?? 4. ZFS ???? (ZFS Datasets)

일반 파일시스템에서는 파티션을 만들어 다양한 유형의 데이터를 분리하고 , 파티션에 다양한 최적화를 적용하고 , 파티션이 사용할 수 있는 공간의 양을 제한할 수 있습니다 . 각 파티션은 디스크에서 특정 양의 공간을 할당받습니다 . 우리 모두 그런 경험이 있습니다 . 우리는 다음 달 , 내년 , 5년 후 이 시스템의 각 파티션에 얼마나 많은 디스크 공간이 필요할지 추측합니다 . 그런데 미래를 내다보면 각 파티션에 할당하기로 결정한 공간의 양이 잘못되었을 가능성이 높습니다 . 모든 데이터를 저장할 공간이 충분하지 않은 파티션은 디스크를 추가하거나 데이터를 이동해야 하므로 시스템 관리가 복잡해집니다 . 파티션에 공간이 너무 많으면 다른 곳에 두는 것이 더 나을 데이터를 파티션에 버리게 됩니다 . 루카스의 UFS2 시스템 중 하나 이상은 */home*의 어딘가에 대한 심볼릭 링크로 */usr/*포트를 가지고 있습니다 . Jude는 보통 */usr/local/var*에 */var*의 일부를 저장합니다 .

ZFS는 여유 공간을 풀링하여 일반적인 파일 시스템에서는 불가능한 파티션의 유연성을 제공함으로써 이 문제를 해결합니다 . 생성하는 각 ZFS 데이터 세트는 그 안에 파일을 저장하는 데 필요한 공간만 사용합니다 . 각 데이터 세트는 풀의 모든 여유 공간에 액세스할 수 있으므로 파티션 크기에 대한 걱정을 덜 수 있습니다 . 6 장에서 설명한 대로 할당량으로 데이터 세트의 크기를 제한하거나 예약을 통해 최소한의 공간을 보장할 수 있습니다 .

일반 파일시스템은 별도의 파티션을 사용하여 다양한 유형의 데이터에 대해 서로 다른 정책과 최적화를 설정합니다 . */var*에는 로그와 데이터베이스처럼 자주 변경되는 파일이 들어 있습니다 . 루트 파일시스템은 성능보다 일관성과 안전성이 중요합니다 . */home*에서는 무엇이든 가능합니다 . 하지만 기존 파일시스템에 대한 정책을 한 번 설정하면 변경하기가 정말 어렵습니다 . UFS용 **tunefs(8)** 유틸리티를 사용하려면 파일시스템을 마운트 해제해야 변경할 수 있습니다 . 아이노드 수와 같은 일부 특성은 파일시스템이 생성된 후에는 변경할 수 없습니다 .

기존 파일시스템의 핵심 문제는 유연성 부족으로 귀결됩니다 . ZFS 데이터 세트는 거의 무한대로 유연합니다 .

????? (Datasets)

데이터 세트는 이름이 지정된 데이터 덩어리입니다 . 이 데이터는 파일 , 디렉터리 , 권한 등 기존 파일 시스템과 비슷할 수 있습니다 . 원시 블록 장치 , 다른 데이터의 사본 또는 디스크에 넣을 수 있는 모든 것일 수 있습니다 .

ZFS는 기존 파일시스템이 파티션을 사용하는 것과 마찬가지로 데이터세트를 사용합니다 . */usr*에 대한 정책과 */home*에 대한 별도의 정책이 필요하신가요 ? 각각 데이터세트를 만들면 됩니다 . iSCSI 대상에 대한 차단 장치가 필요하신가요 ? 바로 데이터 세트입니다 . 데이터 세트의 사본을 원하시나요 ? 그것도 또 다른 데이터 세트입니다 .

데이터 세트에는 계층적 관계가 있습니다 . 하나의 스토리지 풀이 각 최상위 데이터 세트의 부모입니다 . 각 데이터 세트는 하위 데이터 세트를 가질 수 있습니다 . 이 장에서 살펴보겠지만 데이터 세트는 부모로부터 많은 특성을 상속받습니다 .

모든 데이터 세트 작업은 **zfs(8)** 명령으로 수행합니다 . 이 명령에는 다양한 종류의 하위 명령이 있습니다 .

????? ?? (Dataset Types)

현재 ZFS에는 파일시스템 (filesystems), 볼륨 (volumes), 스냅샷 (snapshots), 복제본 (clones), 북마크 (bookmarks) 등 5가지 유형의 데이터 세트가 있습니다 .

파일 시스템 데이터 세트는 기존 파일 시스템과 유사합니다 . 파일과 디렉터리를 저장합니다 . ZFS 파일시스템에는 마운트 지점이 있으며 읽기 전용 , setuid 바이너리 제한 등과 같은 기존 파일시스템의 특성을 지원합니다 . 파일시스템 데이터세트에는 권한 , 파일 생성 및 수정을 위한 타임스탬프 , NFSv4 액세스 제어 플래그 , chflags(2) 등의 기타 정보도 포함됩니다 .

ZFS 볼륨 또는 **zvol**은 블록 장치입니다 . 일반 파일 시스템에서는 iSCSI용 파일 백업 파일 시스템이나 특수 목적의 UFS 파티션을 만들 수 있습니다 . ZFS에서 이러한 블록 디바이스는 파일과 디렉터리의 모든 오버헤드를 우회하여 기본 풀에 직접 상주합니다 . Zvol은 디스크 이미지를 마운트하는 데 사용되는 FreeBSD 메모리 장치를 건너뛰고 장치 노드를 가져옵니다 .

스냅샷은 특정 시점의 데이터 세트에 대한 읽기 전용 사본입니다 . 스냅샷을 사용하면 나중에 사용할 수 있도록 이전 버전의 파일시스템과 그 안에 있는 파일을 보존할 수 있습니다 . 스냅샷은 현재 파일 시스템과 스냅샷에 있는 파일 간의 차이에 따라 일정량의 공간을 사용합니다 .

클론은 기존 데이터 세트의 스냅샷을 기반으로 하는 새로운 데이터 세트로 , 파일 시스템을 포크할 수 있습니다 . 데이터 세트의 모든 항목에 대한 추가 복사본을 얻을 수 있습니다 . 프로덕션 웹 사이트가 포함된 데이터세트를 복제하여 프로덕션 사이트를 건드리지 않고도 해킹할 수 있는 사이트 사본을 만들 수 있습니다 . 복제본은 생성된 원본 스냅샷과의 차이점을 저장하는 데만 공간을 사용합니다 . 7장에서는 스냅샷 , 클론 및 북마크에 대해 다룹니다 .

??? ??? ??? ??? ?????? (Why Do I Want Datasets?)

당연히 데이터 세트가 필요합니다 . 디스크에 파일을 저장하려면 파일 시스템 데이터 세트가 필요합니다 . 그리고 */usr* 및 */var*와 같은 각 기존 유닉스 파티션에 대한 데이터 세트가 필요할 것입니다 . 하지만 ZFS를 사용하면 많은 데이터 세트가 필요합니다 . 아주 많은 데이터 세트가 필요합니다 . 파티션 수에 대한 하드코딩된 제한과 파티션의 유연성이 없는 기존 파일 시스템에서는 이런 일이 벌어질 수 없습니다 . 하지만 많은 데이터세트를 사용하면 데이터에 대한 통제력이 높아집니다 .

각 ZFS 데이터 세트에는 작동을 제어하는 일련의 속성이 있어 관리자가 데이터 세트의 작동 방식과 데이터 보호 수준을 제어할 수 있습니다 . 기존 파일 시스템에서와 마찬가지로 각 데이터세트를 정확하게 조정할 수 있습니다 . 데이터세트 속성은 풀 속성과 매우 유사하게 작동합니다 .

시스템 관리자는 개별 데이터 세트에 대한 제어 권한을 다른 사용자에게 위임하고 , 해당 사용자가 루트 권한 없이도 데이터 세트를 관리할 수 있도록 할 수 있습니다 . 조직에 여러 프로젝트 팀이 있는 경우 , 각 프로젝트 관리자에게 자신만의 공간을 주고 "여기 , 원하는 대로 정리해 보세요 ."라고 말할 수 있습니다 . 업무량을 줄여주는 것은 무엇이든 좋은 일입니다 .

복제 및 스냅샷과 같은 많은 ZFS 기능은 데이터 세트 단위로 작동합니다 . 데이터를 논리적 그룹으로 분리하면 조직을 지원하기 위해 이러한 ZFS 기능을 더 쉽게 사용할 수 있습니다 .

각각 다른 팀에서 관리하는 수십 개의 사이트가 있는 웹 서버를 예로 들어 보겠습니다 . 어떤 팀은 여러 사이트를 담당하는 반면 , 어떤 팀은 한 사이트만 담당합니다 . 어떤 사람들은 여러 팀에 소속되어 있습니다 . 전통적인 파일 시스템 모델을 따른다면 */webserver* 데이터 집합을 만들어 모든 것을 그 안에 넣고 그룹 권한과 **sudo(8)**로 액세스를 제어할 수 있습니다 . 수십 년 동안 이런 식으로 살아왔고 잘 작동하는데 왜

바꾸어야 할까요?

하지만 각 팀에 대한 데이터 집합을 만들고, 그 상위 데이터 집합 내에 각 사이트에 고유한 데이터 집합을 부여하면 가능성은 배가됩니다.

팀에서 테스트를 위해 웹 사이트의 사본이 필요하신가요? 복제하세요. 기존 파일 시스템에서는 사이트 디렉터리 전체를 복사해야 하므로 사이트에 필요한 디스크의 양이 두 배로 늘어나고 시간도 훨씬 더 오래 걸립니다. 복제본은 사이트 간의 차이점에 해당하는 공간만 사용하며 즉시 나타납니다.

팀에서 사이트의 새 버전을 배포하려고 하는데 이전 사이트의 백업이 필요하신가요? 스냅샷을 만드세요. 이 새 사이트는 이전 사이트와 동일한 파일을 많이 사용하므로 디스크 공간 사용량을 줄일 수 있습니다. 또한 배포가 심하게 잘못되었을 때 스냅샷으로 롤백하여 이전 버전을 복원할 수 있습니다.

특정 웹 사이트에 파일 시스템 수준의 성능 조정이나 압축 또는 로컬에서 생성된 일부 속성이 필요하신가요? 해당 사이트에 맞게 설정하세요.

각 팀에 대한 데이터 집합을 만든 다음, 각 팀에서 각자의 사이트에 대한 하위 데이터 집합을 만들도록 할 수 있습니다. 기술에 맞게 인력을 구성하는 것이 아니라 인력에 맞게 데이터 집합을 구성할 수 있습니다.

모든 사이트에서 파일 시스템 설정(속성)을 변경해야 하는 경우, 상위 데이터 집합을 변경하고 하위 데이터 집합이 이를 상속하도록 합니다.

사용자 홈 디렉토리에도 동일한 속성이 적용됩니다.

컴퓨터 간에 데이터 세트를 이동할 수도 있습니다. 웹 사이트가 웹 서버를 넘치나요? 데이터 세트의 절반을 사용자 지정 설정, 모든 복제본 및 스냅샷과 함께 새 서버로 전송하세요.

많은 파일시스템 데이터세트를 사용하는 데는 한 가지 단점이 있습니다. 파일시스템 내에서 파일을 이동하면 파일 이름이 변경됩니다. 서로 다른 파일시스템 간에 파일을 이동하려면 이름만 바꾸는 것이 아니라 파일을 새 위치로 복사하고 이전 위치에서 삭제해야 합니다. 데이터 세트 간 파일 복사는 시간이 더 걸리고 더 많은 여유 공간이 필요합니다. 하지만 ZFS가 여러 데이터 세트에 제공하는 모든 이점에 비하면 사소한 문제입니다. 이 문제는 다른 파일시스템에도 존재하지만, 대부분의 다른 파일시스템을 사용하는 호스트는 파티션이 몇 개 밖에 없기 때문에 잘 드러나지 않습니다.

데이터 세트 보기 (Viewing Datasets)

zfs list 명령은 모든 데이터 세트와 그에 대한 몇 가지 기본 정보를 표시합니다.

```
$ zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
mypool              420M  17.9G    96K  none
mypool/ROOT         418M  17.9G    96K  none
mypool/ROOT/default 418M  17.9G   418M  /
...
```

첫 번째 필드에는 데이터 세트의 이름이 표시됩니다.

USED 및 REFER 아래에서 데이터 집합이 사용하는 디스크 공간의 양에 대한 정보를 확인할 수 있습니다 . ZFS 의 놀라운 유연성과 효율성에 대한 한 가지 단점은 디스크 공간 사용량에 대한 해석을 이해하지 못하면 다소 비현실적으로 보인다는 것입니다 . 6장에서는 디스크 공간과 이를 사용하는 전략에 대해 설명합니다 .

AVAIL 열은 풀 또는 데이터 집합에 남아 있는 여유 공간을 표시합니다 .

마지막으로 마운트포인트는 데이터세트를 마운트해야 하는 위치를 보여줍니다 . 이는 데이터 세트가 마운트된다는 의미는 아니며 , 단지 마운트될 경우 이 위치로 이동한다는 의미일 뿐입니다 . 마운트된 모든 ZFS 파일시스템을 보려면 **zfs mount**를 사용하세요 .

데이터세트를 인수로 제공하면 **zfs list**에 해당 특정 데이터 집합만 표시됩니다 .

```
$ zfs list mypool/lamb
NAME         USED  AVAIL  REFER  MOUNTPOINT
mypool/lamb  192K  17.9G    96K  /lamb
```

-t 플래그와 유형으로 표시되는 데이터세트의 유형을 제한합니다 . 파일 시스템 , 볼륨 또는 스냅샷을 표시할 수 있습니다 . 여기서는 스냅샷만 표시합니다 .

```
$ zfs list -t snapshot
NAME                     USED  AVAIL  REFER  MOUNTPOINT
zroot/var/log/db@backup    0      -  10.0G  -
```

이제 파일시스템을 볼 수 있게 되었으니 직접 만들어 보겠습니다 .

????? ??, ?? ? ???? (Creating, Moving, and Destroying Datasets)

zfs create 명령을 사용해 데이터세트를 생성합니다 . 스냅샷 , 복제본 , 북마크는 7장에서 살펴보겠지만 , 지금은 파일시스템과 볼륨에 대해 알아보겠습니다 .

????? ??? (Creating Filesystems)

파일시스템은 대부분의 시스템에서 가장 일반적인 데이터세트 유형입니다 . 누구나 파일을 저장하고 정리할 공간이 필요합니다 . 풀과 파일 시스템 이름을 지정하여 파일 시스템 데이터집합을 만듭니다 .

```
$ zfs create mypool/lamb
```

이렇게 하면 mypool이라는 ZFS 풀에 새 데이터 세트인 lamb이 생성됩니다 . 풀에 기본 마운트 지점이 있는 경우 새 데이터 세트가 기본적으로 마운트됩니다 (이 장 뒷부분의 "ZFS 파일 시스템 마운트 " 참조).

```
$ mount | grep lamb
mypool/lamb on /lamb (zfs, local, noatime, nfsv4acls)
```

괄호 안의 마운트 설정은 일반적으로 부모 데이터세트에서 상속된 ZFS 속성입니다 . 하위 파일시스템을 만들려면 상위 파일시스템의 전체 경로를 입력합니다 .

```
$ zfs create mypool/lamb/baby
```

이 장의 뒷부분에 나오는 '부모 /자식 관계 '에서 살펴보겠지만 , 데이터세트는 마운트 지점을 비롯한 많은 특성을 부모로부터 상속받습니다 .

?? ??? (Creating Volumes)

-V 플래그와 볼륨 크기를 사용하여 볼륨을 만들려는 볼륨을 **zfs create**에 알려줍니다 . 볼륨 데이터세트의 전체 경로를 입력합니다 .

```
$ zfs create -V 4G mypool/avolume
```

Zvols는 다른 데이터세트와 마찬가지로 데이터세트 목록에 표시됩니다 . **-t volume** 옵션을 추가하여 **zfs list**에 zvol만 표시하도록 할 수 있습니다 .

```
$ zfs list mypool/avolume
NAME             USED  AVAIL  REFER  MOUNTPOINT
ypool/avolume  4.13G  17.9G    64K  -
```

Z볼은 볼륨의 크기와 ZFS 메타데이터를 더한 만큼의 공간을 자동으로 예약합니다 . 이 4GB zvol은 4.13GB의 공간을 사용합니다 .

블록 디바이스로서 zvol에는 마운트 지점이 없습니다 . 하지만 */dev/zvol* 아래에 디바이스 노드가 있으므로 다른 블록 디바이스와 마찬가지로 액세스할 수 있습니다 .

```
$ ls -al /dev/zvol/mypool/avolume
crw-r-----  1 root  operator  0x4d Mar 27 20:22 /dev/zvol/mypool/avolume
```

이 디바이스 노드에서 **newfs(8)**를 실행하고 디스크 이미지를 복사한 후 일반적으로 다른 블록 디바이스처럼 사용할 수 있습니다 .

????? ?? ?? (Renaming Datasets)

데이터세트의 이름을 바꾸려면 이상하게도 **zfs rename** 명령을 사용하면 됩니다 . 데이터세트의 현재 이름을 첫 번째 인수로 지정하고 새 위치를 두 번째 인수로 지정합니다 .

```
$ zfs rename db/production db/old
$ zfs rename db/testing db/production
```

데이터 세트의 이름을 강제로 바꾸려면 **-f** 플래그를 사용합니다 . 프로세스가 실행 중인 파일시스템은 마운트 해제할 수 없지만 **-f** 플래그를 사용하면 강제로 마운트 해제할 수 있습니다 . 데이터세트를 사용 중인 모든 프로세스는 사용 중이던 데이터에 대한 액세스 권한을 잃고 사용자가 원하는 대로 반응합니다 .

> 아마 심할 겁니다 .

????? ???? (Moving Datasets)

데이터세트를 ZFS 트리의 일부에서 다른 부분으로 이동하여 데이터세트를 새 부모의 하위 집합으로 만들 수 있습니다 . 자식은 부모로부터 속성을 상속하므로 데이터세트의 많은 속성이 변경될 수 있습니다 . 데이터세트에 특별히 설정된 속성은 변경되지 않습니다 .

여기서는 내결함성을 개선하기 위해 몇 가지 속성을 설정한 새 상위 데이터세트인 *zroot/var/db* 데이터세트 아래에서 데이터베이스를 이동합니다 .

```
$ zfs rename zroot/var/db/mysql zroot/important/mysql
```

마운트 지점은 상속되므로 , 이렇게 하면 데이터 집합의 마운트 지점이 변경될 수 있습니다 . **rename** 명령에 **-u** 플래그를 추가하면 ZFS가 마운트 지점을 즉시 변경하지 않으므로 프로퍼티를 의도한 값으로 재설정할 시간을 벌 수 있습니다 . 컴퓨터를 다시 시작하거나 데이터세트를 수동으로 다시 마운트하면 새 마운트 지점을 사용한다는 점을 기억하세요 .

스냅샷의 이름을 변경할 수는 있지만 상위 데이터 세트에서 스냅샷을 이동할 수는 없습니다 . 스냅샷은 7장에서 자세히 다룹니다 .

????? ???? (Destroying Datasets)

데이터 세트가 지겨우신가요 ? 헛간 뒤로 끌어다 놓고 **zfs destroy**를 통해 고통에서 벗어나세요 .

```
$ zfs destroy db/old
```

-r 플래그를 추가하면 데이터세트의 모든 자식 (데이터 세트 , 스냅샷 등)을 재귀적으로 파기합니다 . 복제된 데이터세트를 모두 파기하려면 **-R**을 사용합니다 . 데이터세트의 자식이 정확히 무엇인지 알 수 없는 경우가 종종 있으므로 데이터세트를 재귀적으로 파기할 때는 매우 주의하세요 .

데이터세트를 파기할 때 어떤 일이 발생하는지 정확히 확인하려면 **-v** 및 **-n** 플래그를 사용할 수 있습니다 . **-v** 플래그는 소멸되는 항목에 대한 자세한 정보를 출력하고 , **-n**은 **zfs(8)**에 드라이런을 수행하도록 지시합니다 . 이 두 플래그는 트리거를 실행하기 전에 이 명령이 실제로 무엇을 파괴하는지 보여줍니다 .

ZFS ?? (ZFS Properties)

ZFS 데이터세트에는 데이터세트 작동 방식을 제어하는 속성이라고 하는 여러 가지 설정이 있습니다 . 이 중 몇 가지는 데이터세트를 만들 때만 설정할 수 있지만 , 대부분은 데이터세트가 라이브 상태일 때 조정할 수 있습니다 . ZFS는 또한 데이터세트가 사용하는 공간의 양 , 압축 또는 중복 제거 비율 , 데이터세트의 생성 시간 등의 정보를 제공하는 여러 읽기 전용 속성을 제공합니다 .

각 데이터세트는 해당 데이터세트에 속성이 특별히 설정되어 있지 않는 한 부모로부터 속성을 상속받습니다 .

?? ?? (Viewing Properties)

zfs(8) 도구는 특정 속성 또는 데이터 세트의 모든 속성을 검색할 수 있습니다 . **zfs get** 명령과 원하는 속성 , 원하는 경우 데이터 세트 이름을 사용합니다 .

```
$ zfs get compression mypool/lamb
NAME         PROPERTY     VALUE    SOURCE
mypool/lamb  compression  lz4      inherited from mypool
```

NAME 아래에는 요청한 데이터 집합이 표시되고, PROPERTY에는 요청한 속성이 표시됩니다. VALUE는 속성이 설정된 값입니다.

SOURCE는 조금 더 복잡합니다. 기본 소스는 이 속성이 ZFS의 기본값으로 설정되어 있음을 의미합니다. 로컬 소스는 누군가가 이 데이터세트에 이 속성을 의도적으로 설정했음을 의미합니다. 임시 속성은 데이터세트가 마운트될 때 설정되었으며, 데이터세트가 마운트 해제되면 이 속성은 일반적인 값으로 되돌아갑니다. 상속된 속성은 이 장의 뒷부분에 있는 "상위/하위 관계"에서 설명하는 대로 상위 데이터세트에서 가져옵니다.

일부 속성은 소스가 관련이 없거나 본질적으로 명백하기 때문에 소스가 없습니다. 데이터세트가 생성된 날짜와 시간을 기록하는 생성 속성에는 소스가 없습니다. 이 값은 시스템 시계에서 가져온 것입니다.

데이터세트 이름을 지정하지 않으면 **zfs get**은 모든 데이터세트에 대해 이 속성의 값을 표시합니다. 특수 속성 키워드는 모두 데이터세트의 모든 속성을 검색합니다.

```
$ zfs get all mypool/lamb
NAME         PROPERTY   VALUE                  SOURCE
mypool/lamb  type       filesystem             -
mypool/lamb  creation   Fri Mar 27 20:05 2015 -
mypool/lamb  used       192K                   -
...
```

all를 사용하고 데이터세트 이름을 지정하지 않으면 모든 데이터세트에 대한 모든 속성을 가져옵니다. 이것은 많은 정보입니다. 속성 이름을 쉼표로 구분하여 여러 속성을 표시합니다.

```
$ zfs get quota,reservation zroot/home
NAME        PROPERTY     VALUE   SOURCE
zroot/home  quota        none    local
zroot/home  reservation  none    default
```

zfs list와 **-o** 수정자를 사용하여 속성을 볼 수도 있습니다. 이 방법은 여러 데이터세트의 여러 속성을 보려는 경우에 가장 적합합니다. 데이터세트의 이름을 표시하려면 특수 속성 **name**을 사용합니다.

```
$ zfs list -o name,quota,reservation
NAME                QUOTA  RESERV
db                   none    none
zroot                none    none
zroot/ROOT           none    none
zroot/ROOT/default   none    none
```

```
...
zroot/var/log         100G     20G
...
```

데이터세트 이름을 추가하여 해당 데이터세트에 대해 이러한 속성을 이 형식으로 볼 수도 있습니다 .

?? ?? (Changing Properties)

zfs set 명령으로 속성을 변경합니다 . 속성 이름 , 새 설정 , 데이터세트 이름을 입력합니다 . 여기서는 **compression** 속성을 **off**로 변경합니다 .

```
$ zfs set compression=off mypool/lamb/baby
```

zfs get으로 변경 사항을 확인합니다 .

```
$ zfs get compression mypool/lamb/baby
NAME               PROPERTY     VALUE   SOURCE
mypool/lamb/baby   compression  off     local
```

대부분의 속성은 속성이 변경된 후에 기록된 데이터에만 적용됩니다 . **compression** 속성은 디스크에 쓰기 전에 데이터를 압축하도록 ZFS에 지시합니다 . 압축에 대해서는 6장에서 설명합니다 . 압축을 비활성화해도 변경 전에 쓰여진 데이터는 압축이 해제되지 않습니다 . 마찬가지로 압축을 활성화해도 디스크에 이미 있는 데이터가 마술처럼 압축되지 않습니다 . 압축 활성화의 이점을 최대한 활용하려면 모든 파일을 다시 작성해야 합니다 . 새 데이터 세트를 생성하고 zfs 전송을 통해 데이터를 복사한 다음 원본 데이터 세트를 삭제하는 것이 좋습니다 .

?? ?? ?? (Read-Only Properties)

ZFS는 읽기 전용 속성을 사용하여 데이터 집합에 대한 기본 정보를 제공합니다 . 디스크 공간 사용량은 속성으로 표현됩니다 . "디스크가 반쯤 찼습니다 ."라는 속성을 변경하여 사용 중인 데이터의 양을 변경할 수는 없습니다 . (6장에서는 ZFS 디스크 공간 사용량에 대해 다룹니다 .) **creation** 속성은 이 데이터 집합이 언제 만들어졌는지 기록합니다 . 디스크에 데이터를 추가하거나 제거하여 많은 읽기 전용 속성을 변경할 수 있지만 이러한 속성을 직접 쓸 수는 없습니다 .

?? ??? ?? (Filesystem Properties)

기존 파일시스템의 성능과 동작을 관리하기 위한 주요 도구 중 하나는 마운트 옵션입니다 . 기존 파일시스템을 읽기 전용으로 마운트하거나 **noexec** 플래그를 사용하여 파일시스템에서 실행 중인 프로그램을 비활성화할 수 있습니다 . ZFS는 속성을 사용하여 동일한 효과를 얻을 수 있습니다 . 다음은 이러한 익숙한 목표를 달성하는 데 사용되는 속성입니다 .

atime

파일의 atime은 파일이 마지막으로 액세스된 시간을 나타냅니다 . ZFS의 **atime** 속성은 데이터 세트가 액세스 시간을 추적할지 여부를 제어합니다 . 기본값인 on은 파일에 액세스할 때마다 파일의 atime 메타데이터를 업데이트합니다 . **atime**을 사용한다는 것은 디스크를 읽을 때마다 디스크에 기록한다는 의미입니다 .

이 속성을 끄면 (off) 파일을 읽을 때 디스크에 쓰기가 방지되므로 성능이 크게 향상될 수 있습니다 . 파일을 마지막으로 읽은 시점을 파악하는 데 의존하는 메일러 및 기타 유사한 유틸리티와 혼동을 일으킬 수 있습니다 .

atime을 그대로 두면 스냅샷 크기가 커집니다 . 파일에 처음 액세스할 때 해당 파일의 타임이 업데이트됩니다 . 스냅샷에는 원래 액세스 시간이 유지되고 , 라이브 파일시스템에는 새로 업데이트된 액세스 시간이 포함됩니다 . 이것이 기본값입니다 .

exec

exec 속성은 누구나 이 파일 시스템에서 바이너리와 명령을 실행할 수 있는지 여부를 결정합니다 . 기본값은 실행을 허용하는 **on**입니다 . 일부 환경에서는 사용자가 개인 또는 임시 디렉터리에서 프로그램을 실행하는 것을 허용하지 않습니다 . 파일 시스템에서 프로그램을 실행할 수 없도록 하려면 **exec** 속성을 **off**로 설정하세요 .

그러나 **exec** 속성은 사용자가 해석된 스크립트를 실행하는 것을 금지하지 않습니다 . 사용자가 */bin/sh*를 실행할 수 있다면 */bin/sh /home/mydir/script.sh*를 실행할 수 있습니다 . 셸은 실제로 실행되는 것으로 , 스크립트의 명령만 받습니다 .

readonly

이 데이터 세트에 아무 것도 쓰지 못하게 하려면 **readonly** 속성을 **on**으로 설정하세요 . 기본값인 **off**로 설정하면 사용자가 관리 권한 내에서 데이터 집합을 수정할 수 있습니다 .

setuid

많은 사람들이 setuid 프로그램을 위험하다고 생각합니다 . **passwd(1), login(1)** 등 일부 setuid 프로그램은 반드시 setuid여야 하지만 , */home* 및 */tmp*와 같은 파일시스템에 setuid 프로그램을 설치할 필요는 거의 없습니다 . 대부분의 시스템 관리자는 특정 파일시스템을 제외하고는 setuid 프로그램을 허용하지 않습니다 .

> 올바르게 작성된 setuid 프로그램은 위험하지 않습니다 . 그렇기 때문에 실제 setuid 프로그램은 위험합니다 .

ZFS의 **setuid** 속성은 setuid 지원을 토글합니다 . **on**으로 설정하면 파일시스템이 setuid를 지원합니다 . **off**로 설정하면 setuid 플래그가 무시됩니다 .

??? ?? ?? (User-Defined Properties)

ZFS 속성은 정말 훌륭하고 충분히 사용할 수 있지 않나요 ? 이제 직접 추가해 보세요 . 데이터 세트와 함께 자신만의 메타데이터를 저장할 수 있는 기능을 통해 완전히 새로운 자동화 영역을 개발할 수 있습니다 . 자식들이 이러한 속성을 자동으로 상속한다는 사실은 삶을 훨씬 더 쉽게 만들어 줍니다 .

사용자 정의 속성을 나만의 것으로 유지하고 다른 사람의 사용자 정의 속성과 충돌하지 않게 하려면 네임스페이스를 만드세요 . 대부분의 사람들은 사용자 정의 속성 앞에 조직 식별자와 콜론을 붙입니다 . 예를 들어 FreeBSD 관련 프로퍼티는 "org.freebsd:속성명 " 형식 (예 : **org.freebsd:swap**)을 갖습니다 . 일루미오스 프로젝트에서 스왑이라는 이름의 자체 프로퍼티를 생성하는 경우 , 이 프로퍼티를 **org.illumos:swap**이라고 부릅니다 . 두 값은 충돌하지 않습니다 .

예를 들어 , Jude가 데이터세트 속성을 통해 어떤 데이터세트를 백업할지 제어하고 싶다고 가정해 보겠습니다 . 그는 네임스페이스 **com.allanjude**를 생성하고 , 이 네임스페이스 내에 **backup_ignore** 속성을 생성합니다 .

> 자신의 이름을 딴 ZFS 속성의 이름을 지정하면 자신의 작업이 영원히 남게 됩니다 . 이것이 좋은지 나쁜지는 여러분의 작업에 달려 있습니다 .

```
$ zfs set com.allanjude:backup_ignore=on mypool/lamb
```

Jude의 백업 스크립트는 이 속성 값을 확인합니다 . true로 설정되어 있으면 백업 프로세스에서 이 데이터 세트를 건너뜁니다 .

??/?? ?? (Parent/Child Relationships)

데이터세트는 상위 데이터세트에서 속성을 상속합니다 . 데이터세트에서 속성을 설정하면 해당 속성은 해당 데이터세트와 모든 하위 데이터세트에 적용됩니다 . 편의를 위해 **-r** 플래그를 추가하여 데이터세트와 그 모든 하위 데이터세트에 대해 **zfs(8)** 명령을 실행할 수 있습니다 . 여기서는 데이터세트와 모든 하위 데이터세트의 압축 속성을 쿼리합니다 .

```
$ zfs get -r compression mypool/lamb
NAME              PROPERTY     VALUE  SOURCE
mypool/lamb       compression  lz4    inherited from mypool
mypool/lamb/baby  compression  off    local
```

소스 값을 살펴봅니다 . 첫 번째 데이터 집합인 *mypool/lamb*은 상위 풀에서 이 속성을 상속받았습니다 . 두 번째 데이터 집합에서 이 속성은 다른 값을 갖습니다 . 소스가 로컬이므로 이 속성이 이 데이터 집합에 특별히 설정되었음을 의미합니다 .

zfs inherit 명령으로 원래 설정을 복원할 수 있습니다 .

```
$ zfs inherit compression mypool/lamb/baby
$ zfs get -r compression mypool/lamb
NAME              PROPERTY     VALUE SOURCE
mypool/lamb       compression  lz4   inherited from mypool
mypool/lamb/baby  compression  lz4   inherited from mypool
```

이제 자식은 부모로부터 **compression** 속성을 상속받으며 , 부모는 조부모로부터 **compression** 속성을 상속받습니다 .

부모의 속성을 변경하면 새 속성이 자동으로 자식에게 전파됩니다 .

```
$ zfs set compression=gzip-9 mypool/lamb
$ zfs get -r compression mypool/lamb
```

```
NAME              PROPERTY     VALUE  SOURCE
mypool/lamb       compression  gzip-9 local
mypool/lamb/baby  compression  gzip-9 inherited from mypool/lamb
```

부모 데이터 집합에 *gzip-9* 압축을 사용하도록 지시했습니다 . 이는 자식에게까지 영향을 미쳤습니다 .

?? ? ?? ?? (Inheritance and Renaming)

데이터세트를 이동하거나 이름을 변경하여 새 부모를 갖게 되면 부모의 속성이 자동으로 자식에게 전파됩니다 . 로컬로 설정된 속성은 변경되지 않지만 상속된 속성은 새 부모의 속성으로 전환됩니다 .

여기서는 새 상위 데이터세트를 만들고 압축 속성을 확인합니다 .

```
$ zfs create mypool/second
$ zfs get compress mypool/second
NAME           PROPERTY     VALUE SOURCE
mypool/second  compression  lz4   inherited from mypool
```

baby 데이터세트는 *gzip-9* 압축을 사용합니다 . 이 속성은 *mypool/lamb*에서 상속받았습니다 . 이제 *baby*를 second의 자식으로 이동하고 압축 속성이 어떻게 되는지 살펴봅시다 .

```
$ zfs rename mypool/lamb/baby mypool/second/baby
$ zfs get -r compression mypool/second
NAME                PROPERTY     VALUE  SOURCE
mypool/second       compression  lz4    inherited from mypool
mypool/second/baby  compression  lz4    inherited from mypool
```

이제 자식 데이터 집합은 다른 부모에 속하며 새 부모로부터 해당 속성을 상속받습니다 . 자식은 모든 로컬 속성을 유지합니다 .

그러나 아기 데이터 세트의 데이터는 약간 엉켜 있습니다 . 압축을 켜기 전에 작성된 데이터는 압축되지 않습니다 . 데이터 세트가 gzip-9 압축을 사용하는 동안 작성된 데이터는 gzip-9로 압축됩니다 . 지금 작성된 모든 데이터는 lz4로 압축됩니다 . ZFS는 이 모든 것을 자동으로 분류해 주지만 , 생각만 해도 머리가 아픕니다 .

?? ?? (Removing Properties)

속성을 기본값으로 다시 설정할 수는 있지만 , 소스를 상속 또는 기본값으로 다시 변경하는 방법이나 사용자 정의 속성을 설정한 후 제거하는 방법은 명확하지 않습니다 .

사용자 정의 속성을 제거하려면 상속하세요 .

```
$ zfs inherit com.allanjude:backup_ignore mypool/lamb
```

이는 루트 데이터세트에 속성을 설정한 경우에도 작동합니다 .

데이터세트와 모든 하위 데이터세트에서 속성을 기본값으로 재설정하거나 사용자 정의 속성을 완전히 제거하려면 풀의 루트 데이터세트에서 **zfs inherit** 명령을 사용합니다 .

```
$ zfs inherit -r compression mypool
```

직관적이지 않지만 루트 데이터세트에서 사용자 정의 설정을 제거합니다 .

ZFS ????? ????? (Mounting ZFS Filesystems)

기존 파일시스템에서는 각 파티션과 그 유형 , 마운트 위치를 */etc/fstab*에 나열했습니다 . 편의를 위해 플로피나 CD-ROM 드라이브와 같은 임시 마운트도 나열했습니다 . ZFS를 사용하면 너무 많은 수의 파일시스템을 생성할 수 있어 금방 비실용적이 됩니다 .

각 ZFS 파일시스템에는 마운트할 위치를 정의하는 **mountpoint** 속성이 있습니다 . 기본 **mountpoint**는 풀의 **mountpoint**에서 빌드됩니다 . 풀에 마운트 포인트가 없는 경우 , 마운트하려는 데이터 세트에 마운트 포인트를 할당해야 합니다 .

```
$ zfs get mountpoint zroot/usr/home
NAME             PROPERTY     VALUE       SOURCE
zroot/usr/home   mountpoint   /usr/home   inherited from zroot/usr
```

파일시스템은 일반적으로 */usr/home*에 마운트됩니다 . 파일시스템을 수동으로 마운트할 때 이 위치를 재정의할 수 있습니다 .

기본 FreeBSD 설치에 사용되는 *zroot* 풀에는 마운트 지점이 설정되어 있지 않습니다 . *zroot* 바로 아래에 새 데이터세트를 생성하는 경우 , 해당 데이터세트에는 마운트 지점이 없습니다 . 예를 들어 */usr* 아래에 있는 *zroot*에서 만든 데이터 세트는 부모 데이터 세트에서 마운트 지점을 상속합니다 .

루트 파일시스템이 있는 풀 이외의 모든 풀은 일반적으로 풀의 이름을 딴 마운트 지점을 갖습니다 . *db*라는 이름의 풀을 만들면 */db*에 마운트됩니다 . 모든 자식은 변경하지 않는 한 해당 풀에서 마운트 지점을 상속받습니다 .

파일시스템의 **mountpoint** 속성을 변경하면 파일시스템과 마운트 지점을 상속하는 모든 하위 파일시스템이 마운트 해제됩니다 . 새 값이 레거시 값이면 마운트 해제된 상태로 유지됩니다 . 그렇지 않고 이전에 *legacy* 속성이었거나 속성이 *none*이거나 속성이 변경되기 전에 마운트된 경우 새 위치에 자동으로 다시 마운트됩니다 . 또한 모든 공유 파일시스템은 공유가 해제되고 새 위치에서 공유됩니다 .

일반 파일시스템과 마찬가지로 ZFS 파일시스템도 반드시 마운트할 필요는 없습니다 . **canmount** 속성은 파일시스템의 마운트 동작을 제어합니다 . **canmount**가 **yes**로 설정되어 있으면 , 마운트 **-a**와 마찬가지로 **zfs mount -a**를 실행하면 파일시스템이 마운트됩니다 . *etc/rc.conf*에서 ZFS를 활성화하면 FreeBSD는 시작할 때 **zfs mount -a**를 실행합니다 .

canmount 속성이 **noauto**로 설정된 경우 , 데이터세트는 명시적으로만 마운트 및 마운트 해제할 수 있습니다 . 데이터 집합을 만들거나 가져올 때 데이터세트가 자동으로 마운트되지 않으며 , **zfs mount -a** 명령으로 마운트하거나 **zfs unmount -a**로 마운트 해제하지도 않습니다 .

canmount **off** . . . . . .
. . . . . .

. **canmount** . .

canmount . . . . . .

??? ???? ?? ????? (Datasets without Mount Points)

ZFS . . . . . FreeBSD 10.1 . .

```
$ zfs mount
zroot/ROOT/default   /
zroot/tmp            /tmp
zroot/usr/home       /usr/home
zroot/usr/ports      /usr/ports z
root/usr/src        /usr/src
...
```

/usr . . . . . */usr* . . . ?

zfs list . */usr* . . . *zroot/usr* .
. **canmount** . .

```
$ zfs list -o name,canmount,mountpoint -r zroot/usr
NAME          CANMOUNT  MOUNTPOINT
zroot/usr          off  /usr
zroot/usr/home     on   /usr/home
zroot/usr/ports    on   /usr/ports
zroot/usr/src      on   /usr/src
```

canmount *off* *zroot/usr* . *usr/bin* . */usr/local* . */usr* . . *usr/src* . , .

. */var* . .

??? ??? ??? ?? ?? ????? (Multiple Datasets with the Same Mount Point)

canmount *off* . . . . .
canmount *off* . . . . . .

집합의 하위 집합이 동일한 디렉터리에 표시되지만 상속된 특성이 다를 수 있도록 하기 위해서입니다.

FreeBSD의 설치 관리자에는 기본 풀인 zroot에 마운트 지점이 없습니다. 새 데이터세트를 만들 때는 마운트 포인트를 할당해야 합니다.

풀 바로 아래에 생성하는 모든 데이터세트에 **mountpoint**를 할당하고 싶지 않다면, *zroot* 풀에 */*의 마운트 포인트를 할당하고 **canmount**를 *off*로 설정할 수 있습니다. 이렇게 하면 새 데이터세트를 만들 때 상속할 **mountpoint**를 갖게 됩니다. 이것은 동일한 마운트 포인트를 가진 여러 데이터세트를 사용하는 매우 간단한 예입니다.

두 개의 하위 디렉터리 세트가 있는 */opt* 디렉터리가 있다고 가정해 보겠습니다. 이러한 디렉터리 중 일부는 프로그램이 포함되어 있으므로 설치 후에는 절대로 쓰면 안 됩니다. 다른 디렉터리에는 데이터가 들어 있습니다. 파일 시스템 수준에서 프로그램을 실행하는 기능을 잠가야 합니다.

```
$ zfs create db/programs # zfs create db/data
```

이제 이 두 데이터세트에 모두 */opt*의 마운트 지점을 지정하고 마운트할 수 없음을 알려줍니다.

```
$ zfs set canmount=off db/programs
$ zfs set mountpoint=/opt db/programs
```

데이터세트에 프로그램을 설치한 다음 읽기 전용으로 설정합니다.

```
$ zfs set readonly=on db/programs
```

db/data 데이터세트에서는 프로그램을 실행할 수 없으므로 **exec** 및 **setuid**를 해제하세요. 하지만 이 디렉터리에 데이터를 써야 합니다.

```
$ zfs set canmount=off db/data
$ zfs set mountpoint=/opt db/data
$ zfs set setuid=off db/data
$ zfs set exec=off db/data
```

이제 몇 가지 자식 데이터세트를 만듭니다. *db/programs* 데이터세트의 하위 데이터 세트는 해당 데이터 세트의 속성을 상속하고, *db/data* 데이터 세트의 하위 데이터세트는 다른 속성 집합을 상속합니다.

```
$ zfs create db/programs/bin
$ zfs create db/programs/sbin
$ zfs create db/data/test
$ zfs create db/data/production
```

이제 */opt*에 바이너리용 2개와 데이터용 2개, 총 4개의 데이터세트가 마운트되어 있습니다. 사용자가 알다시피 이 디렉터리는 정상적인 디렉터리입니다. 하지만 파일 권한이 어떻게 되어 있든 이 디렉터리 중 두 개는 아무도 쓸 수 없습니다. 사람들이 어떤 속임수를 쓰더라도 시스템은 나머지 두 디렉터리에 있는 실행

파일과 setuid 파일을 인식하지 못합니다 . 데이터나 프로그램을 위해 다른 데이터세트가 필요한 경우 , 원하는 설정으로 데이터세트의 하위 데이터세트로 생성하세요 . 상위 데이터세트에 대한 변경 사항은 모든 하위 데이터세트에 즉시 전파됩니다 .

??? ???? ?? ? (Pools without Mount Points)

풀은 일반적으로 풀의 이름을 딴 디렉터리에 마운트되지만 , 반드시 그런 것은 아닙니다 .

```
$ zfs set mountpoint=none mypool
```

이 풀은 더 이상 마운트되지 않습니다 . 마운트 지점을 지정하지 않으면 풀에 있는 데이터 세트도 마운트되지 않습니다 . 이것이 FreeBSD 설치 관리자가 OS용 풀을 생성하는 방식입니다 .

```
$ zfs set mountpoint=/someplace mypool/lamb
```

필요한 경우 디렉터리가 생성되고 파일 시스템이 마운트됩니다 .

???? ?? ??? ??? ? ??? ???? (Manually Mounting and Unmounting Filesystems)

파일시스템을 수동으로 마운트하려면 **zfs mount**와 데이터세트 이름을 사용합니다 . 이 방법은 **canmount**가 *noauto*으로 설정된 파일시스템에 가장 일반적으로 사용됩니다 .

```
$ zfs mount mypool/usr/src
```

파일시스템과 그 하위 파일시스템을 모두 마운트 해제하려면 **zfs unmount**를 사용합니다 .

```
$ zfs unmount mypool/second
```

데이터 세트를 다른 위치에 임시로 마운트하려면 **-o** 플래그를 사용하여 새 마운트 지점을 지정합니다 . 이 마운트 지점은 데이터 세트를 마운트 해제할 때까지만 지속됩니다 .

```
$ zfs mount -o mountpoint=/mnt mypool/lamb
```

데이터 세트에 **mountpoint**가 정의되어 있는 경우에만 데이터세트를 마운트할 수 있습니다 . 데이터세트에 마운트 지점이 없는데 임시 마운트 지점을 정의하면 오류가 발생합니다 .

ZFS ? /etc/fstab (ZFS and /etc/fstab)

원하는 경우 */etc/fstab*으로 ZFS 파일시스템 마운트 지점의 일부 또는 전부를 관리하도록 선택할 수 있습니다 . 데이터 집합의 **mountpoint** 속성을 *legacy*로 설정합니다 . 그러면 파일시스템이 마운트 해제됩니다 .

```
$ zfs set mountpoint=legacy mypool/second
```

이제 **mount(8)** 명령으로 이 데이터세트를 마운트할 수 있습니다 :

```
$ mount -t zfs mypool/second /tmp/second
```

시스템의 */etc/fstab*에 ZFS 데이터세트를 추가할 수도 있습니다 . 전체 데이터세트 이름을 장치 노드로 사용합니다 . 유형을 *zfs*로 설정합니다 . 표준 파일 시스템 옵션인 *noatime, noexec, readonly* 또는 *ro, nosuid*를 사용할 수 있습니다 . (명시적으로 *atime, exec, rw, suid*의 기본 동작을 지정할 수도 있지만 이는 ZFS의 기본값입니다). 마운트 순서는 정상이지만 fsck 필드는 무시됩니다 . 다음은 */tmp*에 데이터세트 *scratch/junk* nosuid를 마운트하는 */etc/fstab* 항목입니다 .

```
scratch/junk /tmp nosuid 2 0
```

그러나 마운트를 관리할 때는 ZFS 속성을 사용하는 것이 좋습니다 . 속성은 */etc/fstab*이 하는 거의 모든 작업을 수행할 수 있습니다 .

ZFS ?? ?? (Tweaking ZFS Volumes)

Zvol은 매우 간단합니다 . 공간 덩어리를 블록 디바이스로 설정하고 이를 사용하면 됩니다 . 볼륨이 공간을 사용하는 방식과 제공하는 디바이스 노드의 종류를 조정할 수 있습니다 .

?? ?? (Space Reservations)

zvol의 **volsize** 속성은 볼륨의 논리적 크기를 지정합니다 . 기본적으로 볼륨을 생성하면 볼륨 크기와 동일한 양의 데이터세트 공간이 예약됩니다 . (6장 앞부분에서 보면 동일한 크기의 예약이 설정되어 있습니다 .). **volsize**를 변경하면 예약이 변경됩니다 . volsize는 **volblocksize** 속성의 배수로만 설정할 수 있으며 0은 될 수 없습니다 .

예약을 하지 않으면 볼륨 사용 방식에 따라 볼륨의 공간이 부족하여 정의되지 않은 동작이나 데이터 손상이 발생할 수 있습니다 . 이러한 영향은 사용 중에 볼륨 크기를 변경할 때 , 특히 크기를 축소할 때 발생할 수 있습니다 . 볼륨 크기를 조정하면 차단 장치를 사용하는 애플리케이션에 혼란을 줄 수 있습니다 .

Zvol은 씬 프로비저닝 *(thin provisioning)*이라고도 하는 스파스 볼륨 *(sparse volumes)*도 지원합니다 . 스파스 볼륨은 예약이 볼륨 크기보다 작은 볼륨입니다 . 기본적으로 스파스 볼륨을 사용하면 데이터세트가 사용할 수 있는 공간보다 더 많은 공간을 할당할 수 있습니다 . 예를 들어 , 스파스 프로비저닝을 사용하면 5TB 데이터 세트에 1TB 스파스 볼륨 10개를 생성할 수 있습니다 . 볼륨을 많이 사용하지 않는 한 , 아무도 볼륨이 과도하게 커밋되었다는 사실을 알아차리지 못합니다 .

스파스 볼륨은 권장하지 않습니다 . 희소 볼륨에 대한 쓰기는 볼륨 자체가 부분적으로만 가득 차 있어도 '공간 부족 ' 오류와 함께 실패할 수 있습니다 .

zfs create -V 명령에 **-s** 옵션을 지정하여 생성 시 스파스 볼륨을 지정합니다 . 볼륨 크기 변경은 예약에 반영되지 않습니다 . 볼륨이 생성된 후 예약을 줄일 수도 있습니다 .

Zvol ?? (Zvol Mode)

FreeBSD는 일반적으로 운영체제에 zvol을 **geom(4)** 공급자로 노출하여 최대한의 유연성을 제공합니다 . **volmode** 속성으로 이를 변경할 수 있습니다 .

volmode *dev* /dev . , RAID GEOM . . . .

volmode *none* ZFS . , , .

volmode *default* **sysctl vfs.zfs.vol.mode** . zvol 1 geom, 2 dev, 3 none .

 , , . **zfs rename** zvol .

????? ??? (Dataset Integrity)

ZFS VDEV . RAID-Z . , , .

??? (Checksums)

ZFS , ZFS . *on, fletcher2, fletcher4, sha256, off, noparity* .

on OpenZFS 2015 *fletcher4* .

 fletcher4 . *fletcher4* *fletcher4* . *on* ZFS .

off .

noparity RAID-Z .

 ZFS *fletcher2* . *sha256* *fletcher4* .

중복 제거 (deduplication)를 수행할 때 *sha256* 알고리즘을 자주 권장합니다 .

??? (Copies)

ZFS는 중요한 메타데이터의 복사본을 두세 개 저장하며 , 중요한 사용자 데이터에 대해서도 동일한 처리를 할 수 있습니다 . **copies** 속성은 보관할 사용자 데이터의 사본 수를 ZFS에 알려줍니다 . ZFS는 하드웨어 장애에 대비하기 위해 이러한 복사본을 다른 디스크에 저장하거나 , 저장하지 못할 경우 물리적 디스크에 가능한 한 멀리 떨어져 저장하려고 시도합니다 . **copies** 속성을 늘리면 해당 데이터 집합에 대한 메타데이터 복사본 수도 최대 **3**개 까지 증가합니다 .

풀이 2개의 미러 디스크에서 실행되고 **copies**를 3개로 설정한 경우 , 데이터 사본이 6개가 됩니다 . 이 중 하나는 원시 공급자 장치에서 **dd(1)**을 잘못 사용하거나 지붕에서 떨어져도 살아남아야 합니다 .

사본을 늘리거나 줄이면 설정 변경 후에 기록된 데이터에만 영향을 줍니다 . **copies**를 1개에서 2개로 변경한다고 해서 모든 데이터의 사본이 갑자기 중복 생성되지는 않습니다 . 10MB 용량의 임의 데이터 파일을 만듭니다 :

```
$ dd if=/dev/random of=/lamb/random1 bs=1m count=10
10+0 records in
10+0 records out
10485760 bytes transferred in 0.144787 secs (72421935 bytes/sec)
$ zfs set copies=2 mypool/lamb
```

이제 모든 블록이 두 번 저장됩니다 . 복사본 중 하나가 손상되더라도 ZFS는 여전히 파일을 읽을 수 있습니다 . 체크섬이 일치하지 않기 때문에 어떤 블록이 손상되었는지 알 수 있습니다 . 하지만 풀에서 사용되는 공간 (풀 목록의 REFER 공간)을 보세요 .

```
$ zfs list mypool/lamb
NAME          USED  AVAIL  REFER  MOUNTPOINT
mypool/lamb  10.2M  13.7G  10.1M  /lamb
```

우리가 쓴 10MB만 사용되었습니다 . 이 파일은 복사본 속성을 변경하기 전에 썼기 때문에 추가 복사본이 만들어지지 않았습니다 . 그러나 복사본이 2로 설정된 상태에서 다른 파일을 쓰거나 원본 파일을 덮어쓰면 디스크 사용량이 달라지는 것을 볼 수 있습니다 .

```
$ dd if=/dev/random of=/lamb/random2 bs=1m count=10
10+0 records in
10+0 records out
10485760 bytes transferred in 0.141795 secs (73950181 bytes/sec)
```

지금 디스크 사용량을 확인하세요 .

```
$ zfs list mypool/lamb
NAME         USED  AVAIL  REFER  MOUNTPOINT
```

```
mypool/lamb  30.2M  13.7G  30.1M  /lamb
```

총 공간 사용량은 30MB로 , 첫 번째 무작위 데이터 파일에 10, 두 번째 10MB 파일 사본 2개에 20입니다 . **ls(1)**로 파일을 보면 실제 크기만 표시됩니다 :

```
$ ls -l /lamb/random*
-rw-r--r--  1 root  wheel  10485760 Apr  6 15:27 /lamb/random1
-rw-r--r--  1 root  wheel  10485760 Apr  6 15:29 /lamb/random2
```

데이터 세트의 복원력을 제대로 활용하고 싶다면 메타데이터 중복성을 살펴보세요 .

????? ??? (Metadata Redundancy)

각 데이터 세트는 내부 메타데이터의 추가 복사본을 저장하므로 단일 블록이 손상되더라도 손실되는 사용자 데이터의 양이 제한됩니다 . 이 추가 복사본은 VDEV 수준에서 제공되는 모든 중복성 (예 : 미러링 또는 RAID-Z) 에 추가됩니다 . 또한 복사본 속성 (아래)에 지정된 모든 추가 복사본에 추가되며 , 최대 총 3개의 복사본이 생성됩니다 .

redundant_metadata 속성을 사용하면 데이터 세트 메타데이터를 얼마나 중복시킬지 결정할 수 있습니다 . 대부분의 사용자는 이 속성을 변경하지 않아야 합니다 .

redundant_metadata를 *all*(기본값)로 설정하면 ZFS는 모든 메타데이터의 추가 복사본을 저장합니다 . 하나의 디스크 블록이 손상되면 최악의 경우 사용자 데이터의 단일 블록이 손실될 수 있습니다 .

redundant_metadata를 *most*로 설정하면 ZFS는 대부분의 메타데이터 유형에 대해서만 추가 복사본을 저장합니다 . 이렇게 하면 쓰기해야 하는 메타데이터가 줄어들기 때문에 랜덤 쓰기 성능이 향상될 수 있습니다 . 대부분의 메타데이터만 중복 저장하는 경우 , 하나의 온디스크 블록이 손상되면 최악의 경우 약 100개의 사용자 데이터 블록이 손실될 수 있습니다 . 중복 저장되는 메타데이터 블록의 정확한 동작은 향후 릴리스에서 변경될 수 있습니다 .

redundant_metadata를 *most*로 설정하고 **copies**를 3으로 설정하고 데이터 세트가 미러 풀에 있는 경우 , ZFS는 대부분의 메타데이터 사본 6개와 데이터 및 일부 메타데이터 사본 4개를 저장합니다 .

이 속성은 데이터베이스와 같이 메타데이터를 자주 업데이트하는 특정 사용 사례를 위해 설계되었습니다 . 데이터가 이미 충분히 강력한 내결함성으로 보호되고 있다면 데이터베이스가 변경될 때마다 작성해야 하는 메타데이터의 복사본 수를 줄이면 성능이 향상될 수 있습니다 . 이 값은 자신이 무엇을 하고 있는지 알고 있는 경우에만 변경하세요 .

이제 데이터 집합에 대해 이해했으니 풀 유지 관리에 대해 알아보겠습니다 .

Revision #1
Created 24 January 2024 04:04:44 by 고기만두 (MeatDumpling)
Updated 24 January 2024 04:04:52 by 고기만두 (MeatDumpling)