Red Hat Ceph Storage 5

아키텍처 가이드

Red Hat Ceph Storage 아키텍처 가이드

Last Updated: 2022-08-15

Red Hat Ceph Storage 5 아키텍처 가이드

Red Hat Ceph Storage 아키텍처 가이드

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 Ceph Storage 클러스터 및 해당 클라이언트에 대한 아키텍처 정보를 제공합니다. 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. CEPH 아키텍처

Red Hat Ceph Storage 클러스터는 우수한 성능, 안정성 및 확장성을 제공하도록 설계된 분산 데이터 오브젝트 저장소입니다. 분산 개체 저장소는 구조화되지 않은 데이터를 수용하기 때문에 스토리지의 미래이며 클라이언트는 최신 오브젝트 인터페이스와 레거시 인터페이스를 동시에 사용할 수 있기 때문입니다.

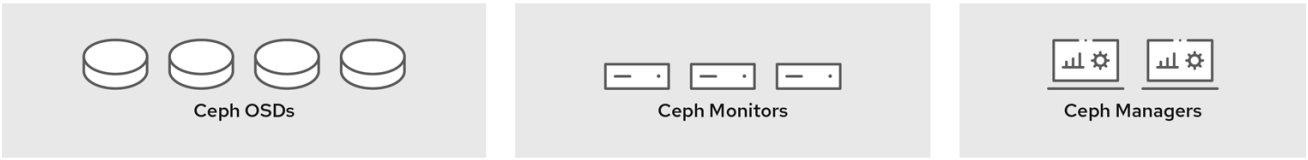
예를 들어 다음과 같습니다.

- API in many languages (C/C++, Java, Python)
- RESTful 인터페이스(S3/Swift)
- 블록 장치 인터페이스
- 파일 시스템 인터페이스

Red Hat Ceph Storage 클러스터의 강력한 기능은 조직의 IT 인프라 및 특히 Red Hat Enterprise Linux OSP와 같은 클라우드 컴퓨팅 플랫폼의 방대한 양의 데이터를 관리하는 기능을 전환할 수 있습니다. Red Hat Ceph Storage 클러스터는 **뛰어난** 확장성을 제공하며 페타바이트 규모의 클라이언트가 데이터를 엑사바이트 이상으로 제공합니다.

모든 Ceph 배포의 핵심은 Red Hat Ceph Storage 클러스터입니다. 세 가지 데몬 유형으로 구성됩니다.

- **Ceph OSD 데몬:** Ceph OSD는 Ceph 클라이언트를 대신하여 데이터를 저장합니다. 또한 Ceph OSD는 Ceph 노드의 CPU, 메모리, 네트워킹을 활용하여 데이터 복제, 삭제, 재조정, 복구, 모니터링 및 보고 기능을 수행합니다.
- **Ceph Monitor:** Ceph Monitor는 Red Hat Ceph Storage 클러스터의 현재 상태를 사용하여 Red Hat Ceph Storage 클러스터의 마스터 사본을 유지 관리합니다. 모니터는 일관성이 높고 Paxos를 사용하여 Red Hat Ceph Storage 클러스터 상태에 대한 계약을 체결해야 합니다.
- **Ceph Manager** 는 Ceph Monitor 대신 배치 그룹, 프로세스 메타데이터 및 호스트 메타데이터에 대한 자세한 정보를 유지 관리합니다. Ceph Manager는 배치 그룹 통계와 같은 여러 읽기 전용 Ceph CLI 쿼리의 실행을 처리합니다. Ceph Manager는 RESTful 모니터링 API도 제공합니다.



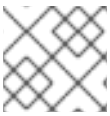
Ceph 클라이언트 인터페이스는 Red Hat Ceph Storage 클러스터에서 데이터를 읽고 씁니다. 클라이언트는 Red Hat Ceph Storage 클러스터와 통신하기 위해 다음 데이터가 필요합니다.

- Ceph 구성 파일 또는 클러스터 이름(일반적으로 **ceph**) 및 모니터 주소.
- 풀 이름입니다.
- 사용자 이름 및 시크릿 키의 경로입니다.

Ceph 클라이언트는 오브젝트 ID와 오브젝트를 저장하는 풀 이름을 유지합니다. 그러나 개체 간 인덱스를 유지하거나 개체 위치를 찾기 위해 중앙 집중식 개체 인덱스와 통신할 필요가 없습니다.However, they do not need to maintain an object-to-OSD index or communicate with a centralized object index to look up object locations. Ceph 클라이언트는 데이터를 저장하고 검색하기 위해 Ceph Monitor에 액세스하여 Red

Hat Ceph Storage 클러스터 맵의 최신 사본을 검색합니다. 그런 다음 Ceph 클라이언트는 librados에 개체 이름과 풀 이름을 제공합니다. **librados** 는 CRUSH(Controlled Replication Under scalable Hashing) 알고리즘을 사용하여 데이터를 저장하고 검색하기 위한 기본 OSD와 개체 이름을 librados에 제공합니다. Ceph 클라이언트는 읽기 및 쓰기 작업을 수행할 수 있는 기본 OSD에 연결됩니다. 클라이언트와 OSD 사이에 중간 서버, 브로커 또는 버스가 없습니다.

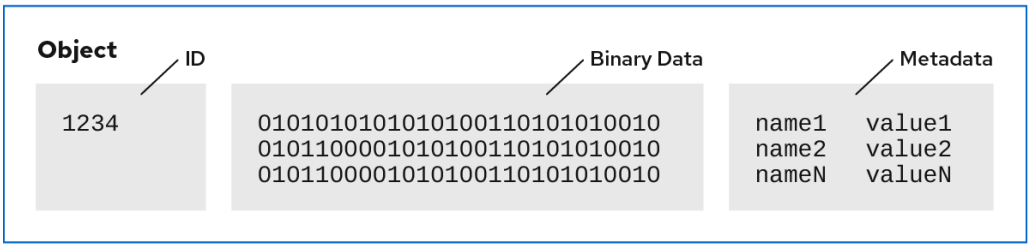
OSD가 데이터를 저장하면 클라이언트가 Ceph 블록 장치, Ceph Object Gateway, Ceph Filesystem 또는 다른 인터페이스이든 관계없이 데이터를 오브젝트로 저장합니다.



참고

개체 ID는 OSD의 스토리지 미디어뿐만 아니라 전체 클러스터에서 고유합니다.

Ceph OSD는 모든 데이터를 플랫 네임스페이스에 오브젝트로 저장합니다. 디렉터리의 계층은 없습니다. 오브젝트에는 클러스터 전체의 고유 식별자, 바이너리 데이터 및 메타데이터가 이름/값 쌍 세트로 구성됩니다.



Ceph 클라이언트는 클라이언트의 데이터 형식에 대한 의미를 정의합니다. 예를 들어, Ceph 블록 장치는 블록 장치 이미지를 클러스터에 저장된 일련의 오브젝트에 매핑합니다.

참고

고유한 ID, 데이터 및 이름/값 쌍으로 구성된 오브젝트는 구조화된 데이터와 비정형 데이터 및 선행 엣지 데이터 스토리지 인터페이스를 모두 나타낼 수 있습니다.

2장. 핵심 CEPH 구성 요소

Red Hat Ceph Storage 클러스터는 무제한 확장성, 고가용성 및 성능을 위해 다수의 Ceph 노드를 보유할 수 있습니다. 각 노드는 다음과 같은 서로 통신하는 비독점 하드웨어 및 지능형 Ceph 데몬을 활용합니다.

- 쓰기 및 데이터 읽기
- 데이터 압축
- 코딩 데이터를 복제하거나 삭제하여 내구성을 보장합니다.
- 클러스터 상태 (heartbeating)라고도 하는 클러스터 상태 모니터링 및 보고
- 동적으로 데이터 재분배 (backfilling)라고도 합니다.
- 데이터 무결성 확인; 및,
- 오류를 복구합니다.

데이터를 읽고 쓰는 Ceph 클라이언트 인터페이스의 경우 Red Hat Ceph Storage 클러스터는 데이터를 저장하는 간단한 풀처럼 보입니다. 그러나 **librados** 및 스토리지 클러스터는 클라이언트 인터페이스에 완전히 투명한 방식으로 많은 복잡한 작업을 수행합니다. Ceph 클라이언트 및 Ceph OSD는 모두 CRUSH(Controlled Replication Under scalable Hashing) 알고리즘을 사용합니다. 다음 섹션에서는 CRUSH를 통해 이러한 작업을 원활하게 수행하는 방법에 대해 자세히 설명합니다.

2.1. 사전 요구 사항

- 분산 스토리지 시스템에 대한 기본적인 이해.

2.2. CEPH 풀

Ceph 스토리지 클러스터는 'Pools'라는 논리 파티션에 데이터 오브젝트를 저장합니다. Ceph 관리자는 블록 장치, 개체 게이트웨이 또는 특정 사용자 그룹과 같은 특정 유형의 데이터에 대한 풀을 생성할 수 있습니다.

Ceph 클라이언트 관점에서 스토리지 클러스터는 매우 간단합니다. Ceph 클라이언트에서 I/O 컨텍스트를 사용하여 데이터를 읽거나 쓸 때 **항상** Ceph 스토리지 클러스터의 스토리지 풀에 연결됩니다. 클라이언트는 풀 이름, 사용자 및 시크릿 키를 지정하므로 풀이 데이터 개체에 대한 액세스 제어가 있는 논리 파티션 역할을 하는 것으로 나타납니다.

실제로 Ceph 풀은 오브젝트 데이터를 저장하기 위한 논리 파티션일 뿐만 아니라. 풀은 Ceph 스토리지 클러스터가 데이터를 배포하고 저장하는 방법에 중요한 역할을 합니다. 그러나 이러한 복잡한 작업은 Ceph 클라이언트에 완전히 투명하게 이루어집니다.

Ceph 풀은 다음을 정의합니다.

- **풀 유형:** 이전 버전의 Ceph에서 풀은 개체의 여러 깊은 복사본을 간단하게 유지했습니다. 현재 Ceph는 오브젝트 복사본을 여러 개 유지 관리하거나 삭제 코딩을 사용하여 내구성을 보장할 수 있습니다. 데이터 내구성 방법은 풀 전체이며 풀을 생성한 후에는 변경되지 않습니다. 풀 유형은 풀을 생성할 때 데이터 내구성 방법을 정의합니다. 풀 유형은 클라이언트에 완전히 투명합니다.
- **배치 그룹:** 엑사바이트 규모의 스토리지 클러스터에서 Ceph 풀은 수백만 개의 데이터 오브젝트를 저장할 수 있습니다. Ceph는 복제본 또는 삭제 코드 청크를 통한 데이터 내구성, 데이터 무결성, CRC 검사, 복제, 재조정 및 복구를 포함하여 다양한 유형의 작업을 처리해야 합니다. 결과적으로 오브젝트별 데이터를 관리하면 확장성 및 성능 병목 현상이 발생합니다. Ceph는 풀을 배치 그룹으로 분할하여 이러한 병목 현상을 해결합니다. CRUSH 알고리즘은 개체를 저장하기 위한 배치

그룹을 계산하고 배치 그룹에 대한 OSD Set을 계산합니다. CRUSH는 각 개체를 배치 그룹에 배치합니다. 그런 다음 CRUSH는 각 배치 그룹을 OSD 세트에 저장합니다. 시스템 관리자는 풀을 생성하거나 수정할 때 배치 그룹 수를 설정합니다.

- **CRUSH 규칙 세트:** CRUSH는 또 다른 중요한 역할을 수행합니다. CRUSH는 장애 도메인 및 성능 도메인을 감지할 수 있습니다. CRUSH는 스토리지 미디어 유형별로 OSD를 식별하고 OSD를 노드, 랙 및 행으로 계층적으로 구성할 수 있습니다. CRUSH를 사용하면 Ceph OSD에서 장애 도메인에서 오브젝트 복사본을 저장할 수 있습니다. 예를 들어, 오브젝트의 사본은 다른 서버 방, 아스플, 랙 및 노드에 저장될 수 있습니다. 랙과 같이 클러스터의 많은 부분이 실패하면 클러스터가 복구될 때까지 클러스터가 저하된 상태에서 작동할 수 있습니다.

또한 CRUSH를 사용하면 클라이언트가 SSDs, SSD 저널을 사용하는 하드 드라이브, 데이터와 동일한 드라이브에 저널이 있는 하드 드라이브와 같은 특정 하드웨어에 데이터를 쓸 수 있습니다. CRUSH 규칙 세트는 풀에 대한 실패 도메인 및 성능 도메인을 결정합니다. 관리자는 풀을 만들 때 CRUSH 규칙 세트를 설정합니다.

참고

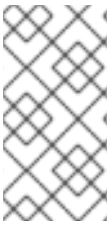
관리자는 풀을 생성한 후 풀의 규칙 세트를 변경할 **수 없습니다**.

- **내구성**: 엑사바이트 규모 스토리지 클러스터에서는 하드웨어 장애가 예상되는데 예외가 아닙니다. 데이터 오브젝트를 사용하여 블록 장치와 같은 보다 세분화된 스토리지 인터페이스를 나타낼 때, 더 세분화된 인터페이스에 대한 하나 이상의 데이터 오브젝트를 손실하면 더 세분화된 스토리지 엔티티의 무결성을 손상시킬 수 있습니다. 따라서 데이터 손실은 불가능합니다. Ceph는 다음 두 가지 방법으로 높은 데이터 내구성을 제공합니다.
  - 복제본 풀은 CRUSH 오류 도메인을 사용하여 여러 개체의 깊은 복사본을 저장하여 하나의 데이터 개체 복사본을 물리적으로 분리합니다. 즉, 사본은 별도의 물리적 하드웨어에 배포됩니다. 이렇게 하면 하드웨어 오류 중에 내구성이 높아집니다.
  - 복원된 풀은 각 개체를 **K+M** 청크로 저장합니다. 여기서 **K** 는 데이터 청크를 나타내고 **M** 은 코딩 청크를 나타냅니다. 합계는 오브젝트를 저장하는 데 사용되는 OSD 수를 나타내고 **M** 값은 실패 할 수 있는 OSD 수를 나타내며 **M** OSD 수가 실패해야 합니다.

클라이언트 관점에서는 Ceph가 우아하고 단순합니다. 클라이언트는 풀에서 읽고 쓰기만 하면 됩니다. 그러나 풀은 데이터 내구성, 성능 및 고가용성에 중요한 역할을 합니다.

2.3. CEPH 인증

Ceph는 사용자를 식별하고 메시지 가로채기(man-in-the-middle) 공격을 방지하기 위해 iRMC 인증 시스템을 제공하여 사용자 및 데몬을 인증합니다.



참고

wget 프로토콜은 OSD에 저장된 네트워크 또는 데이터를 통해 전송되는 데이터의 데이터 암호화를 처리하지 않습니다.

cephx는 인증에 공유 비밀 키를 사용합니다. 즉 클라이언트와 모니터 클러스터 모두 클라이언트의 비밀 키 사본이 있습니다. 인증 프로토콜을 통해 두 당사자 모두 실제로 공개하지 않고 키 사본을 가지고 있

음을 증명할 수 있습니다. 이렇게 하면 상호 인증을 제공하므로 클러스터가 비밀 키를 가지고 있는지 확인하고, 사용자는 클러스터에 비밀 키 사본이 있는지 확인합니다.

numberds

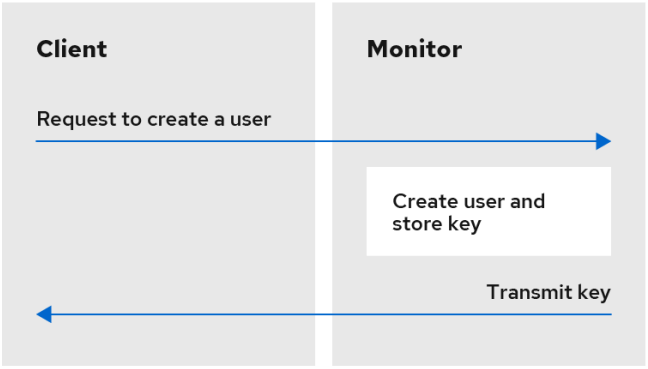
wget 인증 프로토콜은 Kerberos와 유사한 방식으로 작동합니다.

사용자/actor는 Ceph 클라이언트를 호출하여 모니터에 연결합니다. Kerberos와 달리 각 모니터는 사용자를 인증하고 키를 배포할 수 있으므로, wget을 사용할 때 단일 장애 지점이나 병목 현상이 발생하지 않습니다. 모니터는 Ceph 서비스를 얻는 데 사용할 세션 키가 포함된 Kerberos 티켓과 유사한 인증 데이터 구조를 반환합니다. 이 세션 키는 사용자가 Ceph 모니터에서 서비스를 요청할 수 있도록 사용자의 영구 보안 키를 사용하여 자체적으로 암호화됩니다. 그러면 클라이언트에서 세션 키를 사용하여 모니터에서 원하는 서비스를 요청하고 모니터에서 실제로 데이터를 처리하는 OSD에 클라이언트를 인증할 티켓을 제공합니다. Ceph 모니터 및 OSD는 시크릿을 공유하므로 클라이언트는 클러스터의 OSD 또는 메타데이터 서버와 함께 모니터에서 제공하는 티켓을 사용할 수 있습니다. Kerberos와 같이 288 티켓이 만료되므로 공격자는 만료된 티켓 또는 세션 키를 자주 사용할 수 없습니다. 이러한 형태의 인증으로 인해 공격자는 다른 사용자의 아이덴티티 아래 bogus 메시지를 만들거나 다른 사용자의 적법 메시지를 변경하는 것이 만료되기 전에 공개되지 않는 한 통신 미디어에 액세스 할 수 없습니다.

wget 을 사용하려면 관리자가 먼저 사용자를 설정해야 합니다. 다음 다이어그램에서 client.admin 사용자는 명령줄에서 ceph auth get-or-create-key 를 호출하여 사용자 이름 및 비밀 키를 생성합니다. Ceph의 auth 하위 시스템은 사용자 이름과 키를 생성하고, monitor를 사용하여 복사본을 저장하고, 사용자의 시크릿을 client.admin 사용자에게 다시 전송합니다. 즉 클라이언트와 모니터는 비밀 키를 공유합니다.

참고

client.admin 사용자는 안전한 방식으로 사용자에게 사용자 ID 및 비밀 키를 제공해야 합니다.



158_Ceph_0521

2.4. CEPH 배치 그룹

클러스터에 수백만 개의 오브젝트를 저장하고 개별적으로 관리하는 것은 리소스 집약적입니다. 따라서 Ceph는 배치 그룹(PG)을 사용하여 대규모 오브젝트를 보다 효율적으로 관리합니다.

PG는 개체 컬렉션을 포함하는 풀의 하위 집합입니다. Ceph가 풀을 일련의 PG로 분할합니다. 그런 다음 CRUSH 알고리즘은 클러스터 맵과 클러스터의 상태를 고려하여 PG를 균등하게 및 의사 임의로 클러스터의 OSD에 배포합니다.

이것이 작동하는 방식입니다.

시스템 관리자가 풀을 생성할 때 CRUSH는 풀에 대해 사용자 정의 PG 수를 만듭니다. 일반적으로 PG의 수는 데이터의 합리적으로 세분화된 하위 집합이어야 합니다. 예를 들어 풀당 OSD당 OSD 100개가 있으면 각 PG에 풀 데이터의 약 1%가 포함됩니다.

Ceph가 하나의 OSD에서 다른 OSD로 PG를 이동해야 하는 경우 PG의 수가 성능에 영향을 미칩니다. 풀에 PG가 너무 적으면 Ceph는 많은 양의 데이터를 동시에 이동할 것이며 네트워크 로드는 클러스터 성능에 부정적인 영향을 미칩니다. 풀에 PG가 너무 많으면 Ceph는 데이터의 작은 비율을 이동할 때 너무 많은 CPU와 RAM을 사용하므로 클러스터의 성능에 부정적인 영향을 미칩니다. 최적의 성능을 달성하기 위해 PG 수를 계산하는 방법에 대한 자세한 내용은 PG Count 를 참조하십시오.

Ceph는 개체의 복제본을 저장하거나 개체의 이차 코드 청크를 저장하여 데이터 손실을 방지합니다. Ceph는 PG에 오브젝트의 개체 또는 삭제 코드 청크를 저장하므로 Ceph는 오브젝트의 각 사본 또는 오브젝트의 코드 청크마다 "작업 세트"라는 OSD 세트에서 각 PG를 복제합니다. 시스템 관리자는 풀의 PG 수와 복제본 또는 삭제 코드 청크 수를 확인할 수 있습니다. 그러나 CRUSH 알고리즘은 특정 PG에 대해 설정된 OSD를 계산합니다.

CRUSH 알고리즘과 PG를 사용하면 Ceph가 동적입니다. 클러스터 맵 또는 클러스터 상태가 변경되면 Ceph가 하나의 OSD에서 다른 OSD로 자동 이동하는 PG를 이동할 수 있습니다.

다음은 몇 가지 예입니다.

- 클러스터 확장: 클러스터에 새 호스트와 OSD를 추가하면 클러스터 맵이 변경됩니다. CRUSH는 클러스터 전체에서 PG를 OSD에 균등하게 배포하므로 새 호스트와 해당 OSD를 추가하면 CRUSH가 일부 풀의 배치 그룹을 새 OSD에 다시 할당합니다. 즉, 시스템 관리자는 클러스터를 수동으로 리밸런스할 필요가 없습니다. 또한 새 OSD에는 다른 OSD와 거의 동일한 양의 데이터가 포함됩니다. 또한 새 OSD에는 새로 작성된 OSD가 포함되어 있지 않으므로 클러스터에서 "시점"이 발생하지 않습니다.

- OSD Fails: OSD가 실패하면 클러스터의 상태가 변경됩니다. Ceph는 복제본 또는 삭제 코드 청크 중 하나를 일시적으로 손실하고 다른 복사본을 만들어야 합니다. 작업 세트의 기본 OSD가

실패하면 작동 세트의 다음 OSD가 기본 상태가 되고 CRUSH는 새 OSD를 계산하여 추가 복사 또는 삭제 코드 청크를 저장합니다.

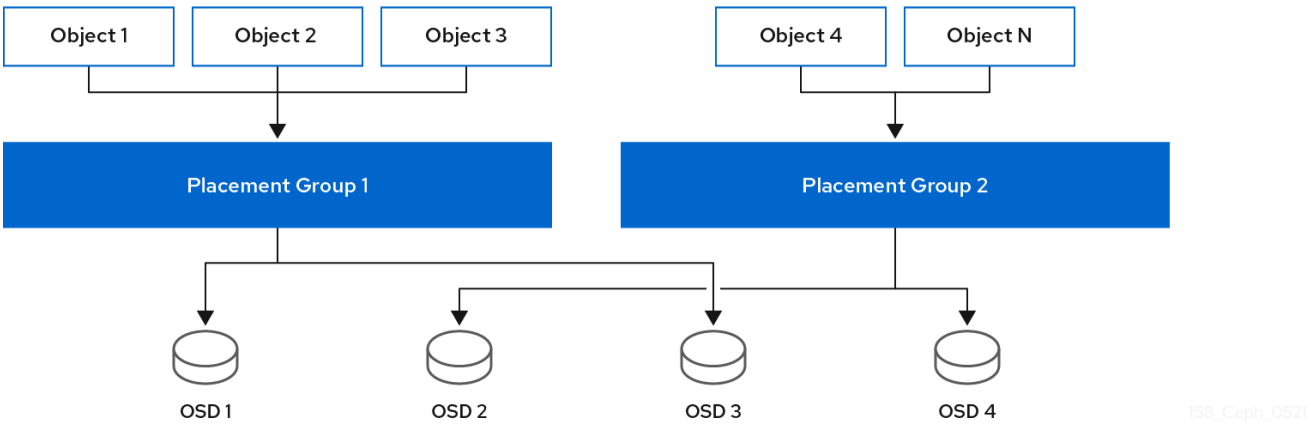
Ceph 스토리지 클러스터는 수백~ thousands개의 PG 컨텍스트 내에서 수백만 개의 오브젝트를 관리하여 오류를 효율적으로 축소 및 복구할 수 있습니다.

Ceph 클라이언트의 경우 librados 를 통한 CRUSH 알고리즘은 개체를 읽고 쓰는 프로세스를 매우 간단하게 합니다. Ceph 클라이언트는 간단히 오브젝트를 풀에 쓰거나 풀에서 오브젝트를 읽기만 하면 됩니다. 작동 세트의 기본 OSD는 Ceph 클라이언트를 대신하여 작동 중인 세트의 개체 또는 삭제 코드 청크를 보조 OSD에 작성할 수 있습니다.

클러스터 맵 또는 클러스터 상태가 변경되면 PG를 저장하는 OSD의 CRUSH 계산도 변경됩니다. 예를 들어 Ceph 클라이언트는 foo 오브젝트를 풀 표시줄에 작성할 수 있습니다. CRUSH는 오브젝트를 PG 1.a 에 할당하고 OSD 5 에 각각 저장하고 OSD 10 및 OSD 15 의 복제본을 각각 저장합니다. OSD 5 가 실패하면 클러스터 상태가 변경됩니다. Ceph 클라이언트가 풀 바 로부터 foo 인 오브젝트를 읽을 때 librados 를 통한 클라이언트는 새 기본 OSD로 OSD 10 에서 자동으로 검색합니다.

librados 를 통한 Ceph 클라이언트는 오브젝트를 작성하고 읽을 때 작동하는 세트의 기본 OSD에 직접 연결됩니다. I/O 작업에서는 중앙 집중식 브로커를 사용하지 않으므로 네트워크 초과 구독은 일반적으로 Ceph에 문제가 되지 않습니다.

다음 다이어그램에서는 CRUSH가 오브젝트를 PG에 할당하는 방법과 PG를 OSD에 할당하는 방법을 보여줍니다. CRUSH 알고리즘은 작업 세트의 각 OSD가 별도의 장애 도메인에 있도록 PG를 OSD에 할당합니다. 일반적으로 OSD는 항상 별도의 서버 호스트에 있고 때로는 별도의 랙에 있습니다.



2.5. CEPH CRUSH 규칙 세트

Ceph는 풀에 CRUSH 규칙 세트를 할당합니다. Ceph 클라이언트가 풀에 데이터를 저장하거나 검색할 때 Ceph는 CRUSH 규칙 세트, 규칙 내에서 규칙 내의 규칙, 데이터를 저장 및 검색하는 규칙의 최상위 버킷을 식별합니다. Ceph에서 CRUSH 규칙을 처리하면 개체의 배치 그룹이 포함된 기본 OSD를 식별합니

다. 이를 통해 클라이언트는 OSD에 직접 연결하고 배치 그룹에 액세스하여 개체 데이터를 읽거나 쓸 수 있습니다.

배치 그룹을 OSD에 매핑하기 위해 CRUSH 맵은 계층적인 버킷 유형 목록을 정의합니다. 버킷 유형 목록은 생성된 CRUSH 맵의 유형에 있습니다. 버킷 계층 구조를 생성하는 목적은 장애 도메인 및/또는 성능 도메인(예: 드라이브 유형, 호스트, 섀시, 랙, 전원 분배 단위, 포드, 행, 공간, 데이터 센터)을 통해 리프 노드를 분리하는 것입니다.

OSD를 나타내는 리프 노드를 제외하고 나머지 계층 구조는 임의적입니다. 기본 유형이 요구 사항에 맞지 않는 경우 관리자는 자체 필요에 따라 정의할 수 있습니다. CRUSH는 일반적으로 계층 구조에서 Ceph OSD 노드를 모델링할 수 있는 보관기 그래프를 지원합니다. 따라서 Ceph 관리자는 단일 CRUSH 맵에 여러 루트 노드가 있는 여러 계층을 지원할 수 있습니다. 예를 들어 관리자는 고성능을 위해 높은 비용의 SSD를 나타내는 계층 구조와 SSD의 중간 성능을 위해 SSD 저널이 있는 비용이 낮은 하드 드라이브의 별도 계층을 생성할 수 있습니다.

2.6. CEPH 입력/출력 작업

Ceph 클라이언트는 Ceph 모니터에서 'Cluster Map'을 검색하고 풀에 바인딩한 다음 풀의 배치 그룹 내의 개체에서 I/O(입력/출력)를 수행합니다. 풀의 CRUSH 규칙 세트와 배치 그룹의 수는 Ceph가 데이터를 배치하는 방법을 결정하는 주요 요소입니다. 최신 버전의 클러스터 맵에서 클라이언트는 클러스터의 모든 모니터와 OSD와 현재 상태를 알고 있습니다. 그러나 클라이언트는 개체 위치에 대해 아무것도 알 수 없습니다.

클라이언트에 필요한 유일한 입력은 오브젝트 ID와 풀 이름입니다. 간단하게: Ceph에서 이름이 지정된 풀에 데이터를 저장합니다. 클라이언트가 풀에 명명된 오브젝트를 저장하려는 경우 개체 이름, 해시 코드, 풀의 PG 수 및 풀 이름을 입력으로 저장하려는 경우 CRUSH(Controlled Replication Under Scalable Hashing)는 배치 그룹의 ID와 배치 그룹의 기본 OSD를 계산합니다.

Ceph 클라이언트는 다음 단계를 사용하여 PG ID를 계산합니다.

1. 클라이언트에서 풀 ID와 오브젝트 ID를 입력합니다. 예를 들어 pool = 간풀 및 object-id = john 입니다.

2. CRUSH는 개체 ID를 사용하고 이를 해시합니다.

3. CRUSH는 PGs 수의 해시 모드를 계산하여 PG ID를 가져옵니다. 예를 들면 58

4. CRUSH는 PG ID에 해당하는 기본 OSD를 계산합니다.

5. 클라이언트는 풀 이름이 지정된 풀 ID를 가져옵니다. 예를 들어, pool 간pool 은 4 번의 풀입니다.

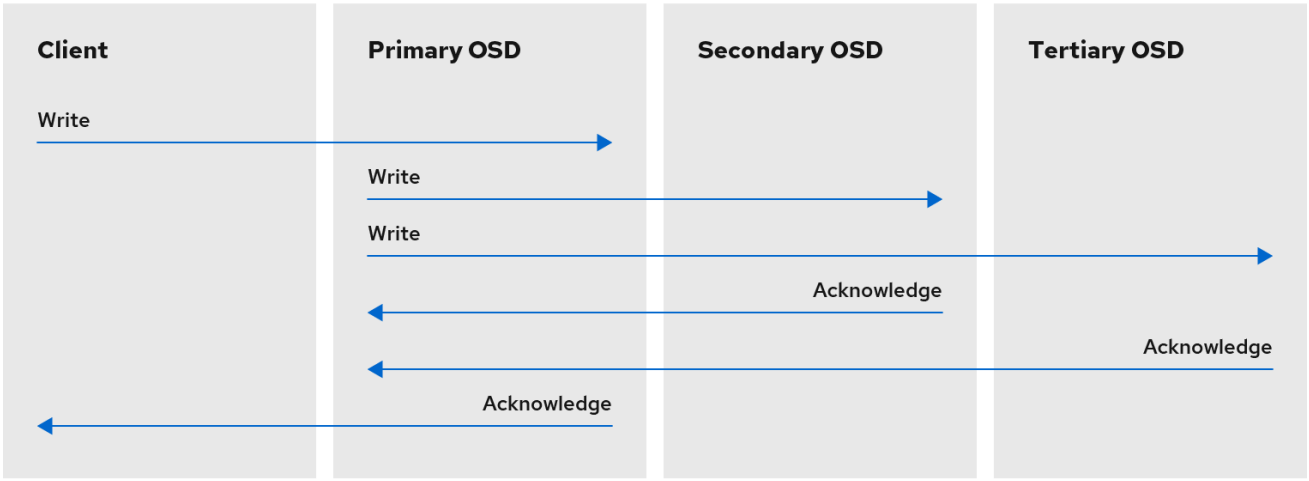
6. 클라이언트는 풀 ID를 PG ID에 추가합니다. 예를 들면 4.58 입니다.

7. 클라이언트는 Sting Set에서 기본 OSD와 직접 통신하여 오브젝트 작업을 수행합니다.

Ceph 스토리지 클러스터의 토폴로지 및 상태는 세션 중에 상대적으로 안정적입니다. librados 를 통해 개체 위치를 계산하도록 Ceph 클라이언트에 권한을 부여하면 클라이언트가 각 읽기/쓰기 작업에 대해 chatty 세션을 통해 스토리지 클러스터에 쿼리해야 하는 것보다 훨씬 빠릅니다. CRUSH 알고리즘을 사용하면 클라이언트가 오브젝트 를 저장할 위치를 계산할 수 있으며 클라이언트는 개체에 직접 데이터를 저장하거나 검색하기 위해 작동 세트의 기본 OSD에 연결할 수 있습니다. 엑사바이트 규모의 클러스터에서 수천 개의 OSD가 있으므로 클라이언트와 Ceph OSD 간의 네트워크 초과 구독은 문제가 되지 않습니다. 클러스터 상태가 변경되면 클라이언트는 Ceph 모니터에서 클러스터 맵에 업데이트를 요청할 수 있습니다.

2.7. CEPH 복제

Ceph 클라이언트와 마찬가지로 Ceph OSD는 Ceph 모니터에 연결하여 클러스터 맵의 최신 복사본을 검색할 수 있습니다. Ceph OSD는 CRUSH 알고리즘을 사용하지만 개체의 복제본을 저장하는 위치를 계산하는 데 사용합니다. 일반적인 쓰기 시나리오에서 Ceph 클라이언트는 CRUSH 알고리즘을 사용하여 개체의 Sting Set에서 배치 그룹 ID와 기본 OSD를 계산합니다. 클라이언트가 오브젝트를 기본 OSD에 쓸 때 기본 OSD는 저장해야 하는 복제본 수를 찾습니다. 값은 osd_pool_default_size 설정에 있습니다. 그런 다음 기본 OSD는 개체 ID, 풀 이름 및 클러스터 맵을 사용하고 CRUSH 알고리즘을 사용하여 작동 세트에 대한 보조 OSD의 ID를 계산합니다. 기본 OSD는 오브젝트를 보조 OSD에 씁니다. 기본 OSD가 보조 OSD에서 확인 작업을 수신하고 기본 OSD 자체에서 쓰기 작업을 완료하면 Ceph 클라이언트에 대한 쓰기 작업이 완료되었음을 확인합니다.



Ceph 클라이언트를 대신하여 데이터 복제를 수행할 수 있으므로 Ceph OSD 데몬은 Ceph 클라이언트가 이를 통해 동시에 높은 데이터 가용성 및 데이터 보안을 보장합니다.

참고

일반적으로 기본 OSD 및 보조 OSD는 별도의 장애 도메인에 있도록 구성됩니다. CRUSH는 장애 도메인에 대한 고려 사항에 따라 보조 OSD의 ID를 계산합니다.

데이터 사본

복제된 스토리지 풀에서 Ceph는 성능이 저하된 상태에서 작동하기 위해 오브젝트의 여러 사본이 필요합니다. Ceph 스토리지 클러스터를 사용하면 클라이언트가 작동 중인 세트의 OSD 중 하나가 실패하더라도 데이터를 읽고 쓸 수 있습니다. 따라서 Ceph는 쓰기 작업을 위해 최소 두 개 이상의 복사본이 정리된 오브젝트의 복사본 3개를 생성합니다. Ceph는 두 개의 OSD가 실패하더라도 데이터를 보존합니다. 그러나 쓰기 작업이 중단됩니다.

소거 코드된 풀에서 Ceph는 성능이 저하된 상태에서 작동할 수 있도록 여러 OSD에서 오브젝트 청크를 저장해야 합니다. 복제된 풀과 유사하게 이상적으로는 복구된 풀로 인해 Ceph 클라이언트가 성능 저하된 상태에서 읽고 쓸 수 있습니다.

중요

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

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

2.8. CEPH 코딩 삭제

Ceph는 많은 삭제 코드 알고리즘 중 하나를 로드할 수 있습니다. 가장 먼저 사용되는 가장 빠른 방법은 Reed-Solomon 알고리즘입니다. 삭제 코드는 실제로 FEC(forward error correction) 코드입니다. FEC 코드는 K 청크의 메시지를 '코드 단어' N 청크라는 긴 메시지로 변환하므로 Ceph가 N 청크의 하위 집합에서 원본 메시지를 복구할 수 있습니다.

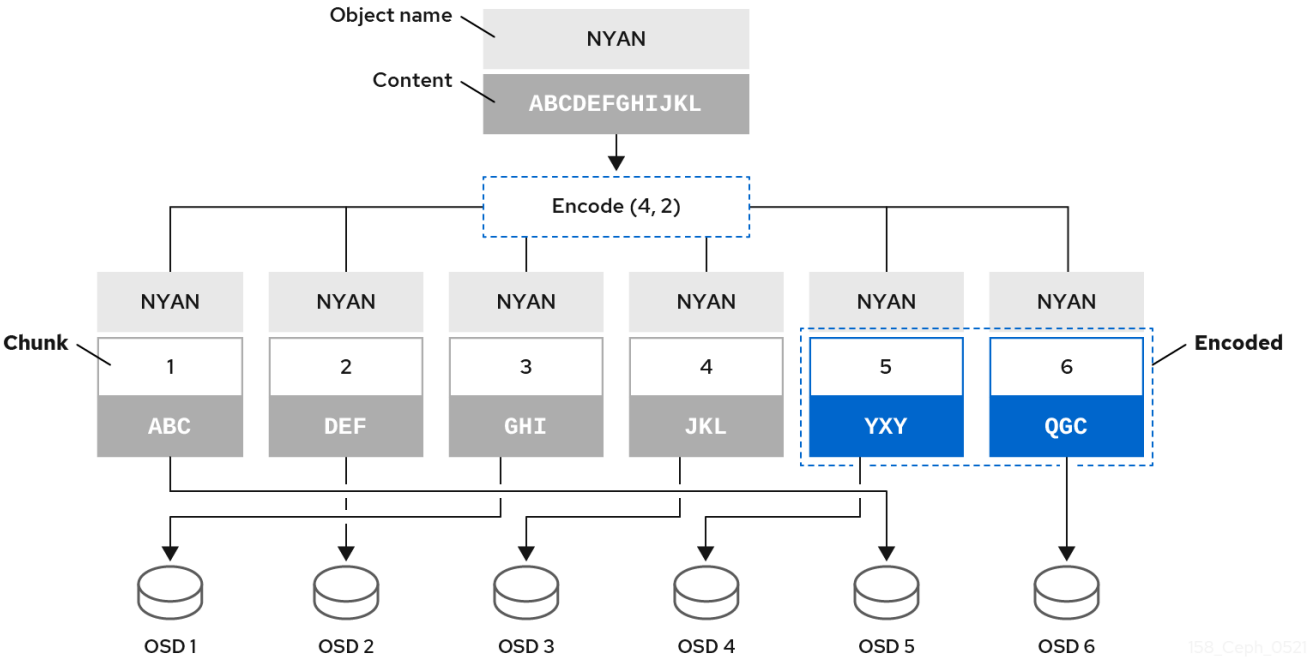
특히 N = K+M 변수 K +M은 원래 데이터 청크 양입니다. 변수 M 은 삭제 코드 알고리즘이 실패로부터 보호를 제공하기 위해 추가하는 추가 또는 중복된 청크를 나타냅니다. 변수 N 은 삭제 코딩 프로세스 이후에 생성된 총 청크 수입니다. M 값은 단순히 N- K 로, 알고리즘은 K 원래 데이터 청크에서 N-K 중복 청크를 계산합니다. 이 방법을 사용하면 Ceph가 모든 원본 데이터에 액세스할 수 있습니다. 시스템은 임의의 N-K 오류에 탄력적입니다. 예를 들어 10 K 의 16 N 구성 또는 삭제 코딩 10/16 에서 삭제 코드 알고리즘은 10 개의 기본 청크 K 에 6 개의 추가 청크를 추가합니다. 예를 들어 M = K-N 또는 16-10 = 6 구성에서 Ceph는 16개의 청크 N 을 16개의 OSD로 분산합니다. 6개의 OSD가 실패하는 경우에도 원본 파일을 10 개의 확인된 N 청크에서 재구축할 수 있습니다. 이는 Red Hat Ceph Storage 클러스터가 데이터가 손실되지 않아 매우 높은 수준의 내결함성을 보장합니다.

복제된 풀과 마찬가지로 삭제 코드된 풀에서는 up 세트의 기본 OSD는 모든 쓰기 작업을 수신합니다. 복제된 풀에서 Ceph는 세트의 보조 OSD에서 배치 그룹에 있는 각 오브젝트의 깊은 복사본을 만듭니다. 코딩 삭제의 경우 프로세스는 약간 다릅니다. 복제 코딩된 풀은 각 개체를 K+M 청크로 저장합니다. K 데이터 청크와 M 코딩 청크로 나뉩니다. Ceph가 작동 중인 세트에서 각 청크를 OSD에 저장하도록 풀은 K+M 크기를 갖도록 구성됩니다. Ceph는 청크 순위를 오브젝트의 속성으로 저장합니다. 기본 OSD는 페이로드를 K+M 청크로 인코딩하여 다른 OSD로 전송합니다. 기본 OSD는 권한 있는 배치 그룹 로그 버전을 유지 관리해야 합니다.

예를 들어, 일반적인 구성에서 시스템 관리자는 6개의 OSD를 사용하고 두 개의 OSD를 계속 손실하기 위해 삭제 코딩된 풀을 생성합니다. 즉, (K+M = 6)와 같이 (M = 2)입니다.

Ceph가 ABCDEFGHIJKL 을 포함하는 개체 NYAN 을 풀에 쓰는 경우, 삭제 인코딩 알고리즘은 단순히

콘텐츠를 4개 부분, DEF,DEF,GHI, JKL 과 같은 네 부분으로 나누어 콘텐츠를 4개의 데이터 청크로 분할합니다. 콘텐츠 길이가 K 의 복수가 아닌 경우 알고리즘은 콘텐츠를 패딩합니다. 이 함수는 두 개의 코딩 청크를 만듭니다. YXY 를 사용한 다섯 번째와 QGC 가 있는 여섯 번째 청크도 생성됩니다. Ceph는 각 청크를 작동 세트에 저장합니다. 여기서 청크는 동일한 이름인 NYAN 이 있지만 다른 OSD에 상주하는 오브젝트에 청크를 저장합니다. 알고리즘은 청크를 해당 이름 외에도 오브젝트 shard_t 의 속성으로 생성한 순서를 유지해야 합니다. 예를 들어, Chunk 1은 ABC 를 포함하고 Ceph는 OSD5 에 저장하고, 청크 5에는 OSD4 에 YXY 및 Ceph 저장소가 포함되어 있습니다.



복구 시나리오에서 클라이언트는 청크 1에서 6까지의 청크를 읽고 삭제 코드된 풀에서 NYAN 을 읽습니다. OSD는 청크 2 및 6이 누락되었음을 알고리즘에 알립니다. 이러한 누락된 청크를 'erasures'라고 합니다. 예를 들어 OSD6이 부족하기 때문에 기본 OSD 6 를 읽을 수 없으며 OSD2 가 느리고 청크 청크를 고려하지 않았기 때문에 청크 2를 읽을 수 없습니다. 그러나 알고리즘에 4개의 청크가 있는 즉시 4개의 청크를 읽습니다. chunk 1에는bc 1이 포함된 GHI, 청크 3, JKL 이 포함된 청크 4와 YXY 가 포함된 청크 5입니다. 그런 다음, 개체의 원래 콘텐츠를 다시 작성합니다. ABCDEFGHIJKL, 그리고 QGC 가 포함 된 청크 6의 원래 내용을 다시 작성합니다.

데이터를 청크로 분할하는 것은 개체 배치와 독립적입니다. CRUSH 규칙 세트는 삭제 코드화된 풀 프로필과 함께 OSD에 청크 배치를 결정합니다. 예를 들어, 삭제 코드 프로파일에서 로컬 복구 코드(lrc) 플러그인을 사용하면 추가 청크가 생성되고 복구할 수 있는 OSD 수가 줄어듭니다. 예를 들어, lrc 프로파일 구성에서 K=4 M=2L=3. .이 알고리즘은 jerasure 플러그인과 마찬가지로 6개의 청크(K+M)를 생성하지만 알고리즘에서 로컬로 2개의 청크를 생성해야 합니다. 알고리즘은 (K+M)/L 과 같은 추가 청크를 생성합니다. 청크 0을 포함하는 OSD가 실패하면 청크 1, 2 및 첫 번째 로컬 청크를 사용하여 이 청크를 복구할 수 있습니다. 이 경우 알고리즘은 5 대신 3개의 청크만 복구해야 합니다.

참고

삭제 코드화된 풀을 사용하면 오브젝트 맵이 비활성화됩니다.

추가 리소스

- 대화형 서명 프로필 및 플러그인인 CRUSH에 대한 자세한 내용은 Red Hat Ceph Storage 5의 스토리지 전략 가이드 를 참조하십시오.
- 오브젝트 맵에 대한 자세한 내용은 Ceph 클라이언트 오브젝트 맵 섹션을 참조하십시오.

2.9. CEPH OBJECTSTORE

ObjectStore 는 OSD의 원시 블록 장치에 하위 수준 인터페이스를 제공합니다. 클라이언트가 데이터를 읽거나 쓸 때 ObjectStore 인터페이스와 상호 작용합니다. Ceph 쓰기 작업은 기본적으로 ACID 트랜잭션입니다. 즉, Atomicity,Consistency,Isolation 및 Durability 을 제공합니다. ObjectStore 는 트랜잭션 이 Atomicity 를 제공하기 위해 all-or-nothing이라고 합니다. ObjectStore 는 개체의 의미도 처리합니다. 스토리지 클러스터에 저장된 오브젝트에는 고유한 식별자, 오브젝트 데이터 및 메타데이터가 있습니다. 따라서 ObjectStore 는 Ceph 개체의 의미가 올바른지 확인하여 일관성 을 제공합니다. ObjectStore 는 또한 쓰기 작업에서 Sequencer 를 호출하여 Ceph 쓰기 작업이 순차적으로 발생되도록 ACID 트랜잭션의 격리 부분을 제공합니다. 반면 OSD 복제 또는 삭제 코딩 기능은 ACID 트랜잭션의 내구성 구성 요소를 제공합니다. ObjectStore 는 스토리지 미디어에 대한 하위 수준 인터페이스이므로 성능 통계도 제공합니다.

Ceph는 데이터 저장을 위한 몇 가지 구체적인 방법을 구현합니다.

- bluestore: 오브젝트 데이터를 저장하는 원시 블록 장치를 사용하는 프로덕션 등급 구현입니다.
- Memstore: RAM에서 직접 읽기/쓰기 작업을 테스트하는 개발자 구현.
- K/V 스토어: Ceph에서 키/값 데이터베이스를 사용하기 위한 내부 구현입니다.

관리자는 일반적으로 BlueStore 만 주소이므로 다음 섹션에서는 이러한 구현만 더 자세히 설명합니다.

2.10. CEPH BLUESTORE

bluestore 는 Ceph를 위한 차세대 스토리지 구현입니다. 스토리지 장치 시장에는 이제 SSD 또는 PCI Express 또는 NVMe를 통한 SSD 및 비휘발성 메모리가 포함되어 있으므로 Ceph에서 사용하는 것은 FileStore 스토리지 구현의 몇 가지 제한 사항을 보여줍니다. FileStore 는 SSD 및 NVMe 스토리지를 원활하게하기 위해 많은 개선사항을 가지고 있지만 다른 제한 사항은 유지됩니다. 그 중에서 배치 그룹을 늘리면 컴퓨팅 비용이 많이 들고 이중 쓰기 페널티는 그대로 유지됩니다. 반면, FileStore 는 블록 장치의 파일 시스템과 상호 작용하며 BlueStore 는 이러한 간접성을 제거하고 개체 스토리지에 원시 블록 장치를

직접 사용합니다. bluestore 는 k/v 데이터베이스에 대해 작은 파티션에서 매우 가벼운 BlueFS 파일 시스템을 사용합니다. bluestore 는 배치 그룹, 오브젝트 및 파일 XATTR을 나타내는 파일인 배치 그룹을 나타내는 디렉터리의 패러다임을 제거합니다. 또한 bluestore는 FileStore 의 이중 쓰기 저하를 제거하므로 쓰기 작업은 대부분의 워크로드에서 BlueStore 와 거의 두 배 빠릅니다.

bluestore는 다음과 같은 데이터를 저장합니다.

- 오브젝트 데이터: BlueStore 에서 Ceph는 오브젝트를 원시 블록 장치에 직접 블록으로 저장합니다. 오브젝트 데이터를 저장하는 원시 블록 장치의 일부에는 파일 시스템이 포함되어 있지 않습니다. 파일 시스템 해제는 간접적인 계층을 제거하여 성능을 향상시킵니다. 그러나 대부분의 BlueStore 성능 개선 사항은 블록 데이터베이스와 write-ahead 로그에서 비롯됩니다.

- 블록 데이터베이스: BlueStore 에서 블록 데이터베이스는 일관성 을 보장하기 위해 개체의 의미를 처리합니다. 오브젝트의 고유 식별자는 블록 데이터베이스의 키입니다. 블록 데이터베이스의 값은 저장된 오브젝트 데이터, 개체의 배치 그룹 및 개체 메타데이터를 참조하는 일련의 블록 주소로 구성됩니다. 블록 데이터베이스는 오브젝트 데이터를 저장하는 동일한 원시 블록 장치의 BlueFS 파티션에 있거나 일반적으로 기본 블록 장치가 하드 디스크 드라이브이고 SSD 또는 NVMe가 성능을 향상시킬 때 별도의 블록 장치에 존재할 수 있습니다. 블록 데이터베이스는 FileStore 에 비해 여러 개선사항을 제공합니다. 즉 BlueStore 의 키/값 의미는 파일 시스템 XATTR의 제한으로 인한 영향을 받지 않습니다. bluestore는 FileStore 의 경우와 같이 한 디렉터리에서 다른 디렉터리로 파일을 이동하는 오버헤드 없이 블록 데이터베이스 내의 다른 배치 그룹에 오브젝트를 빠르게 할당할 수 있습니다. bluestore 에는 새로운 기능도 도입되었습니다. 블록 데이터베이스는 저장된 오브젝트 데이터 및 해당 메타데이터의 체크섬을 저장할 수 있으므로 각 읽기에 대한 전체 데이터 체크섬 작업을 허용하므로 비트 회전을 감지하는 데 주기적인 스크럽보다 효율적입니다. bluestore 는 개체를 압축할 수 있으며 블록 데이터베이스는 개체를 압축하는 데 사용되는 알고리즘을 저장할 수 있으며 읽기 작업에서 압축을 풀기 위한 적절한 알고리즘을 선택합니다.

- write-ahead Log: BlueStore 에서 write-ahead 로그는 FileStore 의 저널링 기능과 유사하게 Atomicity 를 보장합니다. FileStore 와 마찬가지로BlueStore 는 각 트랜잭션의 모든 측면을 기록합니다. 그러나 BlueStore 쓰기 로그 또는 WAL은 이 기능을 동시에 수행할 수 있으므로 FileStore 의 이중 쓰기 저하가 제거됩니다. 결과적으로 BlueStore 는 대부분의 워크로드에 대한 쓰기 작업에 FileStore 만큼 거의 두 배 빠릅니다. bluestore는 개체 데이터를 저장하기 위해 동일한 장치에 WAL을 배포하거나 일반적으로 기본 블록 장치가 하드 디스크 드라이브이고 SSD 또는 NVMe가 성능을 향상시킬 때 다른 장치에 WAL을 배포할 수 있습니다.

참고

별도의 장치가 기본 스토리지 장치보다 빠른 경우 블록 데이터베이스 또는 미리 쓰기 로그를 별도의 블록 장치에 저장하는 것이 유용합니다. 예를 들어 SSD 및 NVMe 장치는 일반적으로 HDD보다 빠릅니다. 블록 데이터베이스와 별도의 장치에 WAL을 배치하면 워크로드의 차이로 인해 성능상의 이점이 있을 수 있습니다.

2.11. CEPH 자체 관리 작업

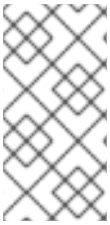
Ceph 클러스터는 자동으로 많은 자체 모니터링 및 관리 작업을 수행합니다. 예를 들어 Ceph OSD는 클러스터 상태를 확인하고 Ceph 모니터에 다시 보고할 수 있습니다. CRUSH를 사용하여 OSD 세트에 오브젝트를 배치 그룹 및 배치 그룹에 할당하면 Ceph OSD에서 CRUSH 알고리즘을 사용하여 클러스터를 재조정하거나 OSD 실패를 동적으로 복구할 수 있습니다.

2.12. CEPH 하트비트

Ceph OSD는 클러스터에 참여하고 해당 상태의 Ceph 모니터에 보고합니다. 가장 낮은 수준에서 Ceph OSD 상태는 실행 중이며 Ceph 클라이언트 요청을 서비스할 수 있는지 여부를 반영합니다. Ceph OSD가 다운 되어 Ceph 스토리지 클러스터에서 이 상태는 Ceph OSD의 오류를 나타낼 수 있습니다. 예를 들어 Ceph OSD가 실행되고 있지 않으면 Ceph OSD가 다운 되었다고 알릴 수 없습니다. Ceph 모니터는 Ceph OSD 데몬을 주기적으로 ping하여 실행 중인지 확인할 수 있습니다. 그러나 하트비트링은 Ceph OSD를 통해 인접한 OSD가 다운 되었는지 확인하고 클러스터 맵을 업데이트하고 Ceph 모니터에 보고할 수 있습니다. 따라서 Ceph 모니터는 경량의 프로세스를 유지할 수 있습니다.

2.13. CEPH 피어링

Ceph는 여러 OSD에 배치 그룹 복사본을 저장합니다. 배치 그룹의 각 사본에는 상태가 있습니다. 이러한 OSD "피어"는 서로 확인하여 PG의 각 사본의 상태에 동의했는지 확인합니다. 피어링 문제는 일반적으로 자동으로 해결됩니다.



참고

Ceph 모니터가 배치 그룹을 저장하는 OSD의 상태에 동의하면 배치 그룹에 최신 콘텐츠가 있는 것은 아닙니다.

Ceph가 작동하는 OSD 세트에 배치 그룹을 저장할 때 이 그룹을 기본 ,2 차 등으로 참조합니다. 규칙에 따라 기본 OSD는 동작 집합의 첫 번째 OSD입니다. 배치 그룹의 첫 번째 사본을 저장하는 기본 은 해당 배치 그룹에 대한 피어 프로세스를 조정하는 역할을 합니다. 기본 OSD는 지정된 배치 그룹의 개체에 대한 클라이언트 시작 쓰기를 기본 설정으로 허용하는 유일한 OSD입니다.

액티비스 세트는 배치 그룹을 저장하는 일련의 OSD입니다. 활동 세트는 현재 배치 그룹을 담당하는 Ceph OSD 데몬 또는 일부 epoch의 특정 배치 그룹을 담당하는 Ceph OSD 데몬을 참조할 수 있습니다.

Acting Set 의 일부인 Ceph OSD 데몬이 항상 작동하지 않을 수 있습니다. 액터링 세트의 OSD가 가동 중인 경우 Up Set 의 일부입니다. OSD가 실패하면 Ceph에서 다른 Ceph OSD로 PG를 다시 매핑할 수 있기 때문에 Up Set 은 중요한 차이점입니다.

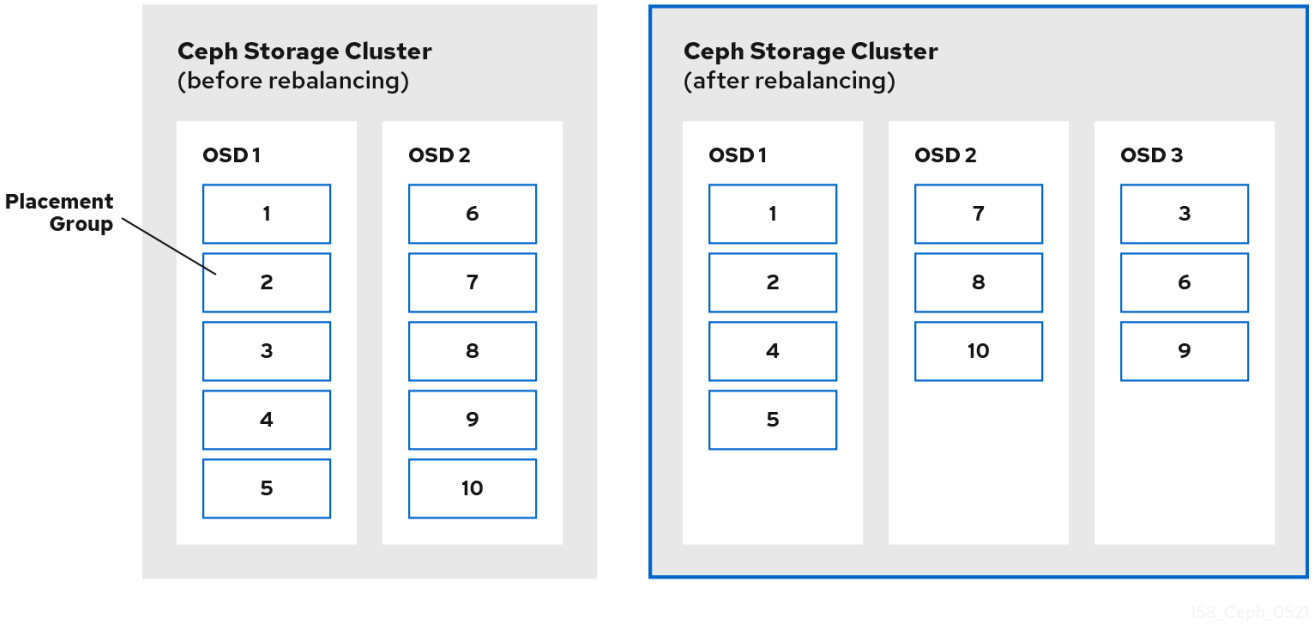
참고

osd.25,osd.32 및 **osd.61**, 첫 번째 **OSD, osd.25** 가 포함된 **PG**에 대한 행위 세트에서는 기본 설정 입니다. 해당 **OSD**가 실패하면 보조,**osd.32** 가 기본 이 되고 **Ceph**는 **Up Set** 에서 **osd.25** 를 제거합니다.

2.14. CEPH 재조정 및 복구

관리자가 **Ceph OSD**를 **Ceph** 스토리지 클러스터에 추가하면 **Ceph**가 클러스터 맵을 업데이트합니다. 수정된 클러스터 맵이 **CRUSH** 계산에 대한 입력을 변경하므로 클러스터 맵에 대한 변경 사항도 개체 배치를 변경합니다. **CRUSH**는 데이터를 균등하게 배치하지만 의사를 임의로 배치합니다. 따라서 관리자가 새 **OSD**를 추가할 때 적은 양의 데이터만 이동합니다. 데이터 양은 일반적으로 새 **OSD**의 수를 클러스터의 총 데이터 양으로 나눈 값입니다. 예를 들어 **OSD**가 **50**개인 클러스터에서는 **OSD**를 추가할 때 데이터의 **1/50**초 또는 **2%**가 이동할 수 있습니다.

다음 다이어그램에서는 다이어그램의 기존 **OSD, OSD 1** 및 **2**에서 다이어그램의 새 **OSD**인 **OSD 3**으로 일부 **PG**를 마이그레이션하는 리밸런싱 프로세스를 보여줍니다. 리밸런싱에도 **CRUSH**가 안정적입니다. 대부분의 배치 그룹은 원래 구성에 남아 있으며 각 **OSD**의 용량이 추가되므로 클러스터 재조정 후 새 **OSD**에서 로드 급증할 수 없습니다.



2.15. CEPH 데이터 무결성

Ceph는 데이터 무결성 유지의 일환으로 잘못된 디스크 섹터와 비트 회전으로부터 보호할 수 있는 다양한 메커니즘을 제공합니다.

- 스크럽: **Ceph OSD** 데몬을 통해 배치 그룹 내에서 오브젝트를 스크럽할 수 있습니다. 즉,

Ceph OSD 데몬은 하나의 배치 그룹의 오브젝트 메타데이터를 다른 OSD에 저장된 배치 그룹의 복제본과 비교할 수 있습니다. 스크럽핑은 일반적으로 매일 수행되며 버그 또는 스토리지 오류가 발생합니다. 또한 Ceph OSD 데몬에서는 오브젝트의 데이터를 비트 단위로 비교하여 더 심층적인 스크럽을 수행합니다. 딥 스크럽-은 일반적으로 weekly-finds bad 섹터는 광경에서 명확하지 않은 드라이브의 잘못된 섹터를 찾습니다.

- *CRC 검사: Red Hat Ceph Storage 5에서 BlueStore 를 사용할 때 Ceph는 쓰기 작업에서 CRC(Relical redundancy Check)를 수행하여 데이터 무결성을 보장할 수 있습니다. 그런 다음 block 데이터베이스에 CRC 값을 저장합니다. Ceph는 읽기 작업에서 CRC 값을 블록 데이터베이스에서 검색하고 검색된 데이터의 생성된 CRC와 비교하여 데이터 무결성을 즉시 보장할 수 있습니다.*

2.16. CEPH 고가용성

CRUSH 알고리즘에서 사용하는 높은 확장성 외에도 Ceph는 고가용성도 유지해야 합니다. 즉, Ceph 클라이언트는 클러스터가 성능 저하 상태에 있거나 모니터가 실패하는 경우에도 데이터를 읽고 쓸 수 있어야 합니다.

2.17. CEPH 모니터 클러스터링

Ceph 클라이언트가 데이터를 읽거나 쓰기 전에 클러스터 맵의 최신 사본을 얻으려면 Ceph Monitor에 문의해야 합니다. Red Hat Ceph Storage 클러스터는 단일 모니터로 작동할 수 있지만 이로 인해 단일 장애 지점이 발생합니다. 즉, 모니터가 중단되면 Ceph 클라이언트가 데이터를 읽거나 쓸 수 없습니다.

향상된 안정성과 내결함성을 위해 Ceph는 모니터 클러스터를 지원합니다. Ceph 모니터, 대기 시간 및 기타 장애로 인해 하나 이상의 모니터가 클러스터의 현재 상태가 될 수 있습니다. 따라서 Ceph는 스토리지 클러스터 상태와 관련된 다양한 모니터 인스턴스 간에 동의해야 합니다. Ceph는 항상 대부분의 모니터와 Paxos 알고리즘을 사용하여 스토리지 클러스터의 현재 상태에 대한 모니터 간에 합의를 설정합니다. Ceph 모니터 노드에는 클럭 드리프트를 방지하기 위해 NTP가 필요합니다.

스토리지 관리자는 일반적으로 홀수의 모니터를 사용하여 Ceph를 배포하므로, 대다수를 결정하는 것이 효율적입니다. 예를 들어, 대다수는 1, 2:3, 3:5, 4:6 등일 수 있습니다.

3장. CEPH 클라이언트 구성 요소

Ceph 클라이언트는 데이터 스토리지 인터페이스를 제공하는 방식에 따라 크게 다릅니다. Ceph 블록 장치는 물리적 스토리지 드라이브와 마찬가지로 마운트되는 블록 스토리지를 제공합니다. Ceph 게이트웨이는 S3 준수 및 Swift 호환 RESTful 인터페이스와 자체 사용자 관리를 포함한 오브젝트 스토리지 서비스를 제공합니다. 그러나 모든 Ceph 클라이언트는 Reliable Autonomic Distributed Object Store (RADOS) 프로토콜을 사용하여 Red Hat Ceph Storage 클러스터와 상호 작용합니다.

모두 동일한 기본 요구 사항이 있습니다.

- Ceph 구성 파일 및 Ceph 모니터 주소입니다.
- 풀 이름입니다.
- 사용자 이름 및 시크릿 키의 경로입니다.

Ceph 클라이언트는 object-watch-notify 및 striping과 같은 몇 가지 유사한 패턴을 따릅니다. 다음 섹션에서는 Ceph 클라이언트에서 사용되는 RADOS, librados 및 공통 패턴에 대해 설명합니다.

3.1. 사전 요구 사항

- 분산 스토리지 시스템에 대한 기본적인 이해.

3.2. CEPH 클라이언트 네이티브 프로토콜

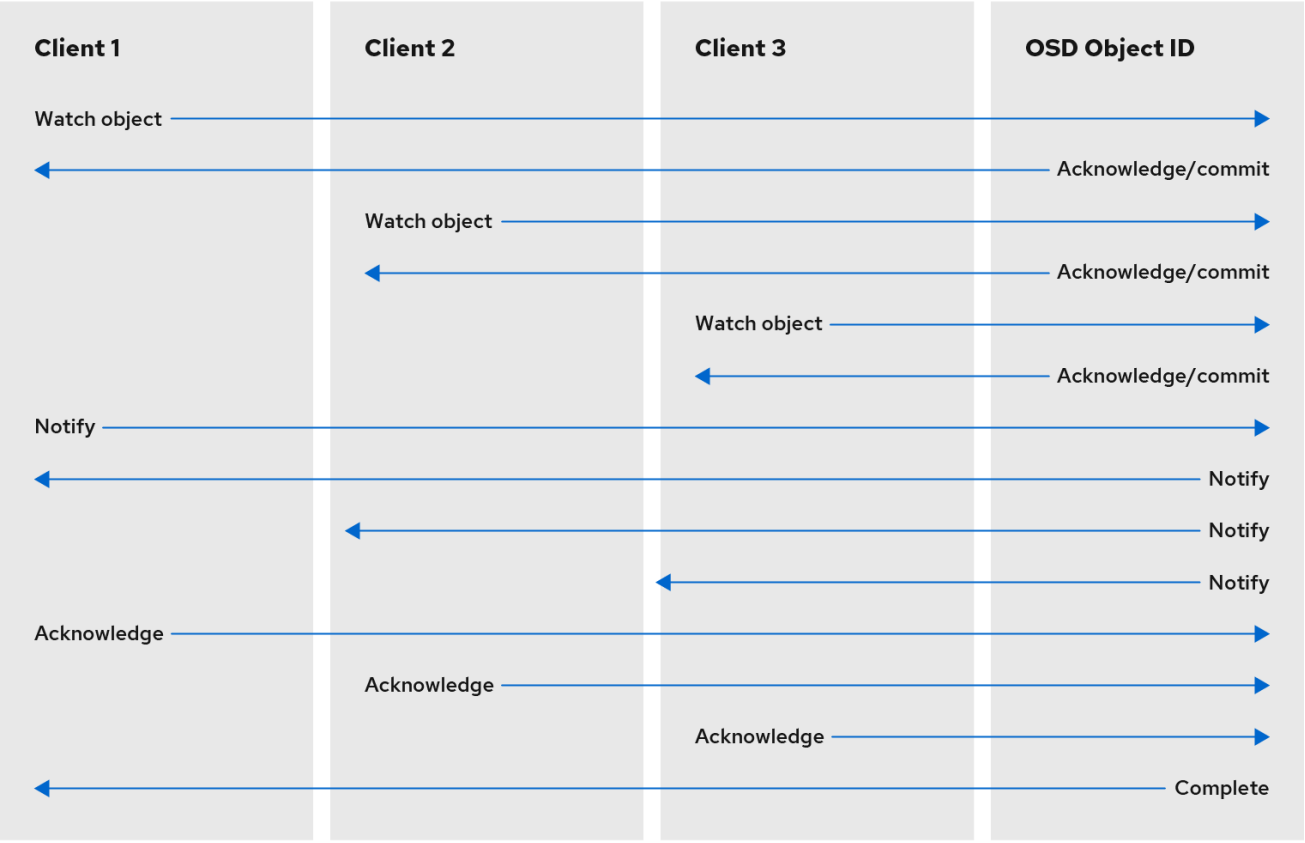
최신 애플리케이션에는 비동기 통신 기능이 있는 간단한 개체 스토리지 인터페이스가 필요합니다. Ceph Storage Cluster는 비동기 통신 기능을 통해 간단한 개체 스토리지 인터페이스를 제공합니다. 인터페이스는 클러스터 전체의 오브젝트에 직접 병렬 액세스를 제공합니다.

- 작업 풀
- 스냅샷

- 읽기/쓰기 오브젝트
  - 생성 또는 제거
  - 전체 오브젝트 또는talPodAutoscaler 범위
  - append 또는 Truncate
- 만들기/Set/Get/Remove XATTRs
- 만들기/Set/Get/Remove Key/Value pairs
- 복합 작업 및 이중 ack 의미

3.3. CEPH 클라이언트 오브젝트 감시 및 알림

Ceph 클라이언트는 오브젝트에 지속적인 관심을 등록하고 세션을 기본 OSD에 유지할 수 있습니다. 클라이언트는 모든 감시자에게 알림 메시지 및 페이로드를 보내고 감시자가 알림을 수신하면 알림을 받을 수 있습니다. 이를 통해 클라이언트는 모든 개체를 동기화/연동 채널로 사용할 수 있습니다.



3.4. CEPH 클라이언트 MANDATORY EXCLUSIVE LOCKS

*필수 **Exclusive Locks**는 여러 마운트가 있는 경우 **RBD**를 단일 클라이언트에 고정하는 기능입니다. 이렇게 하면 마운트된 여러 클라이언트가 동일한 개체에 쓰기를 시도할 때 쓰기 충돌 상황을 해결할 수 있습니다. 이 기능은 이전 섹션에서 설명하는 **object-watch-notify** 를 기반으로 합니다. 따라서 한 클라이언트가 먼저 개체에 배타적 잠금을 설정하는 경우 다른 마운트된 클라이언트는 먼저 피어가 쓰기 전에 개체에 잠금을 배치했는지 확인합니다.*

*이 기능을 활성화하면 특히 스냅샷 생성/삭제 와 같은 작업 중에 내부 **RBD** 구조를 변경하는 경우 한 번에 하나의 클라이언트만 **RBD** 장치를 수정할 수 있습니다. 또한 실패한 고객에 대한 보호 기능도 제공합니다. 예를 들어 가상 머신이 응답하지 않고 다른 위치에서 동일한 디스크로 복사본을 시작하는 경우 첫 번째 시스템은 **Ceph**에서 블랙리스트로 지정되어 새 머신을 손상시킬 수 없습니다.*

*필수 **Exclusive Locks**는 기본적으로 활성화되어 있지 않습니다. 이미지를 생성할 때 **--image-feature** 매개변수를 사용하여 명시적으로 활성화해야 합니다.*

예제

```
[root@mon ~]# rbd create --size 102400 mypool/myimage --image-feature 5
```

여기에서 숫자 **5** 는 **1** 과 **4** 의 총합이며, 여기서 **1** 은 계층 지원을 가능하게 하고, **4** 는 배타적 잠금 지원을 가능하게 합니다. 따라서 위의 명령은 **100GB rbd** 이미지를 만들고 계층 지정 및 배타적 잠금을 활성화합니다.

필수 **Exclusive Locks**는 오브젝트 맵 의 전제 조건이기도 합니다. 독점 잠금 지원을 활성화하지 않으면 개체 맵 지원을 활성화할 수 없습니다.

필수 **Exclusive Locks**는 미러링에 대한 기본 작업도 수행합니다.

3.5. CEPH 클라이언트 오브젝트 맵

오브젝트 맵은 클라이언트가 **rbd** 이미지에 쓸 때 **RADOS** 오브젝트 백업의 존재를 추적하는 기능입니다. 쓰기가 발생하면 해당 쓰기가 지원 **RADOS** 오브젝트 내의 오프셋으로 변환됩니다. 오브젝트 맵 기능이 활성화되면 **RADOS** 오브젝트가 추적됩니다. 따라서 개체가 실제로 존재하는지 알 수 있습니다. 오브젝트 맵은 **librbd** 클라이언트에서 메모리 내 유지되므로, **OSD**가 존재하지 않는 개체에 대해 쿼리하지 못할 수 있습니다. 즉, 개체 맵은 실제로 존재하는 개체의 인덱스입니다.

오브젝트 맵은 **viz**의 특정 작업에 유용합니다.

- 크기 조정
- 내보내기
- 복사
- **flatten**
- **delete**
- 읽기

축소 크기 조정 작업은 후행 개체가 삭제된 부분 삭제와 같습니다.

*내보내기 작업은 **RADOS**에서 요청할 오브젝트를 확인합니다.*

복사 작업은 어떤 개체가 존재하는지 알고 복사해야 합니다. 잠재적으로 수백 및 수천 개의 가능한 오브젝트를 반복할 필요가 없습니다.

***flatten** 작업은 복제본을 상위 **i.e**에서 분리할 수 있도록 모든 상위 오브젝트에 대한 복사 작업을 수행하여 하위 복제본에서 상위 스냅샷에 대한 참조를 제거할 수 있습니다. 따라서 모든 잠재적인 오브젝트 대신 **copy-up**은 존재하는 개체에 대해서만 수행됩니다.*

삭제 작업은 이미지에 존재하는 오브젝트만 삭제합니다.

*읽기 작업은 알고 있는 개체에 대한 읽기가 존재하지 않습니다.**A read operation skips the read for objects it knows does not exist.***

*따라서 크기 조정, 축소, 내보내기, 복사, 병합 및 삭제와 같은 작업의 경우 영향을 받는 모든 **RADOS** 오브젝트에 대해 영향을 받는 모든 **RADOS** 오브젝트에 대한 작업을 실행해야 합니다. 개체 맵이 활성화된 상태에서 개체가 없으면 작업을 실행할 필요가 없습니다.*

*예를 들어 **1TB**의 스파스 **RBD** 이미지가 있는 경우 **RADOS** 개체를 수백 및 수천 개의 지원할 수 있습니다. 오브젝트 맵이 활성화되지 않은 삭제 작업에서는 이미지의 잠재적인 오브젝트 각각에 대해 오브젝트 작업을 제거해야 합니다. 그러나 오브젝트 맵이 활성화된 경우 존재하는 오브젝트에 대한 오브젝트 작업을 제거 만 하면 됩니다.*

*오브젝트 맵은 실제 개체가 없지만 부모의 객체를 얻는 복제본에 대해 중요합니다. 복제된 이미지가 있으면 처음에 복제에 오브젝트가 없으며 모든 읽기가 상위로 리디렉션됩니다. 따라서 오브젝트 맵 없이 읽기를 개선할 수 있습니다. 먼저 복제에 대해 **OSD**에 대한 읽기 작업을 실행해야 합니다. 이 작업이 실패하면 오브젝트 맵이 활성화된 상태에서 부모에 대한 다른 읽기 권한이 발행됩니다. 알고 있는 개체에 대한 읽기가 존재하지 않습니다.*

*오브젝트 맵은 기본적으로 활성화되어 있지 않습니다. 이미지를 생성할 때 **--image-features** 매개변수를 사용하여 명시적으로 활성화해야 합니다. 또한 필수 제거 잠금 은 오브젝트 맵 의 전제 조건입니다. 독점 잠금 지원을 활성화하지 않으면 개체 맵 지원을 활성화할 수 없습니다. 이미지를 생성할 때 오브젝트 맵 지원을 활성화하려면 다음을 실행합니다.*

```
[root@mon ~]# rbd -p mypool create myimage --size 102400 --image-features 13
```

여기에서 숫자 **13** 은 **1,4** 및 **8** 의 요약이며, 여기서 **1** 은 계층화를 지원하며, **4** 는 배타적 잠금 지원을 가능하게 하고 **8** 은 개체 맵 지원을 활성화합니다. 따라서 위의 명령은 **100GB rbd** 이미지를 만들고 계층화, 독점 잠금 및 개체 맵을 활성화합니다.

3.6. CEPH 클라이언트 데이터 스트리핑

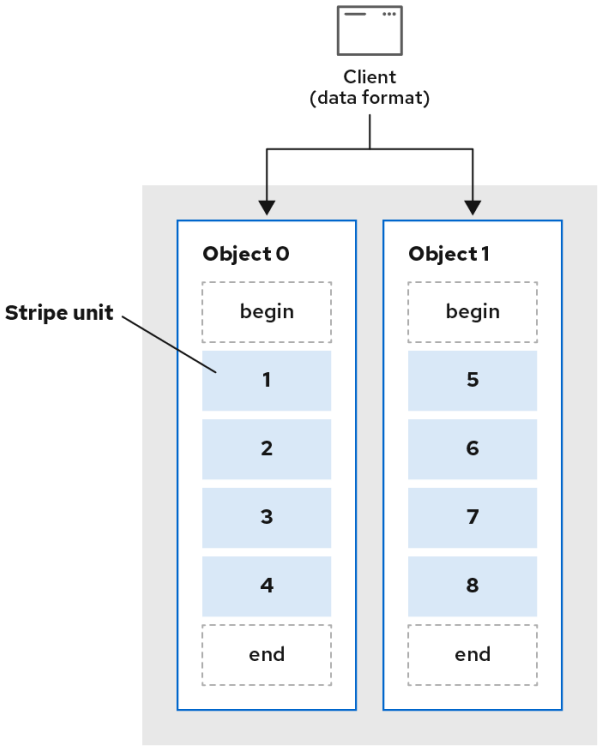
스토리지 장치에는 처리량 제한이 있으며 이는 성능 및 확장성에 영향을 미칩니다. 따라서 스토리지 시스템은 여러 스토리지 장치에 걸쳐 연속 정보를 스트라이핑하여 처리량과 성능을 향상시키는 경우가 많습니다. 가장 일반적인 형태의 데이터 스트라이핑은 **RAID**에서 가져온 것입니다. **Ceph**의 스트리핑과 가장 유사한 **RAID** 유형은 **RAID 0** 또는 **'striped volume'**입니다. **Ceph**의 스트라이핑은 **n-way RAID** 미러링의 신뢰성과 빠른 복구 방법인 **RAID 0** 스트라이핑의 처리량을 제공합니다.

Ceph는 **Ceph** 블록 장치, **Ceph** 파일 시스템 및 **Ceph Object Storage**의 세 가지 유형의 클라이언트를 제공합니다. **Ceph Client**는 해당 데이터를 블록 장치 이미지, **RESTful** 오브젝트, **CephFS** 파일 시스템 디렉터리와 같은 표현 형식에서 **Ceph Storage** 클러스터의 스토리지 오브젝트로 변환합니다.

작은 정보

Ceph Storage Cluster의 **Ceph** 저장소가 제거되지 않습니다. **Ceph Object Storage, Ceph** 블록 장치 및 **Ceph Filesystem**은 여러 **Ceph Storage Cluster** 오브젝트에 걸쳐 데이터를 스트라이프합니다. **librados** 를 사용하여 **Ceph** 스토리지 클러스터에 직접 쓰는 **Ceph** 클라이언트는 이러한 이점을 얻기 위해 스트립링 및 병렬 **I/O**를 수행해야 합니다.

가장 간단한 **Ceph** 스트리핑 형식에는 개체의 스트라이프 수가 **1**개 포함됩니다. **Ceph Client**는 개체가 최대 용량에 있을 때까지 **Ceph Storage Cluster** 오브젝트에 스트라이프 단위를 작성한 다음 추가 데이터 스트라이프를 위해 다른 오브젝트를 만듭니다. 가장 간단한 형태의 스트라이핑은 작은 블록 장치 이미지, **S3** 또는 **Swift** 오브젝트에 충분할 수 있습니다. 그러나 이 간단한 양식에서는 배치 그룹 간에 데이터를 배포할 수 있는 **Ceph**의 기능을 최대한 활용하지 않으므로 성능이 크게 향상되지 않습니다. 다음 다이어그램은 가장 간단한 스트라이핑 양식을 보여줍니다.

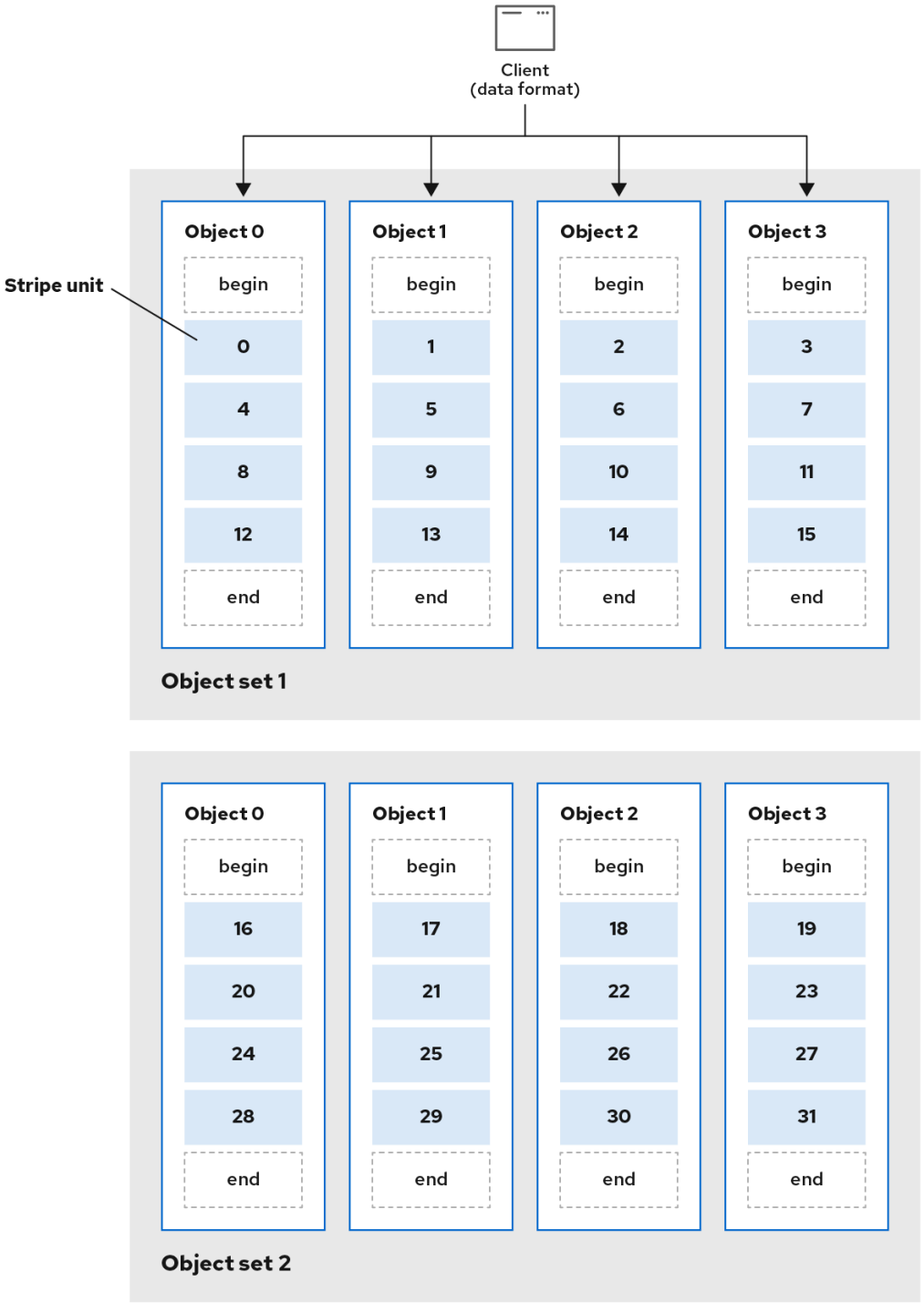


*큰 이미지 크기, 큰 **S3** 또는 **Swift** 오브젝트를 예상하는 경우 오브젝트 세트 내의 여러 오브젝트에 대한 클라이언트 데이터를 제거하여 상당한 읽기/쓰기 성능 향상을 볼 수 있습니다. 클라이언트가 스트라이프 단위를 병렬로 해당 개체에 쓸 때 상당한 쓰기 성능이 발생합니다. 오브젝트는 다른 배치 그룹에 매핑되고 다른 **OSD**에 추가로 매핑되므로 각 쓰기 속도는 최대 쓰기 속도에서 병렬로 수행됩니다. 단일 디스크에 대한 쓰기는 검색당 **6ms**, 예를 들어 **100MB/s**와 같은 하나의 장치의 대역폭에 의해 헤드 이동에 의해 제한됩니다. **Ceph**는 다양한 배치 그룹과 **OSD**에 매핑되는 여러 오브젝트를 통해 쓰기를 분산하여 드라이브당 검색 수를 줄이고 여러 드라이브의 처리량을 결합하여 쓰기 또는 읽기 속도를 훨씬 빠르게 수행할 수 있습니다.*

참고

*스트라이핑은 오브젝트 복제본과는 독립적입니다. **CRUSH**가 **OSD**에서 오브젝트를 복제하므로 스트라이프가 자동으로 복제됩니다.*

*다음 다이어그램에서 클라이언트 데이터는 **4**개의 오브젝트로 구성된 개체 집합(다음 다이어그램의 개체 세트 **1**)에 걸쳐 스트라이핑됩니다. 여기서 첫 번째 스트라이프 단위는 **0** 개체 **0**의 스트라이프 단위 이고, 네 번째 스트라이프 단위는 개체 **3**의 스트라이프 단위 **3** 입니다. 네 번째 스트라이프를 작성한 후 클라이언트는 개체 세트가 가득 찼는지 여부를 결정합니다. 개체 세트가 가득 차 있지 않으면 클라이언트가 첫 번째 개체에 다시 스트라이프 작성을 시작하고 다음 다이어그램에서 개체 **0** 을 참조하십시오. 개체 세트가 가득 차면 클라이언트는 새 개체 세트를 만들고 다음 다이어그램에서 오브젝트 세트 **2** 를 참조하며 다음 다이어그램에서 첫 번째 개체의 스트라이프 단위 **16**을 사용하여 첫 번째 스트라이프 단위로 쓰기를 시작합니다. 아래 다이어그램에서 **4** 를 참조하십시오.*



Ceph 스트라이프 데이터 방법을 결정하는 세 가지 중요한 변수는 다음과 같습니다.

- *오브젝트 크기: Ceph Storage 클러스터의 개체에는 2MB 또는 4MB와 같은 최대 구성 가능한 크기가 있습니다. 개체 크기는 많은 스트라이프 단위를 수용할 수 있을 만큼 커야 하며 스트라이프 단위의 여러 개여야 합니다.*

중요

***Red Hat**은 안전한 **16MB**의 값을 권장합니다.*

- *스트라이프 **Width:** 스트립 에는 구성 가능한 단위 크기(예: **64KB**)가 있습니다. **Ceph Client**는 마지막 스트라이프 단위를 제외하고 개체에 쓰는 데이터를 동일한 크기의 스트라이프 단위로 나눕니다. 스트라이프 너비는 개체가 많은 스트라이프 단위를 포함할 수 있도록 오브젝트 크기의 일부여야 합니다.*

- *스트라이프 수: **Ceph Client**는 스트라이프 수로 결정되는 일련의 오브젝트에 대한 스트라이프 단위 시퀀스를 작성합니다. 일련의 개체를 개체 집합이라고 합니다. **Ceph Client**가 오브젝트 세트의 마지막 오브젝트에 기록한 후 오브젝트 세트의 첫 번째 오브젝트로 돌아갑니다.*

중요

클러스터를 프로덕션 환경에 배치하기 전에 구성 스트립의 성능을 테스트합니다. 이러한 스트라이핑 매개 변수를 변경하고 데이터를 개체에 쓸 수 없습니다.

***Ceph Client**가 데이터를 스트라이프 단위에 제거하고 개체에 스트라이프 단위를 매핑하면 **Ceph**의 **CRUSH** 알고리즘은 개체를 배치 그룹에 매핑하고, 오브젝트가 스토리지 디스크에 파일로 저장되기 전에 배치 그룹을 **Ceph OSD** 데몬에 매핑합니다.*

참고

*클라이언트는 단일 풀에 쓰기 때문에 개체에 스트립된 모든 데이터가 동일한 풀의 배치 그룹에 매핑됩니다. 따라서 동일한 **CRUSH** 맵과 동일한 액세스 제어를 사용합니다.*

4장. CEPH 온-WIRE 암호화

Red Hat Ceph Storage 4 이상부터 네트워크를 통해 모든 Ceph 트래픽의 암호화를 활성화할 수 있습니다. v2의 보안 모드 설정은 Ceph 데몬과 Ceph 클라이언트 간의 통신을 암호화하여 포괄적인 암호화를 제공합니다.

Ceph의 유선 프로토콜의 두 번째 버전인 msgr2 에는 다음과 같은 몇 가지 새로운 기능이 포함되어 있습니다.

- 네트워크를 통해 이동하는 모든 데이터를 암호화하는 보안 모드입니다.
- 인증 페이로드 적용 개선 사항.
- 광고 및 협상의 개선 사항은 다음과 같습니다.

Ceph 데몬은 여러 포트에 바인드되어 legacy, v1-compatible 및 새로운 v2-compatible, Ceph 클라이언트가 동일한 스토리지 클러스터에 연결할 수 있습니다. Ceph Monitor 데몬에 연결된 Ceph 클라이언트 또는 기타 Ceph 데몬은 가능한 경우 v2 프로토콜을 먼저 사용하려고 하지만 그렇지 않은 경우 레거시 v1 프로토콜을 사용합니다. 기본적으로 v1 및 v2 프로토콜 모두 활성화되어 있습니다. 새로운 v2 포트는 3300이며 레거시 v1 포트는 기본적으로 6789입니다.

msgr2 프로토콜은 다음 두 가지 연결 모드를 지원합니다.

- crc
  - wget으로 연결을 설정하는 경우 강력한 초기 인증을 제공합니다.
  - 비트 플립으로부터 보호하기 위해 crc32c 무결성 검사를 제공합니다.
  - 악의적인 메시지 가로채기(man-in-the-middle) 공격에 대한 보호를 제공하지 않습니다.
  - 도파이어가 모든 인증 후 트래픽을 볼 수 없도록 하지 않습니다.

- *보안*
  - ***wget*으로 연결을 설정하는 경우 강력한 초기 인증을 제공합니다.**
  - *모든 인증 후 트래픽에 대한 전체 암호화를 제공합니다.*
  - *암호화 무결성 검사를 제공합니다.*

*기본 모드는 **crc** 입니다.*

*암호화 오버헤드를 포함하도록 **Red Hat Ceph Storage** 클러스터를 계획할 때 클러스터 **CPU** 요구 사항을 고려해야 합니다.*

중요

*현재 보안 모드 사용은 **Red Hat Enterprise Linux**에서 **CephFS** 및 **krbd** 와 같은 **Ceph** 커널 클라이언트에서 지원됩니다. 보안 모드를 사용하는 **Ceph** 클라이언트에서 **OpenStack Nova, Glance, Cinder**와 같은 **librbd** 를 사용할 수 있습니다.*

주소 변경 사항

이전 프로토콜의 두 버전이 동일한 스토리지 클러스터에 공존하는 경우 주소 형식이 변경되었습니다.

- *이전 주소 형식은 **IP_ADDR:PORT/CLIENT_ID (예: 1.2.3.4:5678/91011)**입니다.*
- *새 주소 형식은 **PROTOCOL_VERSION:IP_ADDR:PORT/CLIENT_ID (예: v2:1.2.3.4:5678/91011)**입니다. 여기서 **PROTOCOL_VERSION** 은 **v1** 또는 **v2** 일 수 있습니다.*

*이제 **Ceph** 데몬이 여러 포트에 바인드되므로 데몬은 단일 주소 대신 여러 주소를 표시합니다. 다음은 모니터 맵 덤프의 예입니다.*

```
epoch 1
fsid 50fcf227-be32-4bcb-8b41-34ca8370bd17
last_changed 2021-12-12 11:10:46.700821
```

```
created 2021-12-12 11:10:46.700821
min_mon_release 14 (nautilus)
0: [v2:10.0.0.10:3300/0,v1:10.0.0.10:6789/0] mon.a
1: [v2:10.0.0.11:3300/0,v1:10.0.0.11:6789/0] mon.b
2: [v2:10.0.0.12:3300/0,v1:10.0.0.12:6789/0] mon.c
```

또한 **mon_host** 구성 옵션 및 명령줄에서 **-m** 을 사용하여 주소를 지정하면 새 주소 형식이 지원됩니다.

연결 단계

암호화된 연결을 수행하기 위한 네 가지 단계가 있습니다.

배너

연결 시 클라이언트와 서버가 배너를 보냅니다. 현재 **Ceph** 배너는 **ceph 0n** 입니다.

인증 교환

전송 또는 수신되는 모든 데이터는 연결 기간 동안 프레임에 포함됩니다. 서버는 인증이 완료되었는지와 연결 모드가 될지를 결정합니다. 프레임 형식이 수정되었으며 사용 중인 인증 플래그에 따라 세 가지 형식으로 될 수 있습니다.

메시지 흐름 핸드셰이크 교환

피어는 서로를 식별하고 세션을 설정합니다. 클라이언트는 첫 번째 메시지를 전송하고 서버는 동일한 메시지로 응답합니다. 클라이언트가 잘못된 데몬과 통신하면 서버를 닫을 수 있습니다. 새로운 세션의 경우 클라이언트와 서버는 메시지 교환을 진행합니다. 클라이언트 쿠키는 세션을 식별하는 데 사용되며 기존 세션에 다시 연결할 수 있습니다.

메시지 교환

클라이언트와 서버는 연결이 닫힐 때까지 메시지를 교환하기 시작합니다.

추가 리소스

- **msgr2** 프로토콜 활성화에 대한 자세한 내용은 **Red Hat Ceph Storage Data Security and Hardening Guide** 를 참조하십시오.