Red Hat Ceph Storage 5

스토리지 전략 가이드

Red Hat Ceph Storage 클러스터용 스토리지 전략 생성

Last Updated: 2022-08-14

Red Hat Ceph Storage 5 스토리지 전략 가이드

Red Hat Ceph Storage 클러스터용 스토리지 전략 생성

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

법적 공지

초록

이 문서에서는 CRUSH 계층 구조 생성, 배치 그룹 추정, 생성할 스토리지 풀 유형 결정, 풀 관리 등 스토리지 전략을 생성하는 방법을 설명합니다. Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지에서 참조하십시오.

차례

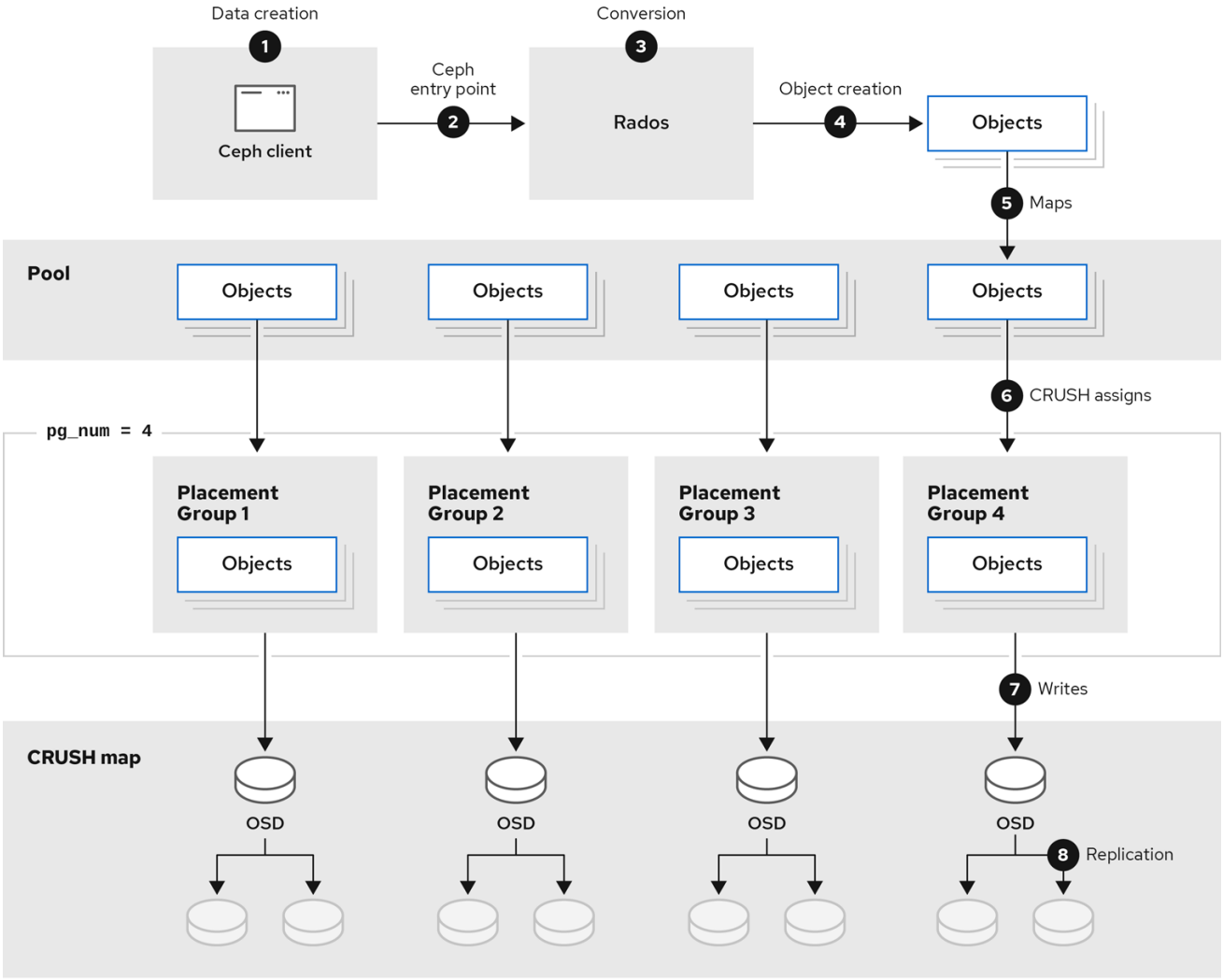
1장. 개요

Ceph 클라이언트의 관점에서 Ceph 스토리지 클러스터와 상호 작용하는 것은 매우 간단합니다.

1. 클러스터에 연결
2. 풀 I/O 컨텍스트 생성

이 간단한 인터페이스는 Ceph 클라이언트가 사용자가 정의한 스토리지 전략 중 하나를 선택하는 방법입니다. 스토리지 전략은 스토리지 용량 및 성능을 제외한 모든 Ceph 클라이언트와 볼 수 없습니다.

아래 다이어그램은 클라이언트에서 Red Hat Ceph Storage 클러스터로 시작하는 논리 데이터 흐름을 보여줍니다.



1.1. 스토리지 전략이란 무엇입니까?

스토리지 전략은 특정 사용 사례에 서비스를 제공하는 데이터를 저장하는 방법입니다. 예를 들어, OpenStack과 같은 클라우드 플랫폼에 대한 볼륨 및 이미지를 저장해야 하는 경우 SSD 기반 저널을 사용하여 합리적으로 수행하는 SAS 드라이브에 데이터를 저장할 수 있습니다. 반대로, S3- 또는 Swift 호환 게이트웨이에 대한 오브젝트 데이터를 저장해야 하는 경우 SATA 드라이브와 같이 보다 경제적 인 것을 사용할 수 있습니다. Ceph는 동일한 Ceph 클러스터에 두 시나리오를 모두 수용할 수 있지만 클라우드 플랫폼에 SAS/SSD 스토리지 전략(예: OpenStack의 Glance 및 Cinder)을 제공하고 개체 저장소에 대한 SATA 스토리지를 제공하는 수단이 필요합니다.

스토리지 전략에는 스토리지 미디어(하드 드라이브, SSD, 나머지), 스토리지 미디어의 성능 및 장애 도메인, 배치 그룹 수, 풀 인터페이스 등이 있습니다. Ceph는 여러 스토리지 전략을 지원합니다. 사용 사례, 비용/해제 성능 장단점 및 데이터 내구성은 스토리지 전략을 구동하는 주요 고려 사항입니다.

1. **사용 사례:** Ceph는 대용량 스토리지 용량을 제공하며 다양한 사용 사례를 지원합니다. 예를 들어 Ceph Block Device 클라이언트는 OpenStack과 같은 클라우드 플랫폼의 주요 스토리지 백엔드입니다. copy-on-write 복제와 같은 고성능 기능을 사용하여 볼륨 및 이미지의 제한 없는 스토리지를 제공합니다. 마찬가지로 Ceph는 OpenShift 환경에 컨테이너 기반 스토리지를 제공할 수 있습니다. 반면, Ceph Object Gateway 클라이언트는 오디오, 비트맵, 비디오 및 기타 데이터와 같은 개체에 대해 RESTful S3 호환 및 Swift 호환 개체 스토리지를 제공하는 클라우드 플랫폼의 선도적인 스토리지 백엔드입니다.

2. **비용/성능:** 빠른 것이 좋습니다. 더 큰 것이 더 좋습니다. 높은 내구성이 더 좋습니다. 그러나 각 감독질에 대한 가격이 있으며 해당 가격 / 거래 비용이 있습니다. 성능 관점에서 다음과 같은 사용 사례를 고려해 보십시오. SSD는 상대적으로 소량의 데이터와 저널링을 위해 매우 빠른 스토리지를 제공할 수 있습니다. 데이터베이스 또는 오브젝트 인덱스를 저장하면 매우 빠른 SSD 풀의 이점을 얻을 수 있지만 다른 데이터에는 너무 비용이 많이 듭니다. SSD 저널링을 사용하는 SAS 드라이브는 볼륨과 이미지의 경제적인 가격으로 빠른 성능을 제공합니다. SSD 저널링이 없는 SATA 드라이브는 전체 성능이 낮은 저렴한 스토리지를 제공합니다. OSD의 CRUSH 계층 구조를 생성할 때 사용 사례와 허용 가능한 비용/성능 문제를 고려해야 합니다.

3. **내구성:** 대규모 클러스터에서 하드웨어 장애는 예외가 아니라 예상되는 것입니다. 그러나 데이터 손실 및 서비스 중단은 허용되지 않습니다. 이러한 이유로 데이터 지속성이 매우 중요합니다. Ceph는 객체의 여러 깊은 사본이나 삭제 및 여러 코딩 청크를 사용하여 데이터 내구성을 해결합니다. 여러 사본 또는 여러 개의 코딩 청크가 추가 비용/분리점의 장단점을 제공합니다. 즉, 적은 수의 사본이나 코딩 청크를 저장하는 것이 더 저렴하지만 성능이 저하된 상태로 서비스 쓰기 요청이 실패할 수 없게 될 수 있습니다. 일반적으로 두 개의 추가 사본(즉, **size = 3**) 또는 두 개의 코딩 청크가 있는 하나의 오브젝트를 사용하면 클러스터가 복구되는 동안 클러스터가 성능이 저하된 상태에서 쓰기를 수행할 수 있습니다. CRUSH 알고리즘은 Ceph가 클러스터 내의 다른 위치에 청크를 저장 또는 코딩할 수 있도록 하여 이 프로세스를 지원합니다. 이렇게 하면 단일 스토리지 장치 또는 노드의 오류가 발생하면 데이터 손실을 방지하기 위해 필요한 모든 사본 또는 코딩 청크가 손실되지 않습니다.

사용 사례, 비용/고정 성능 장단점 및 스토리지 전략의 데이터 내구성을 캡처하여 스토리지 풀로 Ceph 클라이언트에 제공할 수 있습니다.

중요

Ceph의 개체 복사 또는 코딩 청크는 RAID를 더 이상 사용하지 않습니다. Ceph는 이미 데이터 지속성을 처리하므로 RAID를 사용하지 마십시오. 성능이 저하된 RAID는 성능에 부정적인 영향을 미치며 RAID를 사용하여 데이터를 복구하는 것은 깊은 복사본 또는 코딩 청크를 사용하는 것보다 훨씬 느립니다.

1.2. 스토리지 전략 구성

스토리지 전략 구성은 Ceph OSD를 CRUSH 계층 구조에 할당하고 풀에 대한 배치 그룹 수를 정의하는 것입니다. 일반적인 단계는 다음과 같습니다.

1. **스토리지 전략을 정의: 스토리지 전략**에서는 사용 사례, 비용/완전 성능 장단점 및 데이터 내성을 분석해야 합니다. 그런 다음 이 사용 사례에 적합한 OSD를 만듭니다. 예를 들어 고성능 풀에 대해 SSD 지원 OSD를 생성할 수 있습니다. SAS는 고성능 블록 장치 볼륨 및 이미지를 위한 저널 지원 OSD, 또는 저렴한 스토리지용 SATA 지원 OSD를 생성할 수 있습니다. 사용 사례의 각 OSD는 일관된 성능 프로파일을 갖도록 하드웨어 구성이 동일해야 합니다.

2. **CRUSH 계층 구조:** Ceph 규칙은 일반적으로 CRUSH 계층에서 **루트** 인 노드를 선택하고 배치 그룹 및 포함된 개체를 저장하기 위한 적절한 OSD를 식별합니다. 스토리지 전략에 대한 CRUSH 계

층 구조와 CRUSH 규칙을 생성해야 합니다. CRUSH 계층 구조는 CRUSH 규칙 설정을 통해 풀에 직접 할당됩니다.

3. **배치 그룹 계산:** Ceph shard를 배치 그룹으로 나타냄. 풀에 적절한 수의 배치 그룹을 설정해야 하며 동일한 CRUSH 규칙에 여러 개의 풀을 할당하는 경우 정상 최대 배치 그룹 수에 남아 있어야 합니다.

4. **풀 생성:** 마지막으로 풀을 생성하고 복제 또는 삭제 코딩된 스토리지를 사용하는지 확인해야 합니다. 풀의 배치 그룹 수, 풀에 대한 규칙, 크기 또는 **K+M** 코딩 청크와 같은 내구성을 설정해야 합니다.

풀은 스토리지 클러스터에 대한 Ceph 클라이언트 인터페이스이지만 용량 및 성능을 제외하고 스토리지 전략은 Ceph 클라이언트에 완전히 투명합니다.

2장. CRUSH 관리

CRUSH(Controlled Replication Under scalable Hashing) 알고리즘은 컴퓨팅 데이터 스토리지 위치로 데이터를 저장하고 검색하는 방법을 결정합니다.

	충분히 발전된 기술은 매직과 다를 수 있습니다.	
		-- Arthur C. Clarke

2.1. CRUSH 소개

스토리지 클러스터의 CRUSH 맵은 CRUSH 계층 구조 내의 장치 위치와 Ceph가 데이터를 저장하는 방식을 결정하는 각 계층 구조에 대한 규칙을 설명합니다.

CRUSH 맵에는 하나 이상의 노드 계층 구조가 포함되어 있습니다. Ceph의 "buckets"라고 하는 계층 구조의 노드는 해당 유형에 의해 정의된 스토리지 위치 집계입니다. 예를 들어 행, 랙, 섀시, 호스트, 장치 등이 있습니다. 계층 구조의 각 리프는 기본적으로 스토리지 장치 목록에 있는 스토리지 장치 중 하나로 구성됩니다. 리프는 항상 하나의 노드 또는 "bucket"에 포함됩니다. CRUSH 맵에는 CRUSH가 데이터를 저장하고 검색하는 방법을 결정하는 규칙 목록도 있습니다.

참고

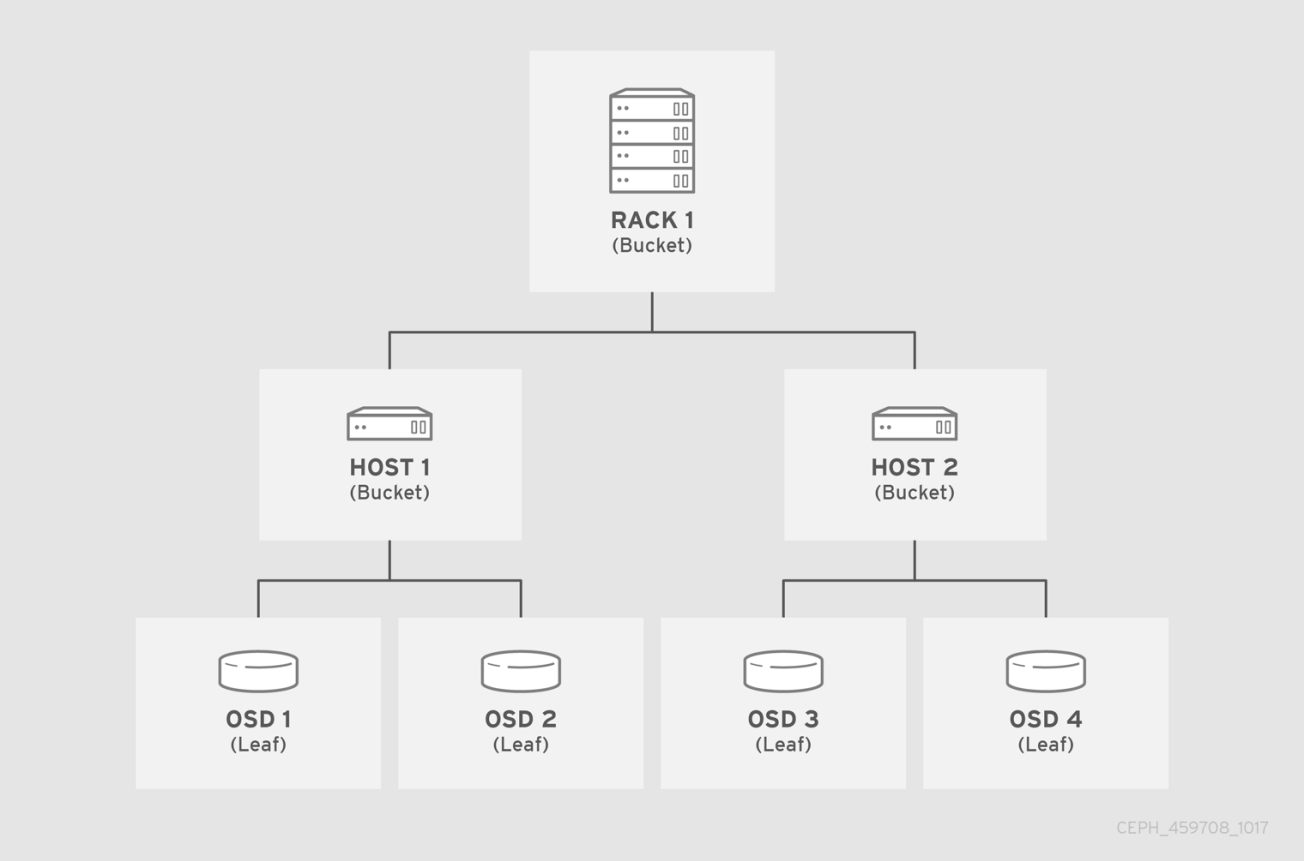
클러스터에 OSD를 추가할 때 스토리지 장치가 CRUSH 맵에 추가됩니다.

CRUSH 알고리즘은 장치별 가중치 값에 따라 스토리지 장치에 데이터 오브젝트를 분배하고 균일한 확률 배포를 오케스트레이션합니다. CRUSH는 관리자가 정의하는 계층적 클러스터 매핑에 따라 개체 및 복제 및 해당 복제본 또는 삭제 코드 청크를 배포합니다. CRUSH 맵은 규칙에 맞게 이를 포함하는 사용 가능한 스토리지 장치와 논리 버킷을 나타내며 규칙을 사용하는 각 풀을 확장합니다.

장애 도메인 또는 성능 도메인에서 배치 그룹을 OSD에 매핑하기 위해 CRUSH 맵은 생성된 CRUSH 맵의 유형 아래에 있는 계층적 버킷 **유형** 목록을 정의합니다. 버킷 계층 구조를 생성하는 목적은 실패 도메인 또는 성능 도메인 또는 둘 다로 리프 노드를 분리하는 것입니다. 장애 도메인에는 호스트, 섀시, 랙, 전원 분배 장치, Pod, 행, 방 및 데이터 센터가 포함됩니다. 성능 도메인에는 특정 구성의 장애 도메인 및 OSD가 포함됩니다. 예를 들어, SSD는 SSD 저널, SATA 드라이브 등을 통해 구동됩니다. 장치에는 **hdd**,**ssd** 및 **nvme** 와 같은 **클래스** 가 있으므로 장치 클래스를 사용하여 CRUSH 계층 구조를 보다 신속하게 구축할 수 있습니다.

OSD를 나타내는 리프 노드를 제외하고 나머지 계층 구조는 임의의 것이며 기본 유형이 요구 사항에 맞지 않는 경우 고유한 요구에 따라 정의할 수 있습니다. CRUSH 맵 버킷 유형을 조직의 하드웨어 이름 지정 규칙에 맞게 조정하고 물리적 하드웨어 이름을 반영하는 인스턴스 이름을 사용하는 것이 좋습니다. 이름 지정 방법을 사용하면 OSD 또는 기타 하드웨어가 작동하지 않고 관리자가 호스트 또는 기타 하드웨어에 대한 원격 또는 물리적 액세스가 필요할 때 클러스터를 쉽게 관리하고 문제를 해결할 수 있습니다.

다음 예에서 버킷 계층에는 4개의 리프 버킷(**osd 1-4**), 두 노드 버킷(**호스트 1-2**) 및 1개의 랙 노드(**rack 1**)가 있습니다.



leaf 노드는 CRUSH 맵의 시작 부분에 있는 **장치 목록에 선언된 스토리지 장치**를 반영하므로 이를 버킷 인스턴스로 선언할 필요가 없습니다. 계층 구조에서 두 번째로 낮은 버킷 유형은 일반적으로 장치를 집계합니다. 즉, 일반적으로 스토리지 미디어를 포함하는 컴퓨터이며 관리자가 선호하는 용어를 사용하여 "node", "server,", "host", "machine" 등과 같이 설명합니다. 밀도가 높은 환경에서는 카드 및 섀시당 여러 호스트/노드를 확인하는 것이 점점 더 일반적입니다. 예를 들어, 노드가 실패하면 여러 호스트/노드와 해당 OSD를 차단할 수 있는 경우 카드 또는 섀시를 가져와야 하는 경우 카드 및 섀시도 실패했는지 확인합니다.

버킷 인스턴스를 선언할 때 해당 유형을 문자열로 지정하고, 고유한 이름을 문자열로 지정하고, 음수 정수로 표시되는 선택적 고유 ID를 할당하고, 항목의 총 용량 또는 기능을 기준으로 가중치를 지정하고, **straw2**와 같은 버킷 알고리즘을 지정합니다. **버킷은 하나 이상의 항목을 포함할 수 있습니다. 항목은 노드 버킷 또는 나사로 구성될 수 있습니다. 항목의 상대적 가중치를 반영하는 가중치를 포함할 수 있습니다.**

2.1.1. 동적 데이터 배치

Ceph 클라이언트 및 Ceph OSD는 모두 CRUSH 맵과 CRUSH 알고리즘을 사용합니다.

- **Ceph 클라이언트: CRUSH를 Ceph 클라이언트에 매핑하여 Ceph 클라이언트가 OSD와 직접 통신할 수 있도록 지원합니다. 즉, Ceph 클라이언트는 단일 장애 지점, 성능 병목, 중앙 집중식 조회 서버에 대한 연결 제한, 스토리지 클러스터 확장성에 대한 물리적 제한을 수행할 수 있는 중앙 집중식 오브젝트 조회 테이블을 사용하지 않습니다.**

- **Ceph OSD: CRUSH를 Ceph OSD에 배포하면 Ceph OSD를 통해 OSD에서 복제, 백필링 및 복구를 처리할 수 있습니다. 즉, Ceph OSD는 Ceph 클라이언트를 대신하여 개체 복제본(또는 청**

크 코딩)의 스토리지를 처리합니다. 또한 Ceph OSD는 클러스터의 재조정(백업)을 재조정하고 오류를 동적으로 복구하기 위해 클러스터에 대해 충분히 알고 있음을 의미합니다.

2.1.2. 실패 도메인 설정

여러 개체 복제본이 있거나 코딩 청크를 사용하면 데이터 손실을 방지할 수 있지만 고가용성 문제를 해결하기에 충분하지 않습니다. CRUSH는 Ceph Storage Cluster의 기본 물리적 조직을 반영하여 장치 장애와 관련된 주소상 가능한 소스를 모델링할 수 있습니다. CRUSH 배치 정책은 클러스터의 토폴로지를 클러스터 맵으로 인코딩하여 서로 다른 실패 도메인에서 개체 복제본을 분리하거나 축소하는 코딩 청크를 유지하면서 원하는 의사-랜덤 배포를 계속 유지할 수 있습니다. 예를 들어 동시 오류가 발생할 가능성을 해결하려면 데이터 복제본 또는 삭제 코딩 청크가 다른 보류, 랙, 전원 공급 장치, 컨트롤러 또는 물리적 위치를 사용하는 장치에 있는지 확인하는 것이 좋습니다. 이렇게 하면 데이터 손실을 방지하고 클러스터 성능이 저하된 상태에서 작동할 수 있습니다.

2.1.3. 성능 도메인 설정

Ceph는 여러 계층을 지원하여 한 유형의 하드웨어 성능 프로필과 다른 유형의 하드웨어 성능 프로필을 분리할 수 있습니다. 예를 들어 CRUSH는 하드 디스크 드라이브 및 SSD에 대한 다른 계층 구조를 생성할 수 있습니다. 기본 하드웨어의 성능 프로파일을 고려한 성능 도메인 - 다양한 성능 특성을 지원할 필요성으로 인해 널리 사용되고 있습니다. 이러한 맵은 두 개 이상의 루트 유형 버킷이 있는 CRUSH 맵일 뿐입니다. 사용 사례 예제는 다음과 같습니다.

- VM: OpenStack, CloudStack, ProxMox 또는 OpenNebula와 같은 클라우드 플랫폼에 백엔드 역할을 하는 Ceph 호스트는 XFS와 같은 가장 안정적이고 성능 파일 시스템(예: SAS 드라이브의 XFS)을 사용하는 경향이 있습니다. XFS는 저널을 위해 저널링을 위해 파티션된 고성능 SSD를 동시에 게시하지 않기 때문입니다. 일관된 성능 프로필을 유지하려면 이러한 사용 사례를 CRUSH 계층 구조에서 유사한 하드웨어를 집계해야 합니다.

- 오브젝트 스토리지 : S3 및 Swift 인터페이스의 오브젝트 스토리지 백엔드로 사용되는 Ceph 호스트는 SATA 드라이브와 같은 비용이 적게 드는 스토리지 미디어(오브젝트 스토리지)에 적합하지 않을 수 있습니다. 또한 개체 스토리지의 hugeabyte당 비용으로 더 많은 경제적 스토리지 호스트를 분리하면서 클라우드 플랫폼에서 볼륨 및 이미지를 저장하기 위한 더 많은 경제적 스토리지 호스트를 분리할 수 있습니다. HTTP는 오브젝트 스토리지 시스템의 병목 현상이되는 경향이 있습니다.

- 콜드 스토리지: 자주 액세스하는 콜드 스토리지 데이터 또는 완화 성능 요구 사항을 사용하여 데이터 검색용으로 설계되었습니다. 이 시스템은 비용이 적게 드는 스토리지 미디어 및 지리저 코딩을 활용합니다. 그러나 삭제 코딩에는 약간의 추가 RAM 및 CPU가 필요할 수 있으므로, 개체 스토리지 또는 VM에 사용되는 호스트와 RAM 및 CPU 요구 사항이 다를 수 있습니다.

- SSD 지원 풀: SSD는 비용이 많이 들지만 하드 디스크 드라이브에 비해 상당한 이점을 제공합니다. SSD에는 검색 시간이 없으며 전체 처리량이 높습니다. 저널링에 SSD를 사용하는 것 외에도 클러스터에서 SSD 지원 풀을 지원할 수 있습니다. 일반적인 사용 사례에는 고성능 SSD 풀

이 있습니다. 예를 들어 **SATA** 드라이브 대신 **Ceph Object Gateway**의 **.rgw.buckets.index** 풀을 **SSD**에 매핑할 수 있습니다.

CRUSH 맵은 장치 클래스의 개념을 지원합니다. **Ceph**는 스토리지 장치의 측면을 검색하고 **hdd,ssd** 또는 **nvme** 와 같은 클래스를 자동으로 할당할 수 있습니다. 그러나 **CRUSH**는 이러한 기본값으로 제한되지 않습니다. 예를 들어, **nmap** 계층 구조를 사용하여 다양한 유형의 워크로드를 구분할 수도 있습니다. 예를 들어, **SSD**는 저널 또는 **write-ahead** 로그, 버킷 인덱스 또는 원시 오브젝트 스토리지에 사용할 수 있습니다. **CRUSH**는 **ssd-bucket-index** 또는 **ssd-object-storage** 와 같은 다양한 장치 클래스를 지원할 수 있으므로 **Ceph**는 다른 워크로드에 동일한 스토리지 미디어를 사용하지 않으므로 보다 예측 가능하고 일관성을 조정할 수 있습니다.

2.1.3.1. 다른 장치 클래스 사용

성능 도메인을 생성하려면 장치 클래스와 단일 **CRUSH** 계층 구조를 사용합니다. **CRUSH** 계층에 **OSD**를 추가한 후 다음을 수행합니다.

1. 각 장치에 클래스를 추가합니다. 예를 들어 다음과 같습니다.

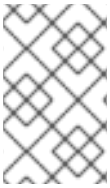
```
ceph osd crush set-device-class <class> <osdId> [<osdId>]
ceph osd crush set-device-class hdd osd.0 osd.1 osd.4 osd.5
ceph osd crush set-device-class ssd osd.2 osd.3 osd.6 osd.7
```

2. 그런 다음 장치를 사용할 규칙을 만듭니다.

```
ceph osd crush rule create-replicated <rule-name> <root> <failure-domain-type> <class>
ceph osd crush rule create-replicated cold default host hdd
ceph osd crush rule create-replicated hot default host ssd
```

3. 마지막으로 규칙을 사용하도록 풀을 설정합니다.

```
ceph osd pool set <poolname> crush_rule <rule-name>
ceph osd pool set cold_tier crush_rule cold
ceph osd pool set hot_tier crush_rule hot
```



참고

CRUSH 맵을 수동으로 편집할 필요가 없습니다.

2.2. CRUSH 계층 구조

CRUSH 맵은 지시된 그래프이므로 여러 계층(예: 성능 도메인)을 수용할 수 있습니다. **CRUSH** 계층을 만들고 수정하는 가장 쉬운 방법은 **Ceph CLI**를 사용하는 것입니다. 그러나 **CRUSH** 맵을 컴파일하고 다시 컴파일하고 다시 컴파일하고 활성화할 수도 있습니다.

Ceph CLI를 사용하여 버킷 인스턴스를 선언할 때 유형을 지정하고 고유한 문자열 이름을 지정해야 합니다. **Ceph**는 버킷 **ID**를 자동으로 할당하고 알고리즘을 **straw2** 로 설정하고, **rjenkins1** 을 반영하고 가중치를 설정합니다. 컴파일된 **CRUSH** 맵을 수정할 때 버킷에 음수 정수(선택 사항), 버킷 알고리즘(일반적으로 **straw2**)에 상대적인 가중치를 지정하고, 해시 알고리즘 **rjenkins1**을 반영합니다.

버킷은 하나 이상의 항목을 포함할 수 있습니다. 항목은 노드 버킷(예: 랙, 행, 호스트)으로 구성되거나 (예: **OSD** 디스크) 남겨 둘 수 있습니다. 항목의 상대적 가중치를 반영하는 가중치를 포함할 수 있습니다.

역컴파일된 **CRUSH** 맵을 수정할 때 다음 구문을 사용하여 노드 버킷을 선언할 수 있습니다.

```
[bucket-type] [bucket-name] {
    id [a unique negative numeric ID]
    weight [the relative capacity/capability of the item(s)]
    alg [the bucket type: uniform | list | tree | straw2 ]
    hash [the hash type: 0 by default]
    item [item-name] weight [weight]
}
```

예를 들어 위의 다이어그램을 사용하여 두 개의 호스트 버킷과 하나의 랙 버킷을 정의합니다. **OSD**는 호스트 버킷 내의 항목으로 선언됩니다.

```
host node1 {
    id -1
    alg straw2
    hash 0
    item osd.0 weight 1.00
    item osd.1 weight 1.00
}

host node2 {
    id -2
    alg straw2
    hash 0
    item osd.2 weight 1.00
    item osd.3 weight 1.00
}

rack rack1 {
    id -3
    alg straw2
    hash 0
```

```
    item node1 weight 2.00
    item node2 weight 2.00
}
```

참고

foregoing 예에서 **rack** 버킷에는 **OSD**가 포함되어 있지 않습니다. 대신 더 낮은 수준의 호스트 버킷을 포함하며, 항목 항목의 가중치 합계를 포함합니다.

2.2.1. CRUSH 위치

CRUSH 위치는 **CRUSH** 맵의 계층 구조 측면에서 **OSD**의 위치입니다. 명령줄 인터페이스에서 **CRUSH** 위치를 표현하면 **CRUSH** 위치 지정자는 **OSD**의 위치를 설명하는 이름/값 쌍 목록의 형식을 취합니다. 예를 들어 **OSD**가 특정 행, 랙, 섀시스 및 호스트에 있고 기본 **CRUSH** 트리의 일부인 경우 해당 **crush** 위치를 다음과 같이 설명할 수 있습니다.

```
root=default row=a rack=a2 chassis=a2a host=a2a1
```

참고:

1. 키 순서는 중요하지 않습니다.

2. 키 이름(= 0의 왼쪽)은 유효한 **CRUSH** 유형 이어야 합니다. 기본적으로 여기에는 루트,데이터 센터 ,방,행,Pod,pdu,랙,섀시 및 호스트가 포함됩니다. **ArgoCD** 맵을 편집하여 필요에 따라 유형을 변경할 수 있습니다.

3. 모든 버킷/키를 지정할 필요는 없습니다. 예를 들어 기본적으로 **Ceph**는 **ceph-osd** 데몬의 위치를 **root=default host={HOSTNAME}** 으로 설정합니다(호스트 이름 **-s**의 출력에 따라).

2.2.2. Bucket 추가

CRUSH 계층 구조에 버킷 인스턴스를 추가하려면 버킷 이름과 해당 유형을 지정합니다. **CRUSH** 맵에서 버킷 이름은 고유해야 합니다.

```
ceph osd crush add-bucket {name} {type}
```

예를 들어 다양한 하드웨어 성능 프로파일에 대해 여러 계층을 사용하려는 경우 하드웨어 또는 사용 사례에 따라 버킷 이름을 지정하는 것이 좋습니다.

예를 들어 **SSD** 저널(**Hdd-journal**)이 있는 **SAS** 디스크의 계층 구조 및 **SATA** 드라이브(**hdd**)의 계층 구조를 생성할 수 있습니다.

```
ceph osd crush add-bucket ssd-root root
ceph osd crush add-bucket hdd-journal-root root
ceph osd crush add-bucket hdd-root root
```

Ceph CLI 출력은 다음과 같습니다.

```
added bucket ssd-root type root to crush map
added bucket hdd-journal-root type root to crush map
added bucket hdd-root type root to crush map
```

중요

버킷 이름에 콜론(**:**)을 사용하는 것은 지원되지 않습니다.

계층 구조에 필요한 각 버킷 유형의 인스턴스를 추가합니다. 다음 예제에서는 **SSD** 호스트 랙이 있는 행에 버킷을 추가하고 오브젝트 스토리지의 호스트 랙을 추가하는 방법을 보여줍니다.

```
ceph osd crush add-bucket ssd-row1 row
ceph osd crush add-bucket ssd-row1-rack1 rack
ceph osd crush add-bucket ssd-row1-rack1-host1 host
ceph osd crush add-bucket ssd-row1-rack1-host2 host
ceph osd crush add-bucket hdd-row1 row
ceph osd crush add-bucket hdd-row1-rack2 rack
ceph osd crush add-bucket hdd-row1-rack1-host1 host
ceph osd crush add-bucket hdd-row1-rack1-host2 host
ceph osd crush add-bucket hdd-row1-rack1-host3 host
ceph osd crush add-bucket hdd-row1-rack1-host4 host
```

이 단계를 완료하면 트리를 확인합니다.

```
ceph osd tree
```

계층 구조는 플랫으로 유지됩니다. **CRUSH** 맵에 추가한 후 버킷을 계층적 위치로 이동해야 합니다.

2.2.3. Bucket 이동

초기 클러스터를 생성할 때 **Ceph**에는 **default** 라는 루트 버킷이 있는 기본 **CRUSH** 맵이 있고 초기

OSD 호스트가 기본 버킷에 표시됩니다. CRUSH 맵에 버킷 인스턴스를 추가하면 CRUSH 계층에 표시되지만 특정 버킷 아래에 표시되지는 않습니다.

CRUSH 계층 구조의 특정 위치로 버킷 인스턴스를 이동하려면 버킷 이름과 해당 유형을 지정합니다. 예를 들어 다음과 같습니다.

```
ceph osd crush move ssd-row1 root=ssd-root
ceph osd crush move ssd-row1-rack1 row=ssd-row1
ceph osd crush move ssd-row1-rack1-host1 rack=ssd-row1-rack1
ceph osd crush move ssd-row1-rack1-host2 rack=ssd-row1-rack1
```

이 단계를 완료하면 트리를 볼 수 있습니다.

```
ceph osd tree
```

참고

또한 **ceph osd crush create-or- 0.0/16**을 사용하여 **OSD**를 이동하는 동안 위치를 생성할 수도 있습니다.

2.2.4. Bucket 제거

CRUSH 계층에서 버킷 인스턴스를 제거하려면 버킷 이름을 지정합니다. 예를 들어 다음과 같습니다.

```
ceph osd crush remove {bucket-name}
```

또는 다음을 수행합니다.

```
ceph osd crush rm {bucket-name}
```

참고

버킷은 이를 제거하기 위해 비어 있어야 합니다.

상위 수준 버킷(예: 기본)을 제거하는 경우 풀에서 해당 버킷을 선택하는 CRUSH 규칙을 사용하는지 확인합니다. 이 경우 CRUSH 규칙을 수정해야 합니다. 그렇지 않으면 피어링이 실패합니다.

2.2.5. 버킷 알고리즘

Ceph CLI를 사용하여 버킷을 생성할 때 **Ceph**는 기본적으로 알고리즘을 **straw2** 로 설정합니다. **Ceph**는 **4**개의 버킷 알고리즘을 지원하며, 각각 성능과 재조직 효율성 간의 절충을 나타냅니다. 사용할 버킷 유형이 확실하지 않은 경우 **straw2** 버킷을 사용하는 것이 좋습니다. 버킷 알고리즘은 다음과 같습니다.

1. **균일성:** 정확히 동일한 가중치를 가진 고유 버킷 집계 장치. 예를 들어, 기업들이 위임하거나 해제하는 경우 일반적으로 동일한 물리적 구성(예: 대량 구매)을 사용하는 많은 시스템에서 이러한 작업을 수행합니다. 스토리지 장치의 가중치가 정확히 동일한 경우, **CRUSH**가 복제본을 일관된 시간 내에 일관된 버킷에 매핑할 수 있도록 균일한 버킷 유형을 사용할 수 있습니다. 고유하지 않은 가중치를 사용하면 다른 버킷 알고리즘을 사용해야 합니다.

2. **list:** 버킷이 콘텐츠를 연결된 목록으로 집계합니다. **RUSH (Replication Under Hashing) $_P$** 알고리즘을 기반으로 하는 목록은 확장되는 클러스터에 대한 자연적이고 직관적인 선택입니다. 즉, 개체가 적절한 확률을 가진 최신 장치로 재배치되거나 이전 장치에 남아 있습니다. 버킷에 항목이 추가될 때 최적의 데이터 마이그레이션 결과를 얻을 수 있습니다. 그러나 목록의 중간 또는 뒷부분에서 제거된 항목은 상당한 양의 불필요한 이동을 초래할 수 있으며, 목록 버킷은 결코 줄어들지 않는 상황에 가장 적합하게 또는 거의 축소 될 수 있습니다.

3. **트리:** 트리 버킷은 바이너리 검색 트리를 사용합니다. 버킷에 더 큰 항목 세트가 포함된 경우 버킷을 나열하는 것보다 효율적입니다. **RUSH(Replication Under Hashing) $_R$** 알고리즘을 기반으로 트리 버킷은 배치 시간을 **0(log $_n$)**으로 줄여 훨씬 더 큰 장치 또는 중첩 버킷을 관리하는 데 적합합니다.

4. **Straw2(기본값):** 목록 및 트리 버킷은 목록 시작 부분에 특정 항목 우선 순위를 부여하는 방식으로 분할 및 정량 전략을 사용합니다(예: 목록 시작 부분에 있거나 전체 하위 트리를 전혀 고려해야 할 필요성). 이로 인해 복제본 배치 프로세스의 성능이 향상되지만 항목의 추가, 제거 또는 재**weighting**으로 인해 버킷의 내용이 변경될 때 하위 결정 재조직 동작을 도입할 수도 있습니다. **straw2** 버킷 유형을 사용하면 프로세스가 서로 유사한 진행을 통해 복제본 배치를 위해 모든 항목이 서로 상당히 "영향"할 수 있습니다.

2.3. CRUSH의 CEPH OSD

OSD의 **CRUSH** 계층 구조가 있으면 **CRUSH** 계층에 **OSD**를 추가합니다. 기존 계층에서 **OSD**를 이동하거나 제거할 수도 있습니다. **Ceph CLI** 사용량에는 다음과 같은 값이 있습니다.

id

설명

OSD의 숫자 **ID**입니다.

유형

정수

필수 항목

있음

예제

0

name

설명

OSD의 전체 이름입니다.

유형

문자열

필수 항목

있음

예제

osd.0

weight

설명

OSD의 CRUSH 가중치입니다.

유형

double

필수 항목

있음

예제

2.0

루트

설명

OSD가 상주하는 계층 또는 트리의 루트 버킷의 이름입니다.

유형

키-값 쌍.

필수 항목

있음

예제

root=default,root=replicated_rule 등

bucket-type

설명

이름이 버킷 유형이고 값은 버킷의 이름입니다. CRUSH 계층 구조에서 OSD의 CRUSH 위치를 지정할 수 있습니다.

유형

키-값 쌍입니다.

필수 항목

없음

예제

datacenter=dc1 room=room1 row=foo rack=bar host=foo-bar-1

2.3.1. CRUSH에서 OSD 보기

ceph osd crush tree 명령은 CRUSH 버킷과 항목을 트리 보기에 출력합니다. 이 명령을 사용하여 특정 버킷에서 OSD 목록을 확인합니다. ceph osd 트리 와 유사한 출력이 출력됩니다.

추가 세부 사항을 반환하려면 다음을 실행합니다.

```
# ceph osd crush tree -f json-pretty
```

이 명령은 다음과 유사한 출력을 반환합니다.

```
[
   {
      "id": -2,
      "name": "ssd",
      "type": "root",
      "type_id": 10,
      "items": [
         {
            "id": -6,
            "name": "dell-per630-11-ssd",
            "type": "host",
            "type_id": 1,
            "items": [
               {
                  "id": 6,
                  "name": "osd.6",
                  "type": "osd",
                  "type_id": 0,
                  "crush_weight": 0.099991,
                  "depth": 2
               }
            ]
         },
         {
            "id": -7,
            "name": "dell-per630-12-ssd",
            "type": "host",
            "type_id": 1,
            "items": [
               {
                  "id": 7,
                  "name": "osd.7",
                  "type": "osd",
                  "type_id": 0,
                  "crush_weight": 0.099991,
                  "depth": 2
               }
            ]
         },
         {
            "id": -8,
            "name": "dell-per630-13-ssd",
            "type": "host",
            "type_id": 1,
            "items": [
               {
                  "id": 8,
                  "name": "osd.8",
                  "type": "osd",
```

```
                "type_id": 0,
                "crush_weight": 0.099991,
                "depth": 2
            }
        ]
    }
]
},
{
    "id": -1,
    "name": "default",
    "type": "root",
    "type_id": 10,
    "items": [
        {
            "id": -3,
            "name": "dell-per630-11",
            "type": "host",
            "type_id": 1,
            "items": [
                {
                    "id": 0,
                    "name": "osd.0",
                    "type": "osd",
                    "type_id": 0,
                    "crush_weight": 0.449997,
                    "depth": 2
                },
                {
                    "id": 3,
                    "name": "osd.3",
                    "type": "osd",
                    "type_id": 0,
                    "crush_weight": 0.289993,
                    "depth": 2
                }
            ]
        },
        {
            "id": -4,
            "name": "dell-per630-12",
            "type": "host",
            "type_id": 1,
            "items": [
                {
                    "id": 1,
                    "name": "osd.1",
                    "type": "osd",
                    "type_id": 0,
                    "crush_weight": 0.449997,
                    "depth": 2
                },
                {
                    "id": 4,
                    "name": "osd.4",
                    "type": "osd",
```

```
                "type_id": 0,
                "crush_weight": 0.289993,
                "depth": 2
            }
        ]
    },
    {
        "id": -5,
        "name": "dell-per630-13",
        "type": "host",
        "type_id": 1,
        "items": [
            {
                "id": 2,
                "name": "osd.2",
                "type": "osd",
                "type_id": 0,
                "crush_weight": 0.449997,
                "depth": 2
            },
            {
                "id": 5,
                "name": "osd.5",
                "type": "osd",
                "type_id": 0,
                "crush_weight": 0.289993,
                "depth": 2
            }
        ]
    }
  ]
}
]
```


2.3.2. CRUSH에 OSD 추가

VirtIO 계층 구조에 Ceph OSD를 추가하는 것은 OSD를 시작하고, Ceph가 배치 그룹을 OSD 에할당하기 전에 마지막 단계입니다.

CRUSH 계층 구조에 추가하기 전에 Ceph OSD를 준비해야 합니다. Ceph Orchestrator와 같은 배포 유틸리티에서 이 단계를 수행할 수 있습니다. 예를 들어 단일 노드에서 Ceph OSD를 생성합니다.

```
ceph orch daemon add osd {host}:{device},[{device}]
```

CRUSH 계층 구조는 영향을 미치지 않으므로 ceph osd crush add 명령을 사용하면 원하는 위치에 CRUSH 계층 구조에 OSD를 추가할 수 있습니다. 지정한 위치는 실제 위치를 *반영해야 합니다*. 버킷을 하나 이상 지정하면 명령은 OSD를 지정한 가장 구체적인 버킷에 배치하고 해당 버킷을 지정한 다른 버킷 아래 이동합니다.

CRUSH 계층 구조에 OSD를 추가하려면 다음을 수행합니다.

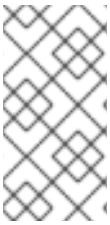
```
ceph osd crush add {id-or-name} {weight}  [{bucket-type}={bucket-name} ...]
```

중요

루트 버킷만 지정하면 명령은 **OSD**를 **root**에 직접 연결합니다. 그러나 **CRUSH** 규칙은 **OSD**가 호스트 내부 또는 섀시(섀시) 내에 있을 것으로 예상하며, 클러스터 토폴로지를 반영하는 다른 버킷의 내부 여야 합니다.

다음 예제에서는 **osd.0** 을 계층 구조에 추가합니다.

```
ceph osd crush add osd.0 1.0 root=default datacenter=dc1 room=room1 row=foo rack=bar host=foo-bar-1
```

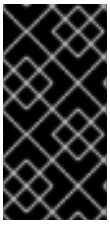


참고

ceph osd crush set 또는 **ceph osd crush create-or- rating**를 사용하여 **CRUSH** 계층 구조에 **OSD**를 추가할 수도 있습니다.

2.3.3. CRUSH 계층 구조 내에서 OSD 이동

스토리지 클러스터 토폴로지가 변경되면 **CRUSH** 계층에서 **OSD**를 이동하여 실제 위치를 반영할 수 있습니다.

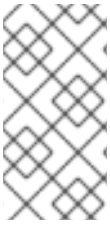


중요

CRUSH 계층에서 **OSD**를 이동하면 **Ceph**가 **OSD**에 할당되는 배치 그룹을 다시 계산하여 잠재적으로 데이터를 다시 배포할 수 있습니다.

CRUSH 계층 구조 내에서 OSD를 이동하려면 다음을 수행합니다.

```
ceph osd crush set {id-or-name} {weight} root={pool-name}  [{bucket-type}={bucket-name} ...]
```



참고

ceph osd crush create-or- 0.0/16을 사용하여 CRUSH 계층 구조 내에서 OSD를 이동할 수도 있습니다.

2.3.4. CRUSH 계층 구조에서 OSD 제거

CRUSH 계층에서 OSD를 제거하는 것이 클러스터에서 OSD를 제거하려는 첫 번째 단계입니다. CRUSH 맵에서 OSD를 제거하면 CRUSH에서 배치 그룹 및 데이터를 적절하게 재조정하는 OSD를 다시 계산합니다. 자세한 내용은 OSD 추가/제거를 참조하십시오.

실행 중인 클러스터의 CRUSH 맵에서 OSD를 제거하려면 다음을 실행합니다.

```
ceph osd crush remove {name}
```

2.4. 장치 클래스

Ceph의 CRUSH 맵은 데이터 배치를 제어하는 데 뛰어난 유연성을 제공합니다. 이는 Ceph의 가장 큰 강점 중 하나입니다. 이전의 Ceph 배포에서는 하드 디스크 드라이브를 거의 독점적으로 사용했습니다. 현재 Ceph 클러스터는 HDD, SSD, NVMe 또는 다양한 종류의 스토리지 장치로 자주 빌드됩니다. 예를 들어, Ceph Object Gateway 배포에서는 클라이언트가 느린 HDD 및 빠른 SSD에 데이터를 저장하기 위한 기타 스토리지 정책에 대한 스토리지 정책을 사용하는 것이 일반적입니다. Ceph Object Gateway 배포에는 버킷 인덱스에 대해 빠른 SSD에서 지원하는 풀이 있을 수 있습니다. 또한 OSD 노드에는 CRUSH 맵에 표시되지 않는 저널 또는 write-ahead 로그에만 사용되는 SSD도 자주 있습니다. 이러한 복잡한 하드웨어 시나리오는 이전에 CRUSH 맵을 수동으로 편집해야 했으며 시간이 오래 걸릴 수 있습니다. 스토리지 장치 클래스마다 다른 CRUSH 계층 구조가 있을 필요는 없습니다.

CRUSH 규칙은 CRUSH 계층 구조 측면에서 작동합니다. 그러나 다른 스토리지 장치 클래스가 동일한 호스트에 있으면 프로세스는 각 장치 클래스에 대해 여러 CRUSH 계층 구조를 만드는 사용자가 더 복잡해지며 CRUSH 계층 관리 작업을 대부분 자동화하는 시작 옵션에서 osd crush 업데이트를 비활성화합니다. 장치 클래스는 사용할 장치 클래스가 어떤 장치 클래스를 크게 단순화하여 CRUSH 관리 작업을 크게 단순화함으로써 이러한 번거 로움을 제거합니다.

참고

ceph osd tree 명령에는 장치 클래스를 반영하는 열이 있습니다.

다음 섹션에서는 장치 클래스 사용에 대해 자세히 설명합니다. 추가 예제 는 다른 장치 클래스 및 CRUSH 스토리지 전략 사용을 참조하십시오.

2.4.1. 장치 클래스 설정

OSD의 장치 클래스를 설정하려면 다음을 실행합니다.

```
# ceph osd crush set-device-class <class> <osdId> [<osdId>...]
```

예를 들어 다음과 같습니다.

```
# ceph osd crush set-device-class hdd osd.0 osd.1
# ceph osd crush set-device-class ssd osd.2 osd.3
# ceph osd crush set-device-class bucket-index osd.4
```

참고

Ceph에서 클래스에 자동으로 클래스를 할당할 수 있습니다. 그러나 클래스 이름은 단순히 임의의 문자열입니다. **hdd,ssd** 또는 **nvme** 를 준수할 필요는 없습니다. **Foregoing** 예에서 **bucket-index** 라는 장치 클래스는 **Ceph Object Gateway** 풀이 버킷 인덱스 워크로드를 독점적으로 사용한다는 **SSD** 장치를 나타낼 수 있습니다. 이미 설정된 장치 클래스를 변경하려면 **ceph osd crush rm-device-class** 를 먼저 사용합니다.

2.4.2. 장치 클래스 제거

OSD의 장치 클래스를 제거하려면 다음을 실행합니다.

```
# ceph osd crush rm-device-class <class> <osdId> [<osdId>...]
```

예를 들어 다음과 같습니다.

```
# ceph osd crush rm-device-class hdd osd.0 osd.1
# ceph osd crush rm-device-class ssd osd.2 osd.3
# ceph osd crush rm-device-class bucket-index osd.4
```

2.4.3. 장치 클래스 이름 변경

해당 클래스를 사용하는 모든 **OSD**에 대해 장치 클래스의 이름을 변경하려면 다음을 실행합니다.

```
# ceph osd crush class rename <oldName> <newName>
```

예를 들어 다음과 같습니다.

```
# ceph osd crush class rename hdd sas15k
```

2.4.4. 장치 클래스 나열

CRUSH 맵의 장치 클래스를 나열하려면 다음을 실행합니다.

```
# ceph osd crush class ls
```

출력은 다음과 같이 표시됩니다.

```
[
    "hdd",
    "ssd",
    "bucket-index"
]
```

2.4.5. 장치 클래스의 OSD 나열

특정 클래스에 속하는 모든 **OSD**를 나열하려면 다음을 실행합니다.

```
# ceph osd crush class ls-osd <class>
```

예를 들어 다음과 같습니다.

```
# ceph osd crush class ls-osd hdd
```

출력은 간단히 **OSD** 번호 목록입니다. 예를 들어 다음과 같습니다.

```
0
1
2
3
4
5
6
```

2.4.6. 클래스별 CRUSH 규칙 나열

동일한 클래스를 참조하는 모든 크러쉬 규칙을 나열하려면 다음을 실행합니다.

```
# ceph osd crush rule ls-by-class <class>
```

예를 들어 다음과 같습니다.

```
# ceph osd crush rule ls-by-class hdd
```

2.5. CRUSH 가중치

CRUSH 알고리즘은 새 데이터 오브젝트를 PG에 할당하고 PGs에 PGs 및 PGs를 OSD에 할당하는 쓰기 요청의 일관된 확률 배포를 목표로 OSD 장치당 테라바이트 단위로 가중치 값을 할당합니다. 이러한 이유로 동일한 유형과 크기의 장치를 사용하여 CRUSH 계층 구조를 만들고 동일한 가중치를 할당하는 것이 좋습니다. 성능 특성이 데이터 배포에 영향을 미치지 않는 경우에도 CRUSH 계층 구조의 일관된 성능 특성을 갖도록 동일한 I/O 및 처리량의 특성을 가진 장치를 사용하는 것이 좋습니다.

균일한 하드웨어를 사용하는 것이 항상 실용적인 것은 아니므로 다른 크기의 OSD 장치를 통합하고 상대 가중치를 사용하여 더 많은 데이터를 더 큰 장치에 배포하고 더 적은 데이터를 작은 장치에 배포할 수 있습니다.

2.5.1. Terabytes에서 OSD의 Weight 설정

CRUSH 맵에서 OSD CRUSH 가중치를 Terabytes로 설정하려면 다음 명령을 실행합니다.

```
ceph osd crush reweight {name} {weight}
```

다음과 같습니다.

name

설명

OSD의 전체 이름입니다.

유형

문자열

필수 항목

있음

예제

osd.0

weight

설명

OSD의 CRUSH 가중치입니다. 이 크기는 Terabytes의 OSD 크기입니다. 여기서 1.0 은 1 Terabyte입니다.

유형

double

필수 항목

있음

예제

2.0

이 설정은 OSD를 생성하거나 OSD를 추가한 직후 CRUSH 가중치를 조정할 때 사용됩니다. 일반적으로 OSD의 수명 동안 변경되지 않습니다.

2.5.2. Bucket의 OSD 가중치 설정

ceph osd crush reweight 을 사용하면 시간이 오래 걸릴 수 있습니다. 다음을 실행하여 버킷(row, rack, node 등)에서 모든 Ceph OSD 가중치를 설정(또는 재설정)할 수 있습니다.

```
osd crush reweight-subtree <name>
```

다음과 같습니다.

name 은 CRUSH 버킷의 이름입니다.

2.5.3. 가중치 에서 OSD 설정

에서 **ceph osd** 의 용도 및 **ceph osd out** 의 목적을 위해 **OSD**는 클러스터 내에 있거나 클러스터 외부에 있습니다. 이렇게 하면 모니터에서 **OSD** 상태를 기록합니다. 그러나 **OSD**가 클러스터에 있지만 문제를 해결할 때까지 사용하지 않으려는 상황이 발생할 수 있습니다(예: 스토리지 드라이브를 교체, 컨트롤러 변경 등).

다음을 실행하여 특정 **OSD**의 가중치(즉, **Terabytes**의 가중치를 변경하지 않고)의 가중치를 늘리거나 줄일 수 있습니다.

```
ceph osd reweight {id} {weight}
```

다음과 같습니다.

- **id** 는 **OSD** 번호입니다.
- 가중치 는 **0.0-1.0**의 범위입니다. 여기서 **0** 은 클러스터에 있지 않으며 (즉, **PG**가 할당되지 않음) **1.0**은 클러스터에 있습니다(즉, **OSD**는 다른 **OSD**와 동일한 개수의 **PG**를 받습니다).

2.5.4. 사용률을 통해 OSD의 가중치 설정

CRUSH는 새 데이터 오브젝트 **PG**와 **PG**를 **OSD**에 할당하는 쓰기 요청의 균일한 확률 배포를 대략적으로 처리하도록 설계되었습니다. 그러나 어떤 경우에도 클러스터의 불균형이 발생할 수 있습니다. 이는 여러 가지 이유로 발생할 수 있습니다. 예를 들어 다음과 같습니다.

- 다중 풀: **NetNamespace** 계층 구조에 여러 개의 풀을 할당할 수 있지만 풀에는 다른 수의 배치 그룹, 크기(스토리지할 복제본 수) 및 개체 크기 특성이 있을 수 있습니다.
- 사용자 지정 클라이언트: 블록 장치, 오브젝트 게이트웨이 및 파일 시스템과 같은 **Ceph** 클라이언트는 클라이언트의 데이터를 공유하고 균일한 크기의 **RADOS** 오브젝트로 클러스터의 오브젝트로 데이터를 스트라이프합니다. 따라서 진행 시나리오를 제외하고 **CRUSH**는 일반적으로 목표를 달성합니다. 그러나 클러스터가 개체 크기를 정상화하지 않고 데이터를 저장하는 데 **librados** 를 사용하여 데이터를 저장할 수 있는 또 다른 경우가 있습니다. 이 시나리오로 인해 클러스터의 불균형이 발생할 수 있습니다(예: **100MB** 오브젝트를 저장하고 **10**개의 **4MB** 개체를 저장하면 **OSD** 수가 다른 것보다 더 많은 데이터를 만들 수 있습니다.
- 확률: 균일한 배포로 인해 **PG**가 더 많아지고 일부는 더 적은 수의 **OSD**가 발생합니다. **OSD** 수가 많은 클러스터의 경우 통계값이 더 늘어날 것입니다.

다음을 실행하여 **OSD**를 다시 스케일링할 수 있습니다.

```
ceph osd reweight-by-utilization [threshold]  [weight_change_amount] [number_of_OSDs] [--no-increasing]
```

예를 들어 다음과 같습니다.

```
ceph osd test-reweight-by-utilization 110 .5 4 --no-increasing
```

다음과 같습니다.

- 임계값 은 데이터 스토리지 로드가 더 높은 **OSD**에서 더 낮은 가중치를 가지며 그에 더 적은 **PG**를 할당할 수 있도록 하는 사용률입니다. 기본값은 **120 %**를 반영하여 **120 %**입니다. **100+** 의 모든 값은 유효한 임계값입니다. 선택 사항:

- **weight_change_amount** 는 가중치를 변경할 양입니다. 유효한 값은 **0.0 - 1.0** 보다 큽니다. 기본값은 **0.05** 입니다. 선택 사항:

- **number_of_OSDs** 는 재사용 할 최대 **OSD** 수입니다. 대규모 클러스터의 경우 **OSD** 수를 다시 스케일링하도록 제한하면 상당한 재조정이 방지됩니다. 선택 사항:

- 기본적으로 **no-increasing** 는 해제 되어 있습니다. **reweight-by-utilization** 또는 **test-reweight-by-utilization** 명령을 사용하면 **osd** 가중치 를 늘릴 수 있습니다. 이 옵션을 이러한 명령과 함께 사용하면 **OSD**의 사용량이 낮은 경우에도 **OSD** 가중치가 늘어나지 않습니다. 선택 사항:

중요

reweight-by-utilization 을 실행하는 것이 권장되며 대규모 클러스터에는 다소 불가피합니다. 사용률은 시간이 지남에 따라 변경될 수 있으며 클러스터 크기 또는 하드웨어 변경에 따라 가중치를 업데이트하여 변경 사용률을 반영해야 할 수 있습니다. 사용률에 따라 재가중을 선택하는 경우 이 명령을 사용률, 하드웨어 또는 클러스터 크기 변경으로 다시 실행해야 할 수 있습니다.

가중치를 할당하는 이 또는 기타 가중치 명령을 실행하면 이 명령에서 할당한 가중치(예: **osd reweight-by-utilization,osd crush weight,osd weight,in** 또는 **out**)가 재정의됩니다.

2.5.5. PG Distribution에서 OSD의 가중치 설정

CRUSH 계층 구조에서 OSD 수가 적은 경우 일부 OSD에서 다른 OSD보다 더 많은 PG를 얻을 수 있으므로 부하가 높아집니다. PG 배포로 OSD를 다시 스케일링하여 다음을 실행하여 이 상황을 해결할 수 있습니다.

```
osd reweight-by-pg <poolname>
```

다음과 같습니다.

- Poolname 은 풀의 이름입니다. Ceph는 풀에서 OSD에 PG를 할당하고 이 풀의 PG 배포에 따라 OSD를 재사용하는 방법을 검토합니다. 동일한 CRUSH 계층 구조에 여러 개의 풀을 할당할 수 있습니다. 하나의 풀의 배포에 따라 OSD를 다시 스케일링하면 동일한 size(복제본 수) 및 PGs가 없는 경우 다른 풀에 대해 동일한 CRUSH 계층에 할당되는 의도하지 않은 효과가 있을 수 있습니다.

2.5.6. CRUSH 트리의 가중치 재계산

CRUSH 트리 버킷은 리프 가중치의 합계여야 합니다. CRUSH 맵 가중치를 수동으로 편집하는 경우 다음을 실행하여 CRUSH 버킷 트리가 버킷 아래의 리프 OSD의 합계를 정확하게 반영하는지 확인해야 합니다.

```
osd crush reweight-all
```

2.6. 기본 선호도

Ceph Client가 데이터를 읽거나 쓸 때 항상 작업 세트의 기본 OSD에 연결합니다. 설정하는 경우 [2, 3, 4], osd.2 가 기본 설정입니다. 다른 OSD와 비교하여 OSD가 주 역할을 하는 데 적합하지 않은 경우도 있습니다(예: 느린 디스크 또는 느린 컨트롤러가 있습니다. 하드웨어 활용도를 극대화하면서 성능 병목 현상(특히 읽기 작업 시)을 극대화하기 위해 CRUSH가 작동하는 세트의 기본 선호도로 OSD를 기본 선호도로 설정할 수 있습니다.

```
ceph osd primary-affinity <osd-id> <weight>
```

기본 선호도는 1 입니다(즉, OSD가 주 역할을 할 수 있음). OSD 기본 범위를 0-1 에서 설정할 수 있습니다. 여기서 0 은 OSD를 기본으로 사용할 수 없음을 나타내고 1은 OSD를 1 로 기본 범위로 사용할 수 있음을 의미합니다. weight가 & lt; 1 이면 CRUSH가 Ceph OSD 데몬을 1차로 선택할 가능성이 적습니다.

2.7. CRUSH 규칙

CRUSH 규칙은 Ceph 클라이언트가 오브젝트를 저장하기 위해 버킷과 기본 OSD를 선택하는 방법과 기본 OSD가 버킷 및 보조 OSD를 선택하는 방법과 복제본 또는 코딩 청크를 저장하는 방법을 정의합니다. 예를 들어 두 개의 오브젝트 복제본에 대해 SSD에서 지원하는 대상 OSD 쌍을 선택하는 규칙과 세 개의 복제본에서 다른 데이터 센터의 SAS 드라이브에서 지원하는 세 개의 대상 OSD를 선택하는 또 다른 규칙을 만들 수 있습니다.

규칙은 다음 형식을 사용합니다.

```
rule <rulename> {

    id <unique number>
    type [replicated | erasure]
    min_size <min-size>
    max_size <max-size>
    step take <bucket-type> [class <class-name>]
    step [choose|chooseleaf] [firstn|indep] <N> <bucket-type>
    step emit
}
```

id

설명

규칙을 식별하기 위한 고유한 전체 번호입니다.

목적

규칙 마스크의 구성 요소입니다.

유형

정수

필수 항목

있음

기본값

0

type

설명

코딩된 스토리지 드라이브 복제 또는 삭제 규칙에 대해 설명합니다.

목적

규칙 마스크의 구성 요소입니다.

유형

문자열

필수 항목

있음

기본값

replicated

유효한 값

현재 복제만

min_size

설명

*풀이 이 숫자보다 복제본을 더 적게 만드는 경우 **CRUSH**는 이 규칙을 선택하지 않습니다.*

유형

정수

목적

규칙 마스크의 구성 요소입니다.

필수 항목

있음

기본값

1

max_size

설명

*풀이 이 번호보다 복제본을 더 많이 만드는 경우 **CRUSH**는 이 규칙을 선택하지 않습니다.*

유형

정수

목적

규칙 마스크의 구성 요소입니다.

필수 항목

있음

기본값

10

Step take <bucket-name> [class <class-name>]

설명

버킷 이름을 가져와서 트리 반복을 시작합니다.

목적

규칙의 구성 요소입니다.

필수 항목

있음

예제

*데이터 단계를 수행하는 단계는 데이터클래스 **ssd**를 가져옵니다.*

step select firstn <num> type <bucket-type>

설명

지정된 유형의 버킷 수를 선택합니다. 수는 일반적으로 풀의 복제본 수(즉, 풀 크기)입니다.

- ***&lt ;num> == 0** 인 경우 **pool-num-replicas** 버킷(사용 가능한 모든)을 선택합니다.*
- ***< num> > 0 && < pool-num-replicas** 이면 여러 버킷을 선택합니다.*

- *& lt;num> < 0 이면 pool-num-replicas - {num} 을 의미합니다.*

목적

규칙의 구성 요소입니다.

사전 요구 사항

단계를 따르거나 단계를 선택합니다.

예제

Step select firstn 1 type row

Step selectleaf firstn <num> type <bucket-type>

설명

***{bucket-type}** 의 버킷을 선택하고 버킷 세트에 있는 각 버킷의 하위 트리에서 리프 노드를 선택합니다. 집합의 버킷 수는 일반적으로 풀의 복제본 수(즉, 풀 크기)입니다.*

- *&lt ;num> == 0 인 경우 pool-num-replicas 버킷(사용 가능한 모든)을 선택합니다.*
- *< num> > 0 && < pool-num-replicas 이면 여러 버킷을 선택합니다.*
- *&lt ;num> < 0 이면 pool-num-replicas - <num> 을 의미합니다.*

목적

규칙의 구성 요소입니다. 사용량은 두 단계를 사용하여 장치를 선택할 필요가 없습니다.

사전 요구 사항

다음 단계는 단계 또는 단계를 선택합니다.

예제

Step selectleaf firstn 0 type row

Step emit

설명

현재 값을 출력하고 스택을 의미합니다. 일반적으로 규칙의 끝에서 사용되지만 동일한 규칙의 다른 트리에서 선택하는 데 사용될 수도 있습니다.

목적

규칙의 구성 요소입니다.

사전 요구 사항

다음 단계는 을 선택합니다.

예제

Step emit

***FirstN vdep** 비교*

설명

***CRUSH** 맵에서 **OSD**가 다운된 경우 **CRUSH**가 사용하는 대체 전략을 제어합니다. 이 규칙을 복제 풀과 함께 사용해야 하는 경우 먼저 수행해야 하며 소거 코드된 풀의 경우 이 규칙을 사용하지 않아야 합니다.*

예제

***PG**가 **OSD 1, 2, 3, 4, 5**에 저장되어 있습니다. 첫 번째 시나리오에서 **CRUSH**는 첫 번째 모드를 사용하여 **1** 및 **2**를 선택한 다음 **3**을 선택하지만 **4** 및 **5**를 다시 시도하여 **4** 및 **5**를 선택한 다음 계속 진행합니다. 마지막 **CRUSH** 매핑 변경은 **1, 2, 3, 4, 5** 에서 **1, 2, 4, 5, 6** 입니다. 두 번째 시나리오에서는 삭제 코드된 풀의 **dep** 모드를 사용하여, **CRUSH**가 실패한 **OSD 3**을 다시 선택하고, **6**을 선택하여 **1, 2, 3, 4, 5** 에서 **1, 2, 6, 4, 5** 로부터 최종 변환을 시도합니다.*

중요

*지정된 **CRUSH** 규칙을 여러 풀에 할당할 수 있지만 단일 풀에 여러 **CRUSH** 규칙이 있을 수 없습니다.*

2.7.1. 규칙 나열

*명령줄에서 **CRUSH** 규칙을 나열하려면 다음을 실행합니다.*

```
ceph osd crush rule list
ceph osd crush rule ls
```

2.7.2. 규칙 덤프

특정 **CRUSH** 규칙의 내용을 덤프하려면 다음을 실행합니다.

```
ceph osd crush rule dump {name}
```

2.7.3. 간단한 규칙 추가

CRUSH 규칙을 추가하려면 사용하려는 계층 구조의 루트 노드, 복제할 버킷 유형(예: **'rack'**, **'row'** 등) 및 버킷 선택 모드를 지정해야 합니다.

```
ceph osd crush rule create-simple {rulename} {root} {bucket-type} {firstn|indep}
```

Ceph는 지정한 유형의 **chooseleaf** 및 **1**개의 버킷으로 규칙을 만듭니다.

예를 들어 다음과 같습니다.

```
ceph osd crush rule create-simple deleteme default host firstn
```

다음 규칙을 생성합니다.

```
{ "id": 1,
  "rule_name": "deleteme",
  "type": 1,
  "min_size": 1,
  "max_size": 10,
  "steps": [
        { "op": "take",
          "item": -1,
          "item_name": "default"},
        { "op": "chooseleaf_firstn",
          "num": 0,
          "type": "host"},
        { "op": "emit"}]}
```

2.7.4. 복제된 규칙 추가

복제된 풀에 대한 **CRUSH** 규칙을 만들려면 다음을 실행합니다.

```
# ceph osd crush rule create-replicated <name> <root> <failure-domain> <class>
```

다음과 같습니다.

- **&lt ;name >:** 규칙의 이름입니다.
- **< root >: CRUSH** 계층 구조의 루트입니다.
- **<failure-domain > :** 실패 도메인 예**: host** 또는 **rack.**
- **< class >:** 스토리지 장치 클래스입니다. 예**: hdd** 또는 **ssd.**

예를 들어 다음과 같습니다.

```
# ceph osd crush rule create-replicated fast default host ssd
```

2.7.5. 평가 코드 규칙 추가

삭제 코딩된 풀과 함께 사용할 **NetNamespace** 규칙을 추가하려면 규칙 이름과 삭제 코드 프로필을 지정할 수 있습니다.

```
ceph osd crush rule create-erasure {rulename} {profilename}
```

2.7.6. 규칙 제거

규칙을 제거하려면 다음을 실행하고 **CRUSH** 규칙 이름을 지정합니다.

```
ceph osd crush rule rm {name}
```

2.8. CRUSH TUNABLES

Ceph 프로젝트는 많은 변경 사항과 많은 새로운 기능으로 기하 급수적으로 증가했습니다. **Ceph**의 첫 번째 상용 지원 주요 릴리스인 **v0.48(Argonaut)**부터 **Ceph**는 **CRUSH** 알고리즘의 특정 매개 변수를 조정하는 기능을 제공합니다. 즉, 설정이 소스 코드에서 고정되지 않습니다.

고려해야 할 몇 가지 중요한 사항:

- **NetNamespace** 값을 조정하면 스토리지 노드 간에 일부 **PG**가 변경될 수 있습니다. **Ceph** 클러스터에서 이미 많은 데이터를 저장하고 있는 경우 이동할 데이터의 일부 용량에 대비해야 합니다.

- **ceph-osd** 및 **ceph-mon** 데몬은 업데이트된 맵을 수신하는 즉시 새 연결의 기능 비트를 요구하기 시작합니다. 그러나 이미 연결된 클라이언트는 효과적으로 에 배치되어 있으며 새로운 기능을 지원하지 않는 경우 잘못된 것입니다. **Ceph** 클라이언트도 업데이트하는 **Ceph Storage** 클러스터 데몬을 업그레이드할 때 확인하십시오.

- **CRUSH** 튜닝 가능 항목이 **non-legacy** 값으로 설정되어 나중에 레거시 값으로 다시 변경되면 기능을 지원하는 데 **ceph-osd** 데몬이 필요하지 않습니다. 그러나 **OSD** 피어링 프로세스에서는 이전 맵을 검사하고 이해해야 합니다. 따라서 클러스터가 이전에 비-레거시 **CRUSH** 값을 사용한 경우 기존 기본값을 사용하여 최신 버전의 맵을 다시 전환한 경우에도 이전 버전의 **ceph-osd** 데몬을 실행하지 않아야 합니다.

2.8.1. CRUSH 튜닝

CRUSH를 튜닝하기 전에 모든 **Ceph** 클라이언트 및 모든 **Ceph** 데몬이 동일한 버전을 사용하는지 확인해야 합니다. 최근에 업그레이드한 경우 데몬을 다시 시작하고 클라이언트를 다시 연결했는지 확인하십시오.

CRUSH 튜닝 가능 항목을 조정하는 가장 간단한 방법은 알려진 프로필로 변경하는 것입니다. 이는 다음과 같습니다.

- **legacy: v0.47(pre-Argonaut)** 이전의 레거시 동작입니다.

- **Argonau t: v0.48(Argonaut)** 릴리스에서 지원하는 레거시 값입니다.

- **Bobtail: v0.56 (Bobtail)** 릴리스에서 지원하는 값입니다.

- **Firefly: v0.80 (Firefly)** 릴리스에서 지원하는 값입니다.

- ***Hammer: v0.94 (Hammer)** 릴리스에서 지원하는 값입니다.*
- ***jewel: v10.0.2 (Jewel)** 릴리스에서 지원하는 값입니다.*
- ***optimal:** 현재 최상의 값입니다.*
- ***default:** 새 클러스터의 현재 기본값입니다.*

명령을 사용하여 실행 중인 클러스터에서 프로필을 선택할 수 있습니다.

```
# ceph osd crush tunables <profile>
```

참고

이로 인해 일부 데이터 이동이 발생할 수 있습니다.

*일반적으로 업그레이드 후 **CRUSH** 튜닝 가능 항목을 설정하거나 경고가 표시되면 설정해야 합니다. 버전 **v0.74**부터 **CRUSH** 튜닝 가능 항목이 최적의 값으로 설정되지 않은 경우 최적 값이 **v0.73**의 기본값으로 설정된 경우 **Ceph**에서 상태 경고를 발행합니다. 이 경고를 제거 하려면 다음 두 가지 옵션이 있습니다.*

1. *기존 클러스터에서 튜닝 가능 항목을 조정합니다. 이 경우 일부 데이터 이동(약 **10 %** 정도)이 발생합니다. 이는 기본 경로이지만 데이터 이동이 성능에 영향을 줄 수 있는 프로덕션 클러스터를 고려해야 합니다. 다음을 사용하여 최적의 튜닝 가능 항목을 활성화할 수 있습니다.*

   ```
   # ceph osd crush tunables optimal
   ```

   *작업이 제대로 작동하지 않고 너무 많은 진행 상황을 발생했거나 클라이언트 호환성 문제 (이전 커널 **cephfs** 또는 **rbd** 클라이언트 또는 **pre-bobtail librados** 클라이언트)가 있는 경우 이전 프로필로 다시 전환할 수 있습니다.*

   ```
   # ceph osd crush tunables <profile>
   ```

   *예를 들어, **pre-v0.48 (Argonaut)** 값을 복원하려면 다음을 실행합니다.*

```
# ceph osd crush tunables legacy
```

2. *ceph.conf 파일의 [mon] 섹션에 다음 옵션을 추가하여 CRUSH를 변경하지 않고 경고를 제거할 수 있습니다.*

```
mon warn on legacy crush tunables = false
```

변경 사항을 적용하려면 모니터를 다시 시작하거나 다음을 사용하여 모니터를 실행하는 데 옵션을 적용합니다.

```
# ceph tell mon.\* injectargs --no-mon-warn-on-legacy-crush-tunables
```

2.8.2. CRUSH 튜닝, 어려운 방법

모든 클라이언트가 최근 코드를 실행 중인지 확인할 수 있는 경우 CRUSH 맵을 추출하고 값을 수정한 다음 클러스터에 다시 삽입하여 튜닝 가능 항목을 조정할 수 있습니다.

- *최신 CRUSH 맵을 추출합니다.*

```
ceph osd getcrushmap -o /tmp/crush
```

- *튜닝 가능 항목을 조정합니다. 이러한 값은 테스트한 대규모 및 소규모 클러스터 모두에 대해 최상의 동작을 제공하는 것으로 나타납니다. 이 작업이 작동하려면 --enable-unsafe-tunables 인수를 crushtool 에 추가로 지정해야 합니다. 극단적인 주의로 이 옵션을 사용하십시오.*

```
crushtool -i /tmp/crush --set-choose-local-tries 0 --set-choose-local-fallback-tries 0 --set-choose-total-tries 50 -o /tmp/crush.new
```

- *수정된 맵 삽입:*

```
ceph osd setcrushmap -i /tmp/crush.new
```

2.8.3. 레거시 값

참조를 위해 CRUSH 튜닝 가능 항목의 레거시 값은 다음을 사용하여 설정할 수 있습니다.

```
crushtool -i /tmp/crush --set-choose-local-tries 2 --set-choose-local-fallback-tries 5 --set-choose-total-tries 19 --set-chooseleaf-descend-once 0 --set-chooseleaf-vary-r 0 -o /tmp/crush.legacy
```

다시 한번 특수 **--enable-unsafe-tunables** 옵션이 필요합니다. 또한 기능 비트가 완벽하게 적용되지 않으므로 레거시 값으로 되돌아간 후 이전 버전의 **ceph-osd** 데몬을 실행해야 합니다.

2.9. CRUSH 맵 편집

일반적으로 **Ceph CLI**를 사용하여 런타임 시 **CRUSH** 맵을 수정하는 것이 **CRUSH** 맵을 수동으로 편집하는 것보다 편리합니다. 그러나 기본 버킷 유형 변경 또는 **straw2** 이외의 버킷 알고리즘을 사용하는 등 편집할 수 있는 경우가 있습니다.

기존 **CRUSH** 맵을 편집하려면 다음을 수행합니다.

1. **CRUSH** 맵을 가져옵니다.
2. **CRUSH** 맵을 분리 합니다.
3. 장치 중 하나와 **Buckets** 및 규칙을 편집합니다.
4. **CRUSH** 맵을 다시 컴파일 합니다.
5. **CRUSH** 맵을 설정합니다.

특정 풀에 대한 **CRUSH** 맵 규칙을 활성화하려면 공통 규칙 번호를 식별하고 풀을 생성할 때 풀의 규칙 번호를 지정합니다.

2.9.1. CRUSH 맵 가져오기

클러스터의 **CRUSH** 맵을 가져오려면 다음을 실행합니다.

```
ceph osd getcrushmap -o {compiled-crushmap-filename}
```

Ceph는 컴파일된 **CRUSH** 맵을 지정한 파일 이름에 출력합니다. **CRUSH** 맵은 컴파일된 형식이므로 편집하기 전에 먼저 컴파일해야 합니다.

2.9.2. CRUSH 맵 분리

CRUSH 맵을 분리하려면 다음을 실행합니다.

```
crushtool -d {compiled-crushmap-filename} -o {decompiled-crushmap-filename}
```

Ceph는 컴파일된 **CRUSH** 맵과 출력(**-o**)을 지정한 파일 이름으로 컴파일(**-d**)합니다.

2.9.3. CRUSH 맵 컴파일

CRUSH 맵을 컴파일하려면 다음을 실행합니다.

```
crushtool -c {decompiled-crush-map-filename} -o {compiled-crush-map-filename}
```

Ceph는 컴파일된 **CRUSH** 맵을 지정한 파일 이름에 저장합니다.

2.9.4. CRUSH 맵 설정

클러스터의 **CRUSH** 맵을 설정하려면 다음을 실행합니다.

```
ceph osd setcrushmap -i  {compiled-crushmap-filename}
```

Ceph는 클러스터의 **CRUSH** 맵으로 지정한 파일 이름의 컴파일된 **CRUSH** 맵을 입력합니다.

2.10. CRUSH 스토리지 전략의 예

대부분의 풀은 기본적으로 대규모 하드 드라이브에서 지원하는 **OSD**에 연결하되 일부 풀은 **SSD**(솔리드 스테이트 드라이브)에서 지원하는 **OSD**에 매핑되어 있어야 합니다. **CRUSH**는 이러한 시나리오를 쉽게 처리할 수 있습니다.

장치 클래스를 사용합니다. 프로세스는 각 장치에 클래스를 추가하는 것은 간단합니다. 예를 들어 다음과 같습니다.

```
# ceph osd crush set-device-class <class> <osdId> [<osdId>]
# ceph osd crush set-device-class hdd osd.0 osd.1 osd.4 osd.5
# ceph osd crush set-device-class ssd osd.2 osd.3 osd.6 osd.7
```

그런 다음 장치를 사용할 규칙을 만듭니다.

```
# ceph osd crush rule create-replicated <rule-name> <root> <failure-domain-type> <device-class>:
# ceph osd crush rule create-replicated cold default host hdd
# ceph osd crush rule create-replicated hot default host ssd
```

마지막으로 규칙을 사용하도록 풀을 설정합니다.

```
ceph osd pool set <poolname> crush_rule <rule-name>
ceph osd pool set cold crush_rule hdd
ceph osd pool set hot crush_rule ssd
```

*하나의 계층 구조가 여러 장치 클래스를 처리할 수 있으므로 **CRUSH** 맵을 수동으로 편집할 필요가 없습니다.*

```
device 0 osd.0 class hdd
device 1 osd.1 class hdd
device 2 osd.2 class ssd
device 3 osd.3 class ssd
device 4 osd.4 class hdd
device 5 osd.5 class hdd
device 6 osd.6 class ssd
device 7 osd.7 class ssd

  host ceph-osd-server-1 {
      id -1
      alg straw2
      hash 0
      item osd.0 weight 1.00
      item osd.1 weight 1.00
      item osd.2 weight 1.00
      item osd.3 weight 1.00
  }

  host ceph-osd-server-2 {
      id -2
      alg straw2
      hash 0
      item osd.4 weight 1.00
      item osd.5 weight 1.00
      item osd.6 weight 1.00
      item osd.7 weight 1.00
  }

  root default {
      id -3
      alg straw2
      hash 0
      item ceph-osd-server-1 weight 4.00
```

```
    item ceph-osd-server-2 weight 4.00
}


rule cold {
    ruleset 0
    type replicated
    min_size 2
    max_size 11
    step take default class hdd
    step chooseleaf firstn 0 type host
    step emit
}


rule hot {
    ruleset 1
    type replicated
    min_size 2
    max_size 11
    step take default class ssd
    step chooseleaf firstn 0 type host
    step emit
}
```

3장. PG(배치 그룹)

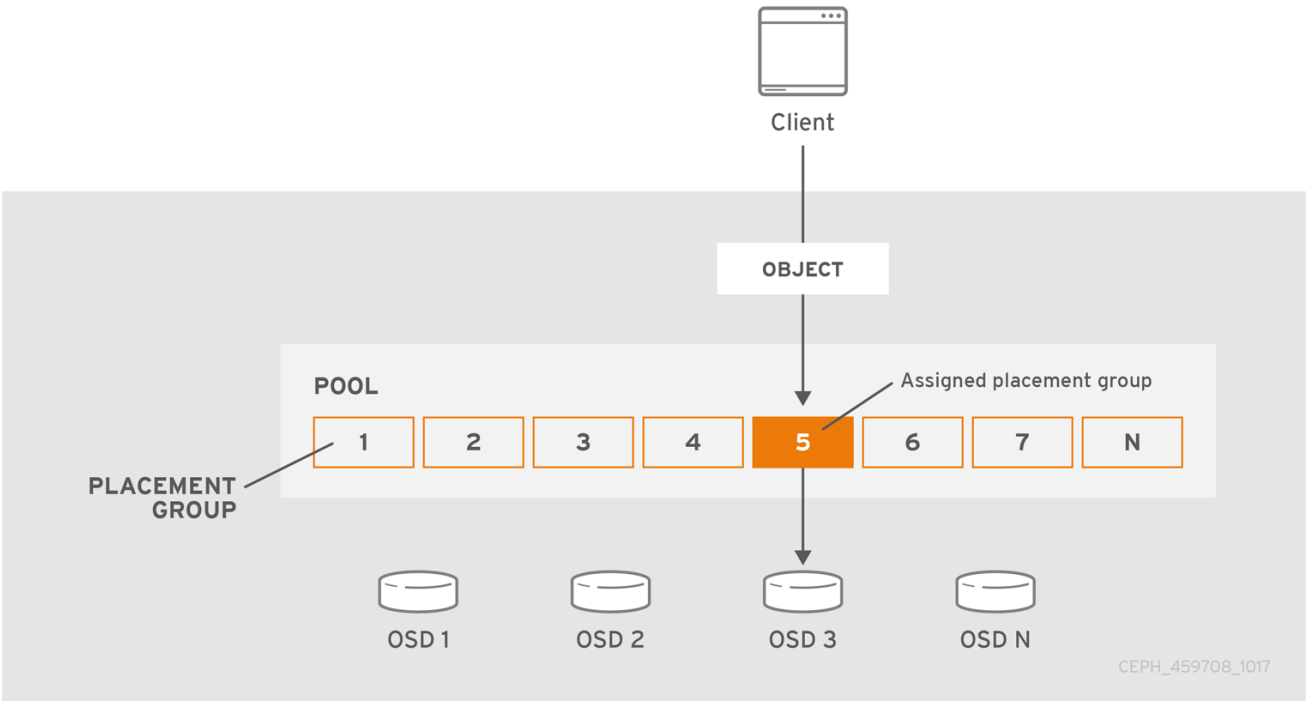
PG(배치 그룹)는 Ceph 클라이언트에 보이지 않지만 Ceph Storage 클러스터에서 중요한 역할을 수행합니다.

Ceph Storage 클러스터에는 엑사바이트 수준의 스토리지 용량에 도달하기 위해 수천 개의 OSD가 필요할 수 있습니다. Ceph 클라이언트는 전체 클러스터의 논리적 하위 집합인 풀에 오브젝트를 저장합니다. 풀에 저장된 오브젝트 수를 수백만 이상으로 쉽게 실행할 수 있습니다. 수백만 개의 개체가 있는 시스템은 개체별 배치를 현실적으로 추적할 수 없으며 여전히 제대로 작동합니다. Ceph는 개체를 배치 그룹에 할당하고 배치 그룹을 OSD에 할당하여 동적이고 효율적으로 재조정할 수 있도록 합니다.

	컴퓨터 과학의 모든 문제는 너무 많은 간접성을 제외하고는 다른 수준의 간접화로 해결할 수 있습니다.	
		-- David Wheeler

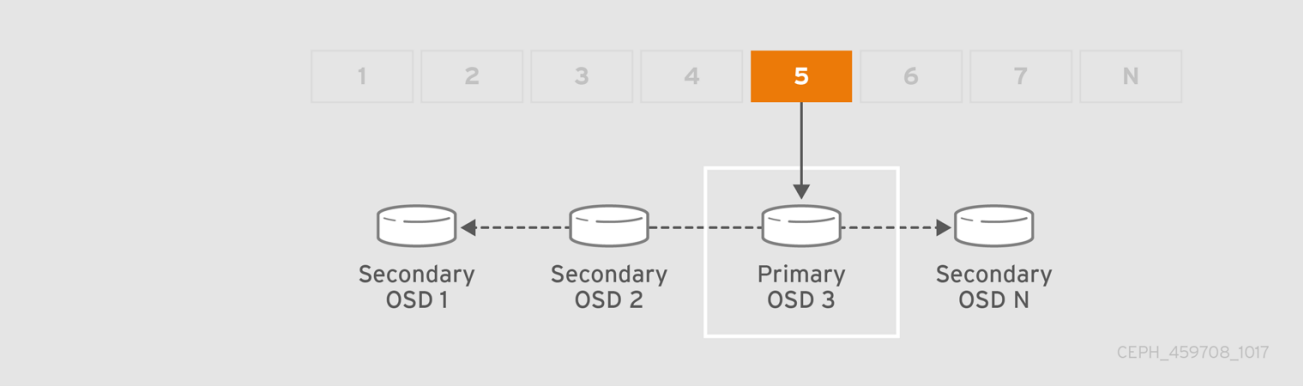
3.1. 배치 그룹 정보

풀 내에서 오브젝트별 개체 배치를 추적하는 것은 규모에 따라 컴퓨팅적으로 비용이 많이 듭니다. Ceph는 대규모의 고성능을 배치 그룹으로 세분화하고, 각 개별 오브젝트를 배치 그룹에 할당하고, 배치 그룹을 기본 OSD에 할당합니다. OSD가 실패하거나 클러스터의 재조정이 발생하면, Ceph는 각 오브젝트를 개별적으로 처리할 필요 없이 배치 그룹의 모든 오브젝트를 이동하거나 복제할 수 있습니다. 이를 통해 Ceph 클러스터는 효율적으로 재조정하거나 복구할 수 있습니다.



CRUSH가 OSD에 배치 그룹을 할당하면 일련의 OSD를 1차로 계산합니다. osd_pool_default_size 설정 - 복제된 풀의 경우 1 을, 삭제 코드화된 풀의 코딩 청크 수에 따라 영구적으로 데이터를 손실하지 않고

실패할 수 있는 배치 그룹을 저장하는 OSD 수가 결정됩니다. 기본 OSD는 CRUSH를 사용하여 보조 OSD를 식별하고 배치 그룹의 콘텐츠를 보조 OSD에 복사합니다. 예를 들어 CRUSH가 개체를 배치 그룹에 할당하고 배치 그룹이 OSD 5에 해당 OSD 1을 계산하면 OSD 1과 OSD 8이 배치 그룹의 보조 OSD이면 기본 OSD 5가 데이터를 OSD 1과 8에 복사합니다. Ceph는 클라이언트를 대신하여 데이터를 복사함으로써 클라이언트 인터페이스를 단순화하고 클라이언트 워크로드를 줄입니다. 동일한 프로세스를 사용하면 Ceph 클러스터가 동적으로 복구 및 재조정할 수 있습니다.



기본 OSD가 실패하고 클러스터에서 표시되지 않으면 CRUSH는 배치 그룹을 배치 그룹의 개체 복사본을 수신하는 다른 OSD에 할당합니다. Up Set 의 또 다른 OSD는 기본 OSD의 역할을 가정합니다.

개체 복제본 수를 늘리거나 청크를 코딩하면 CRUSH에서 필요에 따라 각 배치 그룹을 추가 OSD에 할당합니다.

참고

PGS는 OSD를 소유하지 않습니다. CRUSH는 각 OSD에 임의적으로 여러 배치 그룹을 할당하여 데이터가 클러스터에 균등하게 배포되도록 합니다.

3.2. 배치 그룹 상태

ceph -s 또는 ceph -w 명령을 사용하여 스토리지 클러스터 상태를 확인하는 경우 Ceph는 배치 그룹(PG) 상태를 보고합니다. PG에는 하나 이상의 상태가 있습니다. PG 맵의 PGs의 optimal state는 active + clean 상태입니다.

활성화

PG가 피어링되어 있지만 아직 활성화되지 않았습니다.

활성 상태

Ceph가 PG에 요청을 처리합니다.

backfill_toofull

대상 OSD가 backfillfull 비율 위에 있기 때문에 backfill 작업이 대기 중입니다.

backfill_unfound

unfound 오브젝트로 인해 backfill이 중지되었습니다.

backfill_wait

PG가 backfill을 시작할 때까지 대기 중입니다.

backfilling

Ceph는 최근 작업의 로그와 동기화해야 하는 콘텐츠를 유추하는 대신 PG의 전체 콘텐츠를 스캔하고 동기화하고 있습니다. backfill은 복구의 특별한 경우입니다.

clean

Ceph는 PG에 있는 모든 오브젝트를 정확하게 복제했습니다.

생성

Ceph는 여전히 PG를 생성하고 있습니다.

딥

Ceph는 저장된 체크섬에 대해 PG 데이터를 확인하고 있습니다.

Degraded

Ceph는 PG에 일부 오브젝트를 정확하게 복제하지 않았습니다.

down

필요한 데이터가 있는 복제본이 다운되어 PG가 오프라인 상태가 됩니다. min_size 복제본보다 작은 PG가 down으로 표시됩니다. ceph 상태 세부 정보를 사용하여 백업 OSD 상태를 확인합니다.

forced_backfill

해당 PG의 높은 백필 우선순위는 사용자에 의해 시행됩니다.

forced_recovery

PG의 높은 복구 우선 순위를 사용자가 적용합니다.

incomplete

Ceph는 PG가 발생할 수 있는 쓰기에 대한 정보가 없거나 정상적인 복사본이 없는 것을 감지합니

다. 이 상태가 표시되면 필요한 정보가 포함될 수 있는 실패한 OSD를 시작합니다. 삭제 코딩된 풀의 경우 min_size를 일시적으로 줄일 수 있습니다.In the case of an objectsure coded pool, temporarily reducing min_size might allow recovery.

inconsistent

Ceph는 PG에서 오브젝트의 하나 이상의 복제본에서 불일치를 탐지합니다(예: 오브젝트가 잘못된 크기, 복구 완료 후 하나의 복제본에서 오브젝트가 누락됨).

peering

PG가 피어링 프로세스를 진행하고 있습니다. 피어링 프로세스는 많은 지연 없이 제거되어야 하지만, 이 프로세스가 남아 있고, 피어링 상태의 PG 수가 감소하지 않으면 피어링이 정지될 수 있습니다.

peered

PG는 피어링했지만 풀의 구성된 min_size 매개변수에 도달할 수 있는 충분한 복사본이 없기 때문에 클라이언트 IO를 제공할 수 없습니다. 복구가 이 상태에서 발생할 수 있으므로 PG는 결국 min_size 까지 복구될 수 있습니다.

복구

Ceph는 오브젝트와 해당 복제본을 마이그레이션하거나 동기화하고 있습니다.

recovery_toofull

대상 OSD가 전체 비율을 초과하므로 복구 작업이 대기 중입니다.

recovery_unfound

Unfound 개체로 인해 복구가 중지되었습니다.

recovery_wait

PG가 복구를 시작하기 위해 대기 중입니다.

Remapped

PG는 ArgoCD가 지정된 것과 다른 OSD 세트에 임시로 매핑됩니다.

복구

Ceph는 PG를 확인하고 가능한 경우 이를 발견한 불일치를 복구하고 있습니다.

replay

PG는 OSD가 충돌한 후 클라이언트가 작업을 재생하도록 대기 중입니다.

snaptrim

트리밍 스냅입니다.

snaptrim_error

오류 발생을 멈춥니다.

snaptrim_wait

트리밍에 대기했습니다.

scrubbing

Ceph에서 PG 메타데이터가 불일치를 확인 중입니다.

splitting

Ceph는 PG를 여러 PG로 분할하고 있습니다.

오래된

PG는 알려지지 않은 상태에 있습니다. PG 매핑이 변경되었기 때문에 모니터에 대한 업데이트를 받지 못했습니다.

위sized

PG에는 구성된 풀 복제 수준보다 적은 수의 사본이 있습니다.

알 수 없음

ceph-mgr 은 Ceph Manager가 시작된 이후 OSD에서 PG 상태에 대한 정보를 받지 못했습니다.

추가 리소스

- 자세한 내용은 Ceph 클러스터에서 가능한 배치 그룹 상태 정보를 참조하십시오.

3.3. 배치 그룹 상표

모든 OSD에서 데이터 내구성 및 데이터 배포는 더 많은 배치 그룹을 호출하지만, 성능을 극대화하여 CPU 및 메모리 리소스를 보존하는 데 필요한 최소 용량으로 줄여야 합니다.

3.3.1. 데이터 내결함성

Ceph는 데이터의 영구 손실을 방지하기 위해 노력합니다. 그러나 OSD에 오류가 발생하면 데이터가

완전히 복구될 때까지 영구 데이터 손실 위험이 증가합니다. 영구 데이터 손실은 드문 경우지만 여전히 가능합니다. 다음 시나리오에서는 Ceph가 데이터의 복사본을 세 개 포함하는 단일 배치 그룹의 데이터를 영구적으로 잃을 수 있는 방법을 설명합니다.

- *OSD가 실패하고 포함된 오브젝트의 모든 복사본이 손실됩니다. OSD에 저장된 배치 그룹 내의 모든 오브젝트의 경우 복제본 수는 갑자기 3에서 2로 떨어집니다.*
- *Ceph는 각 배치 그룹에 대한 모든 오브젝트의 세 번째 복사본을 다시 만들 새 OSD를 선택하여 실패한 OSD에 저장된 각 배치 그룹의 복구를 시작합니다.*
- *새 OSD가 세 번째 사본으로 완전히 채워지기 전에 동일한 배치 그룹의 복사본이 포함된 두 번째 OSD가 실패합니다. 일부 객체는 하나의 생존 사본 만 가질 수 있습니다.*
- *Ceph는 또 다른 OSD를 선택하고 오브젝트를 복사하여 원하는 개수의 복사본을 복원합니다.*
- *복구가 완료되기 전에 동일한 배치 그룹의 사본이 포함된 세 번째 OSD가 실패합니다. 이 OSD에 남은 오브젝트 사본만 포함되어 있으면 오브젝트가 영구적으로 손실됩니다.*

하드웨어 실패는 예외가 아니라 기대입니다. 예정된 시나리오를 방지하기 위해 이상적으로 복구 프로세스는 합리적으로 가능한 한 빨리 수행되어야합니다. 클러스터 크기, 하드웨어 구성 및 배치 그룹 수는 총 복구 시간에 중요한 역할을 합니다.

소규모 클러스터는 신속하게 복구할 수 없습니다.

*3개의 복제본 풀에서 512개의 배치 그룹이 있는 OSD 10개가 포함된 클러스터에서 CRUSH는 각 배치 그룹 3개의 OSD를 제공합니다. 각 OSD는 호스팅 종료 (512 * 3) / 10 = ~150 배치 그룹. 첫 번째 OSD가 실패하면 클러스터가 모든 150 배치 그룹에 대해 동시에 복구를 시작합니다.*

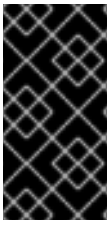
Ceph가 나머지 150개 배치 그룹을 나머지 9개 OSD에 무작위로 저장했을 가능성이 높습니다. 따라서 남아 있는 각 OSD는 다른 모든 OSD에 오브젝트 복사본을 보내고, 나머지 OSD는 현재 할당된 150개 배치 그룹을 담당하므로 일부 새 오브젝트도 수신합니다.

총 복구 시간은 풀을 지원하는 하드웨어에 따라 다릅니다. 예를 들어 10개의 OSD 클러스터에서 호스트에 1TB SSD가 있는 하나의 OSD가 있고 10GB/s 스위치가 각각 10개의 호스트를 연결하는 경우 복구 시간은 M 분 정도 걸립니다. 반대로 호스트에 두 개의 SATA OSD가 포함되어 있고 1GB/s 스위치가 5개의 호스트를 연결하는 경우 복구 시간이 훨씬 더 오래 걸립니다. 흥미로운 점은 이러한 크기의 클러스터에서 배치 그룹의 수는 데이터 내구성에 거의 영향을 미치지 않습니다. 배치 그룹 수는 128 또는 8192 일 수 있으며 복구 속도가 느려지거나 빨라지지 않습니다.

그러나 10개의 OSD 대신 동일한 Ceph 클러스터를 20개의 OSD로 확장하면 복구 속도가 빨라지므로 데이터 내구성이 크게 향상될 수 있습니다. 이유가 무엇입니까? 각 OSD는 이제 150개 대신 75개의 배치 그룹에만 참여하고 있습니다. OSD 클러스터에서는 복구하기 위해 여전히 19개의 OSD가 모두 동일한 양의 복사 작업을 수행해야 합니다. 10개의 OSD 클러스터에서 각 OSD는 약 100GB를 복사해야 했습니다. 20개의 OSD 클러스터에서는 각각 OSD를 각각 50GB만 복사하면 됩니다. 네트워크가 병목 상태인 경우 복구 작업은 두 번 수행됩니다. 즉, OSD 수가 증가함에 따라 복구 시간이 단축됩니다.

대규모 클러스터에서 PG 수가 중요합니다.

예시적인 클러스터가 40개의 OSD로 증가하는 경우 각 OSD는 35개의 배치 그룹만 호스팅합니다. OSD가 종료되면 다른 성능 장애로 개선이 발생하지 않는 한 복구 시간이 감소합니다. 그러나 이 클러스터가 200개의 OSD로 확장되는 경우 각 OSD는 약 7개의 배치 그룹만 호스팅합니다. OSD가 종료되면 이러한 배치 그룹의 최대 21개 (7 * 3) OSD 간에 복구가 수행됩니다. OSD가 40개가 있는 것보다 복구 시간이 길어지므로 배치 그룹 수가 증가해야 합니다.



중요

복구 시간이 단축되어도 복구가 진행되는 동안 배치 그룹을 저장하는 다른 OSD가 실패할 가능성이 있습니다.

위에서 설명한 10개의 OSD 클러스터에서 OSD가 실패하면 약 8개의 배치 그룹(즉, 복구되는 75개 pgs / 9 osds)은 단 하나의 복사만 갖습니다. 그리고 8개의 나머지 OSD 중 하나라도 실패하면 하나의 배치 그룹의 마지막 개체가 손실될 수 있습니다(즉, 남아 있는 사본 하나만 복구된 8 pgs / 8 osds). 따라서 다소 큰 클러스터(예: 50개의 OSD)로 시작하는 것이 좋습니다.

클러스터 크기가 20개의 OSD로 확장되면 OSD 3개가 손실되어 배치 그룹 수가 손상됩니다. 손실된 두 번째 OSD는 8 대신 약 2(즉, 35 pgs / 19 osds 가 복구됨)가 저하되고 세 번째 OSD 손실은 분리 사본이 포함된 두 개의 OSD 중 하나인 경우에만 데이터를 잃게 됩니다. 즉, 복구 시간 프레임 중 하나의 OSD가 0.0001% 일 가능성이 있는 경우 클러스터의 8 * 0.0001% 에서 20개의 OSD가 있는 클러스터에서 2 * 0.0001% 로 이동합니다. 데이터 내구성과 관련하여 OSD가 50개 미만인 클러스터에서는 512개 또는 4096개의 배치 그룹이 거의 동일합니다.

작은 정보

간단히 말해 더 많은 OSD를 사용하면 더 빠른 복구 속도가 빨라지고 배치 그룹 및 해당 개체가 영구적으로 손실되는 문제가 발생할 위험이 낮아집니다.

OSD를 클러스터에 추가할 때 배치 그룹 및 오브젝트로 새 OSD를 채우는 데 시간이 오래 걸릴 수 있습니다. 그러나 오브젝트의 성능 저하는 없으며 OSD를 추가하면 데이터 내구성에 영향을 미치지 않습니다.

3.3.2. 데이터 배포

Ceph는 핫 스팟을 방지하기 위해 노력합니다. 즉, 일부 OSD는 다른 OSD보다 훨씬 많은 트래픽을 수신합니다. CRUSH는 배치 그룹에 오브젝트를 배치 그룹에 균등하게 할당하여 배치 그룹에 균등하게 할당되면 기본 OSD는 클러스터 전체에 균등하게 분산되고 네트워크 초과 서브스크립션 문제와 데이터 배포로 인해 개발할 수 없도록 오브젝트를 저장하는 것이 좋습니다.

NetNamespace는 각 오브젝트에 대한 배치 그룹을 계산하기 때문에 이 배치 그룹 내의 각 OSD에 저장되는 데이터 양을 알 수 없으므로 배치 그룹 수와 OSD 수 간의 비율이 데이터 배포에 상당한 영향을 미칠 수 있습니다.

예를 들어 3개의 복제본 풀에 10개의 OSD가 있는 배치 그룹이 한 개뿐인 경우 Ceph는 3개의 OSD를 사용하여 데이터를 저장하고 다른 방법으로는 데이터를 저장할 수 없습니다. 더 많은 배치 그룹을 사용할 수 있는 경우 CRUSH는 OSD에 개체를 균등하게 분산할 수 있습니다. CRUSH는 배치 그룹도 OSD에 균등하게 할당합니다.

OSD보다 배치 그룹이 2개 이상 있는 순서가 1개 이상인 경우 배포도 완료되어야 합니다. 예를 들어 OSD 3개, 10개의 OSD용 512개 또는 1024개의 배치 그룹에 대한 256개의 배치 그룹 등이 있습니다.

OSD와 배치 그룹 간의 비율은 일반적으로 오브젝트 스트라이핑과 같은 고급 기능을 구현하는 Ceph 클라이언트의 데이터 배포 문제를 해결합니다. 예를 들어 4TB 블록 장치는 4MB 개체로 분할될 수 있습니다.

CRUSH는 개체 크기를 고려하지 않기 때문에 OSD와 배치 그룹 간의 비율은 다른 경우에 균등한 데이터 배포를 처리하지 않습니다. librados 인터페이스를 사용하여 비교적 작은 오브젝트를 저장하고 일부 매우 큰 오브젝트로 인해 데이터 배포가 저하될 수 있습니다. 예를 들어, 10개의 OSD에서 1000개의 배치 그룹에 총 100만 개의 4K 개체가 균등하게 분배됩니다. 각 OSD에서 4GB / 10 = 400MB 를 사용합니다. 400MB 개체가 풀에 추가되면 개체가 배치된 배치 그룹을 지원하는 3개의 OSD는 400MB + 400MB = 800MB 로 채워지고 나머지 7개의 OSD는 400MB로만 사용됩니다.

3.3.3. 리소스 사용량

각 배치 그룹의 경우 OSD 및 Ceph 모니터에는 항상 메모리, 네트워크 및 CPU, 복구 중에 더 많은 메모리가 필요합니다. 배치 그룹 내에서 개체를 클러스터링하여 이 오버헤드를 공유하는 것이 배치 그룹이 존재하는 주요 이유 중 하나입니다.

배치 그룹 수를 최소화하면 상당한 양의 리소스를 절약할 수 있습니다.

3.4. PG COUNT

풀의 배치 그룹 수는 클러스터 피어가 어떻게 데이터를 배포하며 데이터를 분산하고 리밸런스하는 데 중요한 역할을 합니다. 소규모 클러스터는 배치 그룹 수를 늘려 큰 클러스터와 비교하여 많은 성능 개선을 볼 수 없습니다. 그러나 여러 풀이 동일한 OSD에 액세스하는 경우 Ceph OSD에서 리소스를 효율적으로 사용할 수 있도록 PG 수를 신중하게 고려해야 할 수도 있습니다.

작은 정보

Red Hat은 OSD당 100~200개 PG를 권장합니다.

3.4.1. PG 계산기

PG 계산기는 사용자의 배치 그룹 수를 계산하고 특정 사용 사례를 해결합니다. PG 계산기는 Ceph Object Gateway와 같은 Ceph 클라이언트를 사용할 때 일반적으로 동일한 규칙(CRUSH 계층)을 사용하는 풀이 있는 경우 유용합니다. 여전히 Small Clusters 및 Calculating PG 개수에 대한 PG 수의 지침을 사용하여 PG를 수동으로 계산할 수 있습니다. 그러나 PG 계산기는 PG를 계산하는 데 선호되는 방법입니다.

자세한 내용은 Red Hat 고객 포털에서 풀 계산기당 Ceph PG(배치 그룹) 를 참조하십시오.

3.4.2. 기본 PG 수 구성

풀을 생성할 때 풀에 사용할 여러 배치 그룹도 생성합니다. 배치 그룹 수를 지정하지 않으면 Ceph에서 기본값 8 을 사용합니다. 이 값은 매우 낮습니다. 풀의 배치 그룹 수를 늘릴 수 있지만, Ceph 구성 파일에서도 적절한 기본값을 설정하는 것이 좋습니다.

```
osd pool default pg num = 100
osd pool default pgp num = 100
```

배치 그룹 수(total), 객체에 사용되는 배치 그룹 수(PG 분할에서 사용됨)를 모두 설정해야 합니다. 동등해야 합니다.

3.4.3. Small 클러스터의 PG 수

소규모 클러스터는 많은 수의 배치 그룹을 활용할 수 없습니다. OSD 수가 증가함에 따라 pg_num 및 pgp_num 에 대한 올바른 값을 선택하는 것은 클러스터 동작에 상당한 영향을 미치며 문제가 발생할 때 데이터의 내구성에 큰 영향을 미치기 때문에 더 중요합니다. 작은 클러스터에서 PG 계산기 를 사용하는 것이 중요합니다.

3.4.4. PG 수 계산

OSD가 50개 이상인 경우 리소스 사용량, 데이터 내구성 및 배포의 균형을 맞추기 위해 OSD당 약 50-100개의 배치 그룹을 사용하는 것이 좋습니다. OSD가 50개 미만인 경우 Small Clusters의 PG Count 중에서 선택하는 것이 좋습니다. 단일 개체 풀의 경우 다음 수식을 사용하여 기준선을 얻을 수 있습니다.For a single pool of objects, you can use the following formula to get a baseline:

```
            (OSDs * 100)
Total PGs =  ------------
              pool size
```

여기서 풀 크기는 복제된 풀의 복제본 수이거나 코딩된 풀의 K+M 합계입니다(ceph osd erasure-code-profile에 의해 반환됨).

그런 다음 데이터 내구성, 데이터 배포 및 리소스 사용량을 최소화하도록 Ceph 클러스터를 설계하는 방식과 결과가 적합한지 확인해야 합니다.

결과는 2의 가장 가까운 힘으로 반올림되어야합니다. rounding up은 선택 사항이지만 CRUSH가 배치 그룹 간에 개체 수를 균등하게 분산하는 것이 좋습니다.

200개의 OSD가 있고 풀 크기 3개의 복제본이 있는 클러스터의 경우 PG 수를 다음과 같이 추정합니다.

```
(200 * 100)
----------- = 6667. Nearest power of 2: 8192
    3
```

8192개의 배치 그룹을 사용하여 200개의 OSD에 분산하여 OSD당 약 41개의 배치 그룹으로 평가됩니다. 또한 각 풀에서 배치 그룹도 생성되므로 클러스터에서 사용할 수 있는 풀 수도 고려해야 합니다. 최대 PG 수 를 합리적인 지 확인하십시오.

3.4.5. PG 수 최대

오브젝트를 저장하는 데 여러 데이터 풀을 사용하는 경우 적절한 총 배치 그룹에 도달하도록 풀당 배치 그룹 수와 OSD당 배치 그룹 수의 균형을 유지해야 합니다. 시스템 리소스에 대한 과세를 늦추거나 피어링 프로세스를 너무 느릴 수 있도록 하는 것이 목표입니다.

10개의 OSD에서 512개의 배치 그룹이 있는 각 풀은 10개의 풀로 구성된 예시적인 Ceph 스토리지 클러스터에는 OSD당 총 5,120개의 배치 그룹이 10개 또는 OSD당 512개 배치 그룹이 있습니다. 하드웨어 구성에 따라 너무 많은 리소스를 사용하지 않을 수 있습니다. 대조적으로 각각 512개의 배치 그룹이 있는 1,000개의 풀을 생성하는 경우 OSD는 각각 ~50,000개의 배치 그룹을 처리하므로 더 많은 리소스가 필요

합니다. OSD당 너무 많은 배치 그룹을 사용하여 작업을 수행하면 특히 재조정 또는 복구 중에 성능이 크게 저하될 수 있습니다.

Ceph Storage Cluster는 OSD당 최대 300개의 배치 그룹이 있습니다. Ceph 구성 파일에서 다른 최대 값을 설정할 수 있습니다.

> mon pg warn max per osd

작은 정보

Ceph Object Gateway는 10-15개의 풀로 배포되므로 OSD당 100개 미만의 PG를 사용하여 합리적인 최대 수에 도달하는 것을 고려할 수 있습니다.

3.5. 자동 확장 배치 그룹

풀의 배치 그룹(PG) 수는 클러스터 피어링, 데이터 배포 및 리밸런스에서 중요한 역할을 합니다.

PGs를 자동 확장하면 클러스터를 보다 쉽게 관리할 수 있습니다. pg-autoscaling 명령은 PGs 스케일링을 위한 권장 사항을 제공하거나 클러스터 사용 방법에 따라 PG를 자동으로 스케일링합니다.

- 자동 확장의 작동 방식에 대한 자세한 내용은 3.5.1절. "배치 그룹 자동 확장" 을 참조하십시오.
- 자동 확장을 활성화하거나 비활성화하려면 3.5.3절. "배치 그룹 자동 확장 모드 설정" 을 참조하십시오.
- 배치 그룹 스케일링 권장 사항을 보려면 3.5.4절. "배치 그룹 확장 권장 사항 보기" 을 참조하십시오.
- 배치 그룹 자동 확장을 설정하려면 3.5.5절. "배치 그룹 자동 확장 설정" 을 참조하십시오.
- 자동 스케일러 프로필을 수동으로 업데이트하려면 참조하십시오.3.5.6절. "풀의 자동 스케일러 프로필 업데이트"
- 전역적으로 자동 스케일러를 업데이트하려면 다음을 참조하십시오. 3.5.7절. "noautoscale

플래그 업데이트”

- *대상 풀 크기를 설정하려면 **3.5.8**절. “대상 풀 크기 지정” 에서 참조하십시오.*

3.5.1. 배치 그룹 자동 확장

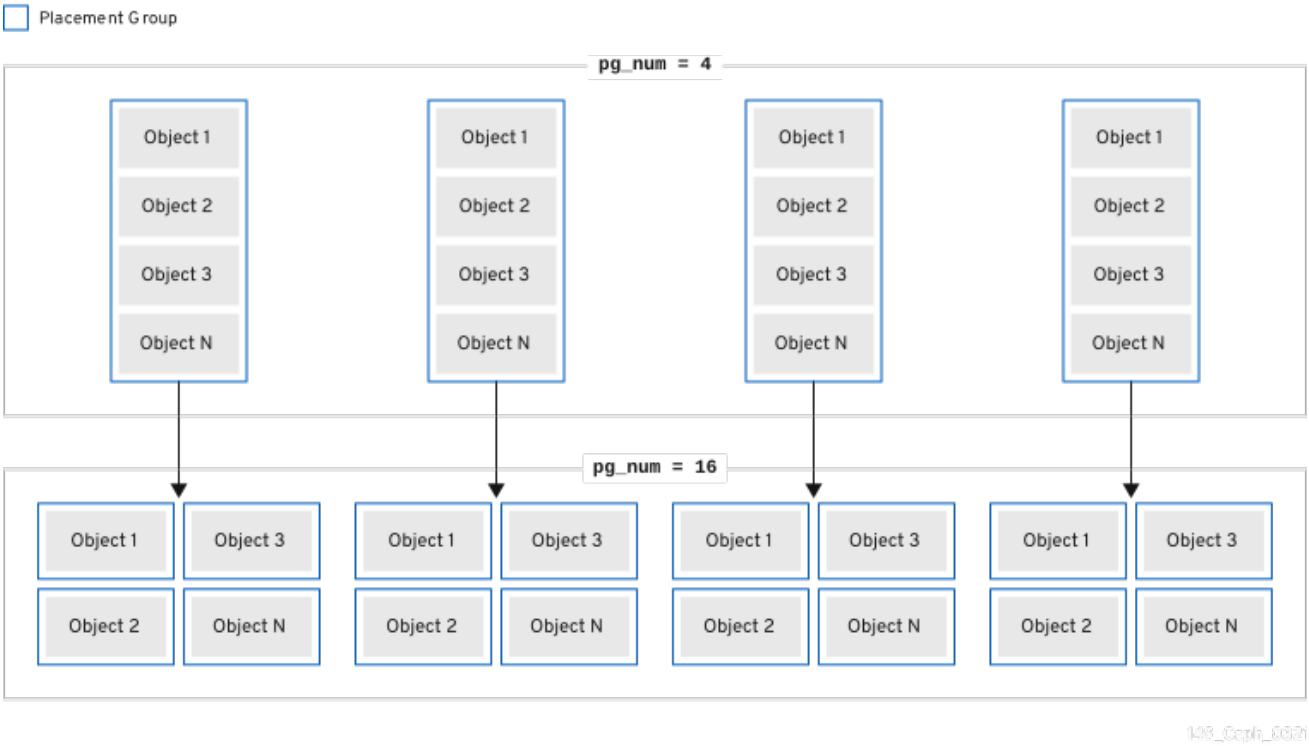
***Auto-scaler**의 작동 방식*

***auto-scaler**는 풀을 분석하고 각 하위 트리에 따라 조정합니다. 각 풀은 다른 **CRUSH** 규칙에 매핑할 수 있으며 각 규칙은 서로 다른 장치에 데이터를 배포할 수 있으므로 **Ceph**는 계층 구조의 각 하위 트리의 사용을 개별적으로 고려합니다. 예를 들어, 클래스 **ssd** 의 **OSD**에 매핑되는 풀과 클래스 **hdd** 의 **OSD**에 매핑되는 풀은 각각 각 장치 유형 수에 따라 최적의 **PG** 수를 갖습니다.*

3.5.2. 배치 그룹 분할 및 병합

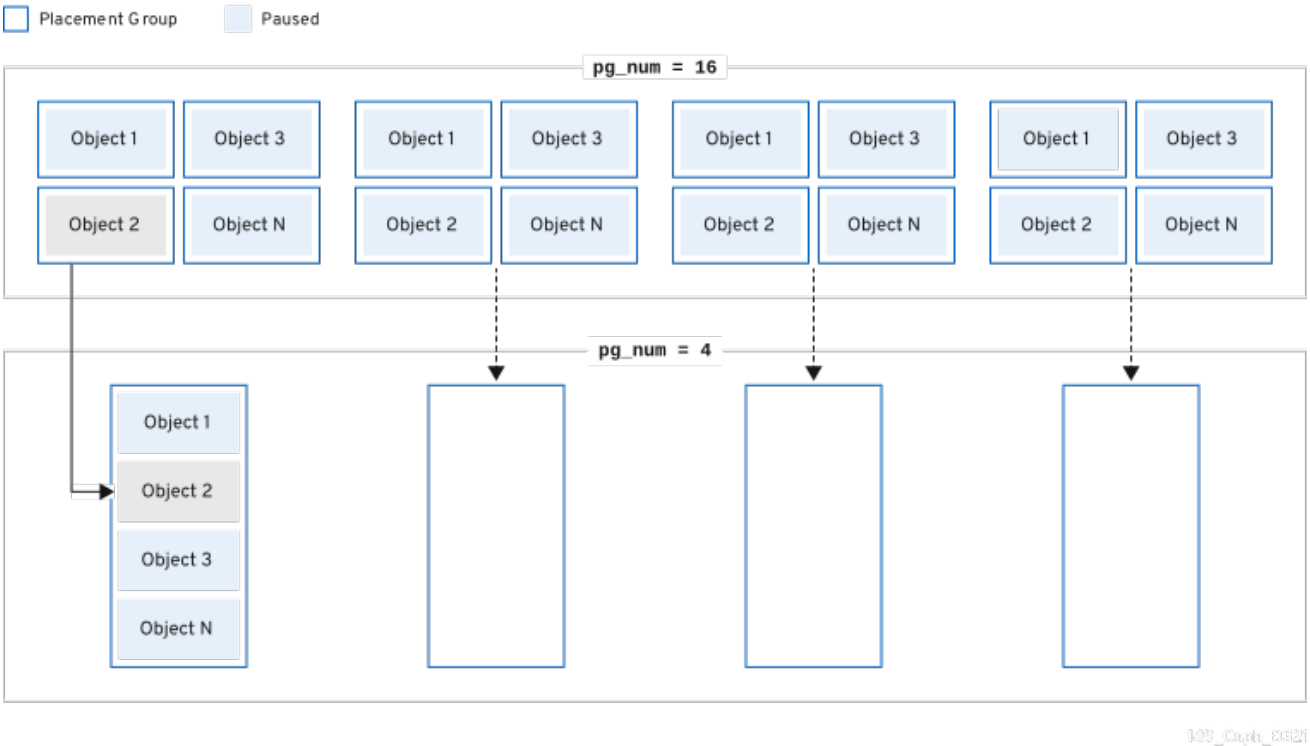
분할

***Red Hat Ceph Storage**는 기존 배치 그룹(**PG**)을 더 작은 **PG**로 분할할 수 있으므로 지정된 풀의 **PG** 의 총 수가 증가합니다. **PG**(기존 배치 그룹)를 분할하면 소규모 **Red Hat Ceph Storage** 클러스터를 통해 스토리지 요구 사항이 증가할 때 시간이 지남에 따라 확장될 수 있습니다. **PG** 자동 확장 기능은 **pg_num** 값을 증가시켜 스토리지 클러스터가 확장됨에 따라 기존 **PG**를 분할할 수 있습니다. **PG** 자동 확장 기능이 비활성화된 경우 **pg_num** 값을 수동으로 늘려 **PG** 분할 프로세스가 시작될 수 있습니다. 예를 들어 **pg_num** 값을 **4** 에서 **16** 으로 늘리면 **4**개 조각으로 분할됩니다. **pg_num** 값을 늘리면 **pgp_num** 값도 증가하지만 **pgp_num** 값은 점진적인 속도로 증가합니다. 오브젝트 데이터를 마이그레이션하면 시스템에 상당한 로드가 추가되므로 스토리지 클러스터의 성능과 클라이언트 워크로드에 미치는 영향을 최소화하기 위해 이러한 점진적 증가가 수행됩니다. 기본적으로 **Ceph** 큐는 **"misplaced"** 상태에 있는 오브젝트 데이터의 **5%** 이상 이동하지 않습니다. 이 기본 백분율은 **target_max_misplaced_ratio** 옵션으로 조정할 수 있습니다.*



병합

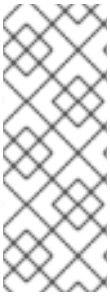
***Red Hat Ceph Storage**는 두 개의 기존 **PG**를 더 큰 **PG**로 병합할 수도 있으므로 **PG**의 총 수가 줄어듭니다. 두 **PG**를 함께 병합하는 것은 특히 풀의 상대적인 양의 개체가 시간이 지남에 따라 감소하거나 선택한 **PG**의 초기 수가 너무 크면 유용할 수 있습니다. **PG** 병합은 유용할 수 있지만 복잡하고 복잡한 절차이기도 합니다. 병합을 수행할 때 **PG**에 **I/O**를 일시 중지하면 한 번에 하나의 **PG**만 병합되어 스토리지 클러스터 성능에 미치는 영향을 최소화할 수 있습니다. **Ceph**는 새 **pg_num** 값에 도달할 때까지 오브젝트 데이터를 병합할 때 느리게 작동합니다.*



3.5.3. 배치 그룹 자동 확장 모드 설정

Red Hat Ceph Storage 클러스터의 각 풀에는 에서, 또는 경고로 설정할 수 있는 PG에 대한 pg_autoscale_mode 속성이 있습니다.

- off: 풀의 자동 확장을 비활성화합니다. 관리자는 각 풀에 적절한 PG 번호를 선택합니다. 자세한 내용은 PG count 섹션을 참조하십시오.
- On: 지정된 풀에 PG 수의 자동 조정 가능.
- warn: PG 수를 조정해야 할 때 상태 경고가 발생합니다.



참고

Red Hat Ceph Storage 5.0 이상 릴리스에서는 pg_autoscale_mode 가 기본적으로 켜져 있습니다. 업그레이드된 스토리지 클러스터는 기존 pg_autoscale_mode 설정을 유지합니다. pg_auto_scale 모드는 새로 생성된 풀에 사용됩니다. PG 수가 자동으로 조정되고 ceph 상태가 PG 개수를 조정하는 동안 복구 상태를 표시할 수 있습니다.

자동 스케일러는 대량 플래그를 사용하여 PG를 완전히 보완해야 하는 풀을 결정하고 풀 전체에서 사용 비율이 아닌 경우에만 축소됩니다. 그러나 풀에 대량 플래그가 없는 경우 풀은 최소 PG로 시작하고 풀에 더 많은 사용량이 있을 때만 풀이 시작됩니다.

참고

자동 스케일러는 중복되는 루트를 식별하고 이러한 루트가 있는 풀이 스케일링되지 않도록 하므로 확장 프로세스에 문제가 발생할 수 있으므로 이러한 루트가 확장되지 않습니다.

절차

- 기존 풀에서 자동 스케일링을 활성화합니다.

구문

```
ceph osd pool set POOL_NAME pg_autoscale_mode on
```

예제

```
[ceph: root@host01 /]# ceph osd pool set testpool pg_autoscale_mode on
```

- *새로 생성된 풀에서 자동 확장을 활성화합니다.*

구문

```
ceph config set global osd_pool_default_pg_autoscale_mode MODE
```

예제

```
[ceph: root@host01 /]# ceph config set global osd_pool_default_pg_autoscale_mode on
```

- ***bulk** 플래그를 사용하여 풀을 생성합니다.*

구문

```
ceph osd pool create POOL_NAME --bulk
```

예제

```
[ceph: root@host01 /]#  ceph osd pool create testpool --bulk
```

- *기존 풀의 대량 플래그를 설정하거나 설정 해제합니다.*

구문

```
ceph osd pool set POOL_NAME bulk TRUE/FALSE/1/0
```

예제

```
[ceph: root@host01 /]#  ceph osd pool set testpool bulk true
```

- *기존 풀의 대량 플래그를 가져옵니다.*

구문

```
ceph osd pool get POOL_NAME bulk
```

예제

```
[ceph: root@host01 /]# ceph osd pool get testpool bulk
```

3.5.4. 배치 그룹 확장 권장 사항 보기

풀, 상대 사용률, 스토리지 클러스터의 **PG** 수에 대한 제안된 변경 사항을 볼 수 있습니다.

사전 요구 사항

- 실행 중인 **Red Hat Ceph Storage** 클러스터
- 모든 노드에 대한 루트 수준 액세스입니다.

절차

- 각 풀, 상대 사용률, 을 사용하여 **PG** 수에 대한 제안된 변경 사항을 볼 수 있습니다.

```
[ceph: root@host01 /]# ceph osd pool autoscale-status
```

출력은 다음과 유사합니다.

```
POOL                    SIZE  TARGET SIZE  RATE  RAW CAPACITY   RATIO  TARGET RATIO
EFFECTIVE RATIO  BIAS  PG_NUM  NEW PG_NUM  AUTOSCALE  BULK
device_health_metrics      0               3.0        374.9G  0.0000                                  1.0       1
on         False
cephfs.cephfs.meta     24632               3.0        374.9G  0.0000                                  4.0
32           on         False
cephfs.cephfs.data         0               3.0        374.9G  0.0000                                  1.0      32
on         False
.rgw.root               1323               3.0        374.9G  0.0000                                  1.0      32
on         False
default.rgw.log         3702               3.0        374.9G  0.0000                                  1.0      32
on         False
default.rgw.control        0               3.0        374.9G  0.0000                                  1.0      32
on         False
default.rgw.meta         382               3.0        374.9G  0.0000                                  4.0       8
on         False
```

SIZE 는 풀에 저장된 데이터 양입니다.

존재하는 경우 관리자는 이 풀에 저장될 것으로 예상되는 데이터의 양입니다. 시스템은 계산에 두 값 중 더 큰 값을 사용합니다.

***RATE** 는 풀에서 사용하는 원시 스토리지 용량을 결정하는 풀의 다중값입니다. 예를 들어 **3** 복제본 풀의 비율은 **3.0** 이며 **k=4,m=2** 세대 코딩된 풀의 비율은 **1.5** 입니다.*

***RAW CAPACITY** 는 풀의 데이터를 저장하는 **OSD**의 총 원시 스토리지 용량입니다.*

***RATIO** 는 풀이 사용하고 있는 총 용량, 즉 **ratio = size * rate / raw** 용량의 비율입니다.*

***CACHEGET RATIO** (있는 경우)는 관리자가 대상 비율이 설정된 다른 풀과 상대적으로 사용되기를 예상하는 스토리지의 비율입니다. 대상 크기 바이트와 비율을 모두 지정하면 비율이 우선합니다. 풀을 생성하는 동안 지정되지 않은 경우 **CACHEGET RATIO** 의 기본값은 **0** 입니다. 풀에서 제공하는 **--target_ratio** 가 많을수록 풀에서 예상하는 **PG**가 커집니다.*

***EFFECTIVE RATIO.**는 두 가지 방법으로 조정한 후 대상 비율입니다. **1.** 대상 크기 세트가 있는 풀에서 사용할 것으로 예상되는 용량을 뺀 값입니다. **2.** 대상 비율이 설정된 풀 간에 대상 비율을 정규화하여 나머지 공간을 전체적으로 대상으로 합니다. 예를 들어, 대상 비율 **1.0**이 있는 **4** 풀은 유효 비율은 **0.25**입니다. 시스템은 실제 비율의 더 큰 비율과 계산에 대한 효과적인 비율을 사용합니다.*

***BIAS.**는 특정 풀에 할당될 것으로 예상되는 양에 대한 이전 정보를 기반으로 풀 **PG**를 수동으로 조정하는 데 사용됩니다. 풀을 만들 때 지정하지 않는 한 기본적으로 값은 **1.0**인 경우입니다. 풀에서 제공하는 **--bias** 가 많을수록 풀에서 예상하는 **PG**가 증가합니다.*

***PG_NUM** 은 풀의 현재 **PG** 수 또는 **pg_num** 변경이 진행 중인 경우 풀의 현재 작업 중인 **PG** 수입니다. **NEW PG_NUM** (있는 경우)은 제안된 **PG(pg_num)**입니다. 이는 항상 **2**의 힘이며 제안된 값이 **3** 인수보다 현재 값과 다른 경우에만 존재합니다.*

***AUTOSCALE** 은 **pg_autoscale_mode** 풀이며, 해제 또는 경고 중 하나입니다.*

***PROFILE, scale-up** 또는 **scale-down** 프로필일 수 있습니다.*

3.5.5. 배치 그룹 자동 확장 설정

*클러스터에서 클러스터 사용량에 따라 **PG**를 자동으로 확장할 수 있도록 하는 것이 **PG**를 확장하는 가*

장 간단한 방법입니다. **Red Hat Ceph Storage**는 전체 시스템에 대해 사용 가능한 총 스토리지 및 대상 **PG** 수를 사용하고 각 풀에 저장된 데이터의 양을 비교하며 그에 따라 **PG**를 예상합니다. 명령은 현재 **PG** 수(**pg_num**)가 계산된 **PG** 번호 또는 제안된 **PG** 번호에서 세 번 이상 꺼져 있는 풀을 변경합니다.

OSD당 **PG**의 대상 수는 구성 가능한 **mon_target_pg_per_osd** 를 기반으로 합니다. 기본값은 **100** 으로 설정됩니다.

절차

1. **To adjust mon_target_pg_per_osd:**

```
ceph config set global mon_target_pg_per_osd number
```

예를 들어 다음과 같습니다.

```
$ ceph config set global mon_target_pg_per_osd 150
```

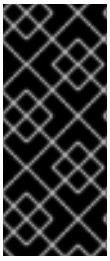
3.5.6. 풀의 자동 스케일러 프로필 업데이트

Red Hat Ceph Storage 5.1부터 배치 그룹(**PG**) 자동 스케일러가 기본적으로 활성화되어 있습니다. 자동 스케일러가 예상 방향으로 확장되지 않으면 프로필을 스케일링하거나 스케일 업으로 설정하여 수동으로 구성할 수 있습니다.

스케일 다운 프로필의 경우 자동 스케일러는 중복된 루트를 식별하고 이러한 루트가 있는 풀이 스케일링 프로세스의 문제가 발생할 수 있으므로 스케일링할 수 없습니다. 자동 스케일러는 이러한 풀에서 확장 배치 그룹을 건너뛰고 다음 경고 메시지가 표시됩니다.

예제

```
pool _POOL_ID_ contains an overlapping root _ID_… skipping scaling
```



중요

*기존 스토리지 클러스터는 **scale-up** 프로필을 사용합니다. **scale-down** 프로필을 사용하려면 최신 **Red Hat Ceph Storage** 버전으로 업그레이드한 다음 **autoscale** 프로필을 축소해야 합니다.*

사전 요구 사항

- *실행 중인 **Red Hat Ceph Storage** 클러스터*
- *모든 노드에 대한 루트 수준 액세스입니다.*

절차

1. ***autoscale** 프로필이 올바르게 구성되었는지 확인합니다.*

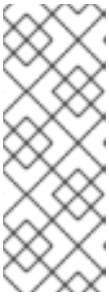
   예제

   ```
   [ceph: root@host01 /]# ceph osd pool autoscale-status
   ```

2. ***autoscale-profile** 을 사용하여 프로필을 업데이트합니다.*

   구문

   ```
   ceph osd pool set autoscale-profile [scale-up | scale-down]
   ```



참고

*기본적으로 **PROFILE** 은 **scale-up** 으로 설정됩니다. 선택한 프로필에 따라 클러스터가 **PG**를 재조정하기 시작합니다. 스케일 다운 프로필은 더 나은 사용자 환경을 제공하기 위해 도입되었지만 **PG**를 너무 크게 스케일링할 수 있으며 풀 생성 중에 문제가 있을 수 있습니다.*

예제

```
[ceph: root@host01 /]# ceph osd pool set autoscale-profile scale-down

autoscale-profile is now scale-down
```

3. *자동 스케일러 활동을 확인합니다.*

예제

```
[ceph: root@host01 /]# ceph progress

[==============================]
[Complete]: PG autoscaler increasing pool 1 PGs from 1 to 64 (7m)
    [==============================]
[Complete]: Global Recovery Event (6m)
    [==============================]
[Complete]: PG autoscaler increasing pool 2 PGs from 32 to 256 (7m)
    [==============================]
[Complete]: PG autoscaler increasing pool 7 PGs from 8 to 256 (6m)
    [==============================]
[Complete]: Global Recovery Event (6m)
    [==============================]
```

*이 명령은 **PH** 확장이 진행 중인 풀에 대한 세부 정보를 제공합니다.*

4. ***NEW_PG_NUM** 열을 확인하여 **PG** 자동 스케일링 상태를 확인합니다.*

예제

```
[ceph: root@host01 /]# ceph osd pool autoscale-status

POOL                    SIZE  TARGET SIZE  RATE  RAW CAPACITY   RATIO  TARGET RATIO
EFFECTIVE RATIO  BIAS  PG_NUM  NEW PG_NUM  AUTOSCALE  PROFILE
device_health_metrics  28951                3.0        254.9G  0.0000                                  1.0
64           1  on         scale-up
cephfs.cephfs.meta     45774                3.0        254.9G  0.0000                                  4.0
256          16  on         scale-up
```

3.5.7. noautoscale 플래그 업데이트

*모든 풀의 자동 스케일러를 활성화하거나 비활성화하려면 **noautoscale** 전역 플래그를 사용할 수 있습니다. 이 글로벌 플래그는 일부 **OSD**가 반송되거나 클러스터가 유지 관리 중인 경우 스토리지 클러스터를 업그레이드하는 동안 유용합니다. 활동이 완료되면 플래그를 설정하고 설정을 해제할 수 있습니다.*

*기본적으로 **noautoscale** 플래그는 **off** 로 설정됩니다. 이 플래그가 설정되면 모든 풀에 **pg_autoscale_mode** 가 **off** 이고 모든 풀에는 자동 스케일러가 비활성화됩니다.*

사전 요구 사항

- *실행 중인 **Red Hat Ceph Storage** 클러스터*
- *모든 노드에 대한 루트 수준 액세스입니다.*

절차

1. ***noautoscale** 플래그의 값을 가져옵니다.*

예제

```
[ceph: root@host01 /]# ceph osd pool get noautoscale
```

2. *활동 전에 **noautoscale** 플래그를 설정합니다.*

예제

```
[ceph: root@host01 /]# ceph osd pool set noautoscale
```

3. *활동 완료 시 **noautoscale** 플래그를 설정 해제합니다.*

예제

```
[ceph: root@host01 /]# ceph osd pool unset noautoscale
```

3.5.8. 대상 풀 크기 지정

*새로 생성된 풀은 총 클러스터 용량의 일부만 사용하고 **PG**가 적은 수의 시스템에 나타납니다. 그러나 대부분의 경우 클러스터 관리자는 시간이 지남에 따라 대부분의 시스템 용량을 사용할 것으로 예상되는 풀을 알고 있습니다. 이 정보를 **Red Hat Ceph Storage**의 대상 크기 라고 하는 경우 이러한 풀은 처음부터 더 적절한 개수의 **PG(pg_num)**를 사용할 수 있습니다. 이 접근 방식은 **pg_num** 의 후속 변경 사항 및 이러한 조정을 수행할 때 데이터 이동과 관련된 오버헤드를 방지합니다.*

다음과 같은 방법으로 풀의 대상 크기를 지정할 수 있습니다.

- *3.5.8.1절. “풀의 절대 크기를 사용하여 대상 크기 지정”*
- *3.5.8.2절. “총 클러스터 용량을 사용하여 대상 크기 지정”*

3.5.8.1. 풀의 절대 크기를 사용하여 대상 크기 지정

절차

1. 풀의 절대 크기를 바이트 단위로 사용하여 대상 크기를 설정합니다.

```
ceph osd pool set pool-name target_size_bytes value
```

예를 들어 **mypool** 이 **100T** 공간을 차지할 것으로 예상되는 시스템에 지시하려면 다음을 수행합니다.

```
$ ceph osd pool set mypool target_size_bytes 100T
```

ceph osd pool create 명령에 **--target-size-bytes <bytes >** 인수를 추가하여 생성 시 풀의 대상 크기를 설정할 수도 있습니다.

3.5.8.2. 총 클러스터 용량을 사용하여 대상 크기 지정

절차

1. 총 클러스터 용량의 비율을 사용하여 대상 크기를 설정합니다.

```
ceph osd pool set pool-name target_size_ratio ratio
```

예를 들면 다음과 같습니다.

```
$ ceph osd pool set mypool target_size_ratio 1.0
```

target_size_ratio 가 설정된 다른 풀과 관련하여 **mypool** 풀이 **1.0**을 사용할 것으로 예상되는 시스템에 지시합니다. **mypool** 이 클러스터의 유일한 풀인 경우 총 용량의 **100%** 사용량을 예상합니다. **target_size_ratio** 가 **1.0**인 두 번째 풀이 있는 경우 두 풀 모두 클러스터 용량의 **50%**를 사용할 것으로 예상됩니다.

ceph osd pool create 명령에 **--target-size-ratio <ratio >** 인수를 추가하여 생성 시 풀의 대상 크기를 설정할 수도 있습니다.

참고

불가능한 대상 크기 값(예: 총 클러스터보다 큰 용량) 또는 1.0보다 큰 용량을 지정하면 클러스터에서 POOL_ECDHEGET_SIZE_RATIO_OVERCOMMSIZE_BYTES_ BYTES_OVER COMMVERCOMMTES_OVERCOMMITTED 상태 경고가 발생합니다.

풀에 target_size_ratio 와 target_size_bytes 를 모두 지정하는 경우 클러스터는 비율만 고려하여 POOL_HAS_CACHEGET_BYTES_AND_RATIO 상태 경고가 발생합니다.

3.6. PG 명령줄 참조

ceph CLI를 사용하면 풀의 배치 그룹 수를 설정하고 가져오고 PG 맵을 보고 PG 통계를 검색할 수 있습니다.

3.6.1. PG 수 설정

풀의 배치 그룹 수를 설정하려면 풀을 생성할 때 배치 그룹 수를 지정해야 합니다. 자세한 내용은 풀 생성을 참조하십시오. 풀에 배치 그룹을 설정하면 배치 그룹 수를 늘릴 수 있지만 배치 그룹 수는 줄어들 수 없습니다. 배치 그룹 수를 늘리려면 다음을 실행합니다.

```
ceph osd pool set {pool-name} pg_num {pg_num}
```

배치 그룹 수를 늘리면 클러스터를 재조정하기 전에 배치 그룹(pgp_num)의 수도 늘려야 합니다. pgp_num 은 pg_num 과 같아야 합니다. 배치 배치 그룹 수를 늘리려면 다음을 실행합니다.

```
ceph osd pool set {pool-name} pgp_num {pgp_num}
```

3.6.2. PG 수 가져오기

풀의 배치 그룹 수를 가져오려면 다음을 실행합니다.

```
ceph osd pool get {pool-name} pg_num
```

3.6.3. 클러스터 PG 통계 가져오기

클러스터에서 배치 그룹에 대한 통계를 가져오려면 다음을 실행합니다.

```
ceph pg dump [--format {format}]
```

유효한 형식은 **plain** (기본값) 및 **json** 입니다.

3.6.4. Stuck PGs에 대한 통계 가져오기

모든 배치 그룹의 통계를 지정된 상태로 유지하려면 다음을 실행합니다.

```
ceph pg dump_stuck {inactive|unclean|stale|undersized|degraded
[inactive|unclean|stale|undersized|degraded...]} {<int>}
```

비활성 배치 그룹은 최신 데이터가 표시되도록 **OSD**를 기다리고 있기 때문에 읽기 또는 쓰기를 처리할 수 없습니다.

불명확한 배치 그룹에는 원하는 횟수를 복제하지 않는 오브젝트가 포함되어 있습니다. 회복을 해야 합니다.

오래된 배치 그룹은 알 수 없는 상태입니다 - 호스팅하는 **OSD**는 잠시 동안 모니터 클러스터에 보고되지 않았습니다(**mon_osd_report_timeout).**

유효한 형식은 **plain** (기본값) 및 **json** 입니다. 임계값은 배치 그룹이 반환된 통계(기본값: **300**초)에 포함하기 전에 배치 그룹이 정지되는 최소 시간(기본값 **300**초)을 정의합니다.

3.6.5. PG 맵 가져오기

특정 배치 그룹에 대한 배치 그룹 맵을 가져오려면 다음을 실행합니다.

```
ceph pg map {pg-id}
```

예를 들어 다음과 같습니다.

```
ceph pg map 1.6c
```

Ceph는 배치 그룹 맵, 배치 그룹, **OSD** 상태를 반환합니다.

```
osdmap e13 pg 1.6c (1.6c) -> up [1,0] acting [1,0]
```

3.6.6. PGs 통계 받기

특정 배치 그룹에 대한 통계를 검색하려면 다음을 실행합니다.

```
ceph pg {pg-id} query
```

3.6.7. 배치 그룹 scrub

배치 그룹을 **scrub**하려면 다음을 실행합니다.

```
ceph pg scrub {pg-id}
```

Ceph는 기본 및 모든 복제본 노드를 확인하고, 배치 그룹에 모든 오브젝트 카탈로그를 생성하고 이를 비교하여 오브젝트가 누락되거나 일치하지 않는지 확인하고 해당 콘텐츠가 일관되게 유지되도록 합니다. 복제본이 모두 일치하면 최종 의미 체계 스위프를 사용하면 모든 스냅샷 관련 개체 메타데이터가 일관되게 유지됩니다. 오류를 로그를 통해 보고됩니다.

3.6.8. 되돌리기

클러스터가 하나 이상의 개체를 손실했으며 손실된 데이터 검색을 중단하기로 결정한 경우 **unfound** 개체를 손실된 것으로 표시해야 합니다.

가능한 모든 위치가 쿼리되어 있고 오브젝트가 계속 손실된 경우에도 손실된 오브젝트를 포기해야 할 수 있습니다. 이는 클러스터가 쓰기 자체를 복구하기 전에 수행된 쓰기에 대해 알아볼 수 있는 비정상적인 오류 조합이 있을 수 있습니다.

현재 지원되는 유일한 옵션은 "반복"으로, 이전 버전의 오브젝트로 롤백하거나 (새 객체인 경우) 완전히 잊어버렸습니다. **"unfound"** 오브젝트를 **"lost"**로 표시하려면 다음을 실행합니다.

```
ceph pg {pg-id} mark_unfound_lost revert|delete
```

중요

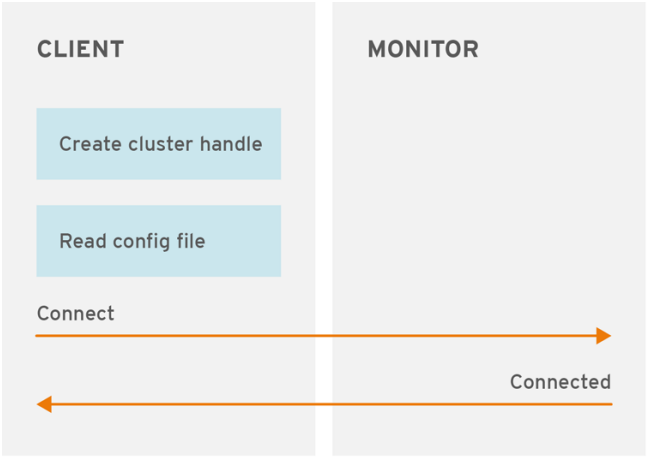
개체가 존재할 것으로 예상되는 애플리케이션을 혼동할 수 있으므로 이 기능을 주의해서 사용하십시오.

4장. POOL

Ceph 클라이언트는 데이터를 풀에 저장합니다. 풀을 생성할 때 클라이언트가 데이터를 저장할 I/O 인터페이스를 생성하고 있습니다. Ceph 클라이언트(즉, 블록 장치, 게이트웨이 및 나머지)의 관점에서 보면 Ceph 스토리지 클러스터와 상호 작용하기 때문에 클러스터 처리를 생성하고 클러스터에 연결한 다음, 오브젝트와 확장된 속성을 읽고 쓰는 I/O 컨텍스트를 만들 수 있습니다.

클러스터 처리 및 클러스터에 연결 만들기Create a Cluster Handle and Connect to the Cluster

Ceph 스토리지 클러스터에 연결하려면 Ceph 클라이언트에는 클러스터 이름(일반적으로 ceph)과 초기 모니터 주소가 필요합니다. 일반적으로 Ceph 클라이언트는 Ceph 구성 파일의 기본 경로를 사용하여 이러한 매개 변수를 검색한 다음 파일에서 읽습니다. 그러나 사용자는 명령줄에서 매개 변수를 지정할 수도 있습니다. 또한 Ceph 클라이언트는 사용자 이름 및 비밀 키(기본적으로 인증) 를 제공합니다. 그런 다음 클라이언트는 Ceph 모니터 클러스터에 연결하고 모니터, OSD 및 풀을 비롯한 최근 클러스터 맵 복사본을 검색합니다.



풀 I/O 컨텍스트 생성

Ceph 클라이언트는 데이터를 읽고 쓰기 위해 Ceph 스토리지 클러스터의 특정 풀에 i/o 컨텍스트를 생성합니다. 지정된 사용자에게 풀에 대한 권한이 있는 경우 Ceph 클라이언트는 지정된 풀에서 읽고 쓸 수 있습니다.

Ceph의 아키텍처를 사용하면 스토리지 클러스터에서 Ceph 클라이언트에 놀라운 간단한 인터페이스를 제공할 수 있으므로, 클라이언트는 풀 이름을 지정하고 I/O 컨텍스트를 생성하여 간단히 정의한 정교한 스토리지 전략 중 하나를 선택할 수 있습니다. 스토리지 전략은 모든 용량 및 성능에서 Ceph 클라이언트로 볼 수 없습니다. 마찬가지로, Ceph 클라이언트의 복잡성을 블록 장치 표현으로 매핑하여 S3/Swift RESTful 서비스를 제공하는 것은 Ceph 스토리지 클러스터에서 보이지 않습니다.

풀은 다음을 제공합니다.

- 복원력: 데이터 손실 없이 실패할 수 있는 OSD 수를 설정할 수 있습니다. 복제된 풀의 경우 원하는 수의 개체 복사본/복제본 수입니다. 일반적인 구성에서는 오브젝트와 하나의 추가 사본(즉, size = 2)을 저장하지만 복사/복제본 수를 확인할 수 있습니다. 레코저 코딩된 풀의 경우 코딩 청크 수(오버저 코드 프로파일의 m=2)입니다.

- 배치 그룹: 풀의 배치 그룹 수를 설정할 수 있습니다. 일반적인 구성에서는 OSD당 약 50-100 개의 배치 그룹을 사용하여 너무 많은 컴퓨팅 리소스를 사용하지 않고도 최적의 균형을 조정합니다. 여러 개의 풀을 설정할 때는 풀과 클러스터 전체에 적절한 개수의 배치 그룹을 설정해야 합니다.

- CRUSH 규칙: 풀에 데이터를 저장할 때 풀에 매핑된 CRUSH 규칙을 사용하면 CRUSH가 각 개체의 배치 규칙과 클러스터의 코딩된 풀에 대한 복제본(또는 청크)을 식별할 수 있습니다. 풀에 대한 사용자 지정 CRUSH 규칙을 만들 수 있습니다.

- 스냅샷: ceph osd pool mksnap 으로 스냅샷을 생성하면 특정 풀의 스냅샷을 효과적으로 수행할 수 있습니다.

- Quotas: ceph osd pool set-quota 로 풀에서 할당량을 설정하면 최대 오브젝트 수 또는 지정된 풀에 저장된 최대 바이트 수를 제한할 수 있습니다.

4.1. 풀 및 스토리지 전략

풀을 관리하려면 풀을 나열, 생성 및 제거할 수 있습니다. 각 풀의 사용률 통계를 볼 수도 있습니다.

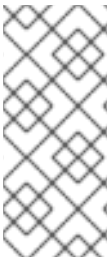
4.2. 풀 나열

클러스터 풀을 나열하려면 다음을 실행합니다.

```
ceph osd lspools
```

4.3. 풀 생성

풀을 생성하기 전에 **Red Hat Ceph Storage 5**의 구성 가이드의 풀, **PG** 및 **CRUSH** 구성 참조 장을 참조하십시오.



참고

Red Hat Ceph Storage 3 이상 릴리스에서는 시스템 관리자가 **Ceph** 클라이언트에서 **I/O** 작업을 수신하도록 풀을 명시적으로 활성화해야 합니다. 자세한 내용은 애플리케이션 사용을 참조하십시오. 풀을 활성화하지 않으면 **HEALTH_WARN** 상태가 됩니다.

기본값은 필요에 따라 필요하지 않으므로 **Ceph** 구성 파일의 배치 그룹 수에 대한 기본값을 조정하는 것이 좋습니다. 예를 들어 다음과 같습니다.

```
osd pool default pg num = 100
osd pool default pgp num = 100
```

복제된 풀을 생성하려면 다음을 실행합니다.

```
ceph osd pool create <pool-name> <pg-num> <pgp-num> [replicated] \
     [crush-rule-name] [expected-num-objects]
```

삭제 코드된 풀을 생성하려면 다음을 실행합니다.

```
ceph osd pool create <pool-name> <pg-num> <pgp-num> erasure \
     [erasure-code-profile] [crush-rule-name] [expected-num-objects]
```

다음과 같습니다.

pool-name

설명

풀의 이름입니다. 이는 고유해야 합니다.

유형

문자열

필수 항목

*네, 필요합니다. 지정하지 않으면 **Ceph** 구성 파일에 나열된 값 또는 기본값으로 설정됩니다.*

기본값

ceph

pg-num

설명

풀의 총 배치 그룹 수입니다. 적절한 수를 계산하는 방법에 대한 자세한 내용은 배치 그룹 섹션 및 풀당 Ceph Placement Groups (PG) 를 참조하십시오. 기본값은 대부분의 시스템에 적합하지 않습니다.

유형

정수

필수 항목

있음

기본값

8

pgp-num

설명

배치 목적으로 총 배치 그룹 수입니다. 이 값은 배치 그룹 분할 시나리오를 제외하고 총 배치 그룹 수와 같아야 합니다.

유형

정수

필수 항목

*네, 필요합니다. 지정하지 않으면 **Ceph** 구성 파일에 나열된 값 또는 기본값으로 설정됩니다.*

기본값

8

복제 또는 삭제

설명

*풀 유형을 복제 하여 개체 또는 삭제 의 여러 복사본을 유지하여 일반화된 **RAID5** 기능을 가져와서 손실된 **OSD**에서 복구할 수 있습니다. 복제된 풀에는 더 많은 원시 스토리지가 필요하지만 모든 **Ceph** 작업을 구현합니다. 소거 코드된 풀은 더 적은 원시 스토리지가 필요하지만 사용 가능한 작업의 하위 집합만 구현합니다.*

유형

문자열

필수 항목

없음

기본값

replicated

crush-rule-name

설명

*풀에 대한 크러쉬 규칙의 이름입니다. 규칙이 있어야 합니다. 복제된 풀의 경우 **name**은 **osd_pool_default_crush_rule** 구성 설정에서 지정하는 규칙입니다. 크기가 조정된 풀의 경우 기본 삭제 코드 프로필 또는 **{pool-name}** 을 지정하지 않으면 이름이 지워집니다. 규칙이 아직 없는 경우 **Ceph**에서 지정된 이름으로 이 규칙을 암시적으로 생성합니다.*

유형

문자열

필수 항목

없음

기본값

*삭제 코드(**Trasure- coded pool**)를 사용합니다. 복제된 풀의 경우 **Ceph** 구성의 **osd_pool_default_crush_rule** 변수 값을 사용합니다.*

expected-num-objects

설명

*풀에 필요한 오브젝트 수입니다. 이 값을 음수 **filestore_merge_threshold** 변수와 함께 설정하면 **Ceph**가 런타임 디렉터리 분할에 대기 시간이 영향을 주지 않도록 풀 생성 시 배치 그룹을 분할합니다.*

유형

정수

필수 항목

없음

기본값

0, 풀 생성 시 분할 없음

erasure-code-profile

설명

소거 코드된 풀의 경우에만 사용됩니다. 삭제 코드 프로필을 사용합니다. Ceph 구성 파일의 **osd erasure-code-profile set** 변수에 의해 정의된 기존 프로필이어야 합니다. 자세한 내용은 Erasure Code Profiles 섹션을 참조하십시오.

유형

문자열

필수 항목

없음

풀을 생성할 때 배치 그룹의 수를 적절한 값(예: 100)으로 설정합니다. OSD당 총 배치 그룹 수도 고려합니다. 배치 그룹은 계산 비용이 많이 들기 때문에 배치 그룹이 많은 풀(예: 각각 100개의 배치 그룹이 있는 풀 50개)이 있는 경우 성능이 저하됩니다. 저하되는 반환 지점은 OSD 호스트의 성능에 따라 달라집니다.

풀에 적합한 배치 그룹 수를 계산하는 방법에 대한 자세한 내용은 배치 그룹 섹션 및 풀당 Ceph Placement Groups(PG) 를 참조하십시오.

4.4. 풀 할당량 설정

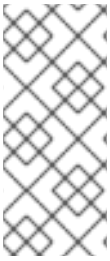
최대 바이트 수 또는 풀당 최대 오브젝트 수 또는 둘 다에 대해 풀 할당량을 설정할 수 있습니다.

```
ceph osd pool set-quota <pool-name> [max_objects <obj-count>] [max_bytes <bytes>]
```

예를 들어 다음과 같습니다.

```
ceph osd pool set-quota data max_objects 10000
```

할당량을 제거하려면 해당 값을 **0** 으로 설정합니다.



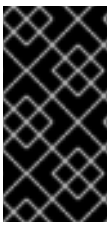
참고

진행 중인 쓰기 작업은 **Ceph**가 클러스터 전체에서 풀 사용량을 전파할 때까지 단기간에 풀 할당량을 덮어쓸 수 있습니다. 이것은 정상적인 행동입니다. 진행 중인 쓰기 작업에서 풀 할당량을 적용하면 상당한 성능 저하가 발생합니다.

4.5. 풀 삭제

풀을 삭제하려면 다음을 실행합니다.

```
ceph osd pool delete <pool-name> [<pool-name> --yes-i-really-really-mean-it]
```



중요

데이터를 보호하기 위해 스토리지 관리자는 기본적으로 풀을 삭제할 수 없습니다. 풀을 삭제하기 전에 **mon_allow_pool_delete** 구성 옵션을 설정합니다.

풀에 자체 규칙이 있는 경우 풀을 삭제한 후 제거하는 것이 좋습니다. 풀에 자체 용도로 엄격하게 사용자가 있는 경우 풀을 삭제한 후 해당 사용자를 삭제하는 것이 좋습니다.

4.6. 풀 이름 지정

풀 이름을 바꾸려면 다음을 실행합니다.

```
ceph osd pool rename <current-pool-name> <new-pool-name>
```

풀의 이름을 변경하고 인증된 사용자의 풀 기능을 사용하는 경우 새 풀 이름으로 사용자의 기능(즉, 제한)을 업데이트해야 합니다.

4.7. 풀 통계 표시

풀의 사용률 통계를 표시하려면 다음을 실행합니다.

```
rados df
```

4.8. 풀의 스냅샷 만들기

풀의 스냅샷을 작성하려면 다음을 실행합니다.

```
ceph osd pool mksnap <pool-name> <snap-name>
```

주의

풀 스냅샷을 만들면 풀 내에서 **RBD** 이미지 스냅샷을 만들 수 없으며 되돌릴 수 없습니다.

4.9. 풀의 스냅샷 제거

풀의 스냅샷을 제거하려면 다음을 실행합니다.

```
ceph osd pool rmsnap <pool-name> <snap-name>
```

4.10. 풀 값 설정

값을 풀에 설정하려면 다음 명령을 실행합니다.

```
ceph osd pool set <pool-name> <key> <value>
```

풀 값 섹션에는 설정할 수 있는 모든 키-값 쌍이 나열됩니다.

4.11. 풀 값 가져오기

풀에서 값을 가져오려면 다음 명령을 실행합니다.

```
ceph osd pool get <pool-name> <key>
```

풀 값 섹션에는 가져올 수 있는 모든 키-값 쌍이 나열됩니다.

4.12. 애플리케이션 활성화

***Red Hat Ceph Storage**는 인증되지 않은 유형의 클라이언트가 풀에 데이터를 작성하지 못하도록 풀에 대한 추가 보호 기능을 제공합니다. 즉, 시스템 관리자가 **Ceph** 블록 장치, **Ceph Object Gateway**, **Ceph Filesystem** 또는 사용자 지정 애플리케이션에서 풀을 **I/O** 작업을 수신하도록 명시적으로 활성화해야 합니다.*

*클라이언트 애플리케이션이 풀에서 **I/O** 작업을 수행할 수 있도록 하려면 다음을 실행합니다.*

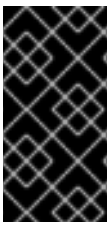
```
[root@host ~]# ceph osd pool application enable <poolname> <app> {--yes-i-really-mean-it}
```

*여기서 **&lt ;app>** 은 다음과 같습니다.*

- ***Ceph Filesystem** 의 경우 **CephFS**입니다.*
- ***Ceph** 블록 장치의 경우 **RBD***
- ***Ceph** 개체 게이트웨이용 **RGW***

참고

*사용자 지정 애플리케이션에 다른 **<app >** 값을 지정합니다.*



중요

*활성화되지 않은 풀은 **HEALTH_WARN** 상태를 생성합니다. 이 시나리오에서는 **ceph** 상태 세부 **-f json-pretty** 의 출력은 다음을 출력합니다.*

```
{
    "checks": {
        "POOL_APP_NOT_ENABLED": {
```

```
            "severity": "HEALTH_WARN",
            "summary": {
                "message": "application not enabled on 1 pool(s)"
            },
            "detail": [
                {
                    "message": "application not enabled on pool '<pool-name>'"
                },
                {
                    "message": "use 'ceph osd pool application enable <pool-name> <app-name>', where
<app-name> is 'cephfs', 'rbd', 'rgw', or freeform for custom applications."
                }
            ]
        }
    },
    "status": "HEALTH_WARN",
    "overall_status": "HEALTH_WARN",
    "detail": [
        "'ceph health' JSON format has changed in luminous. If you see this your monitoring system is
scraping the wrong fields. Disable this with 'mon health preluminous compat warning = false'"
    ]
}
```

참고

***Ceph Block Device*의 풀을 *rbd* 풀 *init <pool-name>*으로 초기화합니다.**

4.13. 애플리케이션 비활성화

클라이언트 애플리케이션이 풀의 I/O 작업을 수행하는 것을 비활성화하려면 다음을 실행합니다.

```
[root@host ~]# ceph osd pool application disable <poolname> <app> {--yes-i-really-mean-it}
```

*여기서 **&lt ;app>** 은 다음과 같습니다.*

- ***Ceph Filesystem** 의 경우 **CephFS**입니다.*
- ***Ceph** 블록 장치의 경우 **RBD***
- ***Ceph** 개체 게이트웨이용 **RGW***

참고

사용자 지정 애플리케이션에 다른 **<app >** 값을 지정합니다.

4.14. 애플리케이션 메타데이터 설정

클라이언트 애플리케이션의 특성을 설명하는 키-값 쌍을 설정하는 기능을 제공합니다.

풀에 클라이언트 애플리케이션 메타데이터를 설정하려면 다음을 실행합니다.

```
[root@host ~]# ceph osd pool application set <poolname> <app> <key> <value>
```

여기서 **&lt ;app>** 은 다음과 같습니다.

- **Ceph Filesystem** 의 경우 **CephFS**입니다.
- **Ceph** 블록 장치의 경우 **RBD**
- **Ceph** 개체 게이트웨이용 **RGW**

참고

사용자 지정 애플리케이션에 다른 **<app >** 값을 지정합니다.

4.15. 애플리케이션 메타데이터 제거

풀에서 클라이언트 애플리케이션 메타데이터를 제거하려면 다음을 실행합니다.

```
[root@host ~]# ceph osd pool application set <poolname> <app> <key>
```

여기서 **&lt ;app>** 은 다음과 같습니다.

- ***Ceph Filesystem** 의 경우 **CephFS**입니다.*
- ***Ceph** 블록 장치의 경우 **RBD***
- ***Ceph** 개체 게이트웨이용 **RGW***

참고

*사용자 지정 애플리케이션에 다른 **<app >** 값을 지정합니다.*

4.16. 오브젝트 복제본 수 설정

복제된 풀에서 오브젝트 복제본 수를 설정하려면 다음 명령을 실행합니다.

```
ceph osd pool set <poolname> size <num-replicas>
```

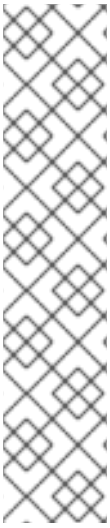
중요

***& lt;num-replicas& gt;** 매개변수에는 개체 자체가 포함됩니다. 오브젝트와 오브젝트의 총 **3**개 인스턴스에 대해 오브젝트 복사본 두 개를 포함하려면 **3** 을 지정합니다.*

예를 들어 다음과 같습니다.

```
ceph osd pool set data size 3
```

각 풀에 대해 이 명령을 실행할 수 있습니다.



참고

오브젝트는 풀 크기 설정에 지정된 것보다 적은 수의 복제본으로 성능이 저하된 모드에서 I/O 작업을 허용할 수 있습니다. I/O에 필요한 최소 복제본 수를 설정하려면 min_size 설정을 사용합니다. 예를 들어 다음과 같습니다.

```
ceph osd pool set data min_size 2
```

이렇게 하면 데이터 풀의 오브젝트가 min_size 설정에 지정된 것보다 적은 수의 복제본으로 I/O가 제공되지 않습니다.

4.17. 오브젝트 복제본 수 가져오기

오브젝트 복제본 수를 가져오려면 다음 명령을 실행합니다.

```
ceph osd dump | grep 'replicated size'
```

Ceph는 복제된 크기 속성이 강조 표시된 풀을 나열합니다. 기본적으로 Ceph는 총 3개 복사본 또는 3개의 크기인 오브젝트의 두 개의 복제본을 만듭니다.

4.18. 풀 값

다음 목록에는 설정하거나 가져올 수 있는 키-값 쌍이 나와 있습니다. 자세한 내용은 풀 값 설정 및 풀 값 가져오기 섹션을 참조하십시오.

크기

설명

풀에 있는 개체의 복제본 수를 지정합니다. 자세한 내용은 Object Replicas 설정 섹션을 참조하십시오. 복제된 풀에만 적용됩니다.

유형

정수

min_size

설명

I/O에 필요한 최소 복제본 수를 지정합니다. 자세한 내용은 Object Replicas 설정 섹션을 참조하십시오. 복제된 풀에만 적용됩니다.

유형

정수

crash_replay_interval

설명

인식되지 않았지만 커밋되지 않은 요청을 클라이언트가 재생할 수 있도록 허용하는 시간(초)을 지정합니다.

유형

정수

pg-num

설명

풀의 총 배치 그룹 수입니다. 적합한 수 계산에 대한 자세한 내용은 ***Red Hat Ceph Storage 5*** *구성 가이드의* ***Pool, placement groups, CRUSH ConfigurationReference*** *섹션을 참조하십시오. 기본값은 대부분의 시스템에 적합하지 않습니다.*

유형

정수

필수 항목

네, 필요합니다.

기본값

8

pgp-num

설명

배치 목적으로 총 배치 그룹 수입니다. 배치 그룹 분할 시나리오 를 제외하고 총 배치 그룹 수와 같아야 합니다.

유형

정수

필수 항목

*네, 필요합니다. 지정되지 않은 경우 기본값 또는 **Ceph** 구성 값을 선택합니다.*

기본값

8

유효한 범위

***pg_num** 변수에 의해 지정된 것과 같거나 작습니다.*

crush_rule

설명

클러스터에서 오브젝트 배치를 매핑하는 데 사용할 규칙입니다.

유형

문자열

hashpspool

설명

*지정된 풀에서 **HASHPSPOOL** 플래그를 활성화하거나 비활성화합니다. 이 옵션을 사용하면 풀 및 배치 그룹이 겹치는 방식을 개선하기 위해 풀 해시 및 배치 그룹 매핑이 변경됩니다.*

유형

정수

유효한 범위

***1** 플래그를 활성화하면 **0** 은 플래그를 비활성화합니다.*

중요

*많은 양의 **OSD** 및 데이터가 있는 클러스터의 프로덕션 풀에서 이 옵션을 활성화하지 마십시오. 풀의 모든 배치 그룹을 다시 매핑해야 하므로 너무 많은 데이터 이동이 발생합니다.*

fast_read

설명

*삭제 코딩을 사용하는 풀에서 이 플래그가 활성화된 경우 읽기 요청 문제가 나중에 모든 **shard**에 읽고, 클라이언트를 제공하기에 충분한 **shard**가 수신될 때까지 기다립니다. 재라이저 및 **isa** 소거 플러그인의 경우, 첫 번째 **K** 응답이 반환되면 클라이언트의 요청은 이러한 응답에서 디코딩된 데이터를 사용하여 즉시 제공됩니다. 이를 통해 성능 향상을 위해 일부 리소스를 할당할 수 있습니다. 현재 이 플래그는 기존 코딩 풀에 대해서만 지원됩니다.*

유형

부울

기본값

0

allow_ec_overwrites

설명

*복구 코딩된 풀에 쓰기가 개체의 일부를 업데이트할 수 있으므로 **Ceph Filesystem**과 **Ceph** 블록 장치가 사용할 수 있습니다.*

유형

부울

compression_algorithm

설명

***BlueStore** 스토리지 백엔드에 사용하도록 인라인 압축 알고리즘을 설정합니다. 이 설정은 **bluestore_compression_algorithm** 구성 설정을 덮어씁니다.*

유형

문자열

유효한 설정

lz4, snappy, zlib, zstd

compression_mode

설명

***BlueStore** 스토리지 백엔드의 인라인 압축 알고리즘에 대한 정책을 설정합니다. 이 설정은 **bluestore_compression_mode** 구성 설정을 덮어씁니다.*

유형

문자열

유효한 설정

없음,수동적,공격적,힘

compression_min_blob_size

설명

***bluestore**는 이 크기보다 작은 청크를 압축하지 않습니다. 이 설정은 **bluestore_compression_min_blob_size** 구성 설정을 덮어씁니다.*

유형

서명되지 않은 정수

compression_max_blob_size

설명

***bluestore**는 데이터를 압축하기 전에 이 크기보다 큰 청크를 **compression_max_blob_size** 의 작은 **Blob**으로 분할합니다.*

유형

서명되지 않은 정수

nodelete

설명

*지정된 풀에 **NODELETE** 플래그를 설정하거나 설정 해제합니다.*

유형

정수

유효한 범위

***1** 세트 플래그입니다. **0 unsets** 플래그입니다.*

nopgchange

설명

*지정된 풀에 **NOPGCHANGE** 플래그를 설정하거나 설정 해제합니다.*

유형

정수

유효한 범위

***1** 플래그를 설정합니다. **0** 은 플래그를 설정 해제합니다.*

nosizechange

설명

*지정된 풀에 **NOSIZECHANGE** 플래그를 설정하거나 설정 해제합니다.*

유형

정수

유효한 범위

***1** 플래그를 설정합니다. **0** 은 플래그를 설정 해제합니다.*

write_fadvise_dontneed

설명

*지정된 풀에 **WRITE_FADVISE_DONTNEED** 플래그를 설정하거나 설정 해제합니다.*

유형

정수

유효한 범위

***1** 플래그를 설정합니다. **0** 은 플래그를 설정 해제합니다.*

noscrub

설명

*지정된 풀에 **NOSCRUB** 플래그를 설정하거나 설정 해제합니다.*

유형

정수

유효한 범위

1 플래그를 설정합니다. 0 은 플래그를 설정 해제합니다.

nodeep-scrub

설명

*지정된 풀에서 **NODEEP_SCRUB** 플래그를 설정하거나 설정 해제합니다.*

유형

정수

유효한 범위

1 플래그를 설정합니다. 0 은 플래그를 설정 해제합니다.

scrub_min_interval

설명

*로드가 낮을 때 풀 스크럽의 최소 간격(초)입니다. 이 값이 **0** 인 경우 **Ceph**는 **osd_scrub_min_interval** 구성 설정을 사용합니다.*

유형

double

기본값

0

scrub_max_interval

설명

*클러스터 로드에 관계없이 풀의 최대 간격(초)입니다. 이 값이 **0** 인 경우 **Ceph**는 **osd_scrub_max_interval** 구성 설정을 사용합니다.*

유형

double

기본값

0

deep_scrub_interval

설명

'deep' *스크럽 풀의 간격(초)입니다. 이 값이* **0** *인 경우* **Ceph**는 **osd_deep_scrub_interval** *구성 설정을 사용합니다.*

유형

double

기본값

0

5장. 코드 풀 삭제

Ceph 스토리지 전략에는 데이터 내구성 요구 사항을 정의하는 작업이 포함됩니다. 데이터 내구성은 데이터를 손실하지 않고 하나 이상의 OSD 손실을 유지할 수 있는 기능을 의미합니다.

Ceph는 풀에 데이터를 저장하고 풀 유형은 다음 두 가지입니다.

- replicated
- erasure-coded

Ceph는 기본적으로 복제된 풀을 사용합니다. 즉, Ceph는 기본 OSD 노드의 모든 오브젝트를 하나 이상의 보조 OSD로 복사합니다.

크기가 조정된 풀은 데이터 내구성을 보장하는 데 필요한 디스크 공간을 줄일 수 있지만 복제보다 비용이 더 많이 듭니다.

이레이저 코딩은 Ceph 스토리지 클러스터에 오브젝트를 저장하는 방법으로, 삭제 코드 알고리즘이 오브젝트를 데이터 청크(k) 및 코딩 청크(m)로 분할하고 이러한 청크를 다른 OSD에 저장합니다.

OSD가 실패하는 경우 Ceph는 다른 OSD에서 나머지 데이터(k) 및 코딩(m) 청크를 검색하고 삭제 코드 알고리즘은 해당 청크에서 오브젝트를 복원합니다.

참고

Red Hat은 쓰기 및 데이터 손실을 방지하기 위해 K+2 이상으로 비해지도록 min_size를 권장합니다.

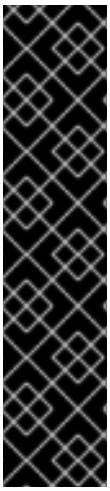
복제보다 스토리지 용량을 보다 효율적으로 축소합니다. n-replication 방식은 Ceph에서 기본적으로 (3x) 개체의 n 복사본을 유지 관리하는 반면, 코딩은 k + m 청크만 유지 관리합니다. 예를 들어 3개의 데이터와 2개의 코딩 청크는 원래 개체의 스토리지 공간을 1.5x를 사용합니다.

이레이저 코딩은 복제보다 적은 스토리지 오버헤드를 사용하지만 삭제 코드 알고리즘은 개체에 액세스하거나 복구할 때 복제보다 더 많은 RAM과 CPU를 사용합니다. 데이터 스토리지에 내구성 및 내결함성이

있어야 하지만 빠른 읽기 성능(예: 콜드 스토리지, 기록 레코드 등)이 필요하지 않은 경우 삭제 코딩이 유용합니다.

Ceph에서 코드가 어떻게 작동하는지에 대한 수치적이고 자세한 설명은 Red Hat Ceph Storage 5의 아키텍처 가이드에서 Ceph Erasure Coding 섹션을 참조하십시오.

Ceph는 k=2 및 m=1 으로 클러스터를 초기화할 때 기본 삭제 코드 프로필을 생성합니다. 즉, Ceph가 3 개의 OSD(k+m == 3)에 오브젝트 데이터를 분배하고 Ceph는 데이터를 손실하지 않고 OSD 중 하나를 유실할 수 있습니다. 버전 코드 프로파일링에 대한 자세한 내용은 Erasure Code Profiles 섹션을 참조하십시오.



중요

.rgw.buckets 풀만 Brsure-coded 및 기타 모든 Ceph Object Gateway 풀 복제로 구성합니다. 그러지 않으면 다음 오류와 함께 새 버킷을 생성하려고 실패합니다.

```
set_req_state_err err_no=95 resorting to 500
```

그 이유는 축소된 풀이 omap 작업을 지원하지 않으며 특정 Ceph Object Gateway 메타데이터 풀에는 omap 지원이 필요하기 때문입니다.

5.1. 샘플 ERASURE-CODED POOL 생성

가장 간단한 삭제 코드 풀은 RAID5와 동일하며 최소 3개의 호스트가 필요합니다.

```
$ ceph osd pool create ecpool 50 50 erasure
pool 'ecpool' created
$ echo ABCDEFGHI | rados --pool ecpool put NYAN -
$ rados --pool ecpool get NYAN -
ABCDEFGHI
```

참고

50 풀 생성 은 배치 그룹 수를 나타냅니다.

5.2. 코드 프로필 삭제

Ceph는 프로필이 있는 삭제 코드된 풀 을 정의합니다. Ceph는 인출 코드된 풀 및 연결된 CRUSH 규칙을 생성할 때 프로필을 사용합니다.

Ceph는 클러스터를 초기화할 때 기본 삭제 코드 프로필을 생성하고 복제된 풀에서 두 개의 사본과 동일한 수준의 중복을 제공합니다. 그러나 스토리지 용량이 25% 더 적게 사용됩니다. 기본 프로필은 k=2 및 m=1 을 정의합니다. 즉, Ceph는 3개의 OSD(k+m=3)에 오브젝트 데이터를 분배하고, Ceph는 데이터를 손실하지 않고 OSD 중 하나를 손실될 수 있습니다.

기본 삭제 코드 프로필은 단일 OSD의 손실을 유지할 수 있습니다. 이는 크기가 2TB인 복제된 풀과 동일하지만 1TB의 데이터를 저장하는 데 1.5TB가 아닌 1.5TB가 필요합니다. 기본 프로필을 표시하려면 다음 명령을 사용합니다.

```
$ ceph osd erasure-code-profile get default
directory=.libs
k=2
m=1
plugin=jerasure
crush-failure-domain=host
technique=reed_sol_van
```

원시 스토리지 요구 사항을 늘리지 않고 중복성을 개선하기 위해 새 프로필을 생성할 수 있습니다. 예를 들어, k=8 및 m=4 인 프로필은 12개(k+m=12) OSD에서 오브젝트를 배포하여 4개의 (m=4) OSD의 손실을 유지할 수 있습니다. Ceph는 개체를 8 개의 청크로 분할하고 복구할 수 있도록 4 개의 코딩 청크를 계산합니다. 예를 들어 개체 크기가 8MB인 경우 각 데이터 청크는 1MB이고 각 코딩 청크의 크기는 데이터 청크와 크기가 1MB입니다. OSD 4개가 동시에 실패하더라도 오브젝트는 손실되지 않습니다.

프로필의 가장 중요한 매개 변수는 스토리지 오버헤드와 데이터 내구성을 정의하기 때문에 k,m 및 crush-failure-domain 입니다.

중요

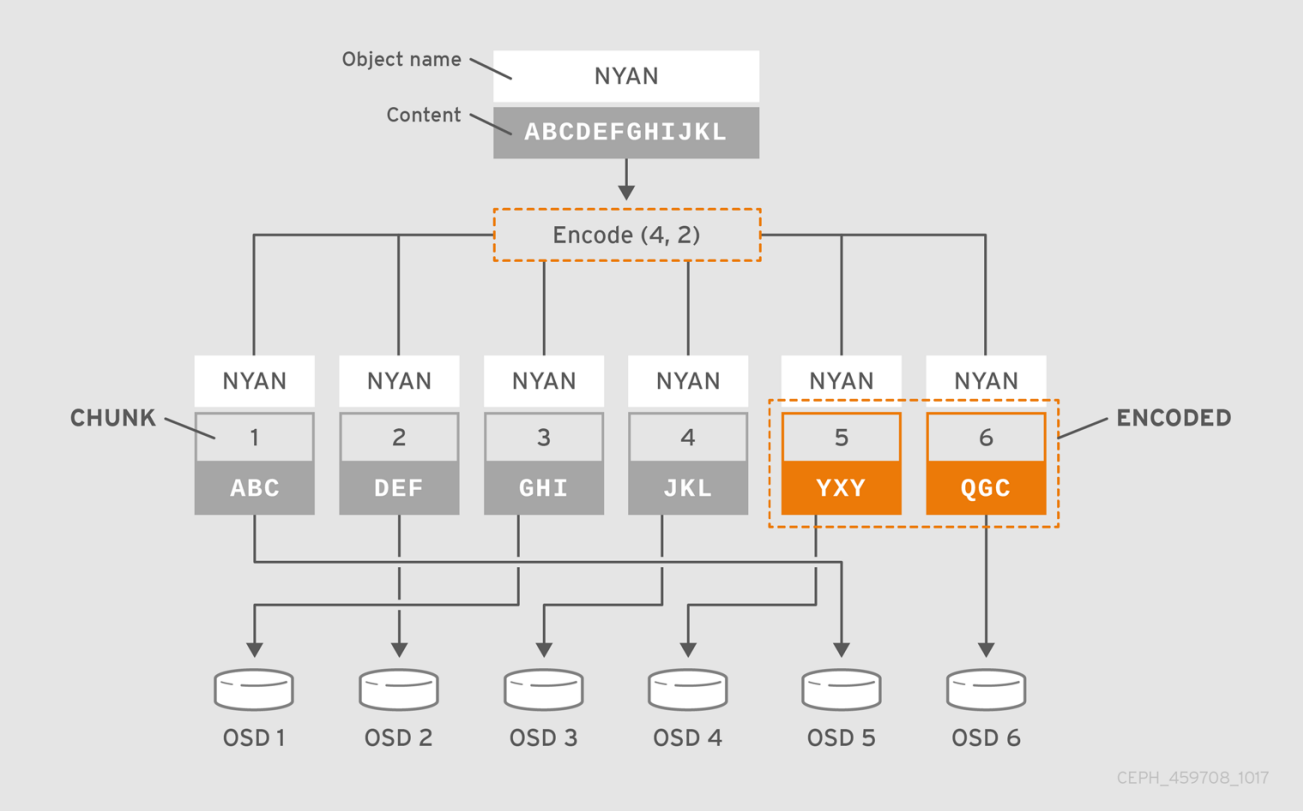
풀을 생성한 후에는 프로필을 변경할 수 없으므로 올바른 프로필을 선택하는 것이 중요합니다. 프로필을 수정하려면 다른 프로필을 사용하여 새 풀을 생성하고 이전 풀에서 새 풀로 오브젝트를 마이그레이션해야 합니다.

예를 들어 원하는 아키텍처에서 스토리지 오버헤드가 40% 오버헤드인 두 개의 랙의 손실을 유지해야 하는 경우 다음 프로필을 정의할 수 있습니다.

```
$ ceph osd erasure-code-profile set myprofile \
   k=4 \
   m=2 \
   crush-failure-domain=rack
$ ceph osd pool create ecpool 12 12 erasure *myprofile*
```

```
$ echo ABCDEFGHIJKL | rados --pool ecpool put NYAN -
$ rados --pool ecpool get NYAN -
ABCDEFGHIJKL
```

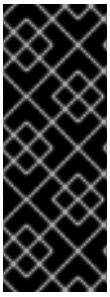
*기본 **OSD**는 **NYAN** 오브젝트를 **4**개의 **(k=4)** 데이터 청크로 나누고 두 개의 추가 청크**(m=2)**를 만듭니다. **m** 값은 데이터를 손실하지 않고도 동시에 손실될 수 있는 **OSD** 수를 정의합니다. **crush-failure-domain=rack** 은 두 개의 청크가 동일한 랙에 저장되지 않도록 **CRUSH** 규칙을 생성합니다.*



중요

***Red Hat**은 **k** 및 **m** 에 대한 다음과 같은 제조형 코딩 값을 지원합니다.*

- ***k=8 m=3***
- ***k=8 m=4***
- ***k=4 m=2***



중요

OSD 수가 코딩 청크(**m**) 수와 동일한 경우 삭제 코딩 풀의 일부 배치 그룹은 불완전한 상태가 됩니다. 손실된 **OSD** 수가 **m** 보다 작으면 배치 그룹이 불완전한 상태가 되지 않습니다. 어떠한 경우에도 데이터 손실이 발생하지 않습니다. 배치 그룹이 불완전한 상태인 경우 복원된 풀의 **min_size** 를 임시로 줄이면 복구가 가능합니다.

5.2.1. OSD deletedsure-code-profile Set

새 삭제 코드 프로필을 생성하려면 다음을 수행합니다.

```
ceph osd erasure-code-profile set <name> \
     [<directory=directory>] \
     [<plugin=plugin>] \
     [<stripe_unit=stripe_unit>] \
     [<crush-device-class>]\
     [<crush-failure-domain>]\
     [<key=value> ...] \
     [--force]
```

다음과 같습니다.

디렉터리

설명

삭제 코드 플러그인이 로드된 디렉터리 이름을 설정합니다.

유형

문자열

필수 항목

아니요.

기본값

/usr/lib/ceph/erasure-code

plugin

설명

삭제 코드 플러그인을 사용하여 코딩 청크를 계산하고 누락된 청크를 복구합니다. 자세한 내용은 **Erasure Code Plug-ins** 섹션을 참조하십시오.

유형

문자열

필수 항목

아니요.

기본값

jerasure

stripe_unit

설명

*스트라이프당 데이터 청크의 데이터 양입니다. 예를 들어 **2**개의 데이터 청크와 스트라이프_unit=**4K** 가 있는 프로필은 **0-4K** 범위를 청크 **0**, **4K-8K**로 청크 **1**에 배치한 다음 청크 **0**에 **8K-12K**를 다시 설정합니다. 최상의 성능을 위해 **4K** 중 여러 개여야 합니다. 풀이 생성되면 모니터 구성 옵션 **osd_pool_erasure_code_stripe_unit** 에서 기본값을 가져옵니다. 이 프로필을 사용하는 풀의 스트라이프_**width**는 이 스트라이프_**unit** 을 곱한 데이터 청크 수입니다.*

유형

문자열

필수 항목

아니요.

기본값

4K

crush-device-class

설명

***hdd** 또는 **ssd** 와 같은 장치 클래스입니다.*

유형

문자열

필수 항목

없음

기본값

*아니요, **CRUSH**는 클래스에 관계없이 모든 장치를 사용합니다.*

crush-failure-domain

설명

호스트 또는 랙 과 같은 실패 도메인.

유형

문자열

필수 항목

없음

기본값

host

key

설명

나머지 키-값 쌍의 의미는 삭제 코드 플러그인으로 정의됩니다.

유형

문자열

필수 항목

아니요.

--force

설명

동일한 이름으로 기존 프로필을 재정의합니다.

유형

문자열

필수 항목

아니요.

5.2.2. OSD 삭제-code-profile Remove

삭제 코드 프로파일을 제거하려면 다음을 수행합니다.

```
ceph osd erasure-code-profile rm <name>
```

풀에서 프로필을 참조하는 경우 삭제가 실패합니다.

5.2.3. OSD erasure-code-profile Get

삭제 코드 프로필을 표시하려면 다음을 수행합니다.

```
ceph osd erasure-code-profile get <name>
```

5.2.4. OSD monthssure-code-profile 목록

모든 삭제 코드 프로필의 이름을 나열하려면 다음을 수행합니다.

```
ceph osd erasure-code-profile ls
```

5.3. OVERWRITES를 사용한 삭제

*기본적으로 코딩된 풀은 전체 개체 쓰기 및 추가 작업을 수행하는 **Ceph Object Gateway**에서만 작동합니다. **Red Hat Ceph Storage 3.x**부터 풀당 코딩된 풀에 대한 부분적인 쓰기를 활성화할 수 있습니다.*

*덮어쓰기와 함께 삭제 코딩된 풀을 사용하면 **Ceph** 블록 장치 및 **CephFS**가 데이터를 삭제 코딩된 풀에 저장할 수 있습니다.*

구문

```
ceph osd pool set <erasure_coded_pool_name> allow_ec_overwrites true
```

예제

```
$ ceph osd pool set ec_pool allow_ec_overwrites true
```

*덮어쓰기를 사용하여 삭제 코딩된 풀을 활성화하면 **BlueStore OSD**를 사용하는 풀에만 존재할 수 있습니다. **BlueStore**의 체크섬은 깊은 스크럽 중에 비트 회전 또는 기타 손상을 감지하는 데 사용됩니다. 삭제 코드 덮어쓰기와 함께 **FileStore**를 사용하는 것은 안전하지 않으며 **BlueStore**와 비교했을 때 성능이 저하됩니다.*

*복원된 풀은 **omap**을 지원하지 않습니다. **Ceph** 블록 장치 및 **CephFS**와 함께 삭제 코딩된 풀을 사용하려면 데이터를 삭제 코딩된 풀에 저장하고 메타데이터를 복제된 풀에 저장합니다.*

*__Ceph__ 블록 장치의 경우 이미지 생성 중에 **--data-pool** 옵션을 사용합니다.*

구문

```
rbd create --size <image_size>M|G|T --data-pool <erasure_coded_pool_name>
<replicated_pool_name>/<image_name>
```

예제

```
$ rbd create --size 1G --data-pool ec_pool rep_pool/image01
```

CephFS에 삭제 코딩된 풀을 사용하는 경우 덮어쓰기를 설정하여 파일 레이아웃을 수행해야 합니다.

5.4. 버전 코드 플러그인 삭제

Ceph는 플러그인 아키텍처를 사용하여 삭제 코딩을 지원하므로 다양한 유형의 알고리즘을 사용하여 삭제 코딩된 풀을 생성할 수 있습니다. Ceph 지원:

- Jerasure (Default)
- 로컬 복구 가능
- ISA (Intel만 해당)

다음 섹션에서는 이러한 플러그인에 대해 자세히 설명합니다.

5.4.1. Jerasure Erasure Code 플러그인

jerasure 플러그인은 가장 일반적이고 유연한 플러그인입니다. 또한 Ceph 삭제 코드된 풀의 기본값입니다.

jerasure 플러그인은 JerasureH 라이브러리를 캡슐화합니다. 매개변수에 대한 자세한 내용은 jerasure 문서를 참조하십시오.

jerasure 플러그인을 사용하여 새 삭제 코드 프로필을 생성하려면 다음 명령을 실행합니다.

```
ceph osd erasure-code-profile set <name> \
    plugin=jerasure \
    k=<data-chunks> \
    m=<coding-chunks> \
    technique=
<reed_sol_van|reed_sol_r6_op|cauchy_orig|cauchy_good|liberation|blaum_roth|liber8tion> \
    [crush-root=<root>] \
```

```
[crush-failure-domain=<bucket-type>] \
[directory=<directory>] \
[--force]
```

다음과 같습니다.

k

설명

각 오브젝트는 데이터크크 부분으로 분할 되며 각각 다른 **OSD**에 저장됩니다.

유형

정수

필수 항목

네, 필요합니다.

예제

4

m

설명

각 오브젝트의 청크 를 계산하여 서로 다른 **OSD**에 저장합니다. 데이터 손실 없이 축소할 수 있는 **OSD** 수입니다.

유형

정수

필수 항목

네, 필요합니다.

예제

2

technique

설명

보다 유연한 기술은 **reed_sol_van** 입니다. **k** 와 **m** 을 설정하는 것으로 충분합니다.

cauchy_good 기술은 더 빠를 수 있지만 패킷을 신중하게 선택해야 합니다. 모든 reed_sol_r6_op,liberation,blaum_roth,liber8tion 은 m=2 로만 구성할 수 있다는 점에서 RAID6 과 동등합니다.

유형

문자열

필수 항목

아니요.

유효한 설정

reed_sol_van reed_sol_r6 _op cauchy _good liberation blaum_roth liber8tion

기본값

reed_sol_van

packetSize

설명

인코딩은 한 번에 바이트 크기 패킷에서 수행됩니다. 올바른 패킷 크기를 선택하는 것은 어렵습니다. jerasure 문서에는 이 항목에 대한 자세한 정보가 포함되어 있습니다.

유형

정수

필수 항목

아니요.

기본값

2048

crush-root

설명

규칙의 첫 번째 단계에 사용된 CRUSH 버킷의 이름입니다. 예를 들어, 단계가 기본값을 사용합니다.

유형

문자열

필수 항목

아니요.

기본값

default

crush-failure-domain

설명

*두 청크가 동일한 실패 도메인이 있는 버킷에 없는지 확인합니다. 예를 들어, 실패 도메인이 호스트 되면 두 개의 청크가 동일한 호스트에 저장되지 않습니다. 스텝(**Fasper host**)을 선택하는 것과 같은 규칙 단계를 만드는 데 사용됩니다.*

유형

문자열

필수 항목

아니요.

기본값

host

디렉터리

설명

삭제 코드 플러그인이 로드된 디렉터리 이름을 설정합니다.

유형

문자열

필수 항목

아니요.

기본값

/usr/lib/ceph/erasure-code

--force

설명

동일한 이름으로 기존 프로필을 재정의합니다.

유형

문자열

필수 항목

아니요.

5.4.2. 로컬에서 복구 가능한 Erasure Code (LRC) 플러그인

***jerasure** 플러그인을 사용하면 **Ceph**가 여러 **OSD**에 삭제 코딩된 오브젝트를 저장할 때 하나의 **OSD** 손실에서 복구하려면 다른 모든 **OSD**에서 읽기가 필요합니다. 예를 들어 **k=8** 및 **m=4** 로 자조를 구성하는 경우 하나의 **OSD**를 손실하려면 다른 **OSD**를 복구하려면 다른 **OSD**를 읽는 것이 필요합니다.*

***lrc** 삭제 코드 플러그인은 더 적은 **OSD**를 사용하여 복구할 수 있도록 로컬 패리티 청크를 생성합니다. 예를 들어 **k=8,m=4** 및 **l=4** 를 사용하여 **lrc** 를 구성하는 경우 **4**개의 **OSD**마다 추가 패리티 청크를 생성합니다. **Ceph**가 단일 **OSD**를 손실하면 오브젝트 데이터를 **OSD 4**개로만 복구할 수 있습니다.*

*모든 호스트가 동일한 스위치에 연결된 경우 흥미로운 사용 사례는 아니지만, **lrc** 규모**sure** 코드 플러그인을 사용할 때 랙 간의 대역폭 사용량 감소를 실제로 관찰할 수 있습니다.*

```
$ ceph osd erasure-code-profile set LRCprofile \
    plugin=lrc \
    k=4 m=2 l=3 \
    crush-failure-domain=host
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

***1.2** 버전에서는 기본 **OSD**가 손실된 청크와 동일한 랙에 있는 경우에만 감소된 대역폭을 관찰할 수 있습니다.*

```
$ ceph osd erasure-code-profile set LRCprofile \
    plugin=lrc \
    k=4 m=2 l=3 \
```

```
    crush-locality=rack \
    crush-failure-domain=host
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

5.4.2.1. LRC 프로필 생성

새 LRC 삭제 코드 프로필을 생성하려면 다음 명령을 실행합니다.

```
ceph osd erasure-code-profile set <name> \
    plugin=lrc \
    k=<data-chunks> \
    m=<coding-chunks> \
    l=<locality> \
    [crush-root=<root>] \
    [crush-locality=<bucket-type>] \
    [crush-failure-domain=<bucket-type>] \
    [directory=<directory>] \
    [--force]
```

다음과 같습니다.

k

설명

각 오브젝트는 데이터크크 부분으로 분할 되며 각각 다른 OSD에 저장됩니다.

유형

정수

필수 항목

네, 필요합니다.

예제

4

m

설명

각 오브젝트의 청크 를 계산하여 서로 다른 OSD에 저장합니다. 데이터 손실 없이 축소할 수 있는 OSD 수입니다.

유형

정수

필수 항목

네, 필요합니다.

예제

2

l

설명

*코딩 및 데이터 청크를 크기 지역성 세트로 그룹화합니다. 예를 들어, **k=4** 및 **m=2** 의 경우 **locality=3** **3**개의 그룹 **3**개가 생성되는 경우입니다. 각 세트는 다른 세트의 청크를 읽지 않고 복구할 수 있습니다.*

유형

정수

필수 항목

네, 필요합니다.

예제

3

crush-root

설명

*규칙의 첫 번째 단계에 사용되는 **crush** 버킷의 이름입니다. 예를 들어, 단계가 기본값을 사용합니다.*

유형

문자열

필수 항목

아니요.

기본값

default

crush-locality

설명

I 에 의해 정의된 각 청크 세트가 저장되는 크러쉬 버킷의 유형입니다. 예를 들어 랙 으로 설정된 경우 각 I 청크 그룹은 다른 랙에 배치됩니다. 이 명령은 랙을 선택하는 단계와 같은 규칙 단계를 생성하는 데 사용됩니다. 설정하지 않으면 이러한 그룹화가 수행되지 않습니다.

유형

문자열

필수 항목

아니요.

crush-failure-domain

설명

두 청크가 동일한 실패 도메인이 있는 버킷에 없는지 확인합니다. 예를 들어, 실패 도메인이 호스트 되면 두 개의 청크가 동일한 호스트에 저장되지 않습니다. 스텝**(Fasper host)**을 선택하는 것과 같은 규칙 단계를 만드는 데 사용됩니다.

유형

문자열

필수 항목

아니요.

기본값

host

디렉터리

설명

삭제 코드 플러그인이 로드된 디렉터리 이름을 설정합니다.

유형

문자열

필수 항목

아니요.

기본값

/usr/lib/ceph/erasure-code

--force

설명

동일한 이름으로 기존 프로필을 재정의합니다.

유형

문자열

필수 항목

아니요.

5.4.2.2. 낮은 수준의 LRC 프로파일을 생성

k *및* ***m*** *의 합계는* ***l*** *매개 변수의 곱이어야 합니다. 하위 수준 구성 매개변수는 이러한 제한을 적용하지 않으며 특정 용도로 사용하는 것이 더 편리할 수 있습니다. 예를 들어* ***4****개의 청크가 있고 다른 그룹에는* ***3*** *개의 청크가 있는 그룹을 정의할 수 있습니다. 예를 들어 데이터 센터 및 랙을 데이터 센터로 반복적으로 정의하는 경우도 있습니다.* ***k/m/l*** *은 낮은 수준의 구성을 생성하여 구현됩니다.*

lrc *삭제 코드 플러그인은 반복적으로 일부 청크 손실에서 복구할 수 있도록 삭제 코드 기술을 적용하여 대부분의 경우 사용 가능한 청크의 서브 세트만 필요합니다.*

예를 들어 세 개의 코딩 단계를 다음과 같이 설명할 때 다음과 같습니다.

```
chunk nr    01234567
step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

c *는 데이터 청크* ***D*** *에서 계산된 코딩 청크이며 청크* ***7*** *의 손실을 마지막* ***4****개의 청크로 복구할 수 있습니다. 그리고 청크* ***2*** *손실은 처음* ***4****개의 청크로 복구될 수 있습니다.*

*최소 테스트 시나리오는 기본 **erasure-code** 프로필을 사용하는 것과 엄격하게 동일합니다. **DD** 는 **K=2** 를 의미합니다. **c** 는 **M=1** 을 의미하며 기본적으로 **jerasure** 플러그인을 사용합니다.*

```
$ ceph osd erasure-code-profile set LRCprofile \
    plugin=lrc \
    mapping=DD_ \
    layers='[ [ "DDc", "" ] ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

***lrc** 플러그인은 교차 대역폭 사용량을 줄이는 데 특히 유용합니다. 모든 호스트가 동일한 스위치에 연결된 경우 흥미로운 사용 사례는 아니지만 대역폭 사용량을 실제로 관찰할 수 있습니다. 청크 레이아웃이 서로 다르지만 **k=4,m=2** 및 **l=3** 과 동일합니다.*

```
$ ceph osd erasure-code-profile set LRCprofile \
    plugin=lrc \
    mapping=__DD__DD \
    layers='[
            [ "_cDD_cDD", "" ],
            [ "cDDD____", "" ],
            [ "____cDDD", "" ],
           ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

***Firefly**에서 감소된 대역폭은 기본 **OSD**가 손실된 청크와 동일한 랙에 있는 경우에만 관찰됩니다.*

```
$ ceph osd erasure-code-profile set LRCprofile \
    plugin=lrc \
    mapping=__DD__DD \
    layers='[
            [ "_cDD_cDD", "" ],
            [ "cDDD____", "" ],
            [ "____cDDD", "" ],
           ]' \
    crush-steps='[
                 [ "choose", "rack", 2 ],
                 [ "chooseleaf", "host", 4 ],
                ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

*이제 **LRC**가 기본 **EC** 백엔드로 **jerasure** 를 사용합니다. 하위 수준 구성을 사용하여 계층별로 **EC** 백엔드와 알고리즘을 지정할 수 있습니다. **layer='["DDc", ""]**의 두 번째 인수는 실제로 이 수준에 사용할 삭제 코드 프로필입니다. 아래 예제에서는 **lrcpool**에서 사용할 **Cauchy** 기술을 사용하여**ISA** 백엔드를 지정합니다.*

```
$ ceph osd erasure-code-profile set LRCprofile \
    plugin=lrc \
    mapping=DD_ \
```

```
    layers='[ [ "DDc", "plugin=isa technique=cauchy" ] ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

각 계층에 대해 다른 삭제 코드 프로필을 사용할 수도 있습니다.

```
$ ceph osd erasure-code-profile set LRCprofile \
    plugin=lrc \
    mapping=__DD__DD \
    layers='[
            [ "_cDD_cDD", "plugin=isa technique=cauchy" ],
            [ "cDDD____", "plugin=isa" ],
            [ "____cDDD", "plugin=jerasure" ],
           ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

5.4.3. CRUSH 배치 제어

기본 **CRUSH** 규칙은 다른 호스트에 있는 **OSD**를 제공합니다. 예를 들면 다음과 같습니다.

```
chunk nr    01234567

step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

각 청크에 하나씩 정확히 **8** 개의 **OSD**가 필요합니다. 호스트가 두 개의 인접한 랙에 있으면 처음 **4**개의 청크를 첫 번째 랙에 배치하고 두 번째 랙의 마지막 **4**개 청크를 배치할 수 있습니다. 단일 **OSD**가 손실되는 경우 두 랙 간에 대역폭을 사용할 필요가 없습니다.

예를 들면 다음과 같습니다.

```
crush-steps='[ [ "choose", "rack", 2 ], [ "chooseleaf", "host", 4 ] ]'
```

는 랙 유형의 두 개의 분쇄 버킷을 선택하는 규칙을 만들고, 각각에 대해 서로 다른 호스트 버킷에 있는 **OSD 4**개를 선택합니다.

더 세분화된 제어를 위해 규칙을 수동으로 생성할 수도 있습니다.

5.5. ISA ERASURE 코드 플러그인

isa 플러그인은 **ISA** 라이브러리를 캡슐화합니다. **Intel** 프로세서에서만 실행됩니다.

isa 플러그인을 사용하여 새 삭제 코드 프로필을 생성하려면 다음 명령을 실행합니다.

```
ceph osd erasure-code-profile set <name> \
    plugin=isa \
    technique=<reed_sol_van|cauchy> \
    [k=<data-chunks>] \
    [m=<coding-chunks>] \
    [crush-root=<root>] \
    [crush-failure-domain=<bucket-type>] \
    [directory=<directory>] \
    [--force]
```

다음과 같습니다.

technique

설명

ISA 플러그인은 두 개의 **Reed Solomon** 형식으로 제공됩니다. **reed_sol_van** 이 설정된 경우 **Vandermonde, cauchy** 가 설정되면 **Cauchy**가 됩니다.

유형

문자열

필수 항목

아니요.

유효한 설정

reed_sol_van cauchy

기본값

reed_sol_van

k

설명

각 오브젝트는 데이터크크 부분으로 분할 되며 각각 다른 **OSD**에 저장됩니다.

유형

정수

필수 항목

아니요.

기본값

7

m

설명

*각 오브젝트의 청크 를 계산하여 서로 다른 **OSD**에 저장합니다. 데이터 손실 없이 축소할 수 있는 **OSD** 수입니다.*

유형

정수

필수 항목

아니요.

기본값

3

crush-root

설명

*규칙의 첫 번째 단계에 사용되는 **crush** 버킷의 이름입니다. 예를 들어, 단계가 기본값을 사용합니다.*

유형

문자열

필수 항목

아니요.

기본값

default

crush-failure-domain

설명

*두 청크가 동일한 실패 도메인이 있는 버킷에 없는지 확인합니다. 예를 들어, 실패 도메인이 호스트 되면 두 개의 청크가 동일한 호스트에 저장되지 않습니다. 스텝(**Fasper host**)을 선택하는 것과 같은 규칙 단계를 만드는 데 사용됩니다.*

유형

문자열

필수 항목

아니요.

기본값

host

디렉터리

설명

삭제 코드 플러그인이 로드된 디렉터리 이름을 설정합니다.

유형

문자열

필수 항목

아니요.

기본값

/usr/lib/ceph/erasure-code

--force

설명

동일한 이름으로 기존 프로필을 재정의합니다.

유형

문자열

필수 항목

아니요.

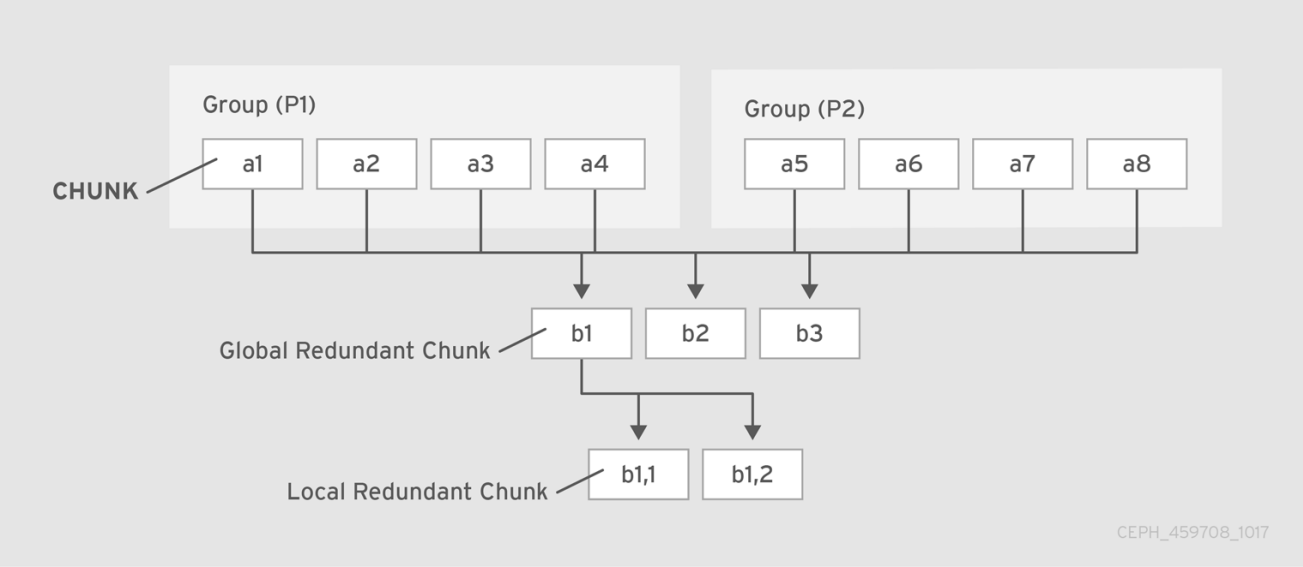
5.6. PYRAMID ERASURE CODE

***pyramid** 삭제 코드는 기본적으로 원래의 삭제 코드 알고리즘을 기반으로 한 개선된 알고리즘으로 데이터 액세스 및 복구를 개선할 수 있습니다. 이는 단순히 기존 **MDS (Maximum Disarance Separable)** 코드의 중단입니다. 여기서 **k** 는 원래 데이터 청크 수이고 **n** 은 코딩 프로세스 후 데이터 청크의 총 수입니다. **Pyramid** 코드는 표준 **MDS** 코드를 기반으로 하므로 인코딩/결제 접근 방식이 동일합니다. **pyramid** 코딩된 데이터 풀은 쓰기 오버헤드와 정상적인 삭제 코딩된 풀과 같은 복구 기능을 사용할 수 있습니다. **pyramid** 코드의 유일한 장점은 일반 삭제 코드와 비교하여 읽기 오버헤드를 크게 줄일 수 있다는 것입니다.*

***(11, 8)** 삭제 코딩된 풀의 예를 살펴보고 **pyramid** 코드 접근 방식을 적용합니다. 코딩된 풀을 **(12, 8) Pyramid** 코딩된 풀로 변환합니다. 삭제 코딩된 풀에는 **8** 개의 데이터 청크와 **11 - 8 = 3** 개의 중복 청크가 있습니다. **Pyramid** 코드는 **8** 데이터 청크를 **2**개의 동일한 크기 그룹으로 나눕니다. 즉 **P1 = {a1, a2, a3, a4}** 및 **P2 = {a5, a6, a7, a8}** 이는 변경되지 않은 삭제 코드에서 중복된 청크 두 개를 유지합니다(약 **b2** 및 **b3)**. 이 두 청크를 전역 중복 청크라고 합니다. **8**개의 모든 데이터 청크를 포함하기 때문입니다. 다음으로 그룹(또는 로컬) 중복 청크 **b1,1**로 표시되는 **P1** 그룹에 대해 새로운 중복 청크가 계산됩니다. 계산은 **P2**에서 **0**으로 모든 데이터 청크를 설정하는 것을 제외하고 원래 삭제 코드에서 **compute b1**인 것처럼 수행됩니다. 마찬가지로, **P2**에 대해 중복 청크 **b1,2** 그룹이 계산됩니다. 그룹 중복 청크는 해당 그룹의 데이터 청크만 영향을 받으며 다른 그룹에서는 영향을 받습니다. 중복 청크는 삭제 코드에서 원래 중복 청크의 예상으로 해석될 수 있습니다. **b1,1 + b1,2 = b1 = b1** 따라서 이 방식에서는 하나의 데이터 청크가 실패하면 일반 삭제 코딩된 풀에 대해 **8**개 대신 읽기 오버헤드 **4**를 사용하여 복구할 수 있습니다. 예를 들어, **P1** 의 **a3** 이 실패하고 **a1** 이 스크럽을 스크럽하는 기본 **OSD**인 경우 **a1,a2,a4** 및 **b1,1** 을 사용하여 **P2** 에서 읽는 대신 **a3** 을 복구할 수 있습니다. 모든 청크를 읽는 것은 동일한 그룹에서 두 개 이상의 청크가 누락되는 경우에만 필요합니다.*

*이 **Pyramid** 코드는 원래의 삭제 코드와 동일한 쓰기 오버헤드를 갖습니다. 데이터 청크가 업데이트될 때마다 **Pyramid Code**는 **3**개의 중복 청크(**b2, b3, b1,1** 또는 **b1,2)**를 업데이트해야 하는 반면, 삭제 코드는 **3**개의 중복 청크(**b1, b2** 및 **b3)**도 업데이트합니다. 또한 정상적인 삭제 코드와 동일하게 **3**개의 임의의 삭제 작업을 복구할 수 있습니다. 위에서 언급한 것과 같이 추가 중복 청크를 사용하는 비용으로 제공되는 유일한 이점만 더 적게 읽을 수 있다는 것입니다. 따라서 **Pyramid**는 액세스 효율성을 위해 스토리지 공간을 거래합니다.*

*다음 다이어그램은 구성된 **pyramid** 코드를 보여줍니다.*


Group (P1)
Group (P2)
CHUNK
a1
a2
a3
a4
a5
a6
a7
a8
Global Redundant Chunk
b1
b2
b3
Local Redundant Chunk
b1,1
b1,2
CEPH_459708_1017