OpenShift Container Platform 4.7

스토리지

OpenShift Container Platform에서 스토리지 구성 및 사용

Last Updated: 2023-04-08

OpenShift Container Platform 4.7 스토리지

OpenShift Container Platform에서 스토리지 구성 및 사용

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

법적 공지

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

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

초록

이 문서에서는 다양한 스토리지 백엔드에서 영구 볼륨을 구성하고 Pod의 동적 할당을 관리하는 방법을 설명합니다.

차례

1장. OPENSHIFT CONTAINER PLATFORM 스토리지 개요 6
1.1. 스토리지 유형 6
1.1.1. 임시 스토리지 6
1.1.2. 영구 스토리지 6
1.2. CSI(CONTAINER STORAGE INTERFACE) 6
1.3. 동적 프로비저닝 6

2장. 임시 스토리지 이해 7
2.1. 개요 7
2.2. 임시 스토리지 유형 7
루트 7
런타임 7
2.3. 임시 데이터 스토리지 관리 7
2.4. 임시 스토리지 모니터링 7

3장. 영구 스토리지 이해 9
3.1. 영구 스토리지 개요 9
3.2. 볼륨 및 클레임의 라이프사이클 9
3.2.1. 스토리지 프로비저닝 9
3.2.2. 클레임 바인딩 9
3.2.3. Pod 및 클레임된 PV 사용 10
3.2.4. 사용 중 스토리지 오브젝트 보호 10
3.2.5. 영구 볼륨 해제 10
3.2.6. 영구 볼륨 회수 정책 10
3.2.7. 수동으로 영구 볼륨 회수 11
3.2.8. 영구 볼륨의 회수 정책 변경 11
3.3. PV(영구 볼륨) 12
3.3.1. PV 유형 13
3.3.2. 용량 13
3.3.3. 액세스 모드 13
3.3.4. 단계 15
3.3.4.1. 마운트 옵션 15
3.4. 영구 볼륨 클레임 16
3.4.1. 스토리지 클래스 17
3.4.2. 액세스 모드 17
3.4.3. 리소스 17
3.4.4. 클레임을 볼륨으로 18
3.5. 블록 볼륨 지원 18
3.5.1. 블록 볼륨 예 19

4장. 영구 스토리지 구성 22
4.1. AWS ELASTIC BLOCK STORE를 사용하는 영구저장장치 22
4.1.1. 추가 리소스 22
4.1.2. EBS 스토리지 클래스 생성 22
4.1.3. 영구 볼륨 클레임 생성 23
4.1.4. 볼륨 형식 23
4.1.5. 노드의 최대 EBS 볼륨 수 23
4.1.6. 추가 리소스 23
4.2. AZURE를 사용하는 영구 스토리지 23
4.2.1. Azure 스토리지 클래스 생성 24
4.2.2. 영구 볼륨 클레임 생성 25
4.2.3. 볼륨 형식 25

4.3. AZURE FILE을 사용하는 영구 스토리지 25
4.3.1. Azure File 공유 영구 볼륨 클레임 생성 26
4.3.2. Pod에서 Azure 파일 공유 마운트 27
4.4. CINDER를 사용하는 영구 스토리지 28
4.4.1. Cinder를 사용한 수동 프로비저닝 28
4.4.1.1. 영구 볼륨 생성 28
4.4.1.2. 영구 볼륨 포맷 29
4.4.1.3. Cinder 볼륨 보안 29
4.5. 파이버 채널을 사용하는 영구 스토리지 30
4.5.1. 프로비저닝 31
4.5.1.1. 디스크 할당량 강제 적용 32
4.5.1.2. 파이버 채널 볼륨 보안 32
4.6. FLEXVOLUME을 사용하는 영구 스토리지 32
4.6.1. FlexVolume 드라이버 정보 32
4.6.2. FlexVolume 드라이버 예 32
4.6.3. FlexVolume 드라이버 설치 33
4.6.4. FlexVolume 드라이버를 사용한 스토리지 사용 34
4.7. GCE 영구 디스크를 사용하는 스토리지 36
4.7.1. GCE 스토리지 클래스 생성 36
4.7.2. 영구 볼륨 클레임 생성 36
4.7.3. 볼륨 형식 37
4.8. HOSTPATH를 사용하는 영구 스토리지 37
4.8.1. 개요 37
4.8.2. 정적으로 hostPath 볼륨을 프로비저닝 38
4.8.3. 권한이 있는 Pod에서 hostPath 공유 마운트 39
4.9. ISCSI를 사용하는 영구 스토리지 40
4.9.1. 프로비저닝 40
4.9.2. 디스크 할당량 강제 적용 41
4.9.3. iSCSI 볼륨 보안 41
4.9.3.1. CHAP(Challenge Handshake Authentication Protocol) 구성 41
4.9.4. iSCSI 다중 경로 42
4.9.5. iSCSI 사용자 정의 이니시에이터 IQN 42
4.10. 로컬 블록을 사용하는 영구 스토리지 43
4.10.1. Local Storage Operator 설치 43
4.10.2. Local Storage Operator를 사용하여 로컬 볼륨을 프로비저닝 45
4.10.3. Local Storage Operator없이 로컬 볼륨 프로비저닝 48
4.10.4. 로컬 볼륨 영구 볼륨 클레임 생성 51
4.10.5. 로컬 클레임 연결 51
4.10.6. 로컬 스토리지 장치에 대한 검색 및 프로비저닝 자동화 52
4.10.7. Local Storage Operator Pod에서 허용 오차 사용 55
4.10.8. Local Storage Operator 리소스 삭제 56
4.10.8.1. 로컬 볼륨 또는 로컬 볼륨 세트 제거 56
4.10.8.2. Local Storage Operator 설치 제거 58
4.11. NFS를 사용하는 영구저장장치 59
4.11.1. 프로비저닝 59
4.11.2. 디스크 할당량 강제 적용 60
4.11.3. NFS 볼륨 보안 61
4.11.3.1. 그룹 ID 61
4.11.3.2. 사용자 ID 62
4.11.3.3. SELinux 63
4.11.3.4. 내보내기 설정 63
4.11.4. 리소스 회수 64
4.11.5. 추가 구성 및 문제 해결 65

4.12. RED HAT OPENSHIFT 컨테이너 스토리지 65
4.13. VMWARE VSPHERE 볼륨을 사용하는 영구 스토리지 67
4.13.1. 동적으로 VMware vSphere 볼륨 프로비저닝 68
4.13.2. 사전 요구 사항 68
4.13.2.1. UI를 사용하여 VMware vSphere 볼륨을 동적으로 프로비저닝 68
4.13.2.2. CLI를 사용하여 VMware vSphere 볼륨을 동적으로 프로비저닝 69
4.13.3. 정적으로 프로비저닝 VMware vSphere 볼륨 69
4.13.3.1. VMware vSphere 볼륨 포맷 71

5장. CSI(CONTAINER STORAGE INTERFACE) 사용 72
5.1. CSI 볼륨 구성 72
5.1.1. CSI 아키텍처 72
5.1.1.1. 외부 CSI 컨트롤러 72
5.1.1.2. CSI 드라이버 데몬 세트 73
5.1.2. OpenShift Container Platform에서 지원되는 CSI 드라이버 73
5.1.3. 동적 프로비저닝 74
5.1.4. CSI 드라이버 사용 예 75
5.2. CSI 인라인 임시 볼륨 75
5.2.1. CSI 인라인 임시 볼륨 개요 76
5.2.1.1. 지원 제한 76
5.2.2. Pod 사양에 CSI 인라인 임시 볼륨 포함 76
5.3. CSI 볼륨 스냅샷 77
5.3.1. CSI 볼륨 스냅샷 개요 77
5.3.2. CSI 스냅샷 컨트롤러 및 사이드카 78
5.3.2.1. 외부 컨트롤러 78
5.3.2.2. 외부 사이드카 78
5.3.3. CSI Snapshot Controller Operator 정보 78
5.3.3.1. 볼륨 스냅샷 CRD 78
5.3.4. 볼륨 스냅샷 프로비저닝 79
5.3.4.1. 동적 프로비저닝 79
5.3.4.2. 수동 프로비저닝 79
5.3.5. 볼륨 스냅샷 생성 79
5.3.6. 볼륨 스냅샷 삭제 82
5.3.7. 볼륨 스냅샷 복원 83
5.4. CSI 볼륨 복제 84
5.4.1. CSI 볼륨 복제 개요 84
5.4.1.1. 지원 제한 84
5.4.2. CSI 볼륨 복제 프로비저닝 84
5.5. AWS ELASTIC BLOCK STORE CSI DRIVER OPERATOR 86
5.5.1. 개요 86
5.5.2. CSI 정보 86
5.6. GCP PD CSI DRIVER OPERATOR 87
5.6.1. 개요 87
5.6.2. CSI 정보 88
5.6.3. GCP PD CSI 드라이버 스토리지 클래스 매개변수 88
5.6.4. 사용자 정의 암호화 영구 볼륨 생성 88
5.7. OPENSTACK CINDER CSI DRIVER OPERATOR 90
5.7.1. 개요 90
5.7.2. CSI 정보 91
5.7.3. OpenStack Cinder CSI를 기본 스토리지 클래스로 설정 91
5.8. OPENSTACK MANILA CSI DRIVER OPERATOR 92
5.8.1. 개요 93
5.8.2. CSI 정보 93

5.8.3. Manila CSI Driver Operator 제한 93
5.8.4. 동적으로 Manila CSI 볼륨 프로비저닝 94
5.9. RED HAT VIRTUALIZATION CSI DRIVER OPERATOR 95
5.9.1. 개요 95
5.9.2. CSI 정보 96
5.9.3. oVirt CSI 드라이버 스토리지 클래스 96
5.9.4. RHV에서 영구 볼륨 생성 97

6장. 영구 볼륨 확장 99
6.1. 볼륨 확장 지원 활성화 99
6.2. CSI 볼륨 확장 99
6.3. 지원되는 드라이버를 사용한 FLEXVOLUME 확장 99
6.4. 파일 시스템을 사용한 PVC(영구 볼륨 클레임) 처리 100
6.5. 볼륨을 분리할 때 실패에서 복구 101

7장. 동적 프로비저닝 102
7.1. 동적 프로비저닝 소개 102
7.2. 사용 가능한 동적 프로비저닝 플러그인 102
7.3. 스토리지 클래스 정의 103
7.3.1. 기본 StorageClass 개체 정의 103
7.3.2. 스토리지 클래스 주석 104
7.3.3. RHOSP Cinder 오브젝트 정의 105
7.3.4. RHOSP Manila CSI(Container Storage Interface) 오브젝트 정의 105
7.3.5. AWS Elastic Block Store (EBS) 객체 정의 105
7.3.6. Azure Disk 오브젝트 정의 106
7.3.7. Azure File 객체 정의 107
7.3.7.1. Azure File 사용시 고려 사항 108
7.3.8. GCE PersistentDisk (gcePD) 오브젝트 정의 109
7.3.9. VMware vSphere 오브젝트 정의 109
7.4. 기본 스토리지 클래스 변경 109

1장. OPENSHIFT CONTAINER PLATFORM 스토리지 개요

OpenShift Container Platform에서는 온프레미스 및 클라우드 공급자를 위해 여러 유형의 스토리지를 지원합니다. OpenShift Container Platform 클러스터에서 영구 및 비영구적 데이터의 컨테이너 스토리지를 관리할 수 있습니다.

1.1. 스토리지 유형

OpenShift Container Platform 스토리지는 임시 스토리지 및 영구 스토리지라는 두 가지 범주로 널리 분류됩니다.

1.1.1. 임시 스토리지

포드와 컨테이너는 본질적으로 임시 또는 일시적이며 상태 비저장 애플리케이션을 위해 설계되었습니다. 임시 스토리지를 사용하면 관리자와 개발자가 일부 작업을 위해 로컬 스토리지를 더 효율적으로 관리할 수 있습니다. 임시 스토리지 개요, 유형 및 관리에 대한 자세한 내용은 임시 스토리지 이해를 참조하십시오.

1.1.2. 영구 스토리지

컨테이너에 배포된 상태 저장 애플리케이션에는 영구 스토리지가 필요합니다. OpenShift Container Platform에서는 PV(영구 볼륨)라는 사전 프로비저닝된 스토리지 프레임워크를 사용하여 클러스터 관리자가 영구 스토리지를 프로비저닝할 수 있습니다. 이러한 볼륨 내부 데이터는 개별 포드의 라이프사이클 이후에 존재할 수 있습니다. 개발자는 PVC(영구 볼륨 클레임)를 사용하여 스토리지 요구 사항을 요청할 수 있습니다. 영구 스토리지 개요, 구성 및 라이프사이클에 대한 자세한 내용은 영구 스토리지 이해를 참조하십시오.

1.2. CSI(CONTAINER STORAGE INTERFACE)

CSI는 다양한 컨테이너 오케스트레이션(CO) 시스템에서 컨테이너 스토리지 관리를 위한 API 사양입니다. 기본 스토리지 인프라에 대한 특정 지식 없이도 컨테이너 네이티브 환경에서 스토리지 볼륨을 관리할 수 있습니다. CSI를 사용하면 사용 중인 스토리지 벤더에 관계없이 스토리지가 다양한 컨테이너 오케스트레이션 시스템에서 균일하게 작동합니다. CSI에 대한 자세한 내용은 CSI (Container Storage Interface) 사용을 참조하십시오.

1.3. 동적 프로비저닝

동적 프로비저닝을 사용하면 필요에 따라 스토리지 볼륨을 생성할 수 있으므로 클러스터 관리자가 스토리지를 사전 프로비저닝할 필요가 없습니다. 동적 프로비저닝에 대한 자세한 내용은 동적 프로비저닝을 참조하십시오.

2장. 임시 스토리지 이해

2.1. 개요

영구 스토리지 외에도 Pod 및 컨테이너는 작업을 위해 임시 또는 임시 로컬 스토리지가 필요할 수 있습니다. 이러한 임시 스토리지의 수명은 개별 Pod의 수명 이상으로 연장되지 않으며 이 임시 스토리지는 여러 Pod 사이에서 공유할 수 없습니다.

Pod는 스크래치 공간, 캐싱 및 로그를 위해 임시 로컬 스토리지를 사용합니다. 로컬 스토리지 회계 및 격리 부족과 관련한 문제는 다음과 같습니다.

- Pod는 사용할 수 있는 로컬 스토리지의 용량을 알 수 없습니다.
- Pod는 보장되는 로컬 스토리지를 요청할 수 없습니다.
- 로컬 스토리지는 최상의 노력 리소스입니다.
- Pod는 로컬 스토리지를 채우는 다른 Pod로 인해 제거할 수 있으며 충분한 스토리지를 회수할 때까지 새 Pod가 허용되지 않습니다.

영구 볼륨에 대해 임시 스토리지는 구조화되지 않으며 시스템에서 실행되는 모든 Pod와 시스템, 컨테이너 런타임 및 OpenShift Container Platform의 다른 사용에서 공간을 공유합니다. 임시 스토리지 프레임워크를 사용하면 Pod에서 일시적인 로컬 스토리지 요구를 지정할 수 있습니다. 또한, OpenShift Container Platform은 적절한 Pod를 예약하고 로컬 스토리지를 과도하게 사용하지 않도록 노드를 보호할 수 있습니다.

임시 스토리지 프레임워크를 사용하면 관리자와 개발자가 이 로컬 스토리지를 더욱 더 쉽게 관리할 수 있지만 I/O 처리량 및 대기 시간과 관련해서는 보장을 하지 않습니다.

2.2. 임시 스토리지 유형

임시 로컬 스토리지는 항상 기본 파티션에서 사용할 수 있습니다. 기본 파티션을 생성하는 방법에는 루트 및 런타임이라는 두 가지 기본적인 방법이 있습니다.

루트
이 파티션에는 기본적으로 kubelet 루트 디렉터리인 **/var/lib/kubelet/** 및 **/var/log/** 디렉터리가 있습니다. 이 파티션은 사용자 Pod, OS, Kubernetes 시스템 데몬 간에 공유할 수 있습니다. 이 파티션은 **EmptyDir** 볼륨, 컨테이너 로그, 이미지 계층 및 container-wriable 계층을 통해 Pod에서 사용할 수 있습니다. kubelet은 이 파티션의 공유 액세스 및 격리를 관리합니다. 이 파티션은 임시입니다. 애플리케이션은 이 파티션에서 디스크 IOPS와 같은 성능 SLA를 기대할 수 없습니다.

런타임
런타임에서 오버레이 파일 시스템에 사용할 수 있는 선택적 파티션입니다. OpenShift Container Platform에서는 이 파티션에 대한 격리와 함께 공유 액세스를 식별하고 제공합니다. 컨테이너 이미지 계층 및 쓰기 가능한 계층이 여기에 저장됩니다. 런타임 파티션이 있는 경우 **루트** 파티션은 이미지 계층 또는 기타 쓰기 가능한 스토리지를 유지하지 않습니다.

2.3. 임시 데이터 스토리지 관리

클러스터 관리자는 비종료 상태의 모든 Pod에서 임시 스토리지에 대한 제한 범위 및 요청 수를 정의하는 할당량을 설정하여 프로젝트 내에서 임시 스토리지를 관리할 수 있습니다. 개발자는 Pod 및 컨테이너 수준에서 이러한 컴퓨팅 리소스에 대한 요청 및 제한을 설정할 수도 있습니다.

2.4. 임시 스토리지 모니터링

/bin/df를 임시 컨테이너 데이터가 위치하는 볼륨에서 임시 스토리지의 사용을 모니터링하는 도구를 사용할 수 있으며, 이는 **/var/lib/kubelet** 및 **/var/lib/containers**입니다. 클러스터 관리자가 **/var/lib/containers**를 별도의 디스크에 배치한 경우에는 **df** 명령을 사용하여 **/var/lib/kubelet**에서 전용으로 사용할 수 있는 공간을 표시할 수 있습니다.

/var/lib에서 사용된 공간 및 사용할 수 있는 공간을 사람이 읽을 수 있는 값으로 표시하려면 다음 명령을 입력합니다.

```
$ df -h /var/lib
```

출력에는 **/var/lib**에서의 임시 스토리지 사용량이 표시됩니다.

출력 예

```
Filesystem  Size  Used Avail Use% Mounted on
/dev/sda1    69G   32G   34G  49% /
```

3장. 영구 스토리지 이해

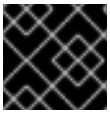
3.1. 영구 스토리지 개요

스토리지 관리는 컴퓨팅 리소스 관리와 다릅니다. OpenShift Container Platform에서는 Kubernetes PV(영구 볼륨) 프레임워크를 사용하여 클러스터 관리자가 클러스터의 영구 스토리지를 프로비저닝할 수 있습니다. 개발자는 PVC(영구 볼륨 클레임)를 사용하여 기본 스토리지 인프라를 구체적으로 잘 몰라도 PV 리소스를 요청할 수 있습니다.

PVC는 프로젝트별로 고유하며 PV를 사용하는 방법과 같이 개발자가 생성 및 사용할 수 있습니다. 자체 PV 리소스는 단일 프로젝트로 범위가 지정되지 않으며, 전체 OpenShift Container Platform 클러스터에서 공유되고 모든 프로젝트에서 요청할 수 있습니다. PV가 PVC에 바인딩된 후에는 해당 PV를 다른 PVC에 바인딩할 수 없습니다. 이는 바인딩 프로젝트인 단일 네임스페이스로 바인딩된 PV의 범위를 지정하는 효과가 있으며, 이는 바인딩된 프로젝트의 범위가 됩니다.

PV는 **PersistentVolume** API 오브젝트로 정의되면, 이는 클러스터 관리자가 정적으로 프로비저닝하거나 **StorageClass** 오브젝트를 사용하여 동적으로 프로비저닝한 클러스터에서의 기존 스토리지 조각을 나타냅니다. 그리고 노드가 클러스터 리소스인 것과 마찬가지로 클러스터의 리소스입니다.

PV는 **볼륨**과 같이 볼륨 플러그인이지만 PV를 사용하는 개별 Pod와 라이프사이클이 독립적입니다. PV 오브젝트는 NFS, iSCSI 또는 클라우드 공급자별 스토리지 시스템에서 스토리지 구현의 세부 정보를 캡처합니다.



중요

인프라의 스토리지의 고가용성은 기본 스토리지 공급자가 담당합니다.

PVC는 **PersistentVolumeClaim** API 오브젝트에 의해 정의되며, 개발자의 스토리지 요청을 나타냅니다. Pod는 노드 리소스를 사용하고 PVC는 PV 리소스를 사용하는 점에서 Pod와 유사합니다. 예를 들어, Pod는 CPU 및 메모리와 같은 특정 리소스를 요청할 수 있지만 PVC는 특정 스토리지 용량 및 액세스 모드를 요청할 수 있습니다. 예를 들어, Pod는 1회 읽기-쓰기 또는 여러 번 읽기 전용으로 마운트될 수 있습니다.

3.2. 볼륨 및 클레임의 라이프사이클

PV는 클러스터의 리소스입니다. PVC는 그러한 리소스에 대한 요청이며, 리소스에 대한 클레임을 검사하는 역할을 합니다. PV와 PVC 간의 상호 작용에는 다음과 같은 라이프사이클이 있습니다.

3.2.1. 스토리지 프로비저닝

PVC에 정의된 개발자의 요청에 대한 응답으로 클러스터 관리자는 스토리지 및 일치하는 PV를 프로비저닝하는 하나 이상의 동적 프로비저너를 구성합니다.

다른 방법으로 클러스터 관리자는 사용할 수 있는 실제 스토리지의 세부 정보를 전달하는 여러 PV를 사전에 생성할 수 있습니다. PV는 API에 위치하며 사용할 수 있습니다.

3.2.2. 클레임 바인딩

PVC를 생성할 때 스토리지의 특정 용량을 요청하고, 필요한 액세스 모드를 지정하며, 스토리지를 설명 및 분류하는 스토리지 클래스를 만듭니다. 마스터의 제어 루프는 새 PVC를 감시하고 새 PVC를 적절한 PV에 바인딩합니다. 적절한 PV가 없으면 스토리지 클래스를 위한 프로비저너가 PV를 1개 생성합니다.

전체 PV의 크기는 PVC 크기를 초과할 수 있습니다. 이는 특히 수동으로 프로비저닝된 PV의 경우 더욱 그러합니다. 초과를 최소화하기 위해 OpenShift Container Platform은 기타 모든 조건과 일치하는 최소 PV로 바인딩됩니다.

일치하는 볼륨이 없거나 스토리지 클래스에 서비스를 제공하는 사용할 수 있는 프로비저너로 생성할 수 없는 경우 클레임은 영구적으로 바인딩되지 않습니다. 일치하는 볼륨을 사용할 수 있을 때 클레임이 바인딩됩니다. 예를 들어, 수동으로 프로비저닝된 50Gi 볼륨이 있는 클러스터는 100Gi 요청하는 PVC와 일치하지 않습니다. 100Gi PV가 클러스터에 추가되면 PVC를 바인딩할 수 있습니다.

3.2.3. Pod 및 클레임된 PV 사용

Pod는 클레임을 볼륨으로 사용합니다. 클러스터는 클레임을 검사하여 바인딩된 볼륨을 찾고 Pod에 해당 볼륨을 마운트합니다. 여러 액세스 모드를 지원하는 그러한 볼륨의 경우 Pod에서 클레임을 볼륨으로 사용할 때 적용되는 모드를 지정해야 합니다.

클레임이 있고 해당 클레임이 바인딩되면, 바인딩된 PV는 필요한 동안 사용자의 소유가 됩니다. Pod의 볼륨 블록에 **persistentVolumeClaim**을 포함하여 Pod를 예약하고 클레임된 PV에 액세스할 수 있습니다.

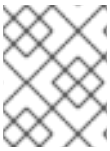
참고

파일이 많은 영구 볼륨을 Pod에 연결하면 해당 Pod가 실패하거나 시작하는 데 시간이 오래 걸릴 수 있습니다. 자세한 내용은 OpenShift에서 파일 수가 많은 영구 볼륨을 사용하는 경우 Pod를 시작하지 못하거나 과도한 시간을 차지하여 "Ready" 상태를 얻을 수 없는 이유는 무엇입니까?

3.2.4. 사용 중 스토리지 오브젝트 보호

사용 중 스토리지 오브젝트 보호 기능은 Pod에서 사용 중인 PVC와 PVC에 바인딩된 PC가 시스템에서 제거되지 않도록 합니다. 제거되면 데이터가 손실될 수 있습니다.

사용 중 스토리지 오브젝트 보호는 기본적으로 활성화됩니다.



참고

PVC를 사용하는 **Pod** 오브젝트가 존재하는 경우 PVC는 Pod에 의해 활성 사용 중이 됩니다.

사용자가 Pod에서 활성 사용 중인 PVC를 삭제하면 PVC가 즉시 제거되지 않습니다. 모든 Pod에서 PVC를 더 이상 활성 사용하지 않을 때까지 PVC의 제거가 연기됩니다. 또한, 클러스터 관리자가 PVC에 바인딩된 PV를 삭제하는 경우에도 PV가 즉시 제거되지 않습니다. PV가 더 이상 PVC에 바인딩되지 않을 때까지 PV 제거가 연기됩니다.

3.2.5. 영구 볼륨 해제

볼륨 사용 작업이 끝나면 API에서 PVC 오브젝트를 삭제하여 리소스를 회수할 수 있습니다. 클레임이 삭제되었지만 다른 클레임에서 아직 사용할 수 없을 때 볼륨은 해제된 것으로 간주됩니다. 이전 클레임의 데이터는 볼륨에 남아 있으며 정책에 따라 처리되어야 합니다.

3.2.6. 영구 볼륨 회수 정책

영구 볼륨 회수 정책은 해제된 볼륨에서 수행할 작업을 클러스터에 명령합니다. 볼륨 회수 정책은 **Retain**, **Recycle** 또는 **Delete**일 수 있습니다.

- **Retain** 회수 정책을 사용하면 이를 지원하는 해당 볼륨 플러그인에 대한 리소스를 수동으로 회수할 수 있습니다.
- **Recycle** 회수 정책은 볼륨이 클레임에서 해제되면 바인딩되지 않은 영구 볼륨 풀로 다시 재활용합니다.

중요

OpenShift Container Platform 4에서는 **Recycle** 회수 정책이 사용되지 않습니다. 기능을 향상하기 위해 동적 프로비저닝이 권장됩니다.

- **Delete** 회수 정책은 OpenShift Container Platform 및 외부 인프라(예: AWS EBS 또는 VMware vSphere)의 연결된 스토리지 자산 모두에서 **PersistentVolume** 오브젝트를 삭제합니다.

참고

동적으로 프로비저닝된 볼륨은 항상 삭제됩니다.

3.2.7. 수동으로 영구 볼륨 회수

PVC(영구 볼륨 클레임)가 삭제되어도 PV(영구 볼륨)는 계속 존재하며 "해제됨"으로 간주됩니다. 그러나 이전 클레임의 데이터가 볼륨에 남아 있으므로 다른 클레임에서 PV를 아직 사용할 수 없습니다.

절차

클러스터 관리자로 PV를 수동으로 회수하려면 다음을 수행합니다.

1. PV를 삭제합니다.

```
$ oc delete pv <pv-name>
```

   AWS EBS, GCE PD, Azure Disk 또는 Cinder 볼륨과 같은 외부 인프라의 연결된 스토리지 자산은 PV가 삭제된 후에도 계속 존재합니다.

2. 연결된 스토리지 자산에서 데이터를 정리합니다.
3. 연결된 스토리지 자산을 삭제합니다. 대안으로, 동일한 스토리지 자산을 재사용하려면, 스토리지 자산 정의를 사용하여 새 PV를 생성합니다.

이제 회수된 PV를 다른 PVC에서 사용할 수 있습니다.

3.2.8. 영구 볼륨의 회수 정책 변경

영구 볼륨의 회수 정책을 변경하려면 다음을 수행합니다.

1. 클러스터의 영구 볼륨을 나열합니다.

```
$ oc get pv
```

출력 예

```
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM          STORAGECLASS    REASON   AGE
 pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
```

```
default/claim1    manual                  10s
 pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO           Delete          Bound
default/claim2    manual                  6s
 pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO           Delete          Bound
default/claim3    manual                  3s
```

2. 영구 볼륨 중 하나를 선택하고 다음과 같이 회수 정책을 변경합니다.

```
$ oc patch pv <your-pv-name> -p '{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'
```

3. 선택한 영구 볼륨에 올바른 정책이 있는지 확인합니다.

```
$ oc get pv
```

출력 예

```
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM           STORAGECLASS    REASON   AGE
 pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO           Delete          Bound
default/claim1    manual                  10s
 pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO           Delete          Bound
default/claim2    manual                  6s
 pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO           Retain          Bound
default/claim3    manual                  3s
```

이전 출력에서 **default/claim3** 클레임에 바인딩된 볼륨이 이제 **Retain** 회수 정책을 갖습니다. 사용자가 **default/claim3** 클레임을 삭제할 때 볼륨이 자동으로 삭제되지 않습니다.

3.3. PV(영구 볼륨)

각 PV에는 **사양** 및 **상태**가 포함됩니다. 이는 볼륨의 사양과 상태이고 예는 다음과 같습니다.

PersistentVolume 오브젝트 정의 예

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 (1)
spec:
  capacity:
    storage: 5Gi (2)
  accessModes:
    - ReadWriteOnce (3)
  persistentVolumeReclaimPolicy: Retain (4)
  ...
status:
  ...
```

(1) 영구 볼륨의 이름입니다.

(2) 볼륨에서 사용할 수 있는 스토리지의 용량입니다.

(3) 읽기-쓰기 및 마운트 권한을 정의하는 액세스 모드입니다.

4 해제된 후 리소스를 처리하는 방법을 나타내는 회수 정책입니다.

3.3.1. PV 유형

OpenShift Container Platform에서 지원되는 영구 볼륨 플러그인은 다음과 같습니다.

- AWS Elastic Block Store (EBS)
- Azure Disk
- Azure File
- Cinder
- 파이버 채널
- GCE 영구 디스크
- HostPath
- iSCSI
- 로컬 볼륨
- NFS
- OpenStack Manila
- Red Hat OpenShift 컨테이너 스토리지
- VMware vSphere

3.3.2. 용량

일반적으로 PV(영구 볼륨)에는 특정 스토리지 용량이 있습니다. 이는 PV의 **용량** 속성을 사용하여 설정됩니다.

현재는 스토리지 용량이 설정 또는 요청할 수 있는 유일한 리소스뿐입니다. 향후 속성에는 IOPS, 처리량 등이 포함될 수 있습니다.

3.3.3. 액세스 모드

영구 볼륨은 리소스 공급자가 지원하는 방식으로 호스트에 볼륨을 마운트될 수 있습니다. 공급자에 따라 기능이 다르며 각 PV의 액세스 모드는 해당 볼륨에서 지원하는 특정 모드로 설정됩니다. 예를 들어, NFS에서는 여러 읽기-쓰기 클라이언트를 지원할 수 있지만 특정 NFS PV는 서버에서 읽기 전용으로 내보낼 수 있습니다. 각 PV는 특정 PV의 기능을 설명하는 자체 액세스 모드 세트를 가져옵니다.

클레임은 액세스 모드가 유사한 볼륨과 매칭됩니다. 유일하게 일치하는 두 가지 기준은 액세스 모드와 크기입니다. 클레임의 액세스 모드는 요청을 나타냅니다. 따라서 더 많이 부여될 수 있지만 절대로 부족하게는 부여되지 않습니다. 예를 들어, 클레임 요청이 RWO이지만 사용 가능한 유일한 볼륨이 NFS PV(RWO+ROX+RWX)인 경우, RWO를 지원하므로 클레임이 NFS와 일치하게 됩니다.

항상 직접 일치가 먼저 시도됩니다. 볼륨의 모드는 사용자의 요청과 일치하거나 더 많은 모드를 포함해야 합니다. 크기는 예상되는 크기보다 크거나 같아야 합니다. NFS 및 iSCSI와 같은 두 개의 볼륨 유형에 동일한 액세스 모드 세트가 있는 경우 둘 중 하나를 해당 모드와 클레임과 일치시킬 수 있습니다. 볼륨 유형과

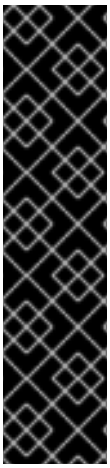
특정 유형을 선택할 수 있는 순서는 없습니다.

모드가 동일한 모든 볼륨이 그룹화된 후 크기 오름차순으로 크기가 정렬됩니다. 바인더는 모드가 일치하는 그룹을 가져온 후 크기가 일치하는 그룹을 찾을 때까지 크기 순서대로 각 그룹에 대해 반복합니다.

다음 표에는 액세스 모드가 나열되어 있습니다.

표 3.1. 액세스 모드

액세스 모드	CLI 약어	설명
ReadWriteOnce	**RWO**	볼륨은 단일 노드에서 읽기-쓰기로 마운트할 수 있습니다.
ReadOnlyMany	**ROX**	볼륨은 여러 노드에서 읽기 전용으로 마운트할 수 있습니다.
ReadWriteMany	**RWX**	볼륨은 여러 노드에서 읽기-쓰기로 마운트할 수 있습니다.



중요

볼륨 액세스 모드는 볼륨 기능에 대한 설명자입니다. 제한 조건이 적용되지 않습니다. 리소스를 잘못된 사용으로 인한 런타임 오류는 스토리지 공급자가 처리합니다.

예를 들어, NFS는 **ReadWriteOnce** 액세스 모드를 제공합니다. 볼륨의 ROX 기능을 사용하려면 클레임을 **읽기 전용**으로 표시해야 합니다. 공급자에서의 오류는 런타임 시 마운트 오류로 표시됩니다.

iSCSI 및 파이버 채널 볼륨은 현재 펜싱 메커니즘을 지원하지 않습니다. 볼륨을 한 번에 하나씩만 사용하는지 확인해야 합니다. 노드 드레이닝과 같은 특정 상황에서는 두 개의 노드에서 볼륨을 동시에 사용할 수 있습니다. 노드를 드레인하기 전에 먼저 이러한 볼륨을 사용하는 Pod가 삭제되었는지 확인합니다.

표 3.2. PV에서 지원되는 액세스 모드

볼륨 플러그인	ReadWriteOnce [1]	ReadOnlyMany	ReadWriteMany
AWS EBS [2]	▮	-	-
Azure File	▮	▮	▮
Azure Disk	▮	-	-
Cinder	▮	-	-
파이버 채널	▮	▮	-
GCE 영구 디스크	▮	-	-
HostPath	▮	-	-
iSCSI	▮	▮	-

볼륨 플러그인	ReadWriteOnce [1]	ReadOnlyMany	ReadWriteMany
로컬 볼륨	▮	-	-
NFS	▮	▮	▮
OpenStack Manila	-	-	▮
Red Hat OpenShift Container Storage	▮	-	▮
VMware vSphere	▮	-	-

1. ReadWriteOnce(RWO) 볼륨은 여러 노드에 마운트할 수 없습니다. 시스템이 이미 실패한 노드에 할당되어 있기 때문에, 노드가 실패하면 시스템은 연결된 RWO 볼륨을 새 노드에 마운트하는 것을 허용하지 않습니다. 이로 인해 다중 연결 오류 메시지가 표시되면 동적 영구 볼륨이 연결된 경우와 같이 중요한 워크로드에서의 데이터가 손실되는 것을 방지하기 위해 종료되거나 충돌이 발생한 노드에서 Pod를 삭제해야 합니다.

2. AWS EBS 기반 Pod에 재생성 배포 전략을 사용합니다.

3.3.4. 단계

볼륨은 다음 단계 중 하나에서 찾을 수 있습니다.

표 3.3. 볼륨 단계

단계	설명
Available	아직 클레임에 바인딩되지 않은 여유 리소스입니다.
Bound	볼륨이 클레임에 바인딩됩니다.
해제됨	클레임이 삭제되었지만, 리소스가 아직 클러스터에 의해 회수되지 않았습니다.
실패	볼륨에서 자동 회수가 실패했습니다.

다음을 실행하여 PV에 바인딩된 PVC의 이름을 확인할 수 있습니다.

```
$ oc get pv <pv-claim>
```

3.3.4.1. 마운트 옵션

mountOptions 속성을 사용하여 PV를 마운트하는 동안 마운트 옵션을 지정할 수 있습니다.

예를 들면 다음과 같습니다.

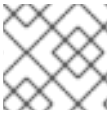
마운트 옵션 예

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  mountOptions: 1
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
  persistentVolumeReclaimPolicy: Retain
  claimRef:
    name: claim1
    namespace: default
```

1 지정된 마운트 옵션이 PV를 디스크에 마운트하는 동안 사용됩니다.

다음 PV 유형에서는 마운트 옵션을 지원합니다.

- AWS Elastic Block Store (EBS)
- Azure Disk
- Azure File
- Cinder
- GCE 영구 디스크
- iSCSI
- 로컬 볼륨
- NFS
- Red Hat OpenShift 컨테이너 스토리지(Ceph RBD 전용)
- VMware vSphere



참고

파이버 채널 및 HostPath PV는 마운트 옵션을 지원하지 않습니다.

3.4. 영구 볼륨 클레임

각 **PersistentVolumeClaim** 오브젝트에는 **spec** 및 **status**가 포함되며, 이는 PVC(영구 볼륨 클레임)의 사양과 상태이고, 예를 들면 다음과 같습니다.

PersistentVolumeClaim 오브젝트 정의 예

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim 1
spec:
  accessModes:
    - ReadWriteOnce 2
  resources:
    requests:
      storage: 8Gi 3
  storageClassName: gold 4
status:
  ...
```

1 PVC의 이름

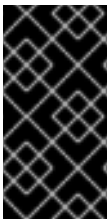
2 읽기-쓰기 및 마운트 권한을 정의하는 액세스 모드

3 PVC에서 사용할 수 있는 스토리지 용량

4 클레임에 필요한 **StorageClass**의 이름

3.4.1. 스토리지 클래스

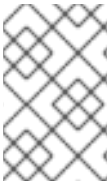
선택 사항으로 클레임은 **storageClassName** 속성에 스토리지 클래스의 이름을 지정하여 특정 스토리지 클래스를 요청할 수 있습니다. PVC와 **storageClassName**이 동일하고 요청된 클래스의 PV만 PVC에 바인딩할 수 있습니다. 클러스터 관리자는 동적 프로비저너를 구성하여 하나 이상의 스토리지 클래스에 서비스를 제공할 수 있습니다. 클러스터 관리자는 PVC의 사양과 일치하는 PV를 생성할 수 있습니다.



중요

클러스터 스토리지 작업자는 사용 중인 플랫폼에 따라 기본 스토리지 클래스를 설치할 수 있습니다. 이 스토리지 클래스는 Operator가 소유하고 제어합니다. 주석 및 레이블 정의 외에는 삭제하거나 변경할 수 없습니다. 다른 동작이 필요한 경우 사용자 정의 스토리지 클래스를 정의해야 합니다.

클러스터 관리자는 모든 PVC의 기본 스토리지 클래스도 설정할 수도 있습니다. 기본 스토리지 클래스가 구성된 경우 PVC는 ""로 설정된 **StorageClass** 또는 **storageClassName** 주석이 스토리지 클래스를 제외하고 PV에 바인딩되도록 명시적으로 요청해야 합니다.



참고

두 개 이상의 스토리지 클래스가 기본값으로 표시되면 **storageClassName**이 명시적으로 지정된 경우에만 PVC를 생성할 수 있습니다. 따라서 1개의 스토리지 클래스만 기본값으로 설정해야 합니다.

3.4.2. 액세스 모드

클레임은 특정 액세스 모드로 스토리지를 요청할 때 볼륨과 동일한 규칙을 사용합니다.

3.4.3. 리소스

Pod와 같은 클레임은 특정 리소스 수량을 요청할 수 있습니다. 이 경우 요청은 스토리지에 대한 요청입니다. 동일한 리소스 모델이 볼륨 및 클레임에 적용됩니다.

3.4.4. 클레임을 볼륨으로

클레임을 볼륨으로 사용하여 Pod 액세스 스토리지 클레임은 클레임을 사용하는 Pod와 동일한 네임스페이스에 있어야 합니다. 클러스터는 Pod의 네임스페이스에서 클레임을 검색하고 이를 사용하여 클레임을 지원하는 **PersistentVolume**을 가져옵니다. 볼륨은 호스트에 마운트되며, 예를 들면 다음과 같습니다.

호스트 및 Pod에 볼륨 마운트 예

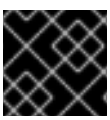
```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: dockerfile/nginx
      volumeMounts:
      - mountPath: "/var/www/html" 1
        name: mypd 2
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim 3
```

1. Pod 내부에 볼륨을 마운트하는 경로입니다.
2. 마운트할 볼륨의 이름입니다. 컨테이너 루트, / 또는 호스트와 컨테이너에서 동일한 경로에 마운트하지 마십시오. 컨테이너가 호스트 **/dev/pts** 파일과 같이 충분한 권한이 있는 경우 호스트 시스템이 손상될 수 있습니다. **/host**를 사용하여 호스트를 마운트하는 것이 안전합니다.
3. 동일한 네임스페이스에 있는 사용할 PVC의 이름입니다.

3.5. 블록 볼륨 지원

OpenShift Container Platform은 원시 블록 볼륨을 정적으로 프로비저닝할 수 있습니다. 이러한 볼륨에는 파일 시스템이 없으며 디스크에 직접 쓰거나 자체 스토리지 서비스를 구현하는 애플리케이션에 성능 이점을 제공할 수 있습니다.

원시 블록 볼륨은 **volumeMode를 지정하여 프로비저닝됩니다. PV 및 PVC 사양에서 block**



중요

권한이 부여된 컨테이너를 허용하려면 원시 블록 볼륨을 사용하는 Pod를 구성해야 합니다.

다음 표는 블록 볼륨을 지원하는 볼륨 플러그인을 보여줍니다.

표 3.4. 블록 볼륨 지원

볼륨 플러그인	수동 프로비저닝	동적 프로비저닝	모두 지원됨
AWS EBS	▮	▮	▮
Azure Disk	▮	▮	▮
Azure File			
Cinder	▮	▮	
파이버 채널	▮		▮
GCP	▮	▮	▮
HostPath			
iSCSI	▮		▮
로컬 볼륨	▮		▮
NFS			
Red Hat OpenShift 컨테이너 스토리지	▮	▮	▮
VMware vSphere	▮	▮	▮

참고

수동으로 프로비저닝할 수 있지만 완전하게 지원되지 않는 블록 볼륨은 기술 프리뷰 기능으로만 포함됩니다. Technology Preview 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 https://access.redhat.com/support/offerings/techpreview/를 참조하십시오.

3.5.1. 블록 볼륨 예

PV 예

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
```

```
    - ReadWriteOnce
  volumeMode: Block (1)
  persistentVolumeReclaimPolicy: Retain
  fc:
    targetWWNs: ["50060e801049cfd1"]
    lun: 0
    readOnly: false
```

(1) 이 PV가 원시 블록 볼륨임을 나타내려면 **volumeMode**를 **Block**으로 설정해야 합니다.

PVC 예

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block (1)
  resources:
    requests:
      storage: 10Gi
```

(1) 원시 블록 PVC가 요청되었음을 나타내려면 **volumeMode**를 **Block**으로 설정해야 합니다.

Pod 사양 예

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-block-volume
spec:
  containers:
    - name: fc-container
      image: fedora:26
      command: ["/bin/sh", "-c"]
      args: [ "tail -f /dev/null" ]
      volumeDevices: (1)
        - name: data
          devicePath: /dev/xvda (2)
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: block-pvc (3)
```

(1) 블록 장치에서는 **volumeMounts** 대신 **volumeDevices**가 사용됩니다. **PersistentVolumeClaim** 소스만 원시 블록 볼륨과 함께 사용할 수 있습니다.

(2) **mountPath** 대신 **devicePath**가 원시 블록이 시스템에 매핑되는 물리 장치의 경로를 나타냅니다.

(3) 볼륨 소스는 **persistentVolumeClaim** 유형이어야 하며 예상되는 PVC의 이름과 일치해야 합니다.

표 3.5. volumeMode에 대해 허용되는 값

값	기본
파일 시스템	예
블록	아니요

표 3.6. 블록 볼륨에 대한 바인딩 시나리오

PV volumeMode	PVC volumeMode	바인딩 결과
파일 시스템	파일 시스템	바인딩
지정되지 않음	지정되지 않음	바인딩
파일 시스템	지정되지 않음	바인딩
지정되지 않음	파일 시스템	바인딩
블록	블록	바인딩
지정되지 않음	블록	바인딩되지 않음
블록	지정되지 않음	바인딩되지 않음
파일 시스템	블록	바인딩되지 않음
블록	파일 시스템	바인딩되지 않음

중요

값을 지정하지 않으면 **Filesystem**의 기본값이 사용됩니다.

4장. 영구 스토리지 구성

4.1. AWS ELASTIC BLOCK STORE를 사용하는 영구저장장치

OpenShift Container Platform은 AWS EBS(Elastic Block Store volume)를 지원합니다. Amazon EC2를 사용하여 영구 스토리지로 OpenShift Container Platform 클러스터를 프로비저닝할 수 있습니다. Kubernetes 및 AWS에 대해 어느 정도 익숙한 것으로 가정합니다.

Kubernetes 영구 볼륨 프레임워크를 사용하면 관리자는 영구 스토리지로 클러스터를 프로비저닝하고 사용자가 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다. AWS Elastic Block Store 볼륨은 동적으로 프로비저닝할 수 있습니다. 영구 볼륨은 단일 프로젝트 또는 네임스페이스에 바인딩되지 않으며, OpenShift Container Platform 클러스터에서 공유할 수 있습니다. 영구 볼륨 클레임은 프로젝트 또는 네임스페이스에 고유하며 사용자가 요청할 수 있습니다.

중요

인프라의 스토리지의 고가용성은 기본 스토리지 공급자가 담당합니다.

4.1.1. 추가 리소스

- In-tree 볼륨 플러그인에서 사용할 수 없는 볼륨 스냅샷과 같은 추가 스토리지 옵션에 대한 정보는 AWS Elastic Block Store CSI Driver Operator 를 참조하십시오.

4.1.2. EBS 스토리지 클래스 생성

스토리지 클래스는 스토리지 수준 및 사용량을 구분하고 조정하는 데 사용됩니다. 스토리지 클래스를 정의하면 사용자는 동적으로 프로비저닝된 영구 볼륨을 얻을 수 있습니다.

절차

1. OpenShift Container Platform 콘솔에서 **스토리지 → 스토리지 클래스**를 클릭합니다.
2. 스토리지 클래스 개요에서 **스토리지 클래스 만들기**를 클릭합니다.
3. 표시되는 페이지에 원하는 옵션을 정의합니다.
   a. 스토리지 클래스를 참조할 이름을 입력합니다.
   b. 선택적 설명을 입력합니다.
   c. 회수 정책을 선택합니다.
   d. 드롭다운 목록에서 **kubernetes.io/aws-ebs**를 선택합니다.

참고

동등한 CSI 드라이버를 사용하여 스토리지 클래스를 생성하려면 드롭다운 목록에서 **ebs.csi.aws.com**을 선택합니다. 자세한 내용은 *AWS Elastic Block Store CSI Driver Operator* 에서 참조하십시오.

   e. 원하는 대로 스토리지 클래스에 대한 추가 매개변수를 입력합니다.
4. **생성**을 클릭하여 스토리지 클래스를 생성합니다.

4.1.3. 영구 볼륨 클레임 생성

사전 요구 사항

OpenShift Container Platform에서 볼륨으로 마운트하기 전에 기본 인프라에 스토리지가 있어야 합니다.

절차

1. OpenShift Container Platform 콘솔에서 **스토리지 → 영구 볼륨 클레임**을 클릭합니다.
2. 영구 볼륨 클레임 생성 개요에서 **영구 볼륨 클레임 생성**을 클릭합니다.
3. 표시되는 페이지에 원하는 옵션을 정의합니다.
   a. 드롭다운 메뉴에서 이전에 생성한 스토리지 클래스를 선택합니다.
   b. 스토리지 클레임의 고유한 이름을 입력합니다.
   c. 액세스 모드를 선택합니다. 이렇게 하면 생성된 스토리지 클레임에 대한 읽기 및 쓰기 액세스가 결정됩니다.
   d. 스토리지 클레임의 크기를 정의합니다.
4. **만들기**를 클릭하여 영구 볼륨 클레임을 생성하고 영구 볼륨을 생성합니다.

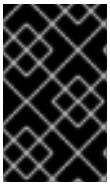
4.1.4. 볼륨 형식

OpenShift Container Platform이 볼륨을 마운트하고 컨테이너에 전달하기 전에 영구 볼륨 정의에 **fsType** 매개변수에 의해 지정된 파일 시스템이 포함되어 있는지 확인합니다. 장치가 파일 시스템으로 포맷되지 않으면 장치의 모든 데이터가 삭제되고 장치는 지정된 파일 시스템에서 자동으로 포맷됩니다.

이를 통해 OpenShift Container Platform이 처음 사용하기 전에 포맷되기 때문에 형식화되지 않은 AWS 볼륨을 영구 볼륨으로 사용할 수 있습니다.

4.1.5. 노드의 최대 EBS 볼륨 수

기본적으로 OpenShift Container Platform은 노드 1개에 연결된 최대 39개의 EBS 볼륨을 지원합니다. 이 제한은 AWS 볼륨 제한과 일치합니다. 볼륨 제한은 인스턴스 유형에 따라 다릅니다.



중요

클러스터 관리자는 in-tree 또는 CSI(Container Storage Interface) 볼륨과 해당 스토리지 클래스를 사용할 수 있지만, 두 볼륨을 동시에 사용하지 않아야 합니다. 연결된 최대 EBS 볼륨 수는 in-tree 및 CSI 볼륨에 대해 별도로 계산됩니다.

4.1.6. 추가 리소스

- in-tree 볼륨 플러그인에서 사용할 수 없는 볼륨 스냅샷과 같은 추가 스토리지 옵션에 액세스하는 방법에 대한 정보는 AWS Elastic Block Store CSI Driver Operator 를 참조하십시오.

4.2. AZURE를 사용하는 영구 스토리지

OpenShift Container Platform은 Microsoft Azure Disk 볼륨을 지원합니다. Azure를 사용하여 영구 스토리지로 OpenShift Container Platform 클러스터를 프로비저닝할 수 있습니다. Kubernetes 및 Azure에 대해 어느 정도 익숙한 것으로 가정합니다. Kubernetes 영구 볼륨 프레임워크를 사용하면 관리자는 영구 스토

리지로 클러스터를 프로비저닝하고 사용자가 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다. Azure 디스크 볼륨은 동적으로 프로비저닝할 수 있습니다. 영구 볼륨은 단일 프로젝트 또는 네임스페이스에 바인딩되지 않으며, OpenShift Container Platform 클러스터에서 공유할 수 있습니다. 영구 볼륨 클레임은 프로젝트 또는 네임스페이스에 고유하며 사용자가 요청할 수 있습니다.

중요

인프라의 스토리지의 고가용성은 기본 스토리지 공급자가 담당합니다.

추가 리소스

- Microsoft Azure Disk

4.2.1. Azure 스토리지 클래스 생성

스토리지 클래스는 스토리지 수준 및 사용량을 구분하고 조정하는 데 사용됩니다. 스토리지 클래스를 정의하면 사용자는 동적으로 프로비저닝된 영구 볼륨을 얻을 수 있습니다.

절차

1. OpenShift Container Platform 콘솔에서 **스토리지 → 스토리지 클래스**를 클릭합니다.
2. 스토리지 클래스 개요에서 **스토리지 클래스 만들기**를 클릭합니다.
3. 표시되는 페이지에 원하는 옵션을 정의합니다.
   a. 스토리지 클래스를 참조할 이름을 입력합니다.
   b. 선택적 설명을 입력합니다.
   c. 회수 정책을 선택합니다.
   d. 드롭다운 목록에서 **kubernetes.io/azure-disk**를 선택합니다.
      i. 스토리지 계정 유형을 입력합니다. 이는 Azure 스토리지 계정 SKU 계층에 해당합니다. 유효한 옵션은 **Premium_LRS**, **Standard_LRS**, **StandardSSD_LRS** 및 **UltraSSD_LRS**입니다.
      ii. 계정 종류를 입력합니다. 유효한 옵션은 **shared**, **dedicated** 및 **managed**입니다.

중요

Red Hat은 **kind 사용을 지원합니다. 스토리지 클래스에서 managed** 입니다.

Shared 및 **Dedicated**를 사용하여 Azure는 관리되지 않은 디스크를 생성합니다. 반면 OpenShift Container Platform은 머신 OS(root) 디스크의 관리 디스크를 생성합니다. Azure Disk는 노드에서 관리 및 관리되지 않은 디스크를 모두 사용하도록 허용하지 않으므로 **Shared** 또는 **Dedicated**로 생성된 관리되지 않은 디스크를 OpenShift Container Platform 노드에 연결할 수 없습니다.

   e. 원하는 대로 스토리지 클래스에 대한 추가 매개변수를 입력합니다.
4. **생성**을 클릭하여 스토리지 클래스를 생성합니다.

추가 리소스

- Azure Disk 스토리지 클래스

4.2.2. 영구 볼륨 클레임 생성

사전 요구 사항

OpenShift Container Platform에서 볼륨으로 마운트하기 전에 기본 인프라에 스토리지가 있어야 합니다.

절차

1. OpenShift Container Platform 콘솔에서 **스토리지 → 영구 볼륨 클레임**을 클릭합니다.
2. 영구 볼륨 클레임 생성 개요에서 **영구 볼륨 클레임 생성**을 클릭합니다.
3. 표시되는 페이지에 원하는 옵션을 정의합니다.
   a. 드롭다운 메뉴에서 이전에 생성한 스토리지 클래스를 선택합니다.
   b. 스토리지 클레임의 고유한 이름을 입력합니다.
   c. 액세스 모드를 선택합니다. 이렇게 하면 생성된 스토리지 클레임에 대한 읽기 및 쓰기 액세스가 결정됩니다.
   d. 스토리지 클레임의 크기를 정의합니다.
4. **만들기**를 클릭하여 영구 볼륨 클레임을 생성하고 영구 볼륨을 생성합니다.

4.2.3. 볼륨 형식

OpenShift Container Platform이 볼륨을 마운트하고 컨테이너에 전달하기 전에 영구 볼륨 정의에 **fsType** 매개변수에 의해 지정된 파일 시스템이 포함되어 있는지 확인합니다. 장치가 파일 시스템으로 포맷되지 않으면 장치의 모든 데이터가 삭제되고 장치는 지정된 파일 시스템에서 자동으로 포맷됩니다.

이를 통해 OpenShift Container Platform이 처음 사용하기 전에 포맷되기 때문에 형식화되지 않은 Azure 볼륨을 영구 볼륨으로 사용할 수 있습니다.

4.3. AZURE FILE을 사용하는 영구 스토리지

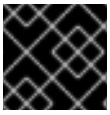
OpenShift Container Platform은 Microsoft Azure File 볼륨을 지원합니다. Azure를 사용하여 영구 스토리지로 OpenShift Container Platform 클러스터를 프로비저닝할 수 있습니다. Kubernetes 및 Azure에 대해 어느 정도 익숙한 것으로 가정합니다.

Kubernetes 영구 볼륨 프레임워크를 사용하면 관리자는 영구 스토리지로 클러스터를 프로비저닝하고 사용자가 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다. Azure File 볼륨을 동적으로 프로비저닝할 수 있습니다.

영구 볼륨은 단일 프로젝트 또는 네임스페이스에 바인딩되지 않으며 OpenShift Container Platform 클러스터에서 공유할 수 있습니다. 영구 볼륨 클레임은 프로젝트 또는 네임스페이스에 고유하며 사용자가 애플리케이션에서 사용하도록 요청할 수 있습니다.

중요

인프라의 스토리지의 고가용성은 기본 스토리지 공급자가 담당합니다.



중요

Azure File 볼륨은 서버 메시지 블록을 사용합니다.

추가 리소스

- Azure File

4.3.1. Azure File 공유 영구 볼륨 클레임 생성

영구 볼륨 클레임을 생성하려면 먼저 Azure 계정 및 키가 포함된 **Secret** 오브젝트를 정의해야 합니다. 이 시크릿은 **PersistentVolume** 정의에 사용되며 애플리케이션에서 사용하기 위해 영구 볼륨 클레임에 의해 참조됩니다.

사전 요구 사항

- Azure File 공유가 있습니다.
- 이 공유에 액세스할 수 있는 인증 정보(특히 스토리지 계정 및 키)를 사용할 수 있습니다.

절차

1. Azure File 인증 정보가 포함된 **Secret** 오브젝트를 생성합니다.

```
$ oc create secret generic <secret-name> --from-literal=azurestorageaccountname=
<storage-account> \ 1
 --from-literal=azurestorageaccountkey=<storage-account-key> 2
```

1 Azure File 스토리지 계정 이름입니다.

2 Azure File 스토리지 계정 키입니다.

2. 생성한 **Secret** 오브젝트를 참조하는 **PersistentVolume** 오브젝트를 생성합니다.

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
 name: "pv0001" 1
spec:
 capacity:
  storage: "5Gi" 2
 accessModes:
  - "ReadWriteOnce"
 storageClassName: azure-file-sc
 azureFile:
  secretName: <secret-name> 3
  shareName: share-1 4
  readOnly: false
```

1 영구 볼륨의 이름입니다.

2 이 영구 볼륨의 크기입니다.

3 Azure File에서 인증 정보를 공유하는 시크릿의 이름입니다.

4 Azure File 공유의 이름입니다.

3. 생성한 영구 볼륨에 매핑되는 **PersistentVolumeClaim** 오브젝트를 생성합니다.

```
apiVersion: "v1"
kind: "PersistentVolumeClaim"
metadata:
  name: "claim1" 1
spec:
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
      storage: "5Gi" 2
  storageClassName: azure-file-sc 3
  volumeName: "pv0001" 4
```

1 영구 볼륨 클레임의 이름입니다.

2 이 영구 볼륨 클레임의 크기입니다.

3 영구 볼륨을 프로비저닝하는 데 사용되는 스토리지 클래스의 이름입니다. **PersistentVolume** 정의에 사용되는 스토리지 클래스를 지정합니다.

4 Azure File 공유를 참조하는 기존 **PersistentVolume** 오브젝트의 이름입니다.

4.3.2. Pod에서 Azure 파일 공유 마운트

영구 볼륨 클레임을 생성한 후 애플리케이션에 의해 내부에서 사용될 수 있습니다. 다음 예시는 Pod 내부에서 이 공유를 마운트하는 방법을 보여줍니다.

사전 요구 사항

- 기본 Azure File 공유에 매핑된 영구 볼륨 클레임이 있습니다.

절차

- 기존 영구 볼륨 클레임을 마운트하는 Pod를 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name 1
spec:
  containers:
    ...
    volumeMounts:
    - mountPath: "/data" 2
      name: azure-file-share
  volumes:
```

```
  - name: azure-file-share
    persistentVolumeClaim:
      claimName: claim1 3
```

1. Pod의 이름입니다.
2. Pod 내부에서 Azure 파일 공유를 마운트하기 위한 경로입니다. 컨테이너 루트, / 또는 호스트와 컨테이너에서 동일한 경로에 마운트하지 마십시오. 컨테이너가 호스트 **/dev/pts** 파일과 같이 충분한 권한이 있는 경우 호스트 시스템이 손상될 수 있습니다. **/host**를 사용하여 호스트를 마운트하는 것이 안전합니다.
3. 이전에 생성된 **PersistentVolumeClaim** 오브젝트의 이름입니다.

4.4. CINDER를 사용하는 영구 스토리지

OpenShift Container Platform은 OpenStack Cinder를 지원합니다. Kubernetes 및 OpenStack에 대해 어느 정도 익숙한 것으로 가정합니다.

Cinder 볼륨은 동적으로 프로비저닝할 수 있습니다. 영구 볼륨은 단일 프로젝트 또는 네임스페이스에 바인딩되지 않으며, OpenShift Container Platform 클러스터에서 공유할 수 있습니다. 영구 볼륨 클레임은 프로젝트 또는 네임스페이스에 고유하며 사용자가 요청할 수 있습니다.

추가 리소스

- OpenStack Block Storage가 가상 하드 드라이브의 영구 블록 스토리지 관리를 제공하는 방법에 대한 자세한 내용은 OpenStack Cinder를 참조하십시오.

4.4.1. Cinder를 사용한 수동 프로비저닝

OpenShift Container Platform에서 볼륨으로 마운트하기 전에 기본 인프라에 스토리지가 있어야 합니다.

사전 요구 사항

- RHOSP(Red Hat OpenStack Platform)용으로 구성된 OpenShift Container Platform
- Cinder 볼륨 ID

4.4.1.1. 영구 볼륨 생성

OpenShift Container Platform에서 생성하기 전에 오브젝트 정의에서 PV(영구 볼륨)를 정의해야 합니다.

절차

1. 오브젝트 정의를 파일에 저장합니다.

cinder-persistentvolume.yaml

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
```

```
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  cinder: 3
    fsType: "ext3" 4
    volumeID: "f37a03aa-6212-4c62-a805-9ce139fab180" 5
```

1 영구 볼륨 클레임 또는 Pod에서 사용되는 볼륨의 이름입니다.

2 이 볼륨에 할당된 스토리지의 용량입니다.

3 RHOSP(Red Hat OpenStack Platform) Cinder 볼륨용 **cinder**를 나타냅니다.

4 볼륨이 처음 마운트될 때 생성되는 파일 시스템입니다.

5 사용할 Cinder 볼륨입니다.

중요

볼륨이 포맷되어 프로비저닝된 후에는 **fstype** 매개변수 값을 변경하지 마십시오. 이 값을 변경하면 데이터가 손실되고 Pod 오류가 발생할 수 있습니다.

2. 이전 단계에서 저장한 오브젝트 정의 파일을 생성합니다.

```
$ oc create -f cinder-persistentvolume.yaml
```

4.4.1.2. 영구 볼륨 포맷

OpenShift Container Platform은 처음 사용하기 전에 형식화되기 때문에 형식화되지 않은 Cinder 볼륨을 PV로 사용할 수 있습니다.

OpenShift Container Platform이 볼륨을 마운트하고 컨테이너에 전달하기 전, 시스템은 PV 정의에 **fsType** 매개변수에 의해 지정된 파일 시스템이 포함되어 있는지 확인합니다. 장치가 파일 시스템으로 포맷되지 않으면 장치의 모든 데이터가 삭제되고 장치는 지정된 파일 시스템에서 자동으로 포맷됩니다.

4.4.1.3. Cinder 볼륨 보안

애플리케이션에서 Cinder PV를 사용하는 경우 배포 구성에 대한 보안을 구성합니다.

사전 요구 사항

- 적절한 **fsGroup** 전략을 사용하는 SCC를 생성해야 합니다.

절차

1. 서비스 계정을 생성하고 SCC에 추가합니다.

```
$ oc create serviceaccount <service_account>
```

```
$ oc adm policy add-scc-to-user <new_scc> -z <service_account> -n <project>
```

2. 애플리케이션 배포 구성에서 서비스 계정 이름과 **securityContext**를 입력합니다.

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 (1)
  selector: (2)
    name: frontend
  template: (3)
    metadata:
      labels: (4)
        name: frontend (5)
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
      serviceAccountName: <service_account> (6)
      securityContext:
        fsGroup: 7777 (7)
```

(1) 실행할 Pod의 사본 수입니다.

(2) 실행할 Pod의 레이블 선택기입니다.

(3) 컨트롤러가 생성하는 Pod용 템플릿입니다.

(4) Pod의 레이블입니다. 선택기 레이블에서 제공되는 레이블이 포함되어야 합니다.

(5) 매개변수를 확장한 후 최대 이름 길이는 63자입니다.

(6) 생성한 서비스 계정을 지정합니다.

(7) Pod에 대한 **fsGroup**을 지정합니다.

4.5. 파이버 채널을 사용하는 영구 스토리지

OpenShift Container Platform은 파이버 채널을 지원하므로 파이버 채널 볼륨을 사용하여 영구 스토리지로 OpenShift Container Platform 클러스터를 프로비저닝할 수 있습니다. Kubernetes 및 Fibre 채널에 대해 어느 정도 익숙한 것으로 가정합니다.

Kubernetes 영구 볼륨 프레임워크를 사용하면 관리자는 영구 스토리지로 클러스터를 프로비저닝하고 사용자가 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다. 영구 볼륨은 단일 프로젝트 또는 네임스페이스에 바인딩되지 않으며, OpenShift Container Platform 클러스터에서 공유할 수 있습니다. 영구 볼륨 클레임은 프로젝트 또는 네임스페이스에 고유하며 사용자가 요청할 수 있습니다.

중요

인프라의 스토리지의 고가용성은 기본 스토리지 공급자가 담당합니다.

추가 리소스

- 파이버 채널 장치 사용

4.5.1. 프로비저닝

PersistentVolume API를 사용하여 파이버 채널 볼륨을 프로비저닝하려면 다음을 사용할 수 있어야 합니다.

- **targetWWN**(파이버 채널 대상의 World Wide Names에 대한 배열).
- 유효한 LUN 번호입니다.
- 파일 시스템 유형입니다.

영구 볼륨과 LUN은 일대일 매핑됩니다.

사전 요구 사항

- 파이버 채널 LUN은 기본 인프라에 있어야 합니다.

PersistentVolume 오브젝트 정의

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  fc:
    wwids: [scsi-3600508b400105e210000900000490000] 1
    targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5'] 2
    lun: 2 3
    fsType: ext4
```

1 WWID(WWID) FC **wwids** 또는 FC **targetWNs** and **lun** 의 조합을 설정해야 하지만 둘 다 동시에 설정해야 합니다. FC WWID 식별자는 모든 스토리지 장치에서 고유하며 장치에 액세스하는 데 사용되는 경로와는 독립적이므로 WWNs 대상을 통해 권장됩니다. WWID 식별자는 SCSI Inquiry를 발행하여 장치 식별 Vital Product Data (**페이지 0x83) 또는 단위 일련 번호 (페이지 0x80**)를 검색하여 얻을 수 있습니다. FC WWID는 장치 경로가 변경되거나 다른 시스템의 장치에 액세스하는 경우에도 디스크의 데이터를 참조하기 위해 **/dev/disk/by-id/** 로 식별됩니다.

2 3 파이버 채널 WWN은 **/dev/disk/by-path/pci-<IDENTIFIER>-fc-0x<WWN>-lun-<LUN#>**로 식별되지만, 앞에 **0x**를 포함한 **WWN**이 오고 이후에 **-**(하이픈)이 포함된 다른 경로가 있는 경로의 일부를 입력할 필요가 없습니다.

중요

볼륨이 포맷되고 프로비저닝된 후 **fstype** 매개변수 값을 변경하면 데이터가 손실되고 Pod 오류가 발생할 수 있습니다.

4.5.1.1. 디스크 할당량 강제 적용

LUN 파티션을 사용하여 디스크 할당량 및 크기 제약 조건을 강제 적용합니다. 각 LUN은 단일 영구 볼륨에 매핑되며 고유한 이름을 영구 볼륨에 사용해야 합니다.

이렇게 하면 최종 사용자가 10Gi와 같은 특정 용량에 의해 영구 스토리지를 요청하고 해당 볼륨과 동등한 용량과 일치시킬 수 있습니다.

4.5.1.2. 파이버 채널 볼륨 보안

사용자는 영구 볼륨 클레임을 사용하여 스토리지를 요청합니다. 이 클레임은 사용자의 네임스페이스에만 존재하며, 동일한 네임스페이스 내의 Pod에서만 참조할 수 있습니다. 네임스페이스에서 영구 볼륨에 대한 액세스를 시도하면 Pod가 실패하게 됩니다.

각 파이버 채널 LUN은 클러스터의 모든 노드에서 액세스할 수 있어야 합니다.

4.6. FLEXVOLUME을 사용하는 영구 스토리지

OpenShift Container Platform은 실행 가능한 모델을 드라이버와 인터페이스에 사용하는 out-of-tree 플러그인인 FlexVolume을 지원합니다.

플러그인이 기본 제공되지 않는 백엔드의 스토리지를 사용하려면, FlexVolume 드라이버를 통해 OpenShift Container Platform을 확장하고 애플리케이션에 영구 스토리지를 제공할 수 있습니다.

Pod는 **flexvolume** in-tree 플러그인을 통해 FlexVolume 드라이버와 상호 작용합니다.

추가 리소스

- 영구 볼륨 확장

4.6.1. FlexVolume 드라이버 정보

FlexVolume 드라이버는 클러스터의 모든 노드의 올바르게 정의된 디렉터리에 위치하는 실행 파일입니다. **flexVolume**을 갖는 **PersistentVolume** 오브젝트가 소스로 표시되는 볼륨을 마운트하거나 마운트 해제해야 할 때마다 OpenShift Container Platform이 FlexVolume 드라이버를 호출합니다.

중요

FlexVolume용 OpenShift Container Platform에서는 연결 및 분리 작업이 지원되지 않습니다.

4.6.2. FlexVolume 드라이버 예

FlexVolume 드라이버의 첫 번째 명령줄 인수는 항상 작업 이름입니다. 다른 매개 변수는 각 작업에 따라 다릅니다. 대부분의 작업에서는 JSON(JavaScript Object Notation) 문자열을 매개변수로 사용합니다. 이 매개변수는 전체 JSON 문자열이며 JSON 데이터가 있는 파일 이름은 아닙니다.

FlexVolume 드라이버에는 다음이 포함됩니다.

- 모든 **flexVolume.options**.
- **fsType** 및 **readwrite**와 같은 **kubernetes.io/** 접두사가 붙은 **flexVolume**의 일부 옵션.
- 설정된 경우, **kubernetes.io/secret/**이 접두사로 사용되는 참조된 시크릿의 콘텐츠

FlexVolume 드라이버 JSON 입력 예

```
{
 "fooServer": "192.168.0.1:1234", (1)
       "fooVolumeName": "bar",
 "kubernetes.io/fsType": "ext4", (2)
 "kubernetes.io/readwrite": "ro", (3)
 "kubernetes.io/secret/<key name>": "<key value>", (4)
 "kubernetes.io/secret/<another key name>": "<another key value>",
}
```

(1) **flexVolume.options**의 모든 옵션.

(2) **flexVolume.fsType**의 값.

(3) **flexVolume.readOnly**에 따른 **Ro**/**rw**.

(4) **flexVolume.secretRef**에서 참조하는 시크릿의 모든 키 및 해당 값.

OpenShift Container Platform은 드라이버의 표준 출력에서 JSON 데이터를 예상합니다. 지정하지 않으면, 출력이 작업 결과를 설명합니다.

FlexVolume 드라이버 기본 출력 예

```
{
 "status": "<Success/Failure/Not supported>",
 "message": "<Reason for success/failure>"
}
```

드라이버의 종료 코드는 성공의 경우 **0**이고 오류의 경우 **1**이어야 합니다.

작업은 idempotent여야 합니다. 즉, 이미 마운트된 볼륨의 마운트의 경우 작업이 성공적으로 수행되어야 합니다.

4.6.3. FlexVolume 드라이버 설치

OpenShift Container Platform 확장에 사용되는 FlexVolume 드라이버는 노드에서만 실행됩니다. FlexVolumes를 구현하려면 호출할 작업 목록 및 설치 경로만 있으면 됩니다.

사전 요구 사항

- FlexVolume 드라이버는 다음 작업을 구현해야 합니다.

  init

  드라이버를 초기화합니다. 이는 모든 노드를 초기화하는 동안 호출됩니다.

  - 인수: 없음
  - 실행 위치: 노드
  - 예상 출력: 기본 JSON

  Mount

디렉터리에 볼륨을 마운트합니다. 여기에는 장치를 검색한 다음 장치를 마운트하는 등 볼륨을 마운트하는 데 필요한 모든 항목이 포함됩니다.

- 인수: **<mount-dir> <json>**
- 실행 위치: 노드
- 예상 출력: 기본 JSON

unmount

디렉터리에서 볼륨의 마운트를 해제합니다. 여기에는 마운트 해제 후 볼륨을 정리하는 데 필요한 모든 항목이 포함됩니다.

- 인수: **<mount-dir>**
- 실행 위치: 노드
- 예상 출력: 기본 JSON

mountdevice

개별 Pod가 마운트를 바인딩할 수 있는 디렉터리에 볼륨의 장치를 마운트합니다.

이 호출은 FlexVolume 사양에 지정된 "시크릿"을 전달하지 않습니다. 드라이버에 시크릿이 필요한 경우 이 호출을 구현하지 마십시오.

- 인수: **<mount-dir> <json>**
- 실행 위치: 노드
- 예상 출력: 기본 JSON

unmountdevice

디렉터리에서 볼륨의 장치를 마운트 해제합니다.

- 인수: **<mount-dir>**
- 실행 위치: 노드
- 예상 출력: 기본 JSON
  - 다른 모든 작업은 **{"status":를 사용하여 JSON을 반환해야 합니다: "지원되지 않음"}** 코드 **1**을 종료합니다.

절차

FlexVolume 드라이버를 설치하려면 다음을 수행합니다.

1. 실행 가능한 파일이 클러스터의 모든 노드에 있는지 확인합니다.
2. 볼륨 플러그인 경로(**/etc/kubernetes/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>**)에 실행 가능 파일을 배치합니다.

예를 들어, **foo** 스토리지용 FlexVolume 드라이버를 설치하려면 실행 파일을 **/etc/kubernetes/kubelet-plugins/volume/exec/openshift.com~foo/foo**에 배치합니다.

4.6.4. FlexVolume 드라이버를 사용한 스토리지 사용

OpenShift Container Platform의 각 **PersistentVolume** 오브젝트는 볼륨과 같이 스토리지 백엔드에서 1개의 스토리지 자산을 나타냅니다.

절차

- **PersistentVolume** 오브젝트를 사용하여 설치된 스토리지를 참조합니다.

FlexVolume 드라이버를 사용한 영구 볼륨 오브젝트 정의 예

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 1Gi 2
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: openshift.com/foo 3
    fsType: "ext4" 4
    secretRef: foo-secret 5
    readOnly: true 6
    options: 7
      fooServer: 192.168.0.1:1234
      fooVolumeName: bar
```

1. 볼륨의 이름입니다. 영구 볼륨 클레임을 통해 또는 Pod에서 식별되는 방법입니다. 이 이름은 백엔드 스토리지의 볼륨 이름과 다를 수 있습니다.
2. 이 볼륨에 할당된 스토리지의 용량입니다.
3. 드라이버의 이름입니다. 이 필드는 필수입니다.
4. 볼륨에 존재하는 파일 시스템입니다. 이 필드는 선택 사항입니다.
5. 시크릿에 대한 참조입니다. 이 시크릿의 키와 값은 호출 시 FlexVolume 드라이버에 제공됩니다. 이 필드는 선택 사항입니다.
6. 읽기 전용 플래그입니다. 이 필드는 선택 사항입니다.
7. FlexVolume 드라이버에 대한 추가 옵션입니다. **options** 필드에 있는 사용자가 지정한 플래그 외에도 다음 플래그도 실행 파일에 전달됩니다.

```
"fsType":"<FS type>",
"readwrite":"<rw>",
"secret/key1":"<secret1>"
...
"secret/keyN":"<secretN>"
```

참고

시크릿은 호출을 마운트하거나 마운트 해제하기 위해서만 전달됩니다.

4.7. GCE 영구 디스크를 사용하는 스토리지

OpenShift Container Platform은 gcePD(GCE 영구 디스크 볼륨)를 지원합니다. GCE를 사용하여 영구 스토리지로 OpenShift Container Platform 클러스터를 프로비저닝할 수 있습니다. Kubernetes 및 GCE에 대해 어느 정도 익숙한 것으로 가정합니다.

Kubernetes 영구 볼륨 프레임워크를 사용하면 관리자는 영구 스토리지로 클러스터를 프로비저닝하고 사용자가 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다.

GCE 영구 디스크 볼륨은 동적으로 프로비저닝할 수 있습니다.

영구 볼륨은 단일 프로젝트 또는 네임스페이스에 바인딩되지 않으며, OpenShift Container Platform 클러스터에서 공유할 수 있습니다. 영구 볼륨 클레임은 프로젝트 또는 네임스페이스에 고유하며 사용자가 요청할 수 있습니다.

중요

인프라의 스토리지의 고가용성은 기본 스토리지 공급자가 담당합니다.

추가 리소스

- GCE 영구 디스크

4.7.1. GCE 스토리지 클래스 생성

스토리지 클래스는 스토리지 수준 및 사용량을 구분하고 조정하는 데 사용됩니다. 스토리지 클래스를 정의하면 사용자는 동적으로 프로비저닝된 영구 볼륨을 얻을 수 있습니다.

절차

1. OpenShift Container Platform 콘솔에서 **스토리지 → 스토리지 클래스**를 클릭합니다.
2. 스토리지 클래스 개요에서 **스토리지 클래스 만들기**를 클릭합니다.
3. 표시되는 페이지에 원하는 옵션을 정의합니다.
   a. 스토리지 클래스를 참조할 이름을 입력합니다.
   b. 선택적 설명을 입력합니다.
   c. 회수 정책을 선택합니다.
   d. 드롭다운 목록에서 **kubernetes.io/gce-pd**를 선택합니다.
   e. 원하는 대로 스토리지 클래스에 대한 추가 매개변수를 입력합니다.
4. **생성**을 클릭하여 스토리지 클래스를 생성합니다.

4.7.2. 영구 볼륨 클레임 생성

사전 요구 사항

OpenShift Container Platform에서 볼륨으로 마운트하기 전에 기본 인프라에 스토리지가 있어야 합니다.

절차

1. OpenShift Container Platform 콘솔에서 **스토리지 → 영구 볼륨 클레임**을 클릭합니다.
2. 영구 볼륨 클레임 생성 개요에서 **영구 볼륨 클레임 생성**을 클릭합니다.
3. 표시되는 페이지에 원하는 옵션을 정의합니다.
   a. 드롭다운 메뉴에서 이전에 생성한 스토리지 클래스를 선택합니다.
   b. 스토리지 클레임의 고유한 이름을 입력합니다.
   c. 액세스 모드를 선택합니다. 이렇게 하면 생성된 스토리지 클레임에 대한 읽기 및 쓰기 액세스가 결정됩니다.
   d. 스토리지 클레임의 크기를 정의합니다.
4. **만들기**를 클릭하여 영구 볼륨 클레임을 생성하고 영구 볼륨을 생성합니다.

4.7.3. 볼륨 형식

OpenShift Container Platform이 볼륨을 마운트하고 컨테이너에 전달하기 전에 영구 볼륨 정의에 **fsType** 매개변수에 의해 지정된 파일 시스템이 포함되어 있는지 확인합니다. 장치가 파일 시스템으로 포맷되지 않으면 장치의 모든 데이터가 삭제되고 장치는 지정된 파일 시스템에서 자동으로 포맷됩니다.

이를 통해 OpenShift Container Platform이 처음 사용하기 전에 형식화되므로 형식화되지 않은 GCE 볼륨을 영구 볼륨으로 사용할 수 있습니다.

4.8. HOSTPATH를 사용하는 영구 스토리지

OpenShift Container Platform 클러스터의 hostPath 볼륨은 호스트 노드 파일 시스템의 파일 또는 디렉터리를 Pod에 마운트합니다. 대부분의 Pod에는 hostPath 볼륨이 필요하지 않지만 테스트에 필요한 빠른 옵션을 제공합니다.

중요

클러스터 관리자는 권한으로 실행되도록 Pod를 구성해야 합니다. 이를 통해 동일한 노드의 Pod에 액세스 권한이 부여됩니다.

4.8.1. 개요

OpenShift Container Platform은 단일 노드 클러스터에서 개발 및 테스트를 위해 hostPath 마운트를 지원합니다.

프로덕션 클러스터에서는 hostPath를 사용할 수 없습니다. 대신, 클러스터 관리자는 GCE 영구 디스크 볼륨, NFS 공유 또는 Amazon EBS 볼륨과 같은 네트워크 리소스를 프로비저닝합니다. 네트워크 리소스는 스토리지 클래스 사용을 지원하여 동적 프로비저닝을 설정합니다.

hostPath 볼륨은 정적으로 프로비저닝해야 합니다.

중요

컨테이너 루트, / 또는 호스트와 컨테이너에서 동일한 경로에 마운트하지 마십시오. 컨테이너에 충분한 권한이 있는 경우 호스트 시스템이 손상될 수 있습니다. **/host**를 사용하여 호스트를 마운트하는 것이 안전합니다. 다음 예는 **/host**의 컨테이너에 마운트되는 호스트의 / 디렉터리를 보여줍니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-host-mount
spec:
  containers:
  - image: registry.access.redhat.com/ubi8/ubi
    name: test-container
    command: ['sh', '-c', 'sleep 3600']
    volumeMounts:
    - mountPath: /host
      name: host-slash
  volumes:
   - name: host-slash
     hostPath:
       path: /
       type: ''
```

4.8.2. 정적으로 hostPath 볼륨을 프로비저닝

hostPath 볼륨을 사용하는 Pod는 수동(정적) 프로비저닝을 통해 참조해야 합니다.

절차

1. PV(영구 볼륨)를 정의합니다. **PersistentVolume** 오브젝트 정의를 사용하여 **pv.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume (1)
  labels:
    type: local
spec:
  storageClassName: manual (2)
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce (3)
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/mnt/data" (4)
```

(1) 볼륨의 이름입니다. 이 이름을 사용하여 영구 볼륨 클레임 또는 Pod에 의해 식별됩니다.

(2) 영구 볼륨 클레임 요청을 이 영구 볼륨에 바인딩하는 데 사용됩니다.

3 볼륨은 단일 노드에서 **읽기-쓰기**로 마운트할 수 있습니다.

4 구성 파일은 볼륨이 클러스터 노드의 **/mnt/data**에 있음을 지정합니다. 컨테이너 루트, / 또는 호스트와 컨테이너에서 동일한 경로에 마운트하지 마십시오. 그러면 호스트 시스템이 손상될 수 있습니다. **/host**를 사용하여 호스트를 마운트하는 것이 안전합니다.

2. 파일에서 PV를 생성합니다.

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

3. PVC(영구 볼륨 클레임)를 정의합니다. **PersistentVolumeClaim** 오브젝트 정의를 사용하여 **pvc.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pvc-volume
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual
```

4. 파일에서 PVC를 생성합니다.

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

4.8.3. 권한이 있는 Pod에서 hostPath 공유 마운트

영구 볼륨 클레임을 생성한 후 애플리케이션에 의해 내부에서 사용될 수 있습니다. 다음 예시는 Pod 내부에서 이 공유를 마운트하는 방법을 보여줍니다.

사전 요구 사항

- 기본 hostPath 공유에 매핑된 영구 볼륨 클레임이 있습니다.

절차

- 기존 영구 볼륨 클레임을 마운트하는 권한이 있는 Pod를 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name 1
spec:
  containers:
    ...
    securityContext:
      privileged: true 2
    volumeMounts:
    - mountPath: /data 3
```

```
    name: hostpath-privileged
...
securityContext: {}
volumes:
  - name: hostpath-privileged
    persistentVolumeClaim:
      claimName: task-pvc-volume 4
```

1. Pod의 이름입니다.
2. 노드의 스토리지에 액세스하려면 Pod는 권한 있음으로 실행해야 합니다.
3. 권한이 있는 Pod 내부에서 호스트 경로 공유를 마운트하기 위한 경로입니다. 컨테이너 루트, / 또는 호스트와 컨테이너에서 동일한 경로에 마운트하지 마십시오. 컨테이너가 호스트 **/dev/pts** 파일과 같이 충분한 권한이 있는 경우 호스트 시스템이 손상될 수 있습니다. **/host**를 사용하여 호스트를 마운트하는 것이 안전합니다.
4. 이전에 생성된 **PersistentVolumeClaim** 오브젝트의 이름입니다.

4.9. ISCSI를 사용하는 영구 스토리지

iSCSI를 사용하여 영구 스토리지로 OpenShift Container Platform 클러스터를 프로비저닝할 수 있습니다. Kubernetes 및 iSCSI에 대해 어느 정도 익숙한 것으로 가정합니다.

Kubernetes 영구 볼륨 프레임워크를 사용하면 관리자는 영구 스토리지로 클러스터를 프로비저닝하고 사용자가 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다.

중요

인프라의 스토리지의 고가용성은 기본 스토리지 공급자가 담당합니다.

중요

Amazon Web Services에서 iSCSI를 사용하는 경우 iSCSI 포트의 노드 간 TCP 트래픽을 포함하도록 기본 보안 정책을 업데이트해야 합니다. 기본적으로 해당 포트는 **860** 및 **3260**입니다.

중요

사용자는 **iscsi-initiator-utils** 패키지를 설치하고 이니시에이터 이름을 **/etc/iscsi/initiatorname.iscsi** 에 구성하여 iSCSI 이니시에이터가 모든 OpenShift Container Platform 노드에 이미 구성되어 있는지 확인해야 합니다. **iscsi-initiator-utils** 패키지는 RHCOS(Red Hat Enterprise Linux CoreOS)를 사용하는 배포에 이미 설치되어 있습니다.

자세한 내용은 스토리지 장치 관리를 참조하십시오.

4.9.1. 프로비저닝

OpenShift Container Platform에서 볼륨으로 마운트하기 전에 기본 인프라에 스토리지가 있는지 확인합니다. iSCSI에는 iSCSI 대상 포털, 유효한 IQN(iSCSI Qualified Name), 유효한 LUN 번호, 파일 시스템 유형 및 **PersistentVolume** API만 있으면 됩니다.

PersistentVolume 오브젝트 정의

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.16.154.81:3260
    iqn: iqn.2014-12.example.server:storage.target00
    lun: 0
    fsType: 'ext4'
```

4.9.2. 디스크 할당량 강제 적용

LUN 파티션을 사용하여 디스크 할당량 및 크기 제약 조건을 강제 적용합니다. 각 LUN은 1개의 영구 볼륨입니다. Kubernetes는 영구 볼륨에 고유 이름을 강제 적용합니다.

이렇게 하면 최종 사용자가 특정 용량(예: **10Gi**)에 의해 영구 스토리지를 요청하고 해당 볼륨과 동등한 용량과 일치시킬 수 있습니다.

4.9.3. iSCSI 볼륨 보안

사용자는 **PersistentVolumeClaim** 오브젝트를 사용하여 스토리지를 요청합니다. 이 클레임은 사용자의 네임스페이스에만 존재하며, 동일한 네임스페이스 내의 Pod에서만 참조할 수 있습니다. 네임스페이스에서 영구 볼륨 클레임에 대한 액세스를 시도하면 Pod가 실패하게 됩니다.

각 iSCSI LUN은 클러스터의 모든 노드에서 액세스할 수 있어야 합니다.

4.9.3.1. CHAP(Challenge Handshake Authentication Protocol) 구성

선택적으로 OpenShift는 CHAP을 사용하여 iSCSI 대상에 자신을 인증할 수 있습니다.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    fsType: ext4
    chapAuthDiscovery: true (1)
```

```
    chapAuthSession: true (2)
    secretRef:
      name: chap-secret (3)
```

(1) iSCSI 검색의 CHAP 인증을 활성화합니다.

(2) iSCSI 세션의 CHAP 인증을 활성화합니다.

(3) 사용자 이름 + 암호로 시크릿 오브젝트의 이름을 지정합니다. 이 **Secret** 오브젝트는 참조된 볼륨을 사용할 수 있는 모든 네임스페이스에서 사용할 수 있어야 합니다.

4.9.4. iSCSI 다중 경로

iSCSI 기반 스토리지의 경우 두 개 이상의 대상 포털 IP 주소에 동일한 IQN을 사용하여 여러 경로를 구성할 수 있습니다. 경로의 구성 요소 중 하나 이상에 실패하면 다중 경로를 통해 영구 볼륨에 액세스할 수 있습니다.

Pod 사양에 다중 경로를 지정하려면 **portals** 필드를 사용합니다. 예를 들면 다음과 같습니다.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260'] (1)
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    fsType: ext4
    readOnly: false
```

(1) **portals** 필드를 사용하여 추가 대상 포털을 추가합니다.

4.9.5. iSCSI 사용자 정의 이니시에이터 IQN

iSCSI 대상이 특정 IQN으로 제한되는 경우 사용자 정의 이니시에이터 IQN(iSCSI Qualified Name)을 구성하지만 iSCSI PV가 연결된 노드는 이러한 IQN을 갖는 것이 보장되지 않습니다.

사용자 정의 이니시에이터 IQN을 지정하려면 **initiatorName** 필드를 사용합니다.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
```

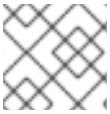
```
  - ReadWriteOnce
iscsi:
  targetPortal: 10.0.0.1:3260
  portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260']
  iqn: iqn.2016-04.test.com:storage.target00
  lun: 0
  initiatorName: iqn.2016-04.test.com:custom.iqn (1)
  fsType: ext4
  readOnly: false
```

1 이니시에이터의 이름을 지정합니다.

4.10. 로컬 블록을 사용하는 영구 스토리지

OpenShift Container Platform은 로컬 볼륨을 사용하여 영구 스토리지를 통해 프로비저닝될 수 있습니다. 로컬 영구 볼륨을 사용하면 표준 영구 볼륨 클레임 인터페이스를 사용하여 디스크 또는 파티션과 같은 로컬 스토리지 장치에 액세스할 수 있습니다.

시스템에서 볼륨 노드 제약 조건을 인식하고 있기 때문에 로컬 볼륨은 노드에 수동으로 Pod를 예약하지 않고 사용할 수 있습니다. 그러나 로컬 볼륨은 여전히 기본 노드의 가용성에 따라 달라지며 일부 애플리케이션에는 적합하지 않습니다.



참고

로컬 볼륨은 정적으로 생성된 영구 볼륨으로 사용할 수 있습니다.

4.10.1. Local Storage Operator 설치

Local Storage Operator는 기본적으로 OpenShift Container Platform에 설치되지 않습니다. 다음 절차에 따라 이 Operator를 설치하고 구성하여 클러스터에서 로컬 볼륨을 활성화합니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔 또는 CLI(명령줄 인터페이스)에 액세스할 수 있습니다.

절차

1. **openshift-local-storage** 프로젝트를 생성합니다.

```
$ oc adm new-project openshift-local-storage
```

2. 선택 사항: 인프라 노드에서 로컬 스토리지 생성을 허용합니다.
Local Storage Operator를 사용하여 로깅 및 모니터링과 같은 구성 요소를 지원하는 인프라 노드에 볼륨을 생성할 수 있습니다.

   Local Storage Operator에 작업자 노드가 아닌 인프라 노드가 포함되도록 기본 노드 선택기를 조정해야 합니다.

   Local Storage Operator가 클러스터 전체 기본 선택기를 상속하지 못하도록 하려면 다음 명령을 입력합니다.

```
$ oc annotate project openshift-local-storage openshift.io/node-selector=''
```

UI에서

웹 콘솔에서 Local Storage Operator를 설치하려면 다음 단계를 따르십시오.

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. **Operators → OperatorHub**로 이동합니다.
3. **Local Storage**를 필터 상자에 입력하여 Local Storage Operator를 찾습니다.
4. **설치**를 클릭합니다.
5. **Operator 설치** 페이지에서 **클러스터의 특정 네임스페이스**를 선택합니다. 드롭다운 메뉴에서 **openshift-local-storage**를 선택합니다.
6. **업데이트 채널** 및 **승인 전략** 값을 원하는 값으로 조정합니다.
7. **설치**를 클릭합니다.

완료되면 Local Storage Operator가 웹 콘솔의 **설치된 Operator** 섹션에 나열됩니다.

CLI에서

1. CLI에서 Local Storage Operator를 설치합니다.
   a. 다음 명령을 실행하여 OpenShift Container Platform 주 버전 및 부 버전을 가져옵니다. 다음 단계의 **channel** 값에 필요합니다.

```
$ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
    grep -o '[0-9]*[.][0-9]*' | head -1)
```

   b. **openshift-local-storage.yaml**과 같은 Local Storage Operator의 Operator 그룹 및 서브스크립션을 정의하는 오브젝트 YAML 파일을 생성합니다.

예: openshift-local-storage.yaml

```
apiVersion: operators.coreos.com/v1alpha2
kind: OperatorGroup
metadata:
  name: local-operator-group
  namespace: openshift-local-storage
spec:
  targetNamespaces:
    - openshift-local-storage
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: openshift-local-storage
spec:
  channel: "${OC_VERSION}"
  installPlanApproval: Automatic (1)
  name: local-storage-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

1 설치 계획에 대한 사용자 승인 정책입니다.

2. 다음 명령을 입력하여 Local Storage Operator 오브젝트를 생성합니다.

```
$ oc apply -f openshift-local-storage.yaml
```

이 시점에서 OLM(Operator Lifecycle Manager)은 Local Storage Operator를 인식합니다. Operator의 ClusterServiceVersion (CSV)이 대상 네임스페이스에 표시되고 Operator가 제공한 API를 작성할 수 있어야 합니다.

3. 모든 Pod 및 Local Storage Operator가 생성되었는지 확인하여 로컬 스토리지 설치를 확인합니다.

a. 필요한 모든 Pod가 생성되었는지 확인합니다.

```
$ oc -n openshift-local-storage get pods
```

출력 예

```
NAME                                      READY   STATUS    RESTARTS   AGE
local-storage-operator-746bf599c9-vlt5t   1/1     Running   0          19m
```

b. CSV(ClusterServiceVersion) YAML 매니페스트를 확인하여 **openshift-local-storage** 프로젝트에서 Local Storage Operator를 사용할 수 있는지 확인합니다.

```
$ oc get csvs -n openshift-local-storage
```

출력 예

```
NAME                                         DISPLAY         VERSION               REPLACES   PHASE
local-storage-operator.4.2.26-202003230335   Local Storage   4.2.26-202003230335              Succeeded
```

모든 확인이 통과되면 Local Storage Operator가 성공적으로 설치됩니다.

4.10.2. Local Storage Operator를 사용하여 로컬 볼륨을 프로비저닝

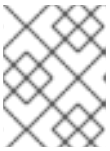
동적 프로비저닝을 통해 로컬 볼륨을 생성할 수 없습니다. 대신 Local Storage Operator에서 영구 볼륨을 생성할 수 있습니다. 로컬 볼륨 프로비저너는 정의된 리소스에 지정된 경로에서 모든 파일 시스템 또는 블록 볼륨 장치를 찾습니다.

사전 요구 사항

- Local Storage Operator가 설치되어 있습니다.
- 다음 조건을 충족하는 로컬 디스크가 있습니다.
  - 노드에 연결되어 있습니다.
  - 마운트되지 않았습니다.
  - 파티션이 포함되어 있지 않습니다.

절차

1. 로컬 볼륨 리소스를 생성합니다. 이 리소스는 로컬 볼륨에 대한 노드 및 경로를 정의해야 합니다.



참고

동일한 장치에 다른 스토리지 클래스 이름을 사용하지 마십시오. 이렇게 하면 여러 영구 볼륨(PV)이 생성됩니다.

예제: 파일 시스템

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage" (1)
spec:
  nodeSelector: (2)
    nodeSelectorTerms:
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - ip-10-0-140-183
          - ip-10-0-158-139
          - ip-10-0-164-33
  storageClassDevices:
    - storageClassName: "local-sc" (3)
      volumeMode: Filesystem (4)
      fsType: xfs (5)
      devicePaths: (6)
        - /path/to/device (7)
```

(1) Local Storage Operator가 설치된 네임스페이스입니다.

(2) 선택 사항: 로컬 스토리지 볼륨이 연결된 노드 목록이 포함된 노드 선택기입니다. 이 예에서는 **oc get node**에서 가져온 노드 호스트 이름을 사용합니다. 값을 정의하지 않으면 Local Storage Operator에서 사용 가능한 모든 노드에서 일치하는 디스크를 찾습니다.

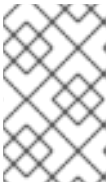
(3) 영구 볼륨 오브젝트를 생성할 때 사용할 스토리지 클래스의 이름입니다. Local Storage Operator가 존재하지 않는 경우 스토리지 클래스를 자동으로 생성합니다. 이 로컬 볼륨 세트를 고유하게 식별하는 스토리지 클래스를 사용해야 합니다.

(4) 로컬 볼륨의 유형을 정의하는 **Filesystem** 또는 **Block** 중 하나에 해당 볼륨 모드입니다.

(5) 로컬 볼륨이 처음 마운트될 때 생성되는 파일 시스템입니다.

(6) 선택할 로컬 스토리지 장치 목록이 포함된 경로입니다.

(7) 이 값을 **LocalVolume** 리소스에 대한 실제 로컬 디스크 파일 경로(예: **/dev/disk/ by-id /wwn**)로 바꿉니다. 프로비저너가 배포되면 이러한 로컬 디스크에 PV가 생성됩니다.



참고

원시 블록 볼륨(**volumeMode: block**)은 파일 시스템과 함께 포맷되지 않습니다. Pod에서 실행되는 모든 애플리케이션이 원시 블록 장치를 사용할 수 있는 경우에만 이 모드를 사용해야 합니다.

예제: 블록

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage" 1
spec:
  nodeSelector: 2
    nodeSelectorTerms:
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - ip-10-0-136-143
          - ip-10-0-140-255
          - ip-10-0-144-180
  storageClassDevices:
    - storageClassName: "localblock-sc" 3
      volumeMode: Block 4
      devicePaths: 5
        - /path/to/device 6
```

1. Local Storage Operator가 설치된 네임스페이스입니다.
2. 선택 사항: 로컬 스토리지 볼륨이 연결된 노드 목록이 포함된 노드 선택기입니다. 이 예에서는 **oc get node**에서 가져온 노드 호스트 이름을 사용합니다. 값을 정의하지 않으면 Local Storage Operator에서 사용 가능한 모든 노드에서 일치하는 디스크를 찾습니다.
3. 영구 볼륨 오브젝트를 생성할 때 사용할 스토리지 클래스의 이름입니다.
4. 로컬 볼륨의 유형을 정의하는 **Filesystem** 또는 **Block** 중 하나에 해당 볼륨 모드입니다.
5. 선택할 로컬 스토리지 장치 목록이 포함된 경로입니다.
6. 이 값을 **LocalVolume** 리소스에 대한 실제 로컬 디스크 파일 경로(예: **dev/disk/ by-id /wwn**)로 바꿉니다. 프로비저너가 배포되면 이러한 로컬 디스크에 PV가 생성됩니다.

2. OpenShift Container Platform 클러스터에 로컬 볼륨 리소스를 생성합니다. 방금 생성한 파일을 지정합니다.

```
$ oc create -f <local-volume>.yaml
```

3. 프로비저너가 생성되었고 해당 데몬 세트가 생성되었는지 확인합니다.

```
$ oc get all -n openshift-local-storage
```

출력 예

```
NAME                                READY   STATUS    RESTARTS   AGE
pod/local-disks-local-provisioner-h97hj      1/1     Running   0          46m
pod/local-disks-local-provisioner-j4mnn      1/1     Running   0          46m
pod/local-disks-local-provisioner-kbdnx      1/1     Running   0          46m
pod/local-disks-local-diskmaker-ldldw        1/1     Running   0          46m
pod/local-disks-local-diskmaker-lvrv4        1/1     Running   0          46m
pod/local-disks-local-diskmaker-phxdq        1/1     Running   0          46m
pod/local-storage-operator-54564d9988-vxvhx  1/1     Running   0          47m

NAME                           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
service/local-storage-operator    ClusterIP   172.30.49.90    <none>        60000/TCP   47m

NAME                                DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/local-disks-local-provisioner   3         3         3       3
3           <none>
46m
daemonset.apps/local-disks-local-diskmaker     3         3         3       3
3           <none>
46m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/local-storage-operator   1/1     1            1           47m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/local-storage-operator-54564d9988   1         1         1       47m
```

필요한 데몬 세트 프로세스 및 현재 개수를 기록해 둡니다. **0**의 개수는 레이블 선택기가 유효하지 않음을 나타냅니다.

4. 영구 볼륨이 생성되었는지 확인합니다.

```
$ oc get pv
```

출력 예

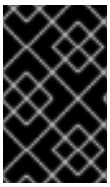
```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS    REASON   AGE
local-pv-1cec77cf   100Gi      RWO            Delete           Available           local-sc                88m
local-pv-2ef7cd2a   100Gi      RWO            Delete           Available           local-sc
82m
local-pv-3fa1c73    100Gi      RWO            Delete           Available           local-sc                48m
```

중요

LocalVolume 오브젝트 편집은 안전하지 않은 작업이므로 이를 수행하면 기존 영구 볼륨의 **fsType** 또는 **volumeMode**가 변경되지 않습니다.

4.10.3. Local Storage Operator없이 로컬 볼륨 프로비저닝

동적 프로비저닝을 통해 로컬 볼륨을 생성할 수 없습니다. 대신 개체 정의에서 영구 볼륨(PV)을 정의하여 영구 볼륨을 생성할 수 있습니다. 로컬 볼륨 프로비저너는 정의된 리소스에 지정된 경로에서 모든 파일 시스템 또는 블록 볼륨 장치를 찾습니다.



중요

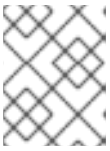
PVC가 삭제될 때 PV를 수동으로 프로비저닝하면 PV를 재사용할 때 데이터 누출의 위험이 발생할 수 있습니다. Local Storage Operator는 로컬 PV를 프로비저닝할 때 장치의 라이프사이클을 자동화하는 것이 좋습니다.

사전 요구 사항

- 로컬 디스크가 OpenShift Container Platform 노드에 연결되어 있습니다.

절차

1. PV를 정의합니다. **PersistentVolume** 오브젝트 정의로 **example-pv-filesystem.yaml** 또는 **example-pv-block.yaml**과 같은 파일을 생성합니다. 이 리소스는 로컬 볼륨에 대한 노드 및 경로를 정의해야 합니다.



참고

동일한 장치에 다른 스토리지 클래스 이름을 사용하지 마십시오. 이렇게 하면 여러 PV가 생성됩니다.

example-pv-filesystem.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv-filesystem
spec:
  capacity:
    storage: 100Gi
  volumeMode: Filesystem (1)
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage (2)
  local:
    path: /dev/xvdf (3)
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

(1) PV의 유형을 정의하는 **Filesystem** 또는 **Block** 중 하나에 해당 볼륨 모드입니다.

(2) PV 리소스를 생성할 때 사용할 스토리지 클래스의 이름입니다. 이 PV 세트를 고유하게 식별하는 스토리지 클래스를 사용합니다.

(3) 선택할 로컬 스토리지 장치 목록이 포함된 경로입니다.

참고

원시 블록 볼륨(**volumeMode: block**)은 파일 시스템과 함께 포맷되지 않습니다. Pod에서 실행되는 모든 애플리케이션이 원시 블록 장치를 사용할 수 있는 경우에만 이 모드를 사용합니다.

example-pv-block.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv-block
spec:
  capacity:
    storage: 100Gi
  volumeMode: Block 1
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage 2
  local:
    path: /dev/xvdf 3
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

1. PV의 유형을 정의하는 **Filesystem** 또는 **Block** 중 하나에 해당 볼륨 모드입니다.
2. PV 리소스를 생성할 때 사용할 스토리지 클래스의 이름입니다. 이 PV 세트를 고유하게 식별하는 스토리지 클래스를 사용해야 합니다.
3. 선택할 로컬 스토리지 장치 목록이 포함된 경로입니다.

2. OpenShift Container Platform 클러스터에 PV 리소스를 생성합니다. 방금 생성한 파일을 지정합니다.

```
$ oc create -f <example-pv>.yaml
```

3. 로컬 PV가 생성되었는지 확인합니다.

```
$ oc get pv
```

출력 예

```
NAME                    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS    REASON   AGE
example-pv-filesystem   100Gi      RWO            Delete           Available                 local-
storage                 3m47s
```

```
example-pv1   1Gi   RWO   Delete   Bound   local-storage/pvc1   local-
storage   12h
example-pv2   1Gi   RWO   Delete   Bound   local-storage/pvc2   local-
storage   12h
example-pv3   1Gi   RWO   Delete   Bound   local-storage/pvc3   local-
storage   12h
```

4.10.4. 로컬 볼륨 영구 볼륨 클레임 생성

로컬 볼륨은 Pod가 액세스할 수 있는 영구 볼륨 클레임(PVC)으로서 정적으로 생성되어야 합니다.

사전 요구 사항

- 영구 볼륨은 로컬 볼륨 프로비저너를 사용하여 생성됩니다.

절차

1. 해당 스토리지 클래스를 사용하여 PVC를 생성합니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: local-pvc-name 1
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem 2
  resources:
    requests:
      storage: 100Gi 3
  storageClassName: local-sc 4
```

1. PVC의 이름입니다.
2. PVC의 유형입니다. 기본값은 **Filesystem**입니다.
3. PVC에서 사용할 수 있는 스토리지 용량입니다.
4. 클레임에 필요한 스토리지 클래스의 이름입니다.

2. 방금 작성한 파일을 지정하여 OpenShift Container Platform 클러스터에서 PVC를 생성합니다.

```
$ oc create -f <local-pvc>.yaml
```

4.10.5. 로컬 클레임 연결

로컬 볼륨이 영구 볼륨 클레임에 매핑된 후 리소스 내부에서 지정할 수 있습니다.

사전 요구 사항

- 동일한 네임스페이스에 영구 볼륨 클레임이 있어야 합니다.

절차

1. 리소스 사양에 정의된 클레임을 포함합니다. 다음 예시는 Pod 내에서 영구 볼륨 클레임을 선언합니다.

```
apiVersion: v1
kind: Pod
spec:
  ...
  containers:
    volumeMounts:
    - name: local-disks 1
      mountPath: /data 2
  volumes:
  - name: localpvc
    persistentVolumeClaim:
      claimName: local-pvc-name 3
```

1 마운트할 볼륨의 이름입니다.

2 볼륨이 마운트된 Pod 내부의 경로입니다. 컨테이너 루트, / 또는 호스트와 컨테이너에서 동일한 경로에 마운트하지 마십시오. 컨테이너가 호스트 **/dev/pts** 파일과 같이 충분한 권한이 있는 경우 호스트 시스템이 손상될 수 있습니다. **/host**를 사용하여 호스트를 마운트하는 것이 안전합니다.

3 사용할 기존 영구 볼륨 클레임의 이름입니다.

2. 방금 생성한 파일을 지정하여 OpenShift Container Platform 클러스터에 리소스를 생성합니다.

```
$ oc create -f <local-pod>.yaml
```

4.10.6. 로컬 스토리지 장치에 대한 검색 및 프로비저닝 자동화

로컬 스토리지 Operator는 로컬 스토리지 검색 및 프로비저닝을 자동화합니다. 이 기능을 사용하면 배포 중에 연결된 장치가 있는 베어 메탈, VMware 또는 AWS 스토어 인스턴스와 같이 동적 프로비저닝을 사용할 수 없는 경우 설치를 단순화할 수 있습니다.

중요

자동 검색 및 프로비저닝은 기술 프리뷰 기능 전용입니다. Technology Preview 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 https://access.redhat.com/support/offerings/techpreview/를 참조하십시오.

다음 절차에 따라 로컬 장치를 자동으로 검색하고 선택한 장치에 대한 로컬 볼륨을 자동으로 프로비저닝하십시오.

주의

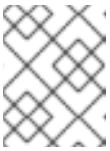
LocalVolumeSet 오브젝트를 주의해서 사용합니다. 로컬 디스크에서 PV(영구 볼륨)를 자동으로 프로비저닝하면 로컬 PV에서 일치하는 모든 장치를 클레임할 수 있습니다. **LocalVolumeSet** 오브젝트를 사용하는 경우 Local Storage Operator가 노드에서 로컬 장치를 관리하는 유일한 엔터티인지 확인합니다.

사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.
- Local Storage Operator가 설치되어 있습니다.
- OpenShift Container Platform 노드에 로컬 디스크가 연결되어 있습니다.
- OpenShift Container Platform 웹 콘솔과 **oc** 명령줄 인터페이스(CLI)에 액세스할 수 있습니다.

절차

1. 웹 콘솔에서 로컬 장치를 자동으로 검색할 수 있도록 하려면 다음을 수행합니다.
   a. *관리자*로 **Operator → 설치된 Operator**로 이동하여 **로컬 볼륨 검색** 탭을 클릭합니다.
   b. **로컬 볼륨 검색 만들기**를 클릭합니다.
   c. 모든 또는 특정 노드에서 사용 가능한 디스크를 검색할지의 여부에 따라 **모든 노드** 또는 **노드 선택**을 선택합니다.



참고

모든 노드 또는 **노드 선택**을 사용하여 필터링하는지의 여부와 관계없이 작업자 노드만 사용할 수 있습니다.

   d. **Create**를 클릭합니다.

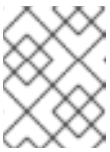
이름이 **auto-discover-devices**인 로컬 볼륨 검색 인스턴스가 표시됩니다.

1. 노드에 사용 가능한 장치 목록을 표시하려면 다음을 수행합니다.
   a. OpenShift Container Platform 웹 콘솔에 로그인합니다.
   b. **컴퓨팅 → 노드**로 이동합니다.
   c. 열기를 원하는 노드 이름을 클릭합니다. "노드 세부 정보" 페이지가 표시됩니다.
   d. 선택한 장치 목록을 표시하려면 **디스크** 탭을 선택합니다.
   로컬 디스크가 추가되거나 제거되면 장치 목록이 지속적으로 업데이트됩니다. 장치를 이름, 상태, 유형, 모델, 용량 및 모드로 필터링할 수 있습니다.
2. 웹 콘솔에서 발견된 장치에 대한 로컬 볼륨을 자동으로 프로비저닝하려면 다음을 수행합니다.
   a. **Operator → 설치된 Operator**로 이동하고 Operator 목록에서 **로컬 스토리지**를 선택합니다.

b. **로컬 볼륨 세트 → 로컬 볼륨 세트 만들기**를 선택합니다.

c. 볼륨 세트 이름과 스토리지 클래스 이름을 입력합니다.

d. 그에 따라 필터를 적용하려면 **모든 노드** 또는 **노드 선택**을 선택합니다.



참고

모든 노드 또는 **노드 선택**을 사용하여 필터링하는지의 여부와 관계없이 작업자 노드만 사용할 수 있습니다.

e. 로컬 볼륨 세트에 적용할 디스크 유형, 모드, 크기 및 제한을 선택하고 **만들기**를 클릭합니다. 몇 분 후에 "Operator 조정됨"을 나타내는 메시지가 표시됩니다.

3. 대신 CLI에서 검색된 장치에 대한 로컬 볼륨을 프로비저닝하려면 다음을 수행합니다.

a. 다음 예와 같이 **local-volume-set.yaml**과 같은 로컬 볼륨 세트를 정의하는 오브젝트 YAML 파일을 생성합니다.

```
apiVersion: local.storage.openshift.io/v1alpha1
kind: LocalVolumeSet
metadata:
  name: example-autodetect
spec:
  nodeSelector:
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - worker-0
              - worker-1
  storageClassName: example-storageclass (1)
  volumeMode: Filesystem
  fsType: ext4
  maxDeviceCount: 10
  deviceInclusionSpec:
    deviceTypes: (2)
      - disk
      - part
    deviceMechanicalProperties:
      - NonRotational
    minSize: 10G
    maxSize: 100G
    models:
      - SAMSUNG
      - Crucial_CT525MX3
    vendors:
      - ATA
      - ST2000LM
```

(1) 검색된 장치에서 프로비저닝된 영구 볼륨에 대해 생성된 스토리지 클래스를 결정합니다. Local Storage Operator가 존재하지 않는 경우 스토리지 클래스를 자동으로 생성합니다. 이 로컬 볼륨 세트를 고유하게 식별하는 스토리지 클래스를 사용해야 합니다.

2 로컬 볼륨 세트 기능을 사용할 때 Local Storage Operator는 논리 볼륨 관리(LVM) 장치 사용을 지원하지 않습니다.

b. 로컬 볼륨 세트 오브젝트를 생성합니다.

```
$ oc apply -f local-volume-set.yaml
```

c. 로컬 영구 볼륨이 스토리지 클래스를 기반으로 동적으로 프로비저닝되었는지 확인합니다.

```
$ oc get pv
```

출력 예

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM   STORAGECLASS        REASON   AGE
local-pv-1cec77cf   100Gi      RWO            Delete           Available           example-
storageclass            88m
local-pv-2ef7cd2a   100Gi      RWO            Delete           Available           example-
storageclass            82m
local-pv-3fa1c73    100Gi      RWO            Delete           Available           example-
storageclass            48m
```

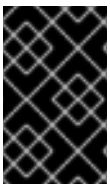
참고

결과는 노드에서 제거된 후 삭제됩니다. 심볼릭 링크는 수동으로 제거해야 합니다.

4.10.7. Local Storage Operator Pod에서 허용 오차 사용

테인트를 노드에 적용하여 일반 워크로드를 실행하지 못하도록 할 수 있습니다. Local Storage Operator가 테인트된 노드를 사용하도록 허용하려면 **Pod** 또는 **DaemonSet** 정의에 허용 오차를 추가해야 합니다. 그러면 생성된 리소스가 이러한 테인트된 노드에서 실행될 수 있습니다.

LocalVolume 리소스를 통해 Local Storage Operator Pod에 허용 오차를 적용하고 노드 사양을 통해 노드에 테인트를 적용합니다. 노드의 테인트는 해당 테인트를 허용하지 않는 모든 Pod를 거절하도록 노드에 지시합니다. 다른 Pod에 없는 특정 테인트를 사용하면 Local Storage Operator Pod가 해당 노드에서도 실행될 수 있습니다.



중요

테인트 및 허용 오차는 key, value 및 effect로 구성되어 있습니다. 인수로는 **key=value:effect**로 표시됩니다. Operator는 이러한 매개 변수 중 하나를 비워두는 것을 허용합니다.

사전 요구 사항

- Local Storage Operator가 설치되어 있습니다.
- 로컬 디스크는 테인트와 함께 OpenShift Container Platform 노드에 연결되어 있습니다.
- 테인트된 노드는 로컬 스토리지를 프로비저닝해야 합니다.

절차

테인트된 노드에서 스케줄링을 위해 로컬 볼륨을 구성하려면 다음을 수행하십시오.

1. 다음 예와 같이 **Pod**를 정의하는 YAML 파일을 수정하고 **LocalVolume** 사양을 추가합니다.

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage"
spec:
  tolerations:
    - key: localstorage 1
      operator: Equal 2
      value: "localstorage" 3
  storageClassDevices:
    - storageClassName: "localblock-sc"
      volumeMode: Block 4
      devicePaths: 5
        - /dev/xvdg
```

1. 노드에 추가한 키를 지정합니다.
2. **키/값** 매개변수가 일치할 것을 요구하도록 **Equal** Operator를 지정합니다. Operator가 **Exists**인 경우 시스템은 키가 존재하는지 확인하고 값을 무시합니다. Operator가 **Equal** 이면 키와 값이 일치해야 합니다.
3. 테인트된 노드의 **로컬** 값을 지정합니다.
4. 볼륨 모드(**파일 시스템** 또는 **블록**)는 로컬 볼륨의 유형을 정의합니다.
5. 선택할 로컬 스토리지 장치 목록이 포함된 경로입니다.

2. 선택 사항: 테인트된 노드에만 로컬 영구 볼륨을 생성하려면 다음 예와 같이 YAML 파일을 수정하고 **LocalVolume** 사양을 추가합니다.

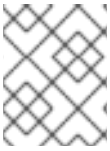
```
spec:
  tolerations:
    - key: node-role.kubernetes.io/master
      operator: Exists
```

정의된 허용 오차가 결과 데몬 세트로 전달되어, 지정된 테인트를 포함하는 노드에 대해 디스크 제조 업체 및 프로비저너 Pod를 생성할 수 있습니다.

4.10.8. Local Storage Operator 리소스 삭제

4.10.8.1. 로컬 볼륨 또는 로컬 볼륨 세트 제거

때때로 로컬 볼륨 및 로컬 볼륨 세트를 삭제해야 합니다. 리소스에서 항목을 제거하고 영구 볼륨을 삭제하는 것은 일반적으로 충분합니다. 동일한 장치 경로를 재사용하거나 다른 스토리지 클래스로 관리하려면 추가 단계가 필요합니다.



참고

다음 절차에서는 로컬 볼륨을 제거하는 예시를 간단히 설명합니다. 동일한 절차를 사용하여 로컬 볼륨 세트 사용자 정의 리소스의 심볼릭 링크를 제거할 수도 있습니다.

사전 요구 사항

- 영구 볼륨이 **해제된** 또는 **사용 가능** 상태여야 합니다.

주의

아직 사용 중인 영구 볼륨을 삭제하면 데이터 손실 또는 손상이 발생할 수 있습니다.

절차

1. 이전에 생성된 로컬 볼륨을 편집하여 원하는 디스크를 제거합니다.
   a. 클러스터 리소스를 편집합니다.

   ```
   $ oc edit localvolume <name> -n openshift-local-storage
   ```

   b. **devicePaths** 아래의 행으로 이동하여 원하는 디스크를 나타내는 모든 항목을 삭제합니다.
2. 생성된 영구 볼륨을 삭제합니다.

   ```
   $ oc delete pv <pv-name>
   ```

3. 노드의 심볼릭 링크를 삭제합니다.

주의

다음 단계는 루트 사용자로 노드에 액세스하는 것입니다. 이 절차의 단계 이후에 노드 상태를 수정하면 클러스터가 불안정해질 수 있습니다.

   a. 노드에서 디버그 Pod를 생성합니다.

   ```
   $ oc debug node/<node-name>
   ```

   b. 루트 디렉터리를 호스트로 변경하십시오.

   ```
   $ chroot /host
   ```

   c. 로컬 볼륨 심볼릭 링크가 포함된 디렉터리로 이동합니다.

```
$ cd /mnt/openshift-local-storage/<sc-name> 1
```

1 로컬 볼륨을 생성하기 위해 사용되는 스토리지 클래스의 이름입니다.

d. 삭제된 장치에 속한 심볼릭 링크를 삭제합니다.

```
$ rm <symlink>
```

4.10.8.2. Local Storage Operator 설치 제거

Local Storage Operator의 설치를 제거하려면 **openshift-local-storage** 프로젝트에서 Operator 및 모든 생성된 리소스를 제거해야 합니다.

> **주의**
>
> 로컬 스토리지 PV를 아직 사용 중 일 때 Local Storage Operator의 설치를 제거하는 것은 권장되지 않습니다. Operator 제거 후 PV는 유지되지만, PV 및 로컬 스토리지 리소스를 제거하지 않고 Operator를 제거한 후 다시 설치하면 알 수 없는 동작이 발생할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 액세스합니다.

절차

1. 프로젝트에 설치된 로컬 볼륨 리소스(예: localvolume , **localvolume set**, **localvolumediscovery**)를 삭제합니다.

```
$ oc delete localvolume --all --all-namespaces
$ oc delete localvolumeset --all --all-namespaces
$ oc delete localvolumediscovery --all --all-namespaces
```

2. 웹 콘솔에서 Local Storage Operator의 설치를 제거합니다.

   a. OpenShift Container Platform 웹 콘솔에 로그인합니다.

   b. **Operators → 설치된 Operator**로 이동합니다.

   c. **Local Storage**를 필터 상자에 입력하여 Local Storage Operator를 찾습니다.

   d. Local Storage Operator 끝에 있는 옵션 메뉴 ⋮ 를 클릭합니다.

   e. **Operator 제거**를 클릭합니다.

   f. 표시되는 창에서 **제거**를 클릭합니다.

3. Local Storage Operator에서 생성한 PV는 삭제될 때까지 클러스터에 남아 있습니다. 이러한 볼륨이 더 이상 사용되지 않으면 다음 명령을 실행하여 해당 볼륨을 삭제합니다.

```
$ oc delete pv <pv-name>
```

4. **openshift-local-storage** 프로젝트를 삭제합니다.

```
$ oc delete project openshift-local-storage
```

4.11. NFS를 사용하는 영구저장장치

OpenShift Container Platform 클러스터는 NFS를 사용하는 영구 스토리지와 함께 프로비저닝될 수 있습니다. PV(영구 볼륨) 및 PVC(영구 볼륨 클레임)는 프로젝트 전체에서 볼륨을 공유하는 편리한 방법을 제공합니다. PV 정의에 포함된 NFS 관련 정보는 **Pod** 정의에서 직접 정의될 수 있지만, 이렇게 하면 볼륨이 별도의 클러스터 리소스로 생성되지 않아 볼륨에서 충돌이 발생할 수 있습니다.

추가 리소스

- 네트워크 파일 시스템(NFS)

4.11.1. 프로비저닝

OpenShift Container Platform에서 볼륨으로 마운트하기 전에 기본 인프라에 스토리지가 있어야 합니다. NFS 볼륨을 프로비저닝하려면, NFS 서버 목록 및 내보내기 경로만 있으면 됩니다.

절차

1. PV에 대한 오브젝트 정의를 생성합니다.

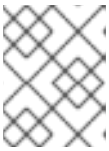
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 5Gi 2
  accessModes:
  - ReadWriteOnce 3
  nfs: 4
    path: /tmp 5
    server: 172.17.0.2 6
  persistentVolumeReclaimPolicy: Retain 7
```

1. 볼륨의 이름입니다. 이는 다양한 **oc <command> pod**에서 PV ID입니다.
2. 이 볼륨에 할당된 스토리지의 용량입니다.
3. 볼륨에 대한 액세스를 제어하는 것으로 표시되지만 실제로 레이블에 사용되며 PVC를 PV에 연결하는 데 사용됩니다. 현재는 **accessModes**를 기반으로 하는 액세스 규칙이 적용되지 않습니다.
4. 사용 중인 볼륨 유형으로, 이 경우에는 **nfs** 플러그인입니다.

5 NFS 서버에서 내보낸 경로입니다.

6 NFS 서버의 호스트 이름 또는 IP 주소입니다.

7 PV의 회수 정책입니다. 이는 릴리스될 때 볼륨에 발생하는 작업을 정의합니다.



참고

각 NFS 볼륨은 클러스터의 모든 스케줄링 가능한 노드에서 마운트할 수 있어야 합니다.

2. PV가 생성되었는지 확인합니다.

```
$ oc get pv
```

출력 예

```
NAME    LABELS    CAPACITY    ACCESSMODES   STATUS      CLAIM     REASON    AGE
pv0001  <none>    5Gi         RWO           Available                       31s
```

3. 새 PV에 바인딩하는 영구 볼륨 클레임을 생성합니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
    - ReadWriteOnce 1
  resources:
    requests:
      storage: 5Gi 2
  volumeName: pv0001
  storageClassName: ""
```

1 액세스 모드는 보안을 적용하지 않고 PV를 PVC에 일치시키기 위한 레이블 역할을 합니다.

2 이 클레임에서는 **5Gi** 이상의 용량을 제공하는 PV를 찾습니다.

4. 영구 볼륨 클레임이 생성되었는지 확인합니다.

```
$ oc get pvc
```

출력 예

```
NAME         STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfs-claim1   Bound     pv0001    5Gi        RWO                           2m
```

4.11.2. 디스크 할당량 강제 적용

디스크 파티션을 사용하여 디스크 할당량 및 크기 제약 조건을 적용할 수 있습니다. 각 파티션은 자체 내보내기일 수 있습니다. 각 내보내기는 1개의 PV입니다. OpenShift Container Platform은 PV에 고유한 이름을 적용하지만 NFS 볼륨 서버와 경로의 고유성은 관리자에 따라 다릅니다.

이렇게 하면 개발자가 10Gi와 같은 특정 용량에 의해 영구 스토리지를 요청하고 해당 볼륨과 동등한 용량과 일치시킬 수 있습니다.

4.11.3. NFS 볼륨 보안

이 섹션에서는 일치하는 권한 및 SELinux 고려 사항을 포함하여 NFS 볼륨 보안에 대해 설명합니다. 사용자는 POSIX 권한, 프로세스 UID, 추가 그룹 및 SELinux의 기본 사항을 이해하고 있어야 합니다.

개발자는 **Pod** 정의의 **볼륨** 섹션에서 직접 PVC 또는 NFS 볼륨 플러그인을 참조하여 NFS 스토리지를 요청합니다.

NFS 서버의 **/etc/exports** 파일에 액세스할 수 있는 NFS 디렉터리가 있습니다. 대상 NFS 디렉터리에는 POSIX 소유자 및 그룹 ID가 있습니다. OpenShift Container Platform NFS 플러그인은 내보낸 NFS 디렉터리에 있는 동일한 POSIX 소유권 및 권한과 함께 컨테이너의 NFS 디렉터리를 마운트합니다. 그러나 컨테이너는 원하는 동작인 NFS 마운트의 소유자와 동일한 유효 UID로 실행되지 않습니다.

예를 들어, 대상 NFS 디렉터리가 NFS 서버에 다음과 같이 표시되는 경우:

```
$ ls -lZ /opt/nfs -d
```

출력 예

```
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs
```

```
$ id nfsnobody
```

출력 예

```
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

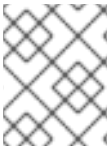
그러면 컨테이너가 SELinux 레이블과 일치하고 **65534**, **nfsnobody** 소유자 또는 디렉터리에 액세스하려면 추가 그룹의 **5555**와 함께 실행해야 합니다.

참고

65534의 소유자 ID는 예시와 같이 사용됩니다. NFS의 **root_squash**는 UID가 **0**인 **루트**를 UID가 **65534**인 **nfsnobody**로 매핑하지만, NFS 내보내기에는 임의의 소유자 ID가 있을 수 있습니다. NFS를 내보내려면 소유자 **65534**가 필요하지 않습니다.

4.11.3.1. 그룹 ID

NFS 내보내기 권한 변경 옵션이 아닐 경우 NFS 액세스를 처리하는 권장 방법은 추가 그룹을 사용하는 것입니다. OpenShift Container Platform의 추가 그룹은 공유 스토리지에 사용되며 NFS가 그 예입니다. 반면 iSCSI와 같은 블록 스토리지는 Pod의 **securityContext**에 있는 **fsGroup** SCC 전략과 **fsGroup** 값을 사용합니다.



참고

영구 스토리지에 액세스하려면, 일반적으로 추가 그룹 ID vs 사용자 ID를 사용하는 것이 좋습니다.

예제 대상 NFS 디렉터리의 그룹 ID는 **5555** 이므로 Pod의 **securityContext** 정의 아래에 **supplementalGroups** 를 사용하여 해당 그룹 ID를 정의할 수 있습니다. 예를 들면 다음과 같습니다.

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    supplementalGroups: [5555] 2
```

1 **SecurityContext**는 특정 컨테이너의 하위가 아닌 Pod 수준에서 정의해야 합니다.

2 Pod에 정의된 GID 배열입니다. 이 경우 배열에는 1개의 요소가 있습니다. 추가 GID는 쉼표로 구분됩니다.

Pod 요구사항을 충족할 수 있는 사용자 지정 SCC가 없는 경우 Pod는 **restricted** SCC와 일치할 수 있습니다. 이 SCC에는 **supplementalGroups** 전략이 **RunAsAny**로 설정되어 있으므로, 범위를 확인하지 않고 제공되는 그룹 ID가 승인됩니다.

그 결과 위의 Pod에서 승인이 전달 및 실행됩니다. 그러나 그룹 ID 범위 확인이 필요한 경우에는 사용자 지정 SCC를 사용하는 것이 좋습니다. 사용자 지정 SCC를 생성하면 최소 및 최대 그룹 ID가 정의되고, 그룹 ID 범위 확인이 적용되며, **5555** 그룹 ID가 허용될 수 있습니다.

참고

사용자 정의 SCC를 사용하려면 먼저 적절한 서비스 계정에 추가해야 합니다. 예를 들어, **Pod** 사양에 다른 값이 지정된 경우를 제외하고 지정된 프로젝트에서 **기본** 서비스 계정을 사용하십시오.

4.11.3.2. 사용자 ID

사용자 ID는 컨테이너 이미지 또는 **Pod** 정의에 정의할 수 있습니다.

참고

일반적으로 사용자 ID를 사용하는 대신 추가 그룹 ID를 사용하여 영구 스토리지에 대한 액세스 권한을 얻는 것이 좋습니다.

위에 표시된 예시 NFS 디렉터리에서 컨테이너는 UID가 **65534**로 설정되고, 현재 그룹 ID를 무시해야 하므로 다음을 **Pod** 정의에 추가할 수 있습니다.

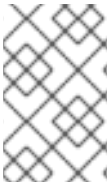
```
spec:
  containers: 1
  - name:
  ...
    securityContext:
      runAsUser: 65534 2
```

1. Pod에는 각 컨테이너에 특정 **securityContext** 정의와 Pod에 정의된 모든 컨테이너에 적용되는 Pod의 **securityContext** 정의가 포함됩니다.
2. **65534**는 **nfsnobody** 사용자입니다.

프로젝트가 **default**이고 SCC가 **restricted**라고 가정하면 Pod에서 요청한 대로 **65534**의 사용자 ID가 허용되지 않습니다. 따라서 Pod가 다음과 같은 이유로 실패합니다.

- **65534**가 사용자 ID로 요청되었습니다.
- Pod에 사용 가능한 모든 SCC를 검사하여 어떤 SCC에서 **65534** 의 사용자 ID를 허용하는지 확인합니다. SCC의 모든 정책을 확인하는 동안 여기에는 중요한 사항은 사용자 ID입니다.
- 사용 가능한 모든 SCC는 **runAsUser** 전략에서 **MustRunAsRange**를 사용하므로 UID 범위 검사가 필요합니다.
- **65534**는 SCC 또는 프로젝트의 사용자 ID 범위에 포함되어 있지 않습니다.

일반적으로 사전 정의된 SCC를 수정하지 않는 것이 좋습니다. 이 상황을 해결하기 위해 선호되는 방법은 사용자 정의 SCC를 생성하는 것입니다. 따라서 최소 및 최대 사용자 ID가 정의되고 UID 범위 검사가 여전히 적용되며, **65534**의 UID가 허용됩니다.



참고

사용자 정의 SCC를 사용하려면 먼저 적절한 서비스 계정에 추가해야 합니다. 예를 들어, **Pod** 사양에 다른 값이 지정된 경우를 제외하고 지정된 프로젝트에서 **기본** 서비스 계정을 사용하십시오.

4.11.3.3. SELinux

RHEL(Red Hat Enterprise Linux) 및 RHCOS(Red Hat Enterprise Linux CoreOS) 시스템은 기본적으로 원격 NFS 서버에서 SELinux를 사용하도록 구성됩니다.

RHEL 이외 및 비RHCOS 시스템의 경우 SELinux는 Pod에서 원격 NFS 서버로 쓰기를 허용하지 않습니다. NFS 볼륨이 올바르게 마운트되지만 읽기 전용입니다. 다음 절차에 따라 올바른 SELinux 권한을 활성화해야 합니다.

사전 요구 사항

- **container-selinux** 패키지가 설치되어 있어야 합니다. 이 패키지는 **virt_use_nfs** SELinux 부울을 제공합니다.

절차

- 다음 명령을 사용하여 **virt_use_nfs** 부울을 활성화합니다. **-P** 옵션을 사용하면 재부팅할 때 이 부울을 지속적으로 사용할 수 있습니다.

```
# setsebool -P virt_use_nfs 1
```

4.11.3.4. 내보내기 설정

컨테이너 사용자가 볼륨을 읽고 쓸 수 있도록 하려면 NFS 서버의 내보낸 각 볼륨은 다음 조건을 준수해야 합니다.

- 모든 내보내는 다음 형식을 사용하여 내보내야 합니다.

```
/<example_fs> *(rw,root_squash)
```

- 마운트 옵션으로 트래픽을 허용하도록 방화벽을 구성해야 합니다.
  - NFSv4의 경우 기본 포트 **2049**(**nfs**)를 구성합니다.

    NFSv4

```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
```

  - NFSv3의 경우 구성할 포트 3개가 있습니다. **2049** (**NFS**), **20048** (**mountd**) 및 **111** (**portmapper**).

    NFSv3

```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
```

```
# iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
```

```
# iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
```

- 대상 Pod에서 액세스할 수 있도록 NFS 내보내기 및 디렉터리를 설정해야 합니다. 컨테이너의 기본 UID에 표시된 대로 내보내기를 컨테이너의 기본 UID로 설정하거나 위 그룹 ID에 표시된 대로 **supplementalGroups**를 사용하여 Pod 그룹 액세스 권한을 제공합니다.

4.11.4. 리소스 회수

NFS는 OpenShift Container Platform **Recyclable** 플러그인 인터페이스를 구현합니다. 자동 프로세스는 각 영구 볼륨에 설정된 정책에 따라 복구 작업을 처리합니다.

기본적으로 PV는 **Retain**으로 설정됩니다.

PVC에 대한 클레임이 삭제되고 PV가 해제되면 PV 오브젝트를 재사용해서는 안 됩니다. 대신 원래 볼륨과 동일한 기본 볼륨 세부 정보를 사용하여 새 PV를 생성해야 합니다.

예를 들어, 관리자가 이름이 **nfs1**인 PV를 생성합니다.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs1
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

사용자가 **PVC1**을 생성하여 **nfs1**에 바인딩합니다. 그리고 사용자가 **PVC1**에서 **nfs1** 클레임을 해제합니다. 그러면 **nfs1**이 **Released** 상태가 됩니다. 관리자가 동일한 NFS 공유를 사용하려면 동일한 NFS 서버 세부 정보를 사용하여 새 PV를 생성해야 하지만 다른 PV 이름을 사용해야 합니다.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs2
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

원래 PV를 삭제하고 동일한 이름으로 다시 생성하는 것은 권장되지 않습니다. **Released**에서 **Available**로 PV의 상태를 수동으로 변경하면 오류가 발생하거나 데이터가 손실될 수 있습니다.

4.11.5. 추가 구성 및 문제 해결

사용되는 NFS 버전과 구성 방법에 따라 적절한 내보내기 및 보안 매핑에 필요한 추가 구성 단계가 있을 수 있습니다. 다음은 적용되는 몇 가지입니다.

NFSv4 마운트에서 소유권이 **nobody:nobody**인 모든 파일이 올바르게 표시되지 않습니다.	• 이는 NFS의 **/etc/idmapd.conf**에 있는 ID 매핑 설정으로 인한 것일 수 있습니다. • 이 Red Hat 해결책을 참조하십시오.
NFSv4에서 ID 매핑 비활성화	• NFS 클라이언트 및 서버 모두에서 다음을 실행합니다. `# echo 'Y' > /sys/module/nfsd/parameters/nfs4_disable_idmapping`

4.12. RED HAT OPENSHIFT 컨테이너 스토리지

Red Hat OpenShift Container Storage는 내부 또는 하이브리드 클라우드 환경에서 파일, 블록 및 객체 스토리지를 지원하는 OpenShift Container Platform의 영구 스토리지 제공 업체입니다. Red Hat 스토리지 솔루션인 Red Hat OpenShift Container Storage는 배포, 관리 및 모니터링을 위해 OpenShift Container Platform과 완전히 통합되어 있습니다.

Red Hat OpenShift Container Storage는 자체 문서 라이브러리를 제공합니다. Red Hat OpenShift Container Storage의 전체 문서 세트는 https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/4.7/에서 확인할 수 있습니다.

중요

OpenShift Container Platform과 함께 설치되는 가상 머신을 호스팅하는 하이퍼컨버지드 노드를 사용하는 가상화용 RHHI(Red Hat Hyperconverged Infrastructure)의 상단에 위치하는 OpenShift Container Storage는 지원되는 구성이 아닙니다. 지원되는 플랫폼에 대한 자세한 내용은 Red Hat OpenShift Container Storage 지원 및 상호 운용 가이드 를 참조하십시오.

Red Hat OpenShift Container Storage 관련 정보를 찾고 있는 경우	다음 Red Hat OpenShift Container Storage 문서 참조
계획	
새로운 알려진 문제, 중요한 버그 수정 및 기술 미리보기	Red Hat OpenShift Container Storage 4.7 릴리스 노트
지원되는 워크로드, 레이아웃, 하드웨어 및 소프트웨어 요구 사항, 크기 조정 및 확장 권장 사항	Red Hat OpenShift Container Storage 4.7 배포 계획
배포	
로컬 또는 클라우드 스토리지용 Amazon Web Services를 사용하여 OpenShift Container Storage 배포	Deploying OpenShift Container Storage 4.7 using Amazon Web Services
베어메탈 인프라의 로컬 스토리지에 OpenShift Container Storage 배포	Deploying OpenShift Container Storage 4.7 using bare metal infrastructure
외부 Red Hat Ceph Storage 클러스터를 사용하기 위해 OpenShift Container Storage 배포	Deploying OpenShift Container Storage 4.7 in external mode
기존 Google Cloud 클러스터에서 Red Hat OpenShift Container Storage 배포 및 관리	Deploying and managing OpenShift Container Storage 4.7 using Google Cloud
IBM Z 인프라에서 로컬 스토리지를 사용하도록 Red Hat OpenShift Container Storage 배포	Deploying OpenShift Container Storage using IBM Z
IBM Power Systems에서 Red Hat OpenShift Container Storage 배포	Deploying OpenShift Container Storage using IBM Power Systems
IBM Cloud에 Red Hat OpenShift Container Storage 배포	Deploying OpenShift Container Storage using IBM Cloud
RHOSP(Red Hat OpenStack Platform)에서 Red Hat OpenShift Container Storage 배포 및 관리	Deploying and managing OpenShift Container Storage 4.7 using Red Hat OpenStack Platform
RHV(Red Hat Virtualization)에서 Red Hat OpenShift Container Storage 배포 및 관리	Deploying and managing OpenShift Container Storage 4.7 using Red Hat Virtualization Platform

Red Hat OpenShift Container Storage 관련 정보를 찾고 있는 경우	다음 Red Hat OpenShift Container Storage 문서 참조
VMware vSphere 클러스터에 Red Hat OpenShift Container Storage 배포	Deploying OpenShift Container Storage 4.7 on VMware vSphere
Red Hat OpenShift Container Storage를 최신 버전으로 업데이트	Updating OpenShift Container Storage
관리	
스냅샷 및 복제를 포함하여 Red Hat OpenShift Container Storage의 코어 서비스 및 호스팅되는 애플리케이션에 스토리지 할당	Managing and allocating resources
Multicloud Object Gateway(NooBaa)를 사용하여 하이브리드 클라우드 또는 다중 클라우드 환경에서 스토리지 리소스 관리	Managing hybrid and multicloud resources
Red Hat OpenShift Container Storage용 스토리지 장치의 안전한 교체	Replacing devices
Red Hat OpenShift Container Storage 클러스터에서의 안전한 노드 교체	Replacing nodes
Red Hat OpenShift Container Storage에서 작업 확장	Scaling storage
Red Hat OpenShift Container Storage 4.7 클러스터 모니터링	Monitoring OpenShift Container Storage 4.7
오류 및 문제 해결	Troubleshooting OpenShift Container Storage 4.7
OpenShift Container Platform 클러스터를 버전 3에서 버전 4로 마이그레이션	Migration Toolkit for Containers

4.13. VMWARE VSPHERE 볼륨을 사용하는 영구 스토리지

OpenShift Container Platform을 사용하면 VMware vSphere의 VMDK(가상 머신 디스크) 볼륨을 사용할 수 있습니다. VMware vSphere를 사용하여 영구 스토리지로 OpenShift Container Platform 클러스터를 프로비저닝할 수 있습니다. Kubernetes 및 VMware vSphere에 대해 어느 정도 익숙한 것으로 가정합니다.

VMware vSphere 볼륨은 동적으로 프로비저닝할 수 있습니다. OpenShift Container Platform은 vSphere에서 디스크를 생성하고 이 디스크를 올바른 이미지에 연결합니다.

참고

OpenShift Container Platform은 새 볼륨을 독립 영구 디스크로 프로비저닝하여 클러스터의 모든 노드에서 볼륨을 자유롭게 연결 및 분리합니다. 결과적으로 스냅샷을 사용하는 볼륨을 백업하거나 스냅샷에서 볼륨을 복원할 수 없습니다. 자세한 내용은 스냅 샷 제한 을 참조하십시오.

Kubernetes 영구 볼륨 프레임워크를 사용하면 관리자는 영구 스토리지로 클러스터를 프로비저닝하고 사용자가 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다.

영구 볼륨은 단일 프로젝트 또는 네임스페이스에 바인딩되지 않으며, OpenShift Container Platform 클러스터에서 공유할 수 있습니다. 영구 볼륨 클레임은 프로젝트 또는 네임스페이스에 고유하며 사용자가 요청할 수 있습니다.

추가 리소스

- VMware vSphere

4.13.1. 동적으로 VMware vSphere 볼륨 프로비저닝

동적으로 VMware vSphere 볼륨 프로비저닝은 권장되는 방법입니다.

4.13.2. 사전 요구 사항

- 사용하는 구성 요소의 요구사항을 충족하는 VMware vSphere 버전 6 인스턴스에 OpenShift Container Platform 클러스터가 설치되어 있어야 합니다. vSphere 버전 지원에 대한 자세한 내용은 vSphere에 클러스터 설치 를 참조하십시오.

다음 절차 중 하나를 사용하여 기본 스토리지 클래스를 사용하여 이러한 볼륨을 동적으로 프로비저닝할 수 있습니다.

4.13.2.1. UI를 사용하여 VMware vSphere 볼륨을 동적으로 프로비저닝

OpenShift Container Platform은 볼륨을 프로비저닝하기 위해 **thin** 디스크 형식을 사용하는 이름이 **thin**인 기본 스토리지 클래스를 설치합니다.

사전 요구 사항

- OpenShift Container Platform에서 볼륨으로 마운트하기 전에 기본 인프라에 스토리지가 있어야 합니다.

절차

1. OpenShift Container Platform 콘솔에서 **스토리지 → 영구 볼륨 클레임**을 클릭합니다.
2. 영구 볼륨 클레임 생성 개요에서 **영구 볼륨 클레임 생성**을 클릭합니다.
3. 결과 페이지에 필요한 옵션을 정의합니다.
   a. **thin** 스토리지 클래스를 선택합니다.
   b. 스토리지 클레임의 고유한 이름을 입력합니다.
   c. 액세스 모드를 선택하여 생성된 스토리지 클레임에 대한 읽기 및 쓰기 액세스를 결정합니다.

d. 스토리지 클레임의 크기를 정의합니다.

4. **만들기**를 클릭하여 영구 볼륨 클레임을 생성하고 영구 볼륨을 생성합니다.

4.13.2.2. CLI를 사용하여 VMware vSphere 볼륨을 동적으로 프로비저닝

OpenShift Container Platform은 볼륨 프로비저닝에 **thin** 디스크 형식을 사용하는 이름이 **thin**인 기본 StorageClass를 설치합니다.

사전 요구 사항

- OpenShift Container Platform에서 볼륨으로 마운트하기 전에 기본 인프라에 스토리지가 있어야 합니다.

절차(CLI)

1. 다음 콘텐츠와 함께 **pvc.yaml** 파일을 생성하여 VMware vSphere PersistentVolumeClaim을 정의할 수 있습니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc 1
spec:
  accessModes:
  - ReadWriteOnce 2
  resources:
    requests:
      storage: 1Gi 3
```

1. 영구 볼륨 오브젝트를 표시하는 고유한 이름입니다.
2. 영구 볼륨 클레임의 액세스 모드입니다. **ReadWriteOnce**를 사용하면 단일 노드에서 읽기 및 쓰기 권한으로 볼륨을 마운트할 수 있습니다.
3. 영구 볼륨 클레임의 크기입니다.

2. 파일에서 **PersistentVolumeClaim** 오브젝트를 만듭니다.

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

4.13.3. 정적으로 프로비저닝 VMware vSphere 볼륨

VMware vSphere 볼륨을 정적으로 프로비저닝하려면 영구 볼륨 프레임워크를 참조하기 위해 가상 머신 디스크를 생성해야 합니다.

사전 요구 사항

- OpenShift Container Platform에서 볼륨으로 마운트하기 전에 기본 인프라에 스토리지가 있어야 합니다.

절차

1. 가상 머신 디스크를 생성합니다. VMDK(가상 머신 디스크)는 정적으로 VMware vSphere 볼륨을 프로비저닝하기 전에 수동으로 생성해야 합니다. 다음 방법 중 하나를 사용합니다.

   - **vmkfstools**를 사용하여 생성합니다. SSH(Secure Shell)를 통해 ESX에 액세스한 후 다음 명령을 사용하여 VMDK 볼륨을 생성합니다.

     ```
     $ vmkfstools -c <size> /vmfs/volumes/<datastore-name>/volumes/<disk-name>.vmdk
     ```

   - **vmware-diskmanager**를 사용하여 생성합니다.

     ```
     $ shell vmware-vdiskmanager -c -t 0 -s <size> -a lsilogic <disk-name>.vmdk
     ```

2. VMDK를 참조하는 영구 볼륨을 생성합니다. **PersistentVolume** 오브젝트 정의를 사용하여 **pv1.yaml** 파일을 생성합니다.

   ```
   apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: pv1 (1)
   spec:
     capacity:
       storage: 1Gi (2)
     accessModes:
       - ReadWriteOnce
     persistentVolumeReclaimPolicy: Retain
     vsphereVolume: (3)
       volumePath: "[datastore1] volumes/myDisk" (4)
       fsType: ext4 (5)
   ```

   (1) 볼륨의 이름입니다. 이 이름을 사용하여 영구 볼륨 클레임 또는 Pod에 의해 식별됩니다.

   (2) 이 볼륨에 할당된 스토리지의 용량입니다.

   (3) vSphere 볼륨을 위해 **vsphereVolume**과 함께 사용되는 볼륨 유형입니다. 레이블은 vSphere VMDK 볼륨을 Pod에 마운트하는 데 사용됩니다. 볼륨의 내용은 마운트 해제 시 보존됩니다. 볼륨 유형은 VMFS 및 VSAN 데이터 저장소를 지원합니다.

   (4) 사용할 기존 VMDK 볼륨입니다. **vmkfstools**를 사용하는 경우 이전에 설명된 대로 볼륨 정의에서 데이터 저장소의 이름을 대괄호 **[]**로 감싸야 합니다.

   (5) 마운트할 파일 시스템 유형입니다. 예를 들어, ext4, xfs 또는 기타 파일 시스템이 이에 해당합니다.

   중요

   볼륨이 포맷 및 프로비저닝된 후 fsType 매개변수 값을 변경하면 데이터가 손실되거나 Pod 오류가 발생할 수 있습니다.

3. 파일에서 **PersistentVolume** 오브젝트를 만듭니다.

   ```
   $ oc create -f pv1.yaml
   ```

4. 이전 단계에서 생성한 영구 볼륨 클레임에 매핑되는 영구 볼륨 클레임을 생성합니다. **PersistentVolumeClaim** 오브젝트 정의를 사용하여 **pvc1.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1 (1)
spec:
  accessModes:
    - ReadWriteOnce (2)
  resources:
   requests:
     storage: "1Gi" (3)
  volumeName: pv1 (4)
```

(1) 영구 볼륨 오브젝트를 표시하는 고유한 이름입니다.

(2) 영구 볼륨 클레임의 액세스 모드입니다. ReadWriteOnce를 사용하면 단일 노드에서 읽기 및 쓰기 권한으로 볼륨을 마운트할 수 있습니다.

(3) 영구 볼륨 클레임의 크기입니다.

(4) 기존 영구 볼륨의 이름입니다.

5. 파일에서 **PersistentVolumeClaim** 오브젝트를 만듭니다.

```
$ oc create -f pvc1.yaml
```

4.13.3.1. VMware vSphere 볼륨 포맷

OpenShift Container Platform이 볼륨을 마운트하고 컨테이너에 전달하기 전에 **PersistentVolume**(PV) 정의에 있는 **fsType** 매개변수 값에 의해 지정된 파일 시스템이 볼륨에 포함되어 있는지 확인합니다. 장치가 파일 시스템으로 포맷되지 않으면 장치의 모든 데이터가 삭제되고 장치는 지정된 파일 시스템에서 자동으로 포맷됩니다.

OpenShift Container Platform은 처음 사용하기 전에 포맷되므로 vSphere 볼륨을 PV로 사용할 수 있습니다.

5장. CSI(CONTAINER STORAGE INTERFACE) 사용

5.1. CSI 볼륨 구성

CSI(Container Storage Interface)를 사용하면 OpenShift Container Platform이 CSI 인터페이스를 영구 스토리지로 구현하는 스토리지 백엔드에서 스토리지를 사용할 수 있습니다.

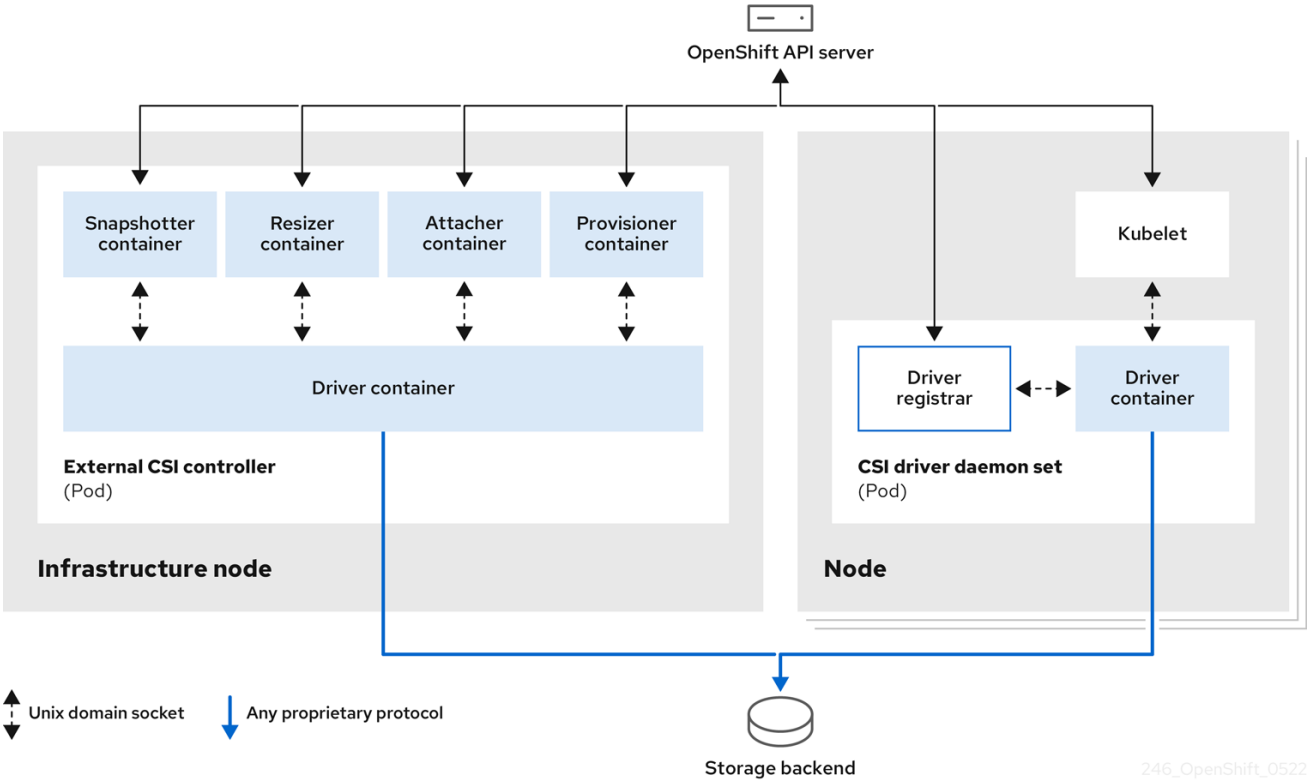
참고

OpenShift Container Platform 4.7은 CSI 사양 의 버전 1.2.0을 지원합니다.

5.1.1. CSI 아키텍처

CSI 드라이버는 일반적으로 컨테이너 이미지로 제공됩니다. 이러한 컨테이너는 OpenShift Container Platform이 실행되는 위치를 인식하지 못합니다. OpenShift Container Platform에서 CSI 호환 스토리지 백엔드를 사용하려면 클러스터 관리자가 OpenShift Container Platform과 스토리지 드라이버 간의 브리지 역할을 하는 여러 구성 요소를 배포해야 합니다.

다음 다이어그램에서는 OpenShift Container Platform 클러스터의 Pod에서 실행되는 구성 요소에 대한 상위 수준 개요를 설명합니다.



다른 스토리지 백엔드에 대해 여러 CSI 드라이버를 실행할 수 있습니다. 각 드라이버에는 드라이버 및 CSI 등록 기관과 함께 자체 외부 컨트롤러 배포 및 데몬 세트가 필요합니다.

5.1.1.1. 외부 CSI 컨트롤러

외부 CSI 컨트롤러는 5개의 컨테이너가 있는 하나 이상의 Pod를 배포하는 배포입니다.

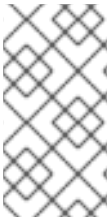
- snapshotter 컨테이너는 **VolumeSnapshot** 및 **VolumeSnapshotContent** 오브젝트를 감시하고 **VolumeSnapshotContent** 오브젝트를 생성 및 삭제합니다.

- resizer 컨테이너는 **PersistentVolumeClaim** 업데이트를 감시하고 **PersistentVolumeClaim** 오브젝트에 더 많은 스토리지를 요청하는 경우 CSI 엔드 포인트에 대해 **ControllerinspectorVolume** 작업을 트리거하는 사이드카 컨테이너입니다.
- 외부 CSI 연결 컨테이너는 OpenShift Container Platform의 **attach** 및 **detach** 호출을 CSI 드라이버에 대한 각 **ControllerPublish** 및 **ControllerUnpublish** 호출로 변환합니다.
- OpenShift Container Platform으로부터의 **provision** 및 **delete** 호출을 CSI 드라이버의 해당 **CreateVolume** 및 **DeleteVolume** 호출로 변환하는 외부 CSI 프로비저너 컨테이너입니다.
- CSI 드라이버 컨테이너

CSI 연결 및 CSI 프로비저너 컨테이너는 UNIX 도메인 소켓을 사용해 CSI 드라이버 컨테이너와 통신하여 CSI 통신이 Pod를 종료하지 않도록 합니다. Pod 외부에서 CSI 드라이버에 액세스할 수 없습니다.

참고

attach, **분리**, **provision** 및 **삭제** 작업에는 일반적으로 스토리지 백엔드에 인증 정보를 사용하기 위해 CSI 드라이버가 필요합니다. 컴퓨팅 노드의 심각한 보안 위반 시 인증 정보가 사용자 프로세스에 유출되지 않도록 인프라 노드에서 CSI 컨트롤러 Pod를 실행합니다.



참고

외부 연결에서는 타사 **attach** 또는 **detach** 작업을 지원하지 않는 CSI 드라이버에서도 실행해야 합니다. 외부 연결은 CSI 드라이버에 **ControllerPublish** 또는 **ControllerUnpublish** 작업을 발행하지 않습니다. 그러나 필요한 OpenShift Container Platform 연결 API를 구현하려면 실행해야 합니다.

5.1.1.2. CSI 드라이버 데몬 세트

CSI 드라이버 데몬 세트는 OpenShift Container Platform이 CSI 드라이버에서 제공한 스토리지를 노드에 마운트하고 사용자 워크로드(Pod)에서 PV(영구 볼륨)로 사용할 수 있는 모든 노드에서 Pod를 실행합니다. CSI 드라이버가 설치된 Pod에는 다음 컨테이너가 포함되어 있습니다.

- CSI 드라이버 등록 기관. CSI 드라이버를 노드에서 실행 중인 **openshift-node** 서비스에 등록합니다. 노드에서 실행되는 **openshift-node** 프로세스는 노드에서 사용 가능한 UNIX 도메인 소켓을 사용하여 CSI 드라이버와 직접 연결합니다.
- CSI 드라이버.

노드에 배포된 CSI 드라이버는 스토리지 백엔드에 최대한 적은 수의 인증 정보가 있어야 합니다. 이러한 호출이 구현된 경우 OpenShift Container Platform은 **NodePublish**/**NodeUnpublish** 및 **NodeStage**/**NodeUnstage**와 같은 CSI 호출의 노드 플러그인 세트만 사용합니다.

5.1.2. OpenShift Container Platform에서 지원되는 CSI 드라이버

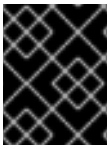
OpenShift Container Platform은 기본적으로 특정 CSI 드라이버를 설치하여 in-tree 볼륨 플러그인에서 사용할 수 없는 사용자 스토리지 옵션을 제공합니다.

지원되는 스토리지 자산에 마운트된 CSI 프로비저닝 영구 볼륨을 생성하기 위해 OpenShift Container Platform은 기본적으로 필요한 CSI 드라이버 Operator, CSI 드라이버 및 필요한 스토리지 클래스를 설치합니다. Operator 및 드라이버의 기본 네임스페이스에 대한 자세한 내용은 특정 CSI Driver Operator 설명서를 참조하십시오.

다음 표에는 OpenShift Container Platform과 함께 설치되는 CSI 드라이버와 볼륨 스냅샷, 복제 및 크기와 같은 CSI 기능을 설명합니다.

표 5.1. OpenShift Container Platform에서 지원되는 CSI 드라이버 및 기능

CSI 드라이버	CSI 볼륨 스냅샷	CSI 복제	CSI 크기 조정
AWS EBS(기술 프리뷰)	▮	-	▮
GCP(Google Cloud Platform) PD(영구 디스크)(기술 프리뷰)	▮	-	▮
OpenStack Cinder	▮	▮	▮
OpenShift Container Storage	▮	▮	▮
OpenStack Manila	▮	-	-
Red Hat Virtualization(oVirt)	-	-	-



중요

CSI 드라이버가 이전 표에 나열되어 있지 않은 경우 지원되는 CSI 기능을 사용하려면 CSI 스토리지 벤더가 제공한 설치 지침을 따라야 합니다.

5.1.3. 동적 프로비저닝

영구 스토리지의 동적 프로비저닝은 CSI 드라이버 및 기본 스토리지 백엔드의 기능에 따라 달라집니다. CSI 드라이버 공급자는 OpenShift Container Platform에서 스토리지 클래스와 구성에 사용 가능한 매개변수를 생성하는 방법을 설명해야 합니다.

동적 프로비저닝을 사용하도록 생성된 스토리지 클래스를 구성할 수 있습니다.

절차

- 설치된 CSI 드라이버에서 특수 스토리지 클래스가 필요하지 않은 모든 PVC를 프로비저닝하는 기본 스토리지 클래스를 생성합니다.

```
# oc create -f - << EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class> 1
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: <provisioner-name> 2
parameters:
EOF
```

1. 생성할 스토리지 클래스의 이름입니다.
2. 설치된 CSI 드라이버의 이름입니다.

5.1.4. CSI 드라이버 사용 예

다음 예시는 템플릿을 변경하지 않고 기본 MySQL 템플릿을 설치합니다.

사전 요구 사항

- CSI 드라이버가 배포되었습니다.
- 동적 프로비저닝을 위해 스토리지 클래스가 생성되었습니다.

절차

- MySQL 템플릿을 생성합니다.

```
# oc new-app mysql-persistent
```

출력 예

```
--> Deploying template "openshift/mysql-persistent" to project default
...
```

```
# oc get pvc
```

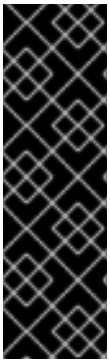
출력 예

```
NAME            STATUS    VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
mysql           Bound     kubernetes-dynamic-pv-3271ffcb4e1811e8   1Gi
RWO            cinder        3s
```

5.2. CSI 인라인 임시 볼륨

CSI(Container Storage Interface) 인라인 임시 볼륨을 사용하여 **Pod**가 배포되고 Pod가 제거될 때 인라인 임시 볼륨을 생성하는 Pod 사양을 정의할 수 있습니다.

이 기능은 지원되는 CSI(Container Storage Interface) 드라이버에서만 사용할 수 있습니다.



중요

CSI 인라인 임시 볼륨은 기술 프리뷰 기능 전용입니다. Technology Preview 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 https://access.redhat.com/support/offerings/techpreview/를 참조하십시오.

5.2.1. CSI 인라인 임시 볼륨 개요

일반적으로 CSI(Container Storage Interface) 드라이버에서 지원하는 볼륨은 **PersistentVolume** 및 **PersistentVolumeClaim** 오브젝트 조합에서만 사용할 수 있습니다.

이 기능을 사용하면 **PersistentVolume** 오브젝트가 아닌 **Pod** 사양에 직접 CSI 볼륨을 지정할 수 있습니다. 인라인 볼륨은 임시 볼륨이며 Pod를 다시 시작하면 유지되지 않습니다.

5.2.1.1. 지원 제한

기본적으로 OpenShift Container Platform에서는 다음과 같은 제한이 있는 CSI 인라인 임시 볼륨을 지원합니다.

- CSI 드라이버에서만 지원을 사용할 수 있습니다. In-tree 및 FlexVolumes는 지원되지 않습니다.
- OpenShift Container Platform에는 CSI 드라이버가 포함되어 있지 않습니다. 커뮤니티 또는 스토리지 벤더가 제공하는 CSI 드라이버를 사용하십시오. CSI 드라이버에서 제공하는 설치 지침을 따르십시오.
- CSI 드라이버는 **임시** 용량을 포함하여 인라인 볼륨 기능을 구현하지 못할 수 있습니다. 자세한 내용은 CSI 드라이버 설명서를 참조하십시오.

5.2.2. Pod 사양에 CSI 인라인 임시 볼륨 포함

OpenShift Container Platform의 **Pod** 사양에 CSI 인라인 임시 볼륨을 포함할 수 있습니다. 런타임 시 중첩된 인라인 볼륨은 관련 Pod의 임시 라이프사이클을 따라 CSI 드라이버가 Pod를 생성 및 삭제할 때 볼륨 작업의 모든 단계를 처리합니다.

절차

1. **Pod** 오브젝트 정의를 생성하여 파일에 저장합니다.
2. 파일에 CSI 인라인 임시 볼륨을 삽입합니다.

my-csi-app.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: my-csi-app
spec:
  containers:
    - name: my-frontend
      image: busybox
      volumeMounts:
      - mountPath: "/data"
        name: my-csi-inline-vol
      command: [ "sleep", "1000000" ]
  volumes: 1
    - name: my-csi-inline-vol
      csi:
        driver: inline.storage.kubernetes.io
        volumeAttributes:
          foo: bar
```

1 풀에서 사용되는 볼륨의 이름입니다.

3. 이전 단계에서 저장한 오브젝트 정의 파일을 생성합니다.

```
$ oc create -f my-csi-app.yaml
```

5.3. CSI 볼륨 스냅샷

이 문서에서는 지원되는 CSI(Container Storage Interface) 드라이버로 볼륨 스냅샷을 사용하여 OpenShift Container Platform의 데이터 손실을 보호하는 방법을 설명합니다. 영구 볼륨에 대해 숙지하는 것이 좋습니다.

5.3.1. CSI 볼륨 스냅샷 개요

*스냅샷*은 특정 시점에서 클러스터의 스토리지 볼륨 상태를 나타냅니다. 볼륨 스냅샷을 사용하여 새 볼륨을 프로비저닝할 수 있습니다.

OpenShift Container Platform은 기본적으로 CSI 볼륨 스냅샷을 지원합니다. 그러나 특정 CSI 드라이버가 필요합니다.

CSI 볼륨 스냅샷을 사용하면 클러스터 관리자가 다음을 수행할 수 있습니다.

- 스냅샷을 지원하는 타사 CSI 드라이버를 배포합니다.
- 기존 볼륨 스냅샷에서 새 PVC(영구 볼륨 클레임)를 생성합니다.
- 기존 PVC의 스냅샷을 가져옵니다.
- 스냅샷을 다른 PVC로 복원합니다.
- 기존 VM 스냅샷을 삭제합니다.

CSI 볼륨 스냅샷을 사용하면 앱 개발자가 다음을 수행할 수 있습니다.

- 애플리케이션 또는 클러스터 수준의 스토리지 백업 솔루션을 개발하기 위한 볼륨 스냅샷을 빌드 블록으로 사용할 수 있습니다.
- 빠르게 이전 개발 버전으로 롤백할 수 있습니다.
- 시간마다 전체 복사를 하지 않고 스토리지를 보다 효율적으로 사용할 수 있습니다.

볼륨 스냅샷을 사용할 때는 다음에 유의하십시오.

- CSI 드라이버에서만 지원을 사용할 수 있습니다. In-tree 및 FlexVolumes는 지원되지 않습니다.
- OpenShift Container Platform은 일부 CSI 드라이버만 제공됩니다. OpenShift Container Platform Driver Operator가 제공하지 않는 CSI 드라이버의 경우 커뮤니티 또는 스토리지 공급 업체에서 제공하는 CSI 드라이버를 사용하는 것이 좋습니다. CSI 드라이버에서 제공하는 설치 지침을 따르십시오.
- CSI 드라이버는 볼륨 스냅샷 기능을 구현하거나 사용하지 않을 수 있습니다. 볼륨 스냅샷을 지원하는 CSI 드라이버는 **csi-external-snapshotter** 사이드카를 사용할 수 있습니다. 자세한 내용은 CSI 드라이버에서 제공하는 설명서를 참조하십시오.

5.3.2. CSI 스냅샷 컨트롤러 및 사이드카

OpenShift Container Platform은 컨트롤 플레인에 배포된 스냅샷 컨트롤러를 제공합니다. 또한, CSI 드라이버 벤더는 CSI 드라이버 설치 중에 설치된 Helper 컨테이너로 CSI 스냅샷 사이드카를 제공합니다.

CSI 스냅샷 컨트롤러 및 사이드카는 OpenShift Container Platform API를 통해 볼륨 스냅샷을 제공합니다. 이러한 외부 구성 요소는 클러스터에서 실행됩니다.

외부 컨트롤러는 CSI Snapshot Controller Operator에 의해 배포됩니다.

5.3.2.1. 외부 컨트롤러

CSI 스냅샷 컨트롤러는 **VolumeSnapshot** 및 **VolumeSnapshotContent** 오브젝트를 바인딩합니다. 컨트롤러는 **VolumeSnapshotContent** 오브젝트를 생성 및 삭제하여 동적 프로비저닝을 관리합니다.

5.3.2.2. 외부 사이드카

CSI 드라이버 벤더는 **csi-external-snapshotter** 사이드카를 제공합니다. 이 컨테이너는 CSI 드라이버와 함께 배포된 별도의 Helper 컨테이너입니다. 사이드카는 **CreateSnapshot** 및 **DeleteSnapshot** 작업을 트리거하여 스냅샷을 관리합니다. 벤더가 제공하는 설치 지침을 따르십시오.

5.3.3. CSI Snapshot Controller Operator 정보

CSI Snapshot Controller Operator는 **openshift-cluster-storage-operator** 네임스페이스에서 실행됩니다. 기본적으로 모든 클러스터에 CVO(Cluster Version Operator)에 의해 설치됩니다.

CSI Snapshot Controller Operator는 **openshift-cluster-storage-operator** 네임스페이스에서 실행되는 CSI 스냅샷 컨트롤러를 설치합니다.

5.3.3.1. 볼륨 스냅샷 CRD

OpenShift Container Platform을 설치하는 동안 CSI Snapshot Controller Operator는 **snapshot.storage.k8s.io/v1** API 그룹에 다음 스냅샷 CRD(사용자 정의 리소스 정의)를 생성합니다.

VolumeSnapshotContent

클러스터 관리자가 프로비저닝한 클러스터의 볼륨으로 가져온 스냅샷입니다.
PersistentVolume 오브젝트와 유사하게 **VolumeSnapshotContent** CRD는 스토리지 백엔드의 실제 스냅샷을 가리키는 클러스터 리소스입니다.

수동으로 프로비저닝된 스냅샷의 경우 클러스터 관리자는 여러 **VolumeSnapshotContent** CRD를 생성합니다. 이는 스토리지 시스템의 실제 볼륨 스냅샷에 대한 세부 정보를 제공합니다.

VolumeSnapshotContent CRD는 네임스페이스가 제공되지 않으며 클러스터 관리자가 사용합니다.

VolumeSnapshot

PersistentVolumeClaim 오브젝트와 유사하게 **VolumeSnapshot CRD는** 스냅샷에 대한 개발자 요청을 정의합니다. CSI Snapshot Controller Operator는 CSI 스냅샷 컨트롤러를 실행하여 **VolumeSnapshot** CRD의 바인딩을 적절한 **VolumeSnapshotContent** CRD로 처리합니다. 바인딩은 일대일 매핑입니다.
VolumeSnapshot CRD에는 네임스페이스가 지정됩니다. 개발자는 CRD를 스냅샷에 대한 개별 요청으로 사용합니다.

VolumeSnapshotClass

클러스터 관리자는 **VolumeSnapshot** 오브젝트에 속한 다른 속성을 지정할 수 있습니다. 이러한 속성은 스토리지 시스템에서 동일한 볼륨에서 가져온 스냅샷과 다를 수 있으며, 이 경우 영구 볼륨 클레임의 동일한 스토리지 클래스를 사용하여 표시하지 않을 수 있습니다.
VolumeSnapshotClass CRD는 스냅샷을 생성할 때 사용할 **csi-external-snapshotter** 사이드카에 대한 매개변수를 정의합니다. 이를 통해 여러 옵션이 지원되는 경우 스토리지 백엔드가 어떤 종류의 스냅샷을 동적으로 생성할지 알 수 있습니다.

동적으로 프로비저닝된 스냅샷은 **VolumeSnapshotClass** CRD를 사용하여 스냅샷을 생성할 때 사용할 스토리지에서 제공되는 특정 매개변수를 지정합니다.

VolumeSnapshotContentClass CRD에는 네임스페이스가 지정되지 않으며, 클러스터 관리자가 사용하여 스토리지 백엔드에 대한 글로벌 구성 옵션을 활성화하기 위한 용도입니다.

5.3.4. 볼륨 스냅샷 프로비저닝

스냅샷을 프로비저닝하는 방법은 동적으로 수동의 두 가지가 있습니다.

5.3.4.1. 동적 프로비저닝

기존 스냅샷을 사용하는 대신 영구 볼륨 클레임에서 스냅샷을 동적으로 수락하도록 요청할 수 있습니다. 매개변수는 **VolumeSnapshotClass** CRD를 사용하여 지정됩니다.

5.3.4.2. 수동 프로비저닝

클러스터 관리자는 여러 **VolumeSnapshotContent** 오브젝트를 수동으로 사전 프로비저닝할 수 있습니다. 이를 통해 클러스터 사용자가 사용할 수 있는 실제 볼륨 스냅샷 세부 정보가 제공됩니다.

5.3.5. 볼륨 스냅샷 생성

VolumeSnapshot 오브젝트를 생성할 때 OpenShift Container Platform은 볼륨 스냅샷을 생성합니다.

사전 요구 사항

- 실행 중인 OpenShift Container Platform 클러스터에 로그인합니다.
- **VolumeSnapshot** 오브젝트를 지원하는 CSI 드라이버를 사용하여 생성된 PVC입니다.
- 스토리지 백엔드를 프로비저닝하는 스토리지 클래스입니다.
- 스냅샷을 사용하려는 PVC(영구 볼륨 클레임)를 사용하는 Pod가 없습니다.

참고

Pod가 이를 사용하는 경우 PVC의 볼륨 스냅샷을 생성하지 마십시오. 그러면 PVC가 quiesced(일시 중지됨)되지 않기 때문에 데이터 손상이 발생할 수 있습니다. 일관된 스냅샷을 유지하기 위해 실행 중인 Pod를 먼저 제거해야 합니다.

절차

볼륨 스냅샷을 동적으로 생성하려면 다음을 수행합니다.

1. 다음 YAML로 설명된 **VolumeSnapshotClass** 오브젝트로 파일을 생성합니다.

   volumesnapshotclass.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-hostpath-snap
driver: hostpath.csi.k8s.io (1)
deletionPolicy: Delete
```

(1) 이 **VolumeSnapshotClass** 오브젝트의 스냅샷을 생성하는 데 사용되는 CSI 드라이버의 이름입니다. 이름은 스냅샷을 수행 중인 PVC를 담당하는 스토리지 클래스의 **Provisioner** 필드와 동일해야 합니다.

2. 다음 명령을 입력하여 이전 단계에서 저장한 오브젝트를 생성합니다.

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

3. **VolumeSnapshot** 오브젝트를 생성합니다.

volumesnapshot-dynamic.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: mysnap
spec:
  volumeSnapshotClassName: csi-hostpath-snap (1)
  source:
    persistentVolumeClaimName: myclaim (2)
```

(1) 볼륨 스냅샷의 특정 클래스에 대한 요청입니다. **volumeSnapshotClassName** 설정이 없으며 기본 볼륨 스냅샷 클래스가 있는 경우 기본 볼륨 스냅샷 클래스 이름을 사용하여 스냅샷이 생성됩니다. 그러나 필드가 없으며 기본 볼륨 스냅샷 클래스가 없는 경우에는 스냅샷이 생성되지 않습니다.

(2) 영구 볼륨에 바인딩된 **PersistentVolumeClaim** 오브젝트의 이름입니다. 이 명령은 스냅샷을 생성할 대상을 정의합니다. 스냅샷을 동적으로 프로비저닝하는 데 필요합니다.

4. 다음 명령을 입력하여 이전 단계에서 저장한 오브젝트를 생성합니다.

```
$ oc create -f volumesnapshot-dynamic.yaml
```

스냅샷을 수동으로 프로비저닝하려면 다음을 수행합니다.

1. 위에서 설명한 볼륨 스냅샷 클래스를 정의하는 것 외에도 **volumeSnapshotContentName** 매개변수 값을 스냅샷의 소스로 제공해야 합니다.

volumesnapshot-manual.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: snapshot-demo
```

```
spec:
  source:
    volumeSnapshotContentName: mycontent (1)
```

(1) 사전 프로비저닝된 스냅샷에는 **volumeSnapshotContentName** 매개변수가 필요합니다.

2. 다음 명령을 입력하여 이전 단계에서 저장한 오브젝트를 생성합니다.

```
$ oc create -f volumesnapshot-manual.yaml
```

검증

클러스터에서 스냅샷을 생성한 후 스냅샷에 대한 추가 세부 정보를 사용할 수 있습니다.

1. 생성된 볼륨 스냅샷에 대한 세부 정보를 표시하려면 다음 명령을 입력합니다.

```
$ oc describe volumesnapshot mysnap
```

다음 예시는 **mysnap** 볼륨 스냅샷에 대한 세부 정보를 표시합니다.

volumesnapshot.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: mysnap
spec:
  source:
    persistentVolumeClaimName: myclaim
  volumeSnapshotClassName: csi-hostpath-snap
status:
  boundVolumeSnapshotContentName: snapcontent-1af4989e-a365-4286-96f8-
d5dcd65d78d6 (1)
  creationTime: "2020-01-29T12:24:30Z" (2)
  readyToUse: true (3)
  restoreSize: 500Mi
```

(1) 컨트롤러가 생성한 실제 스토리지 콘텐츠의 포인터입니다.

(2) 스냅샷이 생성된 시간입니다. 스냅샷에는 이 표시된 시점에서 사용 가능한 볼륨 내용이 포함되어 있습니다.

(3) 값을 **true**로 설정하면 스냅샷을 사용하여 새 PVC로 복원할 수 있습니다.
값을 **false**로 설정하면 스냅샷이 생성됩니다. 하지만 스토리지 백엔드는 스냅샷을 사용할 수 있도록 추가 작업을 수행하여 새 볼륨으로 복원해야 합니다. 예를 들어, Amazon Elastic Block Store 데이터를 다른 저렴한 위치로 이동할 수 있으며, 이 작업에는 몇 분이 걸릴 수 있습니다.

2. 볼륨 스냅샷이 생성되었는지 확인하려면 다음 명령을 입력합니다.

```
$ oc get volumesnapshotcontent
```

실제 컨텐츠에 대한 포인터가 표시됩니다. **boundVolumeSnapshotContentName** 필드가 입력된 경우 **VolumeSnapshotContent** 오브젝트가 존재하고 스냅샷이 생성된 것입니다.

3. 스냅샷이 준비되었는지 확인하려면 **VolumeSnapshot** 오브젝트에 **readyToUse: true**가 있는지 확인합니다.

5.3.6. 볼륨 스냅샷 삭제

OpenShift Container Platform이 볼륨 스냅샷을 삭제하는 방법을 구성할 수 있습니다.

절차

1. 다음 예와 같이 **VolumeSnapshotClass** 오브젝트에 필요한 삭제 정책을 지정합니다.

volumesnapshotclass.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-hostpath-snap
driver: hostpath.csi.k8s.io
deletionPolicy: Delete (1)
```

(1) 볼륨 스냅샷을 삭제할 때 **Delete** 값이 설정되면 **VolumeSnapshotContent** 오브젝트와 함께 기본 스냅샷이 삭제됩니다. **Retain** 값이 설정된 경우 기본 스냅샷 및 **VolumeSnapshotContent** 오브젝트다 모두 유지됩니다.
Retain 값이 설정되고 해당 **VolumeSnapshotContent** 오브젝트를 삭제하지 않고 **VolumeSnapshot** 오브젝트가 삭제되면 해당 콘텐츠는 그대로 유지됩니다. 스냅샷 자체는 스토리지 백엔드에서도 유지됩니다.

2. 다음 명령을 입력하여 볼륨 스냅샷을 삭제합니다.

```
$ oc delete volumesnapshot <volumesnapshot_name>
```

출력 예

```
volumesnapshot.snapshot.storage.k8s.io "mysnapshot" deleted
```

3. 삭제 정책이 **Retain**으로 설정된 경우 다음 명령을 입력하여 볼륨 스냅샷 콘텐츠를 삭제합니다.

```
$ oc delete volumesnapshotcontent <volumesnapshotcontent_name>
```

4. 선택 사항: **VolumeSnapshot** 오브젝트가 성공적으로 삭제되지 않으면 삭제 작업을 계속할 수 있도록 다음 명령을 입력하여 남은 리소스의 종료자를 제거합니다.

중요

VolumeSnapshot 오브젝트에 대한 영구 볼륨 클레임 또는 볼륨 스냅샷 콘텐츠에서 기존 참조가 없음을 확신할 수 있는 경우에만 종료자를 제거하십시오. **--force** 옵션을 사용하면 모든 종료자가 제거될 때까지 삭제 작업에서 스냅샷 오브젝트를 삭제하지 않습니다.

```
$ oc patch -n $PROJECT volumesnapshot/$NAME --type=merge -p '{"metadata":
{"finalizers":null}}'
```

출력 예

```
volumesnapshotclass.snapshot.storage.k8s.io "csi-ocs-rbd-snapclass" deleted
```

종료자가 제거되고 볼륨 스냅샷이 삭제됩니다.

5.3.7. 볼륨 스냅샷 복원

VolumeSnapshot CRD 콘텐츠를 사용하여 기존 볼륨을 이전 상태로 복원할 수 있습니다.

VolumeSnapshot CRD가 바인딩되고 **readyToUse** 값이 **true** 로 설정된 후 해당 리소스를 사용하여 스냅샷에서 데이터로 미리 채워지는 새 볼륨을 프로비저닝할 수 있습니다. .사전 요구 사항 * 실행 중인 OpenShift Container Platform 클러스터에 로그인됩니다. * PVC(영구 볼륨 클레임)) 볼륨 스냅샷을 지원하는 CSI(Container Storage Interface) 드라이버를 사용하여 생성됩니다. * 스토리지 백엔드를 프로비저닝하는 스토리지 클래스입니다. * 볼륨 스냅샷이 생성되었으며 사용할 준비가 되었습니다.

절차

1. 다음과 같이 PVC에서 **VolumeSnapshot** 데이터 소스를 지정합니다.

   pvc-restore.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim-restore
spec:
  storageClassName: csi-hostpath-sc
  dataSource:
    name: mysnap (1)
    kind: VolumeSnapshot (2)
    apiGroup: snapshot.storage.k8s.io (3)
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

   (1) 소스로 사용할 스냅샷을 나타내는 **VolumeSnapshot** 오브젝트의 이름입니다.

   (2) **VolumeSnapshot** 값으로 설정해야 합니다.

   (3) **snapshot.storage.k8s.io** 값으로 설정해야 합니다.

2. 다음 명령을 입력하여 PVC를 생성합니다.

```
$ oc create -f pvc-restore.yaml
```

3. 다음 명령을 입력하여 복원된 PVC가 생성되었는지 확인합니다.

```
$ oc get pvc
```

myclaim-restore와 같은 새 PVC가 표시됩니다.

5.4. CSI 볼륨 복제

볼륨 복제는 OpenShift Container Platform의 데이터 손실을 방지하기 위해 기존 영구 볼륨을 중복합니다. 이 기능은 지원되는 CSI(Container Storage Interface) 드라이버에서만 사용할 수 있습니다. CSI 볼륨 복제를 프로비저닝하기 전에 영구 볼륨에 대해 잘 알고 있어야 합니다.

5.4.1. CSI 볼륨 복제 개요

CSI(Container Storage Interface) 볼륨 복제는 특정 시점에서 기존 영구 볼륨이 복제됩니다.

볼륨 복제는 볼륨 스냅샷과 유사하지만 더 효율적으로 사용할 수 있습니다. 예를 들어, 클러스터 관리자는 기존 클러스터 볼륨의 다른 인스턴스를 생성하여 클러스터 볼륨을 복제할 수 있습니다.

복제는 새 빈 볼륨을 생성하지 않고 백엔드 장치에서 지정된 볼륨을 정확하게 복제합니다. 동적 프로비저닝 후 모든 표준 볼륨을 사용하는 것처럼 볼륨 복제를 사용할 수 있습니다.

복제에는 새 API 오브젝트가 필요하지 않습니다. **PersistentVolumeClaim** 오브젝트의 기존 **dataSource** 필드가 동일한 네임스페이스에서 기존 PersistentVolumeClaim의 이름을 허용하도록 확장됩니다.

5.4.1.1. 지원 제한

기본적으로 OpenShift Container Platform은 이러한 제한이 있는 CSI 볼륨 복제를 지원합니다.

- 대상 PVC(영구 볼륨 클레임)는 소스 PVC와 동일한 네임스페이스에 있어야 합니다.
- 소스 및 대상 스토리지 클래스는 동일해야 합니다.
- CSI 드라이버에서만 지원을 사용할 수 있습니다. In-tree 및 FlexVolumes는 지원되지 않습니다.
- OpenShift Container Platform에는 CSI 드라이버가 포함되어 있지 않습니다. 커뮤니티 또는 스토리지 벤더가 제공하는 CSI 드라이버를 사용하십시오. CSI 드라이버에서 제공하는 설치 지침을 따르십시오.
- CSI 드라이버가 볼륨 복제 기능을 구현하지 못할 수 있습니다. 자세한 내용은 CSI 드라이버 설명서를 참조하십시오.
- OpenShift Container Platform 4.7은 CSI 사양의 버전 1.1.0을 지원합니다.

5.4.2. CSI 볼륨 복제 프로비저닝

복제된 PVC(영구 볼륨 클레임) API 오브젝트를 생성할 때 CSI 볼륨 복제 프로비저닝을 트리거합니다. 복제본은 다른 PVC 콘텐츠로 미리 채워져 다른 영구 볼륨과 동일한 규칙으로 구성됩니다. 한 가지 예외는 동일한 네임스페이스에서 기존 PVC를 참조하는 **dataSource**를 추가해야 한다는 것입니다.

사전 요구 사항

- 실행 중인 OpenShift Container Platform 클러스터에 로그인되어 있습니다.
- 볼륨 복제를 지원하는 CSI 드라이버를 사용하여 PVC가 생성되었습니다.

- 동적 프로비저닝을 위해 스토리지 백엔드가 구성되어 있습니다. 정적 프로비저너에서는 복제 지원을 사용할 수 없습니다.

절차

기존 PVC에서 PVC를 복제하려면 다음을 수행합니다.

1. 다음 YAML로 설명된 **PersistentVolumeClaim** 오브젝트를 사용하여 파일을 생성하고 저장합니다.

   pvc-clone.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-1-clone
  namespace: mynamespace
spec:
  storageClassName: csi-cloning 1
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  dataSource:
    kind: PersistentVolumeClaim
    name: pvc-1
```

   1 스토리지 백엔드를 프로비저닝하는 스토리지 클래스의 이름입니다. 기본 스토리지 클래스를 사용할 수 있으며 **storageClassName**은 사양에서 생략할 수 있습니다.

2. 다음 명령을 실행하여 이전 단계에서 저장한 오브젝트를 생성합니다.

```
$ oc create -f pvc-clone.yaml
```

   새 PVC **pvc-1-clone**이 생성됩니다.

3. 볼륨 복제가 생성되어 다음 명령을 실행하여 준비되었는지 확인합니다.

```
$ oc get pvc pvc-1-clone
```

   pvc-1-clone에 **Bound**로 표시됩니다.

   이제 새로 복제된 PVC를 사용하여 Pod를 구성할 준비가 되었습니다.

4. YAML로 설명된 **Pod** 오브젝트로 파일을 생성하고 저장합니다. 예를 들면 다음과 같습니다.

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: dockerfile/nginx
```

```
    volumeMounts:
    - mountPath: "/var/www/html"
      name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: pvc-1-clone 1
```

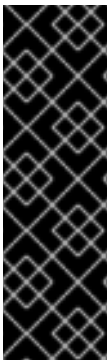
1 CSI 볼륨 복제 작업 중에 생성된 복제 PVC입니다.

생성된 **Pod** 오브젝트가 원래 **dataSource** PVC와 독립적으로 복제된 PVC를 사용, 복제, 삭제하거나 그 스냅샷을 생성할 준비가 되었습니다.

5.5. AWS ELASTIC BLOCK STORE CSI DRIVER OPERATOR

5.5.1. 개요

OpenShift Container Platform은 AWS EBS(Elastic Block Store)의 CSI(Container Storage Interface) 드라이버를 사용하여 PV(영구 볼륨)를 프로비저닝할 수 있습니다.



중요

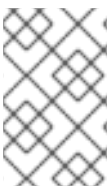
AWS EBS CSI Driver Operator는 기술 프리뷰 기능 전용입니다. Technology Preview 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 https://access.redhat.com/support/offerings/techpreview/를 참조하십시오.

CSI(Container Storage Interface) Operator 및 드라이버를 사용할 때는 영구 스토리지 및 CSI 볼륨 구성에 대해 숙지하는 것이 좋습니다.

AWS EBS 스토리지 자산에 마운트되는 CSI 프로비저닝 PV를 생성하기 위해 OpenShift Container Platform은 **openshift-cluster-csi-drivers** 네임스페이스에 기본적으로 AWS EBS CSI 드라이버와 AWS EBS CSI 드라이버를 설치합니다.

- *AWS EBS CSI Driver Operator*는 기본적으로 PVC를 생성하는 데 사용할 수 있는 StorageClass를 제공합니다. AWS Elastic Block Store를 사용하는 영구 스토리지 에 설명된 대로 AWS EBS StorageClass를 생성하는 옵션이 있습니다.
- *AWS EBS CSI 드라이버*를 사용하면 AWS EBS PV를 생성 및 마운트할 수 있습니다.



참고

OpenShift Container Platform 4.5 클러스터에 AWS EBS CSI Operator 및 드라이버를 설치하는 경우 OpenShift Container Platform 4.7로 업데이트하기 전에 4.5 Operator 및 드라이버를 제거해야 합니다.

5.5.2. CSI 정보

스토리지 벤더는 일반적으로 Kubernetes의 일부로 스토리지 드라이버를 제공합니다. CSI(Container Storage Interface) 구현을 통해 타사 공급자는 코어 Kubernetes 코드를 변경하지 않고 표준 인터페이스를 사용하여 스토리지 플러그인을 제공할 수 있습니다.

CSI Operator는 in-tree 볼륨 플러그인에서 사용할 수 없는 볼륨 스냅샷과 같은 OpenShift Container Platform 사용자 스토리지 옵션을 제공합니다.

중요

OpenShift Container Platform은 기본적으로 in-tree 또는 CSI가 아닌 드라이버를 사용하여 AWS EBS 스토리지를 프로비저닝합니다. 이 in-tree 드라이버는 OpenShift Container Platform의 후속 업데이트에서 제거됩니다. 기존 in-tree 드라이버를 사용하여 프로비저닝된 볼륨은 현재 CSI 드라이버로 마이그레이션할 계획입니다.

OpenShift Container Platform에서 AWS EBS 영구 볼륨을 동적으로 프로비저닝하는 방법에 대한 자세한 내용은 AWS Elastic Block Store를 사용하는 영구 스토리지 를 참조하십시오.

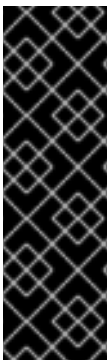
추가 리소스

- AWS Elastic Block Store를 사용하는 영구저장장치
- CSI 볼륨 구성

5.6. GCP PD CSI DRIVER OPERATOR

5.6.1. 개요

OpenShift Container Platform은 GCP(Google Cloud Platform) PD(영구 디스크) 스토리지용 CSI(Container Storage Interface) 드라이버를 사용하여 PV(영구 볼륨)를 프로비저닝할 수 있습니다.



중요

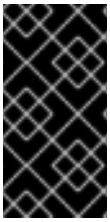
GCP PD CSI Driver Operator는 기술 프리뷰 기능 전용입니다. Technology Preview 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 https://access.redhat.com/support/offerings/techpreview/를 참조하십시오.

CSI(Container Storage Interface) Operator 및 드라이버를 사용할 때는 영구 스토리지 및 CSI 볼륨 구성에 대해 숙지하는 것이 좋습니다.

GCP PD 스토리지 자산에 마운트하는 CSI(영구 볼륨)를 생성하기 위해 OpenShift Container Platform은 **openshift-cluster-csi-drivers** 네임스페이스에서 기본적으로 GCP PD CSI Driver Operator 및 GCP PD CSI 드라이버를 설치합니다.

- **GCP PD CSI Driver Operator**: 기본적으로 Operator는 PVC를 생성하는 데 사용할 수 있는 스토리지 클래스를 제공합니다. GCE 영구 디스크를 사용하는 영구 스토리지 에 설명된 대로 GCP PD 스토리지 클래스를 생성하는 옵션도 있습니다.
- **GCP PD 드라이버**: 드라이버를 사용하면 GCP PD PV를 생성하고 마운트할 수 있습니다.



중요

OpenShift Container Platform은 기본적으로 in-tree 또는 CSI가 아닌 드라이버를 사용하여 GCP PD 스토리지를 프로비저닝합니다. 이 in-tree 드라이버는 OpenShift Container Platform의 후속 업데이트에서 제거됩니다. 기존 in-tree 드라이버를 사용하여 프로비저닝된 볼륨은 현재 CSI 드라이버로 마이그레이션할 계획입니다.

5.6.2. CSI 정보

스토리지 벤더는 일반적으로 Kubernetes의 일부로 스토리지 드라이버를 제공합니다. CSI(Container Storage Interface) 구현을 통해 타사 공급자는 코어 Kubernetes 코드를 변경하지 않고 표준 인터페이스를 사용하여 스토리지 플러그인을 제공할 수 있습니다.

CSI Operator는 in-tree 볼륨 플러그인에서 사용할 수 없는 볼륨 스냅샷과 같은 OpenShift Container Platform 사용자 스토리지 옵션을 제공합니다.

5.6.3. GCP PD CSI 드라이버 스토리지 클래스 매개변수

GCP(Google Cloud Platform) PD(영구 디스크) CSI(Container Storage Interface) 드라이버는 CSI **외부 프로비저너** 사이드카를 컨트롤러로 사용합니다. 이 컨테이너는 CSI 드라이버와 함께 배포된 별도의 Helper 컨테이너입니다. 사이드카는 **CreateVolume** 작업을 트리거하여 PV(영구 볼륨)를 관리합니다.

GCP PD CSI 드라이버는 **csi.storage.k8s.io/fstype** 매개변수 키를 사용하여 동적 프로비저닝을 지원합니다. 다음 표에는 OpenShift Container Platform에서 지원하는 모든 GCP PD CSI 스토리지 클래스 매개변수를 설명합니다.

표 5.2. CreateVolume 매개 변수

매개변수	값	기본	설명
type	**PD-ssd** 또는 **pd-standard**	**pd-standard**	표준 PV 또는 솔리드 스테이트 드라이브 PV 중 하나를 선택할 수 있습니다.
replication-type	**none** 또는 **regional-pd**	**none**	존 또는 지역 PV 중 하나를 선택할 수 있습니다.
disk-encryption-kms-key	새 디스크를 암호화하는 데 사용할 키가 정규화된 리소스 식별자입니다.	빈 문자열	CMEK(고객 관리 암호화 키)를 사용하여 새 디스크를 암호화합니다.

5.6.4. 사용자 정의 암호화 영구 볼륨 생성

PersistentVolumeClaim 오브젝트를 생성할 때 OpenShift Container Platform은 새 PV(영구 볼륨)를 프로비저닝하고 **PersistentVolume** 오브젝트를 생성합니다. 새로 생성된 PV를 암호화하여 클러스터에서 PV를 보호하기 위해 GCP(Google Cloud Platform)에 사용자 지정 암호화 키를 추가할 수 있습니다.

암호화를 위해 새로 연결된 PV는 신규 또는 기존 Google Cloud KMS(키 관리 서비스) 키를 사용하여 클러스터에서 CMEK(고객 관리 암호화 키)를 사용합니다.

사전 요구 사항

- 실행 중인 OpenShift Container Platform 클러스터에 로그인되어 있습니다.

- Cloud KMS 키 링 및 키 버전이 생성되어 있습니다.

CMEK 및 Cloud KMS 리소스에 대한 자세한 내용은 CMEK(고객 관리 암호화 키) 사용을 참조하십시오.

절차

사용자 정의 PV를 생성하려면 다음 단계를 완료합니다.

1. Cloud KMS 키를 사용하여 스토리지 클래스를 생성합니다. 다음 예시에서는 암호화된 볼륨의 동적 프로비저닝을 활성화합니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-gce-pd-cmek
provisioner: pd.csi.storage.gke.io
volumeBindingMode: "WaitForFirstConsumer"
allowVolumeExpansion: true
parameters:
  type: pd-standard
  disk-encryption-kms-key: projects/<key-project-id>/locations/<location>/keyRings/<key-ring>/cryptoKeys/<key> 1
```

1 이 필드는 새 디스크를 암호화하는 데 사용할 키의 리소스 식별자여야 합니다. 값은 대소문자를 구분합니다. 키 ID 값을 제공하는 방법에 대한 자세한 내용은 리소스의 ID 검색 및 Cloud KMS 리소스 ID 가져오기를 참조하십시오.

참고

disk-encryption-✓s-key 매개변수를 기존 스토리지 클래스에 추가할 수 없습니다. 그러나 스토리지 클래스를 삭제하고 동일한 이름과 다른 매개변수 세트로 다시 생성할 수 있습니다. 이렇게 하는 경우 기존 클래스의 프로비저너가 **pd.csi.storage.gke.io**여야 합니다.

2. **oc** 명령을 사용하여 OpenShift Container Platform 클러스터에 스토리지 클래스를 배포합니다.

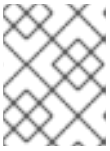
```
$ oc describe storageclass csi-gce-pd-cmek
```

출력 예

```
Name:                  csi-gce-pd-cmek
IsDefaultClass:        No
Annotations:           None
Provisioner:           pd.csi.storage.gke.io
Parameters:            disk-encryption-kms-key=projects/key-project-id/locations/location/keyRings/ring-name/cryptoKeys/key-name,type=pd-standard
AllowVolumeExpansion:  true
MountOptions:          none
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                none
```

3. 이전 단계에서 생성한 스토리지 클래스 오브젝트의 이름과 일치하는 **pvc.yaml** 파일을 생성합니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: podpvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: csi-gce-pd-cmek
  resources:
    requests:
      storage: 6Gi
```



참고

새 스토리지 클래스를 기본값으로 표시한 경우 **storageClassName** 필드를 생략할 수 있습니다.

4. 클러스터에 PVC를 적용합니다.

```
$ oc apply -f pvc.yaml
```

5. PVC 상태를 가져온 후 새로 프로비저닝된 PV에 바인딩되었는지 확인합니다.

```
$ oc get pvc
```

출력 예

```
NAME     STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
podpvc   Bound    pvc-e36abf50-84f3-11e8-8538-42010a800002   10Gi       RWO            csi-
gce-pd-cmek  9s
```

참고

스토리지 클래스에 **WaitForFirstConsumer**로 설정된 **volumeBindingMode** 필드가 있는 경우 이를 확인하기 전에 PVC를 사용하도록 Pod를 생성해야 합니다.

이제 CMEK가 지원하는 PV를 OpenShift Container Platform 클러스터와 함께 사용할 준비가 되었습니다.

추가 리소스

- GCE 영구 디스크를 사용하는 스토리지
- CSI 볼륨 구성

5.7. OPENSTACK CINDER CSI DRIVER OPERATOR

5.7.1. 개요

OpenShift Container Platform은 OpenStack Cinder 용 CSI(Container Storage Interface) 드라이버를 사용하여 PV(영구 볼륨)를 프로비저닝할 수 있습니다.

CSI(Container Storage Interface) Operator 및 드라이버를 사용할 때는 영구 스토리지 및 CSI 볼륨 구성에 대해 숙지하는 것이 좋습니다.

OpenStack Cinder 스토리지 자산에 마운트되는 CSI 프로비저닝 PV를 생성하기 위해 OpenShift Container Platform은 **openshift-cluster-csi-drivers** 네임스페이스에 OpenStack Cinder CSI 드라이버와 OpenStack Cinder CSI 드라이버를 설치합니다.

- *OpenStack Cinder CSI Driver Operator*는 PVC를 생성하는 데 사용할 수 있는 CSI 스토리지 클래스를 제공합니다.
- *OpenStack Cinder CSI 드라이버*를 사용하면 OpenStack Cinder PV를 생성 및 마운트할 수 있습니다.

5.7.2. CSI 정보

스토리지 벤더는 일반적으로 Kubernetes의 일부로 스토리지 드라이버를 제공합니다. CSI(Container Storage Interface) 구현을 통해 타사 공급자는 코어 Kubernetes 코드를 변경하지 않고 표준 인터페이스를 사용하여 스토리지 플러그인을 제공할 수 있습니다.

CSI Operator는 in-tree 볼륨 플러그인에서 사용할 수 없는 볼륨 스냅샷과 같은 OpenShift Container Platform 사용자 스토리지 옵션을 제공합니다.

5.7.3. OpenStack Cinder CSI를 기본 스토리지 클래스로 설정

OpenShift Container Platform에서 기본 스토리지 클래스는 in-tree Cinder 드라이버를 참조합니다. 스토리지 클래스는 OpenShift Container Platform의 후속 업데이트에서 OpenStack Cinder CSI를 참조합니다. 기존 in-tree 스토리지 클래스를 사용하여 프로비저닝된 볼륨은 해당 시점에서 OpenStack Cinder CSI 스토리지 클래스로 마이그레이션할 계획입니다.

OpenStack Cinder CSI 드라이버는 **cinder.csi.openstack.org** 매개변수 키를 사용한 동적 프로비저닝을 지원합니다.

OpenShift Container Platform에서 OpenStack Cinder CSI 프로비저닝을 활성화하려면 기본 in-tree 스토리지 클래스를 **standard-csi**로 덮어쓰는 것이 좋습니다. 다른 방법으로 PVC(영구 볼륨 클레임)를 생성하고 스토리지 클래스를 "standard-csi"로 지정할 수 있습니다.

절차

기본 in-tree 스토리지 클래스를 작성하여 **standard-csi** 스토리지 클래스를 적용하려면 다음 단계를 사용합니다.

1. 스토리지 클래스를 나열합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                 PROVISIONER                RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
standard(default)    cinder.csi.openstack.org   Delete          WaitForFirstConsumer   true
46h
standard-csi         kubernetes.io/cinder       Delete          WaitForFirstConsumer   true
46h
```

2. 기본 StorageClass에 대해 주석 **storageclass.kubernetes.io/is-default-class**의 값을 **false**로 변경합니다.

```
$ oc patch storageclass standard -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. 주석을 **storageclass.kubernetes.io/is-default-class=true**로 추가하거나 수정하여 다른 스토리지 클래스를 기본값으로 설정합니다.

```
$ oc patch storageclass standard-csi -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

4. PVC가 기본적으로 CSI 스토리지 클래스를 참조하는지 확인합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                 PROVISIONER                RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
standard             kubernetes.io/cinder       Delete          WaitForFirstConsumer   true
46h
standard-csi(default)  cinder.csi.openstack.org   Delete          WaitForFirstConsumer   true
46h
```

5. 선택 사항: 스토리지 클래스를 지정하지 않고도 새 PVC를 정의할 수 있습니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cinder-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

특정 스토리지 클래스를 지정하지 않는 PVC는 기본 스토리지 클래스를 사용하여 자동으로 프로비저닝됩니다.

6. 선택 사항: 새 파일을 구성한 후 클러스터에 생성합니다.

```
$ oc create -f cinder-claim.yaml
```

추가 리소스

- CSI 볼륨 구성

5.8. OPENSTACK MANILA CSI DRIVER OPERATOR

5.8.1. 개요

OpenShift Container Platform은 OpenStack Manila 공유 파일 시스템 서비스의 CSI(Container Storage Interface) 드라이버를 사용하여 PV(영구 볼륨)를 프로비저닝할 수 있습니다.

CSI(Container Storage Interface) Operator 및 드라이버를 사용할 때는 영구 스토리지 및 CSI 볼륨 구성에 대해 숙지하는 것이 좋습니다.

Manila 스토리지 자산에 마운트되는 CSI 프로비저닝 PV를 생성하기 위해 OpenShift Container Platform은 Manila 서비스가 활성화된 모든 OpenStack 클러스터에 Manila CSI Driver Operator 및 Manila CSI 드라이버를 설치합니다.

- *Manila CSI Driver Operator*는 사용 가능한 모든 Manila 공유 유형에 대해 PVC를 생성하는 데 필요한 스토리지 클래스를 생성합니다. Operator는 **openshift-cluster-csi-drivers** 네임스페이스에 설치됩니다.
- *Manila CSI 드라이버*를 사용하면 Manila PV를 생성 및 마운트할 수 있습니다. 드라이버는 **openshift-manila-csi-driver** 네임스페이스에 설치됩니다.

5.8.2. CSI 정보

스토리지 벤더는 일반적으로 Kubernetes의 일부로 스토리지 드라이버를 제공합니다. CSI(Container Storage Interface) 구현을 통해 타사 공급자는 코어 Kubernetes 코드를 변경하지 않고 표준 인터페이스를 사용하여 스토리지 플러그인을 제공할 수 있습니다.

CSI Operator는 in-tree 볼륨 플러그인에서 사용할 수 없는 볼륨 스냅샷과 같은 OpenShift Container Platform 사용자 스토리지 옵션을 제공합니다.

5.8.3. Manila CSI Driver Operator 제한

Manila CSI(Container Storage Interface) Driver Operator에 다음과 같은 제한 사항이 적용됩니다.

NFS만 지원

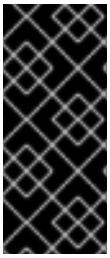
OpenStack Manila는 NFS, CIFS 및 CEPHFS와 같은 많은 네트워크 연결 스토리지 프로토콜을 지원하며 OpenStack 클라우드에서 선택적으로 사용할 수 있습니다. OpenShift Container Platform의 Manila CSI Driver Operator는 NFS 프로토콜만 지원합니다. 기본 OpenStack 클라우드에서 NFS를 사용할 수 없고 활성화되지 않은 경우 Manila CSI Driver Operator를 사용하여 OpenShift Container Platform의 스토리지를 프로비저닝할 수 없습니다.

백엔드가 CephFS-NFS인 경우 스냅샷은 지원되지 않습니다.

PV(영구 볼륨)의 스냅샷을 가져와 볼륨을 스냅샷으로 되돌리려면 Manila에서 이러한 기능을 지원하는지 확인해야 합니다. Red Hat OpenStack 관리자는 스냅샷(**extra-spec snapshot_support**)에 대한 지원을 활성화하고 스냅샷에서**공유(추가-spec create_share_from_snapshot_support**)를 생성하려는 스토리지 클래스와 연결된 공유 유형에서 공유를 생성해야 합니다.

FSGroups는 지원되지 않습니다.

Manila CSI는 여러 독자와 여러 사용자가 액세스할 수 있는 공유 파일 시스템을 제공하므로 FSGroups 사용을 지원하지 않습니다. 이는 ReadWriteOnce 액세스 모드로 생성된 영구 볼륨의 경우에도 마찬가지입니다. 따라서 Manila CSI Driver에서 수동으로 사용하기 위해 수동으로 생성하는 스토리지 클래스에 **fsType** 특성을 지정하지 않는 것이 중요합니다.



중요

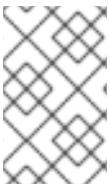
Red Hat OpenStack Platform 16.x 및 17.x에서 NFS를 통한 CephFS의 공유 파일 시스템 서비스(Manila)는 Manila CSI를 통해 OpenShift Container Platform에 공유를 완전히 지원합니다. 그러나 이 솔루션은 대규모 확장을 위한 것이 아닙니다. Red Hat OpenStack Platform용 CephFS NFS Manila-CSI 워크로드 권장 사항에서 중요한 권장 사항을 검토하십시오.

5.8.4. 동적으로 Manila CSI 볼륨 프로비저닝

OpenShift Container Platform은 사용 가능한 Manila 공유 유형마다 스토리지 클래스를 설치합니다.

생성된 YAML 파일은 Manila 및 해당 CSI(Container Storage Interface) 플러그인에서 완전히 분리됩니다. 애플리케이션 개발자는 RWX(ReadWriteMany) 스토리지를 동적으로 프로비저닝하고 YAML 매니페스트를 사용하여 스토리지를 안전하게 사용하는 애플리케이션에 Pod를 배포할 수 있습니다.

PVC 정의의 스토리지 클래스 참조는 제외되어 AWS, GCP, Azure 및 기타 플랫폼에서 OpenShift Container Platform과 함께 사용하는 동일한 Pod 및 PVC(영구 볼륨 클레임) 정의를 사용할 수 있습니다.



참고

Manila 서비스는 선택 사항입니다. RHOSP(Red Hat OpenStack Platform)에서 서비스를 활성화하지 않으면 Manila CSI 드라이버가 설치되지 않고 Manila용 스토리지 클래스가 생성되지 않습니다.

사전 요구 사항

- RHOSP는 적절한 Manila와 함께 배포되어 OpenShift Container Platform에서 볼륨을 동적으로 프로비저닝 및 마운트하는 데 사용할 수 있습니다.

절차(UI)

웹 콘솔을 사용하여 Manila CSI 볼륨을 동적으로 생성하려면 다음을 수행합니다.

1. OpenShift Container Platform 콘솔에서 **스토리지 → 영구 볼륨 클레임**을 클릭합니다.
2. 영구 볼륨 클레임 생성 개요에서 **영구 볼륨 클레임 생성**을 클릭합니다.
3. 결과 페이지에 필요한 옵션을 정의합니다.
   a. 적절한 스토리지 클래스를 선택합니다.
   b. 스토리지 클레임의 고유한 이름을 입력합니다.
   c. 생성 중인 PVC에 대한 읽기 및 쓰기 권한을 지정하려면 액세스 모드를 선택합니다.

중요

이 PVC를 클러스터의 여러 노드의 여러 Pod에 마운트하도록 하는 PV(영구 볼륨)를 사용하려면 RWX를 사용합니다.

4. 스토리지 클레임의 크기를 정의합니다.
5. **만들기**를 클릭하여 영구 볼륨 클레임을 생성하고 영구 볼륨을 생성합니다.

절차(CLI)

CLI(명령줄 인터페이스)를 사용하여 Manila CSI 볼륨을 동적으로 생성하려면 다음을 수행합니다.

1. 다음 YAML로 설명된 **PersistentVolumeClaim** 오브젝트를 사용하여 파일을 생성하고 저장합니다.

   pvc-manila.yaml

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: pvc-manila
   spec:
     accessModes: 1
       - ReadWriteMany
     resources:
       requests:
         storage: 10Gi
     storageClassName: csi-manila-gold 2
   ```

   1 이 PVC를 클러스터의 여러 노드의 여러 Pod에 마운트하도록 하는 PV(영구 볼륨)를 사용하려면 RWX를 사용합니다.

   2 스토리지 백엔드를 프로비저닝하는 스토리지 클래스의 이름입니다. Manila 스토리지 클래스는 Operator에 의해 프로비저닝되며 **csi-manila-** 접두사가 적용됩니다.

2. 다음 명령을 실행하여 이전 단계에서 저장한 오브젝트를 생성합니다.

   ```
   $ oc create -f pvc-manila.yaml
   ```

   새 PVC가 생성됩니다.

3. 볼륨이 생성되고 준비되었는지 확인하려면 다음 명령을 실행합니다.

   ```
   $ oc get pvc pvc-manila
   ```

   pvc-manila에 **Bound**가 표시됩니다.

이제 새 PVC를 사용하여 Pod를 구성할 수 있습니다.

추가 리소스

- CSI 볼륨 구성

5.9. RED HAT VIRTUALIZATION CSI DRIVER OPERATOR

5.9.1. 개요

OpenShift Container Platform은 RHV(Red Hat Virtualization)용 CSI(Container Storage Interface) 드라이버를 사용하여 PV(영구 볼륨)를 프로비저닝할 수 있습니다.

CSI(Container Storage Interface) Operator 및 드라이버를 사용할 때는 영구 스토리지 및 CSI 볼륨 구성에 대해 숙지하는 것이 좋습니다.

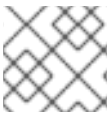
RHV 스토리지 자산에 마운트된 CSI 프로비저닝 PV를 생성하기 위해 OpenShift Container Platform은 **openshift-cluster-csi-drivers** 네임스페이스에 기본적으로 oVirt CSI Driver Operator 및 oVirt CSI 드라이버를 설치합니다.

- *oVirt CSI Driver Operator*는 PVC(영구 볼륨 클레임)를 생성하는 데 사용할 수 있는 기본 **StorageClass** 오브젝트를 제공합니다.
- *oVirt CSI 드라이버*를 사용하면 oVirt PV를 생성 및 마운트할 수 있습니다.

5.9.2. CSI 정보

스토리지 벤더는 일반적으로 Kubernetes의 일부로 스토리지 드라이버를 제공합니다. CSI(Container Storage Interface) 구현을 통해 타사 공급자는 코어 Kubernetes 코드를 변경하지 않고 표준 인터페이스를 사용하여 스토리지 플러그인을 제공할 수 있습니다.

CSI Operator는 in-tree 볼륨 플러그인에서 사용할 수 없는 볼륨 스냅샷과 같은 OpenShift Container Platform 사용자 스토리지 옵션을 제공합니다.



참고

oVirt CSI 드라이버는 스냅샷을 지원하지 않습니다.

5.9.3. oVirt CSI 드라이버 스토리지 클래스

OpenShift Container Platform은 동적으로 프로비저닝된 영구 볼륨을 생성하는 데 사용되는 **ovirt-csi-sc**라는 이름의 **StorageClass** 유형의 기본 오브젝트를 생성합니다.

다양한 구성을 위한 추가 스토리지 클래스를 생성하려면 다음 샘플 YAML에서 설명되는 **StorageClass** 오브젝트로 파일을 생성하고 저장합니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> 1
  annotations:
    storageclass.kubernetes.io/is-default-class: "false" 2
provisioner: csi.ovirt.org
parameters:
  storageDomainName: <rhv-storage-domain-name> 3
  thinProvisioning: "true" 4
  csi.storage.k8s.io/fstype: ext4 5
```

1. StorageClass의 이름입니다.
2. 스토리지 클래스가 클러스터의 기본 스토리지 클래스인 경우 **false**로 설정합니다. **true**로 설정하면 기존 기본 스토리지 클래스를 편집한 후 **false**로 설정해야 합니다.
3. 사용할 RHV 스토리지 도메인 이름입니다.
4. 디스크는 씬으로 프로비저닝해야 합니다.
5. 생성할 파일 시스템 유형입니다.

5.9.4. RHV에서 영구 볼륨 생성

PVC(**PersistentVolumeClaim**) 오브젝트를 생성할 때 OpenShift Container Platform에서 새 PV(영구 볼륨)를 프로비저닝하고 **PersistentVolume** 오브젝트를 생성합니다.

사전 요구 사항

- 실행 중인 OpenShift Container Platform 클러스터에 로그인되어 있습니다.
- **ovirt-credentials** 보안에 올바른 RHV 인증 정보가 제공되었습니다.
- oVirt CSI 드라이버가 설치되어 있습니다.
- 하나 이상의 스토리지 클래스가 정의되어 있습니다.

절차

- RHV에 영구 볼륨을 동적으로 생성하기 위해 we 콘솔을 사용하는 경우:
  1. OpenShift Container Platform 콘솔에서 **스토리지 → 영구 볼륨 클레임**을 클릭합니다.
  2. 영구 볼륨 클레임 생성 개요에서 **영구 볼륨 클레임 생성**을 클릭합니다.
  3. 결과 페이지에 필요한 옵션을 정의합니다.
  4. 적절한 **StorageClass** 오브젝트를 선택합니다. 이는 기본적으로 **ovirt-csi-sc**입니다.
  5. 스토리지 클레임의 고유한 이름을 입력합니다.
  6. 액세스 모드를 선택합니다. 현재는 RWO(ReadWriteOnce)가 지원되는 유일한 액세스 모드입니다.
  7. 스토리지 클레임의 크기를 정의합니다.
  8. 볼륨 모드 선택:
     파일 시스템: 포드에 디렉터리로 마운트됩니다. 이 모드가 기본값입니다.

     블록: 파일 시스템이 없는 블록 장치
  9. **만들기**를 클릭하여 **PersistentVolumeClaim** 오브젝트를 생성하고 **PersistentVolume** 오브젝트를 생성합니다.
- CLI(명령줄 인터페이스)를 사용하여 RHV CSI 볼륨을 동적으로 생성하는 경우 다음을 실행합니다.
  1. 다음 샘플 YAML로 설명된 **PersistentVolumeClaim** 오브젝트를 사용하여 파일을 생성하고 저장합니다.

     pvc-ovirt.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-ovirt
spec:
  storageClassName: ovirt-csi-sc (1)
  accessModes:
```

```
  - ReadWriteOnce
resources:
  requests:
  storage: <volume size>  2
volumeMode: <volume mode>  3
```

1. 필요한 스토리지 클래스의 이름입니다.
2. GiB 단위의 볼륨 크기입니다.
3. 지원되는 옵션:
   - **파일 시스템**: 포드에 디렉터리로 마운트됩니다. 이 모드가 기본값입니다.
   - **블록**: 파일 시스템이 없는 블록 장치.

2. 다음 명령을 실행하여 이전 단계에서 저장한 오브젝트를 생성합니다.

```
$ oc create -f pvc-ovirt.yaml
```

3. 볼륨이 생성되고 준비되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get pvc pvc-ovirt
```

pvc-ovirt는 Bound임을 보여줍니다.

추가 리소스

- CSI 볼륨 구성
- 동적 프로비저닝

6장. 영구 볼륨 확장

6.1. 볼륨 확장 지원 활성화

영구 볼륨을 확장하려면 **StorageClass** 오브젝트에서 **allowVolumeExpansion** 필드가 **true**로 설정되어 있어야 합니다.

절차

- **StorageClass** 오브젝트를 편집하고 **allowVolumeExpansion** 속성을 추가합니다. 다음 예시는 스토리지 클래스 구성 하단에 이 행을 추가하는 방법을 보여줍니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
...
parameters:
  type: gp2
reclaimPolicy: Delete
allowVolumeExpansion: true (1)
```

(1) 이 속성을 **true**로 설정하면 생성 후 PVC가 확장됩니다.

6.2. CSI 볼륨 확장

CSI(Container Storage Interface)를 사용하여 이미 생성된 스토리지 볼륨을 확장할 수 있습니다.

OpenShift Container Platform은 기본적으로 CSI 볼륨 확장을 지원합니다. 그러나 특정 CSI 드라이버가 필요합니다.

OpenShift Container Platform 4.7은 CSI 사양의 버전 1.1.0을 지원합니다.

중요

CSI 볼륨 확장은 기술 프리뷰 기능 전용입니다. Technology Preview 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 https://access.redhat.com/support/offerings/techpreview/를 참조하십시오.

6.3. 지원되는 드라이버를 사용한 FLEXVOLUME 확장

FlexVolume을 사용하여 백엔드 스토리지 시스템에 연결할 때 이미 생성된 후 영구 스토리지 볼륨을 확장할 수 있습니다. OpenShift Container Platform에서 PVC(영구 볼륨 클레임)를 수동으로 업데이트하여 수행합니다.

FlexVolume은 **RequiresFSResize**가 **true**로 설정된 경우 확장을 허용합니다. Pod를 다시 시작할 때 FlexVolume을 확장할 수 있습니다.

다른 볼륨 유형과 유사하게 Pod에서 사용할 때 FlexVolume 볼륨도 확장할 수 있습니다.

사전 요구 사항

- 기본 볼륨 드라이버는 크기 조정을 지원합니다.
- 드라이버에서는 **RequiresFSResize** 기능이 **true**로 설정됩니다.
- 동적 프로비저닝이 사용됩니다.
- 제어 **StorageClass** 오브젝트에 **allowVolumeExpansion**이 **true**로 설정되어 있습니다.

절차

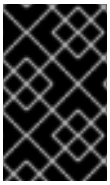
- FlexVolume 플러그인에서 크기 조정을 사용하려면 다음 방법을 사용하여 **ExpandableVolumePlugin** 인터페이스를 구현해야 합니다.

  RequiresFSResize

  true인 경우 용량을 직접 업데이트합니다. **false**인 경우 **ExpandFS** 메서드를 호출하여 파일 시스템의 크기 조정을 완료합니다.

  ExpandFS

  true인 경우 **ExpandFS**를 호출하여 물리 볼륨 확장을 수행한 후 파일 시스템의 크기를 조정합니다. 볼륨 드라이버는 파일 시스템의 크기 조정과 함께 물리 볼륨 크기 조정을 수행할 수도 있습니다.



중요

OpenShift Container Platform은 컨트롤 플레인 노드 (마스터 노드라고도 함)에 FlexVolume 플러그인 설치를 지원하지 않으므로 FlexVolume의 컨트롤 플레인 확장을 지원하지 않습니다.

6.4. 파일 시스템을 사용한 PVC(영구 볼륨 클레임) 처리

GCE PD, EBS 및 Cinder와 같은 파일 시스템 조정이 필요한 볼륨 유형에 따라 PVC를 2단계 프로세스로 합니다. 이 프로세스에는 클라우드 공급자에서 볼륨 오브젝트를 확장한 후 실제 노드에서 파일 시스템을 확장하는 작업이 포함됩니다.

노드의 파일 시스템을 배양하는 것은 볼륨으로 새 Pod가 시작될 때만 수행됩니다.

사전 요구 사항

- 제어 **StorageClass** 오브젝트에서 **allowVolumeExpansion**이 **true**로 설정되어야 합니다.

절차

1. PVC를 편집하고 **spec.resources.requests**를 편집하여 새 크기를 요청합니다. 예를 들어, 다음에서는 **ebs** PVC를 8Gi로 확장합니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ebs
spec:
  storageClass: "storageClassWithFlagSet"
  accessModes:
    - ReadWriteOnce
```

```
resources:
  requests:
    storage: 8Gi (1)
```

(1) **spec.resources.requests**를 더 큰 용량으로 업데이트하면 PVC가 확장됩니다.

2. 클라우드 공급자 오브젝트 크기 조정이 완료되면 PVC가 **FileSystemResizePending**로 설정됩니다. 다음 명령을 입력하여 조건을 확인합니다.

```
$ oc describe pvc <pvc_name>
```

3. 클라우드 공급자 오브젝트 크기 조정이 완료되면 **PersistentVolume** 오브젝트가 **PersistentVolume.Spec.Capacity**의 새로 요청된 크기를 반영합니다. 이 시점에서 PVC에서 새 Pod를 생성하거나 재생성하여 파일 시스템 크기 조정을 완료할 수 있습니다. Pod가 실행되면 새로 요청된 크기를 사용할 수 있으며, **FileSystemResizePending** 조건이 PVC에서 제거됩니다.

6.5. 볼륨을 분리할 때 실패에서 복구

기본 스토리지에 실패하면 OpenShift Container Platform 관리자는 PVC(영구 볼륨 클레임) 상태를 수동으로 복구하고 크기 조정 요청을 취소할 수 있습니다. 그렇지 않으면 관리자가 개입하지 않고 컨트롤러에 의해 크기 조정 요청이 지속적으로 다시 설정됩니다.

절차

1. PVC에 바인딩된 PV(영구 볼륨)를 **Retain** 회수 정책으로 표시합니다. PV를 편집하고 **persistentVolumeReclaimPolicy**를 **Retain**으로 변경하여 수행할 수 있습니다.

2. PVC를 삭제합니다. 이는 나중에 다시 생성됩니다.

3. 새로 생성된 PVC가 **Retain**으로 표시된 PV에 바인딩할 수 있도록 PV를 수동으로 편집하고 PV 사양에서 **claimRef** 항목을 삭제합니다. PV가 **Available**로 표시됩니다.

4. PVC를 작은 크기로 다시 생성하거나 기본 스토리지 공급자가 할당할 수 있는 크기입니다.

5. PVC의 **volumeName** 필드를 PV 이름으로 설정합니다. 이렇게 하면 PVC가 프로비저닝된 PV에만 바인딩됩니다.

6. PV에서 회수 정책을 복구합니다.

7장. 동적 프로비저닝

7.1. 동적 프로비저닝 소개

StorageClass 리소스 객체는 요청 가능한 스토리지를 설명하고 분류할 뿐 만 아니라 필요에 따라 동적으로 프로비저닝된 스토리지에 대한 매개 변수를 전달하는 수단을 제공합니다. **StorageClass** 객체는 다른 수준의 스토리지 및 스토리지에 대한 액세스를 제어하기 위한 관리 메커니즘으로 사용될 수 있습니다. 클러스터 관리자 (**cluster-admin**) 또는 스토리지 관리자 (**storage-admin**)는 사용자가 기본 스토리지 볼륨 소스에 대한 자세한 지식이 없어도 요청할 수 있는 **StorageClass** 오브젝트를 정의하고 생성할 수 있습니다.

OpenShift Container Platform 영구 볼륨 프레임 워크를 사용하면이 기능을 사용할 수 있으며 관리자는 영구 스토리지로 클러스터를 프로비저닝할 수 있습니다. 또한 이 프레임 워크를 통해 사용자는 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다.

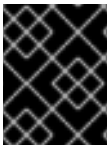
OpenShift Container Platform에서 많은 스토리지 유형을 영구 볼륨으로 사용할 수 있습니다. 관리자가 이를 모두 정적으로 프로비저닝할 수 있지만 일부 스토리지 유형은 기본 제공자 및 플러그인 API를 사용하여 동적으로 만들 수 있습니다.

7.2. 사용 가능한 동적 프로비저닝 플러그인

OpenShift Container Platform은 다음과 같은 프로비저너 플러그인을 제공합니다. 이에는 클러스터의 구성된 제공자 API를 사용하여 새 스토리지 리소스의 동적 프로비저닝을 위한 일반 구현이 포함되어 있습니다.

스토리지 유형	프로비저너 플러그인 이름	참고
Red Hat OpenStack Platform (RHOSP) Cinder	**kubernetes.io/cinder**	
RHOSP Manila Container Storage Interface (CSI)	**manila.csi.openstack.org**	설치 완료되면 OpenStack Manila CSI Driver Operator 및 ManilaDriver가 동적 프로비저닝에 필요한 모든 사용 가능한 Manila 공유 유형에 필요한 스토리지 클래스를 자동으로 생성합니다.
AWS Elastic Block Store (EBS)	**kubernetes.io/aws-ebs**	다른 영역에서 여러 클러스터를 사용할 때 동적 프로비저닝의 경우 각 노드에 **Key=kubernetes.io/cluster/<cluster_name>,Value=<cluster_id>**로 태그를 지정합니다. 여기서 **<cluster_name>** 및 **<cluster_id>**는 클러스터마다 고유합니다.
Azure Disk	**kubernetes.io/azure-disk**	

스토리지 유형	프로비저너 플러그인 이름	참고
Azure File	**kubernetes.io/azure-file**	**persistent-volume-binder** 서비스 계정에는 Azure 스토리지 계정 및 키를 저장할 시크릿을 생성하고 검색할 수 있는 권한이 필요합니다.
GCE Persistent Disk (gcePD)	**kubernetes.io/gce-pd**	멀티 존 설정에서는 현재 클러스터에 노드가 없는 영역에서 PV가 생성되지 않도록 GCE 프로젝트 당 하나의 OpenShift Container Platform 클러스터를 실행하는 것이 좋습니다.
VMware vSphere	**kubernetes.io/vsphere-volume**	



중요

선택한 프로비저너 플러그인에는 관련 문서에 따라 클라우드, 호스트 또는 타사 공급자를 구성해야 합니다.

7.3. 스토리지 클래스 정의

StorageClass 객체는 현재 전역 범위 객체이며 **cluster-admin** 또는 **storage-admin** 사용자가 만들어야 합니다.

중요

클러스터 스토리지 작업자는 사용 중인 플랫폼에 따라 기본 스토리지 클래스를 설치할 수 있습니다. 이 스토리지 클래스는 Operator가 소유하고 제어합니다. 주석 및 레이블 정의 외에는 삭제하거나 변경할 수 없습니다. 다른 동작이 필요한 경우 사용자 정의 스토리지 클래스를 정의해야 합니다.

다음 섹션에서는 **StorageClass** 개체의 기본 정의 및 지원되는 각 플러그인 유형에 대한 구체적인 예를 설명합니다.

7.3.1. 기본 StorageClass 개체 정의

다음 리소스는 스토리지 클래스를 구성하는 데 사용되는 매개변수 및 기본값을 보여줍니다. 이 예에서는 AWS ElasticBlockStore (EBS) 객체 정의를 사용합니다.

StorageClass 정의 예

```
kind: StorageClass (1)
apiVersion: storage.k8s.io/v1 (2)
metadata:
  name: gp2 (3)
  annotations: (4)
```

```
    storageclass.kubernetes.io/is-default-class: 'true'
    ...
provisioner: kubernetes.io/aws-ebs 5
parameters: 6
  type: gp2
...
```

1. (필수) API 객체 유형입니다.
2. (필수) 현재 apiVersion입니다.
3. (필수) 스토리지 클래스의 이름입니다.
4. (선택 사항) 스토리지 클래스의 주석입니다.
5. (필수) 이 스토리지 클래스에 연결된 프로비저너의 유형입니다.
6. (선택 사항) 특정 프로비저너에 필요한 매개 변수로, 플러그인에 따라 변경됩니다.

7.3.2. 스토리지 클래스 주석

스토리지 클래스를 클러스터 전체 기본값으로 설정하려면 스토리지 클래스의 메타데이터에 다음 주석을 추가합니다.

```
storageclass.kubernetes.io/is-default-class: "true"
```

예를 들면 다음과 같습니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
...
```

이렇게 하면 특정 스토리지 클래스를 지정하지 않은 모든 PVC(영구 볼륨 클레임)가 기본 스토리지 클래스를 통해 자동으로 프로비저닝됩니다. 그러나 클러스터에는 두 개 이상의 스토리지 클래스가 있을 수 있지만, 그 중 하나만 기본 스토리지 클래스일 수 있습니다.

참고

베타 주석 **storageclass.beta.kubernetes.io/is-default-class**는 여전히 작동하지만 향후 릴리스에서는 제거될 예정입니다.

스토리지 클래스 설명을 설정하려면 스토리지 클래스의 메타데이터에 다음 주석을 추가합니다.

```
kubernetes.io/description: My Storage Class Description
```

예를 들면 다음과 같습니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
```

```
metadata:
  annotations:
    kubernetes.io/description: My Storage Class Description
...
```

7.3.3. RHOSP Cinder 오브젝트 정의

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gold
provisioner: kubernetes.io/cinder
parameters:
  type: fast (1)
  availability: nova (2)
  fsType: ext4 (3)
```

(1) Cinder에서 생성 된 볼륨 유형입니다. 기본값은 비어 있습니다.

(2) 가용성 영역입니다. 지정하지 않으면 일반적으로 OpenShift Container Platform 클러스터에 노드가 있는 모든 활성 영역에서 볼륨이 라운드 로빈됩니다.

(3) 동적으로 프로비저닝된 볼륨에서 생성된 파일 시스템입니다. 이 값은 동적으로 프로비저닝된 영구 볼륨의 **fsType** 필드에 복사되며 볼륨이 처음 마운트될 때 파일 시스템이 작성됩니다. 기본값은 **ext4**입니다.

7.3.4. RHOSP Manila CSI(Container Storage Interface) 오브젝트 정의

설치 완료되면 OpenStack Manila CSI Driver Operator 및 ManilaDriver가 동적 프로비저닝에 필요한 모든 사용 가능한 Manila 공유 유형에 필요한 스토리지 클래스를 자동으로 생성합니다.

7.3.5. AWS Elastic Block Store (EBS) 객체 정의

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 (1)
  iopsPerGB: "10" (2)
  encrypted: "true" (3)
  kmsKeyId: keyvalue (4)
  fsType: ext4 (5)
```

(1) (필수 사항)**io1**, **gp2**, **sc1**, **st1** 중에서 선택합니다. 기본값은 **gp2**입니다. 유효한 Amazon Resource Name (ARN) 값은 AWS 설명서를 참조하십시오.

2 (선택 사항) **io1** 볼륨만 해당됩니다. GiB마다 초당 I/O 작업 수입니다. AWS 볼륨 플러그인은 이 값과 요청된 볼륨 크기를 곱하여 볼륨의 IOPS를 계산합니다. 값의 상한은 AWS가 지원하는 최대치인

3 (선택 사항) EBS 볼륨을 암호화할지 여부를 나타냅니다. 유효한 값은 **true** 또는 **false**입니다.

4 (선택 사항) 볼륨을 암호화할 때 사용할 키의 전체 ARN입니다. 값을 지정하지 않는 경우에도 **encypted** 가 **true**로 설정되어 있는 경우 AWS가 키를 생성합니다. 유효한 ARN 값은 AWS 설명서를 참조하십시오.

5 (선택 사항) 동적으로 프로비저닝 된 볼륨에서 작성되는 파일 시스템입니다. 이 값은 동적으로 프로비저닝된 영구 볼륨의 **fsType** 필드에 복사되며 볼륨이 처음 마운트될 때 파일 시스템이 작성됩니다. 기본값은 **ext4**입니다.

7.3.6. Azure Disk 오브젝트 정의

azure-advanced-disk-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-premium
provisioner: kubernetes.io/azure-disk
volumeBindingMode: WaitForFirstConsumer 1
allowVolumeExpansion: true
parameters:
  kind: Managed 2
  storageaccounttype: Premium_LRS 3
reclaimPolicy: Delete
```

1 **WaitForFirstConsumer**를 사용하는 것이 좋습니다. 이렇게 하면 볼륨을 프로비저닝하고 사용 가능한 영역에서 사용 가능한 작업자 노드의 Pod를 예약할 수 있는 충분한 스토리지를 허용합니다.

2 가능한 값은 **Shared**(기본값), **Managed**, **Dedicated**입니다.

중요

Red Hat은 **kind 사용을 지원합니다. 스토리지 클래스에서 managed** 입니다.

Shared 및 **Dedicated**를 사용하여 Azure는 관리되지 않은 디스크를 생성합니다. 반면 OpenShift Container Platform은 머신 OS(root) 디스크의 관리 디스크를 생성합니다. Azure Disk는 노드에서 관리 및 관리되지 않은 디스크를 모두 사용하도록 허용하지 않으므로 **Shared** 또는 **Dedicated**로 생성된 관리되지 않은 디스크를 OpenShift Container Platform 노드에 연결할 수 없습니다.

3 Azure 스토리지 계정 SKU층입니다. 기본값은 비어 있습니다. 프리미엄 VM은 **Standard_LRS** 및 **Premium_LRS** 디스크를 모두 연결할 수 있으며 표준 VM은 **Standard_LRS** 디스크만 연결할 수 있습니다. 관리형 VM은 관리 디스크만 연결할 수 있으며 관리되지 않는 VM은 관리되지 않는 디스크만 연결할 수 있습니다.

a. **kind**가 **Shared**로 설정된 경우 Azure는 클러스터와 동일한 리소스 그룹에 있는 일부 공유 스토리지 계정에서 관리되지 않는 디스크를 만듭니다.

b. **kind**가 **Managed**로 설정된 경우 Azure는 새 관리 디스크를 만듭니다.

c. **kind**가 **Dedicated**로 설정되고 **storageAccount**가 지정된 경우 Azure는 클러스터와 동일한 리소스 그룹에서 새로운 관리되지 않는 디스크에 대해 지정된 스토리지 계정을 사용합니다. 이 기능이 작동하려면 다음 사항이 전제가 되어야 합니다.

- 지정된 스토리지 계정이 같은 지역에 있어야 합니다.
- Azure Cloud Provider는 스토리지 계정에 대한 쓰기 권한이 있어야 합니다.

d. **kind**가 **Dedicated**로 설정되어 있고 **storageAccount**가 지정되지 않은 경우 Azure는 클러스터와 동일한 리소스 그룹에 새로운 관리되지 않는 디스크에 대한 새로운 전용 스토리지 계정을 만듭니다.

7.3.7. Azure File 객체 정의

Azure File 스토리지 클래스는 시크릿을 사용하여 Azure 파일 공유를 만드는 데 필요한 Azure 스토리지 계정 이름과 스토리지 계정 키를 저장합니다. 이러한 권한은 다음 절차의 일부로 생성됩니다.

절차

1. 시크릿을 작성하고 볼 수 있는 액세스를 허용하는 **ClusterRole** 오브젝트를 정의합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
#  name: system:azure-cloud-provider
  name: <persistent-volume-binder-role> 1
rules:
- apiGroups: ['']
  resources: ['secrets']
  verbs:     ['get','create']
```

1 시크릿을 표시하고 작성하는 클러스터 역할의 이름입니다.

2. 서비스 계정에 클러스터 역할을 추가합니다.

```
$ oc adm policy add-cluster-role-to-user <persistent-volume-binder-role>
```

출력 예

```
 system:serviceaccount:kube-system:persistent-volume-binder
```

3. Azure File **StorageClass** 오브젝트를 만듭니다.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <azure-file> 1
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus 2
  skuName: Standard_LRS 3
```

```
  storageAccount: <storage-account> 4
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 StorageClass의 이름입니다. 영구 볼륨 클레임은 이 스토리지 클래스를 사용하여 관련 영구 볼륨을 프로비저닝합니다.

2 **eastus**와 같은 Azure 스토리지 계정의 위치입니다. 기본값은 비어 있습니다. 즉, OpenShift Container Platform 클러스터 위치에 새 Azure 스토리지 계정이 만들어집니다.

3 Azure 스토리지 계정의 SKU 계층입니다 (예: **Standard_LRS**). 기본값은 비어 있습니다. 즉, **Standard_LRS** SKU를 사용하여 새 Azure 스토리지 계정이 만들어집니다.

4 Azure 스토리지 계정 이름입니다. 스토리지 계정이 제공되면 **skuName** 및 **location**이 무시됩니다. 스토리지 계정이 제공되지 않으면 스토리지 클래스는 정의된 **skuName** 및 **location**과 일치하는 계정의 리소스 그룹과 연관된 스토리지 계정을 검색합니다.

7.3.7.1. Azure File 사용시 고려 사항

기본 Azure File 스토리지 클래스는 다음 파일 시스템 기능을 지원하지 않습니다.

- 심볼릭 링크
- 하드 링크
- 확장 속성
- 스파스 파일
- 명명된 파이프

또한 Azure File이 마운트되는 디렉터리의 소유자 ID (UID)는 컨테이너의 프로세스 UID와 다릅니다. 마운트된 디렉터리에 사용할 특정 사용자 ID를 정의하기 위해 **StorageClass** 오브젝트에서 **uid** 마운트 옵션을 지정할 수 있습니다.

다음 **StorageClass** 오브젝트는 마운트된 디렉터리의 심볼릭 링크를 활성화한 상태에서 사용자 및 그룹 ID를 변경하는 방법을 보여줍니다.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azure-file
mountOptions:
  - uid=1500 1
  - gid=1500 2
  - mfsymlinks 3
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus
  skuName: Standard_LRS
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 마운트된 디렉터리에 사용할 사용자 ID를 지정합니다.

2 마운트된 디렉터리에 사용할 그룹 ID를 지정합니다.

3 심볼릭 링크를 활성화합니다.

7.3.8. GCE PersistentDisk (gcePD) 오브젝트 정의

gce-pd-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard 1
  replication-type: none
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
reclaimPolicy: Delete
```

1 **pd-standard** 또는 **pd-ssd** 중 하나를 선택합니다. 기본값은 **pd-standard**입니다.

7.3.9. VMware vSphere 오브젝트 정의

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/vsphere-volume 1
parameters:
  diskformat: thin 2
```

1 OpenShift Container Platform과 함께 VMware vSphere를 사용하는 방법에 대한 자세한 내용은 VMware vSphere 설명서를 참조하십시오.

2 **diskformat**: **thin**, **zeroedthick** 및 **eagerzeroedthick**은 모두 유효한 디스크 형식입니다. 디스크 형식 유형에 대한 자세한 내용은 vSphere 설명서를 참조하십시오. 기본값은 **thin**입니다.

7.4. 기본 스토리지 클래스 변경

AWS를 사용하는 경우 다음 프로세스를 사용하여 기본 스토리지 클래스를 변경합니다. 이 프로세스에서는 두 개의 스토리지 클래스(**gp2** 및 **standard**)가 정의되어 있고 기본 스토리지 클래스를 **gp2**에서 **standard**로 변경하는 것을 가정하고 있습니다.

1. 스토리지 클래스를 나열합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                 TYPE
gp2 (default)        kubernetes.io/aws-ebs (1)
standard             kubernetes.io/aws-ebs
```

(1) **(default)**는 기본 스토리지 클래스를 나타냅니다.

2. 기본 스토리지 클래스에 대해 주석 **storageclass.kubernetes.io/is-default-class**의 값을 **false**로 변경합니다.

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. 주석을 **storageclass.kubernetes.io/is-default-class=true**로 추가하거나 수정하여 다른 스토리지 클래스를 기본값으로 설정합니다.

```
$ oc patch storageclass standard -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

4. 변경 사항을 확인합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                 TYPE
gp2                  kubernetes.io/aws-ebs
standard (default)   kubernetes.io/aws-ebs
```