OpenShift Dedicated 4

스토리지

OpenShift Dedicated 클러스터용 스토리지 구성

Last Updated: 2024-11-03

OpenShift Dedicated 4 스토리지

OpenShift Dedicated 클러스터용 스토리지 구성

법적 공지

초록

이 문서에서는 OpenShift Dedicated 클러스터의 스토리지 설정에 대한 정보를 제공합니다.

차례

1장. OPENSHIFT DEDICATED 스토리지 개요

OpenShift Dedicated는 온프레미스 및 클라우드 공급자를 위해 여러 유형의 스토리지를 지원합니다. OpenShift Dedicated 클러스터에서 영구 및 비영구 데이터에 대한 컨테이너 스토리지를 관리할 수 있습니다.

1.1. OPENSHIFT DEDICATED 스토리지의 일반 용어집

이 용어집은 스토리지 콘텐츠에 사용되는 공통 용어를 정의합니다.

액세스 모드

볼륨 액세스 모드는 볼륨 기능을 설명합니다. 액세스 모드를 사용하여 PVC(영구 볼륨 클레임) 및 PV(영구 볼륨)와 일치시킬 수 있습니다. 다음은 액세스 모드의 예입니다.

- RWO(ReadWriteOnce)
- ReadOnlyMany (ROX)
- ReadWriteMany(RWX)
- ReadWriteOncePod (RWOP)

구성 맵

구성 맵은 구성 데이터를 Pod에 삽입하는 방법을 제공합니다. **ConfigMap** 유형의 볼륨에서 구성 맵에 저장된 데이터를 참조할 수 있습니다. 포드에서 실행되는 애플리케이션에서는 이 데이터를 사용할 수 있습니다.

CSI(Container Storage Interface)

다양한 CO(컨테이너 오케스트레이션) 시스템에서 컨테이너 스토리지를 관리하기 위한 API 사양입니다.

동적 프로비저닝

프레임워크를 사용하면 필요에 따라 스토리지 볼륨을 생성할 수 있으므로 클러스터 관리자가 영구 스토리지를 사전 프로비저닝할 필요가 없습니다.

임시 스토리지

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.

fsGroup

fsGroup은 Pod의 파일 시스템 그룹 ID를 정의합니다.

hostPath

OpenShift Container Platform 클러스터의 hostPath 볼륨은 호스트 노드 파일 시스템의 파일 또는 디렉터리를 Pod에 마운트합니다.

KMS 키

KMS(Key Management Service)는 다양한 서비스에서 데이터 암호화의 필요한 수준을 달성하는 데 도움이 됩니다. KMS 키를 사용하여 데이터를 암호화, 암호 해독 및 재암호화할 수 있습니다.

로컬 볼륨

로컬 볼륨은 디스크, 파티션 또는 디렉터리와 같은 마운트된 로컬 스토리지 장치를 나타냅니다.

OpenShift Data Foundation

내부 또는 하이브리드 클라우드에서 파일, 블록 및 오브젝트 스토리지를 지원하는 OpenShift Container Platform의 영구 스토리지 공급자

영구 스토리지

Pod 및 컨테이너는 작업을 위해 영구 스토리지가 필요할 수 있습니다. OpenShift Dedicated는 Kubernetes PV(영구 볼륨) 프레임워크를 사용하여 클러스터 관리자가 클러스터의 영구 스토리지를 프로비저닝할 수 있도록 합니다. 개발자는 PVC를 사용하여 기본 스토리지 인프라에 대한 특별한 지식 없이도 PV 리소스를 요청할 수 있습니다.

PV(영구 볼륨)

OpenShift Dedicated는 Kubernetes PV(영구 볼륨) 프레임워크를 사용하여 클러스터 관리자가 클러스터의 영구 스토리지를 프로비저닝할 수 있도록 합니다. 개발자는 PVC를 사용하여 기본 스토리지 인프라에 대한 특별한 지식 없이도 PV 리소스를 요청할 수 있습니다.

PVC(영구 볼륨 클레임)

PVC를 사용하여 PersistentVolume을 Pod에 마운트할 수 있습니다. 클라우드 환경에 대한 세부 사항을 모르는 상태에서 스토리지에 액세스할 수 있습니다.

Pod

OpenShift Dedicated 클러스터에서 실행되는 볼륨 및 IP 주소와 같은 공유 리소스가 있는 하나 이상의 컨테이너입니다. Pod는 정의, 배포, 관리되는 가장 작은 컴퓨팅 단위입니다.

회수 정책

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

역할 기반 액세스 제어(RBAC)

RBAC(역할 기반 액세스 제어)는 조직 내의 개별 사용자의 역할에 따라 컴퓨터 또는 네트워크 리소스에 대한 액세스를 규제하는 방법입니다.

상태 비저장 애플리케이션

상태 비저장 애플리케이션은 해당 클라이언트와의 다음 세션에서 사용하기 위해 한 세션에 생성된 클라이언트 데이터를 저장하지 않는 애플리케이션 프로그램입니다.

상태 저장 애플리케이션

상태 저장 애플리케이션은 데이터를 영구 디스크 스토리지에 저장하는 애플리케이션 프로그램입니다. 서버, 클라이언트 및 애플리케이션은 영구 디스크 스토리지를 사용할 수 있습니다. OpenShift Dedicated에서 **Statefulset** 오브젝트를 사용하여 Pod 세트 배포 및 스케일링을 관리하고 이러한 Pod의 순서 및 고유성을 보장할 수 있습니다.

고정 프로비저닝

클러스터 관리자는 여러 PV를 생성합니다. PV에는 스토리지 세부 정보가 포함됩니다. PV는 Kubernetes API에 있으며 사용할 수 있습니다.

스토리지

OpenShift Dedicated는 온프레미스 및 클라우드 공급자를 위해 다양한 유형의 스토리지를 지원합니다. OpenShift Dedicated 클러스터에서 영구 및 비영구 데이터에 대한 컨테이너 스토리지를 관리할 수 있습니다.

스토리지 클래스

스토리지 클래스는 관리자가 제공하는 스토리지 클래스를 설명하는 방법을 제공합니다. 다른 클래스는 서비스 수준, 백업 정책, 클러스터 관리자가 결정하는 임의의 정책에 매핑될 수 있습니다.

1.2. 스토리지 유형

OpenShift Dedicated 스토리지는 임시 스토리지와 영구 스토리지의 두 가지 범주로 광범위하게 분류됩니다.

1.2.1. 임시 스토리지

Pod 및 컨테이너는 본질적으로 임시 또는 일시적이며 스테이트리스(stateless) 애플리케이션을 위해 설

계되었습니다. 임시 스토리지를 사용하면 관리자와 개발자가 일부 작업을 위해 로컬 스토리지를 보다 효과적으로 관리할 수 있습니다. 임시 스토리지 개요, 유형 및 관리에 대한 자세한 내용은 임시 스토리지 이해 를 참조하십시오.

1.2.2. 영구 스토리지

컨테이너에 배포된 상태 저장 애플리케이션에는 영구 스토리지가 필요합니다. OpenShift Dedicated는 클러스터 관리자가 영구 스토리지를 프로비저닝할 수 있도록 PV(영구 볼륨)라는 사전 프로비저닝된 스토리지 프레임워크를 사용합니다. 이러한 볼륨 내의 데이터는 개별 Pod의 라이프사이클을 초과할 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 이해 를 참조하십시오.

1.3. CSI(CONTAINER STORAGE INTERFACE)

CSI는 다양한 컨테이너 오케스트레이션(CO) 시스템에서 컨테이너 스토리지를 관리하기 위한 API 사양입니다. 기본 스토리지 인프라에 대한 구체적인 지식 없이도 컨테이너 네이티브 환경 내에서 스토리지 볼륨을 관리할 수 있습니다. CSI에서 스토리지는 사용 중인 스토리지 벤더에 관계없이 다양한 컨테이너 오케스트레이션 시스템에서 일관되게 작동합니다. CSI에 대한 자세한 내용은 CSI(Container Storage Interface) 사용을 참조하십시오.

1.4. 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 프로비저닝 을 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 사용 가능한 로컬 스토리지의 양을 감지할 수 없습니다.
- Pod는 보장되는 로컬 스토리지를 요청할 수 없습니다.
- 로컬 스토리지는 최상의 리소스입니다.
- Pod는 로컬 스토리지를 채우는 다른 Pod로 인해 제거할 수 있으며 충분한 스토리지를 회수할 때까지 새 Pod가 허용되지 않습니다.

영구 볼륨과 달리 임시 스토리지는 구조화되지 않으며 시스템에서 실행되는 모든 Pod와 시스템, 컨테이너 런타임 및 OpenShift Dedicated에서 다른 용도로도 공간을 공유합니다. 임시 스토리지 프레임워크를 사용하면 Pod에서 일시적인 로컬 스토리지 요구를 지정할 수 있습니다. 또한 OpenShift Dedicated는 적절한 Pod를 예약하고 로컬 스토리지를 과도하게 사용하지 않도록 노드를 보호할 수 있습니다.

임시 스토리지 프레임워크를 사용하면 관리자와 개발자가 로컬 스토리지를 보다 효과적으로 관리할 수 있지만 I/O 처리량 및 대기 시간은 직접적인 영향을 받지 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 Dedicated에서는 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.

요청 및 제한을 지정하여 로컬 임시 스토리지를 관리할 수 있습니다. Pod의 각 컨테이너는 다음을 지정할 수 있습니다.

- **spec.containers[].resources.limits.ephemeral-storage**
- **spec.containers[].resources.requests.ephemeral-storage**

2.3.1. 임시 스토리지 제한 및 요청 단위

임시 스토리지에 대한 제한 및 요청은 바이트 수량으로 측정됩니다. 스토리지를 일반 정수로 표현하거나 E, P, T, G, M, k 접미사 중 하나를 사용하여 고정 소수점 숫자로 표시할 수 있습니다. Ei, Pi, Ti, Gi, Mi, Ki와 동등한 두 가지 강력한 기능을 사용할 수도 있습니다.

예를 들어 다음 수량은 모두 거의 동일한 값인 1289748, 129e6, 129M 및ECDHEMi를 나타냅니다.

중요

각 바이트 수의 접미사는 대소문자를 구분합니다. 올바른 케이스를 사용해야 합니다. "400M"에 사용된 것과 같이 대소문자를 구분하는 "M"을 사용하여 요청을 400 메가바이트로 설정합니다. 대소문자를 구분하는 "400Mi"를 사용하여 400 메비 바이트를 요청합니다. 임시 스토리지의 "400m"을 지정하면 스토리지 요청은 0.4바이트에 불과합니다.

2.3.2. 임시 스토리지 요청 및 제한 예

다음 예제 구성 파일은 두 개의 컨테이너가 있는 Pod를 보여줍니다.

- 각 컨테이너는 2GiB의 로컬 임시 스토리지를 요청합니다.
- 각 컨테이너에는 4GiB의 로컬 임시 스토리지 제한이 있습니다.
- Pod 수준에서 kubelet은 해당 Pod의 모든 컨테이너의 제한을 추가하여 전체 Pod 스토리지 제한을 처리합니다.
  - 이 경우 Pod 수준의 총 스토리지 사용량은 모든 컨테이너의 디스크 사용량 합계와 Pod의 **emptyDir** 볼륨 합계입니다.
  - 따라서 Pod에는 4GiB의 로컬 임시 스토리지 요청 및 로컬 임시 스토리지의 8GiB 제한이 있습니다.

할당량 및 제한이 있는 임시 스토리지 구성의 예

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        ephemeral-storage: "2Gi" (1)
      limits:
        ephemeral-storage: "4Gi" (2)
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  - name: log-aggregator
```

```
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        ephemeral-storage: "2Gi"
      limits:
        ephemeral-storage: "4Gi"
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  volumes:
    - name: ephemeral
      emptyDir: {}
```

1. 로컬 임시 스토리지에 대한 컨테이너 요청입니다.
2. 로컬 임시 스토리지에 대한 컨테이너 제한입니다.

2.3.3. 임시 스토리지 구성 효과 Pod 예약 및 제거

Pod 사양의 설정은 스케줄러가 Pod 예약에 대한 결정을 내리는 방법과 kubelet이 Pod를 제거하는 경우 모두에 영향을 미칩니다.

- 먼저 스케줄러는 예약된 컨테이너의 리소스 요청 합계가 노드의 용량보다 적은지 확인합니다. 이 경우 노드의 사용 가능한 임시 스토리지(항상 리소스)가 4GiB를 초과하는 경우에만 Pod를 노드에 할당할 수 있습니다.
- 두 번째는 컨테이너 수준에서 첫 번째 컨테이너가 리소스 제한을 설정하기 때문에 kubelet 제거 관리자는 이 컨테이너의 디스크 사용량을 측정하고 컨테이너의 스토리지 사용량이 제한을 초과하는 경우 Pod를 제거합니다(4GiB). 또한 kubelet 제거 관리자는 총 사용량이 전체 Pod 스토리지 제한 (8GiB)을 초과하는 경우 제거 Pod를 표시합니다.

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/disk/by-partuuid/4cd1448a-01    69G   32G   34G  49% /
```

3장. 영구 스토리지 이해

3.1. 영구 스토리지 개요

스토리지 관리는 컴퓨팅 리소스 관리와 다릅니다. OpenShift Dedicated는 Kubernetes PV(영구 볼륨) 프레임워크를 사용하여 클러스터 관리자가 클러스터의 영구 스토리지를 프로비저닝할 수 있도록 합니다. 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 Dedicated 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 볼륨과 **같은 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.

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 Dedicated는 다른 모든 기준과 일치하는 최소 PV에 바인딩합니다.

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

파일 수가 많은 영구 볼륨을 Pod에 연결하는 경우 해당 Pod가 실패하거나 시작하는 데 시간이 오래 걸릴 수 있습니다. 자세한 내용은 OpenShift에서 파일 수가 많은 영구 볼륨을 사용할 때 Pod가 시작되지 않거나 "Ready" 상태를 달성하는 데 과도한 시간이 걸리는 이유는 무엇입니까?

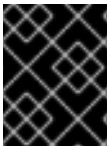
3.2.4. 영구 볼륨 해제

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.

3.2.5. 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 Dedicated 4에서는 **Recycle** 회수 정책이 더 이상 사용되지 않습니다. 기능을 향상하기 위해 동적 프로비저닝이 권장됩니다.

- **삭제** 회수 정책은 OpenShift Dedicated에서 **PersistentVolume** 오브젝트와 Amazon EBS(Amazon EBS) 또는 VMware vSphere와 같은 외부 인프라의 관련 스토리지 자산을 모두 삭제합니다.

참고

동적으로 프로비저닝된 볼륨은 항상 삭제됩니다.

3.2.6. 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 또는 GCE PD 볼륨과 같은 외부 인프라의 관련 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.

3.2.7. 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 Dedicated는 다음 영구 볼륨 플러그인을 지원합니다.

- AWS Elastic Block Store (EBS)
- GCP 영구 디스크
- GCP 파일 저장소

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.

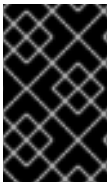
3.3.3. 액세스 모드

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.



중요

볼륨 액세스 모드는 볼륨 기능을 설명합니다. 제한 조건이 적용되지 않습니다. 리소스를 잘못된 사용으로 인한 런타임 오류는 스토리지 공급자가 처리합니다. 공급자에서의 오류는 런타임 시 마운트 오류로 표시됩니다.

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.
ReadWriteOncePod [1]	**RWOP**	볼륨은 단일 노드의 단일 Pod에서 읽기-쓰기로 마운트할 수 있습니다.

1. RWOP는 SELinux 마운트 기능을 사용합니다. 이 기능은 드라이버에 따라 다르며 ODF, AWS EBS, Azure Disk, GCP PD, IBM Cloud Block Storage 볼륨, Cinder 및 vSphere에서 기본적으로 활성화되어 있습니다. 타사 드라이버의 경우 스토리지 공급 업체에 문의하십시오.

표 3.2. 영구 볼륨에 지원되는 액세스 모드

볼륨 플러그인	ReadWriteOnce [1]	ReadWriteOncePod	ReadOnlyMany	ReadWriteMany
AWS EBS [2]	▮	▮		
AWS EFS	▮	▮	▮	▮

볼륨 플러그인	ReadWriteOnce [1]	ReadWriteOncePod	ReadOnlyMany	ReadWriteMany
GCP 영구 디스크	▮	▮		
GCP 파일 저장소	▮	▮	▮	▮
LVM 스토리지	▮	▮		

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. 원시 블록 볼륨만 파이버 채널 및 iSCSI에 대한 RWX(ReadWriteMany) 액세스 모드를 지원합니다. 자세한 내용은 "볼륨 지원 차단"을 참조하십시오.

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.

다음 명령을 실행하여 PV에 바인딩된 PVC의 이름을 볼 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)
- GCE 영구 디스크

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 이름입니다.

2 읽기-쓰기 및 마운트 권한을 정의하는 액세스 모드입니다.

3 PVC에서 사용할 수 있는 스토리지 용량입니다.

4 클레임에 필요한 **StorageClass** 의 이름입니다.

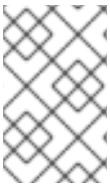
3.4.1. 스토리지 클래스

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 클레임을 사용하는 Pod와 동일한 네임스페이스에 클레임이 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 Dedicated는 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 PV 및 PVC 사양에 **volumeMode:Block**을 지정하여 프로비저닝됩니다.

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
Amazon Elastic Block Store(Amazon EBS)	▮	▮	▮
Amazon Elastic File Storage(Amazon EFS)			
GCP	▮	▮	▮
LVM 스토리지	▮	▮	▮

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)
```

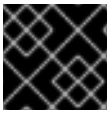
1. 블록 장치에서는 **volumeMounts** 대신 **volumeDevices**가 사용됩니다. **PersistentVolumeClaim** 소스만 원시 블록 볼륨과 함께 사용할 수 있습니다.
2. **mountPath** 대신 **devicePath**가 원시 블록이 시스템에 매핑되는 물리 장치의 경로를 나타냅니다.
3. 볼륨 소스는 **persistentVolumeClaim** 유형이어야 하며 예상되는 PVC의 이름과 일치해야 합니다.

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.

3.6. FSGROUP을 사용하여 POD 타임아웃 감소

스토리지 볼륨에 여러 파일(~1TiB 이상)이 포함된 경우 Pod 시간 초과가 발생할 수 있습니다.

기본적으로 OpenShift Dedicated는 해당 볼륨이 마운트될 때 Pod의 **securityContext** 에 지정된 **fsGroup** 과 일치하도록 각 볼륨의 콘텐츠에 대한 소유권 및 권한을 반복적으로 변경할 수 있습니다. 대규모 볼륨의 경우 소유권 및 권한을 확인하고 변경하는 데 시간이 오래 걸릴 수 있으므로 Pod 시작 속도가

느려질 수 있습니다. **securityContext** 내에서 **fsGroupChangePolicy** 필드를 사용하여 OpenShift Dedicated가 볼륨의 소유권 및 권한을 확인하고 관리하는 방식을 제어할 수 있습니다.

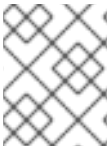
fsGroupChangePolicy 는 Pod 내부에 노출되기 전에 볼륨의 소유권 및 권한을 변경하는 동작을 정의합니다. 이 필드는 **fsGroup**- 제어 소유권 및 권한을 지원하는 볼륨 유형에만 적용됩니다. 이 필드에는 두 가지 가능한 값이 있습니다.

- **OnRootMismatch**: root 디렉터리의 권한 및 소유권이 볼륨의 예상 권한과 일치하지 않는 경우에만 권한 및 소유권을 변경합니다. 이렇게 하면 Pod 타임아웃을 줄이기 위해 소유권 및 볼륨의 권한을 변경하는 데 걸리는 시간을 단축할 수 있습니다.
- **always**: 볼륨이 마운트될 때 볼륨의 권한 및 소유권을 항상 변경합니다.

fsGroupChangePolicy 예

```
securityContext:
  runAsUser: 1000
  runAsGroup: 3000
  fsGroup: 2000
  fsGroupChangePolicy: "OnRootMismatch" 1
  ...
```

1 **OnRootMismatch** 는 재귀 권한 변경을 건너뛰어 Pod 시간 초과 문제를 방지하는 데 도움이 됩니다.



참고

fsGroupChangePolicyfield는 secret, configMap, emptydir과 같은 임시 볼륨 유형에는 영향을 미치지 않습니다.

4장. 영구 스토리지 구성

4.1. AWS ELASTIC BLOCK STORE를 사용하는 영구저장장치

OpenShift Dedicated 클러스터는 Amazon EBS(Amazon Elastic Block Store) 볼륨을 사용하는 4개의 스토리지 클래스로 사전 빌드됩니다. 이러한 스토리지 클래스는 사용할 준비가 되었으며 Kubernetes 및 AWS에 대해 어느 정도 익숙한 것으로 가정합니다.

다음은 사전 빌드된 네 가지 스토리지 클래스입니다.

이름	provisioner
gp2	kubernetes.io/aws-ebs
gp2-csi	ebs.csi.aws.com
gp3 (기본값)	kubernetes.io/aws-ebs
gp3-csi	ebs.csi.aws.com

gp3 스토리지 클래스는 기본값으로 설정되어 있지만 모든 스토리지 클래스를 기본 스토리지 클래스로 선택할 수 있습니다.

Kubernetes 영구 볼륨 프레임워크를 사용하면 관리자는 영구 스토리지로 클러스터를 프로비저닝하고 사용자가 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다. Amazon EBS 볼륨을 동적으로 프로비저닝할 수 있습니다. 영구 볼륨은 단일 프로젝트 또는 네임스페이스에 바인딩되지 않으며 OpenShift Dedicated 클러스터에서 공유할 수 있습니다. 영구 볼륨 클레임은 프로젝트 또는 네임스페이스에 고유하며 사용자가 요청할 수 있습니다. KMS 키를 정의하여 AWS에서 컨테이너-영구 볼륨을 암호화할 수 있습니다. 기본적으로 OpenShift Dedicated 버전 4.10을 사용하는 새로 생성된 클러스터는 gp3 스토리지 및 AWS EBS CSI 드라이버 를 사용합니다.

중요

인프라의 스토리지의 고가용성은 기본 스토리지 공급자가 담당합니다.

4.1.1. 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.

4.1.2. 영구 볼륨 클레임 생성

사전 요구 사항

OpenShift Dedicated에서 볼륨으로 마운트하려면 먼저 기본 인프라에 스토리지가 있어야 합니다.

절차

1. OpenShift Dedicated 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. 이 선택에서는 스토리지 클레임에 대한 읽기 및 쓰기 액세스 권한을 결정합니다.

   d. 스토리지 클레임의 크기를 정의합니다.

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

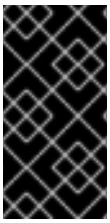
4.1.3. 볼륨 형식

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

이 확인을 통해 OpenShift Dedicated 형식을 처음 사용하기 전에 포맷되지 않은 AWS 볼륨을 영구 볼륨으로 사용할 수 있습니다.

4.1.4. 노드의 최대 EBS 볼륨 수

기본적으로 OpenShift Dedicated는 하나의 노드에 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 계산되므로 각 유형의 최대 39개의 EBS 볼륨을 사용할 수 있습니다.

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

4.1.5. KMS 키를 사용하여 AWS에서 컨테이너 영구 볼륨 암호화

AWS에서 컨테이너-영구 볼륨을 암호화하기 위해 KMS 키를 정의하면 AWS에 배포할 때 명시적인 규정 준수 및 보안 지침이 있는 경우 유용합니다.

사전 요구 사항

- 기본 인프라에는 스토리지가 포함되어야 합니다.
- AWS에서 고객 KMS 키를 생성해야 합니다.

절차

1. 스토리지 클래스를 생성합니다.

```
$ cat << EOF | oc create -f -
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
  name: <storage-class-name> 1
parameters:
  fsType: ext4 2
  encrypted: "true"
  kmsKeyId: keyvalue 3
provisioner: ebs.csi.aws.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
EOF
```

1. 스토리지 클래스의 이름을 지정합니다.
2. 프로비저닝된 볼륨에서 생성된 파일 시스템입니다.
3. container-persistent 볼륨을 암호화할 때 사용할 키의 전체 Amazon Resource Name(ARN)을 지정합니다. 키를 제공하지 않지만 **encrypted** 필드가 **true** 로 설정된 경우 기본 KMS 키가 사용됩니다. AWS 문서의 AWS의 키 ID 및 키 ARN 찾기를 참조하십시오.

2. KMS 키를 지정하는 스토리지 클래스로 PVC(영구 볼륨 클레임)를 생성합니다.

```
$ cat << EOF | oc create -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mypvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  storageClassName: <storage-class-name>
  resources:
    requests:
      storage: 1Gi
EOF
```

3. PVC를 사용할 워크로드 컨테이너를 생성합니다.

```
$ cat << EOF | oc create -f -
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: httpd
      image: quay.io/centos7/httpd-24-centos7
      ports:
        - containerPort: 80
      volumeMounts:
        - mountPath: /mnt/storage
          name: data
  volumes:
    - name: data
```

```
      persistentVolumeClaim:
        claimName: mypvc
EOF
```

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. GCE 영구 디스크를 사용하는 스토리지

OpenShift Dedicated는 GCE 영구 디스크 볼륨(gcePD)을 지원합니다. GCE를 사용하여 영구 스토리지로 OpenShift Dedicated 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 Dedicated 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

4.2.1. 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.

4.2.2. 영구 볼륨 클레임 생성

사전 요구 사항

OpenShift Dedicated에서 볼륨으로 마운트하려면 먼저 기본 인프라에 스토리지가 있어야 합니다.

절차

1. OpenShift Dedicated 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. 이 선택에서는 스토리지 클레임에 대한 읽기 및 쓰기 액세스 권한을 결정합니다.

d. 스토리지 클레임의 크기를 정의합니다.

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

4.2.3. 볼륨 형식

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

이 확인을 사용하면 OpenShift Dedicated 형식을 처음 사용하기 전에 포맷되지 않은 GCE 볼륨을 영구 볼륨으로 사용할 수 있습니다.

5장. CSI(CONTAINER STORAGE INTERFACE) 사용

5.1. CSI 볼륨 구성

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

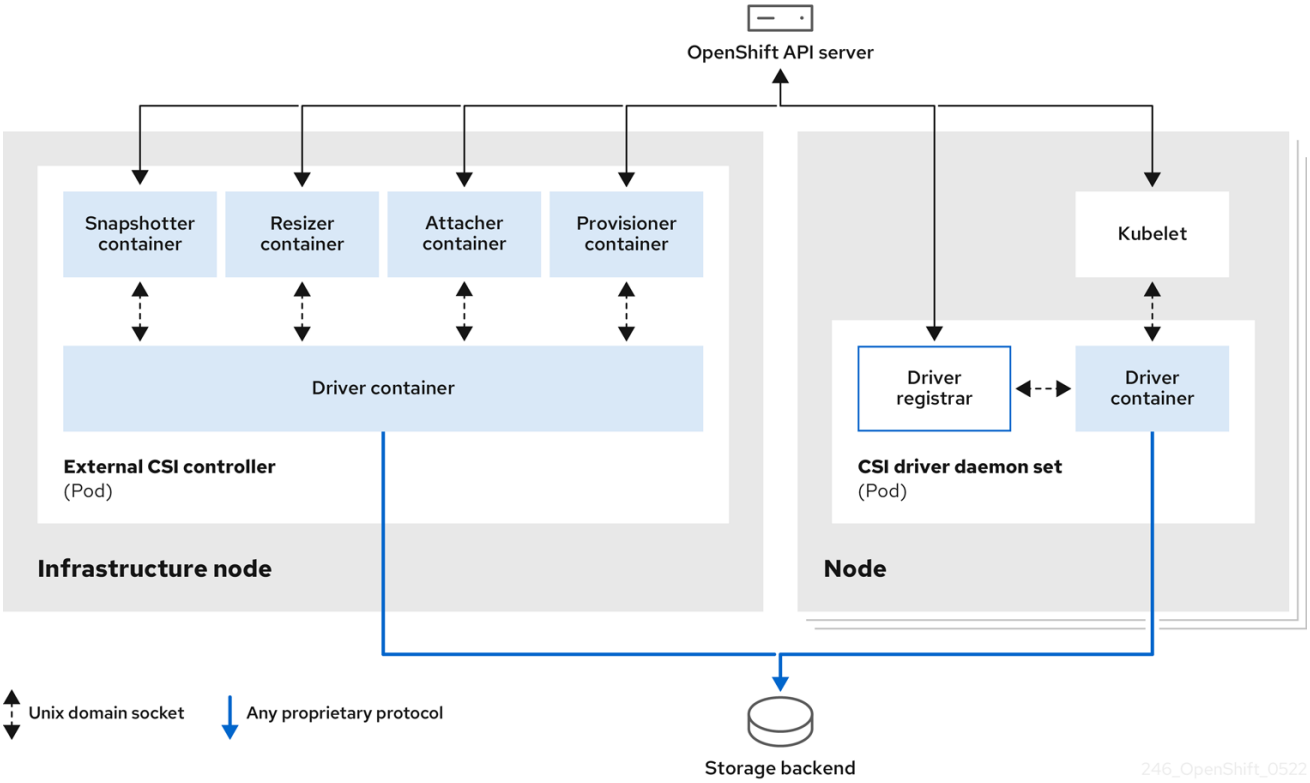
참고

OpenShift Dedicated 4는 CSI 사양 의 버전 1.6.0을 지원합니다.

5.1.1. CSI 아키텍처

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

다음 다이어그램에서는 OpenShift Dedicated 클러스터의 Pod에서 실행되는 구성 요소에 대한 상위 수준 개요를 제공합니다.



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.

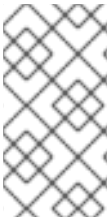
- 스냅샷 컨테이너는 **VolumeSnapshot** 및 **VolumeSnapshotContent** 오브젝트를 감시하고 **VolumeSnapshotContent** 오브젝트의 생성 및 삭제를 담당합니다.

- **PersistentVolumeClaim** 오브젝트에서 더 많은 스토리지를 요청하는 경우 resizer 컨테이너는 **PersistentVolumeClaim** 업데이트를 감시하고 **ControllerExpandVolume** 작업을 CSI 끝점에 대해 트리거하는 사이드카 컨테이너입니다.
- 외부 CSI 연결 컨테이너는 OpenShift Dedicated의 **attach** 및 **detach** 호출을 CSI 드라이버에 대한 각 **ControllerPublish** 및 **ControllerUnpublish** 호출로 변환합니다.
- OpenShift Dedicated의 **provision** 및 **delete** 호출을 CSI 드라이버에 대한 해당 **CreateVolume** 및 **DeleteVolume** 호출로 변환하는 외부 CSI 프로비저너 컨테이너입니다.
- CSI 드라이버 컨테이너입니다.

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

연결,**분리**,**프로비저닝** 및 **삭제** 작업에는 일반적으로 CSI 드라이버가 스토리지 백엔드에 인증 정보를 사용해야 합니다. 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 Dedicated 연결 API를 구현하려면 실행해야 합니다.

5.1.1.2. CSI 드라이버 데몬 세트

CSI 드라이버 데몬 세트는 OpenShift Dedicated가 CSI 드라이버에서 제공하는 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. OpenShift Dedicated는 이러한 호출이 구현된 경우 **NodePublish**/**NodeUnpublish** 및 **NodeStage**/**NodeUnstage** 와 같은 CSI 호출의 노드 플러그인 세트만 사용합니다.

5.1.2. OpenShift Dedicated에서 지원되는 CSI 드라이버

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

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

중요

AWS EFS 및 GCP Filestore CSI 드라이버는 기본적으로 설치되지 않으며 수동으로 설치해야 합니다. AWS EFS CSI 드라이버 설치에 대한 자세한 내용은 AWS Elastic File Service CSI Driver Operator 설정을 참조하십시오. GCP Filestore CSI 드라이버 설치에 대한 자세한 내용은 Google Compute Platform Filestore CSI Driver Operator 를 참조하십시오.

다음 표에서는 OpenShift Dedicated에서 지원하는 CSI 드라이버와 볼륨 스냅샷 및 크기 조정과 같은 CSI 기능을 설명합니다.

중요

다음 표에 CSI 드라이버가 나열되지 않은 경우 지원되는 CSI 기능을 사용하려면 CSI 스토리지 벤더가 제공한 설치 지침을 따라야 합니다.

표 5.1. OpenShift Dedicated에서 지원되는 CSI 드라이버 및 기능

CSI 드라이버	CSI 볼륨 스냅샷	CSI 복제	CSI 크기 조정	인라인 임시 볼륨
AWS EBS	▮		▮	
AWS EFS				
GCP(Google Compute Platform) PD(영구 디스크)	▮	▮	▮	
GCP 파일 저장소	▮		▮	
LVM 스토리지	▮	▮	▮	

5.1.3. 동적 프로비저닝

영구 스토리지의 동적 프로비저닝은 CSI 드라이버 및 기본 스토리지 백엔드의 기능에 따라 달라집니다. CSI 드라이버 공급자는 OpenShift Dedicated에서 스토리지 클래스를 생성하는 방법과 구성에 사용할 수 있는 매개변수를 문서화해야 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. 기본 스토리지 클래스 관리

5.2.1. 개요

기본 스토리지 클래스를 관리하면 다음과 같은 여러 가지 목표를 수행할 수 있습니다.

- 동적 프로비저닝을 비활성화하여 정적 프로비저닝을 강제 적용합니다.
- 기본 스토리지 클래스가 있는 경우 스토리지 Operator가 초기 기본 스토리지 클래스를 다시 생성하지 못하도록 합니다.

- 기본 스토리지 클래스 이름 변경 또는 변경

이러한 목표를 달성하기 위해 **ClusterCSIDriver** 오브젝트에서 **spec.storageClassState** 필드의 설정을 변경합니다. 이 필드에 가능한 설정은 다음과 같습니다.

- **Managed**: (기본값) CSI(Container Storage Interface) Operator는 기본 스토리지 클래스를 적극적으로 관리하므로 클러스터 관리자가 기본 스토리지 클래스에 대한 수동 변경 사항이 제거되고 수동으로 삭제하려는 경우 기본 스토리지 클래스가 지속적으로 생성됩니다.
- **Unmanaged**: 기본 스토리지 클래스를 수정할 수 있습니다. CSI Operator는 스토리지 클래스를 적극적으로 관리하지 않으므로 자동으로 생성되는 기본 스토리지 클래스를 조정하지 않습니다.
- **Removed**: CSI Operator는 기본 스토리지 클래스를 삭제합니다.

5.2.2. 웹 콘솔을 사용하여 기본 스토리지 클래스 관리

사전 요구 사항

- OpenShift Dedicated 웹 콘솔에 액세스합니다.
- cluster-admin 권한을 사용하여 클러스터에 액세스합니다.

절차

웹 콘솔을 사용하여 기본 스토리지 클래스를 관리하려면 다음을 수행합니다.

1. 웹 콘솔에 로그인합니다.
2. **Administration** > **CustomResourceDefinitions** 를 클릭합니다.
3. **CustomResourceDefinitions** 페이지에서 **clustercsidriver** 를 입력하여 **ClusterCSIDriver** 오브젝트를 찾습니다.
4. **ClusterCSIDriver** 를 클릭한 다음 **인스턴스** 탭을 클릭합니다.
5. 원하는 인스턴스의 이름을 클릭한 다음 **YAML** 탭을 클릭합니다.
6. **Managed**,**Unmanaged** 또는 **Removed** 값이 있는 **spec.storageClassState** 필드를 추가합니다.

   예제

```
...
spec:
  driverConfig:
    driverType: ''
  logLevel: Normal
  managementState: Managed
  observedConfig: null
  operatorLogLevel: Normal
  storageClassState: Unmanaged (1)
...
```

   (1) **spec.storageClassState** 필드가 "비관리되지 않음"으로 설정

7. **저장**을 클릭합니다.

5.2.3. CLI를 사용하여 기본 스토리지 클래스 관리

사전 요구 사항

- cluster-admin 권한을 사용하여 클러스터에 액세스합니다.

절차

CLI를 사용하여 스토리지 클래스를 관리하려면 다음 명령을 실행합니다.

```
oc patch clustercsidriver $DRIVERNAME --type=merge -p "{\"spec\":
{\"storageClassState\":\"${STATE}\"}}" 1
```

1 여기서 **${STATE}** 는 "제거" 또는 "관리되지 않음" 또는 "관리되지 않음"입니다.

여기서 **$DRIVERNAME** 은 프로비저너 이름입니다. **oc get sc** 명령을 실행하여 프로비저너 이름을 찾을 수 있습니다.

5.2.4. 존재하지 않거나 여러 기본 스토리지 클래스

5.2.4.1. 여러 기본 스토리지 클래스

기본이 아닌 스토리지 클래스를 기본값으로 표시하고 기존 기본 스토리지 클래스를 설정하지 않거나 기본 스토리지 클래스가 이미 있는 경우 기본 스토리지 클래스를 생성하는 경우 여러 기본 스토리지 클래스가 발생할 수 있습니다. 기본 스토리지 클래스를 여러 개 있는 경우 기본 스토리지 클래스 (**pvc.spec.storageClassName**=nil)를 요청하는 모든 PVC(영구 볼륨 클레임)는 해당 스토리지 클래스의 기본 상태와 관계없이 가장 최근에 생성된 기본 스토리지 클래스를 가져오고 관리자는 여러 기본 스토리지 클래스인 **MultipleDefaultStorageClasses** 가 있는 경고 대시보드에 경고를 받습니다.

5.2.4.2. 기본 스토리지 클래스가 없음

PVC에서 존재하지 않는 기본 스토리지 클래스를 사용할 수 있는 두 가지 가능한 시나리오가 있습니다.

- 관리자는 기본 스토리지 클래스를 제거하거나 기본이 아닌 클래스로 표시한 다음, 사용자가 기본 스토리지 클래스를 요청하는 PVC를 생성합니다.
- 설치 중에 설치 프로그램은 아직 생성되지 않은 기본 스토리지 클래스를 요청하는 PVC를 생성합니다.

이전 시나리오에서는 PVC가 보류 중인 상태로 무기한 유지됩니다. 이 상황을 해결하려면 기본 스토리지 클래스를 생성하거나 기존 스토리지 클래스 중 하나를 기본값으로 선언합니다. 기본 스토리지 클래스가 생성되거나 선언되면 PVC에 새 기본 스토리지 클래스가 제공됩니다. 가능한 경우 PVC는 결국 정적 또는 동적으로 프로비저닝된 PV에 바인딩하고 보류 중 상태로 이동합니다.

5.2.5. 기본 스토리지 클래스 변경

기본 스토리지 클래스를 변경하려면 다음 절차를 사용하십시오.

예를 들어 두 개의 스토리지 클래스인 **gp3** 및 **standard** 가 있고 기본 스토리지 클래스를 **gp3** 에서 **standard** 로 변경하려는 경우.

사전 요구 사항

- cluster-admin 권한을 사용하여 클러스터에 액세스합니다.

절차

기본 스토리지 클래스를 변경하려면 다음을 수행합니다.

1. 스토리지 클래스를 나열합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                TYPE
gp3 (default)       kubernetes.io/aws-ebs (1)
standard            kubernetes.io/aws-ebs
```

(1) **(default)** 는 기본 스토리지 클래스를 나타냅니다.

2. 원하는 스토리지 클래스를 기본값으로 설정합니다.
원하는 스토리지 클래스에 대해 다음 명령을 실행하여 **storageclass.kubernetes.io/is-default-class** 주석을 **true** 로 설정합니다.

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

참고

짧은 시간 동안 여러 개의 기본 스토리지 클래스가 있을 수 있습니다. 그러나 결국 하나의 기본 스토리지 클래스만 있는지 확인해야 합니다.

기본 스토리지 클래스를 여러 개 있는 경우 기본 스토리지 클래스 (**pvc.spec.storageClassName**=nil)를 요청하는 모든 PVC(영구 볼륨 클레임)는 해당 스토리지 클래스의 기본 상태와 관계없이 가장 최근에 생성된 기본 스토리지 클래스를 가져오고 관리자는 여러 기본 스토리지 클래스인 **MultipleDefaultStorageClasses** 가 있는 경고 대시보드에 경고를 받습니다.

3. 이전 기본 스토리지 클래스에서 기본 스토리지 클래스 설정을 제거합니다.
이전 기본 스토리지 클래스의 경우 다음 명령을 실행하여 **storageclass.kubernetes.io/is-default-class** 주석의 값을 **false** 로 변경합니다.

```
$ oc patch storageclass gp3 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-
default-class": "false"}}}'
```

4. 변경 사항을 확인합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                TYPE
gp3                 kubernetes.io/aws-ebs
standard (default)  kubernetes.io/aws-ebs
```

-

5.3. AWS ELASTIC BLOCK STORE CSI DRIVER OPERATOR

5.3.1. 개요

OpenShift Dedicated는 AWS EBS CSI 드라이버 를 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.

AWS EBS 스토리지 자산에 마운트되는 CSI 프로비저닝 PV를 생성하기 위해 OpenShift Dedicated는 **openshift-cluster-csi-drivers** 네임스페이스에 기본적으로 AWS EBS CSI Driver Operator (Red Hat Operator) 및 AWS EBS CSI 드라이버를 설치합니다.

- *AWS EBS CSI Driver Operator*는 기본적으로 PVC를 생성하는 데 사용할 수 있는 StorageClass를 제공합니다. 필요한 경우 이 기본 스토리지 클래스를 비활성화할 수 있습니다(기본 스토리지 클래스 관리참조). Amazon Elastic Block Store를 사용하는 영구 스토리지에 설명된 대로 AWS EBS StorageClass를 생성하는 옵션도 있습니다.
- *AWS EBS CSI 드라이버*를 사용하면 AWS EBS PV를 생성 및 마운트할 수 있습니다.

5.3.2. CSI 정보

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

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

중요

OpenShift Dedicated는 기본적으로 CSI 플러그인을 사용하여 Amazon EBS(Elastic Block Store) 스토리지를 프로비저닝합니다.

OpenShift Dedicated에서 AWS EBS 영구 볼륨을 동적으로 프로비저닝하는 방법에 대한 자세한 내용은 Amazon Elastic Block Store를 사용한 영구 스토리지를 참조하십시오.

추가 리소스

- Amazon Elastic Block Store를 사용하는 영구 스토리지
- CSI 볼륨 구성

5.4. AWS ELASTIC FILE SERVICE CSI DRIVER OPERATOR

중요

이 절차는 OpenShift Dedicated 4.10 이상 버전에만 적용되는 AWS EFS CSI Driver Operator (Red Hat Operator)에만 적용됩니다.

5.4.1. 개요

OpenShift Dedicated는 AWS EFS(Elastic File Service)용 CSI(Container Storage Interface) 드라이버를 사용하여 PV(영구 볼륨)를 프로비저닝할 수 있습니다.

CSI Operator 및 드라이버를 사용할 때는 영구 스토리지 및 CSI 볼륨 구성에 대해 숙지하는 것이 좋습니다.

AWS EFS CSI Driver Operator를 설치한 후 OpenShift Dedicated는 **openshift-cluster-csi-drivers** 네임스페이스에 기본적으로 AWS EFS CSI Operator 및 AWS EFS CSI 드라이버를 설치합니다. 이렇게 하면 AWS EFS CSI Driver Operator에서 AWS EFS 자산에 마운트되는 CSI 프로비저닝 PV를 생성할 수 있습니다.

- *AWS EFS CSI Driver Operator* 는 설치 후 PVC(영구 볼륨 클레임)를 생성하는 데 사용할 스토리지 클래스를 기본적으로 생성하지 않습니다. 그러나 AWS EFS **StorageClass**를 수동으로 생성할 수 있습니다. AWS EFS CSI Driver Operator는 필요에 따라 스토리지 볼륨을 생성할 수 있도록 허용하여 동적 볼륨 프로비저닝을 지원합니다. 이로 인해 클러스터 관리자가 스토리지를 사전 프로비저닝할 필요가 없습니다.
- *AWS EFS CSI 드라이버*를 사용하면 AWS EFS PV를 생성하고 마운트할 수 있습니다.

참고

AWS EFS는 영역 볼륨이 아닌 지역 볼륨만 지원합니다.

5.4.2. CSI 정보

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

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

5.4.3. AWS EFS CSI Driver Operator 설정

1. AWS STS(Secure Token Service)와 함께 AWS EFS를 사용하는 경우 STS에 대한 역할 ARM(Amazon Resource Name)을 가져옵니다. 이는 AWS EFS CSI Driver Operator를 설치하는 데 필요합니다.
2. AWS EFS CSI Driver Operator를 설치합니다.
3. AWS EFS CSI 드라이버를 설치합니다.

5.4.3.1. 보안 토큰 서비스의 Amazon 리소스 이름 가져오기

다음 절차에서는 AWS STS(Security Token Service)에서 OpenShift Dedicated를 사용하여 AWS EFS CSI Driver Operator를 구성하기 위해 역할 ARM(Amazon Resource Name)을 가져오는 방법을 설명합니다.

중요

AWS EFS CSI Driver Operator를 설치하기 전에 다음 절차 *를 수행합니다(AWS EFS CSI Driver Operator 설치* 참조).

사전 요구 사항

- cluster-admin 역할을 가진 사용자로 클러스터에 액세스합니다.
- AWS 계정 인증 정보

절차

1. 다음 콘텐츠를 사용하여 IAM 정책 JSON 파일을 생성합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticfilesystem:DescribeAccessPoints",
        "elasticfilesystem:DescribeFileSystems",
        "elasticfilesystem:DescribeMountTargets",
        "ec2:DescribeAvailabilityZones",
        "elasticfilesystem:TagResource"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticfilesystem:CreateAccessPoint"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:RequestTag/efs.csi.aws.com/cluster": "true"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "elasticfilesystem:DeleteAccessPoint",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/efs.csi.aws.com/cluster": "true"
        }
      }
    }
  ]
}
```

2. 다음 콘텐츠를 사용하여 IAM 신뢰 JSON 파일을 생성합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "Federated": "arn:aws:iam::<your_aws_account_ID>:oidc-
provider/<openshift_oidc_provider>" (1)
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "<openshift_oidc_provider>:sub": [ (2)
            "system:serviceaccount:openshift-cluster-csi-drivers:aws-efs-csi-driver-operator",
            "system:serviceaccount:openshift-cluster-csi-drivers:aws-efs-csi-driver-controller-sa"
          ]
        }
      }
    }
  ]
}
```

(1) AWS 계정 ID와 OpenShift OIDC 공급자 끝점을 지정합니다.

다음 명령을 실행하여 AWS 계정 ID를 가져옵니다.

```
$ aws sts get-caller-identity --query Account --output text
```

다음 명령을 실행하여 OpenShift OIDC 끝점을 가져옵니다.

```
$ openshift_oidc_provider=`oc get authentication.config.openshift.io cluster \
  -o json | jq -r .spec.serviceAccountIssuer | sed -e "s/^https:\/\///"`; \
  echo $openshift_oidc_provider
```

(2) OpenShift OIDC 엔드포인트를 다시 지정합니다.

3. IAM 역할을 생성합니다.

```
ROLE_ARN=$(aws iam create-role \
  --role-name "<your_cluster_name>-aws-efs-csi-operator" \
  --assume-role-policy-document file://<your_trust_file_name>.json \
  --query "Role.Arn" --output text); echo $ROLE_ARN
```

ARN 역할을 복사합니다. AWS EFS CSI Driver Operator를 설치할 때 필요합니다.

4. IAM 정책을 생성합니다.

```
POLICY_ARN=$(aws iam create-policy \
  --policy-name "<your_cluster_name>-aws-efs-csi" \
  --policy-document file://<your_policy_file_name>.json \
  --query 'Policy.Arn' --output text); echo $POLICY_ARN
```

5. IAM 정책을 IAM 역할에 연결합니다.

```
$ aws iam attach-role-policy \
  --role-name "<your_cluster_name>-aws-efs-csi-operator" \
  --policy-arn $POLICY_ARN
```

다음 단계

AWS EFS CSI Driver Operator를 설치합니다.

추가 리소스

- AWS EFS CSI Driver Operator 설치
- AWS EFS CSI 드라이버 설치

5.4.3.2. AWS EFS CSI Driver Operator 설치

AWS EFS CSI Driver Operator(Red Hat Operator)는 기본적으로 OpenShift Dedicated에 설치되지 않습니다. 다음 절차에 따라 클러스터에서 AWS EFS CSI Driver Operator를 설치하고 구성합니다.

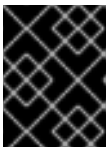
사전 요구 사항

- OpenShift Dedicated 웹 콘솔에 액세스합니다.

절차

웹 콘솔에서 AWS EFS CSI Driver Operator를 설치하려면 다음을 수행합니다.

1. 웹 콘솔에 로그인합니다.
2. AWS EFS CSI Operator를 설치합니다.
   a. **Operators → OperatorHub**를 클릭합니다.
   b. 필터 상자에 AWS EFS CSI를 입력하여 **AWS EFS CSI** Operator를 찾습니다.
   c. **AWS EFS CSI Driver Operator** 버튼을 클릭합니다.



중요

AWS EFS Operator가 아닌 **AWS EFS CSI Driver Operator**를 선택해야 합니다. **AWS EFS Operator**는 커뮤니티 Operator이며 Red Hat에서 지원하지 않습니다.

   a. **AWS EFS CSI Driver Operator** 페이지에서 **설치**를 클릭합니다.
   b. **Operator 설치** 페이지에서 다음을 확인합니다.
      - **역할 ARN** 필드에 AWS EFS(Secure Token Service)를 사용하는 경우 보안 토큰 서비스 프로세스의 *Amazon Resource Name(보안 토큰 서비스) 절차의 마지막 단계에서 복사한 ARN 역할을* 입력합니다.
      - **클러스터의 모든 네임스페이스(기본값)**가 선택됩니다.
      - **설치된 네임스페이스**는 **openshift-cluster-csi-drivers**로 설정됩니다.
   c. **설치**를 클릭합니다.
   설치가 완료되면 AWS EFS CSI Operator가 웹 콘솔의 **설치된 Operators** 섹션에 나열됩니다.

다음 단계

AWS EFS CSI 드라이버를 설치합니다.

5.4.3.3. AWS EFS CSI 드라이버 설치

AWS EFS CSI Driver Operator (Red Hat Operator)를 설치한 후 AWS EFS CSI 드라이버 를 설치합니다.

사전 요구 사항

- OpenShift Dedicated 웹 콘솔에 액세스합니다.

절차

1. **Administration → CustomResourceDefinitions → ClusterCSIDriver** 를 클릭합니다.
2. **Instances** 탭에서 **Create ClusterCSIDriver**를 클릭합니다.
3. 다음 YAML 파일을 사용합니다.

```
apiVersion: operator.openshift.io/v1
kind: ClusterCSIDriver
metadata:
    name: efs.csi.aws.com
spec:
  managementState: Managed
```

4. **생성**을 클릭합니다.
5. 다음 조건이 "True" 상태로 변경될 때까지 기다립니다.
   - AWSEFSDriverNodeServiceControllerAvailable
   - AWSEFSDriverControllerServiceControllerAvailable

5.4.4. AWS EFS 스토리지 클래스 생성

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

AWS EFS CSI Driver Operator (Red Hat Operator) 는 설치 후 기본적으로 스토리지 클래스를 생성하지 않습니다. 그러나 AWS EFS 스토리지 클래스를 수동으로 생성할 수 있습니다.

5.4.4.1. 콘솔을 사용하여 AWS EFS 스토리지 클래스 생성

절차

1. OpenShift Dedicated 콘솔에서 **스토리지 → StorageClasses** 를 클릭합니다.
2. **StorageClasses** 페이지에서 **StorageClass 만들기** 를 클릭합니다.
3. **StorageClass** 페이지에서 다음 단계를 수행합니다.
   a. 스토리지 클래스를 참조할 이름을 입력합니다.
   b. 선택 사항: 설명을 입력합니다.
   c. 회수 정책을 선택합니다.
   d. **Provisioner** 드롭다운 목록에서 **efs.csi.aws.com**을 선택합니다.

e. 선택 사항: 선택한 프로비저너의 구성 매개변수를 설정합니다.

4. **생성**을 클릭합니다.

5.4.4.2. CLI를 사용하여 AWS EFS 스토리지 클래스 생성

절차

- **StorageClass** 오브젝트를 생성합니다.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
parameters:
  provisioningMode: efs-ap (1)
  fileSystemId: fs-a5324911 (2)
  directoryPerms: "700" (3)
  gidRangeStart: "1000" (4)
  gidRangeEnd: "2000" (5)
  basePath: "/dynamic_provisioning" (6)
```

(1) 동적 프로비저닝을 사용하려면 **provisioningMode**가 **efs-ap** 이어야 합니다.

(2) **fileSystemId** 는 수동으로 생성된 EFS 볼륨의 ID여야 합니다.

(3) **directoryPerms**는 볼륨의 루트 디렉터리에 대한 기본 권한입니다. 이 예에서는 소유자가 볼륨에만 액세스할 수 있습니다.

(4) (5) **gidRangeStart** 및 **gidRangeEnd**는 AWS 액세스 지점의 GID를 설정하는 데 사용되는 POSIX 그룹 ID(GID) 범위를 설정합니다. 지정하지 않으면 기본 범위는 50000-7000000입니다. 각 프로비저닝 볼륨이므로 AWS 액세스 지점에는 이 범위의 고유한 GID가 할당됩니다.

(6) **basePath**는 동적으로 프로비저닝된 볼륨을 생성하는 데 사용되는 EFS 볼륨의 디렉터리입니다. 이 경우 PV는 EFS 볼륨에서 "/dynamic_provisioning/<random uuid>"로 프로비저닝됩니다. 하위 디렉터리만 PV를 사용하는 pod에 마운트됩니다.

참고

클러스터 관리자는 각각 다른 EFS 볼륨을 사용하여 여러 **StorageClass** 오브젝트를 생성할 수 있습니다.

5.4.5. AWS에서 EFS 볼륨에 대한 액세스 생성 및 구성

다음 절차에서는 OpenShift Dedicated에서 사용할 수 있도록 AWS에서 EFS 볼륨을 생성하고 구성하는 방법을 설명합니다.

사전 요구 사항

- AWS 계정 인증 정보

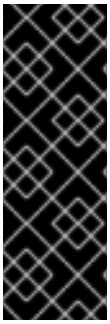
절차

AWS에서 EFS 볼륨에 대한 액세스를 생성하고 구성하려면 다음을 수행합니다.

1. AWS 콘솔에서 https://console.aws.amazon.com/efs을 엽니다.
2. **파일 시스템 생성**을 클릭합니다.
   - 파일 시스템의 이름을 입력합니다.
   - **VPC(Virtual Private Cloud)**의 경우 OpenShift Dedicated의 VPC(Virtual Private Cloud)를 선택합니다.
   - 다른 모든 선택 사항에 대해 기본 설정을 수락합니다.
3. 볼륨 및 마운트 대상이 완전히 생성될 때까지 기다립니다.
   a. https://console.aws.amazon.com/efs#/file-systems로 이동합니다.
   b. 볼륨을 클릭하고 **네트워크** 탭에서 모든 마운트 대상이 사용 가능하게 될 때까지 기다립니다(-1-2분).
4. **네트워크** 탭에서 Security Group ID(다음 단계에서 필요함)를 복사합니다.
5. https://console.aws.amazon.com/ec2/v2/home#SecurityGroups로 이동하여 EFS 볼륨에서 사용하는 보안 그룹을 찾습니다.
6. **인바운드 규칙** 탭에서 인 **바운드 규칙 편집을** 클릭한 다음 다음 설정으로 새 규칙을 추가하여 OpenShift Dedicated 노드가 EFS 볼륨에 액세스할 수 있도록 합니다.
   - **유형**: NFS
   - **프로토콜**: TCP
   - **포트 범위**: 2049
   - **출처**: 노드의 사용자 정의/IP 주소 범위 (예: "10.0.0.0/16")
     이 단계에서는 OpenShift Dedicated에서 클러스터의 NFS 포트를 사용할 수 있습니다.
7. 규칙을 저장합니다.

5.4.6. Amazon Elastic File Storage에 대한 동적 프로비저닝

AWS EFS CSI 드라이버는 다른 CSI 드라이버와 다른 형태의 동적 프로비저닝을 지원합니다. 새 PV를 기존 EFS 볼륨의 하위 디렉터리로 프로비저닝합니다. PV는 서로 독립적입니다. 그러나 모두 동일한 EFS 볼륨을 공유합니다. 볼륨이 삭제되면 프로비저닝된 모든 PV도 삭제됩니다. EFS CSI 드라이버는 이러한 각 하위 디렉터리에 대한 AWS 액세스 지점을 생성합니다. AWS AccessPoint 제한으로 인해 단일 **StorageClass**/EFS 볼륨에서 1000 PV만 동적으로 프로비저닝할 수 있습니다.



중요

PVC.spec.resources는 EFS에 의해 적용되지 않습니다.

아래 예제에서는 5GiB의 공간을 요청합니다. 그러나 생성된 PV는 제한적이며 페타바이트와 같이 원하는 양의 데이터를 저장할 수 있습니다. 손상된 애플리케이션이나 불량 애플리케이션도 볼륨에 너무 많은 데이터를 저장할 경우 상당한 비용이 발생할 수 있습니다.

AWS에서 EFS 볼륨 크기를 모니터링하는 것이 좋습니다.

사전 요구 사항

- Amazon EFS(Elastic File Storage) 볼륨을 생성했습니다.
- AWS EFS 스토리지 클래스를 생성했습니다.

절차

동적 프로비저닝을 활성화하려면 다음을 수행합니다.

- 이전에 생성된 **StorageClass** 를 참조하여 PVC(또는 StatefulSet 또는 Template)를 만듭니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test
spec:
  storageClassName: efs-sc
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

동적 프로비저닝을 설정하는 데 문제가 있는 경우 AWS EFS 문제 해결을 참조하십시오.

추가 리소스

- AWS EFS 볼륨에 대한 액세스 생성 및 구성
- AWS EFS 스토리지 클래스 생성

5.4.7. Amazon Elastic File Storage를 사용하여 정적 PV 생성

동적 프로비저닝 없이 Amazon EFS(Elastic File Storage) 볼륨을 단일 PV로 사용할 수 있습니다. 전체 볼륨이 pod에 마운트됩니다.

사전 요구 사항

- Amazon EFS 볼륨을 생성했습니다.

절차

- 다음 YAML 파일을 사용하여 PV를 생성합니다.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pv
spec:
  capacity: 1
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: efs.csi.aws.com
    volumeHandle: fs-ae66151a 2
    volumeAttributes:
      encryptInTransit: "false" 3
```

1 **spec.capacity**에는 의미가 없으며 CSI 드라이버에서 무시합니다. PVC에 바인딩할 때만 사용됩니다. 애플리케이션은 볼륨에 원하는 양의 데이터를 저장할 수 있습니다.

2 **volumeHandle**은 AWS에서 생성한 EFS 볼륨과 동일해야 합니다. 자체 액세스 지점을 제공하는 경우 **volumeHandle**은 **<EFS volume ID>::<access point ID>** 여야 합니다. 예: **fs-6e633ada::fsap-081a1d293f0004630**.

3 필요한 경우 전송 시 암호화를 비활성화할 수 있습니다. 암호화는 기본적으로 활성화되어 있습니다.

정적 PV를 설정하는 데 문제가 있는 경우 AWS EFS 문제 해결을 참조하십시오.

5.4.8. Amazon Elastic File Storage 보안

다음 정보는 Amazon Elastic File Storage(Amazon EFS) 보안에 중요합니다.

예를 들어 앞에서 설명한 대로 동적 프로비저닝을 사용하여 액세스 지점을 사용하는 경우 Amazon은 파일의 GID를 액세스 지점의 GID로 자동으로 대체합니다. 또한 EFS는 파일 시스템 권한을 평가할 때 액세스 지점의 사용자 ID, 그룹 ID 및 보조 그룹 ID를 고려합니다. EFS는 NFS 클라이언트의 ID를 무시합니다. 액세스 지점에 대한 자세한 내용은 https://docs.aws.amazon.com/efs/latest/ug/efs-access-points.html 을 참조하십시오.

결과적으로 EFS 볼륨은 FSGroup을 자동으로 무시합니다. OpenShift Dedicated는 볼륨의 파일 GID를 FSGroup으로 교체할 수 없습니다. 마운트된 EFS 액세스 지점에 액세스할 수 있는 모든 Pod는 해당 노드의 모든 파일에 액세스할 수 있습니다.

이와 무관하게 전송 중 암호화는 기본적으로 활성화되어 있습니다. 자세한 내용은 https://docs.aws.amazon.com/efs/latest/ug/encryption-in-transit.html을 참조하십시오.

5.4.9. AWS EFS 스토리지 CSI 사용 메트릭

5.4.9.1. 사용량 지표 개요

AWS(Amazon Web Services) EBS(Elastic File Service) 스토리지 CSI(Container Storage Interface) 사용량 메트릭을 사용하면 동적으로 또는 정적으로 프로비저닝된 EFS 볼륨에서 사용하는 공간을 모니터링할 수 있습니다.

중요

이 기능은 메트릭을 켜면 성능이 저하될 수 있으므로 기본적으로 비활성화되어 있습니다.

AWS EFS 사용량 메트릭 기능은 볼륨의 파일을 재귀적으로 진행하여 AWS EFS CSI 드라이버에서 볼륨 지표를 수집합니다. 이 노력으로 성능이 저하될 수 있으므로 관리자는 이 기능을 명시적으로 활성화해야 합니다.

5.4.9.2. 웹 콘솔을 사용하여 사용량 메트릭 활성화

웹 콘솔을 사용하여 AWS(Amazon Web Services) EBS(Elastic File Service) CSI(Container Storage Interface) 사용 지표를 활성화하려면 다음을 수행합니다.

1. **Administration** > **CustomResourceDefinitions** 를 클릭합니다.
2. **Name** 드롭다운 옆에 있는 **CustomResourceDefinitions** 페이지에서 **clustercsidriver** 를 입력합니다.
3. **CRD ClusterCSIDriver** 를 클릭합니다.
4. **YAML** 탭을 클릭합니다.
5. **spec.aws.efsVolumeMetrics.state** 에서 값을 **RecursiveWalk** 로 설정합니다.
   RecursiveWalk 는 볼륨의 파일을 재귀적으로 진행하여 AWS EFS CSI 드라이버의 볼륨 메트릭 컬렉션이 수행됨을 나타냅니다.

   ClusterCSIDriver efs.csi.aws.com YAML 파일의 예

```
spec:
  driverConfig:
    driverType: AWS
    aws:
      efsVolumeMetrics:
        state: RecursiveWalk
        recursiveWalk:
          refreshPeriodMinutes: 100
          fsRateLimit: 10
```

6. 선택 사항: 재귀우가 작동하는 방법을 정의하려면 다음 필드를 설정할 수도 있습니다.
   - **refreshPeriodMinutes**: 볼륨 메트릭의 새로 고침 빈도를 분 단위로 지정합니다. 이 필드를 비워 두면 적절한 기본값이 선택되며, 이는 시간이 지남에 따라 변경될 수 있습니다. 현재 기본값은 240분입니다. 유효한 범위는 1 ~ 43,200 분입니다.
   - **fsRateLimit**: 파일 시스템당 goroutines에서 볼륨 메트릭을 처리하기 위한 속도 제한을 정의합니다. 이 필드를 비워 두면 적절한 기본값이 선택되며, 이는 시간이 지남에 따라 변경될 수 있습니다. 현재 기본값은 5 goroutines입니다. 유효한 범위는 1에서 100 goroutines입니다.
7. **저장**을 클릭합니다.

참고

AWS EFS CSI 사용 지표를 **비활성화하려면** 이전 절차를 사용하지만 **spec.aws.efsVolumeMetrics.state** 의 경우 값을 **RecursiveWalk** 에서 **Disabled** 로 변경합니다.

5.4.9.3. CLI를 사용하여 사용량 메트릭 활성화

CLI를 사용하여 AWS(Amazon Web Services) EBS(Elastic File Service) 스토리지 CSI(Container Storage Interface) 사용 지표를 활성화하려면 다음을 수행합니다.

1. 다음 명령을 실행하여 ClusterCSIDriver를 편집합니다.

```
$ oc edit clustercsidriver efs.csi.aws.com
```

2. **spec.aws.efsVolumeMetrics.state** 에서 값을 **RecursiveWalk** 로 설정합니다.
RecursiveWalk 는 볼륨의 파일을 재귀적으로 진행하여 AWS EFS CSI 드라이버의 볼륨 메트릭 컬렉션이 수행됨을 나타냅니다.

ClusterCSIDriver efs.csi.aws.com YAML 파일의 예

```
spec:
  driverConfig:
    driverType: AWS
    aws:
      efsVolumeMetrics:
        state: RecursiveWalk
        recursiveWalk:
          refreshPeriodMinutes: 100
          fsRateLimit: 10
```

3. 선택 사항: 재귀우가 작동하는 방법을 정의하려면 다음 필드를 설정할 수도 있습니다.

- **refreshPeriodMinutes**: 볼륨 메트릭의 새로 고침 빈도를 분 단위로 지정합니다. 이 필드를 비워 두면 적절한 기본값이 선택되며, 이는 시간이 지남에 따라 변경될 수 있습니다. 현재 기본값은 240분입니다. 유효한 범위는 1 ~ 43,200 분입니다.

- **fsRateLimit**: 파일 시스템당 goroutines에서 볼륨 메트릭을 처리하기 위한 속도 제한을 정의합니다. 이 필드를 비워 두면 적절한 기본값이 선택되며, 이는 시간이 지남에 따라 변경될 수 있습니다. 현재 기본값은 5 goroutines입니다. 유효한 범위는 1에서 100 goroutines입니다.

4. **efs.csi.aws.com** 오브젝트에 변경 사항을 저장합니다.

참고

AWS EFS CSI 사용 지표를 **비활성화하려면** 이전 절차를 사용하지만 **spec.aws.efsVolumeMetrics.state** 의 경우 값을 **RecursiveWalk** 에서 **Disabled** 로 변경합니다.

5.4.10. Amazon Elastic File Storage 문제 해결

다음 정보는 Amazon EFS(Elastic File Storage) 문제 해결 방법에 대한 지침을 제공합니다.

- AWS EFS Operator 및 CSI 드라이버는 **openshift-cluster-csi-drivers**에서 실행됩니다.

- AWS EFS Operator 및 CSI 드라이버의 로그 수집을 시작하려면 다음 명령을 실행합니다.

```
$ oc adm must-gather
[must-gather      ] OUT Using must-gather plugin-in image: quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:125f183d13601537ff15b3239df95d47f0a604da2847b561151fedd699f5e3a5
```

```
[must-gather      ] OUT namespace/openshift-must-gather-xm4wq created
[must-gather      ] OUT clusterrolebinding.rbac.authorization.k8s.io/must-gather-2bd8x
created
[must-gather      ] OUT pod for plug-in image quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:125f183d13601537ff15b3239df95d47f0a604da2847b561151fedd699f5e3a5
created
```

- AWS EFS Operator 오류를 표시하려면 **ClusterCSIDriver** 상태를 확인합니다.

```
$ oc get clustercsidriver efs.csi.aws.com -o yaml
```

- 볼륨을 Pod에 마운트할 수 없는 경우(다음 명령의 출력에 표시된 대로):

```
$ oc describe pod
...
  Type     Reason       Age    From               Message
  ----     ------       ----   ----               -------
  Normal   Scheduled    2m13s  default-scheduler  Successfully assigned default/efs-app to
ip-10-0-135-94.ec2.internal
  Warning  FailedMount  13s    kubelet            MountVolume.SetUp failed for volume "pvc-
d7c097e6-67ec-4fae-b968-7e7056796449" : rpc error: code = DeadlineExceeded desc =
context deadline exceeded (1)
  Warning  FailedMount  10s    kubelet            Unable to attach or mount volumes: unmounted
volumes=[persistent-storage], unattached volumes=[persistent-storage kube-api-access-
9j477]: timed out waiting for the condition
```

(1) 볼륨이 마운트되지 않았음을 나타내는 경고 메시지입니다.

이 오류는 AWS가 OpenShift Dedicated 노드와 Amazon EFS 간에 패킷을 삭제하기 때문에 발생하는 경우가 많습니다.

다음이 올바른지 확인합니다.

- AWS 방화벽 및 보안 그룹
- 네트워킹: 포트 번호 및 IP 주소

5.4.11. AWS EFS CSI Driver Operator 설치 제거

모든 EFS PV는 AWS EFS CSI Driver Operator (Red Hat Operator)를 설치 제거한 후 액세스할 수 없습니다.

사전 요구 사항

- OpenShift Dedicated 웹 콘솔에 액세스합니다.

절차

웹 콘솔에서 AWS EFS CSI Driver Operator를 설치 제거하려면 다음을 수행합니다.

1. 웹 콘솔에 로그인합니다.
2. AWS EFS PV를 사용하는 모든 애플리케이션을 중지합니다.
3. 모든 AWS EFS PV를 삭제합니다.

a. **스토리지 → 영구 볼륨 클레임**을 클릭합니다.

b. AWS EFS CSI Driver Operator에서 사용 중인 각 PVC를 선택하고 PVC 오른쪽에 있는 드롭다운 메뉴를 클릭한 다음 **영구 볼륨 클레임 삭제**를 클릭합니다.

4. AWS EFS CSI 드라이버를 설치 제거합니다.

참고

Operator를 설치 제거하려면 CSI 드라이버를 먼저 제거해야 합니다.

a. **Administration → CustomResourceDefinitions → ClusterCSIDriver** 를 클릭합니다.

b. **인스턴스** 탭에서 **efs.csi.aws.com**의 맨 왼쪽에 있는 드롭다운 메뉴를 클릭한 다음 **ClusterCSIDriver 삭제**를 클릭합니다.

c. 메시지가 표시되면 **삭제**를 클릭합니다.

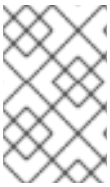
5. AWS EFS CSI Operator를 설치 제거합니다.

a. **Operators → 설치된 Operators**를 클릭합니다.

b. **설치된 Operators** 페이지에서 스크롤하거나 AWS EFS CSI를 **이름으로 검색** 상자에 입력하여 Operator를 찾은 다음 클릭합니다.

c. **설치된 Operator > Operator 세부 정보**페이지 오른쪽 상단에서 **작업 → Operator 설치 제거**를 클릭합니다.

d. **Operator 설치 제거** 창이 표시되면 **제거** 버튼을 클릭하여 네임스페이스에서 Operator를 제거합니다. 클러스터에 Operator가 배포한 애플리케이션을 수동으로 정리해야 합니다.
설치 제거 후 AWS EFS CSI Driver Operator는 더 이상 웹 콘솔의 **설치된 Operator** 섹션에 나열되지 않습니다.



참고

클러스터(**openshift-install destroy cluster**)를 제거하려면 먼저 AWS에서 EFS 볼륨을 삭제해야 합니다. 클러스터의 VPC를 사용하는 EFS 볼륨이 있는 경우 OpenShift Dedicated 클러스터를 삭제할 수 없습니다. Amazon에서는 이러한 VPC를 삭제할 수 없습니다.

5.4.12. 추가 리소스

- CSI 볼륨 구성

5.5. GCP PD CSI DRIVER OPERATOR

5.5.1. 개요

OpenShift Dedicated는 GCP(Google Cloud Platform) PD(영구 디스크) 스토리지용 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.

GCP PD 스토리지 자산에 마운트되는 CSI(영구 볼륨)를 생성하기 위해 OpenShift Dedicated는 **openshift-cluster-csi-drivers** 네임스페이스에 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. 필요한 경우 이 기본 스토리지 클래스를 비활성화할 수 있습니다(기본 스토리지 클래스 관리 참조). GCE 영구 디스크를 사용하는 영구 스토리지에 설명된 대로 GCP PD 스토리지 클래스를 생성하는 옵션도 있습니다.
- **GCP PD 드라이버**: 이 드라이버를 사용하면 GCP PD PV를 생성 및 마운트할 수 있습니다.

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 않고도 표준 인터페이스를 사용하여 스토리지 플러그인을 제공할 수 있습니다.

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

5.5.3. 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 표에서는 OpenShift Dedicated에서 지원하는 모든 GCP PD CSI 스토리지 클래스 매개변수를 설명합니다.

표 5.2. CreateVolume 매개 변수

매개변수	값	기본	설명
type	**PD-ssd**,**pd-standard** 또는 **pd-balanced**	**pd-standard**	표준 PV 또는 솔리드 스테이트 드라이브 PV 중 하나를 선택할 수 있습니다. 드라이버는 값을 확인하지 않으므로 가능한 모든 값이 허용됩니다.
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.

5.5.4. 사용자 정의 암호화 영구 볼륨 생성

PersistentVolumeClaim 오브젝트를 생성할 때 OpenShift Dedicated는 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 Dedicated 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 가져오기 를 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 Dedicated 클러스터에 스토리지 클래스를 배포합니다.

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

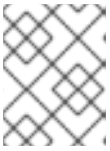
출력 예

```
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.

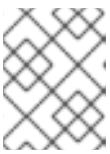
```
$ oc apply -f pvc.yaml
```

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 CMEK 보호 PV를 OpenShift Dedicated 클러스터와 함께 사용할 준비가 되었습니다.

5.5.5. 추가 리소스

- GCE 영구 디스크를 사용하는 스토리지
- CSI 볼륨 구성

5.6. GOOGLE COMPUTE PLATFORM FILESTORE CSI DRIVER OPERATOR

5.6.1. 개요

OpenShift Dedicated는 GCP(Google Compute Platform) 파일 저장소용 CSI(Container Storage Interface) 드라이버를 사용하여 PV(영구 볼륨)를 프로비저닝할 수 있습니다.

CSI Operator 및 드라이버를 사용할 때는 영구 스토리지 및 CSI 볼륨 구성에 대해 숙지하는 것이 좋습니다.

GCP Filestore 스토리지 자산에 마운트되는 CSI 프로비저닝 PV를 생성하려면 **openshift-cluster-csi-drivers** 네임스페이스에 GCP Filestore CSI 드라이버와 GCP Filestore CSI 드라이버를 설치합니다.

- *GCP Filestore CSI Driver Operator* 는 기본적으로 스토리지 클래스를 제공하지 않지만 필요한 경우 스토리지 클래스를 생성할 수 있습니다. GCP Filestore CSI Driver Operator는 필요에 따라 스토리지 볼륨을 생성할 수 있으므로 클러스터 관리자가 스토리지를 사전 프로비저닝할 필요가 없어 동적 볼륨 프로비저닝을 지원합니다.
- *GCP Filestore CSI 드라이버* 를 사용하면 GCP Filestore PV를 생성하고 마운트할 수 있습니다.

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 않고도 표준 인터페이스를 사용하여 스토리지 플러그인을 제공할 수 있습니다.

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

5.6.3. GCP Filestore CSI Driver Operator 설치

GCP(Google Compute Platform) Filestore CSI(Container Storage Interface) Driver Operator는 기본적으로 OpenShift Dedicated에 설치되지 않습니다. 다음 절차에 따라 클러스터에 GCP Filestore CSI Driver Operator를 설치합니다.

사전 요구 사항

- OpenShift Dedicated 웹 콘솔에 액세스합니다.

절차

웹 콘솔에서 GCP Filestore CSI Driver Operator를 설치하려면 다음을 수행합니다.

1. OpenShift Cluster Manager 에 로그인합니다.
2. 클러스터를 선택합니다.
3. **콘솔 열기** 를 클릭하고 인증 정보를 사용하여 로그인합니다.
4. 다음 명령을 실행하여 GCE 프로젝트에서 Filestore API를 활성화합니다.

```
$ gcloud services enable file.googleapis.com  --project <my_gce_project> 1
```

1 **<my_gce_project>**를 Google Cloud 프로젝트로 바꿉니다.

Google Cloud 웹 콘솔을 사용하여 이 작업을 수행할 수도 있습니다.

5. GCP Filestore CSI Operator를 설치합니다.
   a. **Operators → OperatorHub**를 클릭합니다.
   b. 필터 상자에 GCP Filestore를 입력하여 **GCP Filestore** CSI Operator를 찾습니다.
   c. **GCP Filestore CSI Driver Operator** 버튼을 클릭합니다.
   d. **GCP Filestore CSI Driver Operator** 페이지에서 **설치를** 클릭합니다.
   e. **Operator 설치** 페이지에서 다음을 확인합니다.
      - **클러스터의 모든 네임스페이스(기본값)**가 선택됩니다.
      - **설치된 네임스페이스**는 **openshift-cluster-csi-drivers**로 설정됩니다.
   f. **설치**를 클릭합니다.
   설치가 완료되면 GCP Filestore CSI Operator가 웹 콘솔의 **설치된 Operator** 섹션에 나열됩니다.
6. GCP Filestore CSI 드라이버를 설치합니다.
   a. **관리 → CustomResourceDefinitions → ClusterCSIDriver**를 클릭합니다.
   b. **Instances** 탭에서 **Create ClusterCSIDriver**를 클릭합니다.
   다음 YAML 파일을 사용합니다.

```
apiVersion: operator.openshift.io/v1
kind: ClusterCSIDriver
metadata:
    name: filestore.csi.storage.gke.io
spec:
  managementState: Managed
```

   c. **생성**을 클릭합니다.
   d. 다음 조건이 "true" 상태로 변경될 때까지 기다립니다.
      - GCPFilestoreDriverCredentialsRequestControllerAvailable
      - GCPFilestoreDriverNodeServiceControllerAvailable
      - GCPFilestoreDriverControllerServiceControllerAvailable

추가 리소스

- Google 클라우드에서 API 활성화.
- Google Cloud 웹 콘솔을 사용하여 API 활성화.

5.6.4. GCP Filestore 스토리지용 스토리지 클래스 생성

Operator를 설치한 후 GCP(Google Compute Platform) Filestore 볼륨의 동적 프로비저닝을 위한 스토리지 클래스를 생성해야 합니다.

사전 요구 사항

- 실행 중인 OpenShift Dedicated 클러스터에 로그인되어 있습니다.

절차

스토리지 클래스를 생성하려면 다음을 수행합니다.

1. 다음 예제 YAML 파일을 사용하여 스토리지 클래스를 생성합니다.

   YAML 파일의 예

   ```
   kind: StorageClass
   apiVersion: storage.k8s.io/v1
   metadata:
     name: filestore-csi
   provisioner: filestore.csi.storage.gke.io
   parameters:
     connect-mode: DIRECT_PEERING 1
     network: network-name 2
   allowVolumeExpansion: true
   volumeBindingMode: WaitForFirstConsumer
   ```

   1 공유 VPC의 경우 **PRIVATE_SERVICE_ACCESS** 로 설정된 **connect-mode** 매개변수를 사용합니다. 비공유 VPC의 경우 값은 기본 설정인 **DIRECT_PEERING** 입니다.

   2 Filestore 인스턴스를 생성해야 하는 GCP 가상 프라이빗 클라우드(VPC) 네트워크의 이름을 지정합니다.

2. Filestore 인스턴스를 생성해야 하는 VPC 네트워크의 이름을 지정합니다.
   Filestore 인스턴스를 생성해야 하는 VPC 네트워크를 지정하는 것이 좋습니다. VPC 네트워크가 지정되지 않은 경우 CSI(Container Storage Interface) 드라이버는 프로젝트의 기본 VPC 네트워크에 인스턴스를 생성하려고 합니다.

   IPI 설치 시 VPC 네트워크 이름은 일반적으로 접미사 "-network"가 있는 클러스터 이름입니다. 그러나 UPI 설치 시 VPC 네트워크 이름은 사용자가 선택한 모든 값이 될 수 있습니다.

   공유 VPC(**connect-mode** = **PRIVATE_SERVICE_ACCESS**)의 경우 네트워크가 전체 VPC 이름이어야 합니다. 예: **projects/shared-vpc-name/global/networks/gcp-filestore-network**.

   다음 명령을 사용하여 **MachineSets** 오브젝트를 검사하여 VPC 네트워크 이름을 확인할 수 있습니다.

   ```
   $ oc -n openshift-machine-api get machinesets -o yaml | grep "network:"
             - network: gcp-filestore-network
   (...)
   ```

   이 예에서 이 클러스터의 VPC 네트워크 이름은 "gcp-filestore-network"입니다.

5.6.5. 클러스터 및 GCP 파일 저장소 삭제

일반적으로 클러스터를 삭제하면 OpenShift Dedicated 설치 프로그램이 해당 클러스터에 속하는 모든 클라우드 리소스를 삭제합니다. 그러나 GCP(Google Compute Platform) Filestore 리소스의 특수 특성으로 인해 자동화된 정리 프로세스가 드문 경우 모두 제거되지 않을 수 있습니다.

따라서 제거 프로세스에서 모든 클러스터 소유 Filestore 리소스가 삭제되었는지 확인하는 것이 좋습니다.

절차

모든 GCP Filestore PVC가 삭제되었는지 확인하려면 다음을 수행하십시오.

1. GUI 또는 CLI를 사용하여 Google Cloud 계정에 액세스합니다.
2. **kubernetes-io-cluster-${CLUSTER_ID}=owned** 라벨을 사용하여 모든 리소스를 검색합니다. 클러스터 ID는 삭제된 클러스터에 고유하므로 해당 클러스터 ID가 있는 나머지 리소스가 없어야 합니다.
3. 예기치 않은 경우 나머지 리소스가 있으며 삭제합니다.

5.6.6. 추가 리소스

- CSI 볼륨 구성

6장. 일반 임시 볼륨

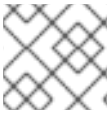
6.1. 개요

일반 임시 볼륨은 영구 볼륨 및 동적 프로비저닝을 지원하는 모든 스토리지 드라이버에서 제공할 수 있는 임시 볼륨의 유형입니다. 일반 임시 볼륨은 일반적으로 프로비저닝 후 비어 있는 스크래치 데이터를 위한 포드별 디렉터리를 제공하는 **emptyDir** 볼륨과 유사합니다.

일반 임시 볼륨은 Pod 사양에 인라인으로 지정되며 Pod의 라이프사이클을 따릅니다. Pod와 함께 생성 및 삭제됩니다.

일반 임시 볼륨에는 다음과 같은 기능이 있습니다.

- 스토리지는 로컬 또는 네트워크 연결일 수 있습니다.
- 볼륨은 Pod가 초과할 수 없는 고정된 크기를 가질 수 있습니다.
- 볼륨에는 드라이버 및 매개변수에 따라 일부 초기 데이터가 있을 수 있습니다.
- 드라이버에서 스냅샷, 복제, 크기 조정, 스토리지 용량 추적 등 일반적인 볼륨 작업이 지원된다고 가정합니다.



참고

일반 임시 볼륨은 오프라인 스냅샷 및 크기 조정을 지원하지 않습니다.

6.2. 라이프사이클 및 영구 볼륨 클레임

볼륨 클레임의 매개변수는 Pod의 볼륨 소스 내에서 허용됩니다. PVC(영구 볼륨 클레임)에 대한 레이블, 주석 및 전체 필드 세트가 지원됩니다. 이러한 Pod가 생성되면 임시 볼륨 컨트롤러에서 Pod와 동일한 네임스페이스에 표시된 템플릿(*일반 임시 볼륨 생성* 프로시저 생성)에서 실제 PVC 오브젝트를 생성하고 Pod가 삭제될 때 PVC가 삭제되었는지 확인합니다. 이렇게 하면 다음 두 가지 방법 중 하나로 볼륨 바인딩 및 프로비저닝을 트리거합니다.

- 스토리지 클래스가 즉시 볼륨 바인딩을 사용하는 경우 즉시.
  스케줄러는 즉각적인 바인딩을 통해 사용 가능한 후 볼륨에 액세스할 수 있는 노드를 선택해야 합니다.
- Pod가 노드에 영구적으로 예약되는 경우(**WaitForFirstConsumervolume** 바인딩 모드).
  이 볼륨 바인딩 옵션은 일반 임시 볼륨에 사용하는 것이 좋습니다. 스케줄러는 Pod에 적합한 노드를 선택할 수 있기 때문입니다.

리소스 소유권 측면에서 일반 임시 스토리지가 있는 Pod는 해당 임시 스토리지를 제공하는 PVC의 소유자입니다. Pod가 삭제되면 Kubernetes 가비지 수집기는 PVC를 삭제합니다. 그러면 일반적으로 스토리지 클래스의 기본 회수 정책이 볼륨을 삭제하기 때문에 볼륨 삭제를 트리거합니다. 보존 정책의 회수 정책이 있는 스토리지 클래스를 사용하여 quasi-ephemeral 로컬 스토리지를 생성할 수 있습니다. 즉, 스토리지는 Pod를 비우며, 이 경우 볼륨 정리가 별도로 수행되는지 확인해야 합니다. 이러한 PVC가 존재하지만 다른 PVC처럼 사용할 수 있습니다. 특히 볼륨 복제 또는 스냅샷에서 데이터 소스로 참조될 수 있습니다. PVC 오브젝트에는 볼륨의 현재 상태도 있습니다.

추가 리소스

- 일반 임시 볼륨 생성

6.3. 보안

일반 임시 볼륨 기능을 활성화하여 Pod를 생성할 수 있는 사용자가 PVC(영구 볼륨 클레임)를 간접적으로 생성할 수 있습니다. 이 기능은 이러한 사용자에게 PVC를 직접 생성할 수 있는 권한이 없는 경우에도 작동합니다. 클러스터 관리자는 이 사실을 알고 있어야 합니다. 보안 모델에 맞지 않는 경우 일반 임시 볼륨이 있는 Pod와 같은 오브젝트를 거부하는 승인 Webhook를 사용합니다.

PVC에 대한 일반 네임스페이스 할당량은 여전히 적용되므로 사용자가 이 새 메커니즘을 사용할 수 있더라도 다른 정책을 우회하는 데 사용할 수 없습니다.

6.4. 영구 볼륨 클레임 이름 지정

자동으로 생성된 PVC(영구 볼륨 클레임)는 Pod 이름과 볼륨 이름의 조합으로 이름이 지정되며 중간에 하이픈(-)이 있습니다. 이 이름 지정 규칙에 따라 다른 Pod 간 및 Pod와 수동으로 생성된 PVC 간에 잠재적으로 충돌이 발생할 수 있습니다.

예를 들어 볼륨 **스크래치** 가 있는 **pod -a** 및 볼륨 **a-scratch** 가 모두 동일한 PVC 이름 **pod-a-scratch** 로 끝납니다.

이러한 충돌이 감지되며 Pod용으로 생성된 경우에만 PVC가 임시 볼륨에 사용됩니다. 이 검사는 소유권 관계를 기반으로 합니다. 기존 PVC를 덮어쓰거나 수정하지 않지만 충돌을 해결하지 않습니다. 올바른 PVC가 없으면 Pod를 시작할 수 없습니다.

중요

이름이 충돌하지 않도록 동일한 네임스페이스 내에서 Pod와 볼륨의 이름을 지정할 때 주의하십시오.

6.5. 일반 임시 볼륨 생성

절차

1. **Pod** 오브젝트 정의를 생성하여 파일에 저장합니다.
2. 파일에 일반 임시 볼륨 정보를 포함합니다.

my-example-pod-with-generic-vols.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: my-app
spec:
  containers:
    - name: my-frontend
      image: busybox:1.28
      volumeMounts:
      - mountPath: "/mnt/storage"
        name: data
      command: [ "sleep", "1000000" ]
  volumes:
    - name: data (1)
      ephemeral:
        volumeClaimTemplate:
```

```
    metadata:
      labels:
        type: my-app-ephvol
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "gp2-csi"
      resources:
        requests:
          storage: 1Gi
```

1 일반 임시 볼륨 클레임.

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 Dedicated 영구 볼륨 프레임워크를 사용하면 이 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 Dedicated에서 많은 스토리지 유형을 영구 볼륨으로 사용할 수 있습니다. 관리자가 모두 정적으로 프로비저닝할 수 있지만 일부 유형의 스토리지는 기본 제공 공급자 및 플러그인 API를 사용하여 동적으로 생성됩니다.

7.2. 사용 가능한 동적 프로비저닝 플러그인

OpenShift Dedicated는 클러스터의 구성된 공급자의 API를 사용하여 새 스토리지 리소스를 생성하는 동적 프로비저닝을 위한 일반적인 구현이 있는 다음 프로비저너 플러그인을 제공합니다.

스토리지 유형	프로비저너 플러그인 이름	참고
Amazon Elastic Block Store(Amazon 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.
GCE Persistent Disk (gcePD)	**kubernetes.io/gce-pd**	다중 영역 구성에서는 현재 클러스터에 노드가 없는 영역에서 PV가 생성되지 않도록 GCE 프로젝트당 하나의 OpenShift Dedicated 클러스터를 실행하는 것이 좋습니다.
IBM Power® Virtual Server Block	**powervs.csi.ibm.com**	설치 후 IBM Power® Virtual Server Block CSI Driver Operator 및 IBM Power® Virtual Server Block CSI Driver가 동적 프로비저닝에 필요한 스토리지 클래스를 자동으로 생성합니다.

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** 오브젝트의 기본 정의와 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: <storage-class-name> (3)
  annotations: (4)
    storageclass.kubernetes.io/is-default-class: 'true'
    ...
provisioner: kubernetes.io/aws-ebs (5)
parameters: (6)
  type: gp3
...
```

(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 클러스터는 두 개 이상의 스토리지 클래스를 보유할 수 있지만, 이 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. AWS Elastic Block Store (EBS) 객체 정의

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 2
  iopsPerGB: "10" 3
  encrypted: "true" 4
  kmsKeyId: keyvalue 5
  fsType: ext4 6
```

1. (필수) 스토리지 클래스의 이름입니다. 영구 볼륨 클래임은 이 스토리지 클래스를 사용하여 관련 영구 볼륨을 프로비저닝합니다.
2. (필수) **io1**,**gp3**,**sc1**,**st1** 중에서 선택합니다. 기본값은 **gp3** 입니다. 유효한 Amazon Resource Name (ARN) 값은 AWS 설명서를 참조하십시오.
3. 선택 사항: **io1** 볼륨만 해당합니다. GiB마다 초당 I/O 작업 수입니다. AWS 볼륨 플러그인은 볼륨의 IOPS를 계산하기 위해 요청된 볼륨의 크기와 함께 이 값을 곱합니다. 값의 상한은 AWS가 지원하는 최대치인 20,000 IOPS입니다. 자세한 내용은 AWS 설명서를 참조하십시오.
4. 선택 사항: EBS 볼륨을 암호화할지 여부를 나타냅니다. 유효한 값은 **true** 또는 **false**입니다.
5. 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.
6. 선택 사항: 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. GCE PersistentDisk (gcePD) 오브젝트 정의

gce-pd-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard 2
  replication-type: none
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
reclaimPolicy: Delete
```

1. StorageClass의 이름입니다. 영구 볼륨 클래임은 이 스토리지 클래스를 사용하여 관련 영구 볼륨을 프로비저닝합니다.
2. **pd-standard** 또는 **pd-ssd** 중 하나를 선택합니다. 기본값은 **pd-standard**입니다.

7.4. 기본 스토리지 클래스 변경

기본 스토리지 클래스를 변경하려면 다음 절차를 사용하십시오.

예를 들어 두 개의 스토리지 클래스인 **gp3** 및 **standard** 가 있고 기본 스토리지 클래스를 **gp3** 에서 **standard** 로 변경하려는 경우.

사전 요구 사항

- cluster-admin 권한을 사용하여 클러스터에 액세스합니다.

절차

기본 스토리지 클래스를 변경하려면 다음을 수행합니다.

1. 스토리지 클래스를 나열합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                 TYPE
gp3 (default)        kubernetes.io/aws-ebs (1)
standard             kubernetes.io/aws-ebs
```

(1) **(default)** 는 기본 스토리지 클래스를 나타냅니다.

2. 원하는 스토리지 클래스를 기본값으로 설정합니다.
원하는 스토리지 클래스에 대해 다음 명령을 실행하여 **storageclass.kubernetes.io/is-default-class** 주석을 **true** 로 설정합니다.

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

참고

짧은 시간 동안 여러 개의 기본 스토리지 클래스가 있을 수 있습니다. 그러나 결국 하나의 기본 스토리지 클래스만 있는지 확인해야 합니다.

기본 스토리지 클래스를 여러 개 있는 경우 기본 스토리지 클래스 (**pvc.spec.storageClassName**=nil)를 요청하는 모든 PVC(영구 볼륨 클레임)는 해당 스토리지 클래스의 기본 상태와 관계없이 가장 최근에 생성된 기본 스토리지 클래스를 가져오고 관리자는 여러 기본 스토리지 클래스인 **MultipleDefaultStorageClasses** 가 있는 경고 대시보드에 경고를 받습니다.

3. 이전 기본 스토리지 클래스에서 기본 스토리지 클래스 설정을 제거합니다.
이전 기본 스토리지 클래스의 경우 다음 명령을 실행하여 **storageclass.kubernetes.io/is-default-class** 주석의 값을 **false** 로 변경합니다.

```
$ oc patch storageclass gp3 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-
default-class": "false"}}}'
```

4. 변경 사항을 확인합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                 TYPE
gp3                  kubernetes.io/aws-ebs
standard (default)   kubernetes.io/aws-ebs
```