latest
          imagePullPolicy: Always
          env:
          - name: "FSROOT"
            value: "/host"
          resources:
            requests:
              cpu: 100m
              memory: 50M
          securityContext:
            privileged: true
          volumeMounts:
          - name: host
            mountPath: "/host"
      volumes:
      - name: host
        hostPath:
          path: "/"
      priorityClassName: system-node-critical
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 10%
      maxSurge: 0
status: {}
```

b. RH 포털에서 최신 URL을 가져오고 생성된 **DaemonSet** 오브젝트에 URL을 입력합니다.

3. 스왑을 허용하도록 **kubelet** 서비스를 구성합니다.

a. 예제에 표시된 대로 **KubeletConfiguration** 파일을 생성합니다.

KubeletConfiguration 파일 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ''  # MCP
      #machine.openshift.io/cluster-api-machine-role: worker # machine
      #node-role.kubernetes.io/worker: '' # node
  kubeletConfig:
    failSwapOn: false
    evictionSoft:
      memory.available: "1Gi"
    evictionSoftGracePeriod:
      memory.available: "10s"
```

클러스터가 기존 KubeletConfiguration 파일을 이미 사용하고 있는 경우spec 섹션에 다음을 추가합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config
# ...
spec
# ...
    kubeletConfig:
      evictionSoft:
        memory.available: 1Gi
      evictionSoftGracePeriod:
        memory.available: 1m30s
      failSwapOn: false
```

b. 다음 명령을 실행합니다.

```
$ oc wait mcp worker --for condition=Updated=True
```

4. 다음과 같이 MachineConfig 오브젝트를 생성하여 스왑을 프로비저닝합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 90-worker-swap
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
```

```
      - contents: |
          [Unit]
          Description=Provision and enable swap
          ConditionFirstBoot=no

          [Service]
          Type=oneshot
          Environment=SWAP_SIZE_MB=5000
          ExecStart=/bin/sh -c "sudo dd if=/dev/zero of=/var/tmp/swapfile
count=$SWAP_SIZE_MB bs=1MiB && sudo chmod 600 /var/tmp/swapfile && sudo
mkswap /var/tmp/swapfile && sudo swapon /var/tmp/swapfile && free -h"

          [Install]
          RequiredBy=kubelet-dependencies.target
        enabled: true
        name: swap-provision.service
```

best-case 시나리오를 위한 스왑 공간이 충분한 경우 오버 커밋된 RAM으로 프로비저닝된 스왑 공간이 있는지 확인하십시오. 다음 공식을 사용하여 노드에서 프로비저닝할 스왑 공간의 양을 계산합니다.

```
NODE_SWAP_SPACE = NODE_RAM * (MEMORY_OVER_COMMIT_PERCENT / 100% -
1)
```

예제:

```
NODE_SWAP_SPACE = 16 GB * (150% / 100% - 1)
                = 16 GB * (1.5 - 1)
                = 16 GB * (0.5)
                =  8 GB
```

5. 다음과 같이 경고 규칙을 배포합니다.

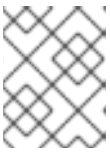
```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: wasp-alerts
  namespace: openshift-monitoring
spec:
  groups:
  - name: wasp.rules
    rules:
    - alert: NodeSwapping
      annotations:
        description: Node {{ $labels.instance }} is swapping at a rate of {{ printf "%.2f"
$value }} MB/s
        runbook_url: https://github.com/openshift-virtualization/wasp-
agent/tree/main/runbooks/alerts/NodeSwapping.md
        summary: A node is swapping memory pages
      expr: |
        # In MB/s
        irate(node_memory_SwapFree_bytes{job="node-exporter"}[5m]) / 1024^2 > 0
      for: 1m
      labels:
        severity: critical
```

6. 다음 예와 같이 OpenShift Container Platform 웹 콘솔을 사용하거나 HyperConverged CR(사용자 정의 리소스) 파일을 편집하여 메모리 과다 할당을 사용하도록 OpenShift Virtualization을 구성합니다.
예제:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  higherWorkloadDensity:
    memoryOvercommitPercentage: 150
```

7. 다음 명령을 입력하여 클러스터의 컴퓨팅 노드에 모든 구성을 적용합니다.

```
$ oc patch --type=merge \
  -f <../manifests/hco-set-memory-overcommit.yaml> \
  --patch-file <../manifests/hco-set-memory-overcommit.yaml>
```



참고

모든 구성을 적용한 후에는 모든 **MachineConfigPool** 롤아웃이 완료된 후에만 스왑 기능을 완전히 사용할 수 있습니다.

검증

1. **wasp-agent** 의 배포를 확인하려면 다음 명령을 실행합니다.

```
$  oc rollout status ds wasp-agent -n wasp
```

배포에 성공하면 다음 메시지가 표시됩니다.

```
daemon set "wasp-agent" successfully rolled out
```

2. 스왑이 올바르게 프로비저닝되었는지 확인하려면 다음을 수행하십시오.
   a. 다음 명령을 실행합니다.

```
$ oc get nodes -l node-role.kubernetes.io/worker
```

   b. 제공된 목록에서 노드를 선택하고 다음 명령을 실행합니다.

```
$ oc debug node/<selected-node> -- free -m
```

스왑이 올바르게 프로비저닝되면 0보다 큰 양이 표시되고 다음과 유사합니다.

	합계	사용됨	무료	공유됨	buff/cache	사용 가능
mEM:	31846	23155	1044	6014	14483	8690

swap:	8191	2337	5854			

3. 다음 명령을 실행하여 OpenShift Virtualization 메모리 과다 할당 구성을 확인합니다.

```
$ oc get -n openshift-cnv HyperConverged kubevirt-hyperconverged -o jsonpath="
{.spec.higherWorkloadDensity.memoryOvercommitPercentage}"
150
```

반환된 값(예: 150)은 이전에 구성한 값과 일치해야 합니다.

7.14. 고급 가상 머신 관리

7.14.1. 가상 머신의 리소스 할당량 작업

가상 시스템의 리소스 할당량을 생성하고 관리합니다.

7.14.1.1. 가상 머신의 리소스 할당량 제한 설정

요청만 사용하는 리소스 할당량은 VM(가상 머신)에서 자동으로 작동합니다. 리소스 할당량에서 제한을 사용하는 경우 VM에 리소스 제한을 수동으로 설정해야 합니다. 리소스 제한은 리소스 요청보다 100MiB 이상이어야 합니다.

프로세스

1. VirtualMachine 매니페스트를 편집하여 VM 제한을 설정합니다. 예를 들면 다음과 같습니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: with-limits
spec:
  running: false
  template:
    spec:
      domain:
# ...
        resources:
          requests:
            memory: 128Mi
          limits:
            memory: 256Mi 1
```

1 limits.memory 값이 requests.memory 값보다 100Mi 이상 크기 때문에 이 구성이 지원됩니다.

2. VirtualMachine 매니페스트를 저장합니다.

7.14.1.2. 추가 리소스

- 프로젝트당 리소스 할당량

- 다중 프로젝트의 리소스 할당량

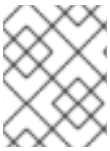
7.14.2. 가상 머신용 노드 지정

노드 배치 규칙을 사용하여 특정 노드에 VM(가상 머신)을 배치할 수 있습니다.

7.14.2.1. 가상 머신의 노드 배치 정보

VM(가상 머신)이 적절한 노드에서 실행되도록 노드 배치 규칙을 구성할 수 있습니다. 다음과 같은 경우 이 작업을 수행할 수 있습니다.

- 여러 개의 VM이 있습니다. 내결함성을 보장하기 위해 서로 다른 노드에서 실행하려고 합니다.
- 두 개의 가상 머신이 있습니다. 중복 노드 간 라우팅을 방지하기 위해 VM을 동일한 노드에서 실행하려고 합니다.
- VM에는 사용 가능한 모든 노드에 존재하지 않는 특정 하드웨어 기능이 필요합니다.
- 노드에 기능을 추가하는 Pod가 있으며 해당 노드에 VM을 배치하여 해당 기능을 사용할 수 있습니다.



참고

가상 머신 배치는 워크로드에 대한 기존 노드 배치 규칙에 의존합니다. 워크로드가 구성 요소 수준의 특정 노드에서 제외되면 해당 노드에 가상 머신을 배치할 수 없습니다.

VirtualMachine 매니페스트의 **spec** 필드에 다음 규칙 유형을 사용할 수 있습니다.

nodeSelector

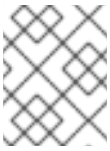
이 필드에서 지정하는 키-값 쌍으로 레이블이 지정된 노드에서 가상 머신을 예약할 수 있습니다. 노드에는 나열된 모든 쌍과 정확히 일치하는 라벨이 있어야 합니다.

유사성

더 많은 표현 구문을 사용하여 노드와 가상 머신의 일치 규칙을 설정할 수 있습니다. 예를 들어, 규칙을 엄격한 요구 사항이 아닌 기본 설정으로 지정할 수 있으므로 규칙이 충족되지 않은 경우에도 가상 머신을 예약할 수 있습니다. 가상 머신 배치에는 Pod 유사성, Pod 비유사성 및 노드 유사성이 지원됩니다. **VirtualMachine** 워크로드 유형이 **Pod** 오브젝트를 기반으로 하므로 Pod 유사성은 가상 머신에서 작동합니다.

허용 오차

일치하는 테인트가 있는 노드에 가상 머신을 예약할 수 있습니다. 테인트가 노드에 적용되는 경우, 해당 노드는 테인트를 허용하는 가상 머신만 허용합니다.



참고

유사성 규칙은 스케줄링 중에만 적용됩니다. 제약 조건이 더 이상 충족되지 않는 경우 OpenShift Container Platform은 실행 중인 워크로드를 다시 예약하지 않습니다.

7.14.2.2. 노드 배치의 예

다음 예시 YAML 파일 조각에서는 **nodePlacement, affinity** 및 **tolerations** 필드를 사용하여 가상 머신의 노드 배치를 사용자 지정합니다.

7.14.2.2.1. 예: nodeSelector를 사용한 VM 노드 배치

이 예에서 가상 시스템에는 **example-key-1 = example-value-1** 및 **example-key-2 = example-value-2** 레이블을 모두 포함하는 메타데이터가 있는 노드가 필요합니다.

> **주의**
>
> 이 설명에 맞는 노드가 없으면 가상 머신이 예약되지 않습니다.

VM 매니페스트 예

```
metadata:
  name: example-vm-node-selector
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      nodeSelector:
        example-key-1: example-value-1
        example-key-2: example-value-2
# ...
```

7.14.2.2.2. 예: Pod 유사성 및 Pod 비유사성을 사용한 VM 노드 배치

이 예에서는 **example-key-1 = example-value-1** 레이블이 있는 실행 중인 pod가 있는 노드에 VM을 예약해야 합니다. 노드에 실행 중인 Pod가 없는 경우 VM은 예약되지 않습니다.

가능한 경우 **example-key-2 = example-value-2** 레이블이 있는 Pod가 있는 노드에 VM이 예약되지 않습니다. 그러나 모든 후보 노드에 이 레이블이 있는 Pod가 있는 경우 스케줄러는 이 제약 조건을 무시합니다.

VM 매니페스트 예

```
metadata:
  name: example-vm-pod-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution: (1)
          - labelSelector:
              matchExpressions:
              - key: example-key-1
                operator: In
                values:
                - example-value-1
            topologyKey: kubernetes.io/hostname
```

```
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution: 2
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: example-key-2
                operator: In
                values:
                - example-value-2
            topologyKey: kubernetes.io/hostname
# ...
```

1 requiredDuringSchedulingIgnoredDuringExecution 규칙 유형을 사용하는 경우 제약 조건이 충족되지 않으면 VM이 예약되지 않습니다.

2 preferredDuringSchedulingIgnoredDuringExecution 규칙 유형을 사용하는 경우 필요한 모든 제약 조건이 충족되면 여전히 VM이 예약됩니다.

7.14.2.2.3. 예: 노드 선호도를 사용한 VM 노드 배치

이 예에서 VM은 **example.io/example-key = example-value-1** 레이블 또는 **example.io/example-key = example-value-2** 레이블이 있는 노드에 예약해야 합니다. 노드에 레이블 중 하나만 있는 경우 제약 조건이 충족됩니다. 레이블이 모두 없으면 VM이 예약되지 않습니다.

가능한 경우 스케줄러는 **example-node-label-key = example-node-label-value** 레이블이 있는 노드를 피합니다. 그러나 모든 후보 노드에 이 레이블이 있으면 스케줄러는 이 제약 조건을 무시합니다.

VM 매니페스트 예

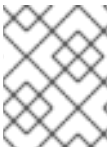
```
metadata:
  name: example-vm-node-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution: 1
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-key
                operator: In
                values:
                - example-value-1
                - example-value-2
          preferredDuringSchedulingIgnoredDuringExecution: 2
          - weight: 1
            preference:
              matchExpressions:
              - key: example-node-label-key
                operator: In
```

```
            values:
            - example-node-label-value
# ...
```

1. **requiredDuringSchedulingIgnoredDuringExecution** 규칙 유형을 사용하는 경우 제약 조건이 충족되지 않으면 VM이 예약되지 않습니다.
2. **preferredDuringSchedulingIgnoredDuringExecution** 규칙 유형을 사용하는 경우 필요한 모든 제약 조건이 충족되면 여전히 VM이 예약됩니다.

7.14.2.2.4. 예: 허용 오차를 사용한 VM 노드 배치

이 예에서는 가상 머신에 예약된 노드가 **key=virtualization:NoSchedule** 테인트로 레이블이 지정됩니다. 이 가상 머신에는 **tolerations**가 일치하므로 테인트된 노드에 예약할 수 있습니다.



참고

해당 테인트가 있는 노드에 스케줄링하는 데 테인트를 허용하는 가상 머신은 필요하지 않습니다.

VM 매니페스트 예

```
metadata:
  name: example-vm-tolerations
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  tolerations:
  - key: "key"
    operator: "Equal"
    value: "virtualization"
    effect: "NoSchedule"
# ...
```

7.14.2.3. 추가 리소스

- 가상화 구성 요소에 대한 노드 지정
- 노드 선택기를 사용하여 특정 노드에 Pod 배치
- 노드 유사성 규칙을 사용하여 노드에 Pod 배치 제어
- 노드 테인트를 사용하여 Pod 배치 제어

7.14.3. 커널 동일 페이지 병합(KSM) 활성화

노드가 과부하될 때 OpenShift Virtualization은 커널 동일 페이지 병합(KSM)을 활성화할 수 있습니다. KSM은 VM(가상 머신)의 메모리 페이지에 있는 동일한 데이터를 중복합니다. VM이 매우 유사한 경우 KSM을 사용하면 단일 노드에서 더 많은 VM을 예약할 수 있습니다.

중요

신뢰할 수 있는 워크로드에서만 KSM을 사용해야 합니다.

7.14.3.1. 사전 요구 사항

- 관리자가 OpenShift Virtualization에서 KSM을 활성화하려는 모든 노드에서 KSM 지원을 구성했는지 확인합니다.

7.14.3.2. OpenShift Virtualization을 사용하여 KSM 활성화 정보

노드에 메모리 과부하가 발생할 때 커널 동일 페이지 병합(KSM)을 활성화하도록 OpenShift Virtualization을 구성할 수 있습니다.

7.14.3.2.1. 구성 방법

OpenShift Container Platform 웹 콘솔을 사용하거나 **HyperConverged** CR(사용자 정의 리소스)을 편집하여 모든 노드에 대해 KSM 활성화 기능을 활성화하거나 비활성화할 수 있습니다. **HyperConverged** CR은 더 세분화된 구성을 지원합니다.

CR 구성
HyperConverged CR의 **spec.configuration.ksmConfiguration** 스탠자를 편집하여 KSM 활성화 기능을 구성할 수 있습니다.

- **ksmConfiguration** 스탠자를 편집하여 기능을 활성화하고 설정을 구성합니다.
- **ksmConfiguration** 스탠자를 삭제하여 기능을 비활성화합니다.
- **ksmConfiguration.nodeLabelSelector** 필드에 노드 선택 구문을 추가하여 노드의 하위 집합에서만 OpenShift Virtualization에서 KSM을 활성화할 수 있습니다.

참고

OpenShift Virtualization에서 KSM 활성화 기능이 비활성화된 경우에도 관리자는 이를 지원하는 노드에서 KSM을 계속 활성화할 수 있습니다.

7.14.3.2.2. KSM 노드 레이블

OpenShift Virtualization은 KSM을 지원하도록 구성된 노드를 식별하고 다음 노드 레이블을 적용합니다.

kubevirt.io/ksm-handler-managed: "false"

OpenShift Virtualization에서 메모리 과부하가 발생하는 노드에서 KSM을 활성화할 때 이 레이블이 **"true"** 로 설정됩니다. 관리자가 KSM을 활성화하는 경우 이 레이블이**"true"** 로 설정되지 않습니다.

kubevirt.io/ksm-enabled: "false"

이 레이블은 OpenShift Virtualization이 KSM을 활성화하지 않은 경우에도 노드에서 KSM을 활성화할 때 **"true"** 로 설정됩니다.

이러한 레이블은 KSM을 지원하지 않는 노드에 적용되지 않습니다.

7.14.3.3. 웹 콘솔을 사용하여 KSM 활성화 구성

OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift Virtualization에서 클러스터의 모든 노드에서 커널 동일 페이지 병합(KSM)을 활성화할 수 있습니다.

절차

1. 사이드 메뉴에서 가상화 → 개요 를 클릭합니다.

2. 설정 탭을 선택합니다.
3. Cluster 탭을 선택합니다.
4. 리소스 관리를 확장합니다.
5. 모든 노드의 기능을 활성화하거나 비활성화합니다.
   - KSM(커널 동일 페이지 병합) 을 on으로 설정합니다.
   - KSM(커널 동일 페이지 병합) 을 off로 설정합니다.

7.14.3.4. CLI를 사용하여 KSM 활성화 구성

HyperConverged CR(사용자 정의 리소스)을 편집하여 OpenShift Virtualization의 커널 동일 페이지 병합(KSM) 활성화 기능을 활성화하거나 비활성화할 수 있습니다. OpenShift Virtualization이 노드의 하위 집합에서만 KSM을 활성화하려면 이 방법을 사용합니다.

절차

1. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged** CR을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **ksmConfiguration** 스탠자를 편집합니다.
   - 모든 노드에 대해 KSM 활성화 기능을 활성화하려면 **nodeLabelSelector** 값을 **{}** 로 설정합니다. 예를 들면 다음과 같습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  configuration:
    ksmConfiguration:
      nodeLabelSelector: {}
# ...
```

   - 노드 하위 집합에서 KSM 활성화 기능을 활성화하려면 **nodeLabelSelector** 필드를 편집합니다. OpenShift Virtualization에서 KSM을 활성화할 노드와 일치하는 구문을 추가합니다. 예를 들어 다음 구성을 사용하면 OpenShift Virtualization에서 **< first_example_key>**와 **< second _example_key >**가 모두 **"true"** 로 설정된 노드에서 KSM을 활성화할 수 있습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  configuration:
    ksmConfiguration:
      nodeLabelSelector:
        matchLabels:
```

```
        <first_example_key>: "true"
        <second_example_key>: "true"
# ...
```

- KSM 활성화 기능을 비활성화하려면 **ksmConfiguration** 스탠자를 삭제합니다. 예를 들면 다음과 같습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  configuration:
# ...
```

3. 파일을 저장합니다.

7.14.3.5. 추가 리소스

- 가상 머신용 노드 지정
- 노드 선택기를 사용하여 특정 노드에 Pod 배치
- RHEL(Red Hat Enterprise Linux) 문서에서 커널 동일한 페이지 병합 관리

7.14.4. 인증서 교체 구성

기존 인증서를 교체하도록 인증서 교체 매개 변수를 구성합니다.

7.14.4.1. 인증서 교체 구성

웹 콘솔에서 또는 **HyperConverged** CR(사용자 정의 리소스)에 설치 후 OpenShift Virtualization을 설치하는 동안 이 작업을 수행할 수 있습니다.

절차

1. 다음 명령을 실행하여 **HyperConverged** CR을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 다음 예와 같이 **spec.certConfig** 필드를 편집합니다. 시스템 과부하를 방지하려면 모든 값이 10분 이상인지 확인합니다. golang **ParseDuration** 형식을 준수하는 문자열로 모든 값을 표현합니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  certConfig:
    ca:
      duration: 48h0m0s
      renewBefore: 24h0m0s (1)
```

```
  server:
    duration: 24h0m0s 2
    renewBefore: 12h0m0s 3
```

1 ca.renewBefore의 값은 ca.duration 값보다 작거나 같아야 합니다.

2 server.duration의 값은 ca.duration 값보다 작거나 같아야 합니다.

3 server.renewBefore의 값은 server.duration 값보다 작거나 같아야 합니다.

3. YAML 파일을 클러스터에 적용합니다.

7.14.4.2. 인증서 교체 매개변수 문제 해결

기본값이 다음 조건 중 하나와 충돌하지 않는 한 하나 이상의 certConfig 값을 삭제하면 기본값으로 되돌아갑니다.

- ca.renewBefore의 값은 ca.duration 값보다 작거나 같아야 합니다.
- server.duration의 값은 ca.duration 값보다 작거나 같아야 합니다.
- server.renewBefore의 값은 server.duration 값보다 작거나 같아야 합니다.

기본값이 이러한 조건과 충돌하면 오류가 발생합니다.

다음 예제에서 server.duration 값을 제거하는 경우 기본값24h0m0s는 ca.duration 값보다 크므로 지정된 조건과 충돌합니다.

예

```
certConfig:
  ca:
    duration: 4h0m0s
    renewBefore: 1h0m0s
  server:
    duration: 4h0m0s
    renewBefore: 4h0m0s
```

이 경우 다음과 같은 오류 메시지가 표시됩니다.

```
error: hyperconvergeds.hco.kubevirt.io "kubevirt-hyperconverged" could not be patched:
admission webhook "validate-hco.kubevirt.io" denied the request: spec.certConfig:
ca.duration is smaller than server.duration
```

오류 메시지는 첫 번째 충돌만 표시합니다. 진행하기 전에 모든 certConfig 값을 검토합니다.

7.14.5. 기본 CPU 모델 구성

HyperConverged CR(사용자 정의 리소스)에서 defaultCPUModel 설정을 사용하여 클러스터 전체 기본 CPU 모델을 정의합니다.

VM(가상 머신) CPU 모델은 VM 및 클러스터 내의 CPU 모델의 가용성에 따라 다릅니다.

- VM에 정의된 CPU 모델이 없는 경우:

- **defaultCPUModel** 은 클러스터 수준에서 정의된 CPU 모델을 사용하여 자동으로 설정됩니다.

- VM과 클러스터의 둘 다 정의된 CPU 모델이 있는 경우:
  - VM의 CPU 모델이 우선합니다.
- VM 및 클러스터에 정의된 CPU 모델이 없는 경우:
  - host-model은 호스트 수준에서 정의된 CPU 모델을 사용하여 자동으로 설정됩니다.

7.14.5.1. 기본 CPU 모델 구성

HyperConverged CR(사용자 정의 리소스)을 업데이트하여 **defaultCPUModel** 을 구성합니다. OpenShift Virtualization이 실행되는 동안 **defaultCPUModel** 을 변경할 수 있습니다.

참고

defaultCPUModel 은 대소문자를 구분합니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.

프로세스

1. 다음 명령을 실행하여 **HyperConverged** CR을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **defaultCPUModel** 필드를 CR에 추가하고 클러스터에 존재하는 CPU 모델의 이름으로 값을 설정합니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
 name: kubevirt-hyperconverged
 namespace: openshift-cnv
spec:
  defaultCPUModel: "EPYC"
```

3. YAML 파일을 클러스터에 적용합니다.

7.14.6. 가상 머신에 UEFI 모드 사용

UEFI(Unified Extensible Firmware Interface) 모드에서 VM(가상 머신)을 부팅할 수 있습니다.

7.14.6.1. 가상 머신의 UEFI 모드 정보

레거시 BIOS와 같이 UEFI(Unified Extensible Firmware Interface)는 컴퓨터가 시작될 때 하드웨어 구성 요소 및 운영 체제 이미지 파일을 초기화합니다. UEFI는 BIOS보다 최신 기능 및 사용자 지정 옵션을 지원하므로 부팅 시간이 단축됩니다.

ESP(EFI System Partition)라는 특수 파티션에 저장된 **.efi** 확장자로 파일에 초기화 및 시작에 대한 모든 정보를 저장합니다. ESP에는 컴퓨터에 설치된 운영 체제용 부트 로더 프로그램도 포함되어 있습니다.

7.14.6.2. UEFI 모드에서 가상 머신 부팅

VirtualMachine 매니페스트를 편집하여 UEFI 모드에서 부팅하도록 가상 머신을 구성할 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. **VirtualMachine** 매니페스트 파일을 편집하거나 생성합니다.**spec.firmware.bootloader** 스탠자를 사용하여 UEFI 모드를 구성합니다.

   보안 부팅이 활성화된 UEFI 모드에서 부팅

```
apiversion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    special: vm-secureboot
  name: vm-secureboot
spec:
  template:
    metadata:
      labels:
        special: vm-secureboot
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: containerdisk
        features:
          acpi: {}
          smm:
            enabled: true (1)
        firmware:
          bootloader:
            efi:
              secureBoot: true (2)
# ...
```

   (1) OpenShift Virtualization에서는 UEFI 모드에서 Secure Boot를 활성화하려면 **SMM**(System Management Mode)을 활성화해야 합니다.

   (2) OpenShift Virtualization은 UEFI 모드를 사용할 때 Secure Boot가 있거나 없는 VM을 지원합니다. Secure Boot가 활성화된 경우 UEFI 모드가 필요합니다. 그러나 Secure Boot를 사용하지 않고 UEFI 모드를 활성화할 수 있습니다.

2. 다음 명령을 실행하여 클러스터에 매니페스트를 적용합니다.

```
$ oc create -f <file_name>.yaml
```

7.14.6.3. 영구 EFI 활성화

클러스터 수준에서 RWX 스토리지 클래스를 구성하고 VM의 EFI 섹션에 있는 설정을 조정하여 VM에서 EFI 지속성을 활성화할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 권한이 있어야합니다.
- RWX 액세스 모드 및 FS 볼륨 모드를 지원하는 스토리지 클래스가 있어야 합니다.

절차

- 다음 명령을 실행하여 **VMPersistentState** 기능 게이트를 활성화합니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op":"replace","path":"/spec/featureGates/VMPersistentState",
"value": true}]'
```

7.14.6.4. 영구 EFI를 사용하여 VM 구성

매니페스트 파일을 편집하여 EFI 지속성을 활성화하도록 VM을 구성할 수 있습니다.

사전 요구 사항

- **VMPersistentState** 기능 게이트가 활성화되었습니다.

절차

- VM 매니페스트 파일을 편집하고 저장하여 설정을 적용합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm
spec:
  template:
    spec:
      domain:
        firmware:
          bootloader:
            efi:
              persistent: true
# ...
```

7.14.7. 가상 머신에 대한 PXE 부팅 구성

OpenShift Virtualization에서는 PXE 부팅 또는 네트워크 부팅을 사용할 수 있습니다. 네트워크 부팅의 경우 로컬로 연결된 스토리지 장치 없이 컴퓨터에서 운영 체제 또는 기타 프로그램을 부팅 및 로드할 수 있습니다. 예를 들어, 새 호스트를 배포할 때 PXE 서버에서 원하는 OS 이미지를 선택할 수 있습니다.

7.14.7.1. 사전 요구 사항

- Linux 브리지가 연결되어 있어야 합니다.

- PXE 서버는 브리지와 동일한 VLAN에 연결되어 있어야 합니다.

7.14.7.2. 지정된 MAC 주소로 PXE 부팅

관리자는 PXE 네트워크에 대한 **NetworkAttachmentDefinition** 오브젝트를 생성한 후 네트워크를 통해 클라이언트를 부팅할 수 있습니다. 그런 다음 가상 머신 인스턴스 구성 파일에서 네트워크 연결 정의를 참조한 후 가상 머신 인스턴스를 시작할 수 있습니다. PXE 서버에 필요한 경우 가상 머신 인스턴스 구성 파일에 MAC 주소를 지정할 수도 있습니다.

사전 요구 사항

- Linux 브리지가 연결되어 있어야 합니다.
- PXE 서버는 브리지와 동일한 VLAN에 연결되어 있어야 합니다.

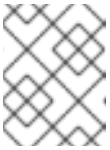
절차

1. 클러스터에서 PXE 네트워크를 구성합니다.
   a. PXE 네트워크 **pxe-net-conf**에 대한 네트워크 연결 정의 파일을 만듭니다.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pxe-net-conf
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "pxe-net-conf",
    "plugins": [
      {
        "type": "cnv-bridge",
        "bridge": "br1",
        "vlan": 1 (1)
      },
      {
        "type": "cnv-tuning" (2)
      }
    ]
  }'
```

(1) 선택 사항: VLAN 태그.

(2) **cnv-tuning** 플러그인은 사용자 지정 MAC 주소를 지원합니다.



참고

VLAN이 요청된 액세스 포트를 통해 가상 머신 인스턴스가 브리지 **br1**에 연결됩니다.

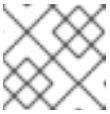
2. 이전 단계에서 만든 파일을 사용하여 네트워크 연결 정의를 생성합니다.

```
$ oc create -f pxe-net-conf.yaml
```

3. 인터페이스 및 네트워크에 대한 세부 정보를 포함하도록 가상 머신 인스턴스 구성 파일을 편집합니다.

   a. PXE 서버에 필요한 경우 네트워크 및 MAC 주소를 지정합니다. MAC 주소를 지정하지 않으면 값이 자동으로 할당됩니다.
   인터페이스가 먼저 부팅되도록 **bootOrder**가 **1**로 설정되어 있는지 확인하십시오. 이 예에서는 인터페이스가 **<pxe-net>**이라는 네트워크에 연결되어 있습니다.

   ```
   interfaces:
   - masquerade: {}
     name: default
   - bridge: {}
     name: pxe-net
     macAddress: de:00:00:00:00:de
     bootOrder: 1
   ```

   참고

   부팅 순서는 인터페이스 및 디스크에 대해 전역적입니다.

   b. 운영 체제가 프로비저닝되면 올바르게 부팅되도록 부팅 장치 번호를 디스크에 할당합니다. 디스크의 **bootOrder** 값을 **2**로 설정합니다.

   ```
   devices:
     disks:
     - disk:
         bus: virtio
       name: containerdisk
       bootOrder: 2
   ```

   c. 네트워크를 이전에 생성한 네트워크 연결 정의에 연결하도록 지정합니다. 이 시나리오에서 **<pxe-net>**는 **<pxe-net-conf>**라는 네트워크 연결 정의에 연결됩니다.

   ```
   networks:
   - name: default
     pod: {}
   - name: pxe-net
     multus:
       networkName: pxe-net-conf
   ```

4. 가상 머신 인스턴스를 생성합니다.

   ```
   $ oc create -f vmi-pxe-boot.yaml
   ```

   출력 예

   ```
   virtualmachineinstance.kubevirt.io "vmi-pxe-boot" created
   ```

5. 가상 머신 인스턴스가 실행될 때까지 기다립니다.

   ```
   $ oc get vmi vmi-pxe-boot -o yaml | grep -i phase
     phase: Running
   ```

6. VNC를 사용하여 가상 머신 인스턴스를 확인합니다.

```
$ virtctl vnc vmi-pxe-boot
```

7. 부팅 화면에서 PXE 부팅에 성공했는지 확인합니다.

8. 가상 머신 인스턴스에 로그인합니다.

```
$ virtctl console vmi-pxe-boot
```

검증

1. 가상 머신의 인터페이스 및 MAC 주소를 확인하고, 브릿지에 연결된 인터페이스에 MAC 주소가 지정되었는지 확인합니다. 이 예제에서는 IP 주소 없이 PXE 부팅에 **eth1**을 사용했습니다. 다른 인터페이스인 **eth0**은 OpenShift Container Platform에서 IP 주소를 가져왔습니다.

```
$ ip addr
```

출력 예

```
...
3. eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default
qlen 1000
  link/ether de:00:00:00:00:de brd ff:ff:ff:ff:ff:ff
```

7.14.7.3. OpenShift Virtualization 네트워킹 용어집

다음 용어는 OpenShift Virtualization 설명서 전체에서 사용됩니다.

CNI(컨테이너 네트워크 인터페이스(Container Network Interface))

컨테이너 네트워크 연결에 중점을 둔 Cloud Native Computing Foundation 프로젝트입니다. OpenShift Virtualization에서는 CNI 플러그인을 사용하여 기본 Kubernetes 네트워킹 기능을 기반으로 빌드합니다.

Multus

Pod 또는 가상 머신에서 필요한 인터페이스를 사용할 수 있도록 여러 CNI가 존재할 수 있는 "메타" CNI 플러그인입니다.

CRD(사용자 정의 리소스 정의(Custom Resource Definition))

사용자 정의 리소스를 정의할 수 있는 Kubernetes API 리소스 또는 CRD API 리소스를 사용하여 정의한 오브젝트입니다.

네트워크 연결 정의(NAD)

Pod, 가상 머신 및 가상 머신 인스턴스를 하나 이상의 네트워크에 연결할 수 있는 Multus 프로젝트에서 도입한 CRD입니다.

노드 네트워크 구성 정책(NNCP)

nmstate 프로젝트에서 도입한 CRD로, 노드에서 요청된 네트워크 구성을 설명합니다. **NodeNetworkConfigurationPolicy** 매니페스트를 클러스터에 적용하는 방식으로 인터페이스 추가 및 제거를 포함하여 노드 네트워크 구성을 업데이트합니다.

7.14.8. 가상 머신에서 대규모 페이지 사용

대규모 페이지를 클러스터의 가상 머신 백업 메모리로 사용할 수 있습니다.

7.14.8.1. 사전 요구 사항

- 노드에 사전 할당된 대규모 페이지가 구성되어 있어야 합니다.

7.14.8.2. 대규모 페이지의 기능

메모리는 페이지라는 블록으로 관리됩니다. 대부분의 시스템에서 한 페이지는 4Ki입니다. 1Mi 메모리는 256페이지와 같고 1Gi 메모리는 256,000페이지에 해당합니다. CPU에는 하드웨어에서 이러한 페이지 목록을 관리하는 내장 메모리 관리 장치가 있습니다. TLB(Translation Lookaside Buffer)는 가상-물리적 페이지 매핑에 대한 소규모 하드웨어 캐시입니다. TLB에 하드웨어 명령어로 전달된 가상 주소가 있으면 매핑을 신속하게 확인할 수 있습니다. 가상 주소가 없으면 TLB 누락이 발생하고 시스템에서 소프트웨어 기반 주소 변환 속도가 느려져 성능 문제가 발생합니다. TLB 크기는 고정되어 있으므로 TLB 누락 가능성을 줄이는 유일한 방법은 페이지 크기를 늘리는 것입니다.

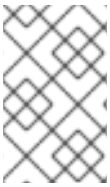
대규모 페이지는 4Ki보다 큰 메모리 페이지입니다. x86_64 아키텍처에서 일반적인 대규모 페이지 크기는 2Mi와 1Gi입니다. 다른 아키텍처에서는 크기가 달라집니다. 대규모 페이지를 사용하려면 애플리케이션이 인식할 수 있도록 코드를 작성해야 합니다. THP(투명한 대규모 페이지)에서는 애플리케이션 지식 없이 대규모 페이지 관리를 자동화하려고 하지만 한계가 있습니다. 특히 페이지 크기 2Mi로 제한됩니다. THP에서는 THP 조각 모음 작업으로 인해 메모리 사용률이 높아지거나 조각화가 발생하여 노드에서 성능이 저하될 수 있으며 이로 인해 메모리 페이지가 잠길 수 있습니다. 이러한 이유로 일부 애플리케이션은 THP 대신 사전 할당된 대규모 페이지를 사용하도록 설계(또는 권장)할 수 있습니다.

OpenShift Virtualization에서는 사전 할당된 대규모 페이지를 사용하도록 가상 머신을 구성할 수 있습니다.

7.14.8.3. 가상 머신용 대규모 페이지 구성

가상 머신 구성에 **memory.hugepages.pageSize** 및 **resources.requests.memory** 매개변수를 포함하여 사전 할당된 대규모 페이지를 사용하도록 가상 머신을 구성할 수 있습니다.

메모리 요청은 페이지 크기로 나눌 수 있어야합니다. 예를 들면 페이지 크기가 **1Gi**인 **500Mi**의 메모리는 요청할 수 없습니다.



참고

호스트와 게스트 OS의 메모리 레이아웃은 관련이 없습니다. 가상 머신 매니페스트에서 요청된 대규모 페이지가 QEMU에 적용됩니다. 게스트 내부의 대규모 페이지는 사용 가능한 가상 머신 인스턴스 메모리 양을 기준으로만 구성할 수 있습니다.

실행 중인 가상 머신을 편집하는 경우 변경 사항을 적용하려면 가상 머신을 재부팅해야 합니다.

사전 요구 사항

- 노드에 사전 할당된 대규모 페이지가 구성되어 있어야 합니다.

절차

1. 가상 머신 구성에서 **spec.domain**에 **resources.requests.memory** 및 **memory.hugepages.pageSize** 매개변수를 추가합니다. 다음 구성 스니펫에서는 가상 머신에서 각 페이지 크기가 **1Gi**인 총 **4Gi**의 메모리를 요청합니다.

```
kind: VirtualMachine
# ...
spec:
  domain:
```

```
    resources:
      requests:
        memory: "4Gi" (1)
    memory:
      hugepages:
        pageSize: "1Gi" (2)
# ...
```

(1) 가상 머신에 대해 요청된 총 메모리 양입니다. 이 값은 페이지 크기로 나눌 수 있어야 합니다.

(2) 각 대규모 페이지의 크기입니다. x86_64 아키텍처에 유효한 값은 **1Gi** 및 **2Mi**입니다. 페이지 크기는 요청된 메모리보다 작아야 합니다.

2. 가상 머신 구성을 적용합니다.

```
$ oc apply -f <virtual_machine>.yaml
```

7.14.9. 가상 머신 전용 리소스 사용

성능 향상을 위해 CPU와 같은 노드 리소스를 가상 머신에 전용으로 지정할 수 있습니다.

7.14.9.1. 전용 리소스 정보

가상 머신에 전용 리소스를 사용하면 가상 머신의 워크로드가 다른 프로세스에서 사용하지 않는 CPU에 예약됩니다. 전용 리소스를 사용하면 가상 머신의 성능과 대기 시간 예측 정확도를 개선할 수 있습니다.

7.14.9.2. 사전 요구 사항

- 노드에 CPU 관리자를 구성해야 합니다. 가상 머신 워크로드를 예약하기 전에 노드에**cpumanager = true** 라벨이 있는지 확인하십시오.
- 가상 머신의 전원을 꺼야 합니다.

7.14.9.3. 가상 머신 전용 리소스 활성화

세부 정보 탭에서 가상 머신 전용 리소스를 활성화합니다. Red Hat 템플릿에서 생성된 가상 머신은 전용 리소스로 구성할 수 있습니다.

프로세스

1. OpenShift Container Platform 콘솔의 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 가상 머신을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 구성 → 스케줄링 탭에서 전용 리소스 옆에 있는 편집 아이콘을 클릭합니다.
4. 전용 리소스(보장된 정책)를 사용하여 이 워크로드 예약을 선택합니다.
5. 저장을 클릭합니다.

7.14.10. 가상 머신 예약

호환성을 위해 VM의 CPU 모델과 정책 특성이 노드에서 지원하는 CPU 모델 및 정책 특성과 일치하도록 하면 노드에 VM(가상 머신)을 예약할 수 있습니다.

7.14.10.1. 정책 특성

VM(가상 머신)을 노드에 예약할 때 호환성을 위해 일치하는 정책 특성과 CPU 기능을 지정하면 VM을 예약할 수 있습니다. VM에 지정되는 정책 특성에 따라 VM이 노드에서 예약되는 방식이 결정됩니다.

정책 특성	설명
force	VM이 노드에 강제로 예약됩니다. 호스트 CPU에서 VM CPU를 지원하지 않는 경우에도 마찬가지입니다.
require	VM이 특정 CPU 모델 및 기능 사양으로 구성되지 않은 경우 VM에 적용되는 기본 정책입니다. 이 기본 정책 특성 또는 다른 정책 특성 중 하나를 사용하여 CPU 노드 검색을 지원하도록 노드를 구성하지 않으면 해당 노드에 VM이 예약되지 않습니다. 호스트 CPU가 VM의 CPU를 지원하거나 하이퍼바이저가 지원되는 CPU 모델을 에뮬레이션할 수 있어야 합니다.
optional	호스트의 물리적 머신 CPU에서 VM을 지원하는 경우 해당 VM이 노드에 추가됩니다.
disable	CPU 노드 검색을 통해 VM을 예약할 수 없습니다.
forbid	호스트 CPU에서 기능을 지원하고 CPU 노드 검색을 사용할 수 있는 경우에도 VM을 예약할 수 없습니다.

7.14.10.2. 정책 특성 및 CPU 기능 설정

각 VM(가상 머신)에 대한 정책 특성 및 CPU 기능을 설정하면 정책 및 기능에 따라 노드에 VM을 예약할 수 있습니다. 설정한 CPU 기능은 호스트 CPU의 지원 여부 또는 하이퍼바이저의 에뮬레이션 여부를 확인하기 위해 검증됩니다.

절차

- **VM 구성 파일의 domain 사양을 편집합니다. 다음 예제에서는 VM(가상 머신)에 대한 CPU 기능 및 require 정책을 설정합니다.**

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          features:
            - name: apic (1)
              policy: require (2)
```

1. VM의 CPU 기능 이름입니다.
2. VM의 정책 특성입니다.

7.14.10.3. 지원되는 CPU 모델을 사용하여 가상 머신 예약

VM(가상 머신)의 CPU 모델을 구성하여 해당 CPU 모델이 지원되는 노드에 예약할 수 있습니다.

절차

- 가상 머신 구성 파일의 **domain** 사양을 편집합니다. 다음 예는 VM에 대해 정의된 특정 CPU 모델을 보여줍니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: Conroe 1
```

1. VM의 CPU 모델입니다.

7.14.10.4. 호스트 모델을 사용하여 가상 머신 예약

VM(가상 머신)의 CPU 모델이 **host-model**로 설정되어 있으면 VM은 예약된 노드의 CPU 모델을 상속합니다.

절차

- VM 구성 파일의 **domain** 사양을 편집합니다. 다음 예제에서는 가상 머신에 지정된**host-model**을 보여줍니다.

```
apiVersion: kubevirt/v1alpha3
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: host-model 1
```

1. 예약된 노드의 CPU 모델을 상속하는 VM입니다.

7.14.10.5. 사용자 정의 스케줄러를 사용하여 가상 머신 예약

사용자 정의 스케줄러를 사용하여 노드에 VM(가상 머신)을 예약할 수 있습니다.

사전 요구 사항

- 보조 스케줄러가 클러스터에 대해 구성되어 있습니다.

프로세스

- **VirtualMachine** 매니페스트를 편집하여 사용자 지정 스케줄러를 VM 구성에 추가합니다. 예를 들면 다음과 같습니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
spec:
  running: true
  template:
    spec:
      schedulerName: my-scheduler 1
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
# ...
```

1 사용자 정의 스케줄러의 이름입니다. **schedulerName** 값이 기존 스케줄러와 일치하지 않으면 **virt-launcher** Pod는 지정된 스케줄러를 찾을 때까지 **Pending** 상태로 유지됩니다.

검증

- **virt-launcher** Pod 이벤트를 확인하여 VM이 **VirtualMachine** 매니페스트에 지정된 사용자 정의 스케줄러를 사용하고 있는지 확인합니다.

  a. 다음 명령을 입력하여 클러스터의 Pod 목록을 확인합니다.

  ```
  $ oc get pods
  ```

  출력 예

  ```
  NAME                            READY   STATUS    RESTARTS   AGE
  virt-launcher-vm-fedora-dpc87   2/2     Running   0          24m
  ```

  b. 다음 명령을 실행하여 Pod 이벤트를 표시합니다.

  ```
  $ oc describe pod virt-launcher-vm-fedora-dpc87
  ```

  출력의 **From** 필드 값은 스케줄러 이름이 **VirtualMachine** 매니페스트에 지정된 사용자 정의 스케줄러와 일치하는지 확인합니다.

  출력 예

```
[...]
Events:
  Type    Reason     Age   From            Message
  ----    ------     ----  ----            -------
  Normal  Scheduled  21m   my-scheduler  Successfully assigned default/virt-
launcher-vm-fedora-dpc87 to node01
[...]
```

추가 리소스

- 보조 스케줄러 배포

7.14.11. PCI 패스스루 구성

PCI(Peripheral Component Interconnect) 패스스루 기능을 사용하면 VM(가상 머신)에서 하드웨어 장치에 액세스하고 관리할 수 있습니다. PCI 패스스루가 구성되면 PCI 장치는 게스트 운영 체제에 물리적으로 연결된 것처럼 작동합니다.

클러스터 관리자는 oc CLI(명령줄 인터페이스)를 사용하여 클러스터에서 사용할 수 있는 호스트 장치를 노출하고 관리할 수 있습니다.

7.14.11.1. GPU 패스스루를 위한 노드 준비

GPU 피연산자가 GPU 패스스루용으로 지정한 작업자 노드에 배포되지 않도록 할 수 있습니다.

7.14.11.1.1. NVIDIA GPU 피연산자가 노드에 배포되지 않도록 방지

클러스터에서 NVIDIA GPU Operator 를 사용하는 경우nvidia.com/gpu.deploy.operands=false 레이블을 GPU 또는 vGPU 피연산자에 대해 구성하지 않으려는 노드에 적용할 수 있습니다. 이 레이블은 GPU 또는 vGPU 피연산자를 구성하는 Pod 생성을 방지하고 이미 존재하는 경우 Pod를 종료합니다.

사전 요구 사항

- OpenShift CLI(oc)가 설치되어 있어야 합니다.

프로세스

- 다음 명령을 실행하여 노드에 레이블을 지정합니다.

```
$ oc label node <node_name> nvidia.com/gpu.deploy.operands=false 1
```

1 & lt;node_name >을 NVIDIA GPU 피연산자를 설치하지 않으려는 노드의 이름으로 바꿉니다.

검증

1. 다음 명령을 실행하여 라벨이 노드에 추가되었는지 확인합니다.

```
$ oc describe node <node_name>
```

2. 선택 사항: GPU 피연산자가 이전에 노드에 배포된 경우 제거를 확인합니다.

a. 다음 명령을 실행하여 **nvidia-gpu-operator** 네임스페이스에서 Pod의 상태를 확인합니다.

```
$ oc get pods -n nvidia-gpu-operator
```

출력 예

```
NAME                          READY   STATUS       RESTARTS   AGE
gpu-operator-59469b8c5c-hw9wj   1/1     Running      0          8d
nvidia-sandbox-validator-7hx98  1/1     Running      0          8d
nvidia-sandbox-validator-hdb7p  1/1     Running      0          8d
nvidia-sandbox-validator-kxwj7  1/1     Terminating  0          9d
nvidia-vfio-manager-7w9fs       1/1     Running      0          8d
nvidia-vfio-manager-866pz       1/1     Running      0          8d
nvidia-vfio-manager-zqtck       1/1     Terminating  0          9d
```

b. **Terminating** 상태의 Pod가 제거될 때까지 Pod 상태를 모니터링합니다.

```
$ oc get pods -n nvidia-gpu-operator
```

출력 예

```
NAME                          READY   STATUS    RESTARTS   AGE
gpu-operator-59469b8c5c-hw9wj   1/1     Running   0          8d
nvidia-sandbox-validator-7hx98  1/1     Running   0          8d
nvidia-sandbox-validator-hdb7p  1/1     Running   0          8d
nvidia-vfio-manager-7w9fs       1/1     Running   0          8d
nvidia-vfio-manager-866pz       1/1     Running   0          8d
```

7.14.11.2. PCI 패스스루를 위한 호스트 장치 준비

7.14.11.2.1. PCI 패스스루를 위한 호스트 장치 준비 정보

CLI를 사용하여 PCI 패스스루를 위한 호스트 장치를 준비하려면 **MachineConfig** 오브젝트를 생성하고 커널 인수를 추가하여 IOMMU(Input-Output Memory Management Unit)를 활성화합니다. PCI 장치를 VFIO(가상 기능 I/O) 드라이버에 연결한 다음 **HyperConverged** CR(사용자 정의 리소스)의 **allowedHostDevices** 필드를 편집하여 클러스터에 노출합니다. OpenShift Virtualization Operator를 처음 설치할 때 **permittedHostDevices** 목록이 비어 있습니다.

CLI를 사용하여 클러스터에서 PCI 호스트 장치를 제거하려면 **HyperConverged** CR에서 PCI 장치 정보를 삭제합니다.

7.14.11.2.2. IOMMU 드라이버를 활성화하려면 커널 인수 추가

커널에서 IOMMU 드라이버를 활성화하려면 **MachineConfig** 오브젝트를 생성하고 커널 인수를 추가합니다.

사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.
- CPU 하드웨어는 Intel 또는 AMD입니다.

- BIOS에서 Directed I/O 확장 또는 AMD IOMMU에 대한 Intel Virtualization Technology를 활성화했습니다.

프로세스

1. 커널 인수를 식별하는 **MachineConfig** 오브젝트를 만듭니다. 다음 예제에서는 Intel CPU에 대한 커널 인수를 보여줍니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker (1)
  name: 100-worker-iommu (2)
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
      - intel_iommu=on (3)
# ...
```

(1) 새 커널 인수를 작업자 노드에만 적용합니다.

(2) **name**은 머신 구성 및 그 용도 중 이 커널 인수의 순위(100)를 나타냅니다. AMD CPU가 있는 경우 커널 인수를 **amd_iommu=on**으로 지정합니다.

(3) Intel CPU의 커널 인수를 **intel_iommu**로 식별합니다.

2. 새 **MachineConfig** 오브젝트를 만듭니다.

```
$ oc create -f 100-worker-kernel-arg-iommu.yaml
```

검증

- 새 **MachineConfig** 오브젝트가 추가되었는지 확인합니다.

```
$ oc get MachineConfig
```

7.14.11.2.3. VFIO 드라이버에 PCI 장치 바인딩

PCI 장치를 VFIO(Virtual Function I/O) 드라이버에 바인딩하려면 각 장치에서 **vendor-ID** 및 **device-ID** 값을 가져오고 값으로 목록을 생성합니다. **MachineConfig** 오브젝트에 이 목록을 추가합니다. **MachineConfig** Operator는 PCI 장치가 있는 노드에 **/etc/modprobe.d/vfio.conf**를 생성하고 PCI 장치를 VFIO 드라이버에 바인딩합니다.

사전 요구 사항

- CPU에 IOMMU를 사용하도록 커널 인수를 추가했습니다.

절차

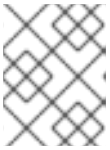
1. **lspci** 명령을 실행하여 PCI 장치의 **vendor-ID** 및 **device-ID**를 가져옵니다.

```
$ lspci -nnv | grep -i nvidia
```

출력 예

```
02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB]
[10de:1eb8] (rev a1)
```

2. Virtual config 파일 **100-worker-vfiopci.bu**를 생성하여 PCI 장치를 VFIO 드라이버에 바인딩합니다.



참고

Butane에 대한 자세한 내용은 "Butane 을 사용하여 머신 구성 생성"을 참조하십시오.

예

```
variant: openshift
version: 4.16.0
metadata:
  name: 100-worker-vfiopci
  labels:
    machineconfiguration.openshift.io/role: worker (1)
storage:
  files:
  - path: /etc/modprobe.d/vfio.conf
    mode: 0644
    overwrite: true
    contents:
      inline: |
        options vfio-pci ids=10de:1eb8 (2)
  - path: /etc/modules-load.d/vfio-pci.conf (3)
    mode: 0644
    overwrite: true
    contents:
      inline: vfio-pci
```

(1) 새 커널 인수를 작업자 노드에만 적용합니다.

(2) 단일 장치를 VFIO 드라이버에 바인딩하려면 이전에 결정한 **vendor-ID** 값 (10de) 및 **device-ID** 값 (1eb8) 을 지정합니다. 공급업체 및 장치 정보를 사용하여 여러 장치 목록을 추가할 수 있습니다.

(3) 작업자 노드에서 vfio-pci 커널 모듈을 로드하는 파일입니다.

3. Butane을 사용하여 작업자 노드로 전달할 구성이 포함된 **MachineConfig** 오브젝트 파일 **100-worker-vfiopci.yaml**을 생성합니다.

```
$ butane 100-worker-vfiopci.bu -o 100-worker-vfiopci.yaml
```

4. 작업자 노드에 **MachineConfig** 오브젝트를 적용합니다.

```
$ oc apply -f 100-worker-vfiopci.yaml
```

5. **MachineConfig** 오브젝트가 추가되었는지 확인합니다.

```
$ oc get MachineConfig
```

출력 예

```
NAME                             GENERATEDBYCONTROLLER                      IGNITIONVERSION
AGE
00-master                        d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0          25h
00-worker                        d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0          25h
01-master-container-runtime      d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-master-kubelet                d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-worker-container-runtime      d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-worker-kubelet                d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
100-worker-iommu                                                            3.2.0          30s
100-worker-vfiopci-configuration                                            3.2.0          30s
```

검증

- VFIO 드라이버가 로드되었는지 확인합니다.

```
$ lspci -nnk -d 10de:
```

출력은 VFIO 드라이버가 사용 중인지 확인합니다.

출력 예

```
04:00.0 3D controller [0302]: NVIDIA Corporation GP102GL [Tesla P40] [10de:1eb8] (rev a1)
        Subsystem: NVIDIA Corporation Device [10de:1eb8]
        Kernel driver in use: vfio-pci
        Kernel modules: nouveau
```

7.14.11.2.4. CLI를 사용하여 클러스터에 PCI 호스트 장치 노출

클러스터에 PCI 호스트 장치를 노출하려면 PCI 장치에 대한 세부 정보를 **HyperConverged** CR(사용자 정의 리소스)의 **spec.permittedHostDevices** 배열에 추가합니다.

절차

1. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged** CR을 편집합니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **spec.permittedHostDevices.pciHostDevices** 어레이에 PCI 장치 정보를 추가합니다. 예를 들면 다음과 같습니다.

설정 파일 예

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  permittedHostDevices: 1
    pciHostDevices: 2
    - pciDeviceSelector: "10DE:1DB6" 3
      resourceName: "nvidia.com/GV100GL_Tesla_V100" 4
    - pciDeviceSelector: "10DE:1EB8"
      resourceName: "nvidia.com/TU104GL_Tesla_T4"
    - pciDeviceSelector: "8086:6F54"
      resourceName: "intel.com/qat"
      externalResourceProvider: true 5
# ...
```

1. 클러스터에서 사용할 수 있는 호스트 장치입니다.
2. 노드에서 사용할 수 있는 PCI 장치 목록입니다.
3. **vendor-ID** 및 **device-ID**는 PCI 장치를 식별해야 합니다.
4. PCI 호스트 장치의 이름입니다.
5. 선택 사항: 이 필드를 **true** 로 설정하면 리소스가 외부 장치 플러그인에서 제공됨을 나타냅니다. OpenShift Virtualization에서는 클러스터에서 이 장치를 사용할 수 있지만 할당 및 모니터링을 외부 장치 플러그인으로 남겨 둡니다.

참고

위의 예제 스니펫은 이름이 **nvidia.com/GV100GL_Tesla_V100**이고 **nvidia.com/TU104GL_Tesla_T4**가 **HyperConverged** CR에서 허용된 호스트 장치 목록에 추가된 두 개의 PCI 호스트 장치를 보여줍니다. 이러한 장치는 OpenShift Virtualization에서 작동하도록 테스트 및 검증되었습니다.

3. 변경 사항을 저장하고 편집기를 종료합니다.

검증

- 다음 명령을 실행하여 PCI 호스트 장치가 노드에 추가되었는지 확인합니다. 예제 출력에서는 각각**nvidia.com/GV100GL_Tesla_V100, nvidia.com/TU104GL_Tesla_T4, intel.com/qat** 리소스 이름과 연결된 하나의 장치가 있음을 보여줍니다.

```
$ oc describe node <node_name>
```

출력 예

```
Capacity:
  cpu:                64
```

```
  devices.kubevirt.io/kvm:       110
  devices.kubevirt.io/tun:       110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:             915128Mi
  hugepages-1Gi:                 0
  hugepages-2Mi:                 0
  memory:                        131395264Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                 1
  pods:                          250
Allocatable:
  cpu:                           63500m
  devices.kubevirt.io/kvm:       110
  devices.kubevirt.io/tun:       110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:             863623130526
  hugepages-1Gi:                 0
  hugepages-2Mi:                 0
  memory:                        130244288Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                 1
  pods:                          250
```

7.14.11.2.5. CLI를 사용하여 클러스터에서 PCI 호스트 장치 제거

클러스터에서 PCI 호스트 장치를 제거하려면 **HyperConverged** CR(사용자 정의 리소스)에서 해당 장치의 정보를 삭제합니다.

절차

1. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged** CR을 편집합니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 적절한 장치의 **pciDeviceSelector**,**resourceName** 및 **externalResourceProvider** (해당되는 경우) 필드를 삭제하여 **spec.permittedHostDevices.pciHostDevices** 어레이에서 PCI 장치 정보를 제거합니다. 이 예에서는 **intel.com/qat** 리소스가 삭제되었습니다.

   설정 파일 예

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  permittedHostDevices:
    pciHostDevices:
    - pciDeviceSelector: "10DE:1DB6"
      resourceName: "nvidia.com/GV100GL_Tesla_V100"
    - pciDeviceSelector: "10DE:1EB8"
      resourceName: "nvidia.com/TU104GL_Tesla_T4"
  # ...
```

-

3. 변경 사항을 저장하고 편집기를 종료합니다.

검증

- 다음 명령을 실행하여 PCI 호스트 장치가 노드에서 제거되었는지 확인합니다. 예제 출력에서는 **intel.com/qat** 리소스 이름과 연결된 장치가 0개 있음을 보여줍니다.

```
$ oc describe node <node_name>
```

출력 예

```
Capacity:
  cpu:                    64
  devices.kubevirt.io/kvm:       110
  devices.kubevirt.io/tun:       110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            915128Mi
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                   131395264Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                0
  pods:                     250
Allocatable:
  cpu:                    63500m
  devices.kubevirt.io/kvm:       110
  devices.kubevirt.io/tun:       110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            863623130526
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                   130244288Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                0
  pods:                     250
```

7.14.11.3. PCI 패스스루의 가상 머신 구성

PCI 장치를 클러스터에 추가하고 나면 가상 머신에 할당할 수 있습니다. 이제 PCI 장치를 가상 머신에 물리적으로 연결된 것처럼 사용할 수 있습니다.

7.14.11.3.1. 가상 머신에 PCI 장치 할당

PCI 장치를 클러스터에서 사용할 수 있는 경우 가상 머신에 할당하고 PCI 패스스루를 활성화할 수 있습니다.

절차

- 가상 시스템에 PCI 장치를 호스트 장치로 할당합니다.

  예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  domain:
    devices:
      hostDevices:
      - deviceName: nvidia.com/TU104GL_Tesla_T4 1
        name: hostdevices1
```

1 호스트 장치로 클러스터에서 허용되는 PCI 장치의 이름입니다. 가상 시스템은 이 호스트 장치에 액세스할 수 있습니다.

검증

- 다음 명령을 사용하여 가상 시스템에서 호스트 장치를 사용할 수 있는지 확인합니다.

```
$ lspci -nnk | grep NVIDIA
```

출력 예

```
$ 02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB]
[10de:1eb8] (rev a1)
```

7.14.11.4. 추가 리소스

- BIOS에서 Intel VT-X 및 AMD-V 가상화 하드웨어 확장 활성화
- 파일 권한 관리
- 머신 구성 개요

7.14.12. 가상 GPU 구성

GPU(그래픽 처리 장치) 카드가 있는 경우 OpenShift Virtualization은 VM(가상 머신)에 할당할 수 있는 가상 GPU(vGPU)를 자동으로 생성할 수 있습니다.

7.14.12.1. OpenShift Virtualization에서 가상 GPU 사용 정보

일부 GPU(그래픽 처리 장치) 카드에서는 vGPU(가상 GPU) 생성을 지원합니다. 관리자가 **HyperConverged** CR(사용자 정의 리소스)에 구성 세부 정보를 제공하는 경우 OpenShift Virtualization은 vGPU 및 기타 중재 장치를 자동으로 생성할 수 있습니다. 이 자동화는 대규모 클러스터에 특히 유용합니다.

참고

기능 및 지원 세부 사항은 하드웨어 벤더의 설명서를 참조하십시오.

중재된 장치

하나 이상의 가상 장치로 구성된 물리적 장치입니다. vGPU는 미디어 장치(mdev)의 유형입니다. 물리적 GPU의 성능은 가상 장치로 나뉩니다. 하나 이상의 VM(가상 머신)에 중재된 장치를 할당할 수 있지만 게스트 수는 GPU와 호환되어야 합니다. 일부 GPU는 여러 게스트를 지원하지 않습니다.

7.14.12.2. 중재된 장치를 위한 호스트 준비

중재된 장치를 구성하려면IOMMU(Input-Output Memory Management Unit) 드라이버를 활성화해야 합니다.

7.14.12.2.1. IOMMU 드라이버를 활성화하려면 커널 인수 추가

커널에서 IOMMU 드라이버를 활성화하려면 **MachineConfig** 오브젝트를 생성하고 커널 인수를 추가합니다.

사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.
- CPU 하드웨어는 Intel 또는 AMD입니다.
- BIOS에서 Directed I/O 확장 또는 AMD IOMMU에 대한 Intel Virtualization Technology를 활성화했습니다.

프로세스

1. 커널 인수를 식별하는 **MachineConfig** 오브젝트를 만듭니다. 다음 예제에서는 Intel CPU에 대한 커널 인수를 보여줍니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker (1)
  name: 100-worker-iommu (2)
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
      - intel_iommu=on (3)
# ...
```

(1) 새 커널 인수를 작업자 노드에만 적용합니다.

(2) **name**은 머신 구성 및 그 용도 중 이 커널 인수의 순위(100)를 나타냅니다. AMD CPU가 있는 경우 커널 인수를 **amd_iommu=on**으로 지정합니다.

(3) Intel CPU의 커널 인수를 **intel_iommu**로 식별합니다.

2. 새 **MachineConfig** 오브젝트를 만듭니다.

```
$ oc create -f 100-worker-kernel-arg-iommu.yaml
```

검증

- 새 **MachineConfig** 오브젝트가 추가되었는지 확인합니다.

```
$ oc get MachineConfig
```

7.14.12.3. NVIDIA GPU Operator 구성

NVIDIA GPU Operator를 사용하여 OpenShift Virtualization에서 GPU 가속 VM(가상 머신)을 실행하기 위한 작업자 노드를 프로비저닝할 수 있습니다.

참고

NVIDIA GPU Operator는 NVIDIA에서만 지원됩니다. 자세한 내용은 Red Hat 지식베이스의 NVIDIA에서 지원 받기를 참조하십시오.

7.14.12.3.1. NVIDIA GPU Operator 사용 정보

NVIDIA GPU Operator를 OpenShift Virtualization과 함께 사용하여 GPU 지원 가상 머신(VM)을 실행하기 위해 작업자 노드를 신속하게 프로비저닝할 수 있습니다. NVIDIA GPU Operator는 OpenShift Container Platform 클러스터에서 NVIDIA GPU 리소스를 관리하고 GPU 워크로드를 위한 노드를 준비할 때 필요한 작업을 자동화합니다.

애플리케이션 워크로드를 GPU 리소스에 배포하려면 먼저 컴퓨팅 통합 장치 아키텍처(CUDA), Kubernetes 장치 플러그인, 컨테이너 런타임 및 자동 노드 레이블 지정 및 모니터링과 같은 기타 기능을 활성화하는 NVIDIA 드라이버와 같은 구성 요소를 설치해야 합니다. 이러한 작업을 자동화하면 인프라의 GPU 용량을 빠르게 확장할 수 있습니다. NVIDIA GPU Operator는 특히 복잡한 인공 지능 및 머신 러닝(AI/ML) 워크로드를 쉽게 프로비저닝할 수 있습니다.

7.14.12.3.2. 중재된 장치를 구성하는 옵션

NVIDIA GPU Operator를 사용할 때 중재된 장치를 구성하는 데 사용할 수 있는 두 가지 방법이 있습니다. Red Hat 테스트 방법은 OpenShift Virtualization 기능을 사용하여 중재된 장치를 예약하는 반면 NVIDIA 방법은 GPU Operator만 사용합니다.

NVIDIA GPU Operator를 사용하여 중재된 장치 구성

이 방법은 NVIDIA GPU Operator만 사용하여 중재된 장치를 구성합니다. 이 방법을 사용하려면 NVIDIA 문서의 OpenShift Virtualization과 함께 NVIDIA GPU Operator를 참조하십시오.

OpenShift Virtualization을 사용하여 중재된 장치 구성

Red Hat에서 테스트하는 이 방법은 OpenShift Virtualization의 기능을 사용하여 중재된 장치를 구성합니다. 이 경우 NVIDIA GPU Operator는 NVIDIA vGPU Manager를 사용하여 드라이버를 설치하는 데만 사용됩니다. GPU Operator는 중재된 장치를 구성하지 않습니다.
OpenShift Virtualization 방법을 사용하는 경우 NVIDIA 문서에 따라 GPU Operator를 계속 구성합니다. 그러나 이 방법은 다음과 같은 방법으로 NVIDIA 문서와 다릅니다.

- **HyperConverged** CR(사용자 정의 리소스)에서 기본 **disableMDEVConfiguration: false** 설정을 덮어쓰지 않아야 합니다.

  중요

  NVIDIA 문서에 설명된 대로 이 기능 게이트를 설정하면 OpenShift Virtualization이 중재된 장치를 구성할 수 없습니다.

- 다음 예와 일치하도록 **ClusterPolicy** 매니페스트를 구성해야 합니다.

  매니페스트 예

  ■

```
kind: ClusterPolicy
apiVersion: nvidia.com/v1
metadata:
  name: gpu-cluster-policy
spec:
  operator:
    defaultRuntime: crio
    use_ocp_driver_toolkit: true
    initContainer: {}
  sandboxWorkloads:
    enabled: true
    defaultWorkload: vm-vgpu
  driver:
    enabled: false 1
  dcgmExporter: {}
  dcgm:
    enabled: true
  daemonsets: {}
  devicePlugin: {}
  gfd: {}
  migManager:
    enabled: true
  nodeStatusExporter:
    enabled: true
  mig:
    strategy: single
  toolkit:
    enabled: true
  validator:
    plugin:
      env:
        - name: WITH_WORKLOAD
          value: "true"
  vgpuManager:
    enabled: true 2
    repository: <vgpu_container_registry> 3
    image: <vgpu_image_name>
    version: nvidia-vgpu-manager
  vgpuDeviceManager:
    enabled: false 4
    config:
      name: vgpu-devices-config
      default: default
  sandboxDevicePlugin:
    enabled: false 5
  vfioManager:
    enabled: false 6
```

1. 이 값을 **false** 로 설정합니다. VM에는 필요하지 않습니다.
2. 이 값을 **true** 로 설정합니다. VM과 함께 vGPU를 사용하는 데 필요합니다.
3. & lt;vgpu_container_registry>를 레지스트리 값으로 바꿉니다.
4.

OpenShift Virtualization에서 NVIDIA GPU Operator 대신 중재된 장치를 구성할 수 있도록 하려면 이 값을 **false** 로 설정합니다.

5 vGPU 장치를 kubelet으로 검색 및 알리지 않으려면 이 값을 **false** 로 설정합니다.

6 **vfio-pci** 드라이버를 로드하지 않으려면 이 값을**false** 로 설정합니다. 대신 OpenShift Virtualization 설명서에 따라 PCI 패스스루를 구성합니다.

추가 리소스

- PCI 패스스루 구성

7.14.12.4. vGPU가 노드에 할당되는 방법

각 물리적 장치에 대해 OpenShift Virtualization은 다음 값을 구성합니다.

- 단일 mdev 유형입니다.
- 선택한 **mdev** 유형의 최대 인스턴스 수입니다.

클러스터 아키텍처는 장치를 생성하고 노드에 할당하는 방법에 영향을 미칩니다.

노드당 여러 카드가 있는 대규모 클러스터

유사한 vGPU 유형을 지원할 수 있는 여러 카드가 있는 노드에서는 관련 장치 유형이 라운드 로빈 방식으로 생성됩니다. 예를 들면 다음과 같습니다.

```
# ...
mediatedDevicesConfiguration:
  mediatedDeviceTypes:
  - nvidia-222
  - nvidia-228
  - nvidia-105
  - nvidia-108
# ...
```

이 시나리오에서 각 노드에는 두 개의 카드가 있으며 둘 다 다음 vGPU 유형을 지원합니다.

```
nvidia-105
# ...
nvidia-108
nvidia-217
nvidia-299
# ...
```

OpenShift Virtualization은 각 노드에서 다음 vGPU를 생성합니다.

- 첫 번째 카드에 nvidia-105 유형의 16 vGPU입니다.
- 두 번째 카드에 nvidia-108 유형의 vGPUs입니다.

하나의 노드에는 하나 이상의 요청된 vGPU 유형을 지원하는 단일 카드가 있습니다.

OpenShift Virtualization은 **mediatedDeviceTypes** 목록에서 먼저 제공되는 지원되는 유형을 사용합니다.

예를 들어 노드 카드의 카드는 **nvidia-223** 및 **nvidia-224** 를 지원합니다. 다음 **mediatedDeviceTypes** 목록이 구성되어 있습니다.

```
# ...
mediatedDevicesConfiguration:
  mediatedDeviceTypes:
  - nvidia-22
  - nvidia-223
  - nvidia-224
# ...
```

이 예제에서 OpenShift Virtualization은 **nvidia-223** 유형을 사용합니다.

7.14.12.5. 중재된 장치 관리

미디어된 장치를 가상 머신에 할당하려면 먼저 장치를 생성하고 클러스터에 노출해야 합니다. 또한 중재된 장치를 재구성하고 제거할 수 있습니다.

7.14.12.5.1. 미디어된 장치 생성 및 노출

관리자는 조정된 장치를 생성하고 **HyperConverged** CR(사용자 정의 리소스)을 편집하여 클러스터에 노출할 수 있습니다.

사전 요구 사항

- IOMMU(Input-Output Memory Management Unit) 드라이버를 활성화했습니다.
- 하드웨어 벤더가 드라이버를 제공하는 경우 중재된 장치를 생성하려는 노드에 설치한 것입니다.
  - NVIDIA 카드를 사용하는 경우 NVIDIA GRID 드라이버를 설치했습니다.

절차

1. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged** CR을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

예 7.1. 중재 장치가 구성된 구성 파일의 예

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration:
    mediatedDeviceTypes:
    - nvidia-231
    nodeMediatedDeviceTypes:
    - mediatedDeviceTypes:
      - nvidia-233
      nodeSelector:
        kubernetes.io/hostname: node-11.redhat.com
```

```
    permittedHostDevices:
      mediatedDevices:
      - mdevNameSelector: GRID T4-2Q
        resourceName: nvidia.com/GRID_T4-2Q
      - mdevNameSelector: GRID T4-8Q
        resourceName: nvidia.com/GRID_T4-8Q
  # ...
```

2. **spec.mediatedDevicesConfiguration** 스탠자에 추가하여 중재된 장치를 생성합니다.

   YAML 스니펫의 예

```
# ...
spec:
  mediatedDevicesConfiguration:
    mediatedDeviceTypes: 1
    - <device_type>
    nodeMediatedDeviceTypes: 2
    - mediatedDeviceTypes: 3
      - <device_type>
      nodeSelector: 4
        <node_selector_key>: <node_selector_value>
# ...
```

1. 필수: 클러스터에 대한 글로벌 설정을 구성합니다.
2. 선택 사항: 특정 노드 또는 노드 그룹에 대한 글로벌 구성을 재정의합니다. 글로벌 **mediatedDeviceTypes** 구성과 함께 사용해야 합니다.
3. **nodeMediatedDeviceTypes** 를 사용하는 경우 필수 항목입니다. 지정된 노드에 대한 글로벌 **mediatedDeviceTypes** 구성을 재정의합니다.
4. **nodeMediatedDeviceTypes** 를 사용하는 경우 필수 항목입니다.키:값 쌍을 포함해야 합니다.

중요

OpenShift Virtualization 4.14 이전에는 **mediatedDeviceTypes** 필드의 이름이 **mediatedDevicesTypes** 로 지정되었습니다. 중재된 장치를 구성할 때 올바른 필드 이름을 사용해야 합니다.

3. 클러스터에 노출하려는 장치의 이름 선택기 및 리소스 이름 값을 식별합니다. 다음 단계에서 이러한 값을 **HyperConverged** CR에 추가합니다.

   a. 다음 명령을 실행하여 **resourceName** 값을 찾습니다.

```
$ oc get $NODE -o json \
  | jq '.status.allocatable \
    | with_entries(select(.key | startswith("nvidia.com/"))) \
    | with_entries(select(.value != "0"))'
```

b. **/sys/bus/pci/devices/<slot>:<bus>:<domain>.<function>/mdev_supported_types/<type>/name** 을 확인하여 **mdevNameSelector** 값을 찾습니다.
예를 들어 **nvidia-231** 유형의 이름 파일에는 선택기 문자열 **GRID T4-2Q** 가 포함되어 있습니다. **GRID T4-2Q** 를 **mdevNameSelector** 값으로 사용하면 노드에서 **nvidia-231** 유형을 사용할 수 있습니다.

4. **HyperConverged** CR의 **spec.permittedHostDevices.mediatedDevices** 스탠자에 **mdevNameSelector** 및 **resourceName** 값을 추가하여 중재된 장치를 클러스터에 노출합니다.

YAML 스니펫의 예

```
# ...
 permittedHostDevices:
  mediatedDevices:
  - mdevNameSelector: GRID T4-2Q 1
    resourceName: nvidia.com/GRID_T4-2Q 2
# ...
```

1 호스트에서 이 값에 매핑되는 중재된 장치를 노출합니다.

2 노드에 할당된 리소스 이름과 일치합니다.

5. 변경 사항을 저장하고 편집기를 종료합니다.

검증

- 선택 사항: 다음 명령을 실행하여 장치가 특정 노드에 추가되었는지 확인합니다.

```
$ oc describe node <node_name>
```

7.14.12.5.2. 미디어된 장치 변경 및 제거 정보

여러 가지 방법으로 중재된 장치를 재구성하거나 제거할 수 있습니다.

- **HyperConverged** CR을 편집하고 **mediatedDeviceTypes** 스탠자의 내용을 변경합니다.
- node **MediatedDeviceTypes** 노드 선택기와 일치하는 노드 레이블을 변경합니다.
- **HyperConverged** CR의 **spec.mediatedDevicesConfiguration** 및 **spec.permittedHostDevices** 스탠자에서 장치 정보를 제거합니다.

참고

spec.permittedHostDevices 스탠자에서 장치 정보를 제거하지 않고 **spec.mediatedDevicesConfiguration** 스탠자에서 장치 정보를 제거하는 경우 동일한 노드에서 새 중재된 장치 유형을 생성할 수 없습니다. 중재된 장치를 올바르게 제거하려면 두 스탠자 모두에서 장치 정보를 제거합니다.

7.14.12.5.3. 클러스터에서 중재된 장치 제거

클러스터에서 중재된 장치를 제거하려면 **HyperConverged** CR(사용자 정의 리소스)에서 해당 장치의 정보를 삭제합니다.

프로세스

1. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged** CR을 편집합니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **HyperConverged** CR의 **spec.mediatedDevicesConfiguration** 및 **spec.permittedHostDevices** 스탠자에서 장치 정보를 제거합니다. 두 항목을 모두 제거하면 나중에 동일한 노드에 중재된 장치 유형을 만들 수 있습니다. 예를 들면 다음과 같습니다.

   설정 파일 예

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration:
    mediatedDeviceTypes: 1
      - nvidia-231
  permittedHostDevices:
    mediatedDevices: 2
    - mdevNameSelector: GRID T4-2Q
      resourceName: nvidia.com/GRID_T4-2Q
```

   1 **nvidia-231** 장치 유형을 제거하려면 **mediatedDeviceTypes** 배열에서 삭제합니다.

   2 **GRID T4-2Q** 장치를 제거하려면 **mdevNameSelector** 필드와 해당 **resourceName** 필드를 삭제합니다.

3. 변경 사항을 저장하고 편집기를 종료합니다.

7.14.12.6. 중재된 장치 사용

하나 이상의 가상 머신에 중재된 장치를 할당할 수 있습니다.

7.14.12.6.1. CLI를 사용하여 VM에 vGPU 할당

가상 GPU(vGPU)와 같은 중재된 장치를 VM(가상 머신)에 할당합니다.

사전 요구 사항

- 중재된 장치는 **HyperConverged** 사용자 정의 리소스에서 구성됩니다.
- VM이 중지되었습니다.

프로세스

- **VirtualMachine** 매니페스트의 **spec.domain.devices.gpus** 스탠자를 편집하여 VM(가상 머신)에 중재된 장치를 할당합니다.

  가상 머신 매니페스트의 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
 domain:
   devices:
     gpus:
     - deviceName: nvidia.com/TU104GL_Tesla_T4 1
       name: gpu1 2
     - deviceName: nvidia.com/GRID_T4-2Q
       name: gpu2
```

1. 중재된 장치와 관련된 리소스 이름입니다.
2. VM에서 장치를 식별하는 이름입니다.

검증

- 장치를 가상 머신에서 사용할 수 있는지 확인하려면 다음 명령을 실행하여 **VirtualMachine** 매니페스트의 **deviceName** 값으로 <**device_name** >을 대체합니다.

```
$ lspci -nnk | grep <device_name>
```

7.14.12.6.2. 웹 콘솔을 사용하여 VM에 vGPU 할당

OpenShift Container Platform 웹 콘솔을 사용하여 가상 머신에 가상 GPU를 할당할 수 있습니다.

참고

사용자 지정 템플릿 또는 YAML 파일에서 생성된 가상 머신에 하드웨어 장치를 추가할 수 있습니다. 특정 운영 체제에 대해 사전 제공 부팅 소스 템플릿에 장치를 추가할 수 없습니다.

사전 요구 사항

- vGPU는 클러스터에서 중재된 장치로 구성됩니다.
  - 클러스터에 연결된 장치를 보려면 사이드 메뉴에서 컴퓨팅 → 하드웨어 장치를 클릭합니다.
- VM이 중지되었습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 장치를 할당할 VM을 선택합니다.
3. 세부 정보 탭에서 GPU 장치를 클릭합니다.
4. GPU 장치 추가를 클릭합니다.
5. 이름 필드에 식별 값을 입력합니다.
6. 장치 이름 목록에서 VM에 추가할 장치를 선택합니다.

7. 저장을 클릭합니다.

검증

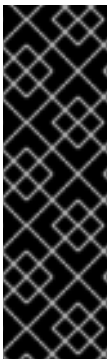
- 장치가 VM에 추가되었는지 확인하려면 YAML 탭을 클릭하고 **VirtualMachine** 구성을 검토합니다. 중재된 장치는 **spec.domain.devices** 스탠자에 추가됩니다.

7.14.12.7. 추가 리소스

- BIOS에서 Intel VT-X 및 AMD-V 가상화 하드웨어 확장 활성화

7.14.13. 가상 머신에서 Descheduler 제거 활성화

Descheduler를 사용하여 Pod를 더 적절한 노드에 다시 예약할 수 있도록 Pod를 제거할 수 있습니다. Pod가 가상 머신인 경우 Pod 제거로 인해 가상 머신이 다른 노드로 실시간 마이그레이션됩니다.



중요

가상 머신의 Descheduler 제거는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

7.14.13.1. Descheduler 프로필

기술 프리뷰 **DevPreviewLongLifecycle** 프로필을 사용하여 가상 머신에서 Descheduler를 활성화합니다. 현재 OpenShift Virtualization에서 사용할 수 있는 유일한 Descheduler 프로필입니다. 적절한 스케줄링을 보장하기 위해 예상 로드에 대한 CPU 및 메모리 요청이 있는 VM을 생성합니다.

DevPreviewLongLifecycle

이 프로필은 노드 간 리소스 사용량의 균형을 유지하고 다음 전략을 활성화합니다.

- **RemovePodsHavingTooManyRestarts**: 컨테이너가 너무 여러 번 다시 시작된 Pod와 모든 컨테이너(Init Containers 포함)에서 재시작 횟수가 100을 초과하는 Pod를 제거합니다. VM 게스트 운영 체제를 다시 시작해도 이 수가 늘어나지 않습니다.
- **LowNodeUtilization**: 활용도가 낮은 노드가 있는 경우 활용도가 높은 노드에서 Pod를 제거합니다. 제거된 Pod의 대상 노드는 스케줄러에 의해 결정됩니다.
  - 모든 임계값(CPU, 메모리, Pod 수)에서 사용량이 20% 미만인 경우 노드는 활용도가 낮은 것으로 간주됩니다.
  - 모든 임계값(CPU, 메모리, Pod 수)에서 사용량이 50%를 초과하면 노드는 과도하게 사용되는 것으로 간주됩니다.

7.14.13.2. Descheduler 설치

Descheduler는 기본적으로 사용할 수 없습니다. Descheduler를 활성화하려면 OperatorHub에서 Kube Descheduler Operator를 설치하고 Descheduler 프로필을 한 개 이상 활성화해야 합니다.

기본적으로 Descheduler는 예측 모드에서 실행되므로 Pod 제거만 시뮬레이션합니다. Pod 제거를 수행하려면 Descheduler가 자동으로 모드를 변경해야 합니다.

중요

클러스터에서 호스팅되는 컨트롤 플레인을 활성화한 경우 사용자 정의 우선순위 임계값을 설정하여 호스팅된 컨트롤 플레인 네임스페이스의 Pod가 제거될 가능성을 줄입니다. 호스팅된 컨트롤 플레인 우선순위 클래스 클래스의 가장 낮은 우선 순위 값(**100000000**)이 있으므로 우선순위 임계값 클래스 이름을 **hypershift-control-plane** 로 설정합니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 OpenShift Container Platform에 로그인되어 있습니다.
- OpenShift Container Platform 웹 콘솔에 액세스합니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. Kube Descheduler Operator에 필요한 네임스페이스를 생성합니다.
   a. 관리 → 네임스페이스로 이동하여 네임스페이스 생성을 클릭합니다.
   b. 이름 필드에 **openshift-kube-descheduler-operator** 를 입력하고 라벨 필드에 **openshift.io/cluster-monitoring=true** 를 입력하여 Descheduler 지표를 활성화한 후생성 을 클릭합니다.
3. Kube Descheduler Operator를 설치합니다.
   a. Operators → OperatorHub로 이동합니다.
   b. 필터 박스에 Kube Descheduler Operator를 입력합니다.
   c. Kube Descheduler Operator를 선택하고 설치를 클릭합니다.
   d. Operator 설치 페이지에서 클러스터의 특정 네임스페이스를 선택합니다. 드롭다운 메뉴에서 openshift-kube-descheduler-operator를 선택합니다.
   e. 업데이트 채널 및 승인 전략 값을 원하는 값으로 조정합니다.
   f. 설치를 클릭합니다.
4. Descheduler 인스턴스를 생성합니다.
   a. Operator → 설치된 Operator 페이지에서 Kube Descheduler Operator를 클릭합니다.
   b. Kube Descheduler 탭을 선택하고 KubeDescheduler 생성을 클릭합니다.
   c. 필요에 따라 설정을 편집합니다.
      i. 제거 시뮬레이션 대신 Pod를 제거하려면 Mode 필드를 자동으로 변경합니다.
      ii. Profiles 섹션을 확장하고 **DevPreviewLongLifecycle** 를 선택합니다. **AffinityAndTaints** 프로필은 기본적으로 활성화되어 있습니다.

중요

현재 OpenShift Virtualization에서 사용할 수 있는 유일한 프로필은 **DevPreviewLongLifecycle** 입니다.

나중에 OpenShift CLI(**oc**)를 사용하여 Descheduler의 프로필 및 설정을 구성할 수도 있습니다.

7.14.13.3. VM(가상 머신)에서 Descheduler 제거 활성화

Descheduler가 설치되면 **VirtualMachine** CR(사용자 정의 리소스)에 주석을 추가하여 VM에서 Descheduler 제거를 활성화할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔 또는 OpenShift CLI(**oc**)에 Descheduler를 설치합니다.
- VM이 실행되고 있지 않은지 확인합니다.

프로세스

1. VM을 시작하기 전에 **VirtualMachine** CR에 **Descheduler.alpha.kubernetes.io/evict** 주석을 추가합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    metadata:
      annotations:
        descheduler.alpha.kubernetes.io/evict: "true"
```

2. 설치 중에 웹 콘솔에서 **DevPreviewLongLifecycle** 프로필을 설정하지 않은 경우 **KubeDescheduler** 오브젝트의 **spec.profile** 섹션에 **DevPreviewLongLifecycle** 를 지정합니다.

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600
  profiles:
  - DevPreviewLongLifecycle
  mode: Predictive (1)
```

(1) 기본적으로 Descheduler는 Pod를 제거하지 않습니다. Pod를 제거하려면 **mode** 를 **Automatic** 으로 설정합니다.

이제 VM에서 Descheduler가 활성화됩니다.

7.14.13.4. 추가 리소스

- Descheduler 개요

7.14.14. 가상 머신의 고가용성 정보

실패한 노드를 수동으로 삭제하여 VM 장애 조치를 트리거하거나 노드 수정을 구성하여 VM(가상 머신)의 고가용성을 활성화할 수 있습니다.

실패한 노드 수동 삭제

노드가 실패하고 머신 상태 점검이 클러스터에 배포되지 않으면 **runStrategy: Always** 가 구성된 가상 머신이 정상 노드로 자동 재배치되지 않습니다. VM 장애 조치를 트리거하려면 **Node** 오브젝트를 수동으로 삭제해야 합니다.

가상 머신 장애 조치를 트리거하는 데 실패한 노드 삭제를 참조하십시오.

노드 수정 구성

OperatorHub에서 Self Node Remediation Operator를 설치하고 머신 상태 점검 또는 노드 수정 검사를 활성화하여 노드 수정을 구성할 수 있습니다.

노드 수정, 펜싱 및 유지 관리에 대한 자세한 내용은 Workload Availability for Red Hat OpenShift설명서를 참조하십시오.

7.14.15. 가상 머신 컨트롤 플레인 튜닝

OpenShift Virtualization에서는 컨트롤 플레인 수준에서 다음과 같은 튜닝 옵션을 제공합니다.

- 고정 **QPS** 및 버스트 속도를 사용하여 수백 개의 VM(가상 머신)을 하나의 일괄 처리로 생성하는 **highBurst** 프로필
- 워크로드 유형에 따른 마이그레이션 설정 조정

7.14.15.1. highBurst 프로필 구성

highBurst 프로필을 사용하여 하나의 클러스터에서 다수의 VM(가상 머신)을 생성하고 유지 관리합니다.

프로세스

- 다음 패치를 적용하여 **highBurst** 튜닝 정책 프로필을 활성화합니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type=json -p='[{"op": "add", "path": "/spec/tuningPolicy", \
  "value": "highBurst"}]'
```

검증

- 다음 명령을 실행하여 **highBurst** 튜닝 정책 프로필이 활성화되었는지 확인합니다.

```
$ oc get kubevirt.kubevirt.io/kubevirt-kubevirt-hyperconverged \
  -n openshift-cnv -o go-template --template='{{range $config, \
  $value := .spec.configuration}} {{if eq $config "apiConfiguration" \
  "webhookConfiguration" "controllerConfiguration" "handlerConfiguration"}} \
  {{"\n"}} {{$config}} = {{$value}} {{end}} {{end}} {{"\n"}}
```

7.14.16. 컴퓨팅 리소스 할당

OpenShift Virtualization에서 VM(가상 머신)에 할당된 컴퓨팅 리소스는 보장된 CPU 또는 시간 초과된 CPU 공유에 의해 지원됩니다.

보장된 CPU(CPU 예약이라고도 함)는 CPU 코어 또는 스레드를 특정 워크로드에 전용하므로 다른 워크로드에서 사용할 수 없습니다. 보장된 CPU를 VM에 할당하면 VM이 예약된 물리적 CPU에만 액세스할 수 있습니다. VM이 보장된 CPU를 사용하도록 전용 리소스를 활성화합니다.

시간 초과된 CPU는 각 워크로드에 대해 공유 물리적 CPU의 시간 슬라이스를 전용으로 사용합니다. VM 생성 중 또는 VM이 오프라인인 경우 슬라이스 크기를 지정할 수 있습니다. 기본적으로 각 vCPU는 물리적 CPU 시간 100밀리초 또는 1/10초를 받습니다.

CPU 예약 유형은 인스턴스 유형 또는 VM 구성에 따라 다릅니다.

7.14.16.1. CPU 리소스 과다 할당

시간 분할을 사용하면 여러 가상 CPU(vCPU)가 단일 물리적 CPU를 공유할 수 있습니다. 이를 *CPU 과다 할당*이라고 합니다. 보장된 VM은 오버 커밋할 수 없습니다.

VM에 CPU를 할당할 때 성능보다 VM 밀도를 우선시하도록 CPU 과다 할당을 구성합니다. vCPU의 CPU 초과 커밋이 증가하면 지정된 노드에 더 많은 VM이 적합합니다.

7.14.16.2. CPU 할당 비율 설정

CPU 할당 Ratio는 vCPU를 물리적 CPU의 시간 슬라이스에 매핑하여 오버 커밋 정도를 지정합니다.

예를 들어 10:1의 매핑 또는 비율은 시간 슬라이스를 사용하여 10개의 가상 CPU를 1개의 물리적 CPU에 매핑합니다.

각 물리적 CPU에 매핑된 기본 vCPU 수를 변경하려면 **HyperConverged** CR에서 **vmiCPUAllocationRatio** 값을 설정합니다. Pod CPU 요청은 vCPU 수를 CPU 할당 비율의 reciprocal으로 곱하여 계산됩니다. 예를 들어 **vmiCPUAllocationRatio** 가 10으로 설정된 경우 OpenShift Virtualization은 해당 VM에 대해 Pod에서 10배 더 적은 CPU를 요청합니다.

프로세스

HyperConverged CR에서 **vmiCPUAllocationRatio** 값을 설정하여 노드 CPU 할당 비율을 정의합니다.

1. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged** CR을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **vmiCPUAllocationRatio**:를 설정합니다.

```
...
spec:
  resourceRequirements:
    vmiCPUAllocationRatio: 1 (1)
# ...
```

(1) **vmiCPUAllocationRatio** 를 1 로 설정하면 Pod에 대해 최대 vCPU가 요청됩니다.

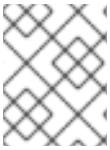
7.14.16.3. 추가 리소스

- Pod의 서비스 품질 클래스

7.14.17. 다중 대기열 기능 정보

멀티 큐 기능을 사용하여 여러 vCPU가 있는 VM(가상 머신)에서 네트워크 처리량과 성능을 확장합니다.

기본적으로 도메인 XML에서 파생되는 **queueCount** 값은 VM에 할당된 vCPU 수에 따라 결정됩니다. vCPU 수가 증가함에 따라 네트워크 성능은 확장되지 않습니다. 또한 virtio-net에는 Tx 및 Rx 대기열이 하나뿐이므로 게스트는 패키지를 병렬로 전송하거나 검색할 수 없습니다.



참고

virtio-net multiqueue를 활성화하면 게스트 인스턴스의 vNIC 수가 vCPU 수에 비례하면 크게 개선되지 않습니다.

7.14.17.1. 알려진 제한 사항

- 호스트에서 virtio-net multiqueue를 사용할 수 있지만 관리자가 게스트 운영 체제에서는 활성화되지 않은 경우에도 Cryostat 벡터가 계속 사용됩니다.
- 각 virtio-net 대기열은 vhost 드라이버에 64KiB의 커널 메모리를 사용합니다.
- CPU가 16개 이상인 VM을 시작하면 **networkInterfaceMultiqueue** 가 'true'(CNV-16107)로 설정된 경우 연결되지 않습니다.

7.14.17.2. 다중 대기열 기능 활성화

VirtIO 모델로 구성된 인터페이스에 대해 다중 대기열 기능을 활성화합니다.

프로세스

1. VM의 **VirtualMachine** 매니페스트 파일에서 **networkInterfaceMultiqueue** 값을 **true** 로 설정하여 다중 큐 기능을 활성화합니다.

```
apiVersion: kubevirt.io/v1
kind: VM
spec:
  domain:
    devices:
      networkInterfaceMultiqueue: true
```

2. **VirtualMachine** 매니페스트 파일을 저장하여 변경 사항을 적용합니다.

7.15. VM 디스크

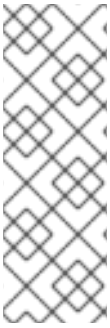
7.15.1. VM 디스크 핫플러그

VM(가상 머신) 또는 VMI(가상 머신 인스턴스)를 중지하지 않고 가상 디스크를 추가하거나 제거할 수 있습니다.

데이터 볼륨 및 PVC(영구 볼륨 클레임)만 핫플러그 및 핫 플러그할 수 있습니다. 컨테이너 디스크를 핫플러그하거나 핫 플러그 해제할 수 없습니다.

핫플러그 디스크는 재부팅 후에도 VM으로 유지됩니다. VM에서 제거하려면 디스크를 분리해야 합니다.

VM에 영구적으로 마운트되도록 핫플러그 디스크를 영구적으로 만들 수 있습니다.



참고

각 VM에는 **virtio-scsi** 컨트롤러가 있으므로 핫플러그 디스크가**scsi** 버스를 사용할 수 있습니다. **virtio-scsi** 컨트롤러는 성능 이점을 유지하면서**virtio** 의 제한을 극복합니다. 확장성이 뛰어나고 4억 개 이상의 디스크 핫 플러그를 지원합니다.

확장 불가능하므로 일반 **virtio** 를 핫플러그 디스크에 사용할 수 없습니다. 각**virtio** 디스크는 VM의 제한된 PCI Express(PCIe) 슬롯 중 하나를 사용합니다. PCIe 슬롯은 다른 장치에서도 사용되며 사전에 예약해야 합니다. 따라서 필요에 따라 슬롯을 사용할 수 없습니다.

7.15.1.1. 웹 콘솔을 사용하여 디스크 핫플러그 및 핫플러그

OpenShift Container Platform 웹 콘솔을 사용하여 VM을 실행하는 동안 VM(가상 머신)에 연결하여 디스크를 핫 플러그할 수 있습니다.

핫플러그 디스크는 연결을 해제할 때까지 VM에 연결된 상태로 유지됩니다.

VM에 영구적으로 마운트되도록 핫플러그 디스크를 영구적으로 만들 수 있습니다.

사전 요구 사항

- 핫플러그에 사용할 수 있는 데이터 볼륨 또는 PVC(영구 볼륨 클레임)가 있어야 합니다.

프로세스

1. 웹 콘솔에서 가상화 → VirtualMachines 로 이동합니다.
2. 실행 중인 VM을 선택하여 세부 정보를 확인합니다.
3. VirtualMachine 세부 정보 페이지에서 구성 → 디스크 를 클릭합니다.
4. 핫플러그 디스크를 추가합니다.
   a. 디스크 추가를 클릭합니다.
   b. Add disk (hot plugged) 창에서 소스 목록에서 디스크를 선택하고저장을 클릭합니다.
5. 선택 사항: 핫플러그 디스크를 분리합니다.
   a. 디스크 옆에 있는 옵션 메뉴 를 클릭하고 분리를 선택합니다.
   b. Detach 를 클릭합니다.
6. 선택 사항: 핫플러그 디스크를 영구적으로 설정합니다.
   a. 디스크 옆에 있는 옵션 메뉴 를 클릭하고 영구적으로 만들기를 선택합니다.
   b. VM을 재부팅하여 변경 사항을 적용합니다.

7.15.1.2. 명령줄을 사용하여 디스크 핫플러그 및 핫플러그

명령줄을 사용하여 VM(가상 머신)을 실행하는 동안 디스크를 핫플러그 및 핫 플러그 해제할 수 있습니다.

VM에 영구적으로 마운트되도록 핫플러그 디스크를 영구적으로 만들 수 있습니다.

사전 요구 사항

- 핫 플러그에 사용할 수 있는 데이터 볼륨 또는 PVC(영구 볼륨 클레임)가 하나 이상 있어야 합니다.

프로세스

- 다음 명령을 실행하여 디스크를 핫 플러그합니다.

```
$ virtctl addvolume <virtual-machine|virtual-machine-instance> \
  --volume-name=<datavolume|PVC> \
  [--persist] [--serial=<label-name>]
```

  - 선택적 **--persist** 플래그를 사용하여 핫플러그 디스크를 가상 머신 사양에 영구적으로 마운트된 가상 디스크로 추가합니다. 가상 시스템을 중지, 다시 시작 또는 재부팅하여 가상 디스크를 영구적으로 마운트합니다. **--persist** 플래그를 지정한 후에는 더 이상 가상 디스크를 핫플러그하거나 핫 플러그 해제할 수 없습니다. **--persist** 플래그는 가상 머신 인스턴스가 아닌 가상 머신에 적용됩니다.

  - 선택적 **--serial** 플래그를 사용하면 선택한 영숫자 문자열 레이블을 추가할 수 있습니다. 이를 통해 게스트 가상 머신에서 핫플러그 디스크를 식별하는 데 도움이 됩니다. 이 옵션을 지정하지 않으면 레이블의 기본값은 핫 플러그된 데이터 볼륨 또는 PVC의 이름으로 설정됩니다.

- 다음 명령을 실행하여 디스크를 핫플러그합니다.

```
$ virtctl removevolume <virtual-machine|virtual-machine-instance> \
  --volume-name=<datavolume|PVC>
```

7.15.2. 가상 머신 디스크 확장

디스크의 PVC(영구 볼륨 클레임)를 확장하여 VM(가상 머신) 디스크의 크기를 늘릴 수 있습니다.

스토리지 공급자가 볼륨 확장을 지원하지 않는 경우 빈 데이터 볼륨을 추가하여 VM의 사용 가능한 가상 스토리지를 확장할 수 있습니다.

VM 디스크 크기를 줄일 수 없습니다.

7.15.2.1. VM 디스크 PVC 확장

디스크의 PVC(영구 볼륨 클레임)를 확장하여 VM(가상 머신) 디스크의 크기를 늘릴 수 있습니다.

PVC에서 파일 시스템 볼륨 모드를 사용하는 경우 디스크 이미지 파일은 파일 시스템 오버헤드용으로 일부 공간을 예약하는 동안 디스크 이미지 파일이 사용 가능한 크기로 확장됩니다.

프로세스

1. 확장하려는 VM 디스크의 **PersistentVolumeClaim** 매니페스트를 편집합니다.

```
$ oc edit pvc <pvc_name>
```

2. 디스크 크기를 업데이트합니다.

```
apiVersion: v1
```

```
kind: PersistentVolumeClaim
metadata:
  name: vm-disk-expand
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 3Gi 1
# ...
```

1 새 디스크 크기를 지정합니다.

볼륨 확장을 위한 추가 리소스

- Windows에서 기본 볼륨 확장
- Red Hat Enterprise Linux에서 데이터를 삭제하지 않고 기존 파일 시스템 파티션 확장
- Red Hat Enterprise Linux에서 논리 볼륨 및 파일 시스템 확장

7.15.2.2. 빈 데이터 볼륨을 추가하여 사용 가능한 가상 스토리지 확장

빈 데이터 볼륨을 추가하여 VM(가상 머신)의 사용 가능한 스토리지를 확장할 수 있습니다.

사전 요구 사항

- 영구 볼륨이 하나 이상 있어야 합니다.

프로세스

1. 다음 예와 같이 **DataVolume** 매니페스트를 생성합니다.

   DataVolume 매니페스트 예

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  storage:
    resources:
      requests:
        storage: <2Gi> 1
  storageClassName: "<storage_class>" 2
```

1 데이터 볼륨에 요청된 사용 가능한 공간의 양을 지정합니다.

2 선택 사항: 스토리지 클래스를 지정하지 않으면 기본 스토리지 클래스가 사용됩니다.

2. 다음 명령을 실행하여 데이터 볼륨을 생성합니다.

```
$ oc create -f <blank-image-datavolume>.yaml
```

데이터 볼륨에 대한 추가 리소스

- 데이터 볼륨에 대한 사전 할당 모드 구성
- 데이터 볼륨 주석 관리

7.15.3. 가상 머신의 공유 볼륨 구성

여러 VM(가상 머신)이 동일한 기본 스토리지를 공유할 수 있도록 공유 디스크를 구성할 수 있습니다. 공유 디스크의 볼륨은 블록 모드여야 합니다.

스토리지를 다음 유형 중 하나로 노출하여 디스크 공유를 구성합니다.

- 일반 VM 디스크
- 공유 볼륨의 Windows Cryostat에 필요한 대로 iSCSi 연결 및 원시 장치 매핑이 있는 LUN(Logical Unit Number) 디스크

디스크 공유 구성 외에도 각 일반 VM 디스크 또는 LUN 디스크에 오류 정책을 설정할 수도 있습니다. 오류 정책은 디스크 읽기 또는 쓰기에서 입력/출력 오류가 발생할 때 하이퍼바이저가 작동하는 방식을 제어합니다.

7.15.3.1. 가상 머신 디스크를 사용하여 디스크 공유 구성

여러 VM(가상 머신)에서 스토리지를 공유할 수 있도록 블록 볼륨을 구성할 수 있습니다.

게스트 운영 체제에서 실행되는 애플리케이션에 따라 VM에 대해 구성해야 하는 스토리지 옵션이 결정됩니다. 디스크 유형 디스크 는 볼륨을 VM에 일반 디스크로 노출합니다.

각 디스크에 오류 정책을 설정할 수 있습니다. 오류 정책은 디스크를 쓰거나 읽는 동안 입력/출력 오류가 발생할 때 하이퍼바이저가 작동하는 방식을 제어합니다. 기본 동작은 VM을 중지하고 Kubernetes 이벤트를 생성합니다.

기본 동작을 수락하거나 오류 정책을 다음 옵션 중 하나로 설정할 수 있습니다.

- 보고: 게스트의 오류를 보고합니다.
- 오류를 무시하는 을 무시합니다. 읽기 또는 쓰기 실패가 감지되지 않습니다.
- **ENOSPACE:** 디스크 공간이 충분하지 않음을 나타내는 오류를 생성합니다.

사전 요구 사항

- 디스크를 공유하는 VM이 다른 노드에서 실행되는 경우 볼륨 액세스 모드는 RWX(**ReadWriteMany**)여야 합니다.
  디스크를 공유하는 VM이 동일한 노드에서 실행되는 경우 RWO(**ReadWriteOnce**) 볼륨 액세스 모드로 충분합니다.
- 스토리지 공급자는 필요한 CSI(Container Storage Interface) 드라이버를 지원해야 합니다.

프로세스

1. 다음 예와 같이 VM에 대한 **VirtualMachine** 매니페스트를 생성하여 필요한 값을 설정합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <vm_name>
spec:
  template:
# ...
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: rootdisk
            errorPolicy: report (1)
            disk1: disk_one (2)
          - disk:
              bus: virtio
            name: cloudinitdisk
            disk2: disk_two
            shareable: true (3)
          interfaces:
          - masquerade: {}
            name: default
```

(1) 오류 정책을 식별합니다.

(2) 장치를 디스크로 식별합니다.

(3) 공유 디스크를 식별합니다.

2. **VirtualMachine** 매니페스트 파일을 저장하여 변경 사항을 적용합니다.

7.15.3.2. LUN을 사용하여 디스크 공유 구성

SCSI 영구 예약을 활성화하여 LUN에서 지원하는 VM(가상 머신) 디스크를 여러 가상 머신 간에 공유할 수 있습니다. shared 옵션을 활성화하면 기본 스토리지에 대해 Windows 페일오버 클러스터링 구현에 필요한 고급 SCSI 명령을 사용할 수 있습니다. 공유할 모든 디스크는 블록 모드여야 합니다.

LUN 유형의 디스크는 볼륨을 VM에 LUN 장치로 노출합니다. 이렇게 하면 VM에서 디스크에서 임의의 iSCSI 명령 패스스루를 실행할 수 있습니다.

SCSI 영구 예약 옵션을 통해 LUN을 예약하여 VM의 데이터를 외부 액세스로부터 보호합니다. 예약을 활성화하려면 기능 게이트 옵션을 구성합니다. 그런 다음 LUN 디스크에서 옵션을 활성화하여 VM에 필요한 SCSI 장치별 입력 및 출력 제어(IOCTLs)를 실행합니다.

각 LUN 디스크에 오류 정책을 설정할 수 있습니다. 오류 정책은 디스크 읽기 또는 쓰기에서 입력/출력 오류가 발생할 때 하이퍼바이저가 작동하는 방식을 제어합니다. 기본 동작은 게스트를 중지하고 Kubernetes 이벤트를 생성합니다.

공유 볼륨에 대해 필요에 따라 iSCSi 연결 및 영구 예약이 있는 LUN 디스크의 경우 오류 정책을 보고 하도록 설정합니다.

사전 요구 사항

- 기능 게이트 옵션을 구성하려면 클러스터 관리자 권한이 있어야 합니다.
- 디스크를 공유하는 VM이 다른 노드에서 실행되는 경우 볼륨 액세스 모드는 RWX(**ReadWriteMany**)여야 합니다.
  디스크를 공유하는 VM이 동일한 노드에서 실행되는 경우 RWO(**ReadWriteOnce**) 볼륨 액세스 모드로 충분합니다.
- 스토리지 공급자는 SCSI 프로토콜을 사용하는 CSI(Container Storage Interface) 드라이버를 지원해야 합니다.
- 클러스터 관리자이고 LUN을 사용하여 디스크 공유를 구성하려면 **HyperConverged** CR(사용자 정의 리소스)에서 클러스터의 기능 게이트를 활성화해야 합니다.

프로세스

1. 다음 예와 같이 VM에 대한 **VirtualMachine** 매니페스트를 편집하거나 생성하여 필요한 값을 설정합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-0
spec:
  template:
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: sata
            name: rootdisk
          - errorPolicy: report (1)
            lun: (2)
              bus: scsi
              reservation: true (3)
            name: na-shared
            serial: shared1234
      volumes:
      - dataVolume:
          name: vm-0
        name: rootdisk
      - name: na-shared
        persistentVolumeClaim:
          claimName: pvc-na-share
```

(1) 오류 정책을 식별합니다.

(2) LUN 디스크를 식별합니다.

(3) 영구 예약이 활성화되어 있는지 확인합니다.

2. **VirtualMachine** 매니페스트 파일을 저장하여 변경 사항을 적용합니다.

7.15.3.2.1. LUN 및 웹 콘솔을 사용하여 디스크 공유 구성

OpenShift Container Platform 웹 콘솔을 사용하여 LUN을 사용하여 디스크 공유를 구성할 수 있습니다.

사전 요구 사항

- 클러스터 관리자는 지속성 기능 게이트 설정을 활성화해야 합니다.

프로세스

1. 웹 콘솔에서 가상화 → **VirtualMachines** 를 클릭합니다.
2. **VM**을 선택하여 **VirtualMachine** 세부 정보 페이지를 엽니다.
3. 스토리지를 확장합니다.
4. 디스크 탭에서 디스크 추가 를 클릭합니다.
5. 이름,소스,크기,인터페이스 및 스토리지 클래스를 지정합니다.
6. **LUN** 을 유형으로 선택합니다.
7. 액세스 모드로 공유 액세스(**RWX**) 를 선택합니다.
8. 볼륨 모드로 **Block** 을 선택합니다.
9. 고급 설정을 확장하고 두 확인란을 모두 선택합니다.
10. 저장을 클릭합니다.

7.15.3.2.2. LUN 및 명령줄을 사용하여 디스크 공유 구성

명령줄을 사용하여 **LUN**을 사용하여 디스크 공유를 구성할 수 있습니다.

프로세스

1. 다음 예와 같이 VM에 대한 VirtualMachine 매니페스트를 편집하거나 생성하여 필요한 값을 설정합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-0
spec:
  template:
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: sata
            name: rootdisk
          - errorPolicy: report
            lun: (1)
              bus: scsi
              reservation: true (2)
            name: na-shared
            serial: shared1234
      volumes:
      - dataVolume:
          name: vm-0
        name: rootdisk
      - name: na-shared
        persistentVolumeClaim:
          claimName: pvc-na-share
```

(1) LUN 디스크를 식별합니다.

(2) 영구 예약이 활성화되어 있는지 확인합니다.

2. VirtualMachine 매니페스트 파일을 저장하여 변경 사항을 적용합니다.

7.15.3.3. PersistentReservation 기능 게이트 활성화

SCSI persistentReservation 기능 게이트를 활성화하고 LUN 지원 블록 모드 VM(가상 머신) 디스크를 여러 가상 머신 간에 공유할 수 있습니다.

persistentReservation 기능 게이트는 기본적으로 비활성화되어 있습니다. 웹 콘솔 또는 명령줄을 사용하여 **persistentReservation** 기능 게이트를 활성화할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 권한이 필요합니다.
- 디스크를 공유하는 **VM**이 다른 노드에서 실행 중인 경우 볼륨 액세스 모드 **RWX(ReadWriteMany)**가 필요합니다. 디스크를 공유하는 **VM**이 동일한 노드에서 실행 중인 경우 **RWO(ReadWriteOnce)** 볼륨 액세스 모드로 충분합니다.
- 스토리지 공급자는 **SCSI** 프로토콜을 사용하는 **CSI(Container Storage Interface)** 드라이버를 지원해야 합니다.

7.15.3.3.1. 웹 콘솔을 사용하여 PersistentReservation 기능 게이트 활성화

여러 가상 머신 간에 **LUN** 지원 블록 모드 **VM(**가상 머신**)** 디스크를 공유할 수 있도록 **PersistentReservation** 기능 게이트를 활성화해야 합니다. 기능 게이트를 활성화하려면 클러스터 관리자 권한이 필요합니다.

프로세스

1. 웹 콘솔에서 가상화 → 개요 를 클릭합니다.
2. 설정 탭을 클릭합니다.
3. 클러스터를 선택합니다.
4. **SCSI** 영구 예약을 확장하고 영구 예약 활성화를 **on**으로 설정합니다.

7.15.3.3.2. 명령줄을 사용하여 PersistentReservation 기능 게이트 활성화

명령줄을 사용하여 **persistentReservation** 기능 게이트를 활성화합니다. 기능 게이트를 활성화하려면 클러스터 관리자 권한이 필요합니다.

프로세스

1. 다음 명령을 실행하여 **persistentReservation** 기능 게이트를 활성화합니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op":"replace","path":"/spec/featureGates/persistentReservation",
"value": true}]'
```

추가 리소스

- 영구 예약 도우미 프로토콜
- **Windows Server 및 Azure Stack HCI의 장애 조치 cluster**

8장. 네트워킹

8.1. 네트워킹 개요

OpenShift Virtualization은 사용자 정의 리소스 및 플러그인을 사용하여 고급 네트워킹 기능을 제공합니다. 가상 머신(VM)은 OpenShift Container Platform 네트워킹 및 해당 에코시스템과 통합됩니다.

참고

단일 스택 IPv6 클러스터에서 OpenShift Virtualization을 실행할 수 없습니다.

8.1.1. OpenShift Virtualization 네트워킹 용어집

다음 용어는 OpenShift Virtualization 설명서 전체에서 사용됩니다.

CNI(컨테이너 네트워크 인터페이스(Container Network Interface))

컨테이너 네트워크 연결에 중점을 둔 Cloud Native Computing Foundation 프로젝트입니다. OpenShift Virtualization에서는 CNI 플러그인을 사용하여 기본 Kubernetes 네트워킹 기능을 기반으로 빌드합니다.

Multus

Pod 또는 가상 머신에서 필요한 인터페이스를 사용할 수 있도록 여러 CNI가 존재할 수 있는 "메타" CNI 플러그인입니다.

CRD(사용자 정의 리소스 정의(Custom Resource Definition))

사용자 정의 리소스를 정의할 수 있는 Kubernetes API 리소스 또는 CRD API 리소스를 사용하여 정의한 오브젝트입니다.

네트워크 연결 정의(NAD)

Pod, 가상 머신 및 가상 머신 인스턴스를 하나 이상의 네트워크에 연결할 수 있는 Multus 프로젝트에서 도입한 CRD입니다.

노드 네트워크 구성 정책(NNCP)

nmstate 프로젝트에서 도입한 CRD로, 노드에서 요청된 네트워크 구성을 설명합니다. NodeNetworkConfigurationPolicy 매니페스트를 클러스터에 적용하는 방식으로 인터페이스 추가 및 제거를 포함하여 노드 네트워크 구성을 업데이트합니다.

8.1.2. 기본 Pod 네트워크 사용

가상 머신을 기본 Pod 네트워크에 연결

각 VM은 기본적으로 기본 내부 포드 네트워크에 연결됩니다. VM 사양을 편집하여 네트워크 인터페이스를 추가하거나 제거할 수 있습니다.

가상 머신을 서비스로 노출

Service 오브젝트를 생성하여 클러스터 또는 클러스터 외부에 VM을 노출할 수 있습니다. 온프레미스 클러스터의 경우 MetalLB Operator를 사용하여 로드 밸런싱 서비스를 구성할 수 있습니다. OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 MetalLB Operator를 설치할 수 있습니다.

8.1.3. VM 보조 네트워크 인터페이스 구성

가상 머신을 Linux 브리지 네트워크에 연결

Kubernetes NMState Operator를 설치하여 보조 네트워크에 대한 Linux 브리지, VLAN 및 본딩을 구성합니다.

다음 단계를 수행하여 Linux 브리지 네트워크를 생성하고 VM을 네트워크에 연결할 수 있습니다.

1. NodeNetworkConfigurationPolicy CRD(사용자 정의 리소스 정의)를 생성하여 Linux 브리지 네트워크 장치를 구성합니다.
2. NetworkAttachmentDefinition CRD를 생성하여 Linux 브리지 네트워크를 구성합니다.
3. VM 구성에 네트워크 세부 정보를 포함하여 VM을 Linux 브리지 네트워크에 연결합니다.

SR-IOV 네트워크에 가상 머신 연결

높은 대역폭 또는 짧은 대기 시간이 필요한 애플리케이션에 대해 베어 메탈 또는 RHOSP(Red Hat OpenStack Platform) 인프라에 설치된 OpenShift Container Platform 클러스터의 추가 네트워크와 함께 SR-IOV(Single Root I/O Virtualization) 네트워크 장치를 사용할 수 있습니다.

SR-IOV 네트워크 장치 및 네트워크 연결을 관리하려면 클러스터에 SR-IOV Network Operator를 설치해야 합니다.

다음 단계를 수행하여 VM을 SR-IOV 네트워크에 연결할 수 있습니다.

1. SriovNetworkNodePolicy CRD 를 생성하여 SR-IOV 네트워크 장치를 구성합니다.

2. SriovNetwork 오브젝트 를 생성하여 SR-IOV 네트워크를 구성합니다.

3. VM 구성에 네트워크 세부 정보를 포함하여 VM을 SR-IOV 네트워크에 연결합니다.

가상 머신을 OVN-Kubernetes 보조 네트워크에 연결

VM을 OVN(Open Virtual Network)-Kubernetes 보조 네트워크에 연결할 수 있습니다. OpenShift Virtualization은 OVN-Kubernetes의 계층 2 및 localnet 토폴로지를 지원합니다.

- 계층 2 토폴로지는 클러스터 전체 논리 스위치로 워크로드를 연결합니다. OVN-Kubernetes CNI(Container Network Network Interface) 플러그인은 Geneve(Generic Network Virtualization Encapsulation) 프로토콜을 사용하여 노드 간에 오버레이 네트워크를 생성합니다. 이 오버레이 네트워크를 사용하여 추가 물리적 네트워킹 인프라를 구성하지 않고도 다른 노드의 VM을 연결할 수 있습니다.

- localnet 토폴로지는 보조 네트워크를 물리적 오버레이에 연결합니다. 이렇게 하면 east-west 클러스터 트래픽과 클러스터 외부에서 실행되는 서비스에 대한 액세스를 모두 사용할 수 있지만 클러스터 노드에서 기본 OVS(Open vSwitch) 시스템에 대한 추가 구성이 필요합니다.

OVN-Kubernetes 보조 네트워크를 구성하고 VM을 해당 네트워크에 연결하려면 다음 단계를 수행합니다.

1. 네트워크 연결 정의(NAD)를 생성하여 OVN-Kubernetes 보조 네트워크를 구성합니다.

참고

localnet 토폴로지의 경우 CryostatD를 생성하기 전에 NodeNetworkConfigurationPolicy 오브젝트를 생성하여 OVS 브리지를 구성해야 합니다.

2. VM 사양에 네트워크 세부 정보를 추가하여 VM을 OVN-Kubernetes 보조 네트워크에 연결합니다.

보조 네트워크 인터페이스 핫플러그

VM을 중지하지 않고 보조 네트워크 인터페이스를 추가하거나 제거할 수 있습니다. OpenShift Virtualization은 VirtIO 장치 드라이버를 사용하는 Linux 브리지 인터페이스에 대한 핫 플러그 및 핫 플러그를 지원합니다.

SR-IOV와 함께 DPDK 사용

DPDK(Data Plane Development Kit)는 빠른 패킷 처리를 위한 라이브러리 및 드라이버 세트를 제공합니다. SR-IOV 네트워크에서 DPDK 워크로드를 실행하도록 클러스터 및 VM을 구성할 수 있습니다.

실시간 마이그레이션을 위한 전용 네트워크 구성

실시간 마이그레이션을 위해 전용 Multus 네트워크를 구성할 수 있습니다. 전용 네트워크는 실시간 마이그레이션 중에 테넌트 워크로드에 대한 네트워크 포화 상태로 인한 영향을 최소화합니다.

클러스터 FQDN을 사용하여 가상 머신에 액세스

FQDN(정규화된 도메인 이름)을 사용하여 클러스터 외부에서 보조 네트워크 인터페이스에 연결된 VM에 액세스할 수 있습니다.

IP 주소 구성 및 보기

VM을 생성할 때 보조 네트워크 인터페이스의 IP 주소를 구성할 수 있습니다. IP 주소는 cloud-init를 사용하여 프로비저닝됩니다. OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 VM의 IP 주소를 볼 수 있습니다. 네트워크 정보는 QEMU 게스트 에이전트에 의해 수집됩니다.

8.1.4. OpenShift Service Mesh와 통합

서비스 메시에 가상 머신 연결

OpenShift Virtualization은 OpenShift Service Mesh와 통합됩니다. Pod와 가상 머신 간의 트래픽을 모니터링, 시각화 및 제어할 수 있습니다.

8.1.5. MAC 주소 풀 관리

네트워크 인터페이스의 MAC 주소 풀 관리

KubeMacPool 구성 요소는 공유 MAC 주소 풀의 VM 네트워크 인터페이스에 MAC 주소를 할당합니다. 이렇게 하면 각 네트워크 인터페이스에 고유한 MAC 주소가 할당됩니다. 해당 VM에서 생성된 가상 머신 인스턴스는 재부팅 시 할당된 MAC 주소를 유지합니다.

8.1.6. SSH 액세스 구성

가상 머신에 대한 SSH 액세스 구성

다음 방법을 사용하여 VM에 대한 SSH 액세스를 구성할 수 있습니다.

- virtctl ssh 명령

  SSH 키 쌍을 생성하고 VM에 공개 키를 추가하고 개인 키로 virtctl ssh 명령을 실행하여 VM에 연결합니다.

  런타임 시 RHEL(Red Hat Enterprise Linux) 9 VM에 공개 SSH 키를 추가하거나 먼저 cloud-init 데이터 소스를 사용하여 구성할 수 있는 게스트 운영 체제를 사용하여 VM에 부팅할 수 있습니다.

- virtctl port-forward 명령

  virtctl port-foward 명령을 .ssh/config 파일에 추가하고 OpenSSH를 사용하여 VM에 연결합니다.

- Service

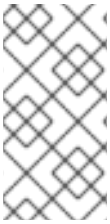
  서비스를 생성하고 서비스를 VM과 연결하고 서비스에서 노출하는 IP 주소 및 포트에 연결합니다.

- 보조 네트워크

  보조 네트워크를 구성하고 VM을 보조 네트워크 인터페이스에 연결한 다음 할당된 IP 주소에 연결합니다.

8.2. 가상 머신을 기본 POD 네트워크에 연결

masquerade 바인딩 모드를 사용하도록 네트워크 인터페이스를 구성하여 가상 머신을 기본 내부 Pod 네트워크에 연결할 수 있습니다.



참고

실시간 마이그레이션 중에 네트워크 인터페이스를 통해 기본 Pod 네트워크로 전달되는 트래픽이 중단됩니다.

8.2.1. 명령줄에서 가상 모드 구성

가상 모드를 사용하여 Pod IP 주소를 통해 나가는 가상 머신의 트래픽을 숨길 수 있습니다. 가상 모드에서는 NAT(Network Address Translation)를 사용하여 가상 머신을 Linux 브리지를 통해 Pod 네트워크 백엔드에 연결합니다.

가상 머신 구성 파일을 편집하여 가상 모드를 사용하도록 설정하고 트래픽이 가상 머신에 유입되도록 허용하십시오.

사전 요구 사항

- 가상 머신은 DHCP를 사용하여 IPv4 주소를 가져오도록 구성해야 합니다.

프로세스

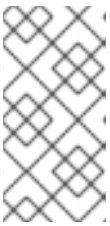
1. 가상 머신 구성 파일의 interfaces 스펙을 편집합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              masquerade: {} 1
              ports: 2
                - port: 80
# ...
      networks:
      - name: default
        pod: {}
```

1 가상 모드를 사용하여 연결합니다.

2

선택 사항: 가상 머신에서 노출할 포트를 나열합니다. 각각 **port** 필드에 지정된 포트를 나열합니다. **port** 값은 **0**에서 **65536** 사이의 숫자여야 합니다. 포트 배열을 사용하지 않으면 유효한 범위의 모든 포트가 들어오는 트래픽에 열려 있습니다. 이 예에서는 포트 **80** 에서 들어오는 트래픽이 허용됩니다.



참고

포트 **49152** 및 **49153**은 **libvirt** 플랫폼에서 사용하도록 예약되며 이러한 포트에 대한 기타 들어오는 트래픽은 모두 삭제됩니다.

2. 가상 머신을 생성합니다.

```
$ oc create -f <vm-name>.yaml
```

8.2.2. 듀얼 스택(IPv4 및 IPv6)을 사용하여 가상 모드 구성

cloud-init를 사용하여 기본 **pod** 네트워크에서 **IPv6** 및 **IPv4**를 모두 사용하도록 새 **VM**(가상 머신)을 구성할 수 있습니다.

가상 머신 인스턴스 구성의 **Network.pod.vmIPv6NetworkCIDR** 필드에는 **VM**의 정적 **IPv6** 주소와 게이트웨이 **IP** 주소가 결정됩니다. 이는 **virt-launcher Pod**에서 **IPv6** 트래픽을 가상 머신으로 라우팅하는 데 사용되며 외부적으로 사용되지 않습니다. **Network.pod.vmIPv6NetworkCIDR** 필드는 **CIDR(Classless Inter-Domain Routing)** 표기법으로 **IPv6** 주소 블록을 지정합니다. 기본값은 **fd10:0:2::2/120** 입니다. 네트워크 요구 사항에 따라 이 값을 편집할 수 있습니다.

가상 시스템이 실행 중이면 가상 시스템의 들어오고 나가는 트래픽이 **virt-launcher Pod**의 **IPv4** 주소와 고유한 **IPv6** 주소로 라우팅됩니다. 그런 다음 **virt-launcher Pod**는 **IPv4** 트래픽을 가상 시스템의 **DHCP** 주소로 라우팅하고 **IPv6** 트래픽을 가상 시스템의 **IPv6** 주소로 정적으로 설정합니다.

사전 요구 사항

- **OpenShift Container Platform** 클러스터는 듀얼 스택용으로 구성된 **OVN-Kubernetes CNI(Container Network Interface)** 네트워크 플러그인을 사용해야 합니다.

프로세스

1. 새 가상 시스템 구성에서 **masquerade**가 있는 인터페이스를 포함하고 **cloud-init**를 사용하

여 IPv6 주소 및 기본 게이트웨이를 구성합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm-ipv6
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              masquerade: {} 1
              ports:
                - port: 80 2
# ...
      networks:
      - name: default
        pod: {}
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth0:
                dhcp4: true
                addresses: [ fd10:0:2::2/120 ] 3
                gateway6: fd10:0:2::1 4
```

1

가상 모드를 사용하여 연결합니다.

2

포트 80에서 가상 머신으로 들어오는 트래픽을 허용합니다.

3

가상 머신 인스턴스 구성의 Network.pod.vmIPv6NetworkCIDR 필드에 의해 결정된 정적 IPv6 주소입니다. 기본값은 fd10:0:2::2/120 입니다.

4

가상 머신 인스턴스 구성의 Network.pod.vmIPv6NetworkCIDR 필드에 의해 결정된 게이트웨이 IP 주소입니다. 기본값은 fd10:0:2::1 입니다.

2. 네임스페이스에서 가상 머신을 생성합니다.

```
$ oc create -f example-vm-ipv6.yaml
```

검증

- IPv6가 구성되었는지 확인하려면 가상 시스템을 시작하고 가상 시스템 인스턴스의 인터페이스 상태를 확인하여 IPv6 주소가 있는지 확인합니다.

```
$ oc get vmi <vmi-name> -o jsonpath="{.status.interfaces[*].ipAddresses}"
```

8.2.3. 점보 프레임 지원 정보

OVN-Kubernetes CNI 플러그인을 사용하는 경우 기본 Pod 네트워크에 연결된 두 개의 VM(가상 머신) 간에 조각화되지 않은 점보 프레임 패킷을 보낼 수 있습니다. 점보 프레임에는 최대 전송 단위(MTU) 값이 1500바이트보다 큽니다.

VM은 다음 방법 중 하나로 클러스터 관리자가 설정한 클러스터 네트워크의 MTU 값을 자동으로 가져옵니다.

- libvirt: 게스트 OS에 에뮬레이션된 장치에서 PCI(Peripheral Component Interconnect) 구성 레지스터를 통해 들어오는 데이터를 해석할 수 있는 최신 버전의 VirtIO 드라이버가 있는 경우.
- DHCP: 게스트 DHCP 클라이언트에서 DHCP 서버 응답에서 MTU 값을 읽을 수 있는 경우

참고

VirtIO 드라이버가 없는 Windows VM의 경우 netsh 또는 유사한 도구를 사용하여 MTU를 수동으로 설정해야 합니다. Windows DHCP 클라이언트에서 MTU 값을 읽지 않기 때문입니다.

8.2.4. 추가 리소스

- 클러스터 네트워크의 MTU 변경
- 네트워크에 대한 MTU 최적화

8.3. 서비스를 사용하여 가상 머신 노출

Service 오브젝트를 생성하여 클러스터 또는 클러스터 외부에 가상 머신을 노출할 수 있습니다.

8.3.1. 서비스 정보

Kubernetes 서비스는 클라이언트의 네트워크 액세스를 포드 세트에서 실행되는 애플리케이션에 노출합니다. 서비스는 추상화, 로드 밸런싱 및 **NodePort** 및 **LoadBalancer** 유형의 경우 외부 세계에 노출을 제공합니다.

ClusterIP

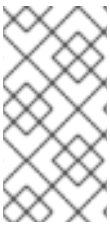
내부 **IP** 주소에 서비스를 노출하고 클러스터 내의 다른 애플리케이션에 **DNS** 이름으로 노출합니다. 단일 서비스는 여러 가상 머신에 매핑할 수 있습니다. 클라이언트가 서비스에 연결하려고 하면 사용 가능한 백엔드 간에 클라이언트 요청이 부하 분산됩니다. **ClusterIP** 는 기본 서비스 유형입니다.

NodePort

클러스터에서 선택한 각 노드의 동일한 포트에 서비스를 노출합니다. **NodePort** 를 사용하면 노드 자체에 클라이언트에서 외부에서 액세스할 수 있는 한 클러스터 외부에서 포트에 액세스할 수 있습니다.

LoadBalancer

현재 클라우드에 외부 로드 밸런서를 생성하고(지원되는 경우) 고정 외부 **IP** 주소를 서비스에 할당합니다.



참고

온프레미스 클러스터의 경우 **MetalLB Operator**를 배포하여 로드 밸런싱 서비스를 구성할 수 있습니다.

추가 리소스

- **MetalLB Operator** 설치
- **MetalLB**를 사용하도록 서비스 구성

8.3.2. 듀얼 스택 지원

클러스터에 대해 **IPv4** 및 **IPv6** 이중 스택 네트워킹을 사용하도록 설정한 경우 **Service** 개체에

spec.ipFamilyPolicy 및 **spec.ipFamilies** 필드를 정의하여 **IPv4, IPv6** 또는 둘 다 사용하는 서비스를 생성할 수 있습니다.

spec.ipFamilyPolicy 필드는 다음 값 중 하나로 설정할 수 있습니다.

SingleStack

컨트롤 플레인은 처음 구성된 서비스 클러스터 **IP** 범위를 기반으로 서비스의 클러스터 **IP** 주소를 할당합니다.

PreferDualStack

컨트롤 플레인은 듀얼 스택이 구성된 클러스터에서 서비스에 **IPv4** 및 **IPv6** 클러스터 **IP** 주소를 모두 할당합니다.

RequireDualStack

이 옵션은 듀얼 스택 네트워킹이 활성화되지 않은 클러스터에 실패합니다. 듀얼 스택이 구성된 클러스터의 경우 해당 동작은 값이 **PreferDualStack**으로 설정된 경우와 동일합니다. 컨트롤 플레인은 **IPv4** 및 **IPv6** 주소 범위의 클러스터 **IP** 주소를 할당합니다.

spec.ipFamilies 필드를 다음 배열 값 중 하나로 설정하여 단일 스택에 사용할 **IP** 제품군을 정의하거나 이중 스택의 **IP** 제품군 순서를 정의할 수 있습니다.

- **[IPv4]**
- **[IPv6]**
- **[IPv4, IPv6]**
- **[IPv6, IPv4]**

8.3.3. 명령줄을 사용하여 서비스 생성

명령줄을 사용하여 서비스를 생성하고 **VM**(가상 머신)과 연결할 수 있습니다.

사전 요구 사항

- 서비스를 지원하도록 클러스터 네트워크를 구성했습니다.

프로세스

1. VirtualMachine 매니페스트를 편집하여 서비스 생성 레이블을 추가합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  running: false
  template:
    metadata:
      labels:
        special: key (1)
# ...
```

(1) spec.template.metadata.labels 스탠자에 special: key 를 추가합니다.

참고

가상 머신의 라벨은 Pod로 전달됩니다. special: 키 레이블은 서비스 매니페스트의 spec.selector 속성의 레이블과 일치해야 합니다.

2. VirtualMachine 매니페스트 파일을 저장하여 변경 사항을 적용합니다.

3. VM을 노출하는 서비스 매니페스트를 생성합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  namespace: example-namespace
spec:
# ...
  selector:
    special: key (1)
  type: NodePort (2)
```

```
ports: 3
  protocol: TCP
  port: 80
  targetPort: 9376
  nodePort: 30000
```

1 VirtualMachine 매니페스트의 spec.template.metadata.labels 스탠자에 추가한 라벨을 지정합니다.

2 ClusterIP,NodePort 또는 LoadBalancer 를 지정합니다.

3 가상 머신에서 노출하려는 네트워크 포트 및 프로토콜 컬렉션을 지정합니다.

4. 서비스 매니페스트 파일을 저장합니다.

5. 다음 명령을 실행하여 서비스를 생성합니다.

```
$ oc create -f example-service.yaml
```

6. VM을 다시 시작하여 변경 사항을 적용합니다.

검증

- Service 오브젝트를 쿼리하여 사용 가능한지 확인합니다.

```
$ oc get service -n example-namespace
```

8.3.4. 추가 리소스

- NodePort를 사용하여 수신 클러스터 트래픽 구성
- 로드 밸런서를 사용하여 수신 클러스터 트래픽 구성

8.4. 내부 FQDN을 사용하여 가상 머신에 액세스

헤드리스 서비스를 사용하여 안정적인 FQDN(정규화된 도메인 이름)의 기본 내부 pod 네트워크에 연결된 VM(가상 머신)에 액세스할 수 있습니다.

Kubernetes *헤드리스 서비스*는 Pod 세트를 나타내는 클러스터 IP 주소를 할당하지 않는 일종의 서비스입니다. 서비스에 대한 단일 가상 IP 주소를 제공하는 대신 헤드리스 서비스는 서비스와 연결된 각 Pod에 대한 DNS 레코드를 생성합니다. 특정 TCP 또는 UDP 포트를 노출하지 않고도 FQDN을 통해 VM을 노출할 수 있습니다.

중요

OpenShift Container Platform 웹 콘솔을 사용하여 VM을 생성한 경우 VirtualMachine 세부 정보 페이지의 개요 탭에서 네트워크 타일에 나열된 내부 FQDN을 찾을 수 있습니다. VM에 연결하는 방법에 대한 자세한 내용은 내부 FQDN을 사용하여 가상 머신에 연결을 참조하십시오.

8.4.1. CLI를 사용하여 프로젝트에서 헤드리스 서비스 생성

네임스페이스에서 헤드리스 서비스를 생성하려면 clusterIP: None 매개변수를 서비스 YAML 정의에 추가합니다.

사전 요구 사항

- OpenShift CLI(oc)가 설치되어 있습니다.

프로세스

1. 다음 예와 같이 VM을 노출하는 서비스 매니페스트를 생성합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: mysubdomain (1)
spec:
  selector:
    expose: me (2)
  clusterIP: None (3)
  ports: (4)
  - protocol: TCP
    port: 1234
    targetPort: 1234
```

1. 서비스 이름입니다. **VirtualMachine** 매니페스트 파일의 **spec.subdomain** 속성과 일치해야 합니다.

2. 이 서비스 선택기는 **VirtualMachine** 매니페스트 파일의 **expose:me** 레이블과 일치해야 합니다.

3. 헤드리스 서비스를 지정합니다.

4. 서비스에서 노출하는 포트 목록입니다. 포트를 하나 이상 정의해야 합니다. 헤드리스 서비스에 영향을 미치지 않으므로 임의의 값이 될 수 있습니다.

2. 서비스 매니페스트 파일을 저장합니다.

3. 다음 명령을 실행하여 서비스를 생성합니다.

```
$ oc create -f headless_service.yaml
```

8.4.2. CLI를 사용하여 가상 머신을 헤드리스 서비스에 매핑

FQDN(정규화된 도메인 이름)을 사용하여 클러스터 내에서 **VM**(가상 머신)에 연결하려면 먼저 **VM**을 헤드리스 서비스에 매핑해야 합니다. **VM** 구성 파일에서 **spec.hostname** 및 **spec.subdomain** 매개변수를 설정합니다.

하위 도메인과 일치하는 이름이 있는 헤드리스 서비스가 있으면 **< vm.spec.hostname>.<vm.subdomain>.<vm.subdomain>.<vm.metadata.namespace>.svc.cluster.local** 형식의 **VM**에 대한 고유한 **DNS A** 레코드가 생성됩니다.

프로세스

1. 다음 명령을 실행하여 **VirtualMachine** 매니페스트를 편집하여 서비스 선택기 레이블 및 하위 도메인을 추가합니다.

```
$ oc edit vm <vm_name>
```

VirtualMachine 매니페스트 파일의 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
spec:
  template:
    metadata:
      labels:
        expose: me (1)
    spec:
      hostname: "myvm" (2)
      subdomain: "mysubdomain" (3)
# ...
```

(1) expose:me 레이블은 이전에 생성한 서비스 매니페스트의 spec.selector 속성과 일치해야 합니다.

(2) 이 속성을 지정하지 않으면 결과 DNS A 레코드는 < vm.metadata.name>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local 형식을 사용합니다.

(3) spec.subdomain 속성은 Service 오브젝트의 metadata.name 값과 일치해야 합니다.

2. 변경 사항을 저장하고 편집기를 종료합니다.

3. VM을 다시 시작하여 변경 사항을 적용합니다.

8.4.3. 내부 FQDN을 사용하여 가상 머신에 연결

FQDN(정규화된 도메인 이름)을 사용하여 VM(가상 머신)에 연결할 수 있습니다.

사전 요구 사항

- virtctl 툴을 설치했습니다.
- 웹 콘솔에서 VM의 내부 FQDN을 식별하거나 VM을 헤드리스 서비스에 매핑하여 확인했습니다. 내부 FQDN 형식은 < vm.spec.hostname>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local 입니다.

프로세스

1. 다음 명령을 입력하여 VM 콘솔에 연결합니다.

```
$ virtctl console vm-fedora
```

2. 요청된 FQDN을 사용하여 VM에 연결하려면 다음 명령을 실행합니다.

```
$ ping myvm.mysubdomain.<namespace>.svc.cluster.local
```

출력 예

```
PING myvm.mysubdomain.default.svc.cluster.local (10.244.0.57) 56(84) bytes of data.
64 bytes from myvm.mysubdomain.default.svc.cluster.local (10.244.0.57): icmp_seq=1
ttl=64 time=0.029 ms
```

이전 예에서 myvm.mysubdomain.default.svc.cluster.local 은 현재 VM에 할당된 클러스터 IP 주소인 10.244.0.57 을 가리킵니다.

8.4.4. 추가 리소스

- 서비스를 사용하여 VM 노출

8.5. 가상 머신을 LINUX 브리지 네트워크에 연결

기본적으로 OpenShift Virtualization은 단일 내부 pod 네트워크와 함께 설치됩니다.

다음 단계를 수행하여 Linux 브리지 네트워크를 생성하고 VM(가상 머신)을 네트워크에 연결할 수 있습니다.

1. Linux 브리지 노드 네트워크 구성 정책(NNCP)을 생성합니다.
2. 웹 콘솔 또는 명령줄 을 사용하여 Linux 브리지 네트워크 연결 정의(NAD)를 생성합니다.
3. 웹 콘솔 또는 명령줄 을 사용하여 CryostatD를 인식하도록 VM을 구성합니다.

8.5.1. Linux 브리지 NNCP 생성

Linux 브리지 네트워크에 대한 NodeNetworkConfigurationPolicy (NNCP) 매니페스트를 생성할 수 있습니다.

사전 요구 사항

- Kubernetes NMState Operator가 설치되어 있습니다.

프로세스

- NodeNetworkConfigurationPolicy 매니페스트를 생성합니다. 이 예제에는 고유한 정보로 교체해야 하는 샘플 값이 포함되어 있습니다.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy (1)
spec:
  desiredState:
    interfaces:
      - name: br1 (2)
        description: Linux bridge with eth1 as a port (3)
        type: linux-bridge (4)
        state: up (5)
```

```
    ipv4:
      enabled: false 6
    bridge:
      options:
        stp:
          enabled: false 7
      port:
        - name: eth1 8
```

1

정책 이름입니다.

2

인터페이스 이름입니다.

3

선택 사항: 사람이 읽을 수 있는 인터페이스 설명입니다.

4

인터페이스 유형입니다. 이 예제에서는 브리지를 만듭니다.

5

생성 후 인터페이스에 요청되는 상태입니다.

6

이 예에서는 **IPv4**를 비활성화합니다.

7

이 예에서는 **STP**를 비활성화합니다.

8

브리지가 연결된 노드 **NIC**입니다.

8.5.2. Linux 브리지 생성

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 **Linux** 브리지 네트워크 연결 정의(**NAD**)를 생성할 수 있습니다.

8.5.2.1. 웹 콘솔을 사용하여 Linux 브리지 생성

OpenShift Container Platform 웹 콘솔을 사용하여 네트워크 연결 정의(NAD)를 생성하여 Pod 및 가상 머신에 계층 2 네트워킹을 제공할 수 있습니다.

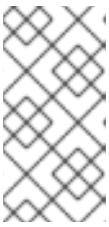
Linux 브리지 네트워크 연결 정의는 가상 머신을 VLAN에 연결하는 가장 효율적인 방법입니다.

주의

가상 머신의 네트워크 연결 정의에서 IP 주소 관리(IPAM) 구성은 지원되지 않습니다.

프로세스

1. 웹 콘솔에서 네트워킹 → NetworkAttachmentDefinitions 를 클릭합니다.
2. 네트워크 연결 정의 생성 을 클릭합니다.



참고

네트워크 연결 정의는 Pod 또는 가상 머신과 동일한 네임스페이스에 있어야 합니다.

3. 고유한 이름과 선택적 설명을 입력합니다.
4. 네트워크 유형 목록에서 CNV Linux 브리지 를 선택합니다.
5. 브리지 이름 필드에 브리지 이름을 입력합니다.
6. 선택 사항: 리소스에 VLAN ID가 구성된 경우 VLAN 태그 번호 필드에 ID 번호를 입력합니다.

7. 선택 사항: MAC 스푸핑 검사를 선택하여 MAC 스푸핑 필터링을 활성화합니다. 이 기능은 단일 MAC 주소만 Pod를 종료할 수 있도록 허용하여 MAC 스푸핑 공격에 대한 보안을 제공합니다.

8. 생성을 클릭합니다.

8.5.2.2. 명령줄을 사용하여 Linux 브리지 생성

명령줄을 사용하여 Pod 및 VM(가상 머신)에 계층 2 네트워킹을 제공하기 위해 네트워크 연결 정의(NAD)를 생성할 수 있습니다.

CryostatD와 VM은 동일한 네임스페이스에 있어야 합니다.

주의

가상 머신의 네트워크 연결 정의에서 IP 주소 관리(IPAM) 구성은 지원되지 않습니다.

사전 요구 사항

- 노드는 nftables를 지원해야 하며 MAC 스푸핑 검사를 사용하려면 nft 바이너리를 배포해야 합니다.

프로세스

1. 다음 예와 같이 NetworkAttachmentDefinition 구성에 VM을 추가합니다.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network (1)
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/bridge-interface (2)
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "bridge-network", (3)
```

```
 "type": "cnv-bridge", 4
 "bridge": "bridge-interface", 5
 "macspoofchk": true, 6
 "vlan": 100, 7
 "preserveDefaultVlan": false 8
}'
```

1

NetworkAttachmentDefinition 개체의 이름입니다.

2

선택 사항: 노드 선택을 위한 주석 키-값 쌍입니다. 여기서 **bridge-interface** 는 일부 노드에 구성된 브릿지 이름과 일치해야 합니다. 네트워크 연결 정의에 이 주석을 추가하면 **bridge-interface** 브리지가 연결된 노드에서만 가상 머신 인스턴스가 실행됩니다.

3

구성의 이름입니다. 구성 이름이 네트워크 연결 정의의 **name** 값과 일치하는 것이 좋습니다.

4

이 네트워크 연결 정의에 네트워크를 제공하는 **CNI(Container Network Interface)** 플러그인의 실제 이름입니다. 다른 **CNI**를 사용하려는 경우를 제외하고 이 필드를 변경하지 마십시오.

5

노드에 구성된 **Linux** 브리지의 이름입니다.

6

선택 사항: **MAC** 스푸핑 검사를 활성화하는 플래그입니다. **true**로 설정하면 **Pod** 또는 게스트 인터페이스의 **MAC** 주소를 변경할 수 없습니다. 이 속성은 단일 **MAC** 주소만 **Pod**를 종료할 수 있도록 허용하여 **MAC** 스푸핑 공격에 대한 보안을 제공합니다.

7

선택 사항: **VLAN** 태그. 노드 네트워크 구성 정책에 추가 **VLAN** 구성이 필요하지 않습니다.

8

선택 사항: **VM**이 기본 **VLAN**을 통해 브리지에 연결되는지 여부를 나타냅니다. 기본값은 **true**입니다.

참고

Linux 브리지 네트워크 연결 정의는 가상 머신을 VLAN에 연결하는 가장 효율적인 방법입니다.

2. 네트워크 연결 정의를 만듭니다.

```
$ oc create -f network-attachment-definition.yaml 1
```

1 여기서 network-attachment-definition.yaml 은 네트워크 연결 정의 매니페스트의 파일 이름입니다.

검증

- 다음 명령을 실행하여 네트워크 연결 정의가 생성되었는지 확인합니다.

```
$ oc get network-attachment-definition bridge-network
```

8.5.3. VM 네트워크 인터페이스 구성

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 VM(가상 머신) 네트워크 인터페이스를 구성할 수 있습니다.

8.5.3.1. 웹 콘솔을 사용하여 VM 네트워크 인터페이스 구성

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)의 네트워크 인터페이스를 구성할 수 있습니다.

사전 요구 사항

- 네트워크에 대한 네트워크 연결 정의를 생성했습니다.

프로세스

1. 가상화 → VirtualMachines 로 이동합니다.

2. **VM을 클릭하여 VirtualMachine 세부 정보 페이지를 확인합니다.**

3. **구성 탭에서 네트워크 인터페이스 탭을 클릭합니다.**

4. **네트워크 인터페이스 추가를 클릭합니다.**

5. **인터페이스 이름을 입력하고 네트워크 목록에서 네트워크 연결 정의를 선택합니다.**

6. **저장을 클릭합니다.**

7. **VM을 다시 시작하여 변경 사항을 적용합니다.**

네트워킹 필드

이름	설명
이름	네트워크 인터페이스 컨트롤러의 이름입니다.
모델	네트워크 인터페이스 컨트롤러의 모델을 나타냅니다. 지원되는 값은 **e1000e** 및 **virtio**입니다.
네트워크	사용 가능한 네트워크 연결 정의 목록입니다.
유형	사용 가능한 바인딩 방법 목록입니다. 네트워크 인터페이스에 적합한 바인딩 방법을 선택합니다. ● 기본 Pod 네트워크: **masquerade** ● Linux 브리지 네트워크: **bridge** ● SR-IOV 네트워크: **SR-IOV**
MAC 주소	네트워크 인터페이스 컨트롤러의 MAC 주소입니다. MAC 주소를 지정하지 않으면 주소가 자동으로 할당됩니다.

8.5.3.2. 명령줄을 사용하여 VM 네트워크 인터페이스 구성

명령줄을 사용하여 브리지 네트워크에 대한 VM(가상 머신) 네트워크 인터페이스를 구성할 수 있습니다.

사전 요구 사항

- 구성을 편집하기 전에 가상 머신을 종료합니다. 실행 중인 가상 머신을 편집하는 경우 변경 사항을 적용하려면 가상 머신을 다시 시작해야 합니다.

프로세스

1. 다음 예와 같이 브리지 인터페이스와 네트워크 연결 정의를 **VM** 구성에 추가합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - masquerade: {}
              name: default
            - bridge: {}
              name: bridge-net 1
# ...
      networks:
        - name: default
          pod: {}
        - name: bridge-net 2
          multus:
            networkName: a-bridge-network 3
```

1. 브리지 인터페이스의 이름입니다.

2. 네트워크의 이름입니다. 이 값은 해당 **spec.template.spec.domain.devices.interfaces** 항목의 **name** 값과 일치해야 합니다.

3. 네트워크 연결 정의의 이름입니다.

2. 설정을 적용합니다.

```
$ oc apply -f example-vm.yaml
```

3. 선택 사항: 실행 중인 가상 머신을 편집한 경우 변경 사항을 적용하려면 가상 머신을 다시 시작해야 합니다.

8.6. SR-IOV 네트워크에 가상 머신 연결

다음 단계를 수행하여 VM(가상 머신)을 SR-IOV(Single Root I/O Virtualization) 네트워크에 연결할 수 있습니다.

- SR-IOV 네트워크 장치 구성
- SR-IOV 네트워크 구성
- VM을 SR-IOV 네트워크에 연결

8.6.1. SR-IOV 네트워크 장치 구성

SR-IOV Network Operator는 SriovNetworkNodePolicy.sriovnetwork.openshift.io CustomResourceDefinition을 OpenShift Container Platform에 추가합니다. SriovNetworkNodePolicy CR(사용자 정의 리소스)을 만들어 SR-IOV 네트워크 장치를 구성할 수 있습니다.

참고

SriovNetworkNodePolicy 오브젝트에 지정된 구성을 적용하면 SR-IOV Operator가 노드를 비우고 경우에 따라 노드를 재부팅할 수 있습니다.

구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- SR-IOV Network Operator가 설치되어 있습니다.
- 비운 노드에서 제거된 워크로드를 처리하기 위해 클러스터에 사용 가능한 노드가 충분합니다.
- SR-IOV 네트워크 장치 구성에 대한 컨트롤 플레인 노드를 선택하지 않았습니다.

절차

1. SriovNetworkNodePolicy 오브젝트를 생성한 후 YAML을 <name>-sriov-node-network.yaml 파일에 저장합니다. <name>을 이 구성의 이름으로 바꿉니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  numVfs: <num> 7
  nicSelector: 8
    vendor: "<vendor_code>" 9
    deviceID: "<device_id>" 10
    pfNames: ["<pf_name>", ...] 11
    rootDevices: ["<pci_bus_id>", "..."] 12
  deviceType: vfio-pci 13
  isRdma: false 14
```

1. CR 오브젝트의 이름을 지정합니다.
2. SR-IOV Operator가 설치된 네임스페이스를 지정합니다.

3

SR-IOV 장치 플러그인의 리소스 이름을 지정합니다. 리소스 이름에 대해 여러 SriovNetworkNodePolicy 오브젝트를 생성할 수 있습니다.

4

구성할 노드를 선택하려면 노드 선택기를 지정합니다. 선택한 노드의 SR-IOV 네트워크 장치만 구성됩니다. SR-IOV CNI(Container Network Interface) 플러그인 및 장치 플러그인은 선택한 노드에만 배포됩니다.

5

선택 사항: 0에서 99 사이의 정수 값을 지정합니다. 숫자가 작을수록 우선 순위가 높아지므로 우선 순위 10은 우선 순위 99보다 높습니다. 기본값은 99입니다.

6

선택 사항: 가상 기능의 최대 전송 단위(MTU) 값을 지정합니다. 최대 MTU 값은 NIC 모델마다 다를 수 있습니다.

7

SR-IOV 물리적 네트워크 장치에 생성할 가상 기능(VF) 수를 지정합니다. Intel NIC(Network Interface Controller)의 경우 VF 수는 장치에서 지원하는 총 VF보다 클 수 없습니다. Mellanox NIC의 경우 VF 수는 128보다 클 수 없습니다.

8

nicSelector 매핑은 Operator가 구성할 이더넷 장치를 선택합니다. 모든 매개변수에 값을 지정할 필요는 없습니다. 의도하지 않게 이더넷 장치를 선택할 가능성을 최소화하기 위해 이더넷 어댑터를 충분히 정밀하게 식별하는 것이 좋습니다. rootDevices를 지정하면 vendor, deviceID 또는 pfNames의 값도 지정해야 합니다. pfNames와 rootDevices를 동시에 지정하는 경우 동일한 장치를 가리키는지 확인하십시오.

9

선택 사항: SR-IOV 네트워크 장치의 공급업체 16진 코드를 지정합니다. 허용되는 유일한 값은 8086 또는 15b3입니다.

10

선택 사항: SR-IOV 네트워크 장치의 장치 16진수 코드를 지정합니다. 허용되는 값은 158b, 1015, 1017입니다.

11

선택 사항:이 매개변수는 이더넷 장치에 대한 하나 이상의 PF(물리적 기능) 이름 배열을 허용합니다.

12

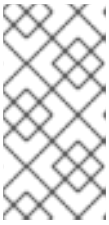
이 매개변수는 이더넷 장치의 물리적 기능을 위해 하나 이상의 **PCI** 버스 주소 배열을 허용합니다. 주소를 **0000:02: 00.1** 형식으로 입력합니다.

13

vfio-pci 드라이버 유형은 **OpenShift Virtualization**의 가상 기능에 필요합니다.

14

선택 사항: 원격 직접 메모리 액세스(RDMA) 모드 사용 여부를 지정합니다. **Mellanox** 카드의 경우 **isRdma**를 **false**로 설정합니다. 기본값은 **false**입니다.



참고

isRDMA 플래그가 **true**로 설정된 경우 **RDMA** 가능 **VF**를 일반 네트워크 장치로 계속 사용할 수 있습니다. 어느 모드에서나 장치를 사용할 수 있습니다.

2. 선택 사항: **SriovNetworkNodePolicy.Spec.NodeSelector** 를 사용하여 **SR-IOV** 가능 클러스터 노드에 레이블을 지정하지 않은 경우 레이블을 지정합니다. 노드 레이블 지정에 대한 자세한 내용은 "노드에서 라벨을 업데이트하는 방법 이해"를 참조하십시오.

3. **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f <name>-sriov-node-network.yaml
```

<name>은 이 구성의 이름을 지정합니다.

구성 업데이트를 적용하면 **sriov-network-operator** 네임스페이스의 모든 **Pod**가 **Running** 상태로 전환됩니다.

4. **SR-IOV** 네트워크 장치가 구성되어 있는지 확인하려면 다음 명령을 입력합니다. **<node_name>**을 방금 구성한 **SR-IOV** 네트워크 장치가 있는 노드 이름으로 바꿉니다.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

8.6.2. SR-IOV 추가 네트워크 구성

SriovNetwork 오브젝트를 생성하여 **SR-IOV** 하드웨어를 사용하는 추가 네트워크를 구성할 수 있습니다.

SriovNetwork 오브젝트를 생성하면 **SR-IOV Network Operator**가 **NetworkAttachmentDefinition** 오브젝트를 자동으로 생성합니다.

참고

SriovNetwork 오브젝트가 **running** 상태의 **Pod** 또는 가상 머신에 연결된 경우 **SriovNetwork** 오브젝트를 수정하거나 삭제하지 마십시오.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

절차

1. 다음 **SriovNetwork** 오브젝트를 생성한 후 **YAML**을 **<name>-sriov-network.yaml** 파일에 저장합니다. **<name>**을 이 추가 네트워크의 이름으로 변경합니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  networkNamespace: <target_namespace> 4
  vlan: <vlan> 5
  spoofChk: "<spoof_check>" 6
  linkState: <link_state> 7
  maxTxRate: <max_tx_rate> 8
  minTxRate: <min_rx_rate> 9
  vlanQoS: <vlan_qos> 10
  trust: "<trust_vf>" 11
  capabilities: <capabilities> 12
```

1

<name>을 오브젝트의 이름으로 바꿉니다. SR-IOV Network Operator는 동일한 이름으로 NetworkAttachmentDefinition 오브젝트를 생성합니다.

2

SR-IOV Network Operator가 설치된 네임스페이스를 지정합니다.

3

<sriov_resource_name>을 이 추가 네트워크에 대한 SR-IOV 하드웨어를 정의하는 SriovNetworkNodePolicy 오브젝트의 spec.resourceName 매개변수 값으로 바꿉니다.

4

<target_namespace>를 SriovNetwork의 대상 네임스페이스로 바꿉니다. 대상 네임스페이스의 pod 또는 가상 머신만 SriovNetwork에 연결할 수 있습니다.

5

선택 사항: <vlan>을 추가 네트워크의 VLAN(Virtual LAN) ID로 바꿉니다. 정수 값은 0에서 4095 사이여야 합니다. 기본값은 0입니다.

6

선택 사항: <spoof_check>를 VF의 위조 확인 모드로 바꿉니다. 허용되는 값은 문자열 "on" 및 "off"입니다.

중요

SR-IOV Network Operator가 지정한 값을 따옴표로 묶거나 CR을 거부해야 합니다.

7

선택 사항: <link_state>를 가상 기능(VF)의 링크 상태로 바꿉니다. 허용되는 값은 enable, disable 및 auto입니다.

8

선택 사항: VF의 경우 <max_tx_rate>를 최대 전송 속도(Mbps)로 바꿉니다.

9

참고

인텔 NIC는 minTxRate 매개변수를 지원하지 않습니다. 자세한 내용은 BZ#1772847에서 참조하십시오.

10

선택 사항: <vlan_qos>를 VF의 IEEE 802.1p 우선 순위 레벨로 바꿉니다. 기본값은 0입니다.

11

선택 사항: <trust_vf>를 VF의 신뢰 모드로 바꿉니다. 허용되는 값은 문자열 "on" 및 "off"입니다.

중요

SR-IOV Network Operator가 지정한 값을 따옴표로 묶거나 CR을 거부해야 합니다.

12

선택 사항: <capabilities>를 이 네트워크에 구성할 수 있는 기능으로 바꿉니다.

2. 오브젝트를 생성하려면 다음 명령을 입력합니다. <name>을 이 추가 네트워크의 이름으로 변경합니다.

```
$ oc create -f <name>-sriov-network.yaml
```

3. 선택 사항: 이전 단계에서 생성한 SriovNetwork 오브젝트에 연결된 NetworkAttachmentDefinition 오브젝트가 존재하는지 확인하려면 다음 명령을 입력합니다. <namespace>를 SriovNetwork 오브젝트에 지정한 네임스페이스로 바꿉니다.

```
$ oc get net-attach-def -n <namespace>
```

8.6.3. 명령줄을 사용하여 가상 머신을 SR-IOV 네트워크에 연결

VM 구성에 네트워크 세부 정보를 포함하여 VM(가상 머신)을 SR-IOV 네트워크에 연결할 수 있습니다.

프로세스

1. 다음 예와 같이 VM 구성의 spec.domain.devices.interfaces 및 spec.networks 스탠자에 SR-IOV 네트워크 세부 정보를 추가합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  domain:
    devices:
      interfaces:
      - name: default
        masquerade: {}
      - name: nic1 1
        sriov: {}
  networks:
  - name: default
    pod: {}
  - name: nic1 2
    multus:
        networkName: sriov-network 3
# ...
```

1. SR-IOV 인터페이스의 고유 이름을 지정합니다.

2. SR-IOV 인터페이스의 이름을 지정합니다. 이전에 정의한 interfaces.name과 동일해야 합니다.

3. SR-IOV 네트워크 연결 정의의 이름을 지정합니다.

2. 가상 머신 구성을 적용합니다.

```
$ oc apply -f <vm_sriov>.yaml 1
```

1. 가상 머신 YAML 파일의 이름입니다.

8.6.4. 웹 콘솔을 사용하여 SR-IOV 네트워크에 VM 연결

VM 구성에 네트워크 세부 정보를 포함하여 VM을 SR-IOV 네트워크에 연결할 수 있습니다.

사전 요구 사항

- 네트워크에 대한 네트워크 연결 정의를 생성해야 합니다.

프로세스

1. 가상화 → VirtualMachines 로 이동합니다.
2. VM을 클릭하여 VirtualMachine 세부 정보 페이지를 확인합니다.
3. 구성 탭에서 네트워크 인터페이스 탭을 클릭합니다.
4. 네트워크 인터페이스 추가를 클릭합니다.
5. 인터페이스 이름을 입력합니다.
6. 네트워크 목록에서 SR-IOV 네트워크 연결 정의를 선택합니다.
7. 유형 목록에서 SR-IOV 를 선택합니다.
8. 선택 사항: 네트워크 모델 또는 Mac 주소를 추가합니다.
9. 저장을 클릭합니다.
10. VM을 다시 시작하거나 실시간 마이그레이션하여 변경 사항을 적용합니다.

8.6.5. 추가 리소스

- 성능 향상을 위해 DPDK 워크로드 구성

8.7. SR-IOV와 함께 DPDK 사용

DPDK(Data Plane Development Kit)는 빠른 패킷 처리를 위한 라이브러리 및 드라이버 세트를 제공합니다.

SR-IOV 네트워크에서 DPDK 워크로드를 실행하도록 클러스터 및 VM(가상 머신)을 구성할 수 있습니다.

8.7.1. DPDK 워크로드에 대한 클러스터 구성

네트워크 성능을 개선하기 위해 DPDK(Data Plane Development Kit) 워크로드를 실행하도록 OpenShift Container Platform 클러스터를 구성할 수 있습니다.

사전 요구 사항

- cluster-admin 권한이 있는 사용자로 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(oc)가 설치되어 있습니다.
- SR-IOV Network Operator가 설치되어 있습니다.
- Node Tuning Operator가 설치되어 있습니다.

프로세스

1. 컴퓨팅 노드 토폴로지를 매핑하여 DPDK 애플리케이션에 대해 분리된 NUMA(Non-Uniform Memory Access) CPU와 운영 체제(OS)용으로 예약된 CPU를 결정합니다.
2. 사용자 지정 역할을 사용하여 컴퓨팅 노드의 하위 집합에 레이블을 지정합니다(예: worker-dpdk).

```
$ oc label node <node_name> node-role.kubernetes.io/worker-dpdk=""
```

3. spec.machineConfigSelector 오브젝트에 worker-dpdk 라벨이 포함된 새 MachineConfigPool 매니페스트를 생성합니다.

MachineConfigPool 매니페스트의 예

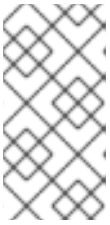
```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-dpdk
  labels:
    machineconfiguration.openshift.io/role: worker-dpdk
spec:
  machineConfigSelector:
    matchExpressions:
      - key: machineconfiguration.openshift.io/role
        operator: In
        values:
          - worker
          - worker-dpdk
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-dpdk: ""
```

4. 이전 단계에서 생성한 라벨이 지정된 노드 및 머신 구성 풀에 적용되는 PerformanceProfile 매니페스트를 생성합니다. 성능 프로필은 DPDK 애플리케이션용으로 분리된 CPU 및 하우스 보관을 위해 예약된 CPU를 지정합니다.

PerformanceProfile 매니페스트 예

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: profile-1
spec:
  cpu:
    isolated: 4-39,44-79
    reserved: 0-3,40-43
  globallyDisableIrqLoadBalancing: true
  hugepages:
    defaultHugepagesSize: 1G
    pages:
    - count: 8
```

```
    node: 0
    size: 1G
  net:
   userLevelNetworking: true
  nodeSelector:
   node-role.kubernetes.io/worker-dpdk: ""
  numa:
   topologyPolicy: single-numa-node
```



참고

MachineConfigPool 및 PerformanceProfile 매니페스트를 적용한 후 컴퓨팅 노드가 자동으로 다시 시작됩니다.

5. PerformanceProfile 오브젝트의 status.runtimeClass 필드에서 생성된 RuntimeClass 리소스의 이름을 검색합니다.

```
$ oc get performanceprofiles.performance.openshift.io profile-1 -
o=jsonpath='{.status.runtimeClass}{"\n"}'
```

6. HyperConverged CR(사용자 정의 리소스)을 편집하여 이전에 가져온 RuntimeClass 이름을 virt-launcher Pod의 기본 컨테이너 런타임 클래스로 설정합니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
   --type='json' -p='[{"op": "add", "path": "/spec/defaultRuntimeClass", "value":"
<runtimeclass-name>"}]'
```

참고

HyperConverged CR을 편집하면 변경 사항을 적용한 후 생성되는 모든 VM에 영향을 미치는 글로벌 설정이 변경됩니다.

7. DPDK 지원 컴퓨팅 노드에서 SMT(Simultaneous multithreading)를 사용하는 경우 HyperConverged CR을 편집하여 AlignCPUs enabler를 활성화합니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
   --type='json' -p='[{"op": "replace", "path": "/spec/featureGates/alignCPUs", "value":
true}]'
```

참고

AlignCPUs 를 활성화하면 **OpenShift Virtualization**에서 최대 두 개의 추가 전용 **CPU**를 요청하여 에뮬레이터 스레드 격리를 사용할 때 총 **CPU** 수를 균등하게 지정할 수 있습니다.

8. **spec.deviceType** 필드가 **vfio-pci** 로 설정된 **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

SriovNetworkNodePolicy 매니페스트의 예

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intel_nics_dpdk
  deviceType: vfio-pci
  mtu: 9000
  numVfs: 4
  priority: 99
  nicSelector:
    vendor: "8086"
    deviceID: "1572"
    pfNames:
      - eno3
    rootDevices:
      - "0000:19:00.2"
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

추가 리소스

- CPU 관리자 및 토폴로지 관리자 사용
- 대규모 페이지 구성
- 사용자 정의 머신 구성 풀 생성

8.7.2. DPDK 워크로드에 대한 프로젝트 구성

SR-IOV 하드웨어에서 DPDK 워크로드를 실행하도록 프로젝트를 구성할 수 있습니다.

사전 요구 사항

- 클러스터는 DPDK 워크로드를 실행하도록 구성되어 있습니다.

프로세스

1. DPDK 애플리케이션의 네임스페이스를 생성합니다.

```
$ oc create ns dpdk-checkup-ns
```

2. SriovNetwork NodePolicy 오브젝트를 참조하는 SriovNetwork 오브젝트를 생성합니다. SriovNetwork 오브젝트를 생성하면 SR-IOV Network Operator가 NetworkAttachmentDefinition 오브젝트를 자동으로 생성합니다.

SriovNetwork 매니페스트의 예

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-sriovnetwork
  namespace: openshift-sriov-network-operator
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "10.56.217.1"
    }
  networkNamespace: dpdk-checkup-ns (1)
  resourceName: intel_nics_dpdk (2)
  spoofChk: "off"
  trust: "on"
  vlan: 1019
```

1 NetworkAttachmentDefinition 오브젝트가 배포된 네임스페이스입니다.

2 DPDK 워크로드에 대한 클러스터를 구성할 때 생성된 SriovNetworkNodePolicy 오브젝트의 spec.resourceName 속성 값입니다.

3. 선택 사항: 가상 머신 대기 시간 검사를 실행하여 네트워크가 올바르게 구성되었는지 확인합니다.

4. 선택 사항: DPDK 검사를 실행하여 네임스페이스가 DPDK 워크로드에 대해 준비되었는지 확인합니다.

추가 리소스

- 노드 작업
- 가상 머신 대기 시간 점검
- DPDK 점검

8.7.3. DPDK 워크로드를 위한 가상 머신 구성

VM(가상 머신)에서 DPDK(Data Packet Development Kit) 워크로드를 실행하여 대기 시간을 줄이고 사용자 공간에서 더 빠른 패킷 처리를 위해 처리량을 높일 수 있습니다. DPDK는 하드웨어 기반 I/O 공유에 SR-IOV 네트워크를 사용합니다.

사전 요구 사항

- 클러스터는 DPDK 워크로드를 실행하도록 구성되어 있습니다.

- VM을 실행할 프로젝트를 생성하고 구성했습니다.

프로세스

1. SR-IOV 네트워크 인터페이스, CPU 토폴로지, CRI-O 주석 및 대규모 페이지에 대한 정보를 포함하도록 VirtualMachine 매니페스트를 편집합니다.

VirtualMachine 매니페스트의 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel-dpdk-vm
spec:
  running: true
  template:
    metadata:
      annotations:
        cpu-load-balancing.crio.io: disable (1)
        cpu-quota.crio.io: disable (2)
        irq-load-balancing.crio.io: disable (3)
    spec:
      domain:
        cpu:
          sockets: 1 (4)
          cores: 5 (5)
          threads: 2
          dedicatedCpuPlacement: true
          isolateEmulatorThread: true
        interfaces:
          - masquerade: {}
            name: default
          - model: virtio
            name: nic-east
            pciAddress: '0000:07:00.0'
            sriov: {}
          networkInterfaceMultiqueue: true
          rng: {}
      memory:
        hugepages:
          pageSize: 1Gi (6)
          guest: 8Gi
      networks:
        - name: default
          pod: {}
        - multus:
```

```
        networkName: dpdk-net 7
      name: nic-east
# ...
```

1

이 주석은 컨테이너에서 사용하는 CPU에 대해 로드 밸런싱이 비활성화되도록 지정합니다.

2

이 주석은 컨테이너에서 사용하는 CPU에 대해 CPU 할당량이 비활성화됨을 지정합니다.

3

이 주석은 컨테이너에서 사용하는 CPU에 대해 Interrupt Request(IRQ) 로드 밸런싱이 비활성화됨을 지정합니다.

4

VM 내부의 소켓 수입니다. 동일한 NUMA(Non-Uniform Memory Access) 노드에서 CPU를 예약하려면 이 필드를 1 로 설정해야 합니다.

5

VM 내부의 코어 수입니다. 이 값은 1 보다 크거나 같아야 합니다. 이 예에서는 VM이 5개의 하이퍼스레드 또는 10개의 CPU로 예약됩니다.

6

대규모 페이지의 크기입니다. x86-64 아키텍처에서 사용 가능한 값은 1Gi 및 2Mi입니다. 이 예에서 요청은 크기가 1Gi인 8개의 대규모 페이지에 대한 것입니다.

7

SR-IOV NetworkAttachmentDefinition 오브젝트의 이름입니다.

2. 편집기를 저장하고 종료합니다.

3. VirtualMachine 매니페스트를 적용합니다.

```
$ oc apply -f <file_name>.yaml
```

4. 게스트 운영 체제를 구성합니다. 다음 예제에서는 RHEL 8 OS의 구성 단계를 보여줍니다.

a. GRUB 부트로더 명령줄 인터페이스를 사용하여 대규모 페이지를 구성합니다. 다음 예제에서는 8개의 1G 대규모 페이지가 지정됩니다.

```
$ grubby --update-kernel=ALL --args="default_hugepagesz=1GB hugepagesz=1G
hugepages=8"
```

b. TuneD 애플리케이션에서 cpu-partitioning 프로필을 사용하여 대기 시간이 짧은 튜닝을 수행하려면 다음 명령을 실행합니다.

```
$ dnf install -y tuned-profiles-cpu-partitioning
```

```
$ echo isolated_cores=2-9 > /etc/tuned/cpu-partitioning-variables.conf
```

처음 두 CPU(0 및 1)는 하우스 유지 작업을 위해 별도로 설정되고 나머지 CPU는 DPDK 애플리케이션을 위해 격리됩니다.

```
$ tuned-adm profile cpu-partitioning
```

c. driverctl 장치 드라이버 제어 유틸리티를 사용하여 SR-IOV NIC 드라이버를 재정의합니다.

```
$ dnf install -y driverctl
```

```
$ driverctl set-override 0000:07:00.0 vfio-pci
```

5. VM을 다시 시작하여 변경 사항을 적용합니다.

8.8. 가상 머신을 OVN-KUBERNETES 보조 네트워크에 연결

VM(가상 머신)을 OVN(Open Virtual Network)-Kubernetes 보조 네트워크에 연결할 수 있습니다. OpenShift Virtualization은 OVN-Kubernetes의 계층 2 및 localnet 토폴로지를 지원합니다.

- 계층 2 토폴로지는 클러스터 전체 논리 스위치로 워크로드를 연결합니다. OVN-Kubernetes

CNI(Container Network Network Interface) 플러그인은 Geneve(Generic Network Virtualization Encapsulation) 프로토콜을 사용하여 노드 간에 오버레이 네트워크를 생성합니다. 이 오버레이 네트워크를 사용하여 추가 물리적 네트워킹 인프라를 구성하지 않고도 다른 노드의 VM을 연결할 수 있습니다.

- localnet 토폴로지는 보조 네트워크를 물리적 오버레이에 연결합니다. 이렇게 하면 east-west 클러스터 트래픽과 클러스터 외부에서 실행되는 서비스에 대한 액세스를 모두 사용할 수 있지만 클러스터 노드에서 기본 OVS(Open vSwitch) 시스템에 대한 추가 구성이 필요합니다.

참고

OVN-Kubernetes 보조 네트워크는 VM로의 트래픽 흐름을 제어하기 위해 MultiNetworkPolicy CRD(사용자 정의 리소스 정의)를 제공하는 다중 네트워크 정책 API와 호환됩니다. ipBlock 속성을 사용하여 특정 CIDR 블록에 대한 네트워크 정책 수신 및 송신 규칙을 정의할 수 있습니다.

OVN-Kubernetes 보조 네트워크를 구성하고 VM을 해당 네트워크에 연결하려면 다음 단계를 수행합니다.

1. 네트워크 연결 정의(NAD)를 생성하여 OVN-Kubernetes 보조 네트워크를 구성합니다.

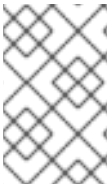
참고

localnet 토폴로지의 경우 CryostatD를 생성하기 전에 NodeNetworkConfigurationPolicy 오브젝트를 생성하여 OVS 브리지를 구성해야 합니다.

2. VM 사양에 네트워크 세부 정보를 추가하여 VM을 OVN-Kubernetes 보조 네트워크에 연결합니다.

8.8.1. OVN-Kubernetes CryostatD 생성

OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 OVN-Kubernetes 계층 2 또는 localnet 네트워크 연결 정의(NAD)를 생성할 수 있습니다.



참고

가상 머신의 네트워크 연결 정의에서 IP 주소 관리(IPAM) 구성은 지원되지 않습니다.

8.8.1.1. CLI를 사용하여 계층 2 토폴로지용 CryostatD 생성

포드를 계층 2 오버레이 네트워크에 연결하는 방법을 설명하는 네트워크 연결 정의(NAD)를 생성할 수 있습니다.

사전 요구 사항

- cluster-admin 권한이 있는 사용자로 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(oc)가 설치되어 있습니다.

프로세스

1. NetworkAttachmentDefinition 오브젝트를 생성합니다.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: l2-network
  namespace: my-namespace
spec:
  config: |2
    {
          "cniVersion": "0.3.1", 1
          "name": "my-namespace-l2-network", 2
          "type": "ovn-k8s-cni-overlay", 3
          "topology":"layer2", 4
          "mtu": 1300, 5
          "netAttachDefName": "my-namespace/l2-network" 6
    }
```

1 CNI 사양 버전입니다. 필수 값은 0.3.1 입니다.

2 네트워크의 이름입니다. 이 속성은 네임스페이스가 지정되지 않습니다. 예를 들어 두 개의 다른 네임스페이스에 존재하는 두 개의 다른 NetworkAttachmentDefinition 오브젝트

에서 참조되는 **l2-network** 라는 네트워크가 있을 수 있습니다. 이 기능은 다른 네임스페이스의 **VM**을 연결하는 데 유용합니다.

3

구성할 **CNI** 플러그인의 이름입니다. 필수 값은 **ovn-k8s-cni-overlay** 입니다.

4

네트워크의 토폴로지 구성입니다. 필요한 값은 **layer2** 입니다.

5

선택 사항: **MTU**(최대 전송 단위) 값입니다. 기본값은 커널에 의해 자동으로 설정됩니다.

6

NetworkAttachmentDefinition 오브젝트의 **metadata** 스탠자에 있는 **namespace** 및 **name** 필드의 값입니다.

참고

위의 예제에서는 서브넷이 정의되지 않은 클러스터 전체 오버레이를 구성합니다. 즉, 네트워크를 구현하는 논리 스위치는 계층 **2** 통신만 제공합니다. 고정 **IP** 주소를 설정하거나 동적 **IP** 주소에 대해 네트워크에 **DHCP** 서버를 배포하여 가상 머신을 생성할 때 **IP** 주소를 구성해야 합니다.

2. 매니페스트를 적용합니다.

```
$ oc apply -f <filename>.yaml
```

8.8.1.2. CLI를 사용하여 localnet 토폴로지용 CryostatD 생성

기본 물리적 네트워크에 **Pod**를 연결하는 방법을 설명하는 네트워크 연결 정의(**NAD**)를 생성할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 사용자로 클러스터에 액세스할 수 있습니다.

- OpenShift CLI(oc)가 설치되어 있습니다.
- Kubernetes NMState Operator가 설치되어 있습니다.
- OVN-Kubernetes 보조 네트워크를 OVS(Open vSwitch) 브리지에 매핑하는 NodeNetworkConfigurationPolicy 오브젝트를 생성했습니다.

프로세스

1. NetworkAttachmentDefinition 오브젝트를 생성합니다.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: localnet-network
  namespace: default
spec:
  config: |2
    {
          "cniVersion": "0.3.1", (1)
          "name": "localnet-network", (2)
          "type": "ovn-k8s-cni-overlay", (3)
          "topology": "localnet", (4)
          "netAttachDefName": "default/localnet-network" (5)
    }
```

(1) CNI 사양 버전입니다. 필수 값은 0.3.1 입니다.

(2) 네트워크의 이름입니다. 이 속성은 OVS 브리지 매핑을 정의하는 NodeNetworkConfigurationPolicy 오브젝트의 spec.desiredState.ovn.bridge-mappings.localnet 필드의 값과 일치해야 합니다.

(3) 구성할 CNI 플러그인의 이름입니다. 필수 값은 ovn-k8s-cni-overlay 입니다.

(4) 네트워크의 토폴로지 구성입니다. 필요한 값은 localnet 입니다.

5 NetworkAttachmentDefinition 오브젝트의 metadata 스탠자에 있는 namespace 및 name 필드의 값입니다.

2. 매니페스트를 적용합니다.

```
$ oc apply -f <filename>.yaml
```

8.8.2. OVN-Kubernetes 보조 네트워크에 가상 머신 연결

OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 VM(가상 머신)을 OVN-Kubernetes 보조 네트워크 인터페이스에 연결할 수 있습니다.

8.8.2.1. CLI를 사용하여 OVN-Kubernetes 보조 네트워크에 가상 머신 연결

VM 구성에 네트워크 세부 정보를 포함하여 VM(가상 머신)을 OVN-Kubernetes 보조 네트워크에 연결할 수 있습니다.

사전 요구 사항

- cluster-admin 권한이 있는 사용자로 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(oc)가 설치되어 있습니다.

프로세스

1. 다음 예와 같이 VirtualMachine 매니페스트를 편집하여 OVN-Kubernetes 보조 네트워크 인터페이스 세부 정보를 추가합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-server
spec:
  running: true
  template:
    spec:
      domain:
        devices:
```

```
        interfaces:
        - name: default
          masquerade: {}
        - name: secondary 1
          bridge: {}
      resources:
        requests:
          memory: 1024Mi
    networks:
    - name: default
      pod: {}
    - name: secondary 2
      multus:
        networkName: <nad_name> 3
# ...
```

1 OVN-Kubernetes 보조 인터페이스의 이름입니다.

2 네트워크의 이름입니다. spec.template.spec.domain.devices.interfaces.name 필드의 값과 일치해야 합니다.

3 NetworkAttachmentDefinition 오브젝트의 이름입니다.

2. VirtualMachine 매니페스트를 적용합니다.

```
$ oc apply -f <filename>.yaml
```

3. 선택 사항: 실행 중인 가상 머신을 편집한 경우 변경 사항을 적용하려면 가상 머신을 다시 시작해야 합니다.

8.8.2.2. 웹 콘솔을 사용하여 계층 2 토폴로지용 CryostatD 생성

포드를 계층 2 오버레이 네트워크에 연결하는 방법을 설명하는 네트워크 연결 정의(NAD)를 생성할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 사용자로 클러스터에 액세스할 수 있습니다.

프로세스

1. 웹 콘솔에서 네트워킹 → **NetworkAttachmentDefinitions** 로 이동합니다.
2. 네트워크 연결 정의 생성 을 클릭합니다. 네트워크 연결 정의는 이를 사용하는 **Pod** 또는 가상 머신과 동일한 네임스페이스에 있어야 합니다.
3. 고유한 이름과 선택적 설명을 입력합니다.
4. 네트워크 유형 목록에서 **OVN Kubernetes L2** 오버레이 네트워크를 선택합니다.
5. 생성을 클릭합니다.

8.8.2.3. 웹 콘솔을 사용하여 localnet 토폴로지용 CryostatD 생성

OpenShift Container Platform 웹 콘솔을 사용하여 워크로드를 물리적 네트워크에 연결하는 네트워크 연결 정의(**NAD**)를 생성할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 사용자로 클러스터에 액세스할 수 있습니다.
- **nmstate** 를 사용하여 **localnet**을 **OVS** 브리지 매핑으로 구성합니다.

프로세스

1. 웹 콘솔에서 네트워킹 → **NetworkAttachmentDefinitions** 로 이동합니다.
2. 네트워크 연결 정의 생성 을 클릭합니다. 네트워크 연결 정의는 이를 사용하는 **Pod** 또는 가상 머신과 동일한 네임스페이스에 있어야 합니다.

3. 고유한 이름과 선택적 설명을 입력합니다.

4. 네트워크 유형 목록에서 **OVN Kubernetes 보조 localnet 네트워크**를 선택합니다.

5. 브리지 매핑 필드에 사전 구성된 **localnet** 식별자의 이름을 입력합니다.

6. 선택 사항: **MTU**를 지정된 값으로 명시적으로 설정할 수 있습니다. 커널에서 기본값을 선택합니다.

7. 선택 사항: **VLAN**에 트래픽을 캡슐화합니다. 기본값은 **None**입니다.

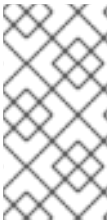
8. 생성을 클릭합니다.

8.8.3. 추가 리소스

- OVN-Kubernetes 추가 네트워크 구성
- Kubernetes NMState Operator 정보
- OVN-Kubernetes 추가 네트워크 매핑 구성
- 추가 네트워크 연결 구성

8.9. 보조 네트워크 인터페이스 핫플러그

VM(가상 머신)을 중지하지 않고 보조 네트워크 인터페이스를 추가하거나 제거할 수 있습니다. OpenShift Virtualization에서는 VirtIO 장치 드라이버를 사용하는 보조 인터페이스에 대한 핫 플러그를 지원합니다.

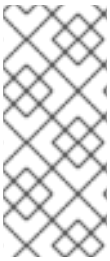


참고

SR-IOV(Single Root I/O Virtualization) 인터페이스에 핫 연결 해제는 지원되지 않습니다.

8.9.1. virtio 제한 사항

각 VirtIO 인터페이스는 VM의 제한된 PCI(Peripheral Connect Interface) 슬롯 중 하나를 사용합니다. 총 32개의 슬롯이 준비되어 있습니다. PCI 슬롯은 다른 장치에서도 사용되며 미리 예약해야 하므로 필요에 따라 슬롯을 사용할 수 없습니다. OpenShift Virtualization은 핫플러그 인터페이스를 위해 최대 4개의 슬롯을 예약합니다. 여기에는 기존의 플러그인 네트워크 인터페이스가 포함됩니다. 예를 들어 VM에 두 개의 기존 인터페이스가 있는 경우 두 개의 네트워크 인터페이스를 핫플러그할 수 있습니다.



참고

핫 플러그에 사용할 수 있는 실제 슬롯 수는 머신 유형에 따라 다릅니다. 예를 들어 q35 시스템 유형의 기본 PCI 토폴로지는 핫 플러그를 하나의 추가 PCIe 장치를 지원합니다. PCI 토폴로지 및 핫 플러그 지원에 대한 자세한 내용은 libvirt 설명서 를 참조하십시오.

인터페이스를 핫플러그한 후 VM을 다시 시작하면 해당 인터페이스가 표준 네트워크 인터페이스의 일부가 됩니다.

8.9.2. CLI를 사용하여 보조 네트워크 인터페이스 핫플러그

VM이 실행되는 동안 보조 네트워크 인터페이스를 VM(가상 머신)에 핫플러그합니다.

사전 요구 사항

- 네트워크 연결 정의는 VM과 동일한 네임스페이스에 구성됩니다.
- virtctl 툴을 설치했습니다.
- OpenShift CLI(oc)가 설치되어 있습니다.

프로세스

1. 네트워크 인터페이스를 핫플러그할 VM이 실행되지 않는 경우 다음 명령을 사용하여 시작합

니다.

```
$ virtctl start <vm_name> -n <namespace>
```

2. 다음 명령을 사용하여 실행 중인 VM에 새 네트워크 인터페이스를 추가합니다. VM 사양을 편집하면 새 네트워크 인터페이스가 VM 및 VMI(가상 머신 인스턴스) 구성에 추가되지만 실행 중인 VM에 연결되지는 않습니다.

```
$ oc edit vm <vm_name>
```

VM 구성 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
template:
  spec:
    domain:
      devices:
        interfaces:
        - name: defaultnetwork
          masquerade: {}
        # new interface
        - name: <secondary_nic> 1
          bridge: {}
    networks:
    - name: defaultnetwork
      pod: {}
    # new network
    - name: <secondary_nic> 2
      multus:
        networkName: <nad_name> 3
# ...
```

1 새 네트워크 인터페이스의 이름을 지정합니다.

2 네트워크의 이름을 지정합니다. template.spec.domain.devices.interfaces 목록에 정의된 새 네트워크 인터페이스의 이름과 동일해야 합니다.

3 NetworkAttachmentDefinition 오브젝트의 이름을 지정합니다.

3. 실행 중인 VM에 네트워크 인터페이스를 연결하려면 다음 명령을 실행하여 VM을 실시간 마이그레이션합니다.

```
$ virtctl migrate <vm_name>
```

검증

1. 다음 명령을 사용하여 VM 실시간 마이그레이션이 성공했는지 확인합니다.

```
$ oc get VirtualMachineInstanceMigration -w
```

출력 예

```
NAME                        PHASE          VMI
kubevirt-migrate-vm-lj62q   Scheduling     vm-fedora
kubevirt-migrate-vm-lj62q   Scheduled      vm-fedora
kubevirt-migrate-vm-lj62q   PreparingTarget   vm-fedora
kubevirt-migrate-vm-lj62q   TargetReady    vm-fedora
kubevirt-migrate-vm-lj62q   Running        vm-fedora
kubevirt-migrate-vm-lj62q   Succeeded      vm-fedora
```

2. VMI 상태를 확인하여 새 인터페이스가 VM에 추가되었는지 확인합니다.

```
$ oc get vmi vm-fedora -ojsonpath="{ @.status.interfaces }"
```

출력 예

```
[
 {
  "infoSource": "domain, guest-agent",
  "interfaceName": "eth0",
  "ipAddress": "10.130.0.195",
  "ipAddresses": [
```

```
      "10.130.0.195",
      "fd02:0:0:3::43c"
     ],
     "mac": "52:54:00:0e:ab:25",
     "name": "default",
     "queueCount": 1
    },
    {
     "infoSource": "domain, guest-agent, multus-status",
     "interfaceName": "eth1",
     "mac": "02:d8:b8:00:00:2a",
     "name": "bridge-interface", 1
     "queueCount": 1
    }
]
```


1

VMI 상태에 핫플러그 인터페이스가 표시됩니다.

8.9.3. CLI를 사용하여 보조 네트워크 인터페이스 핫플러그

실행 중인 VM(가상 머신)에서 보조 네트워크 인터페이스를 제거할 수 있습니다.

참고

SR-IOV(Single Root I/O Virtualization) 인터페이스에 핫 연결 해제는 지원되지 않습니다.

사전 요구 사항

- VM이 실행 중이어야 합니다.
- OpenShift Virtualization 4.14 이상을 실행하는 클러스터에서 VM을 생성해야 합니다.
- VM에 브리지 네트워크 인터페이스가 연결되어 있어야 합니다.

프로세스

1. VM 사양을 편집하여 보조 네트워크 인터페이스를 핫플러그합니다. 인터페이스 상태를 **absent** 로 설정하면 게스트에서 네트워크 인터페이스를 분리하지만 인터페이스는 여전히 Pod에 있습니다.

```
$ oc edit vm <vm_name>
```

VM 구성 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
template:
  spec:
    domain:
      devices:
        interfaces:
          - name: defaultnetwork
            masquerade: {}
          # set the interface state to absent
          - name: <secondary_nic>
            state: absent (1)
            bridge: {}
    networks:
      - name: defaultnetwork
        pod: {}
      - name: <secondary_nic>
        multus:
          networkName: <nad_name>
# ...
```

(1) 인터페이스 상태를 **absent** 로 설정하여 실행 중인 VM에서 분리합니다. VM 사양에서 인터페이스 세부 정보를 제거해도 보조 네트워크 인터페이스를 핫플러그하지 않습니다.

2. VM을 마이그레이션하여 Pod에서 인터페이스를 제거합니다.

```
$ virtctl migrate <vm_name>
```

8.9.4. 추가 리소스

- virtctl 설치
- Linux 브리지 네트워크 연결 정의 생성
- 가상 머신을 Linux 브리지 네트워크에 연결
- SR-IOV 네트워크 연결 정의 생성
- SR-IOV 네트워크에 가상 머신 연결

8.10. 서비스 메시에 가상 머신 연결

OpenShift Virtualization이 OpenShift Service Mesh와 통합되었습니다. IPv4를 사용하여 기본 Pod 네트워크에서 가상 머신 워크로드를 실행하는 Pod 간 트래픽을 모니터링, 시각화 및 제어할 수 있습니다.

8.10.1. 서비스 메시에 가상 머신 추가

VM(가상 머신) 워크로드를 서비스 메시에 추가하려면 sidecar.istio.io/inject 주석을 true 로 설정하여 VM 구성 파일에서 자동 사이드카 삽입을 활성화합니다. 그런 다음 VM을 서비스로 노출하여 메시에서 애플리케이션을 확인합니다.

중요

포트 충돌을 방지하려면 Istio 사이드카 프록시에서 사용하는 포트를 사용하지 마십시오. 여기에는 포트 15000, 15001, 15006, 15008, 15020, 15021 및 15090이 포함됩니다.

사전 요구 사항

- Service Mesh Operator가 설치되어 있어야 합니다.
- Service Mesh Control Plane을 생성하셨습니다.
- VM 프로젝트를 서비스 메시 멤버 롤에 추가했습니다.

프로세스

1. VM 구성 파일을 편집하여 **sidecar.istio.io/inject: "true"** 주석을 추가합니다.

설정 파일 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-istio
  name: vm-istio
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-istio
        app: vm-istio (1)
      annotations:
        sidecar.istio.io/inject: "true" (2)
    spec:
      domain:
        devices:
          interfaces:
          - name: default
            masquerade: {} (3)
          disks:
          - disk:
              bus: virtio
            name: containerdisk
          - disk:
              bus: virtio
            name: cloudinitdisk
        resources:
          requests:
            memory: 1024M
      networks:
      - name: default
        pod: {}
      terminationGracePeriodSeconds: 180
      volumes:
      - containerDisk:
          image: registry:5000/kubevirt/fedora-cloud-container-disk-demo:devel
        name: containerdisk
```

(1)

2

자동 사이드카 삽입을 활성화하는 주석입니다.

3

기본 Pod 네트워크와 함께 사용할 바인딩 방법(masquerade 모드)입니다.

2. VM 구성을 적용합니다.

```
$ oc apply -f <vm_name>.yaml 1
```

1

가상 머신 YAML 파일의 이름입니다.

3. VM을 서비스 메시에 노출하는 서비스 오브젝트를 생성합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: vm-istio
spec:
  selector:
    app: vm-istio 1
  ports:
    - port: 8080
      name: http
      protocol: TCP
```

1

서비스에서 대상으로 하는 Pod 세트를 결정하는 서비스 선택기입니다. 이 속성은 VM 구성 파일의 spec.metadata.labels 필드에 해당합니다. 위의 예에서 vm-istio라는 Service 오브젝트는 app= vm-istio 레이블이 있는 모든 Pod에서 TCP 포트 8080을 대상으로 합니다.

4. 서비스를 생성합니다.

```
$ oc create -f <service_name>.yaml 1
```

1

서비스 YAML 파일의 이름입니다.

8.10.2. 추가 리소스

- Service Mesh Operator 설치
- Service Mesh Control Plane 생성
- 서비스 메시 멤버 롤에 프로젝트 추가

8.11. 실시간 마이그레이션을 위한 전용 네트워크 구성

실시간 마이그레이션을 위해 전용 Multus 네트워크를 구성할 수 있습니다. 전용 네트워크는 실시간 마이그레이션 중에 테넌트 워크로드에 대한 네트워크 포화 상태로 인한 영향을 최소화합니다.

8.11.1. 실시간 마이그레이션을 위한 전용 보조 네트워크 구성

실시간 마이그레이션을 위해 전용 보조 네트워크를 구성하려면 먼저 CLI를 사용하여 브리지 네트워크 연결 정의(NAD)를 생성해야 합니다. 그런 다음 NetworkAttachmentDefinition 오브젝트의 이름을 HyperConverged CR(사용자 정의 리소스)에 추가합니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.
- cluster-admin 역할의 사용자로 클러스터에 로그인했습니다.
- 각 노드에는 NIC(네트워크 인터페이스 카드)가 두 개 이상 있습니다.
- 실시간 마이그레이션의 NIC는 동일한 VLAN에 연결됩니다.

프로세스

1. 다음 예에 따라 NetworkAttachmentDefinition 매니페스트를 생성합니다.

설정 파일 예

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network 1
  namespace: openshift-cnv 2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", 3
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", 4
      "range": "10.200.5.0/24" 5
    }
  }'
```

1 NetworkAttachmentDefinition 오브젝트의 이름을 지정합니다.

2 3 실시간 마이그레이션에 사용할 NIC의 이름을 지정합니다.

4 CryostatD에 대한 네트워크를 제공하는 CNI 플러그인의 이름을 지정합니다.

5 보조 네트워크의 IP 주소 범위를 지정합니다. 이 범위는 기본 네트워크의 IP 주소를 겹치지 않아야 합니다.

2. 다음 명령을 실행하여 기본 편집기에서 HyperConverged CR을 엽니다.

```
oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

3. HyperConverged CR의 spec.liveMigrationConfig 스탠자에 NetworkAttachmentDefinition 오브젝트의 이름을 추가합니다.

HyperConverged 매니페스트의 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    network: <network> 1
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
# ...
```

1 실시간 마이그레이션에 사용할 Multus NetworkAttachmentDefinition 오브젝트의 이름을 지정합니다.

4. 변경 사항을 저장하고 편집기를 종료합니다. virt-handler Pod가 다시 시작되고 보조 네트워크에 연결합니다.

검증

- 가상 머신이 실행되는 노드가 유지보수 모드로 배치되면 VM이 클러스터의 다른 노드로 자동으로 마이그레이션됩니다. VMI(가상 머신 인스턴스) 메타데이터에서 대상 IP 주소를 확인하여 마이그레이션이 기본 pod 네트워크가 아닌 보조 네트워크를 통해 발생했는지 확인할 수 있습니다.

```
$ oc get vmi <vmi_name> -o jsonpath='{.status.migrationState.targetNodeAddress}'
```

8.11.2. 웹 콘솔을 사용하여 전용 네트워크 선택

OpenShift Container Platform 웹 콘솔을 사용하여 실시간 마이그레이션 전용 네트워크를 선택할 수 있습니다.

사전 요구 사항

- 실시간 마이그레이션을 위해 Multus 네트워크를 구성했습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에서 가상화 > 개요 로 이동합니다.
2. 설정 탭을 클릭한 다음 실시간 마이그레이션 을 클릭합니다.
3. 실시간 마이그레이션 네트워크 목록에서 네트워크를 선택합니다.

8.11.3. 추가 리소스

- 실시간 마이그레이션 제한 및 타임아웃 구성

8.12. IP 주소 구성 및 보기

VM(가상 머신)을 생성할 때 IP 주소를 구성할 수 있습니다. IP 주소는 cloud-init를 사용하여 프로비저닝됩니다.

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 VM의 IP 주소를 볼 수 있습니다. 네트워크 정보는 QEMU 게스트 에이전트에 의해 수집됩니다.

8.12.1. 가상 머신용 IP 주소 구성

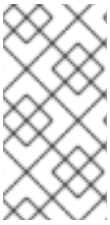
웹 콘솔 또는 명령줄을 사용하여 VM(가상 머신)을 생성할 때 고정 IP 주소를 구성할 수 있습니다.

명령줄을 사용하여 VM을 생성할 때 동적 IP 주소를 구성할 수 있습니다.

IP 주소는 cloud-init를 사용하여 프로비저닝됩니다.

8.12.1.1. 명령줄을 사용하여 가상 머신을 생성할 때 IP 주소 구성

VM(가상 머신)을 생성할 때 정적 또는 동적 IP 주소를 구성할 수 있습니다. IP 주소는 cloud-init를 사용하여 프로비저닝됩니다.



참고

VM이 Pod 네트워크에 연결되어 있으면 업데이트하지 않는 한 Pod 네트워크 인터페이스가 기본 경로입니다.

사전 요구 사항

- 가상 머신이 보조 네트워크에 연결되어 있습니다.
- 가상 머신의 동적 IP를 구성하기 위해 보조 네트워크에 DHCP 서버를 사용할 수 있습니다.

프로세스

- 가상 머신 구성의 spec.template.spec.volumes.cloudInitNoCloud.networkData 스탠자를 편집합니다.
  - 동적 IP 주소를 구성하려면 인터페이스 이름을 지정하고 DHCP를 활성화합니다.

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
    spec:
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth1: 1
                dhcp4: true
```

1

- 고정 IP를 구성하려면 인터페이스 이름과 IP 주소를 지정합니다.

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
    spec:
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth1: (1)
                addresses:
                - 10.10.10.14/24 (2)
```

(1) 인터페이스 이름을 지정합니다.

(2) 고정 IP 주소를 지정합니다.

8.12.2. 가상 머신의 IP 주소 보기

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 VM의 IP 주소를 볼 수 있습니다.

네트워크 정보는 QEMU 게스트 에이전트에 의해 수집됩니다.

8.12.2.1. 웹 콘솔을 사용하여 가상 머신의 IP 주소 보기

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)의 IP 주소를 볼 수 있습니다.

참고

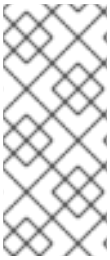
보조 네트워크 인터페이스의 IP 주소를 보려면 VM에 QEMU 게스트 에이전트를 설치해야 합니다. Pod 네트워크 인터페이스에는 QEMU 게스트 에이전트가 필요하지 않습니다.

프로세스

1. OpenShift Container Platform 콘솔의 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 세부 정보 탭을 클릭하여 IP 주소를 확인합니다.

8.12.2.2. 명령줄을 사용하여 가상 머신의 IP 주소 보기

명령줄을 사용하여 VM(가상 머신)의 IP 주소를 볼 수 있습니다.



참고

보조 네트워크 인터페이스의 IP 주소를 보려면 VM에 QEMU 게스트 에이전트를 설치해야 합니다. Pod 네트워크 인터페이스에는 QEMU 게스트 에이전트가 필요하지 않습니다.

프로세스

- 다음 명령을 실행하여 가상 머신 인스턴스 구성을 가져옵니다.

```
$ oc describe vmi <vmi_name>
```

출력 예

```
# ...
Interfaces:
  Interface Name:  eth0
  Ip Address:      10.244.0.37/24
  Ip Addresses:
```

```
  10.244.0.37/24
  fe80::858:aff:fef4:25/64
Mac:          0a:58:0a:f4:00:25
Name:         default
Interface Name:  v2
Ip Address:      1.1.1.7/24
Ip Addresses:
  1.1.1.7/24
  fe80::f4d9:70ff:fe13:9089/64
Mac:          f6:d9:70:13:90:89
Interface Name:  v1
Ip Address:      1.1.1.1/24
Ip Addresses:
  1.1.1.1/24
  1.1.1.2/24
  1.1.1.4/24
  2001:de7:0:f101::1/64
  2001:db8:0:f101::1/64
  fe80::1420:84ff:fe10:17aa/64
Mac:          16:20:84:10:17:aa
```

8.12.3. 추가 리소스

- QEMU 게스트 에이전트 설치

8.13. 외부 FQDN을 사용하여 가상 머신에 액세스

FQDN(정규화된 도메인 이름)을 사용하여 클러스터 외부에서 보조 네트워크 인터페이스에 연결된 VM(가상 머신)에 액세스할 수 있습니다.

중요

FQDN을 사용하여 클러스터 외부에서 VM에 액세스하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

8.13.1. 보조 네트워크에 대한 DNS 서버 구성

CNAO(Cluster Network Addons Operator)는 HyperConverged CR(사용자 정의 리소스)에서 deployKubeSecondaryDNS 기능 게이트를 활성화할 때 DNS(Domain Name Server) 서버 및 모니터링 구성 요소를 배포합니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.
- 클러스터에 대한 로드 밸런서를 구성했습니다.
- cluster-admin 권한이 있는 클러스터에 로그인했습니다.

프로세스

1. 다음 예에 따라 oc expose 명령을 실행하여 클러스터 외부에 DNS 서버를 노출하는 로드 밸런서 서비스를 생성합니다.

```
$ oc expose -n openshift-cnv deployment/secondary-dns --name=dns-lb \
  --type=LoadBalancer --port=53 --target-port=5353 --protocol='UDP'
```

2. 다음 명령을 실행하여 외부 IP 주소를 검색합니다.

```
$ oc get service -n openshift-cnv
```

출력 예

```
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)         AGE
dns-lb    LoadBalancer   172.30.27.5    10.46.41.94     53:31829/TCP    5s
```

3. 다음 명령을 실행하여 기본 편집기에서 HyperConverged CR을 편집합니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

4. 다음 예에 따라 **DNS** 서버 및 모니터링 구성 요소를 활성화합니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  featureGates:
    deployKubeSecondaryDNS: true
  kubeSecondaryDNSNameServerIP: "10.46.41.94" (1)
# ...
```

(1) 로드 밸런서 서비스에서 노출하는 외부 **IP** 주소를 지정합니다.

5. 파일을 저장하고 편집기를 종료합니다.

6. 다음 명령을 실행하여 클러스터 **FQDN**을 검색합니다.

```
$ oc get dnses.config.openshift.io cluster -o jsonpath='{.spec.baseDomain}'
```

출력 예

```
openshift.example.com
```

7. 다음 방법 중 하나를 사용하여 **DNS** 서버를 가리킵니다.

- 로컬 머신의 **resolv.conf** 파일에 **kubeSecondaryDNSNameServerIP** 값을 추가합니다.

참고

resolv.conf 파일을 편집하면 기존 DNS 설정을 덮어씁니다.

- kubeSecondaryDNSNameServerIP 값과 클러스터 FQDN을 엔터프라이즈 DNS 서버 레코드에 추가합니다. 예를 들면 다음과 같습니다.

```
vm.<FQDN>. IN NS ns.vm.<FQDN>.
```

```
ns.vm.<FQDN>. IN A 10.46.41.94
```

8.13.2. 클러스터 FQDN을 사용하여 보조 네트워크의 VM에 연결

클러스터의 FQDN(정규화된 도메인 이름)을 사용하여 보조 네트워크 인터페이스에 연결된 실행 중인 VM(가상 머신)에 액세스할 수 있습니다.

사전 요구 사항

- VM에 QEMU 게스트 에이전트가 설치되어 있어야 합니다.
- VM의 IP 주소는 공용입니다.
- 보조 네트워크에 대한 DNS 서버를 구성했습니다.
- 클러스터의 FQDN(정규화된 도메인 이름)을 검색했습니다.

프로세스

1. 다음 명령을 실행하여 VM 구성에서 네트워크 인터페이스 이름을 검색합니다.

```
$ oc get vm -n <namespace> <vm_name> -o yaml
```

출력 예

```
apiVersion: kubevirt.io/v1
```

```
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  running: true
  template:
    spec:
      domain:
        devices:
          interfaces:
            - bridge: {}
              name: example-nic
# ...
      networks:
      - multus:
          networkName: bridge-conf
        name: example-nic (1)
```

(1) 네트워크 인터페이스의 이름을 확인합니다.

2. ssh 명령을 사용하여 VM에 연결합니다.

```
$ ssh <user_name>@<interface_name>.<vm_name>.<namespace>.vm.<cluster_fqdn>
```

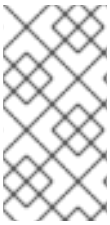
8.13.3. 추가 리소스

- 로드 밸런서를 사용하여 수신 클러스터 트래픽 구성
- MetalLB로 로드 밸런싱
- 가상 머신용 IP 주소 구성

8.14. 네트워크 인터페이스의 MAC 주소 풀 관리

KubeMacPool 구성 요소는 공유 MAC 주소 풀의 VM(가상 머신) 네트워크 인터페이스에 MAC 주소를 할당합니다. 이렇게 하면 각 네트워크 인터페이스에 고유한 MAC 주소가 할당됩니다.

해당 VM에서 생성된 가상 머신 인스턴스는 재부팅 시 할당된 MAC 주소를 유지합니다.



참고

KubeMacPool은 가상 머신과 독립적으로 생성된 가상 머신 인스턴스는 처리하지 않습니다.

8.14.1. 명령줄을 사용하여 KubeMacPool 관리

명령줄을 사용하여 KubeMacPool을 비활성화하고 다시 활성화할 수 있습니다.

KubeMacPool은 기본적으로 활성화되어 있습니다.

프로세스

- 두 네임스페이스에서 KubeMacPool을 비활성화하려면 다음 명령을 실행합니다.

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io=ignore
```

- 두 네임스페이스에서 KubeMacPool을 다시 활성화하려면 다음 명령을 실행합니다.

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io-
```

9장. 스토리지

9.1. 스토리지 구성 개요

기본 스토리지 클래스, 스토리지 프로필, CDI(Containerized Data Importer), 데이터 볼륨 및 자동 부팅 소스 업데이트를 구성할 수 있습니다.

9.1.1. 스토리지

다음 스토리지 구성 작업은 필수입니다.

기본 스토리지 클래스 구성

클러스터의 기본 스토리지 클래스를 구성해야 합니다. 그렇지 않으면 클러스터에서 자동 부팅 소스 업데이트를 수신할 수 없습니다.

스토리지 프로필 구성

CDI에서 스토리지 공급자를 인식하지 못하는 경우 스토리지 프로필을 구성해야 합니다. 스토리지 프로필은 관련 스토리지 클래스를 기반으로 권장 스토리지 설정을 제공합니다.

다음 스토리지 구성 작업은 선택 사항입니다.

파일 시스템 오버헤드를 위한 추가 PVC 공간 예약

기본적으로 파일 시스템 PVC의 5.5%는 오버헤드용으로 예약되어 해당 용량에 따라 VM 디스크에 사용 가능한 공간을 줄일 수 있습니다. 다른 오버헤드 값을 구성할 수 있습니다.

hostpath 프로비전 프로그램을 사용하여 로컬 스토리지 구성

HPP(Hostpath provisioner)를 사용하여 가상 머신의 로컬 스토리지를 구성할 수 있습니다. OpenShift Virtualization Operator를 설치하면 HPP Operator가 자동으로 설치됩니다.

네임스페이스 간에 데이터 볼륨을 복제하도록 사용자 권한 구성

사용자가 네임스페이스 간에 데이터 볼륨을 복제할 수 있도록 RBAC 역할을 구성할 수 있습니다.

9.1.2. 컨테이너화된 데이터 가져오기

다음과 같은 CDI(Containerized Data Importer) 구성 작업을 수행할 수 있습니다.

네임스페이스의 리소스 요청 제한 덮어쓰기

CPU 및 메모리 리소스 제한이 적용되는 네임스페이스에 VM 디스크를 가져오고, 업로드하고, 복제하도록 CDI를 구성할 수 있습니다.

CDI 스크래치 공간 구성

CDI에는 VM 이미지 가져오기 및 업로드와 같은 일부 작업을 완료하기 위해 스크래치 공간(임시 스토리지)이 필요합니다. 이 프로세스 동안 CDI는 대상 DV(데이터 볼륨)를 지원하는 PVC 크기와 같은 스크래치 공간 PVC를 프로비저닝합니다.

9.1.3. 데이터 볼륨

다음 데이터 볼륨 구성 작업을 수행할 수 있습니다.

데이터 볼륨에 대한 사전 할당 활성화

CDI는 데이터 볼륨을 생성할 때 쓰기 성능을 개선하기 위해 디스크 공간을 사전 할당할 수 있습니다. 특정 데이터 볼륨에 대해 사전 할당을 실행할 수 있습니다.

데이터 볼륨 주석 관리

데이터 볼륨 주석을 사용하면 Pod 동작을 관리할 수 있습니다. 하나 이상의 주석을 데이터 볼륨에 추가하면 생성된 가져오기 Pod로 전파할 수 있습니다.

9.1.4. 부팅 소스 업데이트

다음 부팅 소스 업데이트 구성 작업을 수행할 수 있습니다.

자동 부팅 소스 업데이트 관리

부팅 소스를 사용하면 VM(가상 머신)을 더 쉽게 쉽게 생성할 수 있습니다. 자동 부팅 소스 업데이트가 활성화된 경우 CDI에서 새 VM에 대해 복제할 수 있도록 이미지를 가져오기, 폴링 및 업데이트합니다. 기본적으로 CDI는 Red Hat 부팅 소스를 자동으로 업데이트합니다. 사용자 정의 부팅 소스에 대한 자동 업데이트를 활성화할 수 있습니다.

9.2. 스토리지 프로필 구성

스토리지 프로필은 관련 스토리지 클래스를 기반으로 권장 스토리지 설정을 제공합니다. 각 스토리지 클래스에 대해 스토리지 프로필이 할당됩니다.

CDI(Containerized Data Importer)는 스토리지 공급자의 기능을 식별하고 상호 작용하도록 구성된 경

우 스토리지 공급자를 인식합니다.

인식된 스토리지 유형의 경우 CDI는 PVC 생성을 최적화하는 값을 제공합니다. 스토리지 프로필을 사용자 지정하여 스토리지 클래스에 대한 자동 설정을 구성할 수도 있습니다. CDI에서 스토리지 공급자를 인식하지 못하는 경우 스토리지 프로필을 구성해야 합니다.

중요

Red Hat OpenShift Data Foundation과 함께 OpenShift Virtualization을 사용하는 경우 가상 머신 디스크를 생성할 때 RBD 블록 모드 PVC(영구 볼륨 클레임)를 지정합니다. RBD 블록 모드 볼륨은 Ceph FS 또는 RBD 파일 시스템 모드 PVC보다 더 효율적이며 더 나은 성능을 제공합니다.

RBD 블록 모드 PVC를 지정하려면 'ocs-storagecluster-ceph-rbd' 스토리지 클래스와 VolumeMode: Block 을 사용합니다.

9.2.1. 스토리지 프로파일 사용자 정의

프로비저너의 스토리지 클래스에 대해 StorageProfile 오브젝트를 편집하여 기본 매개변수를 지정할 수 있습니다. 이러한 기본 매개변수는 DataVolume 오브젝트에 구성되지 않은 경우에만 PVC(영구 볼륨 클레임)에 적용됩니다.

스토리지 클래스 매개변수는 수정할 수 없습니다. 변경하려면 스토리지 클래스를 삭제하고 다시 생성합니다. 그런 다음 스토리지 프로필에 이전에 만든 사용자 지정을 다시 적용해야 합니다.

스토리지 프로필의 빈 status 섹션은 스토리지 프로비저너가 CDI(Containerized Data Interface)에서 인식되지 않았음을 나타냅니다. CDI에서 인식하지 못하는 스토리지 프로비저너가 있는 경우 스토리지 프로필을 사용자 정의하는 것이 필요합니다. 이 경우 관리자는 스토리지 프로필에 적절한 값을 설정하여 성공적으로 할당되도록 합니다.

주의

데이터 볼륨을 생성하고 YAML 속성을 생략하고 이러한 특성이 스토리지 프로필에 정의되지 않으면 요청된 스토리지가 할당되지 않고 기본 PVC(영구 볼륨 클레임)가 생성되지 않습니다.

사전 요구 사항

- 계획된 구성이 스토리지 클래스 및 해당 공급자에 의해 지원되는지 확인하십시오. 스토리지 프로필에 호환되지 않는 구성을 지정하면 볼륨 프로비저닝이 실패합니다.

프로세스

1. 스토리지 프로파일을 편집합니다. 이 예에서는 **CDI**에서 프로비저너를 인식하지 못합니다.

```
$ oc edit storageprofile <storage_class>
```

스토리지 프로필 예

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <unknown_provisioner_class>
# ...
spec: {}
status:
  provisioner: <unknown_provisioner>
  storageClass: <unknown_provisioner_class>
```

2. 스토리지 프로파일에 필요한 속성 값을 제공합니다.

스토리지 프로필 예

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <unknown_provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce (1)
    volumeMode:
      Filesystem (2)
```

```
status:
  provisioner: <unknown_provisioner>
  storageClass: <unknown_provisioner_class>
```

1

선택한 **accessModes**입니다.

2

선택한 **volumeMode**입니다.

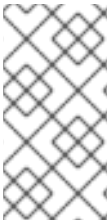
변경 사항을 저장하면 선택한 값이 스토리지 프로필 **status** 요소에 표시됩니다.

9.2.1.1. 스토리지 프로필을 사용하여 기본 복제 전략 설정

스토리지 프로필을 사용하여 스토리지 클래스의 기본 복제 방법을 설정하여 *복제 전략*을 생성할 수 있습니다. 예를 들어, 스토리지 벤더가 특정 복제 방법만 지원하는 경우 복제 전략을 설정하면 유용할 수 있습니다. 또한 리소스 사용을 제한하거나 성능을 극대화하는 방법을 선택할 수 있습니다.

스토리지 프로필의 **cloneStrategy** 특성을 다음 값 중 하나로 설정하여 전략을 복제할 수 있습니다.

- 스냅샷은 스냅샷 이 구성될 때 기본적으로 사용됩니다. **CDI**는 스토리지 공급자를 인식하고 공급자는 **CSI(Container Storage Interface)** 스냅샷을 지원하는 경우 스냅샷 방법을 사용합니다. 이 복제 전략에서는 임시 볼륨 스냅샷을 사용하여 볼륨을 복제합니다.
- 복사 는 소스 **Pod**와 대상 **Pod**를 사용하여 소스 볼륨의 데이터를 대상 볼륨으로 복사합니다. 호스트 지원 복제는 가장 효율적인 복제 방법입니다.
- **CSI-clone** 은 **CSI** 복제 **API**를 사용하여 임시 볼륨 스냅샷을 사용하지 않고 기존 볼륨을 효율적으로 복제합니다. 스토리지 프로파일이 정의되지 않은 경우 기본적으로 사용되는 **snapshot** 또는 **copy**와 달리 **CSI** 볼륨 복제는 프로비저너의 스토리지 클래스에 대해 **StorageProfile** 오브젝트에 지정된 경우에만 사용됩니다.



참고

YAML spec 섹션에서 기본 claimPropertySets 를 수정하지 않고 CLI를 사용하여 복제 전략을 설정할 수도 있습니다.

스토리지 프로필 예

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce 1
    volumeMode:
      Filesystem 2
  cloneStrategy: csi-clone 3
status:
  provisioner: <provisioner>
  storageClass: <provisioner_class>
```

1

액세스 모드를 지정합니다.

2

볼륨 모드를 지정합니다.

3

기본 복제 전략을 지정합니다.

표 9.1. 스토리지 공급자 및 기본 동작

스토리지 공급자	기본 동작
rook-ceph.rbd.csi.ceph.com	스냅샷

스토리지 공급자	기본 동작
openshift-storage.rbd.csi.ceph.com	스냅샷
csi-vxflexos.dellemc.com	CSI Clone
csi-isilon.dellemc.com	CSI Clone
csi-powermax.dellemc.com	CSI Clone
csi-powerstore.dellemc.com	CSI Clone
hspc.csi.hitachi.com	CSI Clone
csi.hpe.com	CSI Clone
spectrumscale.csi.ibm.com	CSI Clone
rook-ceph.rbd.csi.ceph.com	CSI Clone
openshift-storage.rbd.csi.ceph.com	CSI Clone
cephfs.csi.ceph.com	CSI Clone
openshift-storage.cephfs.csi.ceph.com	CSI Clone

9.3. 자동 부팅 소스 업데이트 관리

다음 부팅 소스에 대한 자동 업데이트를 관리할 수 있습니다.

- 모든 Red Hat 부팅 소스
- 모든 사용자 정의 부팅 소스
- 개별 Red Hat 또는 사용자 정의 부팅 소스

부팅 소스를 사용하면 VM(가상 머신)을 더 쉽게 쉽게 생성할 수 있습니다. 자동 부팅 소스 업데이트가 활성화되면 CDI(Containerized Data Importer)가 이미지를 가져오고, 폴링하고, 업데이트하여 새 VM에 대해 복제할 수 있도록 합니다. 기본적으로 CDI는 Red Hat 부팅 소스를 자동으로 업데이트합니다.

9.3.1. Red Hat 부팅 소스 업데이트 관리

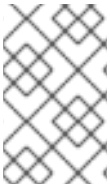
enableCommonBootImageImport 기능 게이트를 비활성화하여 모든 시스템 정의 부팅 소스에 대한 자동 업데이트를 비활성화할 수 있습니다. 이 기능 게이트를 비활성화하면 모든 **DataImportCron** 오브젝트가 삭제됩니다. 관리자가 수동으로 삭제할 수 있지만 운영 체제 이미지를 저장하는 이전에 가져온 부팅 소스 오브젝트는 제거되지 않습니다.

enableCommonBootImageImport 기능 게이트를 비활성화하면 **DataSource** 오브젝트가 재설정되어 더 이상 원래 부팅 소스를 가리키지 않습니다. 관리자는 **DataSource** 오브젝트에 대한 새 **PVC(**영구 볼륨 클레임**)** 또는 볼륨 스냅샷을 생성한 다음 운영 체제 이미지로 채워 부팅 소스를 수동으로 제공할 수 있습니다.

9.3.1.1. 모든 시스템 정의 부팅 소스에 대한 자동 업데이트 관리

자동 부팅 소스 가져오기 및 업데이트를 비활성화하면 리소스 사용량을 줄일 수 있습니다. 연결이 끊긴 환경에서 자동 부팅 소스 업데이트를 비활성화하면 **CDIDataImportCronOutdated** 경고가 로그를 채울 수 없습니다.

모든 시스템 정의 부팅 소스에 대한 자동 업데이트를 비활성화하려면 값을 **false** 로 설정하여 **enableCommonBootImageImport** 기능 게이트를 끕니다. 이 값을 **true** 로 설정하면 기능 게이트가 다시 활성화되고 자동 업데이트가 다시 설정됩니다.



참고

사용자 지정 부팅 소스는 이 설정의 영향을 받지 않습니다.

프로세스

- **HyperConverged CR(**사용자 정의 리소스**)**을 편집하여 자동 부팅 소스 업데이트 기능 게이트를 전환합니다.
  - 자동 부팅 소스 업데이트를 비활성화하려면 **HyperConverged CR**의 **spec.featureGates.enableCommonBootImageImport** 필드를 **false** 로 설정합니다. 예를 들면 다음과 같습니다.

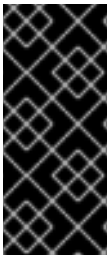
```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/enableCommonBootImageImport", \
  "value": false}]'
```

자동 부팅 소스 업데이트를 다시 활성화하려면 **HyperConverged CR**의 **spec.featureGates.enableCommonBootImageImport** 필드를 **true** 로 설정합니다. 예를 들면 다음과 같습니다.

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/enableCommonBootImageImport", \
  "value": true}]'
```

9.3.2. 사용자 정의 부팅 소스 업데이트 관리

OpenShift Virtualization에서 제공하지 않는 *사용자 정의* 부팅 소스는 기능 게이트에서 제어하지 않습니다. **HyperConverged CR**(사용자 정의 리소스)을 편집하여 개별적으로 관리해야 합니다.



중요

스토리지 클래스를 구성해야 합니다. 그렇지 않으면 클러스터에서 사용자 정의 부팅 소스에 대한 자동 업데이트를 수신할 수 없습니다. 자세한 내용은 스토리지 클래스 정의를 참조하십시오.

9.3.2.1. 사용자 정의 부팅 소스 업데이트를 위한 스토리지 클래스 구성

HyperConverged CR(사용자 정의 리소스)을 편집하여 기본 스토리지 클래스를 덮어쓸 수 있습니다.

중요

부팅 소스는 기본 스토리지 클래스를 사용하여 스토리지에서 생성됩니다. 클러스터에 기본 스토리지 클래스가 없는 경우 사용자 정의 부팅 소스에 대한 자동 업데이트를 구성하기 전에 하나를 정의해야 합니다.

프로세스

1. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged CR**을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **storageClassName** 필드에 값을 입력하여 새 스토리지 클래스를 정의합니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
```

```
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: rhel8-image-cron
    spec:
      template:
        spec:
          storageClassName: <new_storage_class> 1
      schedule: "0 */12 * * *" 2
      managedDataSource: <data_source> 3
# ...
```

1 스토리지 클래스를 정의합니다.

2 필수: cron 형식으로 지정된 작업의 스케줄입니다.

3 필수: 사용할 데이터 소스입니다.

> For the custom image to be detected as an available boot source, the value of the `spec.dataVolumeTemplates.spec.sourceRef.name` parameter in the VM template must match this value.

3. 현재 기본 스토리지 클래스에서 **storageclass.kubernetes.io/is-default-class** 주석을 제거합니다.

   a. 다음 명령을 실행하여 현재 기본 스토리지 클래스의 이름을 검색합니다.

   ```
   $ oc get storageclass
   ```

   출력 예

   ```
   NAME                 PROVISIONER                RECLAIMPOLICY
   VOLUMEBINDINGMODE    ALLOWVOLUMEEXPANSION  AGE
   csi-manila-ceph      manila.csi.openstack.org   Delete       Immediate
   ```

```
false                 11d
hostpath-csi-basic (default)  kubevirt.io.hostpath-provisioner Delete
WaitForFirstConsumer false                 11d 1
```

1

이 예에서 현재 기본 스토리지 클래스의 이름은 **hostpath-csi-basic** 입니다.

b.

다음 명령을 실행하여 현재 기본 스토리지 클래스에서 주석을 제거합니다.

```
$ oc patch storageclass <current_default_storage_class> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}' 1
```

1

& lt;current_default_storage_class >를 기본 스토리지 클래스의 **storageClassName** 값으로 바꿉니다.

4.

다음 명령을 실행하여 새 스토리지 클래스를 기본값으로 설정합니다.

```
$ oc patch storageclass <new_storage_class> -p '{"metadata":{"annotations":
{"storageclass.kubernetes.io/is-default-class":"true"}}}' 1
```

1

& lt;new_storage_class >를 **HyperConverged CR**에 추가한 **storageClassName** 값으로 바꿉니다.

9.3.2.2. 사용자 정의 부팅 소스에 대한 자동 업데이트 활성화

OpenShift Virtualization은 기본적으로 시스템 정의 부팅 소스를 자동으로 업데이트하지만 사용자 정의 부팅 소스를 자동으로 업데이트하지는 않습니다. **HyperConverged CR**(사용자 정의 리소스)을 편집하여 자동 업데이트를 수동으로 활성화해야 합니다.

사전 요구 사항

- 클러스터에는 기본 스토리지 클래스가 있습니다.

프로세스

1. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged CR**을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **HyperConverged CR**을 편집하여 **dataImportCronTemplates** 섹션에 적절한 템플릿 및 부팅 소스를 추가합니다. 예를 들면 다음과 같습니다.

사용자 정의 리소스의 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: centos7-image-cron
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true" (1)
    spec:
      schedule: "0 */12 * * *" (2)
      template:
        spec:
          source:
            registry: (3)
              url: docker://quay.io/containerdisks/centos:7-2009
          storage:
            resources:
              requests:
                storage: 10Gi
      managedDataSource: centos7 (4)
      retentionPolicy: "None" (5)
```

(1) 이 주석은 **volumeBindingMode** 가 **WaitForFirstConsumer** 로 설정된 스토리지 클래스에 필요합니다.

(2)

cron 형식으로 지정된 작업의 스케줄입니다.

3

를 사용하여 레지스트리 소스에서 데이터 볼륨을 생성합니다. 노드 docker 캐시를 기반으로 하는 노드 pullMethod 가 아닌 기본 Pod pullMethod 를 사용합니다. 노드 Docker 캐시는 Container.Image 를 통해 레지스트리 이미지를 사용할 수 있지만 CDI 가져오기는 액세스할 수 없는 경우 유용합니다.

4

사용자 지정 이미지가 사용 가능한 부팅 소스로 감지되려면 이미지의 managedDataSource 이름이 VM 템플릿 YAML 파일의 spec.dataVolumeTemplates.spec.sourceRef.name 에 있는 템플릿의 DataSource 이름과 일치해야 합니다.

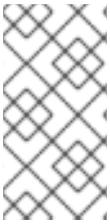
5

cron 작업이 삭제될 때 모두 데이터 볼륨 및 데이터 소스를 유지합니다. cron 작업이 삭제될 때 데이터 볼륨 및 데이터 소스를 삭제하려면 None 을 사용합니다.

3. 파일을 저장합니다.

9.3.2.3. 볼륨 스냅샷 부팅 소스 활성화

운영 체제 기본 이미지를 저장하는 스토리지 클래스와 연결된 StorageProfile 의 매개변수를 설정하여 볼륨 스냅샷 부팅 소스를 활성화합니다. DataImportCron 은 원래 PVC 소스만 유지하도록 설계되었지만 VolumeSnapshot 소스는 특정 스토리지 유형의 PVC 소스보다 더 잘 확장됩니다.



참고

단일 스냅샷에서 복제할 때 더 잘 확장하려면 검증된 스토리지 프로필에서 볼륨 스냅샷을 사용합니다.

사전 요구 사항

- 운영 체제 이미지를 사용하여 볼륨 스냅샷에 액세스할 수 있어야 합니다.
- 스토리지에서 스냅샷을 지원해야 합니다.

프로세스

1. 다음 명령을 실행하여 부팅 소스를 프로비저닝하는 데 사용되는 스토리지 클래스에 해당하는 스토리지 프로필 오브젝트를 엽니다.

```
$ oc edit storageprofile <storage_class>
```

2. StorageProfile 의 dataImportCronSourceFormat 사양을 검토하여 VM이 기본적으로 PVC 또는 볼륨 스냅샷을 사용 중인지 확인합니다.

3. dataImportCronSourceFormat 사양을 스냅샷 으로 업데이트하여 필요한 경우 스토리지 프로필을 편집합니다.

스토리지 프로필 예

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
# ...
spec:
  dataImportCronSourceFormat: snapshot
```

검증

1. 부팅 소스를 프로비저닝하는 데 사용되는 스토리지 클래스에 해당하는 스토리지 프로필 오브젝트를 엽니다.

```
$ oc get storageprofile <storage_class>  -oyaml
```

2. StorageProfile 의 dataImportCronSourceFormat 사양이 'snapshot'으로 설정되어 있고 DataImportCron 이 가리키는 모든 DataSource 오브젝트가 볼륨 스냅샷을 참조하는지 확인합니다.

이제 이러한 부팅 소스를 사용하여 가상 머신을 생성할 수 있습니다.

9.3.3. 단일 부팅 소스에 대한 자동 업데이트 비활성화

HyperConverged CR(사용자 정의 리소스)을 편집하여 사용자 정의 또는 시스템 정의 여부에 관계없이 개별 부팅 소스에 대한 자동 업데이트를 비활성화할 수 있습니다.

프로세스

1. **다음 명령을 실행하여 기본 편집기에서 HyperConverged CR을 엽니다.**

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **spec.dataImportCronTemplates 필드를 편집하여 개별 부팅 소스에 대한 자동 업데이트를 비활성화합니다.**

사용자 정의 부팅 소스

- **spec.dataImportCronTemplates 필드에서 부팅 소스를 제거합니다. 자동 업데이트는 기본적으로 사용자 정의 부팅 소스에 대해 비활성화됩니다.**

시스템 정의 부팅 소스

a. **spec.dataImportCronTemplates 에 부팅 소스를 추가합니다.**

참고

자동 업데이트는 시스템 정의 부팅 소스에 대해 기본적으로 활성화되지만 추가하지 않는 한 이러한 부팅 소스는 CR에 나열되지 않습니다.

b. **dataimportcrontemplate.kubevirt.io/enable 주석의 값을 'false' 로 설정합니다.**

예를 들면 다음과 같습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      annotations:
```

```
      dataimportcrontemplate.kubevirt.io/enable: 'false'
    name: rhel8-image-cron
# ...
```

3. 파일을 저장합니다.

9.3.4. 부팅 소스 상태 확인

HyperConverged CR(사용자 정의 리소스)을 확인하여 부팅 소스가 시스템 정의 또는 사용자 정의인지 확인할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 HyperConverged CR의 콘텐츠를 확인합니다.

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o yaml
```

출력 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
# ...
status:
# ...
  dataImportCronTemplates:
  - metadata:
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
      name: centos-7-image-cron
    spec:
      garbageCollect: Outdated
      managedDataSource: centos7
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              url: docker://quay.io/containerdisks/centos:7-2009
          storage:
            resources:
              requests:
```

```
            storage: 30Gi
        status: {}
      status:
        commonTemplate: true 1
    # ...
     - metadata:
        annotations:
          cdi.kubevirt.io/storage.bind.immediate.requested: "true"
        name: user-defined-dic
      spec:
        garbageCollect: Outdated
        managedDataSource: user-defined-centos-stream8
        schedule: 55 8/12 * * *
        template:
          metadata: {}
          spec:
            source:
              registry:
                pullMethod: node
                url: docker://quay.io/containerdisks/centos-stream:8
            storage:
              resources:
                requests:
                  storage: 30Gi
        status: {}
      status: {} 2
    # ...
```

1 시스템 정의 부팅 소스를 나타냅니다.

2 사용자 지정 부팅 소스를 나타냅니다.

2. **status.dataImportCronTemplates.status** 필드를 검토하여 부팅 소스의 상태를 확인합니다.

- 필드에 **commonTemplate: true** 가 포함된 경우 시스템 정의 부팅 소스입니다.
- **status.dataImportCronTemplates.status** 필드에 **{}** 값이 있는 경우 사용자 지정 부팅 소스입니다.

9.4. 파일 시스템 오버헤드의 PVC 공간 예약

Filesystem 볼륨 모드를 사용하는 PVC(영구 볼륨 클레임)에 가상 머신 디스크를 추가할 때 VM 디스크용 PVC 및 파일 시스템 오버헤드(예: 메타데이터)에 충분한 공간이 있는지 확인해야 합니다.

기본적으로 OpenShift Virtualization은 오버헤드를 위해 PVC 공간의 5.5%를 예약하므로 해당 용량에 따라 가상 머신 디스크에 사용 가능한 공간을 줄일 수 있습니다.

HCO 오브젝트를 편집하여 다른 오버헤드 값을 구성할 수 있습니다. 전체적으로 값을 변경하고 특정 스토리지 클래스의 값을 지정할 수 있습니다.

9.4.1. 기본 파일 시스템 오버헤드 값 덮어쓰기

HCO 오브젝트의 spec.filesystemOverhead 속성을 편집하여 OpenShift Virtualization에서 파일 시스템 오버헤드를 위해 예약하는 PVC(영구 볼륨 클레임) 공간을 변경합니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.

프로세스

1. 다음 명령을 실행하여 편집할 HCO 오브젝트를 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. spec.filesystemOverhead 필드를 편집하여 선택한 값으로 채웁니다.

```
# ...
spec:
  filesystemOverhead:
    global: "<new_global_value>" 1
    storageClass:
      <storage_class_name>: "<new_value_for_this_storage_class>" 2
```

1. 설정된 값이 없는 스토리지 클래스에 사용되는 기본 파일 시스템 오버헤드 백분율입니다. 예를 들어, global: "0.07"은 파일 시스템 오버헤드용으로 PVC의 7%를 예약합니다.

2

지정된 스토리지 클래스의 파일 시스템 오버헤드 백분율입니다. 예를 들어, **mystorageclass: "0.04"**는 **mystorageclass** 스토리지 클래스의 **PVC**의 기본 오버헤드 값을 **4%**로 변경합니다.

3. 편집기를 저장하고 종료하여 **HCO** 오브젝트를 업데이트합니다.

검증

- 다음 명령 중 하나를 실행하여 **CDIConfig** 상태를 보고 변경 사항을 확인합니다.

일반적으로 **CDIConfig** 변경 사항을 확인하려면 다음을 수행합니다.

```
$ oc get cdiconfig -o yaml
```

CDIConfig 에 대한 특정 변경 사항을 보려면 다음을 수행합니다.

```
$ oc get cdiconfig -o jsonpath='{.items..status.filesystemOverhead}'
```

9.5. HOSTPATH 프로비전 프로그램을 사용하여 로컬 스토리지 구성

HPP(Hostpath provisioner)를 사용하여 가상 머신의 로컬 스토리지를 구성할 수 있습니다.

OpenShift Virtualization Operator를 설치하면 **Hostpath Provisioner Operator**가 자동으로 설치됩니다. **HPP**는 **Hostpath Provisioner Operator**가 생성한 **OpenShift Virtualization**을 위해 설계된 로컬 스토리지 프로비전 프로그램입니다. **HPP**를 사용하려면 기본 스토리지 풀이 있는 **HPP CR(**사용자 정의 리소스**)**을 생성합니다.

9.5.1. 기본 스토리지 풀을 사용하여 hostpath 프로비전 프로그램 생성

storagePools 스탠자를 사용하여 **HPP CR(**사용자 정의 리소스**)**을 생성하여 기본 스토리지 풀로 **HPP(Hostpath** 프로비전 프로그램**)**를 구성합니다. 스토리지 풀은 **CSI** 드라이버에서 사용하는 이름과 경로를 지정합니다.

중요

운영 체제와 동일한 파티션에 스토리지 풀을 생성하지 마십시오. 그렇지 않으면 운영 체제 파티션이 용량으로 채워지기 때문에 성능에 영향을 미치거나 노드가 불안정하거나 사용할 수 없게 됩니다.

사전 요구 사항

- spec.storagePools.path 에 지정된 디렉터리에는 읽기/쓰기 액세스 권한이 있어야 합니다.

프로세스

1. 다음 예와 같이 storagePools 스탠자를 사용하여 hpp_cr.yaml 파일을 생성합니다.

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  storagePools: 1
  - name: any_name
    path: "/var/myvolumes" 2
workload:
  nodeSelector:
    kubernetes.io/os: linux
```

1 storagePools 스탠자는 여러 항목을 추가할 수 있는 배열입니다.

2 이 노드 경로 아래에 스토리지 풀 디렉터리를 지정합니다.

2. 파일을 저장하고 종료합니다.

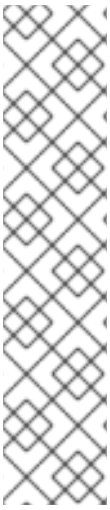
3. 다음 명령을 실행하여 HPP를 생성합니다.

```
$ oc create -f hpp_cr.yaml
```

9.5.1.1. 스토리지 클래스 생성 정보

스토리지 클래스를 생성할 때 해당 스토리지 클래스에 속하는 PV(영구 볼륨)의 동적 프로비저닝에 영향을 주는 매개변수를 설정합니다. StorageClass 오브젝트를 생성한 후에는 이 오브젝트의 매개변수를 업데이트할 수 없습니다.

hostpath 프로비전 프로그램(HPP)을 사용하려면 storagePools 스탠자를 사용하여 CSI 드라이버에 대한 관련 스토리지 클래스를 생성해야 합니다.



참고

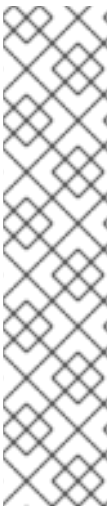
가상 머신은 로컬 PV를 기반으로 하는 데이터 볼륨을 사용합니다. 로컬 PV는 특정 노드에 바인딩됩니다. 디스크 이미지는 가상 머신에서 사용할 수 있는 반면 가상 머신은 이전에 로컬 스토리지 PV가 고정된 노드에 예약할 수 없습니다.

이 문제를 해결하려면 Kubernetes Pod 스케줄러를 사용하여 PVC(영구 볼륨 클레임)를 올바른 노드의 PV에 바인딩합니다. volumeBindingMode 매개변수가 WaitForFirstConsumer 로 설정된 StorageClass 값을 사용하면 PVC를 사용하여 Pod가 생성될 때까지 PV의 바인딩 및 프로비저닝이 지연됩니다.

9.5.1.2. storagePools 스탠자를 사용하여 CSI 드라이버의 스토리지 클래스 생성

HPP(Hostpath provisioner)를 사용하려면 CSI(Container Storage Interface) 드라이버에 연결된 스토리지 클래스를 생성해야 합니다.

스토리지 클래스를 생성할 때 해당 스토리지 클래스에 속하는 PV(영구 볼륨)의 동적 프로비저닝에 영향을 주는 매개변수를 설정합니다. StorageClass 오브젝트를 생성한 후에는 이 오브젝트의 매개변수를 업데이트할 수 없습니다.



참고

가상 머신은 로컬 PV를 기반으로 하는 데이터 볼륨을 사용합니다. 로컬 PV는 특정 노드에 바인딩됩니다. 디스크 이미지는 가상 머신에서 사용할 수 있지만 이전에 로컬 스토리지 PV가 고정된 노드에 가상 머신을 예약할 수 없습니다.

이 문제를 해결하려면 Kubernetes Pod 스케줄러를 사용하여 PVC(영구 볼륨 클레임)를 올바른 노드의 PV에 바인딩합니다. volumeBindingMode 매개변수가 WaitForFirstConsumer 로 설정된 StorageClass 값을 사용하면 PVC를 사용하여 Pod가 생성될 때까지 PV의 바인딩 및 프로비저닝이 지연됩니다.

프로세스

1. storageclass_csi.yaml 파일을 생성하여 스토리지 클래스를 정의합니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hostpath-csi
provisioner: kubevirt.io.hostpath-provisioner
reclaimPolicy: Delete 1
volumeBindingMode: WaitForFirstConsumer 2
parameters:
  storagePool: my-storage-pool 3
```

1. reclaimPolicy에 사용할 수 있는 값은 Delete 및 Retain 두 가지입니다. 값을 지정하지 않으면 기본값은 Delete 입니다.

2. volumeBindingMode 매개변수는 동적 프로비저닝 및 볼륨 바인딩이 발생하는 시기를 결정합니다. PVC(영구 볼륨 클레임)를 사용하는 Pod가 생성될 때까지 PV(영구 볼륨)의 바인딩 및 프로비저닝을 지연하려면 WaitForFirstConsumer 를 지정합니다. 이렇게 하면 PV에서 Pod의 스케줄링 요구 사항을 충족할 수 있습니다.

3. HPP CR에 정의된 스토리지 풀의 이름을 지정합니다.

2. 파일을 저장하고 종료합니다.

3. 다음 명령을 실행하여 StorageClass 오브젝트를 만듭니다.

```
$ oc create -f storageclass_csi.yaml
```

9.5.2. PVC 템플릿으로 생성된 스토리지 풀 정보

대규모 PV(영구 볼륨)가 단일 있는 경우 HPP(Hostpath provisioner) 사용자 정의 리소스(CR)에 PVC 템플릿을 정의하여 스토리지 풀을 생성할 수 있습니다.

PVC 템플릿으로 생성된 스토리지 풀에는 여러 HPP 볼륨이 포함될 수 있습니다. PV를 더 작은 볼륨으

로 분할하면 데이터 할당의 유연성이 향상됩니다.

PVC 템플릿은 PersistentVolumeClaim 오브젝트의 spec 스탠자를 기반으로 합니다.

PersistentVolumeClaim 오브젝트의 예

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iso-pvc
spec:
  volumeMode: Block (1)
  storageClassName: my-storage-class
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

(1) 이 값은 블록 볼륨 모드 PV에만 필요합니다.

HPP CR의 pvcTemplate 사양을 사용하여 스토리지 풀을 정의합니다. Operator는 HPP CSI 드라이버가 포함된 각 노드의 pvcTemplate 사양에서 PVC를 생성합니다. PVC 템플릿에서 생성된 PVC는 단일 대규모 PV를 사용하므로 HPP에서 더 작은 동적 볼륨을 생성할 수 있습니다.

PVC 템플릿에서 생성된 스토리지 풀과 기본 스토리지 풀을 결합할 수 있습니다.

9.5.2.1. PVC 템플릿을 사용하여 스토리지 풀 생성

HPP CR(사용자 정의 리소스)에 PVC 템플릿을 지정하여 여러 HPP(Hostpath provisioner) 볼륨에 대한 스토리지 풀을 생성할 수 있습니다.

중요

운영 체제와 동일한 파티션에 스토리지 풀을 생성하지 마십시오. 그렇지 않으면 운영 체제 파티션이 용량으로 채워지기 때문에 성능에 영향을 미치거나 노드가 불안정하거나 사용할 수 없게 됩니다.

사전 요구 사항

- spec.storagePools.path 에 지정된 디렉터리에는 읽기/쓰기 액세스 권한이 있어야 합니다.

프로세스

1. 다음 예에 따라 storagePools 스탠자에 PVC(영구 볼륨) 템플릿을 지정하는 HPP CR의 hpp_pvc_template_pool.yaml 파일을 생성합니다.

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  storagePools: 1
  - name: my-storage-pool
    path: "/var/myvolumes" 2
    pvcTemplate:
      volumeMode: Block 3
      storageClassName: my-storage-class 4
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 5Gi 5
  workload:
    nodeSelector:
      kubernetes.io/os: linux
```

1 storagePools 스탠자는 기본 및 PVC 템플릿 스토리지 풀을 모두 포함할 수 있는 배열입니다.

2 이 노드 경로 아래에 스토리지 풀 디렉터리를 지정합니다.

3

4

storageClassName 매개변수가 생략되면 기본 스토리지 클래스가 **PVC**를 생성하는 데 사용됩니다. **storageClassName** 을 생략하는 경우 **HPP** 스토리지 클래스가 기본 스토리지 클래스가 아닌지 확인합니다.

5

정적 또는 동적으로 프로비저닝된 스토리지를 지정할 수 있습니다. 두 경우 모두 요청된 스토리지 크기가 사실상 분할하려는 볼륨에 적합한지 확인하거나 **PVC**를 큰 **PV**에 바인딩할 수 없습니다. 사용 중인 스토리지 클래스가 동적으로 프로비저닝된 스토리지를 사용하는 경우 일반적인 요청 크기와 일치하는 할당 크기를 선택합니다.

2. 파일을 저장하고 종료합니다.

3. 다음 명령을 실행하여 스토리지 풀로 **HPP**를 생성합니다.

```
$ oc create -f hpp_pvc_template_pool.yaml
```

9.6. 네임스페이스 간에 데이터 볼륨을 복제할 수 있는 사용자 권한 활성화

네임스페이스의 격리 특성으로 인해 기본적으로 사용자는 다른 네임스페이스에 리소스를 복제할 수 없습니다.

사용자가 가상 머신을 다른 네임스페이스에 복제할 수 있도록 하려면 **cluster-admin** 역할의 사용자가 새 클러스터 역할을 만들어야 합니다. 이 클러스터 역할을 사용자에게 바인딩하면 사용자가 가상 머신을 대상 네임스페이스에 복제할 수 있습니다.

9.6.1. 데이터 볼륨 복제를 위한 RBAC 리소스 생성

datavolumes 리소스에 대한 모든 작업 권한을 활성화하는 새 클러스터 역할을 만듭니다.

사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.

프로세스

1. ClusterRole 매니페스트를 만듭니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: <datavolume-cloner> 1
rules:
- apiGroups: ["cdi.kubevirt.io"]
  resources: ["datavolumes/source"]
  verbs: ["*"]
```

1 클러스터 역할의 고유 이름입니다.

2. 클러스터에 클러스터 역할을 만듭니다.

```
$ oc create -f <datavolume-cloner.yaml> 1
```

1 이전 단계에서 만든 ClusterRole 매니페스트 파일 이름입니다.

3. 소스 및 대상 네임스페이스 모두에 적용되고 이전 단계에서 만든 클러스터 역할을 참조하는 RoleBinding 매니페스트를 만듭니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: <allow-clone-to-user> 1
  namespace: <Source namespace> 2
subjects:
- kind: ServiceAccount
  name: default
  namespace: <Destination namespace> 3
roleRef:
  kind: ClusterRole
  name: datavolume-cloner 4
  apiGroup: rbac.authorization.k8s.io
```

1 역할 바인딩의 고유 이름입니다.

2

소스 데이터 볼륨의 네임스페이스입니다.

3

데이터 볼륨이 복제되는 네임스페이스입니다.

4

이전 단계에서 만든 클러스터 역할의 이름입니다.

4. 클러스터에 역할 바인딩을 만듭니다.

```
$ oc create -f <datavolume-cloner.yaml> 1
```

1

이전 단계에서 만든 **RoleBinding** 매니페스트 파일 이름입니다.

9.7. CPU 및 메모리 할당량을 덮어쓰도록 CDI 구성

CPU 및 메모리 리소스 제한 사항이 적용되는 네임스페이스에 가상 머신 디스크를 가져오고, 업로드하고, 복제하도록 **CDI(Containerized Data Importer)**를 구성할 수 있습니다.

9.7.1. 네임스페이스의 CPU 및 메모리 할당량 정보

ResourceQuota 오브젝트로 정의하는 *리소스 할당량*은 네임스페이스에 제한을 적용하여 해당 네임스페이스 내의 리소스에서 사용할 수 있는 총 컴퓨팅 리소스 양을 제한합니다.

HyperConverged 사용자 지정 리소스 **(CR)**는 **CDI(Containerized Data Importer)**에 대한 사용자 구성을 정의합니다. **CPU** 및 메모리 요청 및 한계 값은 기본값인 **0**으로 설정되어 있습니다. 이렇게 하면 컴퓨팅 리소스 요구 사항 없이 **CDI**에서 생성한 **Pod**에 기본값을 제공하고 할당량으로 제한되는 네임스페이스에서 해당 **Pod**를 실행할 수 있습니다.

AutoResourceLimits 기능 게이트가 활성화되면 **OpenShift Virtualization**은 **CPU** 및 메모리 제한을 자동으로 관리합니다. 네임스페이스에 **CPU** 및 메모리 할당량이 모두 있는 경우 메모리 제한이 기본 할당을 두 배로 설정하며 **CPU** 제한은 **vCPU**당 하나씩입니다.

9.7.2. CPU 및 메모리 기본값 덮어쓰기

spec.resourceRequirements.storageWorkloads 스탠자를 HyperConverged CR(사용자 정의 리소스)에 추가하여 CPU 및 메모리 요청의 기본 설정과 사용 사례에 대한 제한을 수정합니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.

절차

1. 다음 명령을 실행하여 HyperConverged CR을 편집합니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. spec.resourceRequirements.storageWorkloads 스탠자를 CR에 추가하여 사용 사례에 따라 값을 설정합니다. 예를 들면 다음과 같습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  resourceRequirements:
    storageWorkloads:
      limits:
        cpu: "500m"
        memory: "2Gi"
      requests:
        cpu: "250m"
        memory: "1Gi"
```

3. 편집기를 저장하고 종료하여 HyperConverged CR을 업데이트합니다.

9.7.3. 추가 리소스

- 프로젝트당 리소스 할당량

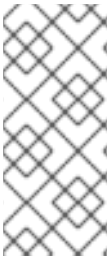
9.8. CDI 스크래치 공간 준비

9.8.1. 스크래치 공간 정보

CDI(Containerized Data Importer)에는 가상 머신 이미지 가져오기 및 업로드와 같은 일부 작업을 완료하기 위해 스크래치 공간(임시 스토리지)이 필요합니다. 이 프로세스 동안 CDI는 대상 DV(데이터 볼륨)를 지원하는 PVC 크기와 같은 스크래치 공간 PVC를 프로비저닝합니다. 스크래치 공간 PVC는 작업이 완료되거나 중단된 후 삭제됩니다.

HyperConverged사용자 지정 리소스의 spec.scratchSpaceStorageClass 필드에서 스크래치 공간 PVC를 바인딩하는 데 사용되는 스토리지 클래스를 정의할 수 있습니다.

정의된 스토리지 클래스가 클러스터의 스토리지 클래스와 일치하지 않으면 클러스터에 정의된 기본 스토리지 클래스가 사용됩니다. 클러스터에 기본 스토리지 클래스가 정의되어 있지 않은 경우에는 원래 DV 또는 PVC를 프로비저닝하는 데 사용된 스토리지 클래스가 사용됩니다.



참고

CDI에서는 원본 데이터 볼륨을 지원하는 PVC에 관계없이 file 볼륨 모드로 스크래치 공간을 요청해야 합니다. 원본 PVC를 block 볼륨 모드로 지원하는 경우 file 볼륨 모드 PVC를 프로비저닝할 수 있는 스토리지 클래스를 정의해야 합니다.

수동 프로비저닝

스토리지 클래스가 없는 경우 CDI는 프로젝트에서 이미지의 크기 요구 사항과 일치하는 PVC를 사용합니다. 이러한 요구 사항과 일치하는 PVC가 없는 경우에는 CDI 가져오기 Pod가 적절한 PVC를 사용할 수 있거나 타임아웃 기능에서 Pod를 종료할 때까지 Pending 상태로 유지됩니다.

9.8.2. 스크래치 공간이 필요한 CDI 작업

유형	이유
레지스트리 가져오기	CDI에서는 이미지를 스크래치 공간으로 다운로드하고 레이어를 추출하여 이미지 파일을 찾아야 합니다. 그런 다음 해당 이미지 파일을 원시 디스크로 변환하기 위해 QEMU-IMG로 전달합니다.
이미지 업로드	QEMU-IMG에서는 STDIN의 입력을 허용하지 않습니다. 대신 변환을 위해 QEMU-IMG로 전달할 수 있을 때까지 업로드할 이미지를 스크래치 공간에 저장합니다.
보관된 이미지의 HTTP 가져오기	QEMU-IMG에서는 CDI에서 지원하는 아카이브 형식 처리 방법을 확인할 수 없습니다. 대신, QEMU-IMG에 전달할 때까지 해당 이미지를 보관하지 않고 스크래치 공간에 저장합니다.

유형	이유
인증된 이미지의 HTTP 가져오기	QEMU-IMG에서 인증을 부적절하게 처리합니다. 대신, QEMU-IMG로 전달할 때까지 이미지를 스크래치 공간에 저장하고 인증합니다.
사용자 정의 인증서의 HTTP 가져오기	QEMU-IMG에서는 HTTPS 끝점의 사용자 정의 인증서를 부적절하게 처리합니다. 대신, CDI에서는 파일을 QEMU-IMG에 전달할 때까지 이미지를 스크래치 공간에 다운로드합니다.

9.8.3. 스토리지 클래스 정의

spec.scratchSpaceStorageClass 필드를 HyperConverged CR(사용자 정의 리소스)에 추가하여 CR(Containerized Data Importer)에서 스크래치 공간을 할당할 때 사용하는 스토리지 클래스를 정의할 수 있습니다.

사전 요구 사항

- **OpenShift CLI(oc)를 설치합니다.**

절차

1. **다음 명령을 실행하여 HyperConverged CR을 편집합니다.**

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **spec.scratchSpaceStorageClass 필드를 CR에 추가하여 해당 값을 클러스터에 존재하는 스토리지 클래스의 이름으로 설정합니다.**

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  scratchSpaceStorageClass: "<storage_class>" 1
```

1. **스토리지 클래스를 지정하지 않으면 CDI는 채워지는 영구 볼륨 클레임의 스토리지 클래스를 사용합니다.**

3. 기본 편집기를 저장하고 종료하여 **HyperConverged CR**을 업데이트합니다.

9.8.4. CDI 지원 작업 매트릭스

이 매트릭스에는 끝점에 대한 콘텐츠 유형에 따라 지원되는 **CDI** 작업과 이러한 작업 중 스크래치 공간이 필요한 작업이 표시되어 있습니다.

콘텐츠 유형	HTTP	HTTPS	HTTP 기본 인증	레지스트리	업로드
KubeVirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt(RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ 지원되는 작업

□ 지원되지 않는 작업

* 스크래치 공간 필요

** 사용자 정의 인증 기관이 필요한 경우 스크래치 공간 필요

9.8.5. 추가 리소스

- 동적 프로비저닝

9.9. 데이터 볼륨에 사전 할당 사용

CDI(Containerized Data Importer)는 데이터 볼륨을 생성할 때 쓰기 성능을 개선하기 위해 디스크 공간을 사전 할당할 수 있습니다.

특정 데이터 볼륨에 대해 사전 할당을 실행할 수 있습니다.

9.9.1. 사전 할당 정보

CDI(Containerized Data Importer)는 데이터 볼륨에 QEMU 사전 할당 모드를 사용하여 쓰기 성능을 향상시킬 수 있습니다. 사전 할당 모드를 사용하여 작업 가져오기 및 업로드 및 빈 데이터 볼륨을 생성할 때 사용할 수 있습니다.

사전 할당이 활성화된 경우 CDI는 기본 파일 시스템 및 장치 유형에 따라 더 나은 사전 할당 방법을 사용합니다.

fallocate

파일 시스템이 이를 지원하는 경우, CDI는 posix_fallocate 함수를 사용하여 운영 체제의 fallocate 호출을 통해 공간을 미리 할당하며, 이를 통해 블록을 할당하고 초기화되지 않음으로 표시합니다.

full

fallocate 모드를 사용할 수 없는 경우 full 모드는 기본 스토리지에 데이터를 작성하여 이미지의 공간을 할당합니다. 스토리지 위치에 따라, 비어 있는 할당된 모든 공간을 0으로 만들 수 있습니다.

9.9.2. 데이터 볼륨 사전 할당 활성화

데이터 볼륨 매니페스트에 spec.preallocation 필드를 포함하여 특정 데이터 볼륨에 대한 사전 할당을 활성화할 수 있습니다. 웹 콘솔에서 또는 OpenShift 클라이언트(oc)를 사용하여 사전 할당 모드를 활성화할 수 있습니다.

모든 CDI 소스 유형에서 사전 할당 모드가 지원됩니다.

절차

- 데이터 볼륨 매니페스트에 spec.preallocation 필드를 지정합니다.

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: preallocated-datavolume
spec:
```

```
  source: (1)
    registry:
      url: <image_url> (2)
  storage:
    resources:
      requests:
        storage: 1Gi
# ...
```

(1)

모든 CDI 소스 유형은 사전 할당을 지원합니다. 그러나 복제 작업에는 사전 할당이 무시됩니다.

(2)

레지스트리에 있는 데이터 소스의 URL을 지정합니다.

9.10. 데이터 볼륨 주석 관리

DV(데이터 볼륨) 주석을 사용하면 Pod 동작을 관리할 수 있습니다. 하나 이상의 주석을 데이터 볼륨에 추가하면 생성된 가져오기 Pod로 전파할 수 있습니다.

9.10.1. 예: 데이터 볼륨 주석

이 예에서는 가져오기 Pod에서 사용하는 네트워크를 제어하도록 DV(데이터 볼륨) 주석을 구성할 수 있는 방법을 보여줍니다. v1.multus-cni.io/default-network: bridge-network 주석을 사용하면 Pod에서 bridge-network라는 multus 네트워크를 기본 네트워크로 사용합니다. 가져오기 Pod에서 클러스터 및 보조 multus 네트워크의 기본 네트워크를 모두 사용하도록 하려면 k8s.v1.cni.cncf.io/networks: <network_name> 주석을 사용합니다.

Multus 네트워크 주석 예

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: datavolume-example
  annotations:
    v1.multus-cni.io/default-network: bridge-network (1)
# ...
```

1

Multus 네트워크 주석

10장. 실시간 마이그레이션

10.1. 실시간 마이그레이션 정보

실시간 마이그레이션은 가상 워크로드를 중단하지 않고 실행 중인 VM(가상 머신)을 클러스터의 다른 노드로 이동하는 프로세스입니다. 기본적으로 실시간 마이그레이션 트래픽은 TLS(Transport Layer Security)를 사용하여 암호화됩니다.

10.1.1. 실시간 마이그레이션 요구 사항

실시간 마이그레이션에는 다음과 같은 요구 사항이 있습니다.

- 클러스터에 RWX(ReadWriteMany) 액세스 모드를 사용한 공유 스토리지가 있어야 합니다.
- 클러스터에 충분한 RAM 및 네트워크 대역폭이 있어야 합니다.

참고

노드 드레이닝을 지원하기 위해 클러스터에 메모리 요청 용량이 충분한지 확인하여 실시간 마이그레이션을 수행해야 합니다. 다음 계산을 사용하여 필요한 예비 메모리를 확인할 수 있습니다.

> Product of (Maximum number of nodes that can drain in parallel) and (Highest total VM memory request allocations across nodes)

클러스터에서 병렬로 실행할 수 있는 기본 마이그레이션 수는 5입니다.

- VM에서 호스트 모델 CPU를 사용하는 경우 노드에서 CPU를 지원해야 합니다.
- 실시간 마이그레이션 을 위한 전용 Multus 네트워크를 구성하는 것이 좋습니다. 전용 네트워크는 마이그레이션 중에 테넌트 워크로드에 대한 네트워크 포화 상태로 인한 영향을 최소화합니다.

10.1.2. 일반적인 실시간 마이그레이션 작업

다음 실시간 마이그레이션 작업을 수행할 수 있습니다.

- 실시간 마이그레이션 설정 구성
- 실시간 마이그레이션 시작 및 취소
- **OpenShift Virtualization** 웹 콘솔의 마이그레이션 탭에서 모든 실시간 마이그레이션의 진행 상황을 모니터링합니다.
- 웹 콘솔의 **Metrics** 탭에서 **VM** 마이그레이션 지표를 확인합니다.

10.1.3. 추가 리소스

- 실시간 마이그레이션을 위한 **Prometheus** 쿼리
- **VM** 마이그레이션 튜닝
- **VM** 실행 전략
- **VM** 및 클러스터 제거 전략

10.2. 실시간 마이그레이션 구성

마이그레이션 프로세스가 클러스터를 압도하지 않도록 실시간 마이그레이션 설정을 구성할 수 있습니다.

실시간 마이그레이션 정책을 구성하여 **VM**(가상 머신) 그룹에 다른 마이그레이션 구성을 적용할 수 있습니다.

10.2.1. 실시간 마이그레이션 제한 및 타임아웃 구성

openshift-cnv 네임스페이스에 있는 **HyperConverged CR**(사용자 정의 리소스)을 업데이트하여 클

러스터의 실시간 마이그레이션 제한 및 타임아웃을 구성합니다.

절차

- HyperConverged CR을 편집하고 필요한 실시간 마이그레이션 매개변수를 추가합니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

설정 파일 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  liveMigrationConfig:
    bandwidthPerMigration: 64Mi (1)
    completionTimeoutPerGiB: 800 (2)
    parallelMigrationsPerCluster: 5 (3)
    parallelOutboundMigrationsPerNode: 2 (4)
    progressTimeout: 150 (5)
```

(1)

각 마이그레이션의 대역폭 제한입니다. 여기서 값은 초당 바이트 수입니다. 예를 들어, 2048Mi 값은 2048MiB/s를 의미합니다. 기본값: 0, 이는 무제한입니다.

(2)

이 시점에 메모리 GiB당 초 단위로 마이그레이션이 완료되지 않으면 마이그레이션이 취소됩니다. 예를 들어 메모리가 6GiB인 VM은 4800초 내에 마이그레이션이 완료되지 않으면 시간 초과됩니다. Migration Method가 BlockMigration인 경우 마이그레이션 디스크의 크기가 계산에 포함됩니다.

(3)

클러스터에서 병렬로 실행되고 있는 마이그레이션의 수입니다. 기본값: 5.

(4)

5

이 시간(초) 내에 메모리 복사를 진행하지 못하면 마이그레이션이 취소됩니다. 기본값: 150.

참고

해당 키/값 쌍을 삭제하고 파일을 저장하여 spec.liveMigrationConfig 필드의 기본값을 복원할 수 있습니다. 예를 들어 progressTimeout: <value>를 삭제하여 기본 progressTimeout: 150을 복원합니다.

10.2.2. 실시간 마이그레이션 정책

실시간 마이그레이션 정책을 생성하여 VM 또는 프로젝트 레이블로 정의된 VM 그룹에 다른 마이그레이션 구성을 적용할 수 있습니다.

작은 정보

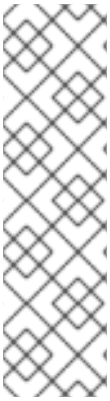
OpenShift Virtualization 웹 콘솔을 사용하여 실시간 마이그레이션 정책을 생성할 수 있습니다.

10.2.2.1. 명령줄을 사용하여 실시간 마이그레이션 정책 생성

명령줄을 사용하여 실시간 마이그레이션 정책을 생성할 수 있습니다. 실시간 마이그레이션 정책은 라벨 조합을 사용하여 선택한 VM(가상 머신)에 적용됩니다.

- VM 레이블(예: 크기,os 또는 gpu)
- 우선순위,대역폭 또는 hpc-workload와 같은 프로젝트 라벨

정책이 특정 VM 그룹에 적용되려면 VM 그룹의 모든 레이블이 정책 레이블과 일치해야 합니다.



참고

VM에 여러 실시간 마이그레이션 정책이 적용되는 경우 일치하는 라벨이 가장 많은 정책이 우선합니다.

여러 정책이 이 기준을 충족하는 경우 정책은 일치하는 레이블 키의 알파벳 순서에 따라 정렬되며 해당 순서의 첫 번째 정책이 우선합니다.

프로세스

1. 다음 예와 같이 **MigrationPolicy** 오브젝트를 생성합니다.

```
apiVersion: migrations.kubevirt.io/v1alpha1
kind: MigrationPolicy
metadata:
  name: <migration_policy>
spec:
  selectors:
    namespaceSelector: 1
      hpc-workloads: "True"
      xyz-workloads-type: ""
    virtualMachineInstanceSelector: 2
      workload-type: "db"
      operating-system: ""
```

1 프로젝트 레이블을 지정합니다.

2 VM 레이블을 지정합니다.

2. 다음 명령을 실행하여 마이그레이션 정책을 생성합니다.

```
$ oc create migrationpolicy -f <migration_policy>.yaml
```

10.2.3. 추가 리소스

- 실시간 마이그레이션을 위한 전용 Multus 네트워크 구성

10.3. 실시간 마이그레이션 시작 및 취소

OpenShift Container Platform 웹 콘솔 또는 명령줄 을 사용하여 VM(가상 머신)의 실시간 마이그레이션을 시작할 수 있습니다.

웹 콘솔 또는 명령줄 을 사용하여 실시간 마이그레이션을 취소할 수 있습니다. VM은 원래 노드에 남아 있습니다.

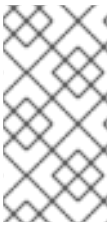
작은 정보

virtctl migrate <vm_name> 및 virtctl migrate-cancel <vm_name > 명령을 사용하여 실시간 마이그레이션을 시작하고 취소할 수도 있습니다.

10.3.1. 실시간 마이그레이션 시작

10.3.1.1. 웹 콘솔을 사용하여 실시간 마이그레이션 시작

OpenShift Container Platform 웹 콘솔을 사용하여 실행 중인 VM(가상 머신)을 클러스터의 다른 노드로 실시간 마이그레이션할 수 있습니다.



참고

마이그레이션 작업은 모든 사용자에게 표시되지만 클러스터 관리자만 실시간 마이그레이션을 시작할 수 있습니다.

사전 요구 사항

- VM이 편두워야 합니다.
- VM이 호스트 모델 CPU로 구성된 경우 클러스터에 CPU 모델을 지원하는 사용 가능한 노드가 있어야 합니다.

프로세스

1. 웹 콘솔에서 가상화 → VirtualMachines 로 이동합니다.
2. VM 옆에 있는 옵션 메뉴

에서 마이그레이션 을 선택합니다.

3. 마이그레이션을 클릭합니다.

10.3.1.2. 명령줄을 사용하여 실시간 마이그레이션 시작

명령줄을 사용하여 VM에 대한 VirtualMachineInstanceMigration 오브젝트를 생성하여 실행 중인 VM(가상 머신)의 실시간 마이그레이션을 시작할 수 있습니다.

프로세스

1. 마이그레이션할 VM에 대한 VirtualMachineInstanceMigration 매니페스트를 생성합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: <migration_name>
spec:
  vmiName: <vm_name>
```

2. 다음 명령을 실행하여 오브젝트를 생성합니다.

```
$ oc create -f <migration_name>.yaml
```

VirtualMachineInstanceMigration 오브젝트는 VM의 실시간 마이그레이션을 트리거합니다. 이 오브젝트는 수동으로 삭제하지 않는 한 가상 머신 인스턴스가 실행되는 동안 클러스터에 존재합니다.

검증

- 다음 명령을 실행하여 VM 상태를 가져옵니다.

```
$ oc describe vmi <vm_name> -n <namespace>
```

출력 예

```
# ...
Status:
  Conditions:
    Last Probe Time:       <nil>
    Last Transition Time:  <nil>
    Status:                True
    Type:                  LiveMigratable
  Migration Method:  LiveMigration
  Migration State:
    Completed:                true
    End Timestamp:            2018-12-24T06:19:42Z
    Migration UID:            d78c8962-0743-11e9-a540-fa163e0c69f1
    Source Node:              node2.example.com
    Start Timestamp:          2018-12-24T06:19:35Z
    Target Node:              node1.example.com
    Target Node Address:      10.9.0.18:43891
    Target Node Domain Detected:  true
```

10.3.2. 실시간 마이그레이션 취소

10.3.2.1. 웹 콘솔을 사용하여 실시간 마이그레이션 취소

OpenShift Container Platform 웹 콘솔을 사용하여 **VM**(가상 머신)의 실시간 마이그레이션을 취소할 수 있습니다.

프로세스

1. 웹 콘솔에서 가상화 → **VirtualMachines** 로 이동합니다.
2. **VM** 옆에 있는 옵션 메뉴 ⋮ 에서 마이그레이션 취소 를 선택합니다.

10.3.2.2. 명령줄을 사용하여 실시간 마이그레이션 취소

마이그레이션과 연결된 **VirtualMachineInstanceMigration** 오브젝트를 삭제하여 가상 머신의 실시간 마이그레이션을 취소합니다.

프로세스

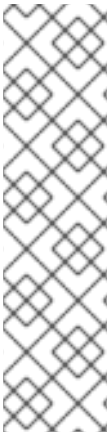
- 이 예제에서 실시간 마이그레이션 작업인 **migration-job**을 트리거한 **VirtualMachineInstanceMigration** 오브젝트를 삭제합니다.

```
$ oc delete vmim migration-job
```

11장. 노드

11.1. 노드 유지보수

oc adm 유틸리티 또는 NodeMaintenance CR(사용자 정의 리소스)을 사용하여 노드를 유지보수 모드로 전환할 수 있습니다.



참고

node-maintenance-operator (NMO)는 더 이상 OpenShift Virtualization과 함께 제공되지 않습니다. OpenShift Container Platform 웹 콘솔의 OperatorHub 에서 독립형 Operator로 배포되거나 OpenShift CLI(oc)를 사용하여 배포됩니다.

노드 수정, 펜싱 및 유지 관리에 대한 자세한 내용은 Workload Availability for Red Hat OpenShift 설명서를 참조하십시오.

중요

VM(가상 머신)에는 실시간 마이그레이션할 공유 RWX(ReadWriteMany) 액세스 모드가 있는 PVC(영구 볼륨 클레임)가 있어야 합니다.

Node Maintenance Operator는 신규 또는 삭제된 NodeMaintenance CR을 감시합니다. 새 NodeMaintenance CR이 감지되면 새 워크로드가 예약되지 않고 나머지 클러스터에서 노드가 차단됩니다. 제거할 수 있는 모든 Pod는 노드에서 제거됩니다. NodeMaintenance CR이 삭제되면 CR에서 참조되는 노드를 새 워크로드에 사용할 수 있습니다.

참고

노드 유지관리 작업에 NodeMaintenance CR을 사용하면 표준 OpenShift Container Platform 사용자 정의 리소스 처리를 사용하여 oc adm cordon 및 oc adm drain 명령과 동일한 결과를 얻을 수 있습니다.

11.1.1. 제거 전략

노드를 유지보수 모드에 배치하면 노드가 예약 불가로 표시되고 해당 노드에서 모든 VM과 Pod가 드레이닝됩니다.

VM(가상 머신) 또는 클러스터에 대한 제거 전략을 구성할 수 있습니다.

VM 제거 전략

VM LiveMigrate 제거 전략을 사용하면 노드가 유지보수 모드에 배치되거나 드레인된 경우 VMI(가상 머신 인스턴스)가 중단되지 않습니다. 이 제거 전략이 있는 VMI는 다른 노드로 실시간 마이그레이션됩니다.

OpenShift Virtualization 웹 콘솔 또는 명령줄 을 사용하여 VM(가상 머신)에 대한 제거 전략을 구성할 수 있습니다.

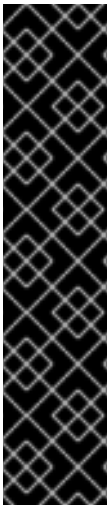
중요

기본 제거 전략은 LiveMigrate 입니다. LiveMigrate 제거 전략이 있는 비정상적인 VM으로 인해 VM이 노드에서 제거되지 않기 때문에 노드가 인프라 업그레이드를 드레이닝하거나 차단하지 못할 수 있습니다. 이 경우 VM을 수동으로 종료하지 않는 한 마이그레이션이 Pending 또는 Scheduling 상태로 유지됩니다.

업그레이드할 수 없는 VM의 제거 전략을 LiveMigrateIfPossible 로 설정해야 합니다. 이 경우 업그레이드를 차단하지 않거나 마이그레이션하지 않아야 하는 VM의 경우 None 으로 설정해야 합니다.

클러스터 제거 전략

워크로드 연속성 또는 인프라 업그레이드의 우선 순위를 지정하도록 클러스터에 대한 제거 전략을 구성할 수 있습니다.



중요

클러스터 제거 전략을 구성하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

표 11.1. 클러스터 제거 전략

제거 전략	설명	인터럽트 워크플로	블록 업그레이드
LiveMigrate[1]	업그레이드에 비해 워크로드 연속성에 우선 순위를 지정합니다.	없음	제공됨[2]
LiveMigrateIfPossible	환경이 업데이트되도록 워크로드 연속성에 대한 업그레이드 우선 순위를 지정합니다.	제공됨	없음
없음[3]	제거 전략이 없는 VM을 종료합니다.	제공됨	없음

1. **다중 노드 클러스터의 기본 제거 전략**

2. **VM에서 업그레이드를 차단하는 경우 VM을 수동으로 종료해야 합니다.**

3. **단일 노드 OpenShift의 기본 제거 전략입니다.**

11.1.1.1. 명령줄을 사용하여 VM 제거 전략 구성

명령줄을 사용하여 VM(가상 머신)에 대한 제거 전략을 구성할 수 있습니다.

중요

기본 제거 전략은 LiveMigrate 입니다. LiveMigrate 제거 전략이 있는 비정상적인 VM으로 인해 VM이 노드에서 제거되지 않기 때문에 노드가 인프라 업그레이드를 드레이닝하거나 차단하지 못할 수 있습니다. 이 경우 VM을 수동으로 종료하지 않는 한 마이그레이션이 Pending 또는 Scheduling 상태로 유지됩니다.

업그레이드할 수 없는 VM의 제거 전략을 LiveMigrateIfPossible 로 설정해야 합니다. 이 경우 업그레이드를 차단하지 않거나 마이그레이션하지 않아야 하는 VM의 경우 None 으로 설정해야 합니다.

프로세스

1. **다음 명령을 실행하여 VirtualMachine 리소스를 편집합니다.**

```
$ oc edit vm <vm_name> -n <namespace>
```

제거 전략의 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <vm_name>
spec:
  template:
    spec:
      evictionStrategy: LiveMigrateIfPossible (1)
# ...
```

(1) 제거 전략을 지정합니다. 기본값은 **LiveMigrate** 입니다.

2. **VM**을 다시 시작하여 변경 사항을 적용합니다.

```
$ virtctl restart <vm_name> -n <namespace>
```

11.1.1.2. 명령줄을 사용하여 클러스터 제거 전략 구성

명령줄을 사용하여 클러스터에 대한 제거 전략을 구성할 수 있습니다.

중요

클러스터 제거 전략을 구성하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

프로세스

1. 다음 명령을 실행하여 하이퍼컨버지드 리소스를 편집합니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 다음 예와 같이 클러스터 제거 전략을 설정합니다.

클러스터 제거 전략의 예

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  evictionStrategy: LiveMigrate
# ...
```

11.1.2. 전략 실행

spec.running: true 로 구성된 VM(가상 머신)이 즉시 재시작됩니다. spec.runStrategy 키는 특정 조건에서 VM이 작동하는 방식을 결정할 수 있는 유연성을 제공합니다.

중요

spec.runStrategy 및 spec.running 키는 함께 사용할 수 없습니다. 이 중 하나만 사용할 수 있습니다.

두 키가 모두 있는 VM 구성이 잘못되었습니다.

11.1.2.1. 전략 실행

spec.runStrategy 키에는 네 가지 가능한 값이 있습니다.

Always

다른 노드에서 VM(가상 머신)이 생성될 때 VMI(가상 머신 인스턴스)가 항상 존재합니다. 어떤 이

유로든 원래 VMI가 중지되면 새 VMI가 생성됩니다. 이 동작은 running: true 와 동일합니다.

RerunOnFailure

이전 인스턴스가 실패하면 다른 노드에서 VMI가 다시 생성됩니다. VM이 종료될 때와 같이 성공적으로 중지되면 인스턴스가 다시 생성되지 않습니다.

수동

start,stop, restart virtctl 클라이언트 명령을 사용하여 VMI 상태를 수동으로 제어합니다. VM이 자동으로 다시 시작되지 않습니다.

Halted

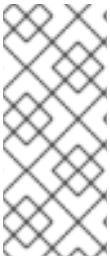
VM이 생성될 때 VMI가 존재하지 않습니다. 이 동작은 running: false 와 동일합니다.

virtctl start,stop 및 restart 명령의 다양한 조합은 실행 전략에 영향을 미칩니다.

다음 표에서는 상태 간 VM의 전환을 설명합니다. 첫 번째 열에는 VM의 초기 실행 전략이 표시되어 있습니다. 나머지 열에는 virtctl 명령 및 해당 명령이 실행된 후 새 실행 전략이 표시됩니다.

표 11.2. virtctl 명령 전후에 전략 실행

초기 실행 전략	시작	중지	재시작
Always	-	Halted	Always
RerunOnFailure	-	Halted	RerunOnFailure
Manual	Manual	Manual	Manual
Halted	Always	-	-



참고

설치 관리자 프로비저닝 인프라를 사용하여 설치한 클러스터의 노드가 머신 상태 점검에 실패하고 사용할 수 없는 경우 runStrategy: Always 또는 runStrategy: RerunOnFailure 가 새 노드에 다시 예약됩니다.

11.1.2.2. 명령줄을 사용하여 VM 실행 전략 구성

명령줄을 사용하여 VM(가상 머신)에 대한 실행 전략을 구성할 수 있습니다.

중요

spec.runStrategy 및 spec.running 키는 함께 사용할 수 없습니다. 두 키의 값이 포함된 VM 구성은 유효하지 않습니다.

프로세스

- 다음 명령을 실행하여 VirtualMachine 리소스를 편집합니다.

```
$ oc edit vm <vm_name> -n <namespace>
```

실행 전략 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  runStrategy: Always
# ...
```

11.1.3. 베어 메탈 노드 유지관리

베어 메탈 인프라에 OpenShift Container Platform을 배포할 때 클라우드 인프라에 배포하는 것과 비교하여 고려해야 할 추가 고려 사항이 있습니다. 클러스터 노드가 사용 후 삭제로 간주되는 클라우드 환경에서와 달리 베어 메탈 노드를 다시 프로비저닝하려면 유지관리 작업에 더 많은 시간과 노력이 필요합니다.

예를 들어 치명적인 커널 오류가 발생하거나 NIC 카드 하드웨어 장애가 발생하는 것과 같이 베에메탈 노드에 장애가 발생한 경우 문제가 발생한 노드가 복구 되거나 교체되는 동안 장애가 발생한 노드의 워크로드를 클러스터의 다른 곳에서 다시 시작해야 합니다. 클러스터 관리자는 노드 유지관리 모드를 통해 노드의 전원을 정상적으로 끄고 워크로드를 클러스터의 다른 부분으로 이동하여 워크로드가 중단되지 않도록 할 수 있습니다. 유지보수 관리 중에 자세한 진행 상황 및 노드 상태 세부 정보가 제공됩니다.

11.1.4. 추가 리소스

- 실시간 마이그레이션 정보

11.2. 더 이상 사용되지 않는 CPU 모델에 대한 노드 라벨링 관리

노드에서 VM CPU 모델 및 정책을 지원하는 경우 노드에서 VM(가상 머신)을 예약할 수 있습니다.

11.2.1. 더 이상 사용되지 않는 CPU 모델에 대한 노드 레이블 설정 정보

OpenShift Virtualization Operator는 사용되지 않는 CPU 모델의 미리 정의된 목록을 사용하여 노드가 스케줄링된 VM에 유효한 CPU 모델만 지원하도록 합니다.

기본적으로 다음 CPU 모델은 노드에 대해 생성된 레이블 목록에서 제거됩니다.

예 11.1. 더 이상 사용되지 않는 CPU 모델

```
"486"
Conroe
athlon
core2duo
coreduo
kvm32
kvm64
n270
pentium
pentium2
pentium3
pentiumpro
phenom
qemu32
qemu64
```

이 사전 정의된 목록은 HyperConverged CR에 표시되지 않습니다. 이 목록에서 CPU 모델을 *제거할* 수는 없지만 HyperConverged CR의 spec.obsolicallyCPUs.cpuModels 필드를 편집하여 목록에 추가할 수 있습니다.

11.2.2. CPU 기능의 노드 레이블링 정보

반복 프로세스를 거치는 동안 최소 CPU 모델의 기본 CPU 기능이 노드에 대해 생성되는 라벨 목록에서 제거됩니다.

예를 들면 다음과 같습니다.

- 환경에 두 가지 CPU 모델, Penryn 및 Haswell이 지원될 수 있습니다.

- **Penryn이 minCPU의 CPU 모델로 지정되면 Penryn의 각 기본 CPU 기능은 Haswell에서 지원하는 각 CPU 기능 목록과 비교됩니다.**

예 11.2. Penryn에서 지원하는 CPU 기능

```
apic
clflush
cmov
cx16
cx8
de
fpu
fxsr
lahf_lm
lm
mca
mce
mmx
msr
mtrr
nx
pae
pat
pge
pni
pse
pse36
sep
sse
sse2
sse4.1
ssse3
syscall
tsc
```

예 11.3. Haswell에서 지원하는 CPU 기능

```
aes
apic
avx
avx2
bmi1
bmi2
clflush
cmov
cx16
cx8
de
erms
fma
fpu
fsgsbase
```

```
fxsr
hle
invpcid
lahf_lm
lm
mca
mce
mmx
movbe
msr
mtrr
nx
pae
pat
pcid
pclmuldq
pge
pni
popcnt
pse
pse36
rdtscp
rtm
sep
smep
sse
sse2
sse4.1
sse4.2
ssse3
syscall
tsc
tsc-deadline
x2apic
xsave
```

- **Penryn** 및 **Haswell**이 특정 **CPU** 기능을 모두 지원하면 해당 기능에 대한 레이블이 생성되지 않습니다. 라벨은 **Haswell**에서만 지원하고 **Penryn**에서는 지원하지 않는 **CPU** 기능에 대해 생성됩니다.

예 11.4. CPU 기능 반복 후 생성된 노드 레이블

```
aes
avx
avx2
bmi1
bmi2
erms
fma
fsgsbase
hle
invpcid
movbe
```

```
pcid
pclmuldq
popcnt
rdtscp
rtm
sse4.2
tsc-deadline
x2apic
xsave
```

11.2.3. 더 이상 사용되지 않는 CPU 모델 구성

HyperConverged CR(사용자 정의 리소스)을 편집하여 더 이상 사용되지 않는 **CPU** 모델 목록을 구성할 수 있습니다.

절차

- **HyperConverged** 사용자 지정 리소스를 편집하여 **obsoleteCPUs** 배열에 더 이상 사용되지 않는 **CPU** 모델을 지정합니다. 예를 들면 다음과 같습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  obsoleteCPUs:
    cpuModels: 1
      - "<obsolete_cpu_1>"
      - "<obsolete_cpu_2>"
    minCPUModel: "<minimum_cpu_model>" 2
```

1

cpuModels 배열의 예제 값을 더 이상 사용되지 않는 **CPU** 모델로 교체합니다. 지정한 모든 값은 더 이상 사용되지 않는 **CPU** 모델에 사전 정의된 목록에 추가됩니다. 사전 정의된 목록은 **CR**에 표시되지 않습니다.

2

이 값을 기본 **CPU** 기능에 사용할 최소 **CPU** 모델로 바꿉니다. 값을 지정하지 않으면 기본적으로 **Penryn**이 사용됩니다.

11.3. 노드 조정 방지

node-labeller가 노드를 조정하지 못하도록 하려면 skip-node 주석을 사용합니다.

11.3.1. skip-node 주석 사용

node-labeller에서 노드를 건너뛰려면 oc CLI를 사용하여 해당 노드에 주석을 답니다.

사전 요구 사항

- OpenShift CLI(oc)가 설치되어 있습니다.

절차

- 다음 명령을 실행하여 건너뛰려는 노드에 주석을 답니다.

```
$ oc annotate node <node_name> node-labeller.kubevirt.io/skip-node=true 1
```

1

<node_name>을 건너뛸 관련 노드의 이름으로 바꿉니다.

노드 주석을 제거하거나 false로 설정한 후 다음 주기에서 조정이 재개됩니다.

11.3.2. 추가 리소스

- 더 이상 사용되지 않는 CPU 모델에 대한 노드 라벨링 관리

11.4. 가상 머신 장애 조치를 트리거하도록 실패한 노드 삭제

노드가 실패하고 머신 상태 점검 이 클러스터에 배포되지 않으면 runStrategy: Always 가 구성된 VM(가상 머신)이 정상 노드에 자동으로 재배치되지 않습니다. VM 장애 조치를 트리거하려면 Node 오브젝트를 수동으로 삭제해야 합니다.

참고

설치 관리자 프로비저닝 인프라 를 사용하여 클러스터를 설치하고 머신 상태 점검을 올바르게 구성한 경우 다음 이벤트가 발생합니다.

- 실패한 노드는 자동으로 재활용됩니다.
- runStrategy 가 Always 또는 RerunOnFailure 로 설정된 가상 머신은 정상 노드에 자동으로 예약됩니다.

11.4.1. 사전 요구 사항

- 가상 머신이 실행 중이었던 노드에 NotReady 조건이 있습니다.
- 실패한 노드에서 실행 중이던 가상 머신의 runStrategy 가 Always 로 설정되어 있습니다.
- OpenShift CLI(oc)가 설치되어 있습니다.

11.4.2. 베어 메탈 클러스터에서 노드 삭제

CLI를 사용하여 노드를 삭제하면 Kubernetes에서 노드 오브젝트가 삭제되지만 노드에 존재하는 Pod는 삭제되지 않습니다. 복제 컨트롤러에서 지원하지 않는 기본 Pod는 OpenShift Container Platform에 액세스할 수 없습니다. 복제 컨트롤러에서 지원하는 Pod는 사용 가능한 다른 노드로 다시 예약됩니다. 로컬 매니페스트 Pod를 삭제해야 합니다.

절차

다음 단계를 완료하여 베어 메탈에서 실행 중인 OpenShift Container Platform 클러스터에서 노드를 삭제합니다.

1. 노드를 예약 불가능으로 표시합니다.

```
$ oc adm cordon <node_name>
```

2. 노드의 모든 Pod를 드레이닝합니다.

```
$ oc adm drain <node_name> --force=true
```

노드가 오프라인 상태이거나 응답하지 않는 경우 이 단계가 실패할 수 있습니다. 노드가 응답하지 않더라도 공유 스토리지에 쓰는 워크로드를 계속 실행되고 있을 수 있습니다. 데이터 손상을 방지하려면 계속하기 전에 물리적 하드웨어의 전원을 끕니다.

3. 클러스터에서 노드를 삭제합니다.

```
$ oc delete node <node_name>
```

노드 오브젝트가 클러스터에서 삭제되어도 재부팅 후 또는 **kubelet** 서비스가 재시작되면 클러스터에 다시 참여할 수 있습니다. 노드와 노드의 모든 데이터를 영구적으로 삭제하려면 노드를 해제해야 합니다.

4. 물리 하드웨어의 전원을 끈 경우 노드가 클러스터에 다시 참여할 수 있도록 해당 하드웨어를 다시 켭니다.

11.4.3. 가상 머신 장애 조치 확인

비정상 노드에서 모든 리소스가 종료되면 **VM**이 재배치될 때마다 정상 노드에 새 **VMI**(가상 머신 인스턴스)가 자동으로 생성됩니다. **VMI**가 생성되었는지 확인하려면 **oc CLI**를 사용하여 모든 **VMI**를 확인합니다.

11.4.3.1. CLI를 사용하여 모든 가상 머신 인스턴스 나열

oc CLI(명령줄 인터페이스)를 사용하면 독립 실행형 **VMI**(가상 머신 인스턴스) 및 가상 머신에 속하는 **VMI**를 포함하여 클러스터의 모든 **VMI**를 나열할 수 있습니다.

절차

- 다음 명령을 실행하여 **VMI**를 모두 나열합니다.

```
$ oc get vmis -A
```

12장. 모니터링

12.1. 모니터링 개요

다음 툴을 사용하여 클러스터 및 VM(가상 머신)의 상태를 모니터링할 수 있습니다.

OpenShift Virtualization VM 상태 모니터링

OpenShift Container Platform 웹 콘솔의 홈 → 개요 페이지로 이동하여 웹 콘솔에서 OpenShift Virtualization 환경의 전반적인 상태를 확인합니다. 상태 카드는 경고 및 조건에 따라 OpenShift Virtualization의 전반적인 상태를 표시합니다.

OpenShift Container Platform 클러스터 검사 프레임워크

OpenShift Container Platform 클러스터 점검 프레임워크를 사용하여 클러스터에서 자동화된 테스트를 실행하여 다음 조건을 확인합니다.

- 보조 네트워크 인터페이스에 연결된 두 VM 간의 네트워크 연결 및 대기 시간
- 패킷 손실이 없는 DPDK(Data Plane Development Kit) 워크로드를 실행하는 VM
- OpenShift Virtualization에 대해 클러스터 스토리지가 최적으로 구성됨

가상 리소스에 대한 Prometheus 쿼리

vCPU, 네트워크, 스토리지 및 게스트 메모리 스왑 사용 및 실시간 마이그레이션 진행 상황을 쿼리합니다.

VM 사용자 정의 지표

내부 VM 지표 및 프로세스를 노출하도록 node-exporter 서비스를 구성합니다.

VM 상태 점검

준비 상태, 활성 상태, 게스트 에이전트 ping 프로브 및 VM 워치독을 구성합니다.

Runbooks

OpenShift Container Platform 웹 콘솔에서 OpenShift Virtualization 경고를 트리거하는 문제를 진단하고 해결합니다.

12.2. OPENSHIFT VIRTUALIZATION 클러스터 검사 프레임워크

OpenShift Virtualization에는 클러스터 유지 관리 및 문제 해결에 사용할 수 있는 다음과 같은 사전 정의된 점검이 포함되어 있습니다.

대기 시간 점검

보조 네트워크 인터페이스에 연결된 두 개의 VM(가상 머신) 간에 네트워크 연결을 확인하고 대기 시간을 측정합니다.

DPDK 점검

노드가 패킷 손실이 0인 DPDK(Data Plane Development Kit) 워크로드로 VM을 실행할 수 있는지 확인합니다.

스토리지 점검

클러스터 스토리지가 OpenShift Virtualization에 대해 최적으로 구성되었는지 확인합니다.

중요

OpenShift Virtualization 클러스터 검사 프레임워크는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

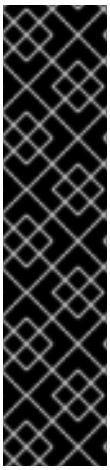
12.2.1. OpenShift Virtualization 클러스터 검사 프레임워크 정보

*점검*은 특정 클러스터 기능이 예상대로 작동하는지 확인할 수 있는 자동화된 테스트 워크로드입니다. 클러스터 검사 프레임워크는 기본 Kubernetes 리소스를 사용하여 점검을 구성하고 실행합니다.

클러스터 관리자와 개발자는 사전 정의된 점검을 사용하여 클러스터 유지 관리를 개선하고, 예기치 않은 동작을 해결하고, 오류를 최소화하고, 시간을 절약할 수 있습니다. 또한 수표 결과를 검토하고 추가 분석을 위해 전문가와 공유할 수 있습니다. 공급업체는 제공하는 기능 또는 서비스에 대한 점검을 작성하고 게시하고 고객 환경이 올바르게 구성되었는지 확인할 수 있습니다.

기존 네임스페이스에서 사전 정의된 점검을 실행하려면 점검에 대한 서비스 계정을 설정하고, 서비스

계정에 대한 **Role** 및 **RoleBinding** 오브젝트를 생성하고, 점검 권한을 활성화하고, 입력 구성 맵 및 점검 작업을 생성해야 합니다. 점검을 여러 번 실행할 수 있습니다.



중요

항상 다음을 수행해야 합니다.

- 검사를 적용하기 전에 검사 이미지가 신뢰할 수 있는 소스의인지 확인합니다.
- **Role** 및 **RoleBinding** 오브젝트를 생성하기 전에 점검 권한을 검토합니다.

12.2.2. 웹 콘솔에서 클러스터 검사 실행

웹 콘솔을 사용하여 클러스터에서 대기 시간 또는 스토리지 점검을 실행합니다.

웹 콘솔에서 대기 시간 점검 및 스토리지 점검을 처음 실행할 때 다음 절차를 사용하십시오. 추가 점검의 경우 두 점검 탭에서 점검 실행을 클릭하고 드롭다운 메뉴에서 적절한 점검을 선택합니다.

중요

대기 시간 점검을 실행하기 전에 먼저 **VM**의 보조 인터페이스를 노드의 인터페이스에 연결하려면 클러스터 노드에 브리지 인터페이스를 생성해야 합니다. 브리지 인터페이스를 생성하지 않으면 **VM**이 시작되지 않고 작업이 실패합니다.

12.2.2.1. 웹 콘솔에서 대기 시간 점검 실행

대기 시간 점검을 실행하여 네트워크 연결을 확인하고 보조 네트워크 인터페이스에 연결된 두 가상 머신 간의 대기 시간을 측정합니다.

사전 요구 사항

- **NetworkAttachmentDefinition** 을 네임스페이스에 추가해야 합니다.

프로세스

1. 웹 콘솔에서 가상화 → 확인 으로 이동합니다.

2. 네트워크 대기 시간 탭을 클릭합니다.

3. 권한 설치를 클릭합니다.

4. **Run checkup** 을 클릭합니다.

5. 이름 필드에 검사 이름을 입력합니다.

6. 드롭다운 메뉴에서 **NetworkAttachmentDefinition** 을 선택합니다.

7. 선택 사항: 샘플 기간 (초) 필드에서 대기 시간 샘플 기간을 설정합니다.

8. 선택 사항: 원하는 최대 대기 시간(밀리초)을 활성화하고 시간 간격을 정의하여 최대 대기 시간 기간을 정의합니다.

9. 선택 사항: 노드 선택을 활성화하고 소스 노드 및 대상 노드를 지정하여 특정 노드를 대상으로 지정합니다.

10. 실행을 클릭합니다.

Latency 검사 탭의 **Checkups** 목록에서 대기 시간 점검 상태를 볼 수 있습니다. 자세한 내용은 점검 이름을 클릭합니다.

12.2.2.2. 웹 콘솔에서 스토리지 점검 실행

스토리지 점검을 실행하여 스토리지가 가상 머신에서 올바르게 작동하는지 확인합니다.

프로세스

1.

1. 웹 콘솔에서 가상화 → 확인 으로 이동합니다.

2. 스토리지 탭을 클릭합니다.

3. 권한 설치를 클릭합니다.

4. **Run checkup** 을 클릭합니다.

5. 이름 필드에 검사 이름을 입력합니다.

6. 시간 제한(분) 필드에 검사 시간 값을 입력합니다.

7. 실행을 클릭합니다.

스토리지 탭의 **Checkups** 목록에서 스토리지 점검 상태를 볼 수 있습니다. 자세한 내용은 점검 이름을 클릭합니다.

12.2.3. CLI에서 대기 시간 점검 실행

사전 정의된 점검을 사용하여 네트워크 연결을 확인하고 보조 네트워크 인터페이스에 연결된 두 가상 머신(VM) 간의 대기 시간을 측정합니다. 대기 시간 검사에서는 **ping** 유틸리티를 사용합니다.

다음 단계를 수행하여 대기 시간 점검을 실행합니다.

1. 서비스 계정, 역할 및 역할 바인딩을 생성하여 대기 시간 점검에 대한 클러스터 액세스 권한을 제공합니다.

2. 점검을 실행하고 결과를 저장할 입력을 제공하는 구성 맵을 생성합니다.

3. 점검을 실행할 작업을 생성합니다.

4. 구성 맵에서 결과를 검토합니다.

5. 선택 사항: 점검을 재실행하려면 기존 구성 맵 및 작업을 삭제한 다음 새 구성 맵 및 작업을 생성합니다.

6. 완료되면 대기 시간 점검 리소스를 삭제합니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.
- 클러스터에는 작업자 노드가 두 개 이상 있습니다.
- 네임스페이스에 대한 네트워크 연결 정의를 구성했습니다.

프로세스

1. 대기 시간 점검을 위한 ServiceAccount,Role, RoleBinding 매니페스트를 생성합니다.

예 12.1. 역할 매니페스트 파일의 예

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vm-latency-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kubevirt-vm-latency-checker
rules:
- apiGroups: ["kubevirt.io"]
  resources: ["virtualmachineinstances"]
  verbs: ["get", "create", "delete"]
- apiGroups: ["subresources.kubevirt.io"]
  resources: ["virtualmachineinstances/console"]
  verbs: ["get"]
- apiGroups: ["k8s.cni.cncf.io"]
  resources: ["network-attachment-definitions"]
  verbs: ["get"]
---
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kubevirt-vm-latency-checker
subjects:
- kind: ServiceAccount
  name: vm-latency-checkup-sa
roleRef:
  kind: Role
  name: kubevirt-vm-latency-checker
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kiagnose-configmap-access
rules:
- apiGroups: [ "" ]
  resources: [ "configmaps" ]
  verbs: ["get", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kiagnose-configmap-access
subjects:
- kind: ServiceAccount
  name: vm-latency-checkup-sa
roleRef:
  kind: Role
  name: kiagnose-configmap-access
  apiGroup: rbac.authorization.k8s.io
```

2. ServiceAccount,Role, RoleBinding 매니페스트를 적용합니다.

```
$ oc apply -n <target_namespace> -f <latency_sa_roles_rolebinding>.yaml 1
```

1 <target_namespace >는 점검을 실행할 네임스페이스입니다. NetworkAttachmentDefinition 오브젝트가 있는 기존 네임스페이스여야 합니다.

3. 점검에 대한 입력 매개변수가 포함된 ConfigMap 매니페스트를 생성합니다.

입력 구성 맵 예

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-latency-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-vm-latency
data:
  spec.timeout: 5m
  spec.param.networkAttachmentDefinitionNamespace: <target_namespace>
  spec.param.networkAttachmentDefinitionName: "blue-network" 1
  spec.param.maxDesiredLatencyMilliseconds: "10" 2
  spec.param.sampleDurationSeconds: "5" 3
  spec.param.sourceNode: "worker1" 4
  spec.param.targetNode: "worker2" 5
```

1 **NetworkAttachmentDefinition** 오브젝트의 이름입니다.

2 선택 사항: 가상 머신 간에 필요한 최대 대기 시간(밀리초)입니다. 측정된 대기 시간이 이 값을 초과하면 점검이 실패합니다.

3 선택 사항: 대기 시간 확인 기간(초)입니다.

4 선택 사항: 지정된 경우 대기 시간은 이 노드에서 대상 노드로 측정됩니다. 소스 노드가 지정된 경우 **spec.param.targetNode** 필드를 비워 둘 수 없습니다.

5 선택 사항: 지정된 경우 대기 시간은 소스 노드에서 이 노드로 측정됩니다.

4. 대상 네임스페이스에 구성 맵 매니페스트를 적용합니다.

```
$ oc apply -n <target_namespace> -f <latency_config_map>.yaml
```

-

5. 점검을 실행할 작업 매니페스트를 생성합니다.

작업 매니페스트 예

```
apiVersion: batch/v1
kind: Job
metadata:
  name: kubevirt-vm-latency-checkup
  labels:
    kiagnose/checkup-type: kubevirt-vm-latency
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccountName: vm-latency-checkup-sa
      restartPolicy: Never
      containers:
        - name: vm-latency-checkup
          image: registry.redhat.io/container-native-virtualization/vm-network-latency-
checkup-rhel9:v4.16.0
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
              drop: ["ALL"]
            runAsNonRoot: true
            seccompProfile:
              type: "RuntimeDefault"
          env:
            - name: CONFIGMAP_NAMESPACE
              value: <target_namespace>
            - name: CONFIGMAP_NAME
              value: kubevirt-vm-latency-checkup-config
            - name: POD_UID
              valueFrom:
                fieldRef:
                  fieldPath: metadata.uid
```

6. 작업 매니페스트를 적용합니다.

```
$ oc apply -n <target_namespace> -f <latency_job>.yaml
```

7. 작업이 완료될 때까지 기다립니다.

```
$ oc wait job kubevirt-vm-latency-checkup -n <target_namespace> --for
condition=complete --timeout 6m
```

8. 다음 명령을 실행하여 대기 시간 점검 결과를 검토합니다. 측정된 최대 대기 시간이 **spec.param.maxDesiredLatencyMilliseconds** 속성 값보다 크면 점검이 실패하고 오류를 반환합니다.

```
$ oc get configmap kubevirt-vm-latency-checkup-config -n <target_namespace> -o
yaml
```

출력 구성 맵 예(성공)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-latency-checkup-config
  namespace: <target_namespace>
  labels:
    kiagnose/checkup-type: kubevirt-vm-latency
data:
  spec.timeout: 5m
  spec.param.networkAttachmentDefinitionNamespace: <target_namespace>
  spec.param.networkAttachmentDefinitionName: "blue-network"
  spec.param.maxDesiredLatencyMilliseconds: "10"
  spec.param.sampleDurationSeconds: "5"
  spec.param.sourceNode: "worker1"
  spec.param.targetNode: "worker2"
  status.succeeded: "true"
  status.failureReason: ""
  status.completionTimestamp: "2022-01-01T09:00:00Z"
  status.startTimestamp: "2022-01-01T09:00:07Z"
  status.result.avgLatencyNanoSec: "177000"
  status.result.maxLatencyNanoSec: "244000" (1)
  status.result.measurementDurationSec: "5"
  status.result.minLatencyNanoSec: "135000"
  status.result.sourceNode: "worker1"
  status.result.targetNode: "worker2"
```

(1) 나노초 단위로 측정된 최대 대기 시간입니다.

9. 선택 사항: 점검 실패의 경우 자세한 작업 로그를 보려면 다음 명령을 사용합니다.

```
$ oc logs job.batch/kubevirt-vm-latency-checkup -n <target_namespace>
```

10. 다음 명령을 실행하여 이전에 생성한 작업 및 구성 맵을 삭제합니다.

```
$ oc delete job -n <target_namespace> kubevirt-vm-latency-checkup
```

```
$ oc delete config-map -n <target_namespace> kubevirt-vm-latency-checkup-config
```

11. 선택 사항: 다른 점검을 실행할 계획이 없는 경우 역할 매니페스트를 삭제합니다.

```
$ oc delete -f <latency_sa_roles_rolebinding>.yaml
```

12.2.4. DPDK 점검

사전 정의된 점검을 사용하여 OpenShift Container Platform 클러스터 노드에서 패킷 손실이 없는 DPDK(Data Plane Development Kit) 워크로드로 VM(가상 머신)을 실행할 수 있는지 확인합니다. DPDK 검사에서는 트래픽 생성기와 테스트 DPDK 애플리케이션을 실행하는 VM 간에 트래픽을 실행합니다.

다음 단계를 수행하여 DPDK 검사를 실행합니다.

1. DPDK 검사에 대한 서비스 계정, 역할 및 역할 바인딩을 생성합니다.

2. 점검을 실행하고 결과를 저장할 입력을 제공하는 구성 맵을 생성합니다.

3. 점검을 실행할 작업을 생성합니다.

4. 구성 맵에서 결과를 검토합니다.

5. 선택 사항: 점검을 재실행하려면 기존 구성 맵 및 작업을 삭제한 다음 새 구성 맵 및 작업을 생성합니다.

6. 완료되면 DPDK 점검 리소스를 삭제합니다.

사전 요구 사항

- OpenShift CLI(oc)가 설치되어 있습니다.
- 클러스터는 DPDK 애플리케이션을 실행하도록 구성되어 있습니다.
- 프로젝트는 DPDK 애플리케이션을 실행하도록 구성되어 있습니다.

프로세스

1. DPDK 검사에 대한 ServiceAccount,Role, RoleBinding 매니페스트를 생성합니다.

예 12.2. 서비스 계정, 역할 및 역할 바인딩 매니페스트 파일의 예

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dpdk-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kiagnose-configmap-access
rules:
  - apiGroups: [ "" ]
    resources: [ "configmaps" ]
    verbs: [ "get", "update" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kiagnose-configmap-access
subjects:
  - kind: ServiceAccount
    name: dpdk-checkup-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kiagnose-configmap-access
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kubevirt-dpdk-checker
```

```
  rules:
    - apiGroups: [ "kubevirt.io" ]
      resources: [ "virtualmachineinstances" ]
      verbs: [ "create", "get", "delete" ]
    - apiGroups: [ "subresources.kubevirt.io" ]
      resources: [ "virtualmachineinstances/console" ]
      verbs: [ "get" ]
    - apiGroups: [ "" ]
      resources: [ "configmaps" ]
      verbs: [ "create", "delete" ]
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: kubevirt-dpdk-checker
  subjects:
    - kind: ServiceAccount
      name: dpdk-checkup-sa
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: kubevirt-dpdk-checker
```

2. ServiceAccount,Role, RoleBinding 매니페스트를 적용합니다.

```
$ oc apply -n <target_namespace> -f <dpdk_sa_roles_rolebinding>.yaml
```

3. 점검에 대한 입력 매개변수가 포함된 ConfigMap 매니페스트를 생성합니다.

입력 구성 맵 예

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dpdk-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-dpdk
data:
  spec.timeout: 10m
  spec.param.networkAttachmentDefinitionName: <network_name> (1)
  spec.param.trafficGenContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-
checkup-traffic-gen:v0.4.0 (2)
  spec.param.vmUnderTestContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-
checkup-vm:v0.4.0" (3)
```

1. NetworkAttachmentDefinition 오브젝트의 이름입니다.

2. 트래픽 생성기의 컨테이너 디스크 이미지입니다. 이 예에서는 업스트림 프로젝트 Quay 컨테이너 레지스트리에서 이미지를 가져옵니다.

3. 테스트 중인 VM의 컨테이너 디스크 이미지입니다. 이 예에서는 업스트림 프로젝트 Quay 컨테이너 레지스트리에서 이미지를 가져옵니다.

4. 대상 네임스페이스에 ConfigMap 매니페스트를 적용합니다.

```
$ oc apply -n <target_namespace> -f <dpdk_config_map>.yaml
```

5. 점검을 실행할 작업 매니페스트를 생성합니다.

작업 매니페스트 예

```
apiVersion: batch/v1
kind: Job
metadata:
  name: dpdk-checkup
  labels:
    kiagnose/checkup-type: kubevirt-dpdk
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccountName: dpdk-checkup-sa
      restartPolicy: Never
      containers:
        - name: dpdk-checkup
          image: registry.redhat.io/container-native-virtualization/kubevirt-dpdk-checkup-
rhel9:v4.16.0
          imagePullPolicy: Always
          securityContext:
            allowPrivilegeEscalation: false
```

```
        capabilities:
          drop: ["ALL"]
        runAsNonRoot: true
        seccompProfile:
          type: "RuntimeDefault"
      env:
        - name: CONFIGMAP_NAMESPACE
          value: <target-namespace>
        - name: CONFIGMAP_NAME
          value: dpdk-checkup-config
        - name: POD_UID
          valueFrom:
            fieldRef:
              fieldPath: metadata.uid
```

6. 작업 매니페스트를 적용합니다.

```
$ oc apply -n <target_namespace> -f <dpdk_job>.yaml
```

7. 작업이 완료될 때까지 기다립니다.

```
$ oc wait job dpdk-checkup -n <target_namespace> --for condition=complete --
timeout 10m
```

8. 다음 명령을 실행하여 점검 결과를 검토합니다.

```
$ oc get configmap dpdk-checkup-config -n <target_namespace> -o yaml
```

출력 구성 맵 예(성공)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dpdk-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-dpdk
data:
  spec.timeout: 10m
  spec.param.NetworkAttachmentDefinitionName: "dpdk-network-1"
  spec.param.trafficGenContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-
checkup-traffic-gen:v0.4.0"
  spec.param.vmUnderTestContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-
```

```
checkup-vm:v0.4.0"
  status.succeeded: "true" 1
  status.failureReason: "" 2
  status.startTimestamp: "2023-07-31T13:14:38Z" 3
  status.completionTimestamp: "2023-07-31T13:19:41Z" 4
  status.result.trafficGenSentPackets: "480000000" 5
  status.result.trafficGenOutputErrorPackets: "0" 6
  status.result.trafficGenInputErrorPackets: "0" 7
  status.result.trafficGenActualNodeName: worker-dpdk1 8
  status.result.vmUnderTestActualNodeName: worker-dpdk2 9
  status.result.vmUnderTestReceivedPackets: "480000000" 10
  status.result.vmUnderTestRxDroppedPackets: "0" 11
  status.result.vmUnderTestTxDroppedPackets: "0" 12
```

1

검사 성공(**true**)인지(**false**)인지 여부를 지정합니다.

2

검사에 실패하는 경우 실패 이유

3

검사 시작 시간(**RFC 3339** 시간 형식)입니다.

4

검사 완료 시간(**RFC 3339** 시간 형식)입니다.

5

트래픽 생성기에서 전송된 패킷 수입니다.

6

트래픽 생성기에서 전송된 오류 패킷 수입니다.

7

트래픽 생성기가 수신한 오류 패킷 수입니다.

8

트래픽 생성기 VM이 예약된 노드입니다.

9

테스트 중인 VM이 예약된 노드입니다.

10

테스트 중인 VM에서 수신한 패킷 수입니다.

11

DPDK 애플리케이션에서 삭제한 수신 트래픽 패킷입니다.

12

DPDK 애플리케이션에서 삭제된 송신 트래픽 패킷입니다.

9. 다음 명령을 실행하여 이전에 생성한 작업 및 구성 맵을 삭제합니다.

```
$ oc delete job -n <target_namespace> dpdk-checkup
```

```
$ oc delete config-map -n <target_namespace> dpdk-checkup-config
```

10. 선택 사항: 다른 점검을 실행하지 않으려면 ServiceAccount,Role, RoleBinding 매니페스트를 삭제합니다.

```
$ oc delete -f <dpdk_sa_roles_rolebinding>.yaml
```

12.2.4.1. DPDK 점검 구성 맵 매개변수

다음 표는 클러스터 DPDK 준비 상태 점검을 실행할 때 입력 ConfigMap 매니페스트의 data 스탠자에 설정할 수 있는 필수 및 선택적 매개변수를 보여줍니다.

표 12.1. DPDK 확인 구성 맵 입력 매개변수

매개변수	설명	필수 항목입니다.
spec.timeout	체크아웃에 실패하기 전의 시간(분)입니다.	True

매개변수	설명	필수 항목입니다.
spec.param.networkAttachmentDefinitionName	연결된 SR-IOV NIC의 **NetworkAttachmentDefinition** 오브젝트 이름입니다.	True
spec.param.trafficGenContainerDiskImage	트래픽 생성기의 컨테이너 디스크 이미지입니다.	True
spec.param.trafficGenTargetNodeName	트래픽 생성기 VM을 예약할 노드입니다. 노드는 DPDK 트래픽을 허용하도록 구성해야 합니다.	False
spec.param.trafficGenPacketsPerSecond	초당 패킷 수(k) 또는 million(m)입니다. 기본값은 8m입니다.	False
spec.param.vmUnderTestContainerDiskImage	테스트 중인 VM의 컨테이너 디스크 이미지입니다.	True
spec.param.vmUnderTestTargetNodeName	테스트 중인 VM을 예약해야 하는 노드입니다. 노드는 DPDK 트래픽을 허용하도록 구성해야 합니다.	False
spec.param.testDuration	트래픽 생성기가 실행되는 기간(분)입니다. 기본값은 5분입니다.	False
spec.param.portBandwidthGbps	SR-IOV NIC의 최대 대역폭입니다. 기본값은 10Gbps입니다.	False
spec.param.verbose	**true** 로 설정하면 검사 로그의 상세 정보 표시가 증가합니다. 기본값은 **false**입니다.	False

12.2.4.2. RHEL 가상 머신용 컨테이너 디스크 이미지 빌드

qcow2 형식으로 사용자 지정 **RHEL(Red Hat Enterprise Linux) 8 OS** 이미지를 빌드하고 이를 사용하여 컨테이너 디스크 이미지를 생성할 수 있습니다. 클러스터에서 액세스할 수 있는 레지스트리에 컨테이너 디스크 이미지를 저장하고 **DPDK** 검사 구성 맵의 **spec.param.vmContainerDiskImage** 속성에 이미지 위치를 지정할 수 있습니다.

컨테이너 디스크 이미지를 빌드하려면 이미지 빌더 **VM**(가상 머신)을 생성해야 합니다. *이미지 빌더 VM* 은 사용자 지정 **RHEL** 이미지를 빌드하는 데 사용할 수 있는 **RHEL 8 VM**입니다.

사전 요구 사항

- 이미지 빌더 **VM**은 **RHEL 8.7**을 실행해야 하며 **/var** 디렉터리에 최소 **2**개의 **CPU** 코어, **4GiB**

RAM 및 20GB의 여유 공간이 있어야 합니다.

- 이미지 빌더 툴과 해당 CLI(composer-cli)를 VM에 설치했습니다.
- virt-customize 툴을 설치했습니다.

```
# dnf install libguestfs-tools
```

- Podman CLI 툴(podman)을 설치했습니다.

프로세스

1. RHEL 8.7 이미지를 빌드할 수 있는지 확인합니다.

```
# composer-cli distros list
```

참고

composer-cli 명령을 root가 아닌 것으로 실행하려면 사용자를 weldr 또는 root 그룹에 추가합니다.

```
# usermod -a -G weldr user
```

```
$ newgrp weldr
```

2. 다음 명령을 입력하여 설치할 패키지, 커널 사용자 정의 및 부팅 시 비활성화할 서비스가 포함된 TOML 형식으로 이미지 블루프린트 파일을 생성합니다.

```
$ cat << EOF > dpdk-vm.toml
name = "dpdk_image"
description = "Image to use with the DPDK checkup"
version = "0.0.1"
distro = "rhel-87"

[[customizations.user]]
name = "root"
password = "redhat"

[[packages]]
name = "dpdk"
```

```
[[packages]]
name = "dpdk-tools"

[[packages]]
name = "driverctl"

[[packages]]
name = "tuned-profiles-cpu-partitioning"

[customizations.kernel]
append = "default_hugepagesz=1GB hugepagesz=1G hugepages=1"

[customizations.services]
disabled = ["NetworkManager-wait-online", "sshd"]
EOF
```

3. 다음 명령을 실행하여 블루프린트 파일을 이미지 빌더 툴로 푸시합니다.

```
# composer-cli blueprints push dpdk-vm.toml
```

4. 블루프린트 이름과 출력 파일 형식을 지정하여 시스템 이미지를 생성합니다. 구성 프로세스를 시작할 때 이미지의 **UUID(Universally Unique Identifier)**가 표시됩니다.

```
# composer-cli compose start dpdk_image qcow2
```

5. 작성 프로세스가 완료될 때까지 기다립니다. 다음 단계를 계속 진행하려면 작성 상태가 **FINISHED** 로 표시되어야 합니다.

```
# composer-cli compose status
```

6. 다음 명령을 입력하여 **UUID**를 지정하여 **qcow2** 이미지 파일을 다운로드합니다.

```
# composer-cli compose image <UUID>
```

7. 다음 명령을 실행하여 사용자 지정 스크립트를 생성합니다.

```
$ cat <<EOF >customize-vm
#!/bin/bash

# Setup hugepages mount
mkdir -p /mnt/huge
echo "hugetlbfs /mnt/huge hugetlbfs defaults,pagesize=1GB 0 0" >> /etc/fstab
```

```
# Create vfio-noiommu.conf
echo "options vfio enable_unsafe_noiommu_mode=1" > /etc/modprobe.d/vfio-
noiommu.conf

# Enable guest-exec,guest-exec-status on the qemu-guest-agent configuration
sed -i '/^BLACKLIST_RPC=/ { s/guest-exec-status//; s/guest-exec//g }'
/etc/sysconfig/qemu-ga
sed -i '/^BLACKLIST_RPC=/ { s/,\+/,/g; s/^,\|,$//g }' /etc/sysconfig/qemu-ga
EOF
```

8. **virt-customize** 툴을 사용하여 이미지 빌더 툴에서 생성한 이미지를 사용자 지정합니다.

```
$ virt-customize -a <UUID>-disk.qcow2 --run=customize-vm --selinux-relabel
```

9. 컨테이너 디스크 이미지를 빌드하는 모든 명령이 포함된 **Dockerfile**을 생성하려면 다음 명령을 입력합니다.

```
$ cat << EOF > Dockerfile
FROM scratch
COPY --chown=107:107 <UUID>-disk.qcow2 /disk/
EOF
```

다음과 같습니다.

<UUID>-disk.qcow2

qcow2 형식으로 사용자 지정 이미지의 이름을 지정합니다.

10. 다음 명령을 실행하여 컨테이너를 빌드하고 태그를 지정합니다.

```
$ podman build . -t dpdk-rhel:latest
```

11. 다음 명령을 실행하여 컨테이너 디스크 이미지를 클러스터에서 액세스할 수 있는 레지스트리로 푸시합니다.

```
$ podman push dpdk-rhel:latest
```

12. **DPDK** 검사 구성 맵의 **spec.param.vmUnderTestContainerDiskImage** 속성의 컨테이너 디스크 이미지에 대한 링크를 제공합니다.

12.2.5. 스토리지 점검 실행

사전 정의된 점검을 사용하여 **OpenShift Container Platform** 클러스터 스토리지가 **OpenShift Virtualization** 워크로드를 실행하도록 최적으로 구성되었는지 확인합니다.

사전 요구 사항

- **OpenShift CLI(oc)**가 설치되어 있습니다.
- 클러스터 관리자가 다음 예와 같이 스토리지 점검 서비스 계정 및 네임스페이스에 필요한 **cluster-reader** 권한을 생성했습니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kubevirt-storage-checkup-clustereader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-reader
subjects:
- kind: ServiceAccount
  name: storage-checkup-sa
  namespace: <target_namespace> 1
```

1 점검을 실행할 네임스페이스입니다.

프로세스

1. 스토리지 점검에 대한 **ServiceAccount,Role, RoleBinding** 매니페스트 파일을 생성합니다.

예 12.3. 서비스 계정, 역할 및 역할 바인딩 매니페스트의 예

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: storage-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: storage-checkup-role
rules:
  - apiGroups: [ "" ]
```

```
    resources: [ "configmaps" ]
    verbs: ["get", "update"]
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachines" ]
    verbs: [ "create", "delete" ]
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachineinstances" ]
    verbs: [ "get" ]
  - apiGroups: [ "subresources.kubevirt.io" ]
    resources: [ "virtualmachineinstances/addvolume",
"virtualmachineinstances/removevolume" ]
    verbs: [ "update" ]
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachineinstancemigrations" ]
    verbs: [ "create" ]
  - apiGroups: [ "cdi.kubevirt.io" ]
    resources: [ "datavolumes" ]
    verbs: [ "create", "delete" ]
  - apiGroups: [ "" ]
    resources: [ "persistentvolumeclaims" ]
    verbs: [ "delete" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: storage-checkup-role
subjects:
  - kind: ServiceAccount
    name: storage-checkup-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: storage-checkup-role
```

2. 대상 네임스페이스에 ServiceAccount,Role, RoleBinding 매니페스트를 적용합니다.

```
$ oc apply -n <target_namespace> -f <storage_sa_roles_rolebinding>.yaml
```

3. ConfigMap 및 작업 매니페스트 파일을 생성합니다. 구성 맵에는 검사 작업에 대한 입력 매개 변수가 포함되어 있습니다.

입력 구성 맵 및 작업 매니페스트 예

```
---
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: storage-checkup-config
  namespace: $CHECKUP_NAMESPACE
data:
  spec.timeout: 10m
---
apiVersion: batch/v1
kind: Job
metadata:
  name: storage-checkup
  namespace: $CHECKUP_NAMESPACE
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccount: storage-checkup-sa
      restartPolicy: Never
      containers:
        - name: storage-checkup
          image: quay.io/kiagnose/kubevirt-storage-checkup:main
          imagePullPolicy: Always
          env:
            - name: CONFIGMAP_NAMESPACE
              value: $CHECKUP_NAMESPACE
            - name: CONFIGMAP_NAME
              value: storage-checkup-config
```

4. 대상 네임스페이스에 **ConfigMap** 및 작업 매니페스트 파일을 적용하여 점검을 실행합니다.

```
$ oc apply -n <target_namespace> -f <storage_configmap_job>.yaml
```

5. 작업이 완료될 때까지 기다립니다.

```
$ oc wait job storage-checkup -n <target_namespace> --for condition=complete --
timeout 10m
```

6. 다음 명령을 실행하여 점검 결과를 검토합니다.

```
$ oc get configmap storage-checkup-config -n <target_namespace> -o yaml
```

출력 구성 맵 예(성공)

```
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: storage-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-storage
data:
  spec.timeout: 10m
  status.succeeded: "true" (1)
  status.failureReason: "" (2)
  status.startTimestamp: "2023-07-31T13:14:38Z" (3)
  status.completionTimestamp: "2023-07-31T13:19:41Z" (4)
  status.result.cnvVersion: 4.16.2
  status.result.defaultStorageClass: trident-nfs (5)
  status.result.goldenImagesNoDataSource: <data_import_cron_list> (6)
  status.result.goldenImagesNotUpToDate: <data_import_cron_list> (7)
  status.result.ocpVersion: 4.16.0
  status.result.storageMissingVolumeSnapshotClass: <storage_class_list>
  status.result.storageProfilesWithEmptyClaimPropertySets: <storage_profile_list> (8)
  status.result.storageProfilesWithSpecClaimPropertySets: <storage_profile_list>
  status.result.storageWithRWX: |-
    ocs-storagecluster-ceph-rbd
    ocs-storagecluster-ceph-rbd-virtualization
    ocs-storagecluster-cephfs
    trident-iscsi
    trident-minio
    trident-nfs
    windows-vms
  status.result.vmBootFromGoldenImage: VMI "vmi-under-test-dhkb8" successfully
booted
  status.result.vmHotplugVolume: |-
    VMI "vmi-under-test-dhkb8" hotplug volume ready
    VMI "vmi-under-test-dhkb8" hotplug volume removed
  status.result.vmLiveMigration: VMI "vmi-under-test-dhkb8" migration completed
  status.result.vmVolumeClone: 'DV cloneType: "csi-clone"'
  status.result.vmsWithNonVirtRbdStorageClass: <vm_list> (9)
  status.result.vmsWithUnsetEfsStorageClass: <vm_list> (10)
```

(1)

검사 성공(**true**)인지(**false**)인지 여부를 지정합니다.

(2)

검사에 실패하는 경우 실패 이유

(3)

검사 시작 시간(**RFC 3339** 시간 형식)입니다.

4

검사 완료 시간(**RFC 3339** 시간 형식)입니다.

5

기본 스토리지 클래스가 있는지 여부를 지정합니다.

6

데이터 소스가 준비되지 않은 골든 이미지 목록입니다.

7

데이터 가져오기 **cron**이 최신 상태가 아닌 골든 이미지 목록입니다.

8

알 수 없는 프로비저너가 있는 스토리지 프로필 목록입니다.

9

가상화 스토리지 클래스가 있는 경우 **Ceph RBD** 스토리지 클래스를 사용하는 가상 머신 목록입니다.

10

스토리지 클래스에 **GID** 및 **UID**가 설정되지 않은 **EFS(Elastic File Store)** 스토리지 클래스를 사용하는 가상 머신 목록입니다.

7. 다음 명령을 실행하여 이전에 생성한 작업 및 구성 맵을 삭제합니다.

```
$ oc delete job -n <target_namespace> storage-checkup
```

```
$ oc delete config-map -n <target_namespace> storage-checkup-config
```

8. 선택 사항: 다른 점검을 실행하지 않으려면 **ServiceAccount,Role, RoleBinding** 매니페스트를 삭제합니다.

```
$ oc delete -f <storage_sa_roles_rolebinding>.yaml
```

12.2.6. 추가 리소스

- 여러 네트워크에 가상 머신 연결
- Intel NIC와 함께 DPDK 모드에서 가상 기능 사용
- SR-IOV 및 Node Tuning Operator를 사용하여 DPDK 라인 속도 달성
- 이미지 빌더 설치
- Red Hat Subscription Manager를 사용하여 Red Hat 고객 포털에 RHEL 시스템을 등록하고 가입하는 방법

12.3. 가상 리소스에 대한 PROMETHEUS 쿼리

OpenShift Virtualization에서는 vCPU, 네트워크, 스토리지 및 게스트 메모리 스왑을 포함하여 클러스터 인프라 리소스의 사용을 모니터링하는 데 사용할 수 있는 메트릭을 제공합니다. 메트릭을 사용하여 실시간 마이그레이션 상태를 쿼리할 수도 있습니다.

12.3.1. 사전 요구 사항

- vCPU 지표를 사용하려면 MachineConfig 오브젝트에 schedstats=enable 커널 인수를 적용해야 합니다. 이 커널 인수를 사용하면 디버깅 및 성능 튜닝에 사용되는 스케줄러 통계가 활성화되고 스케줄러에 약간의 부하가 추가됩니다. 자세한 내용은 노드에 커널 인수 추가를 참조하십시오.
- 게스트 메모리 스와핑 쿼리가 데이터를 반환하려면 가상 게스트에서 메모리 스와핑을 활성화해야 합니다.

12.3.2. 메트릭 쿼리

OpenShift Container Platform 모니터링 대시보드를 사용하면 Pacemaker에서 표시되는 메트릭을 검사하기 위해 Prometheus Query Language(PromQL) 쿼리를 실행할 수 있습니다. 이 기능을 사용하면 클러스터 상태 및 모니터링 중인 모든 사용자 정의 워크로드에 대한 정보가 제공됩니다.

클러스터 관리자는 모든 핵심 OpenShift Container Platform 및 사용자 정의 프로젝트에 대한 메트릭을 쿼리할 수 있습니다.

개발자는 메트릭을 쿼리할 때 프로젝트 이름을 지정해야 합니다. 선택한 프로젝트의 메트릭을 확인하는 데 필요한 권한이 있어야 합니다.

12.3.2.1. 클러스터 관리자로서 모든 프로젝트의 메트릭 쿼리

클러스터 관리자 또는 모든 프로젝트에 대한 보기 권한이 있는 사용자로 Metrics UI에서 모든 기본 OpenShift Container Platform 및 사용자 정의 프로젝트에 대한 메트릭에 액세스할 수 있습니다.

사전 요구 사항

- cluster-admin 클러스터 역할 또는 모든 프로젝트에 대한 보기 권한이 있는 사용자로 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(oc)가 설치되어 있습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 관리자 화면에서 모니터링 → 메트릭 을 선택합니다.
2. 하나 이상의 쿼리를 추가하려면 다음 중 하나를 수행합니다.

옵션	설명
사용자 지정 쿼리를 생성합니다.	**표현식** 필드에 PromQL(Prometheus Query Language) 쿼리를 추가합니다. PromQL 표현식을 입력하면 자동 완성 제안이 드롭다운 목록에 표시됩니다. 이러한 제안에는 함수, 메트릭, 라벨 및 시간 토큰이 포함됩니다. 키보드 화살표를 사용하여 이러한 제안된 항목 중 하나를 선택한 다음 Enter를 눌러 표현식에 항목을 추가할 수 있습니다. 또한 제안된 항목 위로 마우스 포인터를 이동하여 해당 항목에 대한 간략한 설명을 볼 수도 있습니다.
여러 쿼리를 추가합니다.	**쿼리 추가**를 선택합니다.
기존 쿼리를 복제합니다.	쿼리 옆에 있는 옵션 메뉴 를 선택한 다음 **중복 쿼리** 를 선택합니다.

옵션	설명
쿼리가 실행되지 않도록 비활성화합니다.	쿼리 옆에 있는 옵션 메뉴  를 선택하고 **쿼리 비활성화** 를 선택합니다.

3. **생성한 쿼리를 실행하려면 쿼리 실행을 선택합니다. 쿼리의 메트릭은 플롯에 시각화됩니다. 쿼리가 유효하지 않으면 UI에 오류 메시지가 표시됩니다.**

참고

대량의 데이터에서 작동하는 쿼리 시간이 초과되거나 시계열 그래프에 있을 때 브라우저가 과부하될 수 있습니다. 이를 방지하려면 그래프 숨기기를 선택하고 메트릭 테이블만 사용하여 쿼리를 조정합니다. 그런 다음 실행 가능한 쿼리를 검색한 후 플롯을 활성화하여 그래프를 그립니다.

참고

기본적으로 쿼리 테이블은 모든 메트릭과 해당 현재 값을 나열하는 확장된 보기를 표시합니다. 쿼리에 대해 확장된 보기를 최소화하려면 ˅를 선택할 수 있습니다.

4. **선택 사항: 이제 페이지 URL에 실행한 쿼리가 포함되어 있습니다. 나중에 이 쿼리 세트를 다시 사용하려면 이 URL을 저장합니다.**

5. **시각화된 메트릭을 살펴봅니다. 처음에 활성화된 모든 쿼리의 모든 메트릭이 플롯에 표시됩니다. 다음 중 하나를 수행하여 표시되는 메트릭을 선택할 수 있습니다.**

옵션	설명
쿼리에서 모든 메트릭을 숨깁니다.	쿼리의 옵션 메뉴  를 클릭하고 **모든 시리즈 숨기기** 를 클릭합니다.
특정 메트릭을 숨깁니다.	쿼리 테이블로 이동하여 메트릭 이름 옆에 있는 색상이 지정된 사각형을 클릭합니다.

옵션	설명
플롯으로 확대하고 시간 범위를 변경합니다.	either: ● 수평으로 플롯을 클릭하고 드래그하여 시간 범위를 시각적으로 선택합니다. ● 왼쪽 상단 코너의 메뉴를 사용하여 시간 범위를 선택합니다.
시간 범위를 재설정합니다.	**확대/축소 재설정** 을 선택합니다.
특정 시점에서 모든 쿼리에 대한 출력을 표시합니다.	해당 시점에서 플롯에 마우스 커서를 유지합니다. 쿼리 출력이 팝업 상자에 나타납니다.
플롯을 숨깁니다.	**그래프 숨기기를** 선택합니다.

12.3.2.2. 개발자로 사용자 정의 프로젝트의 메트릭 쿼리

사용자 정의 프로젝트의 메트릭에 대해 개발자 또는 프로젝트에 대한 보기 권한이 있는 사용자로 액세스할 수 있습니다.

개발자 관점에서 **Metrics UI**에는 선택한 프로젝트에 대한 사전 정의된 **CPU, 메모리, 대역폭 및 네트워크 패킷 쿼리가 포함되어 있습니다. 프로젝트에 대한 CPU, 메모리, 대역폭, 네트워크 패킷 및 애플리케이션 메트릭에 대해 사용자 정의 Prometheus Query Language(PromQL) 쿼리를 실행할 수도 있습니다.**

참고

개발자는 관리자 관점이 아닌 개발자 관점만 사용할 수 있습니다. 개발자는 한 번에 하나의 프로젝트의 메트릭만 쿼리할 수 있습니다.

사전 요구 사항

- 개발자로 또는 메트릭을 확인하는 프로젝트에 대한 보기 권한이 있는 사용자로 클러스터에 액세스할 수 있습니다.
- 사용자 정의 프로젝트에 대한 모니터링을 활성화했습니다.

- 사용자 정의 프로젝트에 서비스를 배포했습니다.

- 서비스에서 모니터링 방법을 정의하는 데 사용할 **ServiceMonitor CRD**(사용자 정의 리소스 정의(**Custom Resource Definition**))가 생성되었습니다.

프로세스

1. **OpenShift Container Platform** 웹 콘솔의 개발자 화면에서 **모니터링** → **메트릭** 을 선택합니다.

2. **Project:** 목록에서 메트릭을 보려는 프로젝트를 선택합니다.

3. 쿼리 선택 목록에서 쿼리를 선택하거나 **PromQL** 표시를 선택하여 선택한 쿼리를 기반으로 사용자 정의 **PromQL** 쿼리를 만듭니다. 쿼리의 메트릭은 플롯에 시각화됩니다.

참고

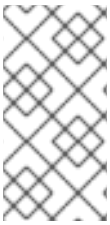
개발자 화면에서는 한 번에 하나의 쿼리만 실행할 수 있습니다.

4. 다음 중 하나를 수행하여 시각화된 메트릭을 살펴봅니다.

옵션	설명
플롯으로 확대하고 시간 범위를 변경합니다.	either: ● 수평으로 플롯을 클릭하고 드래그하여 시간 범위를 시각적으로 선택합니다. ● 왼쪽 상단 코너의 메뉴를 사용하여 시간 범위를 선택합니다.
시간 범위를 재설정합니다.	**확대/축소 재설정** 을 선택합니다.
특정 시점에서 모든 쿼리에 대한 출력을 표시합니다.	해당 시점에서 플롯에 마우스 커서를 유지합니다. 쿼리 출력이 팝업 상자에 나타납니다.

12.3.3. 가상화 메트릭

다음 메트릭 설명에는 예제 **Prometheus Query Language(PromQL)** 쿼리가 포함됩니다. 이러한 메트릭은 **API**가 아니며 버전 간에 변경될 수 있습니다.



참고

다음 예제에서는 기간을 지정하는 **topk** 쿼리를 사용합니다. 해당 기간 동안 가상 머신을 삭제해도 쿼리 출력에 계속 표시될 수 있습니다.

12.3.3.1. vCPU 메트릭

다음 쿼리는 **I/O(입력/출력)** 대기 중인 가상 머신을 식별할 수 있습니다.

kubevirt_vmi_vcpu_wait_seconds_total

가상 시스템의 **vCPU**에 대해 대기 시간(초)을 반환합니다. 유형: **Cryostat.**

'0' 이상의 값은 **vCPU**가 실행하려고 하지만 호스트 스케줄러는 아직 실행할 수 없음을 의미합니다. 이러한 실행 불가능은 **I/O**에 문제가 있음을 나타냅니다.

참고

vCPU 지표를 쿼리하려면 먼저 **MachineConfig** 오브젝트에 **schedstats=enable** 커널 인수를 적용해야 합니다. 이 커널 인수를 사용하면 디버깅 및 성능 튜닝에 사용되는 스케줄러 통계가 활성화되고 스케줄러에 약간의 부하가 추가됩니다.

vCPU 대기 시간 쿼리의 예

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_vcpu_wait_seconds_total[6m]))) > 0 1
```

1

이 쿼리는 **6**분 동안 매 순간에 **I/O**를 기다리는 상위 **3**개의 **VM**을 반환합니다.

12.3.3.2. 네트워크 메트릭

다음 쿼리는 네트워크를 포화 상태로 만드는 가상 머신을 식별할 수 있습니다.

kubevirt_vmi_network_receive_bytes_total

가상 시스템의 네트워크에서 수신된 총 트래픽(바이트 단위)을 반환합니다. 유형: Cryostat.

kubevirt_vmi_network_transmit_bytes_total

가상 시스템의 네트워크에서 전송된 총 트래픽(바이트 단위)을 반환합니다. 유형: Cryostat.

네트워크 트래픽 쿼리의 예

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_network_receive_bytes_total[6m])) +
sum by (name, namespace) (rate(kubevirt_vmi_network_transmit_bytes_total[6m]))) > 0 1
```

1

이 쿼리는 6분 동안 매번 가장 많은 네트워크 트래픽을 전송하는 상위 3 개의 VM을 반환합니다.

12.3.3.3. 스토리지 메트릭

12.3.3.3.1. 스토리지 관련 트래픽

다음 쿼리는 대량의 데이터를 작성하는 VM을 식별할 수 있습니다.

kubevirt_vmi_storage_read_traffic_bytes_total

가상 머신의 스토리지 관련 트래픽의 총 양(바이트)을 반환합니다. 유형: Cryostat.

kubevirt_vmi_storage_write_traffic_bytes_total

가상 시스템의 스토리지 관련 트래픽의 총 스토리지 쓰기(바이트)를 반환합니다. 유형: Cryostat.

스토리지 관련 트래픽 쿼리의 예

-

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_storage_read_traffic_bytes_total[6m]))
+ sum by (name, namespace) (rate(kubevirt_vmi_storage_write_traffic_bytes_total[6m]))) > 0
1
```

1

이 쿼리는 6분 동안 매번 가장 많은 스토리지 트래픽을 수행하는 상위 3 개의 VM을 반환합니다.

12.3.3.3.2. 스토리지 스냅샷 데이터

kubevirt_vmsnapshot_disks_restored_from_source

소스 가상 머신에서 복원한 총 가상 머신 디스크 수를 반환합니다. 유형: 게이지.

kubevirt_vmsnapshot_disks_restored_from_source_bytes

소스 가상 머신에서 복원된 공간(바이트)을 반환합니다. 유형: 게이지.

스토리지 스냅샷 데이터 쿼리의 예

```
kubevirt_vmsnapshot_disks_restored_from_source{vm_name="simple-vm",
vm_namespace="default"} 1
```

1

이 쿼리는 소스 가상 머신에서 복원한 총 가상 머신 디스크 수를 반환합니다.

```
kubevirt_vmsnapshot_disks_restored_from_source_bytes{vm_name="simple-vm",
vm_namespace="default"} 1
```

1

이 쿼리는 소스 가상 머신에서 복원된 공간(바이트)을 반환합니다.

12.3.3.3.3. I/O 성능

다음 쿼리는 스토리지 장치의 I/O 성능을 확인할 수 있습니다.

kubevirt_vmi_storage_iops_read_total

가상 시스템이 초당 수행하는 쓰기 I/O 작업 양을 반환합니다. 유형: Cryostat.

kubevirt_vmi_storage_iops_write_total

가상 시스템이 초당 수행하는 읽기 I/O 작업의 양을 반환합니다. 유형: Cryostat.

I/O 성능 쿼리의 예

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_storage_iops_read_total[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_storage_iops_write_total[6m]))) > 0 1
```

1

이 쿼리는 6분 동안 매 순간 초당 최대 I/O 작업을 수행하는 상위 3 개의 VM을 반환합니다.

12.3.3.4. 게스트 메모리 스왑 메트릭

다음 쿼리는 메모리 스와핑을 가장 많이 수행하는 스왑 사용 게스트를 식별할 수 있습니다.

kubevirt_vmi_memory_swap_in_traffic_bytes

가상 게스트가 스와핑하는 메모리의 총 양(바이트 단위)을 반환합니다. 유형: 게이지.

kubevirt_vmi_memory_swap_out_traffic_bytes

가상 게스트가 스왑 아웃하는 메모리의 총 양(바이트 단위)을 반환합니다. 유형: 게이지.

메모리 스와핑 쿼리의 예

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_memory_swap_in_traffic_bytes[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_memory_swap_out_traffic_bytes[6m]))) > 0 1
```

1

이 쿼리는 6분 동안 매 순간 게스트가 가장 많은 메모리 스왑을 수행하는 상위 3개의 VM을 반환합니다.

참고

메모리 스와핑은 가상 시스템이 메모리 부족 상태에 있음을 나타냅니다. 가상 시스템의 메모리 할당을 늘리면 이 문제가 완화될 수 있습니다.

12.3.3.5. 실시간 마이그레이션 메트릭

다음 메트릭을 쿼리하여 실시간 마이그레이션 상태를 표시할 수 있습니다.

kubevirt_vmi_migration_data_processed_bytes

새 VM(가상 머신)으로 마이그레이션한 게스트 운영 체제 데이터의 양입니다. 유형: 게이지.

kubevirt_vmi_migration_data_remaining_bytes

마이그레이션할 때까지 남아 있는 게스트 운영 체제 데이터의 양입니다. 유형: 게이지.

kubevirt_vmi_migration_memory_transfer_rate_bytes

게스트 운영 체제에서 메모리가 더러워지는 비율입니다. 더티 메모리는 변경되었지만 아직 디스크에 기록되지 않은 데이터입니다. 유형: 게이지.

kubevirt_vmi_migrations_in_pending_phase

보류 중인 마이그레이션 수입니다. 유형: 게이지.

kubevirt_vmi_migrations_in_scheduling_phase

스케줄링 마이그레이션 수입니다. 유형: 게이지.

kubevirt_vmi_migrations_in_running_phase

실행 중인 마이그레이션 수입니다. 유형: 게이지.

kubevirt_vmi_migration_succeeded

성공적으로 완료된 마이그레이션 수입니다. 유형: 게이지.

kubevirt_vmi_migration_failed

실패한 마이그레이션 수입니다. 유형: 게이지.

12.3.4. 추가 리소스

- 모니터링 개요
- Prometheus 쿼리
- Prometheus 쿼리 예

12.4. 가상 머신의 사용자 정의 메트릭 노출

OpenShift Container Platform에는 핵심 플랫폼 구성 요소에 대한 모니터링을 제공하는 사전 구성된 사전 설치된 자체 업데이트 모니터링 스택이 포함되어 있습니다. 이 모니터링 스택은 **Prometheus** 모니터링 시스템을 기반으로 합니다. **Prometheus**는 시계열 데이터베이스이며 메트릭에 대한 규칙 평가 엔진입니다.

OpenShift Container Platform 모니터링 스택을 사용하는 것 외에도 **CLI**를 사용하여 사용자 정의 프로젝트에 대한 모니터링을 활성화하고 **node-exporter** 서비스를 통해 가상 머신에 노출되는 사용자 정의 지표를 쿼리할 수 있습니다.

12.4.1. 노드 내보내기 서비스 구성

node-exporter 에이전트는 메트릭을 수집하려는 클러스터의 모든 가상 머신에 배포됩니다. 가상 머신과 연결된 내부 지표 및 프로세스를 노출하도록 **node-exporter** 에이전트를 서비스로 구성합니다.

사전 요구 사항

- **OpenShift Container Platform CLI oc**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

- **openshift-monitoring** 프로젝트에서 **cluster-monitoring-config ConfigMap** 오브젝트를 생성합니다.

- **enableUserWorkload** 를 **true** 로 설정하여 **openshift-user-workload-monitoring** 프로젝트에서 **user-workload-monitoring-config ConfigMap** 오브젝트를 구성합니다.

프로세스

1. **Service YAML** 파일을 생성합니다. 다음 예에서 파일을 **node-exporter-service.yaml** 이라고 합니다.

```
kind: Service
apiVersion: v1
metadata:
  name: node-exporter-service 1
  namespace: dynamation 2
  labels:
    servicetype: metrics 3
spec:
  ports:
    - name: exmet 4
      protocol: TCP
      port: 9100 5
      targetPort: 9100 6
  type: ClusterIP
  selector:
    monitor: metrics 7
```

1. 가상 시스템에서 지표를 표시하는 **node-exporter** 서비스입니다.

2. 서비스가 생성되는 네임스페이스입니다.

3. 서비스의 레이블입니다. **ServiceMonitor** 는 이 서비스와 일치하도록 이 레이블을 사용합니다.

4. **ClusterIP** 서비스의 포트 **9100**에 지표를 표시하는 포트에 지정된 이름입니다.

5 **node-exporter-service** 에서 요청을 수신 대기하는 데 사용하는 대상 포트입니다.

6 **monitor** 레이블로 구성된 가상 머신의 **TCP** 포트 번호입니다.

7 가상 머신의 **Pod**와 일치하는 데 사용되는 레이블입니다. 이 예에서는 레이블 **monitor** 가 있는 모든 가상 머신의 **Pod**가 있고 지표 값이 일치합니다.

2. **node-exporter** 서비스를 생성합니다.

```
$ oc create -f node-exporter-service.yaml
```

12.4.2. 노드 내보내기 서비스를 사용하여 가상 머신 구성

의 **node-exporter** 파일을 가상 머신에 다운로드합니다. 그런 다음 가상 머신이 부팅될 때 **node-exporter** 서비스를 실행하는 **systemd** 서비스를 생성합니다.

사전 요구 사항

- 구성 요소의 **Pod**가 **openshift-user-workload-monitoring** 프로젝트에서 실행되고 있습니다.
- 이 사용자 정의 프로젝트를 모니터링해야 하는 사용자에게 **monitoring-edit** 역할을 부여합니다.

프로세스

1. 가상 머신에 로그인합니다.
2. **node-exporter** 파일에 적용되는 디렉터리 경로를 사용하여 의 **node-exporter** 파일을 가상 머신에 다운로드합니다.

```
$ wget
https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporte
r-1.3.1.linux-amd64.tar.gz
```

3. 실행 파일을 추출하여 **/usr/bin** 디렉터리에 배치합니다.

```
$ sudo tar xvf node_exporter-1.3.1.linux-amd64.tar.gz \
   --directory /usr/bin --strip 1 "*/node_exporter"
```

4. **/etc/systemd/system** 에 **node_exporter.service** 파일을 만듭니다. 이 **systemd** 서비스 파일은 가상 머신이 재부팅될 때 **node-exporter** 서비스를 실행합니다.

```
[Unit]
Description=Prometheus Metrics Exporter
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
User=root
ExecStart=/usr/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

5. **systemd** 서비스를 활성화하고 시작합니다.

```
$ sudo systemctl enable node_exporter.service
$ sudo systemctl start node_exporter.service
```

검증

- **node-exporter** 에이전트가 가상 시스템의 지표를 보고하는지 확인합니다.

```
$ curl http://localhost:9100/metrics
```

출력 예

```
go_gc_duration_seconds{quantile="0"} 1.5244e-05
go_gc_duration_seconds{quantile="0.25"} 3.0449e-05
go_gc_duration_seconds{quantile="0.5"} 3.7913e-05
```

12.4.3. 가상 머신의 사용자 정의 모니터링 레이블 생성

단일 서비스에서 여러 가상 머신에 대한 쿼리를 활성화하려면 가상 머신의 **YAML** 파일에 사용자 지정 레이블을 추가합니다.

사전 요구 사항

- **OpenShift Container Platform CLI oc**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- 웹 콘솔에 액세스하여 가상 머신을 다시 시작합니다.

프로세스

1. 가상 머신 구성 파일의 **template** 사양을 편집합니다. 이 예에서 레이블 모니터에 는 값 지표가 있습니다.

```
spec:
  template:
    metadata:
      labels:
        monitor: metrics
```

2. 가상 머신을 중지하고 다시 시작하여 **monitor** 레이블에 지정된 라벨 이름으로 새 **Pod**를 생성합니다.

12.4.3.1. 메트릭에 대한 node-exporter 서비스 쿼리

/metrics 표준 이름 아래의 **HTTP** 서비스 끝점을 통해 가상 머신에 대한 메트릭이 노출됩니다. 메트릭을 쿼리할 때 **Prometheus**는 가상 머신에서 노출하는 메트릭 끝점에서 메트릭을 직접 스크랩하고 볼 수 있도록 이러한 지표를 표시합니다.

사전 요구 사항

- cluster-admin 권한 또는 monitoring-edit 역할의 사용자로 클러스터에 액세스할 수 있습니다.
- node-exporter 서비스를 구성하여 사용자 정의 프로젝트에 대한 모니터링을 활성화했습니다.

프로세스

1. 서비스의 네임스페이스를 지정하여 HTTP 서비스 끝점을 가져옵니다.

```
$ oc get service -n <namespace> <node-exporter-service>
```

2. node-exporter 서비스에 사용 가능한 모든 지표를 나열하려면 지표 리소스를 쿼리합니다.

```
$ curl http://<172.30.226.162:9100>/metrics | grep -vE "^#|^$"
```

출력 예

```
node_arp_entries{device="eth0"} 1
node_boot_time_seconds 1.643153218e+09
node_context_switches_total 4.4938158e+07
node_cooling_device_cur_state{name="0",type="Processor"} 0
node_cooling_device_max_state{name="0",type="Processor"} 0
node_cpu_guest_seconds_total{cpu="0",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="0",mode="user"} 0
node_cpu_seconds_total{cpu="0",mode="idle"} 1.10586485e+06
node_cpu_seconds_total{cpu="0",mode="iowait"} 37.61
node_cpu_seconds_total{cpu="0",mode="irq"} 233.91
node_cpu_seconds_total{cpu="0",mode="nice"} 551.47
node_cpu_seconds_total{cpu="0",mode="softirq"} 87.3
node_cpu_seconds_total{cpu="0",mode="steal"} 86.12
node_cpu_seconds_total{cpu="0",mode="system"} 464.15
node_cpu_seconds_total{cpu="0",mode="user"} 1075.2
node_disk_discard_time_seconds_total{device="vda"} 0
node_disk_discard_time_seconds_total{device="vdb"} 0
node_disk_discarded_sectors_total{device="vda"} 0
node_disk_discarded_sectors_total{device="vdb"} 0
node_disk_discards_completed_total{device="vda"} 0
node_disk_discards_completed_total{device="vdb"} 0
node_disk_discards_merged_total{device="vda"} 0
node_disk_discards_merged_total{device="vdb"} 0
node_disk_info{device="vda",major="252",minor="0"} 1
node_disk_info{device="vdb",major="252",minor="16"} 1
node_disk_io_now{device="vda"} 0
```

```
node_disk_io_now{device="vdb"} 0
node_disk_io_time_seconds_total{device="vda"} 174
node_disk_io_time_seconds_total{device="vdb"} 0.054
node_disk_io_time_weighted_seconds_total{device="vda"} 259.79200000000003
node_disk_io_time_weighted_seconds_total{device="vdb"} 0.039
node_disk_read_bytes_total{device="vda"} 3.71867136e+08
node_disk_read_bytes_total{device="vdb"} 366592
node_disk_read_time_seconds_total{device="vda"} 19.128
node_disk_read_time_seconds_total{device="vdb"} 0.039
node_disk_reads_completed_total{device="vda"} 5619
node_disk_reads_completed_total{device="vdb"} 96
node_disk_reads_merged_total{device="vda"} 5
node_disk_reads_merged_total{device="vdb"} 0
node_disk_write_time_seconds_total{device="vda"} 240.66400000000002
node_disk_write_time_seconds_total{device="vdb"} 0
node_disk_writes_completed_total{device="vda"} 71584
node_disk_writes_completed_total{device="vdb"} 0
node_disk_writes_merged_total{device="vda"} 19761
node_disk_writes_merged_total{device="vdb"} 0
node_disk_written_bytes_total{device="vda"} 2.007924224e+09
node_disk_written_bytes_total{device="vdb"} 0
```

12.4.4. 노드 내보내기 서비스에 대한 ServiceMonitor 리소스 생성

Prometheus 클라이언트 라이브러리를 사용하고 **/metrics** 끝점에서 메트릭을 스크랩하여 **node-exporter** 서비스에서 노출하는 메트릭에 액세스하고 볼 수 있습니다. **ServiceMonitor CRD**(사용자 정의 리소스 정의)를 사용하여 노드 내보내기 서비스를 모니터링합니다.

사전 요구 사항

- **cluster-admin** 권한 또는 **monitoring-edit** 역할의 사용자로 클러스터에 액세스할 수 있습니다.
- **node-exporter** 서비스를 구성하여 사용자 정의 프로젝트에 대한 모니터링을 활성화했습니다.

프로세스

1. **ServiceMonitor** 리소스 구성에 대한 **YAML** 파일을 생성합니다. 이 예에서 서비스 모니터는 모든 서비스와 레이블 지표 와 일치하고 **30**초마다 **exmet** 포트를 쿼리합니다.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
```

```
metadata:
  labels:
    k8s-app: node-exporter-metrics-monitor
  name: node-exporter-metrics-monitor 1
  namespace: dynamation 2
spec:
  endpoints:
  - interval: 30s 3
    port: exmet 4
    scheme: http
  selector:
    matchLabels:
      servicetype: metrics
```

1 **ServiceMonitor** 의 이름입니다.

2 **ServiceMonitor** 가 생성되는 네임스페이스입니다.

3 포트를 쿼리할 간격입니다.

4 **30**초마다 쿼리되는 포트의 이름입니다.

2. **node-exporter** 서비스에 대한 **ServiceMonitor** 구성을 생성합니다.

```
$ oc create -f node-exporter-metrics-monitor.yaml
```

12.4.4.1. 클러스터 외부에서 노드 내보내기 서비스에 액세스

클러스터 외부에서 **node-exporter** 서비스에 액세스하여 노출된 지표를 볼 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한 또는 **monitoring-edit** 역할의 사용자로 클러스터에 액세스할 수 있습니다.
-

node-exporter 서비스를 구성하여 사용자 정의 프로젝트에 대한 모니터링을 활성화했습니다.

프로세스

1. node-exporter 서비스를 노출합니다.

```
$ oc expose service -n <namespace> <node_exporter_service_name>
```

2. 경로에 대한 FQDN(완전화된 도메인 이름)을 가져옵니다.

```
$ oc get route -o=custom-columns=NAME:.metadata.name,DNS:.spec.host
```

출력 예

```
NAME                 DNS
node-exporter-service   node-exporter-service-dynamation.apps.cluster.example.org
```

3. curl 명령을 사용하여 node-exporter 서비스에 대한 지표를 표시합니다.

```
$ curl -s http://node-exporter-service-dynamation.apps.cluster.example.org/metrics
```

출력 예

```
go_gc_duration_seconds{quantile="0"} 1.5382e-05
go_gc_duration_seconds{quantile="0.25"} 3.1163e-05
go_gc_duration_seconds{quantile="0.5"} 3.8546e-05
go_gc_duration_seconds{quantile="0.75"} 4.9139e-05
go_gc_duration_seconds{quantile="1"} 0.000189423
```

12.4.5. 추가 리소스

- 모니터링 스택 구성
- 사용자 정의 프로젝트 모니터링 활성화
- 메트릭 관리
- 모니터링 대시보드 검토
- 상태 점검을 사용하여 애플리케이션 상태 모니터링
- 구성 맵 생성 및 사용
- 가상 머신 상태 제어

12.5. 가상 머신에 대한 하향식 메트릭 노출

관리자는 먼저 **DownwardMetrics** 기능 게이트를 활성화한 다음 **DownwardMetrics** 장치를 구성하여 제한된 호스트 및 **VM**(가상 머신) 메트릭을 게스트 **VM**에 노출할 수 있습니다.

사용자는 명령줄 또는 **vm-dump-metrics** 툴을 사용하여 메트릭 결과를 볼 수 있습니다.

참고

RHEL(Red Hat Enterprise Linux) 9에서 명령줄을 사용하여 하향 메트릭을 확인합니다. 명령줄을 사용하여 하향 메트릭 보기를 참조하십시오.

vm-dump-metrics 툴은 **RHEL**(Red Hat Enterprise Linux) 9 플랫폼에서 지원되지 않습니다.

12.5.1. DownwardMetrics 기능 게이트 활성화 또는 비활성화

다음 작업 중 하나를 수행하여 **DownwardMetrics** 기능 게이트를 활성화하거나 비활성화할 수 있습니

다.

- 기본 편집기에서 **HyperConverged CR**(사용자 정의 리소스) 편집
- 명령줄 사용

12.5.1.1. YAML 파일에서 Downward 메트릭 기능 게이트 활성화 또는 비활성화

호스트 가상 머신에 대한 하향식 메트릭을 노출하려면 **YAML** 파일을 편집하여 **DownwardMetrics** 기능 게이트를 활성화할 수 있습니다.

사전 요구 사항

- 기능 게이트를 활성화하려면 관리자 권한이 있어야 합니다.

프로세스

1. 다음 명령을 실행하여 기본 편집기에서 **HyperConverged CR**(사용자 정의 리소스)을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 다음과 같이 **DownwardMetrics** 기능 게이트를 활성화하거나 비활성화하도록 선택합니다.
   - **DownwardMetrics** 기능 게이트를 활성화하려면 **spec.featureGates.downwardMetrics** 를 **true** 로 설정합니다. 예를 들면 다음과 같습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
    featureGates:
      downwardMetrics: true
# ...
```

   - **DownwardMetrics** 기능 게이트를 비활성화하려면 **spec.featureGates.downwardMetrics** 를 **false** 로 설정합니다. 예를 들면 다음과 같습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
    featureGates:
      downwardMetrics: false
# ...
```

12.5.1.2. 명령줄에서 하향 메트릭 기능 게이트 활성화 또는 비활성화

호스트 가상 머신에 대한 하향식 메트릭을 노출하려면 명령줄을 사용하여 **DownwardMetrics** 기능 게이트를 활성화할 수 있습니다.

사전 요구 사항

- 기능 게이트를 활성화하려면 관리자 권한이 있어야 합니다.

프로세스

- 다음과 같이 **DownwardMetrics** 기능 게이트를 활성화하거나 비활성화하도록 선택합니다.
  - 다음 예에 표시된 명령을 실행하여 **DownwardMetrics** 기능 게이트를 활성화합니다.

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/downwardMetrics" \
  "value": true}]'
```

  - 다음 예에 표시된 명령을 실행하여 **DownwardMetrics** 기능 게이트를 비활성화합니다.

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/downwardMetrics" \
  "value": false}]'
```

12.5.2. 하향식 메트릭 장치 구성

Downward Metrics 장치가 포함된 구성 파일을 생성하여 호스트 **VM**에 대한 하향식 메트릭을 캡처할 수 있습니다. 이 장치를 추가하면 **virtio-serial** 포트를 통해 지표가 노출되도록 설정됩니다.

사전 요구 사항

- 먼저 **downwardMetrics** 기능 게이트를 활성화해야 합니다.

프로세스

- 다음 예와 같이 **DownwardMetrics** 장치가 포함된 **YAML** 파일을 편집하거나 생성합니다.

DownwardMetrics 구성 파일 예

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: fedora
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        name: fedora-volume
      spec:
        sourceRef:
          kind: DataSource
          name: fedora
          namespace: openshift-virtualization-os-images
        storage:
          resources: {}
          storageClassName: hostpath-csi-basic
  instancetype:
    name: u1.medium
  preference:
    name: fedora
  running: true
  template:
    metadata:
      labels:
        app.kubernetes.io/name: headless
    spec:
      domain:
        devices:
          downwardMetrics: {} 1
      subdomain: headless
      volumes:
        - dataVolume:
            name: fedora-volume
          name: rootdisk
        - cloudInitNoCloud:
            userData: |
```

```
      #cloud-config
      chpasswd:
        expire: false
      password: '<password>' 2
      user: fedora
    name: cloudinitdisk
```

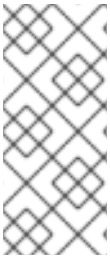
1 downwardMetrics 장치입니다.

2 fedora 사용자의 암호입니다.

12.5.3. 하향 메트릭 보기

다음 옵션 중 하나를 사용하여 Downward 메트릭을 볼 수 있습니다.

- CLI(명령줄 인터페이스)
- vm-dump-metrics 툴



참고

RHEL(Red Hat Enterprise Linux) 9에서 명령줄을 사용하여 하향 메트릭을 확인합니다. vm-dump-metrics 툴은 RHEL(Red Hat Enterprise Linux) 9 플랫폼에서 지원되지 않습니다.

12.5.3.1. 명령줄을 사용하여 하향식 메트릭 보기

게스트 가상 머신(VM) 내부에서 명령을 입력하여 하향식 메트릭을 볼 수 있습니다.

프로세스

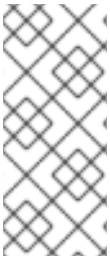
- 다음 명령을 실행합니다.

```
$ sudo sh -c 'printf "GET /metrics/XML\n\n" > /dev/virtio-ports/org.github.vhostmd.1'
```

```
$ sudo cat /dev/virtio-ports/org.github.vhostmd.1
```

12.5.3.2. vm-dump-metrics 툴을 사용하여 하향식 지표 보기

하향식 지표를 보려면 **vm-dump-metrics** 툴을 설치한 다음 툴을 사용하여 지표 결과를 노출합니다.



참고

RHEL(Red Hat Enterprise Linux) 9에서 명령줄을 사용하여 하향 메트릭을 확인합니다. **vm-dump-metrics** 툴은 **RHEL(Red Hat Enterprise Linux) 9** 플랫폼에서 지원되지 않습니다.

프로세스

1. 다음 명령을 실행하여 **vm-dump-metrics** 툴을 설치합니다.

```
$ sudo dnf install -y vm-dump-metrics
```

2. 다음 명령을 실행하여 메트릭 결과를 검색합니다.

```
$ sudo vm-dump-metrics
```

출력 예

```
<metrics>
  <metric type="string" context="host">
    <name>HostName</name>
    <value>node01</value>
[...]
  <metric type="int64" context="host" unit="s">
    <name>Time</name>
    <value>1619008605</value>
  </metric>
  <metric type="string" context="host">
    <name>VirtualizationVendor</name>
    <value>kubevirt.io</value>
  </metric>
</metrics>
```

12.6. 가상 머신 상태 점검

VirtualMachine 리소스에서 준비 및 활성 프로브를 정의하여 VM(가상 머신) 상태 점검을 구성할 수 있습니다.

12.6.1. 준비 및 활성 프로브 정보

준비 및 활성 프로브를 사용하여 비정상 VM(가상 머신)을 감지하고 처리합니다. VM 사양에 프로브를 하나 이상 추가하여 트래픽이 준비되지 않은 VM에 도달하지 않고 VM이 응답하지 않을 때 새 VM이 생성되도록 할 수 있습니다.

준비 상태 프로브 는 VM이 서비스 요청을 수락할 준비가 되었는지 여부를 결정합니다. 프로브가 실패하면 VM이 준비될 때까지 VM이 사용 가능한 엔드포인트 목록에서 제거됩니다.

활성 상태 프로브 는 VM의 응답 여부를 결정합니다. 프로브가 실패하면 VM이 삭제되고 응답성을 복원하기 위해 새 VM이 생성됩니다.

VirtualMachine 오브젝트의 spec.readinessProbe 및 spec.livenessProbe 필드를 설정하여 준비 및 활성 프로브를 구성할 수 있습니다. 이러한 필드는 다음 테스트를 지원합니다.

HTTP GET

프로브는 웹 후크를 사용하여 VM의 상태를 결정합니다. HTTP 응답 코드가 200에서 399 사이인 경우 테스트에 성공합니다. HTTP GET 테스트는 HTTP 상태 코드를 완전히 초기화할 때 반환하는 애플리케이션에 사용할 수 있습니다.

TCP 소켓

프로브는 VM에 대한 소켓을 열려고 시도합니다. VM은 프로브에서 연결을 설정할 수 있는 경우에만 정상으로 간주됩니다. TCP 소켓 테스트는 초기화가 완료된 후 수신 대기를 시작하는 애플리케이션에 사용할 수 있습니다.

게스트 에이전트 ping

프로브는 guest-ping 명령을 사용하여 QEMU 게스트 에이전트가 가상 머신에서 실행 중인지 확인합니다.

12.6.1.1. HTTP 준비 상태 프로브 정의

VM(가상 머신) 구성의 spec.readinessProbe.httpGet 필드를 설정하여 HTTP 준비 상태 프로브를 정의합니다.

프로세스

1. VM 구성 파일에 준비 프로브의 세부 정보를 포함합니다.

HTTP GET 테스트가 있는 샘플 준비 상태 프로브

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        httpGet: 1
          port: 1500 2
          path: /healthz 3
          httpHeaders:
          - name: Custom-Header
            value: Awesome
        initialDelaySeconds: 120 4
        periodSeconds: 20 5
        timeoutSeconds: 10 6
        failureThreshold: 3 7
        successThreshold: 3 8
# ...
```

1 VM에 연결하기 위해 수행할 HTTP GET 요청입니다.

2 프로브가 쿼리하는 VM의 포트입니다. 위의 예에서 프로브는 포트 1500을 쿼리합니다.

3

HTTP 서버에서 액세스할 경로입니다. 위의 예에서 서버의 /healthz 경로에 대한 처리기가 성공 코드를 반환하면 VM이 정상으로 간주됩니다. 핸들러에서 실패 코드를 반환하면 사용 가능한 엔드포인트 목록에서 VM이 제거됩니다.

4

준비 프로브가 시작되기 전에 VM이 시작된 후 시간(초)입니다.

5

프로브 수행 사이의 지연 시간(초)입니다. 기본 지연 시간은 10초입니다. 이 값은 timeoutSeconds 보다 커야 합니다.

6

프로브가 시간 초과되고 VM이 실패한 것으로 간주되는 비활성 시간(초)입니다. 기본값은 1입니다. 이 값은 periodSeconds 보다 작아야 합니다.

7

프로브가 실패할 수 있는 횟수입니다. 기본값은 3입니다. 지정된 횟수의 시도 후 Pod가 Unready로 표시됩니다.

8

프로브에서 실패 후 성공한 것으로 간주하기 위해 성공으로 보고해야 하는 횟수입니다. 기본값은 1입니다.

2. 다음 명령을 실행하여 VM을 생성합니다.

```
$ oc create -f <file_name>.yaml
```

12.6.1.2. TCP 준비 프로브 정의

VM(가상 머신) 구성의 spec.readinessProbe.tcpSocket 필드를 설정하여 TCP 준비 프로브를 정의합니다.

프로세스

1. VM 구성 파일에 TCP 준비 프로브 세부 정보를 포함합니다.

TCP 소켓 테스트를 사용하는 샘플 준비 상태 프로브

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        initialDelaySeconds: 120 1
        periodSeconds: 20 2
        tcpSocket: 3
          port: 1500 4
        timeoutSeconds: 10 5
# ...
```

1

준비 프로브가 시작되기 전에 VM이 시작된 후 시간(초)입니다.

2

프로브 수행 사이의 지연 시간(초)입니다. 기본 지연 시간은 10초입니다. 이 값은 timeoutSeconds 보다 커야 합니다.

3

수행할 TCP 작업입니다.

4

프로브가 쿼리하는 VM의 포트입니다.

5

프로브가 시간 초과되고 VM이 실패한 것으로 간주되는 비활성 시간(초)입니다. 기본 값은 1입니다. 이 값은 periodSeconds 보다 작아야 합니다.

2. 다음 명령을 실행하여 VM을 생성합니다.

```
$ oc create -f <file_name>.yaml
```

12.6.1.3. HTTP 활성 프로브 정의

VM(가상 머신) 구성의 spec.livenessProbe.httpGet 필드를 설정하여 HTTP 활성 프로브를 정의합니다. 준비 프로브와 동일한 방식으로 활성 프로브에 대한 HTTP 및 TCP 테스트를 모두 정의할 수 있습니다. 이 절차에서는 HTTP GET 테스트를 사용하여 샘플 활성 프로브를 구성합니다.

프로세스

1. VM 구성 파일에 HTTP 활성 프로브의 세부 정보를 포함합니다.

HTTP GET 테스트가 포함된 샘플 활성 프로브

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      livenessProbe:
        initialDelaySeconds: 120 (1)
        periodSeconds: 20 (2)
        httpGet: (3)
          port: 1500 (4)
          path: /healthz (5)
          httpHeaders:
          - name: Custom-Header
            value: Awesome
        timeoutSeconds: 10 (6)
# ...
```

(1)

2

프로브 수행 사이의 지연 시간(초)입니다. 기본 지연 시간은 **10**초입니다. 이 값은 **timeoutSeconds** 보다 커야 합니다.

3

VM에 연결하기 위해 수행할 **HTTP GET** 요청입니다.

4

프로브가 쿼리하는 **VM**의 포트입니다. 위의 예에서 프로브는 포트 **1500**을 쿼리합니다. **VM**은 **cloud-init**를 통해 포트 **1500**에 최소 **HTTP** 서버를 설치하고 실행합니다.

5

HTTP 서버에서 액세스할 경로입니다. 위의 예에서 서버의 **/healthz** 경로에 대한 처리기가 성공 코드를 반환하면 **VM**이 정상으로 간주됩니다. 핸들러에서 실패 코드를 반환하면 **VM**이 삭제되고 새 **VM**이 생성됩니다.

6

프로브가 시간 초과되고 **VM**이 실패한 것으로 간주되는 비활성 시간(초)입니다. 기본값은 **1**입니다. 이 값은 **periodSeconds** 보다 작아야 합니다.

2. 다음 명령을 실행하여 **VM**을 생성합니다.

```
$ oc create -f <file_name>.yaml
```

12.6.2. 워치독 정의

다음 단계를 수행하여 워치독을 정의하여 게스트 운영 체제의 상태를 모니터링할 수 있습니다.

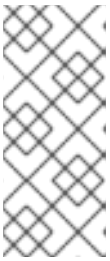
1. **VM**(가상 머신)에 대한 워치독 장치를 구성합니다.

2. 게스트에 **watchdog** 에이전트를 설치합니다.

워치독 장치는 에이전트를 모니터링하고 게스트 운영 체제가 응답하지 않는 경우 다음 작업 중 하나를 수행합니다.

- **poweroff: VM**의 전원이 즉시 꺼집니다. **spec.running** 이 **true** 로 설정되어 있거나 **spec.runStrategy** 가 수동으로 설정되지 않은 경우 **VM**이 재부팅됩니다.

- **reset: VM**이 재부팅되고 게스트 운영 체제가 반응 할 수 없습니다.



참고

재부팅 시간으로 인해 활성 프로브가 시간 초과될 수 있습니다. 클러스터 수준 보호에서 실패한 활성 프로브를 감지하면 **VM**이 강제로 다시 예약되어 재부팅 시간이 증가할 수 있습니다.

- **shutdown: VM**이 모든 서비스를 중지하여 정상적으로 전원을 끕니다.

참고

Windows VM에서는 워치독을 사용할 수 없습니다.

12.6.2.1. 가상 머신에 대한 워치독 장치 구성

VM(가상 머신**)**에 대한 워치독 장치를 구성합니다.

사전 요구 사항

- **VM**에는 **i6300esb** 워치독 장치에 대한 커널 지원이 있어야 합니다. **RHEL(Red Hat Enterprise Linux)** 이미지는 **i6300esb**를 지원합니다.

프로세스

1. 다음 콘텐츠를 사용하여 **YAML** 파일을 생성합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm2-rhel84-watchdog
  name: <vm-name>
```

```
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm2-rhel84-watchdog
    spec:
      domain:
        devices:
          watchdog:
            name: <watchdog>
            i6300esb:
              action: "poweroff" 1
# ...
```

1

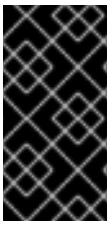
poweroff,reset 또는 shutdown 을 지정합니다.

위의 예제에서는 **poweroff** 작업을 사용하여 **RHEL8 VM**에서 **i6300esb** 워치독 장치를 구성하고 장치를 **/dev/watchdog**로 노출합니다.

이제 워치독 바이너리에서 이 장치를 사용할 수 있습니다.

2. 다음 명령을 실행하여 클러스터에 **YAML** 파일을 적용합니다.

```
$ oc apply -f <file_name>.yaml
```



중요

이 절차는 워치독 기능을 테스트하는 데만 제공되며 프로덕션 시스템에서 실행해서는 안 됩니다.

1. 다음 명령을 실행하여 **VM**이 워치독 장치에 연결되어 있는지 확인합니다.

```
$ lspci | grep watchdog -i
```

2. 다음 명령 중 하나를 실행하여 워치독이 활성 상태인지 확인합니다.

- 커널 패닉을 트리거합니다.

```
# echo c > /proc/sysrq-trigger
```

- 워치독 서비스를 중지합니다.

```
# pkill -9 watchdog
```

12.6.2.2. 게스트에 워치독 에이전트 설치

게스트에 워치독 에이전트를 설치하고 워치독 서비스를 시작합니다.

프로세스

1. root 사용자로 가상 머신에 로그인합니다.
2. watchdog 패키지 및 해당 종속 항목을 설치합니다.

```
# yum install watchdog
```

3. /etc/watchdog.conf 파일에서 다음 행의 주석을 제거하고 변경 사항을 저장합니다.

```
#watchdog-device = /dev/watchdog
```

4. 워치독 서비스가 부팅 시 시작되도록 활성화합니다.

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

12.6.3. 게스트 에이전트 ping 프로브 정의

VM(가상 머신) 구성의 spec.readinessProbe.guestAgentPing 필드를 설정하여 게스트 에이전트 ping 프로브를 정의합니다.

중요

게스트 에이전트 **ping** 프로브는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

사전 요구 사항

- **QEMU** 게스트 에이전트는 가상 머신에 설치하고 활성화해야 합니다.

프로세스

1. **VM** 구성 파일에 게스트 에이전트 **ping** 프로브 세부 정보를 포함합니다. 예를 들면 다음과 같습니다.

게스트 에이전트 **ping** 프로브 샘플

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        guestAgentPing: {} 1
        initialDelaySeconds: 120 2
        periodSeconds: 20 3
        timeoutSeconds: 10 4
        failureThreshold: 3 5
        successThreshold: 3 6
# ...
```

1

VM에 연결하기 위한 게스트 에이전트 **ping** 프로브입니다.

2

선택 사항: 게스트 에이전트 프로브가 시작되기 전에 **VM**이 시작된 후 시간(초)입니다.

3

선택사항: 프로브 수행 사이의 지연 시간(초)입니다. 기본 지연 시간은 **10**초입니다. 이 값은 **timeoutSeconds** 보다 커야 합니다.

4

선택 사항: 프로브가 시간 초과되고 **VM**이 실패한 것으로 간주되는 비활성 시간(초)입니다. 기본값은 **1**입니다. 이 값은 **periodSeconds** 보다 작아야 합니다.

5

선택 사항: 프로브가 실패할 수 있는 횟수입니다. 기본값은 **3**입니다. 지정된 횟수의 시도 후 **Pod**가 **Unready**로 표시됩니다.

6

선택 사항: 프로브에서 실패 후 성공한 것으로 간주하기 위해 성공으로 보고해야 하는 횟수입니다. 기본값은 **1**입니다.

2. 다음 명령을 실행하여 **VM**을 생성합니다.

```
$ oc create -f <file_name>.yaml
```

12.6.4. 추가 리소스

- 상태 점검을 사용하여 애플리케이션 상태 모니터링

12.7. OPENSHIFT VIRTUALIZATION RUNBOOKS

OpenShift Virtualization Operator용 **Runbook**은 **openshift/runbooks Git** 리포지토리에서 유지 관리되며 **GitHub**에서 볼 수 있습니다. **OpenShift Virtualization** 경고를 트리거하는 문제를 진단하고 해결하려면 **runbooks**의 절차를 따르십시오.

OpenShift Virtualization 경고는 웹 콘솔의 가상화 → 개요 탭에 표시됩니다.

12.7.1. CDIDataImportCronOutdated

- CDIDataImportCronOutdated 경고의 실행북 을 확인합니다.

12.7.2. CDIDataVolumeUnusualRestartCount

- CDIDataVolumeUnusualRestartCount 경고의 실행북 을 확인합니다.

12.7.3. CDIDefaultStorageClassDegraded

- CDIDefaultStorageClassDegraded 경고 의 runbook 을 확인합니다.

12.7.4. CDIMultipleDefaultVirtStorageClasses

- CDIMultipleDefaultVirtStorageClasses 경고의 runbook 을 확인합니다.

12.7.5. CDINoDefaultStorageClass

- CDINoDefaultStorageClass 경고 의 runbook 을 확인합니다.

12.7.6. CDINotReady

- CDINotReady 경고 의 runbook 을 확인합니다.

12.7.7. CDIOperatorDown

- CDIOperatorDown 경고 의 runbook 을 확인합니다.

12.7.8. CDIStorageProfilesIncomplete

- CDIStorageProfilesIncomplete 경고의 runbook 을 확인합니다.

12.7.9. CnaoDown

- CnaoDown 경고 의 runbook을 봅니다.

12.7.10. CnaoNMstateMigration

- CnaoNMstateMigration 경고 의 실행북을 확인합니다.

12.7.11. HCOInstallationIncomplete

- HCOInstallationIncomplete 경고 의 실행북을 봅니다.

12.7.12. HPPNotReady

- HPPNotReady 경고 의 실행북 을 확인합니다.

12.7.13. HPPOperatorDown

- HPPOperatorDown 경고 의 실행북 을 확인합니다.

12.7.14. HPPSharingPoolPathWithOS

- HPPSharingPoolPathWithOS 경고의 runbook 을 확인합니다.

12.7.15. KubemacpoolDown

- KubemacpoolDown 경고 의 runbook 을 확인합니다.

12.7.16. KubeMacPoolDuplicateMacsFound

- KubeMacPoolDuplicateMacsFound 경고의 runbook 을 확인합니다.

12.7.17. KubeVirtComponentExceedsRequestedCPU

- KubeVirtComponentExceedsRequestedCPU 경고가 더 이상 사용되지 않습니다.

12.7.18. KubeVirtComponentExceedsRequestedMemory

- KubeVirtComponentExceedsRequestedMemory 경고가 더 이상 사용되지 않습니다.

12.7.19. KubeVirtCRModified

- KubeVirtCR Cryostat 경고 의 실행북 을 확인합니다.

12.7.20. KubeVirtDeprecatedAPIRequested

- KubeVirtDeprecatedAPIRequested 경고 의 실행북 을 확인합니다.

12.7.21. KubeVirtNoAvailableNodesToRunVMs

- KubeVirtNoAvailableNodesToRunVMs 경고의 runbook 을 확인합니다.

12.7.22. KubevirtVmHighMemoryUsage

- KubevirtVmHighMemoryUsage 경고의 runbook 을 확인합니다.

12.7.23. KubeVirtVMIExcessiveMigrations

- KubeVirtVMIExcessiveMigrations 경고의 runbook 을 확인합니다.

12.7.24. LowKVMNodesCount

- LowKVMNodesCount 경고 의 실행북 을 확인합니다.

12.7.25. LowReadyVirtControllersCount

- LowReadyVirtControllersCount 경고의 runbook 을 확인합니다.

12.7.26. LowReadyVirtOperatorsCount

- LowReadyVirtOperatorsCount 경고의 runbook 을 확인합니다.

12.7.27. LowVirtAPICount

- LowVirtAPICount 경고 의 실행북 을 확인합니다.

12.7.28. LowVirtControllersCount

- LowVirtControllersCount 경고의 실행북 을 확인합니다.

12.7.29. LowVirtOperatorCount

- LowVirtOperatorCount 경고 의 runbook 을 확인합니다.

12.7.30. NetworkAddonsConfigNotReady

- NetworkAddonsConfigNotReady 경고의 runbook 을 확인합니다.

12.7.31. NoLeadingVirtOperator

- NoLeadingVirtOperator 경고 의 runbook 을 확인합니다.

12.7.32. NoReadyVirtController

- NoReadyVirtController 경고 의 runbook 을 확인합니다.

12.7.33. NoReadyVirtOperator

- NoReadyVirtOperator 경고 의 runbook을 봅니다.

12.7.34. OrphanedVirtualMachineInstances

- Orphaned CryostatInstances 경고 의 runbook 을 확인합니다.

12.7.35. OutdatedVirtualMachineInstanceWorkloads

- Outdated CryostatInstanceWorkloads 경고의 runbook 을 확인합니다.

12.7.36. SingleStackIPv6Unsupported

- SingleStackIPv6Unsupported 경고의 실행북 을 확인합니다.

12.7.37. SSPCommonTemplatesModificationReverted

- SSPCommonTemplatesModificationReverted 경고의 실행북 을 확인합니다.

12.7.38. SSPDown

- SSPDown 경고 의 runbook을 봅니다.

12.7.39. SSPFailingToReconcile

- SSPFailingToReconcile 경고의 실행북 을 확인합니다.

12.7.40. SSPHighRateRejectedVms

- SSPHighRateRejectedVms 경고의 실행북 을 확인합니다.

12.7.41. SSPTemplateValidatorDown

- SSPTemplateValidatorDown 경고 의 runbook 을 확인합니다.

12.7.42. 지원되지 않는HCOModification

- UnsupportedHCOModification 경고에 대한 runbook 을 확인합니다.

12.7.43. VirtAPIDown

- VirtAPIDown 경고 의 실행북 을 확인합니다.

12.7.44. VirtApiRESTErrorsBurst

- VirtApiRESTErrorsBurst 경고 의 실행북 을 확인합니다.

12.7.45. VirtApiRESTErrorsHigh

- VirtApiRESTErrorsHigh 경고 의 실행북 을 확인합니다.

12.7.46. VirtControllerDown

- VirtControllerDown 경고 의 실행북 을 확인합니다.

12.7.47. VirtControllerRESTErrorsBurst

- VirtControllerRESTErrorsBurst 경고 의 실행북 을 확인합니다.

12.7.48. VirtControllerRESTErrorsHigh

- VirtControllerRESTErrorsHigh 경고 의 실행북 을 확인합니다.

12.7.49. VirtHandlerDaemonSetRolloutFailing

- VirtHandlerDaemonSetRolloutFailing 경고의 실행북 을 확인합니다.

12.7.50. VirtHandlerRESTErrorsBurst

- VirtHandlerRESTErrorsBurst 경고 의 실행북 을 확인합니다.

12.7.51. VirtHandlerRESTErrorsHigh

- VirtHandlerRESTErrorsHigh 경고 의 실행북 을 확인합니다.

12.7.52. VirtOperatorDown

- VirtOperatorDown 경고 의 실행북 을 확인합니다.

12.7.53. VirtOperatorRESTErrorsBurst

- VirtOperatorRESTErrorsBurst 경고 의 실행북 을 확인합니다.

12.7.54. VirtOperatorRESTErrorsHigh

- VirtOperatorRESTErrorsHigh 경고 의 실행북 을 확인합니다.

12.7.55. VirtualMachineCRCErrors

- 경고의 이름이 VMStorageClassWarning 으로 변경되었기 때문에 VirtualMachineCRCErrors 경고의 runbook이 더 이상 사용되지 않습니다.
  - VMStorageClassWarning 경고 의 실행북 을 확인합니다.

12.7.56. VMCannotBeEvicted

- VMCannotBeEvicted 경고의 실행북 을 확인합니다.

12.7.57. VMStorageClassWarning

- VMStorageClassWarning 경고 의 실행북 을 확인합니다.

13장. 지원

13.1. 지원 개요

다음 툴을 사용하여 환경에 대한 데이터를 수집하고 클러스터 및 VM(가상 머신)의 상태를 모니터링하고 OpenShift Virtualization 리소스의 문제를 해결할 수 있습니다.

13.1.1. 웹 콘솔

OpenShift Container Platform 웹 콘솔은 클러스터 및 OpenShift Virtualization 구성 요소 및 리소스에 대한 리소스 사용량, 경고, 이벤트 및 추세를 표시합니다.

표 13.1. 모니터링 및 문제 해결을 위한 웹 콘솔 페이지

페이지	설명
개요 페이지	클러스터 세부 정보, 상태, 경고, 인벤토리 및 리소스 사용량
가상화 → 개요 탭	OpenShift Virtualization 리소스, 사용량, 경고 및 상태
가상화 → 상위 소비자 탭	CPU, 메모리 및 스토리지의 상위 소비자
가상화 → 마이그레이션 탭	실시간 마이그레이션 진행
VirtualMachines → VirtualMachine → VirtualMachine **세부 정보 → 메트릭** 탭	VM 리소스 사용량, 스토리지, 네트워크 및 마이그레이션
VirtualMachines → VirtualMachine → VirtualMachine **세부 정보 → 이벤트** 탭	VM 이벤트 목록
VirtualMachines → VirtualMachine → VirtualMachine details → Cryo stat 탭	VM 상태 조건 및 볼륨 스냅샷 상태

13.1.2. Red Hat 지원을 위한 데이터 수집

Red Hat 지원에 지원 케이스 를 제출하면 디버깅 정보를 제공하는 것이 도움이 됩니다. 다음 단계를 수행하여 디버깅 정보를 수집할 수 있습니다.

환경에 대한 데이터 수집

Prometheus 및 Alertmanager를 구성하고 OpenShift Container Platform 및 OpenShift Virtualization에 대한 must-gather 데이터를 수집합니다.

VM에 대한 데이터 수집

VM에서 must-gather 데이터 및 메모리 덤프를 수집합니다.

OpenShift Virtualization용 must-gather 툴

must-gather 툴을 구성하고 사용합니다.

13.1.3. 문제 해결

OpenShift Virtualization 구성 요소 및 VM 문제를 해결하고 웹 콘솔에서 경고를 트리거하는 문제를 해결합니다.

이벤트

VM, 네임스페이스 및 리소스에 대한 중요한 라이프사이클 정보를 확인합니다.

로그

OpenShift Virtualization 구성 요소 및 VM에 대한 로그를 보고 구성합니다.

데이터 볼륨 문제 해결

조건 및 이벤트를 분석하여 데이터 볼륨 문제를 해결합니다.

13.2. RED HAT 지원을 위한 데이터 수집

Red Hat 지원에 지원 케이스 를 제출할 때 다음 툴을 사용하여 OpenShift Container Platform 및 OpenShift Virtualization에 대한 디버깅 정보를 제공하는 것이 좋습니다.

must-gather 툴

must-gather 툴은 리소스 정의 및 서비스 로그를 포함하여 진단 정보를 수집합니다.

Prometheus

Prometheus는 시계열 데이터베이스이며 메트릭에 대한 규칙 평가 엔진입니다. Prometheus는 처리를 위해 Alertmanager에 경고를 보냅니다.

Alertmanager

Alertmanager 서비스는 Prometheus에서 수신한 경고를 처리합니다. Alertmanager는 또한 경고를 외부 알림 시스템으로 전송합니다.

OpenShift Container Platform 모니터링 스택에 대한 자세한 내용은 OpenShift Container Platform 모니터링 정보를 참조하십시오.

13.2.1. 환경에 대한 데이터 수집

환경에 대한 데이터를 수집하면 근본 원인을 분석하고 결정하는 데 필요한 시간이 최소화됩니다.

사전 요구 사항

- Prometheus 지표 데이터의 보존 시간을 최소 7일로 설정합니다.
- 관련 경고를 캡처하고 클러스터 외부에서 보고 유지할 수 있도록 전용box에 경고 알림을 전송하도록 Alertmanager를 구성합니다.
- 영향을 받는 노드 및 가상 머신의 정확한 수를 기록합니다.

프로세스

1. 클러스터에 대한 must-gather 데이터를 수집합니다.
2. 필요한 경우 Red Hat OpenShift Data Foundation에 must-gather 데이터를 수집합니다.
3. OpenShift Virtualization에 대한 must-gather 데이터를 수집합니다.
4. 클러스터에 대한 Prometheus 지표를 수집합니다.

13.2.2. 가상 머신에 대한 데이터 수집

VM(가상 머신)의 오작동에 대한 데이터를 수집하면 근본 원인을 분석하고 결정하는 데 필요한 시간이 최소화됩니다.

사전 요구 사항

- Linux VM: 최신 QEMU 게스트 에이전트를 설치합니다.
- Windows VMs:
  - Windows 패치 업데이트 세부 정보를 기록합니다.
  - 최신 VirtIO 드라이버를 설치합니다.
  - 최신 QEMU 게스트 에이전트를 설치합니다.
  - RDP(원격 데스크탑 프로토콜)가 활성화된 경우 데스크탑 뷰어를 사용하여 연결하여 연결 소프트웨어에 문제가 있는지 확인합니다.

프로세스

1. /usr/bin/ gather 스크립트를 사용하여 VM의 must-gather 데이터를 수집합니다.
2. 재시작 *하기 전에* 충돌한 VM의 스크린샷을 수집합니다.
3. 해결 시도 *전에* VM에서 메모리 덤프를 수집합니다.
4. 오작동하는 VM에 공통된 요인을 기록합니다. 예를 들어 VM에는 동일한 호스트 또는 네트워크가 있습니다.

13.2.3. OpenShift Virtualization에 must-gather 툴 사용

must-gather 명령을 OpenShift Virtualization 이미지로 실행하여 OpenShift Virtualization 리소스에 대한 데이터를 수집할 수 있습니다.

기본 데이터 수집에는 다음 리소스에 대한 정보가 포함됩니다.

- 하위 오브젝트를 포함한 **OpenShift Virtualization Operator** 네임스페이스
- **OpenShift Virtualization** 사용자 정의 리소스 정의
- 가상 머신이 포함된 네임스페이스
- 기본 가상 머신 정의

인스턴스 유형 정보는 기본적으로 현재 수집되지 않습니다. 그러나 명령을 실행하여 선택적으로 수집할 수 있습니다.

프로세스

- 다음 명령을 실행하여 **OpenShift Virtualization**에 대한 데이터를 수집합니다.

```
$ oc adm must-gather
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-
rhel9:v4.16.0 \
  -- /usr/bin/gather
```

13.2.3.1. must-gather 툴 옵션

oc adm must-gather 명령을 실행하여 필요한 이미지를 명시적으로 지정할 필요 없이 클러스터에 배포된 모든 **Operator** 및 제품의 이미지를 수집해야 합니다. 또는 다음 옵션에 대해 스크립트 및 환경 변수 조합을 지정할 수 있습니다.

- 네임스페이스에서 자세한 **VM**(가상 머신) 정보 수집
- 지정된 **VM**에 대한 자세한 정보 수집
- 이미지, 이미지 스트림 및 **image-stream-tags** 정보 수집
- **must-gather** 툴에서 사용하는 최대 병렬 프로세스 수 제한

13.2.3.1.1. 매개 변수

환경 변수

호환되는 스크립트에 대한 환경 변수를 지정할 수 있습니다.

NS=<namespace_name>

지정한 네임스페이스에서 **virt-launcher Pod** 세부 정보를 포함한 가상 머신 정보를 수집합니다. **VirtualMachine** 및 **VirtualMachineInstance CR** 데이터는 모든 네임스페이스에 대해 수집됩니다.

VM=<vm_name>

특정 가상 머신에 대한 세부 정보를 수집합니다. 이 옵션을 사용하려면 **NS** 환경 변수를 사용하여 네임스페이스도 지정해야 합니다.

PROS=<number_of_processes>

must-gather 툴에서 사용하는 최대 병렬 프로세스 수를 수정합니다. 기본값은 **5** 입니다.

중요

병렬 프로세스를 너무 많이 사용하면 성능 문제가 발생할 수 있습니다. 최대 병렬 프로세스 수를 늘리는 것은 권장되지 않습니다.

스크립트

각 스크립트는 특정 환경 변수 조합과만 호환됩니다.

/usr/bin/gather

기본 **must-gather** 스크립트를 사용하여 모든 네임스페이스에서 클러스터 데이터를 수집하고 기본 **VM** 정보만 포함합니다. 이 스크립트는 **PROS** 변수와만 호환됩니다.

/usr/bin/gather --vms_details

VM 로그 파일, **VM** 정의, 컨트롤 플레인 로그 및 **OpenShift Virtualization** 리소스에 속하는 네임스페이스를 수집합니다. 네임스페이스를 지정하면 해당 하위 오브젝트가 포함됩니다. 네임스페이스 또는 **VM**을 지정하지 않고 이 매개변수를 사용하는 경우 **must-gather** 툴에서 클러스터의 모든 **VM**에 대해 이 데이터를 수집합니다. 이 스크립트는 모든 환경 변수와 호환되지만 **VM** 변수를 사용하는 경우 네임스페이스를 지정해야 합니다.

/usr/bin/gather --images

이미지, 이미지 스트림 및 **image-stream-tags** 사용자 정의 리소스 정보를 수집합니다. 이 스크

립트는 **PROS** 변수와만 호환됩니다.

/usr/bin/gather --instancetypes

인스턴스 유형 정보를 수집합니다. 이 정보는 현재 기본적으로 수집되지 않지만 선택적으로 수집할 수 있습니다.

13.2.3.1.2. 사용법 및 예

환경 변수는 선택 사항입니다. 직접 또는 하나 이상의 호환 환경 변수를 사용하여 스크립트를 실행할 수 있습니다.

표 **13.2.** 호환 가능한 매개변수

스크립트	호환되는 환경 변수
/usr/bin/gather	* **PROS=<number_of_processes>**
/usr/bin/gather --vms_details	* 네임스페이스의 경우: **NS=<namespace_name>** * VM=< **vm_name> NS=<namespace_name**>의 경우 * **PROS=<number_of_processes>**
/usr/bin/gather --images	* **PROS=<number_of_processes>**

구문

단일 패스에서 클러스터의 모든 **Operator** 및 제품에 대해 **must-gather** 로그를 수집하려면 다음 명령을 실행합니다.

```
$ oc adm must-gather --all-images
```

개별 **must-gather** 이미지에 매개변수를 추가로 전달해야 하는 경우 다음 명령을 사용합니다.

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.0 \
  -- <environment_variable_1> <environment_variable_2> <script_name>
```

기본 데이터 수집 병렬 프로세스

기본적으로 **5**개의 프로세스가 병렬로 실행됩니다.

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.0 \
  -- PROS=5 /usr/bin/gather 1
```

1

기본값을 변경하여 병렬 프로세스 수를 수정할 수 있습니다.

자세한 VM 정보

다음 명령은 my namespace 네임스페이스 에서 my-vm VM에 대한 자세한 VM 정보를 수집합니다.

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.0 \
  -- NS=mynamespace VM=my-vm /usr/bin/gather --vms_details 1
```

1

VM 환경 변수를 사용하는 경우 NS 환경 변수는 필수입니다.

image, image-stream 및 image-stream-tags 정보

다음 명령은 클러스터에서 이미지, 이미지 스트림 및 image-stream-tags 정보를 수집합니다.

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.0 \
  /usr/bin/gather --images
```

인스턴스 유형 정보

다음 명령은 클러스터에서 인스턴스 유형 정보를 수집합니다.

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.0 \
  /usr/bin/gather --instancetypes
```

13.3. 문제 해결

OpenShift Virtualization은 VM(가상 머신) 및 가상화 구성 요소를 해결하기 위한 툴과 로그를 제공합니다.

웹 콘솔에 제공된 툴을 사용하거나 oc CLI 툴을 사용하여 OpenShift Virtualization 구성 요소의 문제를 해결할 수 있습니다.

13.3.1. 이벤트

OpenShift Container Platform 이벤트는 중요한 라이프 사이클 정보에 대한 레코드이며 가상 머신, 네임스페이스 및 리소스 문제를 모니터링하고 해결하는 데 유용합니다.

- VM 이벤트: 웹 콘솔의 VirtualMachine 세부 정보 페이지의 Events 탭으로 이동합니다.

네임스페이스 이벤트

다음 명령을 실행하여 네임스페이스 이벤트를 볼 수 있습니다.

```
$ oc get events -n <namespace>
```

특정 이벤트에 대한 자세한 내용은 이벤트 목록을 참조하십시오.

리소스 이벤트

다음 명령을 실행하여 리소스 이벤트를 볼 수 있습니다.

```
$ oc describe <resource> <resource_name>
```

13.3.2. Pod 로그

웹 콘솔 또는 CLI를 사용하여 OpenShift Virtualization Pod의 로그를 볼 수 있습니다. 웹 콘솔에서 LokiStack을 사용하여 집계된 로그 를 볼 수도 있습니다.

13.3.2.1. OpenShift Virtualization Pod 로그 상세 정보 표시 구성

HyperConverged CR(사용자 정의 리소스)을 편집하여 OpenShift Virtualization Pod 로그의 상세 정보 표시 수준을 구성할 수 있습니다.

프로세스

1. 특정 구성 요소에 대한 로그 세부 정보 표시 수준을 설정하려면 다음 명령을 실행하여 기본 텍스트 편집기에서 HyperConverged CR을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. spec.logVerbosityConfig 스탠자를 편집하여 하나 이상의 구성 요소의 로그 수준을 설정합니다. 예를 들면 다음과 같습니다.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  logVerbosityConfig:
    kubevirt:
      virtAPI: 5 (1)
      virtController: 4
      virtHandler: 3
      virtLauncher: 2
      virtOperator: 6
```

(1) 로그 상세 정보 표시 값은 1-9 범위의 정수여야 합니다. 여기서 더 많은 수는 더 자세한 로그를 나타냅니다. 이 예에서는 우선순위 수준이 5 이상인 경우 virtAPI 구성 요소 로그가 노출됩니다.

3. 편집기를 저장하고 종료하여 변경 사항을 적용합니다.

13.3.2.2. 웹 콘솔을 사용하여 virt-launcher Pod 로그 보기

OpenShift Container Platform 웹 콘솔을 사용하여 가상 머신의 virt-launcher Pod 로그를 볼 수 있습니다.

프로세스

1. 가상화 → VirtualMachines 로 이동합니다.
2. 가상 머신을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 일반 타일에서 포드 이름을 클릭하여 Pod 세부 정보 페이지를 엽니다.

4. 로그 탭을 클릭하여 로그를 확인합니다.

13.3.2.3. CLI를 사용하여 OpenShift Virtualization Pod 로그 보기

oc CLI 툴을 사용하여 OpenShift Virtualization Pod의 로그를 볼 수 있습니다.

프로세스

1. 다음 명령을 실행하여 OpenShift Virtualization 네임스페이스에서 Pod 목록을 확인합니다.

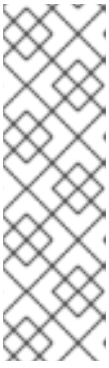
```
$ oc get pods -n openshift-cnv
```

예 13.1. 출력 예

```
NAME                            READY   STATUS    RESTARTS   AGE
disks-images-provider-7gqbc     1/1     Running   0          32m
disks-images-provider-vg4kx     1/1     Running   0          32m
virt-api-57fcc4497b-7qfmc       1/1     Running   0          31m
virt-api-57fcc4497b-tx9nc       1/1     Running   0          31m
virt-controller-76c784655f-7fp6m   1/1     Running   0          30m
virt-controller-76c784655f-f4pbd   1/1     Running   0          30m
virt-handler-2m86x              1/1     Running   0          30m
virt-handler-9qs6z              1/1     Running   0          30m
virt-operator-7ccfdbf65f-q5snk  1/1     Running   0          32m
virt-operator-7ccfdbf65f-vllz8  1/1     Running   0          32m
```

2. 다음 명령을 실행하여 Pod 로그를 확인합니다.

```
$ oc logs -n openshift-cnv <pod_name>
```



참고

Pod가 시작되지 않으면 --previous 옵션을 사용하여 마지막 시도에서 로그를 볼 수 있습니다.

로그 출력을 실시간으로 모니터링하려면 -f 옵션을 사용합니다.

예 13.2. 출력 예

{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-handler.go:453","timestamp":"2022-04-17T08:58:37.373695Z"}
{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-handler.go:453","timestamp":"2022-04-17T08:58:37.373726Z"}
{"component":"virt-handler","level":"info","msg":"setting rate limiter to 5 QPS and 10 Burst","pos":"virt-handler.go:462","timestamp":"2022-04-17T08:58:37.373782Z"}
{"component":"virt-handler","level":"info","msg":"CPU features of a minimum baseline CPU model: map[apic:true clflush:true cmov:true cx16:true cx8:true de:true fpu:true fxsr:true lahf_lm:true lm:true mca:true mce:true mmx:true msr:true mtrr:true nx:true pae:true pat:true pge:true pni:true pse:true pse36:true sep:true sse:true sse2:true sse4.1:true ssse3:true syscall:true tsc:true]","pos":"cpu_plugin.go:96","timestamp":"2022-04-17T08:58:37.390221Z"}
{"component":"virt-handler","level":"warning","msg":"host model mode is expected to contain only one model","pos":"cpu_plugin.go:103","timestamp":"2022-04-17T08:58:37.390263Z"}
{"component":"virt-handler","level":"info","msg":"node-labeller is running","pos":"node_labeller.go:94","timestamp":"2022-04-17T08:58:37.391011Z"}

13.3.3. 게스트 시스템 로그

VM 게스트의 부팅 로그를 보면 문제를 진단하는 데 도움이 될 수 있습니다. 게스트 로그에 대한 액세스를 구성하고 OpenShift Container Platform 웹 콘솔 또는 oc CLI를 사용하여 볼 수 있습니다.

이 기능은 기본적으로 비활성화되어 있습니다. VM에 이 설정이 명시적으로 활성화 또는 비활성화되지 않은 경우 클러스터 전체 기본 설정을 상속합니다.

중요

자격 증명 또는 기타 개인 식별 정보(PII)와 같은 민감한 정보가 직렬 콘솔에 기록되면 다른 모든 표시 텍스트로 기록됩니다. Red Hat은 SSH를 사용하여 직렬 콘솔 대신 중요한 데이터를 전송하는 것이 좋습니다.

13.3.3.1. 웹 콘솔을 사용하여 VM 게스트 시스템 로그에 대한 기본 액세스 활성화

웹 콘솔을 사용하여 VM 게스트 시스템 로그에 대한 기본 액세스를 활성화할 수 있습니다.

프로세스

1. 사이드 메뉴에서 가상화 → 개요 를 클릭합니다.
2. 설정 탭을 클릭합니다.

3. 클러스터 → 게스트 관리를 클릭합니다.

4. 에서 게스트 시스템 로그 액세스 활성화를 설정합니다.

13.3.3.2. CLI를 사용하여 VM 게스트 시스템 로그에 대한 기본 액세스 활성화

HyperConverged CR(사용자 정의 리소스)을 편집하여 VM 게스트 시스템 로그에 대한 기본 액세스를 활성화할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 기본 편집기에서 HyperConverged CR을 엽니다.

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. disableSerialConsoleLog 값을 업데이트합니다. 예를 들면 다음과 같습니다.

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  virtualMachineOptions:
    disableSerialConsoleLog: true (1)
#...
```

(1) VM에서 직렬 콘솔 액세스를 활성화하려면 기본적으로 disableSerialConsoleLog 값을 false 로 설정합니다.

13.3.3.3. 웹 콘솔을 사용하여 단일 VM의 게스트 시스템 로그 액세스 설정

웹 콘솔을 사용하여 단일 VM의 VM 게스트 시스템 로그에 대한 액세스를 구성할 수 있습니다. 이 설정은 클러스터 전체 기본 구성보다 우선합니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.

2. 가상 머신을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.

3. 구성 탭을 클릭합니다.

4. 게스트 시스템 로그 액세스를 on 또는 off로 설정합니다.

13.3.3.4. CLI를 사용하여 단일 VM의 게스트 시스템 로그 액세스 설정

VirtualMachine CR을 편집하여 단일 VM의 VM 게스트 시스템 로그에 대한 액세스를 구성할 수 있습니다. 이 설정은 클러스터 전체 기본 구성보다 우선합니다.

프로세스

1. 다음 명령을 실행하여 가상 머신 매니페스트를 편집합니다.

```
$ oc edit vm <vm_name>
```

2. logSerialConsole 필드의 값을 업데이트합니다. 예를 들면 다음과 같습니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          logSerialConsole: true 1
#...
```

1. 게스트의 직렬 콘솔 로그에 대한 액세스를 활성화하려면 logSerialConsole 값을 true로 설정합니다.

3. 다음 명령을 실행하여 VM에 새 구성을 적용합니다.

```
$ oc apply vm <vm_name>
```

4. 선택 사항: 실행 중인 VM을 편집한 경우 VM을 다시 시작하여 새 구성을 적용합니다. 예를 들면 다음과 같습니다.

```
$ virtctl restart <vm_name> -n <namespace>
```

13.3.3.5. 웹 콘솔을 사용하여 게스트 시스템 로그 보기

웹 콘솔을 사용하여 VM(가상 머신) 게스트의 직렬 콘솔 로그를 볼 수 있습니다.

사전 요구 사항

- 게스트 시스템 로그 액세스가 활성화됩니다.

프로세스

1. 사이드 메뉴에서 가상화 → VirtualMachines 를 클릭합니다.
2. 가상 머신을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. Cryostat 탭 을 클릭합니다.
4. 게스트 시스템 로그를 클릭하여 직렬 콘솔을 로드합니다.

13.3.3.6. CLI를 사용하여 게스트 시스템 로그 보기

oc logs 명령을 실행하여 VM 게스트의 직렬 콘솔 로그를 볼 수 있습니다.

사전 요구 사항

- 게스트 시스템 로그 액세스가 활성화됩니다.

프로세스

- 다음 명령을 실행하여 < namespace > 및 < vm_name >에 대한 고유한 값을 대체하여 로그를 확인합니다.

```
$ oc logs -n <namespace> -l kubevirt.io/domain=<vm_name> --tail=-1 -c guest-
console-log
```

13.3.4. 로그 집계

로그를 집계하고 필터링하여 문제 해결을 용이하게 수행할 수 있습니다.

13.3.4.1. LokiStack을 사용하여 집계된 OpenShift Virtualization 로그 보기

웹 콘솔에서 LokiStack을 사용하여 OpenShift Virtualization Pod 및 컨테이너에 대해 집계된 로그를 볼 수 있습니다.

사전 요구 사항

- LokiStack을 배포했습니다.

프로세스

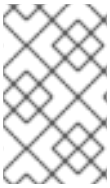
1. 웹 콘솔에서 모니터링 → 로그 로 이동합니다.
2. 로그 유형 목록에서 OpenShift Virtualization 컨트롤 플레인 Pod 및 컨테이너의 경우 virt-launcher Pod 로그 또는 인프라 의 경우 애플리케이션을 선택합니다.
3. 쿼리 표시를 클릭하여 쿼리 필드를 표시합니다.
4. 쿼리 필드에 LogQL 쿼리를 입력하고 쿼리 실행을 클릭하여 필터링된 로그를 표시합니다.

13.3.4.2. OpenShift Virtualization LogQL 쿼리

웹 콘솔의 모니터링 → 로그 페이지에서 Loki Query Language(LogQL) 쿼리를 실행하여 OpenShift Virtualization 구성 요소에 대해 집계된 로그 를 보고 필터링할 수 있습니다.

기본 로그 유형은 *인프라* 입니다. virt-launcher 로그 유형은 ***application*** 입니다.

선택 사항: 줄 필터 표현식을 사용하여 문자열 또는 정규식을 포함하거나 제외할 수 있습니다.



참고

쿼리가 많은 수의 로그와 일치하면 쿼리가 시간 초과될 수 있습니다.

표 13.3. OpenShift Virtualization LogQL 예제 쿼리

Component	LogQL 쿼리
All	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"
cdi-apiserver **cdi-deployment** **cdi-operator**	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="storage"
hco-operator	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="deployment"
kubemacpool	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="network"
virt-api **virt-controller** **virt-handler** **virt-operator**	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="compute"

Component	LogQL 쿼리
ssp-operator	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="schedule"
컨테이너	{log_type=~".+",kubernetes_container_name=~"<container>|<container>"} (1) |json|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" (1) 파이프(|)로 구분된 하나 이상의 컨테이너를 지정합니다.
virt-launcher	이 쿼리를 실행하기 전에 로그 유형 목록에서 **애플리케이션을** 선택해야 합니다. {log_type=~".+", kubernetes_container_name="compute"}|json |!= "custom-ga-command" (1) (1) **|!= "custom-ga-command"** 는 **custom-ga-command** 문자열이 포함된 libvirt 로그를 제외합니다. (BZ#2177684)

행 필터 표현식을 사용하여 문자열 또는 정규식을 포함하거나 제외하도록 로그 행을 필터링할 수 있습니다.

표 13.4. 줄 필터 표현식

줄 필터 표현식	설명
|= "<string>"	로그 줄에는 문자열이 포함되어 있습니다.
!= "<string>"	로그 줄에는 문자열이 포함되어 있지 않음
|~ "<regex>"	로그 행에는 정규식이 포함되어 있습니다.
!~ "<regex>"	로그 행에는 정규식이 포함되지 않음

줄 필터 표현식의 예

```
{log_type=~".+"}|json
|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"
|= "error" != "timeout"
```

LokiStack 및 LogQL에 대한 추가 리소스

- 로그 스토리지 정보
- LokiStack 배포
- Grafana 문서의 LogQL 로그 쿼리

13.3.5. 일반적인 오류 메시지

OpenShift Virtualization 로그에 다음 오류 메시지가 표시될 수 있습니다.

ErrImagePull 또는 ImagePullBackOff

참조되는 이미지의 잘못된 배포 구성 또는 문제를 나타냅니다.

13.3.6. 데이터 볼륨 문제 해결

DataVolume 오브젝트의 **Conditions** 및 **Events** 섹션을 확인하여 문제를 분석하고 해결할 수 있습니다.

13.3.6.1. 데이터 볼륨 조건 및 이벤트 정보

명령으로 생성된 **Conditions** 및 **Events** 섹션의 출력을 검사하여 데이터 볼륨 문제를 진단할 수 있습니다.

```
$ oc describe dv <DataVolume>
```

Conditions 섹션에는 다음과 같은 유형이 표시됩니다.

- Bound
- Running
- Ready

Events 섹션에서는 다음과 같은 추가 정보를 제공합니다.

- 이벤트 Type
- 로깅 Reason
- 이벤트 Source
- 추가 진단 정보가 포함된 Message

oc describe의 출력에 항상 Events가 포함되는 것은 아닙니다.

Status,Reason 또는 Message 가 변경되면 이벤트가 생성됩니다. 조건과 이벤트는 모두 데이터 볼륨의 상태 변화에 반응합니다.

예를 들어 가져오기 작업 중에 URL을 잘못 입력하면 가져오기 작업에서 404 메시지를 생성합니다. 이러한 메시지 변경으로 인해 원인이 포함된 이벤트가 생성됩니다. Conditions 섹션의 출력도 업데이트됩니다.

13.3.6.2. 데이터 볼륨 조건 및 이벤트 분석

describe 명령으로 생성된 Conditions 및 Events 섹션을 검사하여 PVC(영구 볼륨 클레임)와 관련된 데이터 볼륨 상태 및 작업이 활발하게 실행되고 있거나 완료되었는지의 여부를 확인합니다. 데이터 볼륨의 상태와 어떻게 해서 현재 상태가 되었는지에 대한 구체적인 정보를 제공하는 메시지가 표시될 수도 있습니다.

조건은 다양한 형태로 조합될 수 있습니다. 각각 고유의 컨텍스트에서 평가해야 합니다.

다음은 다양한 조합의 예입니다.

- 바인딩 - 이 예제에 성공적으로 바인딩된 PVC가 표시됩니다.

  Type은 Bound이므로 Status가 True입니다. PVC가 바인딩되지 않은 경우 Status는 False입니다.

  PVC가 바인딩되면 PVC가 바인딩되었음을 알리는 이벤트가 생성됩니다. 이 경우 Reason은 Bound이고 Status는 True입니다. Message는 데이터 볼륨이 속한 PVC를 나타냅니다.

  Events 섹션의 Message에서는 PVC가 바인딩된 기간(Age) 및 리소스(From)(이 경우 datavolume-controller)를 포함한 추가 세부 정보를 제공합니다.

  출력 예

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T03:58:24Z
    Last Transition Time:  2020-07-15T03:58:24Z
    Message:               PVC win10-rootdisk Bound
    Reason:                Bound
    Status:                True
    Type:                  Bound
...
  Events:
    Type    Reason    Age    From                Message
    ----    ------    ----   ----                -------
    Normal  Bound     24s    datavolume-controller  PVC example-dv Bound
```

- Running - 이 경우 Type 은 Running 이고 Status 는 False 이므로 시도된 작업이 실패하여 상태를 True 에서 False 로 변경한 이벤트가 발생했음을 나타냅니다.

그러나 **Reason**이 **Completed**이고 **Message** 필드에 **Import Complete**라고 표시됩니다.

Events 섹션의 **Reason** 및 **Message**에는 실패한 작업에 대한 추가 문제 해결 정보가 포함되어 있습니다. 이 예제에서는 **Message**에 **Events** 섹션의 첫 번째 **Warning**에 나열된 **404**로 인해 연결할 수 없다는 내용이 표시됩니다.

이러한 정보를 통해 가져오기 작업이 실행 중이며 데이터 볼륨에 액세스하려는 다른 작업에 대한 경합이 발생한다는 것을 알 수 있습니다.

출력 예

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T04:31:39Z
    Last Transition Time:  2020-07-15T04:31:39Z
    Message:               Import Complete
    Reason:                Completed
    Status:                False
    Type:                  Running
...
  Events:
    Type     Reason      Age              From                Message
    ----     ------      ----             ----                -------
    Warning  Error       12s (x2 over 14s)  datavolume-controller  Unable to connect
    to http data source: expected status code 200, got 404. Status: 404 Not Found
```

- **Ready – Type**이 **Ready**이고 **Status**가 **True**이면 다음 예제와 같이 데이터 볼륨을 사용할 준비가 된 것입니다. 데이터 볼륨을 사용할 준비가 되지 않은 경우에는 **Status**가 **False**입니다.

출력 예

```
Status:
  Conditions:
    Last Heart Beat Time: 2020-07-15T04:31:39Z
    Last Transition Time:  2020-07-15T04:31:39Z
    Status:                True
    Type:                  Ready
```

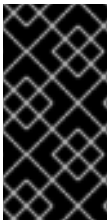
14장. 백업 및 복원

14.1. VM 스냅샷을 사용하여 백업 및 복원

스냅샷을 사용하여 VM(가상 머신)을 백업하고 복원할 수 있습니다. 스냅샷은 다음 스토리지 공급자가 지원합니다.

- Red Hat OpenShift Data Foundation
- Kubernetes Volume Snapshot API를 지원하는 CSI(Container Storage Interface) 드라이버가 있는 기타 클라우드 스토리지 공급자

온라인 스냅샷의 기본 시간 기한은 5분(5m)이며 필요한 경우 변경할 수 있습니다.



중요

온라인 스냅샷은 핫플러그 가상 디스크가 있는 가상 머신에서 지원됩니다. 그러나 가상 머신 사양에 없는 핫플러그 디스크는 스냅샷에 포함되지 않습니다.

무결성이 가장 높은 온라인(실행 상태) VM의 스냅샷을 생성하려면 운영 체제에 포함되지 않은 경우 QEMU 게스트 에이전트를 설치합니다. QEMU 게스트 에이전트는 기본 Red Hat 템플릿에 포함되어 있습니다.

QEMU 게스트 에이전트는 시스템 워크로드에 따라 VM 파일 시스템을 가능한 한 많이 정지하여 일관된 스냅샷을 사용합니다. 이렇게 하면 스냅샷을 생성하기 전에 진행 중인 I/O가 디스크에 기록됩니다. 게스트 에이전트가 없으면 정지를 수행할 수 없으며 최상의 스냅샷을 생성합니다. 스냅샷이 수행된 조건은 웹 콘솔 또는 CLI에 표시되는 스냅샷 표시에 반영됩니다.

14.1.1. 스냅샷 정보

*스냅샷*은 특정 시점의 VM(가상 머신) 상태 및 데이터를 나타냅니다. 스냅샷을 사용하면 백업 및 재해 복구를 위해 기존 VM을 (스냅샷에 표시된) 이전 상태로 복원하거나 이전 개발 버전으로 신속하게 롤백할 수 있습니다.

VM 스냅샷은 전원이 꺼진(중지됨 상태) VM에서 생성되거나 전원이 켜진(실행 중 상태)에서 생성됩니다.

실행 중인 VM의 스냅샷을 생성하는 경우 컨트롤러는 QEMU 게스트 에이전트가 설치되어 실행 중인지 확인합니다. 이 경우 스냅샷을 생성하기 전에 VM 파일 시스템을 중지하고 스냅샷을 만든 후 파일 시스템을 취소합니다.

스냅샷에는 VM에 연결된 각 CSI(Container Storage Interface) 볼륨 복사본과 VM 사양 및 메타데이터 복사본이 저장됩니다. 스냅샷을 생성한 후에는 변경할 수 없습니다.

다음 스냅샷 작업을 수행할 수 있습니다.

- 새 프로젝트 생성
- 스냅샷에서 가상 머신 복사본 생성
- 특정 VM에 연결된 모든 스냅샷 나열
- 스냅샷에서 VM 복원
- 기존 VM 스냅샷 삭제

VM 스냅샷 컨트롤러 및 사용자 정의 리소스

VM 스냅샷 기능에는 스냅샷을 관리하기 위해 사용자 정의 리소스 정의(CRD)로 정의된 세 가지 새 API 오브젝트가 도입되었습니다.

- VirtualMachineSnapshot: 스냅샷을 생성하라는 사용자 요청을 나타냅니다. 여기에는 VM의 현재 상태 정보가 포함됩니다.
- VirtualMachineSnapshotContent: 클러스터의 프로비저닝 리소스를 나타냅니다(스냅샷). VM 스냅샷 컨트롤러에서 생성하며 VM을 복원하는 데 필요한 모든 리소스에 대한 참조를 포함합니다.
- VirtualMachineRestore: 스냅샷에서 VM을 복원하라는 사용자 요청을 나타냅니다.

VM 스냅샷 컨트롤러는 VirtualMachineSnapshot 오브젝트와 이 오브젝트에 대해 생성된 VirtualMachineSnapshotContent 오브젝트를 일대일 매핑으로 바인딩합니다.

14.1.2. 스냅샷 생성

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 VM(가상 머신)의 스냅샷을 생성할 수 있습니다.

14.1.2.1. 웹 콘솔을 사용하여 스냅샷 생성

OpenShift Container Platform 웹 콘솔을 사용하여 VM(가상 머신)의 스냅샷을 생성할 수 있습니다.

VM 스냅샷에는 다음 요구 사항을 충족하는 디스크가 포함됩니다.

- 데이터 볼륨 또는 영구 볼륨 클레임 중 하나
- CSI(Container Storage Interface) 볼륨 스냅샷을 지원하는 스토리지 클래스에 속해야 합니다.

프로세스

1. 웹 콘솔에서 가상화 → VirtualMachines 로 이동합니다.
2. VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. VM이 실행 중인 경우 옵션 메뉴  를 클릭하고 중지 를 선택하여 전원을 끕니다.
4. 스냅샷 탭을 클릭한 후 스냅샷 찍기를 클릭합니다.
5. 스냅샷 이름을 입력합니다.

6. 이 스냅샷에 포함된 디스크를 확장하여 스냅샷에 스토리지 볼륨이 포함되어 있는지 확인합니다.

7. VM에 스냅샷에 포함할 수 없는 디스크가 있고 계속 진행하려면 이 경고를 인식하고 계속 진행하시겠습니까.

8. 저장을 클릭합니다.

14.1.2.2. 명령줄을 사용하여 스냅샷 생성

VirtualMachineSnapshot 오브젝트를 생성하여 오프라인 또는 온라인 VM에 대한 VM(가상 머신) 스냅샷을 생성할 수 있습니다.

사전 요구 사항

- PVC(영구 볼륨 클레임)이 CSI(Container Storage Interface) 볼륨 스냅샷을 지원하는 스토리지 클래스에 있는지 확인합니다.
- OpenShift CLI(oc)를 설치합니다.
- 선택 사항: 스냅샷을 생성할 VM의 전원을 끕니다.

프로세스

1. 다음 예와 같이 새 VirtualMachineSnapshot 의 이름과 소스 VM의 이름을 지정하는 VirtualMachineSnapshot 오브젝트를 정의하는 YAML 파일을 생성합니다.

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineSnapshot
metadata:
  name: <snapshot_name>
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
```

2. VirtualMachineSnapshot 오브젝트를 생성합니다.

```
$ oc create -f <snapshot_name>.yaml
```

스냅샷 컨트롤러는 VirtualMachineSnapshotContent 오브젝트를 생성하여 VirtualMachineSnapshot에 바인딩하고 VirtualMachineSnapshot 오브젝트의 status 및 readyToUse 필드를 업데이트합니다.

3. 선택 사항: 온라인 스냅샷을 생성하는 경우 wait 명령을 사용하여 스냅샷 상태를 모니터링할 수 있습니다.

a. 다음 명령을 실행합니다.

```
$ oc wait <vm_name> <snapshot_name> --for condition=Ready
```

b. 스냅샷 상태를 확인합니다.

- InProgress - 온라인 스냅샷 작업이 아직 진행 중입니다.
- Succeeded - 온라인 스냅샷 작업이 성공적으로 완료되었습니다.
- Failed - 온라인 스냅샷 작업이 실패했습니다.

참고

온라인 스냅샷의 기본 시간 기한은 **5분(5m)**입니다. **5**분 내에 스냅샷이 성공적으로 완료되지 않으면 상태가 **failed**로 설정됩니다. 나중에 파일 시스템이 손상되고 **VM**이 수정되지 않지만 **failed** 스냅샷 이미지를 삭제할 때까지 상태는 실패로 유지됩니다.

기본 시간 기한을 변경하려면 스냅샷 작업이 시간 초과되기 전에 지정할 시간**(m)** 또는 초**(s)**를 사용하여 **VM** 스냅샷 사양에 **FailureDeadline** 속성을 추가합니다.

시간 기한을 설정하지 않으려면 **0**을 지정할 수 있지만 **0**은 응답하지 않는 **VM**이 발생할 수 있으므로 일반적으로 권장되지 않습니다.

m 또는 **s** 와 같은 시간 단위를 지정하지 않으면 기본값은 초**(s)**입니다.

검증

1. **VirtualMachineSnapshot** 오브젝트가 생성되고 **VirtualMachineSnapshotContent** 에 바인딩되고 **readyToUse** 플래그가 **true** 로 설정되어 있는지 확인합니다.

```
$ oc describe vmsnapshot <snapshot_name>
```

출력 예

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineSnapshot
metadata:
  creationTimestamp: "2020-09-30T14:41:51Z"
  finalizers:
  - snapshot.kubevirt.io/vmsnapshot-protection
  generation: 5
  name: mysnap
  namespace: default
  resourceVersion: "3897"
  selfLink:
/apis/snapshot.kubevirt.io/v1alpha1/namespaces/default/virtualmachinesnapshots/my-
vmsnapshot
  uid: 28eedf08-5d6a-42c1-969c-2eda58e2a78d
spec:
  source:
    apiGroup: kubevirt.io
```

```
    kind: VirtualMachine
    name: my-vm
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:42:03Z"
    reason: Operation complete
    status: "False" 1
    type: Progressing
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:42:03Z"
    reason: Operation complete
    status: "True" 2
    type: Ready
  creationTime: "2020-09-30T14:42:03Z"
  readyToUse: true 3
  sourceUID: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
  virtualMachineSnapshotContentName: vmsnapshot-content-28eedf08-5d6a-42c1-
969c-2eda58e2a78d 4
```

1 **status** 필드의 **Progressing** 조건은 스냅샷이 아직 생성 중인지를 나타냅니다.

2 **status** 필드의 **Ready** 조건은 스냅샷 생성 프로세스가 완료되었는지를 나타냅니다.

3 스냅샷을 사용할 준비가 되었는지를 나타냅니다.

4 스냅샷이 스냅샷 컨트롤러에서 생성한 **VirtualMachineSnapshotContent** 오브젝트에 바인딩되도록 지정합니다.

2. **VirtualMachineSnapshotContent** 리소스의 **spec:volumeBackups** 속성을 확인하여 예상 **PVC**가 스냅샷에 포함되어 있는지 확인합니다.

14.1.3. 스냅샷 표시를 사용하여 온라인 스냅샷 확인

스냅샷 표시는 온라인 **VM**(가상 시스템) 스냅샷 작업에 대한 컨텍스트 정보입니다. 오프라인 **VM**(가상

시스템) 스냅샷 작업에는 표시를 사용할 수 없습니다. 표시는 온라인 스냅샷 생성에 대한 세부 정보를 설명하는 데 유용합니다.

사전 요구 사항

- 온라인 VM 스냅샷을 생성하려고 시도해야 합니다.

프로세스

1. 다음 작업 중 하나를 수행하여 스냅샷 표시의 출력을 표시합니다.
   - 명령줄을 사용하여 VirtualMachineSnapshot 오브젝트 YAML의 상태 스탠자에서 표시기 출력을 확인합니다.
   - 웹 콘솔에서 스냅샷 세부 정보 화면에서 VirtualMachineSnapshot → Status 를 클릭합니다.
2. status.indications 매개변수의 값을 확인하여 온라인 VM 스냅샷의 상태를 확인합니다.
   - Online에서는 온라인 스냅샷 생성 중에 VM이 실행 중임을 나타냅니다.
   - GuestAgent 는 온라인 스냅샷 생성 중에 QEMU 게스트 에이전트가 실행 중임을 나타냅니다.
   - NoGuestAgent는 온라인 스냅샷을 생성하는 동안 QEMU 게스트 에이전트가 실행되지 않았음을 나타냅니다. QEMU 게스트 에이전트가 설치되지 않았거나 실행 중이거나 다른 오류로 인해 QEMU 게스트 에이전트를 사용하여 파일 시스템을 중지하고 해석할 수 없습니다.

14.1.4. 스냅샷에서 가상 머신 복원

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 스냅샷에서 VM(가상 머신)을 복원할 수 있습니다.

14.1.4.1. 웹 콘솔을 사용하여 스냅샷에서 VM 복원

VM(가상 머신)을 OpenShift Container Platform 웹 콘솔의 스냅샷에 표시된 이전 구성으로 복원할

수 있습니다.

프로세스

1. 웹 콘솔에서 가상화 → VirtualMachines 로 이동합니다.
2. VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. VM이 실행 중인 경우 옵션 메뉴  를 클릭하고 중지 를 선택하여 전원을 끕니다.
4. 스냅샷 탭을 클릭하여 VM과 연결된 스냅샷 목록을 확인합니다.
5. 스냅샷을 선택하여 스냅샷 세부 정보 화면을 엽니다.
6. 옵션 메뉴  를 클릭하고 스냅샷에서 VirtualMachine 복원을 선택합니다.
7. 복원을 클릭합니다.

14.1.4.2. 명령줄을 사용하여 스냅샷에서 VM 복원

명령줄을 사용하여 기존 VM(가상 머신)을 이전 구성으로 복원할 수 있습니다. 오프라인 VM 스냅샷에서만 복원할 수 있습니다.

사전 요구 사항

- 복원할 VM의 전원을 끕니다.

프로세스

1. YAML 파일을 생성하여 복원할 VM의 이름과 다음 예와 같이 소스로 사용할 스냅샷 이름을 지정하는 VirtualMachineRestore 오브젝트를 정의합니다.

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineRestore
metadata:
  name: <vm_restore>
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
  virtualMachineSnapshotName: <snapshot_name>
```

2. VirtualMachineRestore 오브젝트를 생성합니다.

```
$ oc create -f <vm_restore>.yaml
```

스냅샷 컨트롤러는 VirtualMachineRestore 오브젝트의 상태 필드를 업데이트하고 기존 VM 구성을 스냅샷의 콘텐츠로 교체합니다.

검증

- VM이 스냅샷에 표시된 이전 상태로 복원되고 전체 플래그가 true 로 설정되어 있는지 확인합니다.

```
$ oc get vmrestore <vm_restore>
```

출력 예

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineRestore
metadata:
creationTimestamp: "2020-09-30T14:46:27Z"
generation: 5
name: my-vmrestore
namespace: default
ownerReferences:
- apiVersion: kubevirt.io/v1
  blockOwnerDeletion: true
  controller: true
  kind: VirtualMachine
  name: my-vm
  uid: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
```

```
  resourceVersion: "5512"
  selfLink:
/apis/snapshot.kubevirt.io/v1alpha1/namespaces/default/virtualmachinerestores/my-
vmrestore
  uid: 71c679a8-136e-46b0-b9b5-f57175a6a041
  spec:
    target:
      apiGroup: kubevirt.io
      kind: VirtualMachine
      name: my-vm
  virtualMachineSnapshotName: my-vmsnapshot
  status:
  complete: true (1)
  conditions:
  - lastProbeTime: null
  lastTransitionTime: "2020-09-30T14:46:28Z"
  reason: Operation complete
  status: "False" (2)
  type: Progressing
  - lastProbeTime: null
  lastTransitionTime: "2020-09-30T14:46:28Z"
  reason: Operation complete
  status: "True" (3)
  type: Ready
  deletedDataVolumes:
  - test-dv1
  restoreTime: "2020-09-30T14:46:28Z"
  restores:
  - dataVolumeName: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-
datavolumedisk1
  persistentVolumeClaim: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-
datavolumedisk1
  volumeName: datavolumedisk1
  volumeSnapshotName: vmsnapshot-28eedf08-5d6a-42c1-969c-2eda58e2a78d-
volume-datavolumedisk1
```

(1) VM을 스냅샷에 표시된 상태로 복원하는 프로세스가 완료되었는지를 나타냅니다.

(2) status 필드의 Progressing 조건은 VM이 아직 복원 중인지를 나타냅니다.

(3) status 필드의 Ready 조건은 VM 복원 프로세스가 완료되었는지를 나타냅니다.

14.1.5. 스냅샷 삭제

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 VM(가상 머신)의 스냅샷을 삭제할 수 있습니다.

14.1.5.1. 웹 콘솔을 사용하여 스냅샷 삭제

웹 콘솔을 사용하여 기존 VM(가상 머신) 스냅샷을 삭제할 수 있습니다.

프로세스

1. 웹 콘솔에서 가상화 → VirtualMachines 로 이동합니다.
2. VM을 선택하여 VirtualMachine 세부 정보 페이지를 엽니다.
3. 스냅샷 탭을 클릭하여 VM과 연결된 스냅샷 목록을 확인합니다.
4. 스냅샷 옆에 있는 옵션 메뉴

를 클릭하고 스냅샷 삭제 를 선택합니다.

5. 삭제를 클릭합니다.

14.1.5.2. CLI에서 가상 머신 스냅샷 삭제

적절한 VirtualMachineSnapshot 오브젝트를 삭제하여 기존 VM(가상 머신) 스냅샷을 삭제할 수 있습니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.

프로세스

- VirtualMachineSnapshot 오브젝트를 삭제합니다.

```
$ oc delete vmsnapshot <snapshot_name>
```

스냅샷 컨트롤러는 VirtualMachineSnapshot을 연결된 VirtualMachineSnapshotContent 오브젝트와 함께 삭제합니다.

검증

- 스냅샷이 삭제되고 더 이상 이 VM에 연결되어 있지 않은지 확인합니다.

```
$ oc get vmsnapshot
```

14.1.6. 추가 리소스

- CSI 볼륨 스냅샷

14.2. 가상 머신 백업 및 복원

OADP Operator를 설치하고 백업 위치를 구성하여 OpenShift Virtualization을 사용하여 OADP(OpenShift API for Data Protection)를 설치할 수 있습니다. 그런 다음 데이터 보호 애플리케이션을 설치할 수 있습니다.

OpenShift API for Data Protection 을 사용하여 가상 머신을 백업하고 복원합니다.

참고

OpenShift Virtualization을 사용한 OpenShift API for Data Protection에서는 다음과 같은 백업 및 복원 스토리지 옵션을 지원합니다.

- CSI(Container Storage Interface) 백업
- DataMover를 사용한 CSI(Container Storage Interface) 백업

다음 스토리지 옵션은 제외됩니다.

- 파일 시스템 백업 및 복원
- 볼륨 스냅샷 백업 및 복원

자세한 내용은 파일 시스템 백업: Kopia 또는 Restic을 사용하여 애플리케이션 백업을 참조하십시오.

제한된 네트워크 환경에 OADP Operator를 설치하려면 먼저 기본 OperatorHub 소스를 비활성화하고 Operator 카탈로그를 미러링해야 합니다. 자세한 내용은 제한된 네트워크에서 Operator Lifecycle Manager 사용을 참조하십시오.

14.2.1. OpenShift Virtualization을 사용하여 OADP 설치 및 구성

클러스터 관리자는 OADP Operator를 설치하여 OADP를 설치합니다.

최신 버전의 OADP Operator는 Velero 1.14 를 설치합니다.

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. 스토리지 공급자의 지침에 따라 **OADP Operator**를 설치합니다.

2. **kubevirt** 및 **openshift OADP** 플러그인으로 **DPA(Data Protection Application)**를 설치합니다.

3. **Backup CR**(사용자 정의 리소스)을 생성하여 가상 머신을 백업합니다.

주의

Red Hat 지원은 다음 옵션으로 제한됩니다.

- **CSI** 백업
- **DataMover**를 사용한 **CSI** 백업.

Restore CR을 생성하여 **Backup CR**을 복원합니다.

추가 리소스

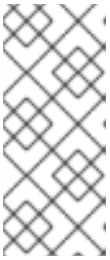
- **OADP** 플러그인
- 백업 사용자 정의 리소스(CR)
- **CR** 복원
- 제한된 네트워크에서 **Operator Lifecycle Manager** 사용

14.2.2. 데이터 보호 애플리케이션 1.3 설치

DataProtectionApplication API의 인스턴스를 생성하여 **DPA(Data Protection Application)**를 설치합니다.

사전 요구 사항

- **OADP Operator**를 설치해야 합니다.
- 오브젝트 스토리지를 백업 위치로 구성해야 합니다.
- 스냅샷을 사용하여 **PV**를 백업하는 경우 클라우드 공급자는 기본 스냅샷 **API** 또는 **CSI(Container Storage Interface)** 스냅샷을 지원해야 합니다.
- 백업 및 스냅샷 위치에서 동일한 인증 정보를 사용하는 경우 기본 이름 **cloud-credentials** 로 **Secret** 을 생성해야 합니다.
- 백업 및 스냅샷 위치에서 다른 인증 정보를 사용하는 경우 다음 두 개의 **Secret** 을 생성해야 합니다.
  - 백업 위치에 대한 사용자 정의 이름이 있는 **Secret**입니다. 이 **Secret**을 **DataProtectionApplication CR**에 추가합니다.
  - 스냅샷 위치에 대한 다른 사용자 지정 이름이 있는 **Secret** 입니다. 이 **Secret**을 **DataProtectionApplication CR**에 추가합니다.



참고

설치 중에 백업 또는 스냅샷 위치를 지정하지 않으려면 빈 **credentials-velero** 파일을 사용하여 기본 **Secret**을 생성할 수 있습니다. 기본 **Secret** 이 없으면 설치에 실패합니다.

프로세스

1. **Operators** → 설치된 **Operator** 를 클릭하고 **OADP Operator**를 선택합니다.

2. 제공된 API 아래에서 DataProtectionApplication 상자에서 인스턴스 생성 을 클릭합니다.

3. YAML 보기를 클릭하고 DataProtectionApplication 매니페스트의 매개변수를 업데이트합니다.

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
 name: <dpa_sample>
 namespace: openshift-adp (1)
spec:
 configuration:
  velero:
   defaultPlugins:
    - kubevirt (2)
    - gcp (3)
    - csi (4)
    - openshift (5)
   resourceTimeout: 10m (6)
  nodeAgent: (7)
   enable: true (8)
   uploaderType: kopia (9)
   podConfig:
    nodeSelector: <node_selector> (10)
 backupLocations:
  - velero:
     provider: gcp (11)
     default: true
     credential:
      key: cloud
      name: <default_secret> (12)
     objectStorage:
      bucket: <bucket_name> (13)
      prefix: <prefix> (14)
```

(1) OADP의 기본 네임스페이스는 openshift-adp 입니다. 네임스페이스는 변수이며 구성 가능합니다.

(2) kubevirt 플러그인은 OpenShift Virtualization에 필수입니다.

(3) 백업 공급자의 플러그인을 지정합니다(예: gcp).

4

CSI 스냅샷을 사용하여 **PV**를 백업하려면 **csi** 플러그인이 필요합니다. **csi** 플러그인은 **Velero CSI 베타 스냅샷 API** 를 사용합니다. 스냅샷 위치를 구성할 필요가 없습니다.

5

openshift 플러그인은 필수입니다.

6

Velero CRD 가용성, **volumeSnapshot** 삭제, 백업 리포지토리 가용성과 같이 시간 초과가 발생하기 전에 여러 **Velero** 리소스를 대기하는 시간(분)을 지정합니다. 기본값은 **10m** 입니다.

7

관리 요청을 서버로 라우팅하는 관리 에이전트입니다.

8

nodeAgent 를 활성화하고 파일 시스템 백업을 수행하려면 이 값을 **true** 로 설정합니다.

9

Built-in DataMover를 사용하려면 업로드기로 **kopia** 를 입력합니다. **nodeAgent** 는 데몬 세트를 배포합니다. 즉, **nodeAgent Pod**가 각 작동 중인 노드에서 실행됩니다. **Backup CR**에 **spec.defaultVolumesToFsBackup: true** 를 추가하여 파일 시스템 백업을 구성할 수 있습니다.

10

Kopia를 사용할 수 있는 노드를 지정합니다. 기본적으로 **Kopia**는 모든 노드에서 실행됩니다.

11

백업 공급자를 지정합니다.

12

백업 공급자에 기본 플러그인을 사용하는 경우 **Secret** 의 올바른 기본 이름(예: **cloud-credentials-gcp**)을 지정합니다. 사용자 지정 이름을 지정하면 사용자 지정 이름이 백업 위치에 사용됩니다. **Secret** 이름을 지정하지 않으면 기본 이름이 사용됩니다.

13

14 버킷이 여러 용도로 사용되는 경우 **Velero** 백업의 접두사를 지정합니다(예: **velero**).

4. 생성을 클릭합니다.

검증

1. 다음 명령을 실행하여 **OADP(OpenShift API for Data Protection)** 리소스를 확인하여 설치를 확인합니다.

```
$ oc get all -n openshift-adp
```

출력 예

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/oadp-operator-controller-manager-67d9494d47-6l8z8   2/2     Running   0          2m8s
pod/node-agent-9cq4q                                 1/1     Running   0          94s
pod/node-agent-m4lts                                 1/1     Running   0          94s
pod/node-agent-pv4kr                                 1/1     Running   0          95s
pod/velero-588db7f655-n842v                          1/1     Running   0          95s

NAME                                                     TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)    AGE
service/oadp-operator-controller-manager-metrics-service   ClusterIP   172.30.70.140
<none>        8443/TCP   2m8s
service/openshift-adp-velero-metrics-svc                 ClusterIP   172.30.10.0      <none>
8085/TCP   8h

NAME                  DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset.apps/node-agent   3         3         3       3            3           <none>          96s

NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/oadp-operator-controller-manager   1/1     1            1           2m9s
deployment.apps/velero                          1/1     1            1           96s

NAME                                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/oadp-operator-controller-manager-67d9494d47   1         1         1       2m9s
replicaset.apps/velero-588db7f655                          1         1         1       96s
```

2. 다음 명령을 실행하여 **DPA(DataProtectionApplication)**가 조정되었는지 확인합니다.

```
$ oc get dpa dpa-sample -n openshift-adp -o jsonpath='{.status}'
```

출력 예

```
{"conditions":[{"lastTransitionTime":"2023-10-27T01:23:57Z","message":"Reconcile
complete","reason":"Complete","status":"True","type":"Reconciled"}]}
```

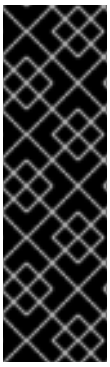
3. 유형이 **Reconciled** 으로 설정되어 있는지 확인합니다.

4. 백업 스토리지 위치를 확인하고 다음 명령을 실행하여 **PHASE** 가 사용 가능한지 확인합니다.

```
$ oc get backupStorageLocation -n openshift-adp
```

출력 예

```
NAME          PHASE       LAST VALIDATED   AGE     DEFAULT
dpa-sample-1  Available   1s               3d16h   true
```



중요

Red Hat은 OADP 1.3.x 이상에서 OpenShift Virtualization 4.14 이상을 지원합니다.

1.3.0 이전 OADP 버전은 OpenShift Virtualization의 백업 및 복원에 지원되지 않습니다.

14.3. 재해 복구

OpenShift Virtualization에서는 재해 복구(DR) 솔루션 사용을 지원하여 사이트 중단 후 환경을 복구할 수 있습니다. 이러한 방법을 사용하려면 OpenShift Virtualization 배포를 사전에 계획해야 합니다.

14.3.1. 재해 복구 방법 정보

재해 복구(DR) 개념, 아키텍처 및 계획 고려 사항에 대한 개요는 Red Hat 지식베이스의 Red Hat OpenShift Virtualization 재해 복구 가이드를 참조하십시오.

OpenShift Virtualization의 두 가지 주요 DR 방법은 메트로폴리탄 재해 복구 (Metro-DR) 및 Regional-DR입니다.

Metro-DR

Metro-DR은 동기 복제를 사용합니다. 기본 사이트 및 보조 사이트 모두에서 스토리지에 기록되므로 데이터가 항상 사이트 간에 동기화됩니다. 스토리지 공급자는 동기화가 성공했는지 확인하기 때문에 환경이 스토리지 공급자의 처리량 및 대기 시간 요구 사항을 충족해야 합니다.

regional-DR

regional-DR은 비동기 복제를 사용합니다. 기본 사이트의 데이터는 정기적으로 보조 사이트와 동기화됩니다. 이 유형의 복제에서는 기본 사이트와 보조 사이트 간에 대기 시간이 길어질 수 있습니다.

14.3.1.1. Red Hat OpenShift Data Foundation 용 Metro-DR

OpenShift Virtualization은 기본 사이트와 보조 사이트에 설치된 관리형 OpenShift Virtualization 클러스터 간에 양방향 동기 데이터 복제를 제공하는 OpenShift Data Foundation 용 Metro-DR 솔루션을 지원합니다. 이 솔루션은 RHACM(Red Hat Advanced Cluster Management), Red Hat Ceph Storage 및 OpenShift Data Foundation 구성 요소를 결합합니다.

사이트 재해 발생 시 이 솔루션을 사용하여 기본 사이트에서 보조 사이트로 애플리케이션을 실패하고 재해 사이트를 복원한 후 애플리케이션을 기본 사이트로 다시 재배치할 수 있습니다.

이 동기 솔루션은 대도시 거리 데이터 센터만 사용할 수 있으며 10밀리초 미만의 대기 시간을 사용할 수 있습니다.

OpenShift Virtualization과 OpenShift Data Foundation용 Metro-DR 솔루션 사용에 대한 자세한 내용은 Red Hat 지식베이스를 참조하십시오.