FreeBSD Mastery: ZFS

Michael W. Lucas와 Allan Jude의 FreeBSD Mastery: ZFS를 번역한 것입니다.
FreeBSD관련 책은 거의 나오는 것이 없으므로 가능한한 책도 좀 사주시길 부탁드립니다.
이건 제가 한글로 보고 싶어서 번역한 내용입니다.

- 챕터 0 : 소개
- 챕터 1: ZFS 소개
- 챕터 2 : 가상 장치 (Virtual Devices)
- 챕터 3. 풀(Pools)
- 챕터 4. ZFS 데이터셋 (ZFS Datasets)

챕터 0 : 소개

대부분의 시스템 관리 교육은 파일시스템에 초점을 맞춥니다. 컴퓨터의 파일시스템은 컴퓨터의 성능과 동작의 많은 부분을 좌우합니다. 지난 수십 년 동안 우리는 주요 파일시스템이 잘못 구성되었거나, 선택

오늘날 가장 널리 사용되는 파일시스템 중 일부는 컴퓨팅 규모로 볼 때 오래된 것입니다. 우리는 5년이 지나 너무 느리다는 이유로 하드웨어를 폐기하고 그 자리에 30년 된 파일시스템을 설치합니다. e

ZFS란 무엇인가?

ZFS는 파일시스템이 데이터 무결성을 보장할 수 있어야 한다는 핵심 아이디어로 구축된 최신 파일시스템입니다. ZFS는 디스크의 모든 데이터에 대해 체크섬을 계산하여 스토리지 미디어에 오류가 발

파일시스템 설계자들은 30년 전에 이러한 아이디어를 가지고 있었지만, 당시의 하드웨어로는 이 정도의 오류 검사를 적절한 성능으로 수행할 수 없었습니다. ZFS의 개발자들은 현재의 하드웨어와 하드

오늘날의 전통적인 파일시스템은 "지금은 이 정도면 충분하다"는 철학으로 작성된 것으로 보입니다. 많은 파일시스템은 임의의 크기 제한으로 인해 5년, 10년, 심지어 20년 동안은 충분했지만 결국

ZFS는 가까운 미래에도 살아남을 수 있도록 의도적으로 설계되었습니다. 많은 새로운 파일시스템은 내부적으로 64비트 식별자와 인덱스를 사용하므로 향후 10년 또는 20년 동안 변경 없이 사용

엄밀히 말하면 ZFS는 단순한 파일 시스템이 아닙니다. 파일시스템과 볼륨 관리자가 결합된 형태입니다. 이 두 가지 기능을 하나의 소프트웨어에 결합하면 나중에 설명하겠지만 몇 가지 제한이 따르지만 [

ZFS 역사

매트 아렌스와 제프 본윅은 썬 마이크로시스템즈의 솔라리스® 운영 체제용 ZFS를 만들었습니다. Sun은 모든 규모의 시스템을 판매했지만, 주로 하이엔드 서버 하드웨어에 주력했습니다. Sun

그러던 중 오라클이 썬 마이크로시스템을 인수했습니다. 오라클은 MySQL과 같은 일부 오픈 소스 소프트웨어를 보유하고 있지만, 대부분의 소프트웨어는 독점 소프트웨어입니다. 오라클은 ZFS 개

하지만 ZFS 코드는 이미 공개되어 있었고, CDDL 라이선스 조건에 따라 오라클은 사람들이 이를 사용하는 것을 막을 수 없었습니다. 다양한 오픈 소스 프로젝트가 자체적인 ZFS 개발 노력을 [

현재 OpenZFS 프로젝트(http://open-zfs.org)가 오픈 소스 버전의 ZFS를 총괄하고 있습니다. OpenZFS는 Linux, OS X, Illumos, FreeB

다른 FreeBSD와 달리, ZFS는 제한적인 라이선스를 가지고 있습니다. CDDL은 특허 소송을 제기할 수 있는 능력을 제한하며, ZFS에 대한 기여는 자동으로 CDDL의 적용을 받습

사전 요구 사항

이 책은 ZFS에 관심이 있는 FreeBSD 시스템 관리자를 위해 작성되었습니다. 우리는 여러분이 설치, 사용자 구성, GEOM 기반 스토리지 관리 등 FreeBSD의 기본 사항에 익숙

2007)와 같은 책이나 다른 FreeBSD 마스터리 타이틀을 구입하여 지식을 보강하거나 온라인 문서와 매뉴얼 페이지를 참조할 수 있습니다.

FreeBSD가 아닌 다른 운영 체제에서 OpenZFS를 사용하는 경우, 이 책은 실용적인 ZFS 지식과 경험을 많이 제공합니다. FreeBSD와 관련된 내용은 무시해도 되지만, ZF

이 책은 오라클 ZFS에 대해 테스트하지 않았습니다. 오라클은 비공개 소스인 ZFS를 독자적인 방향으로 발전시켜 왔으며, 오라클 솔라리스를 실행해야 하는 경우 오라클의 공식 문서를 읽는 것이 더

디스크 기술에 대해 알아야 합니다. FreeBSD의 ZFS는 모든 GEOM 제공업체에서 실행할 수 있지만, 원시 디스크에서 실행하면 특정 이점이 있습니다. RAID 컨테이너에서 ZFS

RAID 컨트롤러를 여러 개의 디스크처럼 작동하도록 설정할 수 있어야 합니다. 또한 ZFS를 플레이할 수 있는 테스트 머신이 있어야 합니다. 이 책을 구입하고 바로 메인 데이터베이스 서버를 ZF

ZFS는 어디에서 사용하나요?

ZFS는 어디에서나 사용할 수 있지만 일부 환경에서는 제대로 작동하지 않을 수 있습니다.

특정 가상화 시스템에서는 ZFS가 최선의 선택이 아닐 수도 있습니다. 저희는 ZFS 파일 시스템에 막히는 Linux KVM 기반 가상화 시스템을 두 개 이상 사용해 보았으며, 다른 시스템에서

ZFS는 최신 하드웨어용으로 작성되었습니다. 최소한 몇 기가바이트의 RAM이 있어야 합니다. 라즈베리 파이와 같은 임베디드 시스템은 UFS2와 같은 기존 파일시스템에 더 적합합니다.

특정 하드웨어의 특정 고강도 워크로드는 특히 실제 512바이트 블록이 있는 하드 드라이브가 있는 경우 ZFS보다 UFS2에서 더 잘 수행됩니다. 데이터베이스의 성능을 최대한 끌어올리고 싶다면

ZFS 하드웨어

많은 사람들이 ZFS에 하이엔드 하드웨어를 추천합니다. 저희도 하이엔드 하드웨어를 좋아합니다. 멋지죠. 하지만 하드웨어의 한계를 이해한다면 일반 하드웨어에서도 ZFS가 잘 작동합니다. 인터넷에

RAM

Sun의 문서에서 ZFS를 잘 사용하려면 ECC RAM이 필요하다고 말한 것은 놀라운 일이 아닙니다. 썬은 하이엔드 서버를 판매했기 때문입니다. 하지만 매트 아렌스에 따르면, "ECC가 [

물론 ECC RAM을 선호하는 일반적인 논거는 여전히 유효합니다. 비 ECC 메모리가 장착된 시스템에서는 메모리 손상이 발생할 수 있으며, 이러한 오류 중 일부가 디스크에 기록될 수 있습니다.

RAID 컨트롤러

하드웨어 RAID 컨트롤러를 사용하지 마세요. 절대로. 하드웨어 RAID 장치 위에서 ZFS를 실행하면 RAID의 모든 단점과 ZFS의 모든 단점이 있는 구성이 만들어집니다. 디스크 컨트롤

모든 RAID는 소프트웨어 RAID입니다. 하드웨어 RAID 컨트롤러는 사용자 지정 운영 체제를 실행하여 RAID 작업을 수행하며, 이 과정에서 운영 체제에서 하드웨어를 가립니다. 이는 스

운영 체제가 개선되었습니다. 하드웨어는 수십억 배 더 강력해졌습니다. 환경이 바뀌었습니다.

ZFS는 하드웨어에 직접 액세스할 수 있도록 설계되었습니다. 의도적으로 중요한 메타데이터를 여러 디스크에 저장합니다. 그리고 이러한 디스크에 오류가 있는지 감시하고 오류를 기반으로 결정을 내립니

ZFS 어레이를 재구축하는 것이 RAID 어레이를 재구축하는 것보다 훨씬 빠르며, 이는 파일 시스템과 중복성을 통합한 ZFS의 기능 덕분입니다. 하드웨어 RAID로 중복성을 관리하면 이러한 |

그러나 많은 RAID 컨트롤러는 일종의 RAID가 없는 디스크를 사용할 수 없습니다. 컨트롤러에 "디스크만 여러 개", 즉 JBOD를 구성하더라도 이러한 컨트롤러는 실제로 각 드라이브를 단일 | 없습니다. 이렇게 하면 ZFS의 모든 풀 이동성이 사라집니다.

일부 하드웨어 RAID 카드는 재플래시하여 JBOD 컨트롤러로 사용할 수 있습니다. 불량 플래시는 RAID 카드를 망가뜨릴 수 있습니다. 두 저자 모두 하드웨어 RAID를 다시 사용하지 않을

전생에 아주 나쁜 사람이었기 때문에 하드웨어 RAID를 사용할 수밖에 없다면 운영 체제에 단일 디스크를 제시하세요.

RAID 컨트롤러가 각 디스크를 RAID 0으로 포맷해야 한다고 고집하면 문제가 발생합니다. 컨트롤러에서 "다시 쓰기" 모드를 비활성화하세요. 그렇지 않으면 컨트롤러의 쓰기 캐시로 인해 파일 [

호스트 버스 어댑터는 RAID 카드보다 비용뿐 아니라 시간 면에서도 훨씬 저렴합니다. 사용하세요.

SATA 대 SAS 대 SSD

데이터 저장 장치는 다양한 유형으로 제공됩니다. ZFS는 SAS 드라이브, SATA 드라이브, 회전식 플래터, SSD 장치 또는 운영 체제에서 지원하는 기타 모든 저장 매체를 사용할 수 있습

그렇다고 해서 두 드라이브가 동등하다는 것은 아닙니다. SAS 드라이브는 일반적으로 동일한 부하에서 SATA 드라이브보다 훨씬 더 오래 사용할 수 있습니다. 플래시 드라이브는 어떤 종류의 회전

ZFS는 초고속 스토리지를 읽기 및 쓰기 캐시로 특별히 활용할 수 있습니다. SAS 또는 SATA 기반 스토리지 어레이에 솔리드 스테이트 디스크 몇 개를 추가하면 ZFS의 성능을 크게 높일

디스크 중복성(DiskRedundancy)

스토리지 어레이에서 여러 개의 디스크가 동시에 손실되는 경우는 드물지 않습니다. 전력 서지는 여러 개의 디스크를 손상시킬 수 있습니다. 집중적인 어레이 재구축은 나머지 디스크에 스트레스를 줍니다.

동일한 디스크를 여러 개 구매하면 모두 같은 날 또는 같은 배치에서 만들어질 수 있습니다. 제조 공장의 날씨가 좋지 않으면 큰 타격을 입을 수 있습니다. 안타깝게도 디스크 소매업체는 다른 날짜 또는 다

물리적 중복성(Physical Redundancy)

FreeBSD는 다중 경로 스토리지를 지원하므로 많은 하드웨어 문제를 해결할 수 있습니다. 하드웨어를 재배치하면 시스템의 가용성과 안정성을 높일 수 있습니다. 두 개의 외부 디스크 어레이가 있는

설치하기 전에 하드웨어를 살펴보세요. 하드웨어가 어떻게 고장날 수 있는지, 키트를 적절히 배치하면 어떻게 고장을 방지할 수 있는지 고려하세요.

디스크 설치 및 라벨링

라벨 기계를 사용하여 각 하드 드라이브에 작은 스티커를 붙이지 마세요. (물리적으로 디스크에 라벨을 붙여야 하지만) 그런 것들은 접착제가 절대 붙지 않습니다. FreeBSD는 하드 드라이브나 파

좋지만, 실제 장치로 다시 추적해야 합니다. 많은 ZFS 사용자들은 하드 드라이브가 많은 컴퓨터를 가지고 있기 때문에 문제가 더욱 악화됩니다.

FreeBSD 도구를 사용하면 고장난 드라이브의 일련 번호를 얻을 수 있습니다. 그러나 하드웨어에 따라 일련 번호를 확인하기 위해 각 드라이브를 물리적으로 검사해야 할 수도 있습니다. 이 작업에

설치 중에 미리 준비해 두면 장애가 발생한 디스크, 심지어 원격 시설에 있는 디스크도 바로 복구할 수 있습니다. 전 세계 여러 곳에서 매우 고밀도의 스토리지 어레이를 운영하는 Jude는 이 방식을 사

스토리지 어레이의 이름 지정 및 번호 지정 체계를 마련하세요. 대부분의 스토리지 어레이에는 표준 명명 체계가 있으며, 장비에 인쇄되어 있는 경우가 많습니다. 장비에 이미 번호가 매겨진 선반이 있다면

어레이에 설치할 때 각 드라이브의 일련 번호를 기록해 두세요. 각 드라이브 트레이를 설치할 때 실제 위치와 일련 번호에 따라 물리적으로 라벨을 붙입니다. 예, 이 작업이 지루하긴 하지만 결국에는 이 정

이제 FreeBSD를 설치하거나 라이브 미디어를 부팅합니다. 캠컨트롤 devlist를 사용해 모든 저장장치 목록을 확인한 다음, 각 저장장치 노드에서 diskinfo -v를 실행해 일련번호를

이제 장치 노드 목록과 관련 일련 번호, 물리적 위치 및 일련 번호 목록이 생겼습니다. 이 목록들을 서로 일치시킨 다음 GPT 레이블을 사용하여 사용 중인 디스크 파티션에 위치와 일련 번호를 붙이세

숫자가 가장 고유하므로 앞부분을 잘라내야 합니다.

예를 들어, 선반 2에 있는 디스크 9의 일련 번호가 WD-WCAW36477223이면 /dev/gpt/s2d9-WD-WCAW36477223과 같은 레이블이 표시될 수 있습니다.

시스템에서 이 레이블을 사용하려면 이 레이블만 사용해야 합니다. 시스템에서 GPTID 및 디스크 식별 레이블을 비활성화합니다. 이렇게 하면 나중에 혼동을 피할 수 있습니다. 이렇게 설정하면 하드웨

사전 계획을 세우면 가동 중단으로 인한 충격이 훨씬 줄어듭니다. 이를 적극 권장합니다.

챕터 1: ZFS 소개

시작하셨습니다. ZFS를 사용해 본 적이 없다면, 잠시 시간을 내어 테스트 시스템이나 가상 머신에 ZFS가 포함된 새로운 FreeBSD를 설치해 보세요. 암호화나 화려한 커스터마이징 옵션은 [

ZFS는 기존 파일 시스템과 볼륨 관리자의 기능을 결합한 것입니다. 따라서 개별 파일에 대한 권한과 어떤 파일이 어떤 디렉터리에 있는지부터 어떤 저장 장치가 어떤 용도로 사용되고 그 저장 장치가 어

방향을 잡기 위해 스토리지 스택에서 가장 눈에 잘 띄는 부분부터 시작해서 아래로 내려갑니다. 각 계층이 서로 어떻게 결합되어 있는지 이해했다면, 이 책의 나머지 부분은 기초부터 시작하여 위로 올라갑

ZFS 데이터 세트 (ZFS Datasets)

ZFS 파일시스템은 기존 파일시스템과 정확히 유사하지 않으므로 데이터 세트라고 부릅니다. 고전적인 [illegible] 스템일 [illegible] 과, 그리고 [illegible] 리 [illegible], [illegible], linux의 e
dump(8), restore(8) 및 이와 유사한 명령어를 사용하는 데 익숙하실 것입니다. ZFS는 이러한 모든 기능을 zfs(8) 프로그램에 통합하여 ZFS 데이터세트를 생성, 삭제,

zfslist로 기존 ZFS 데이터세트를 보는 것으로 시작하세요.

```
$ zfs list
NAME              USED  AVAIL  REFER  MOUNTPOINT
zroot            429M  13.0G    96K  none
zroot/ROOT        428M  13.0G    96K  none
zroot/ROOT/default  428M  13.0G   428M  /
zroot/tmp         104K  13.0G   104K
/tmp zroot/usr         428K  13.0G    96K  /usr
...
```

이것은 mount(8)과 df(1)의 출력을 결합한 것으로, UFS나 extfs를 관리해 본 사람이라면 누구나 익숙하게 볼 수 있을 것입니다.

각 데이터세트에는 이름이 있습니다. ZFS 데이터 세트 이름은 데이터 세트가 있는 ZFS storage polol, 즉 zpool로 시작합니다. 첫 번째 항목은 그냥 평범한 zroot라고 합니

다음 두 열은 사용된 공간과 사용 가능한 공간의 양을 보여줍니다. 풀 루트에는 429MB와 13GB의 여유 공간이 있습니다.

REFER 열은 ZFS에만 있는 특별한 열입니다. 이것은 데이터 집 [illegible] 서 [illegible] 세 [illegible] 할 수 [illegible] 42 [illegible] 데 [illegible] 터 [illegible] 을 [illegible] 양 [illegible] 에 [illegible] 대 [illegible] 한 [illegible] 정 [illegible] 보 [illegible] 입 [illegible] 니 [illegible] 다. [illegible] 공간의 [illegible] 양 [illegible] 과 [illegible] 는 [illegible] 다 [illegible] 르 [illegible] 며 [illegible] 이 [illegible] 부 [illegible] 분 [illegible] 은 [illegible] 잠 [illegible] 시 [illegible] 후 [illegible] 에 [illegible] 다 [illegible] 룹 [illegible] 니 [illegible] 다. [illegible] 풀 [illegible] 의 [illegible] 여 [illegible] 유 [illegible] 공 [illegible] 간 [illegible] 과 [illegible] 는 [illegible] 다 [illegible] 르 [illegible] 게 [illegible] 3 [illegible] 부 [illegible] 분 [illegible] 에 [illegible] 서 [illegible] 는 [illegible] 풀 [illegible] 의 [illegible] 여 [illegible] 유 [illegible] 공 [illegible] 간 [illegible] 이 [illegible] 데 [illegible] 이 [illegible] 터 [illegible] 세 [illegible] 트 [illegible] 에 [illegible] 서 [illegible] 사 [illegible] 용 [illegible] 할 [illegible] 수 [illegible] 있 [illegible] 는 [illegible] 공 [illegible] 간 [illegible] 과 [illegible] 는 [illegible] 다 [illegible] 릅 [illegible] 니 [illegible] 다. [illegible] 풀 [illegible] 의 [illegible] 여 [illegible] 유 [illegible] 공 [illegible] 간 [illegible] 은 [illegible] 모 [illegible] 든 [illegible] 데 [illegible] 이 [illegible] 터 [illegible] 세 [illegible] 트 [illegible] 가 [illegible] 공 [illegible] 유 [illegible] 하 [illegible] 고 [illegible] 있 [illegible] 습 [illegible] 니 [illegible] 다.
데이터 세트의 자식에는 이 책 전체에서 볼 수 있듯이 스냅샷, 볼륨 및 자식 데이터 세트가 포함됩니다.

마지막으로 [illegible] 일 [illegible] 에 [illegible] 템 [illegible] 이 [illegible] 운 [illegible] 트 [illegible] 는 [illegible] 지 [illegible] 점 [illegible] 은 [illegible] 트 [illegible] 되 [illegible] 었 [illegible] 습 [illegible] 니 [illegible] 다 [illegible] 않 [illegible] 았 [illegible] 습 [illegible] 니 [illegible] 다.

두 [illegible] 인 [illegible] **/ROOT**를 살펴보세요. 이것은 [illegible] 시 [illegible] 폴 [illegible] 을 [illegible] 만 [illegible] 들 [illegible] 었 [illegible] 을 [illegible] 때 [illegible] 마 [illegible] 운 [illegible] 트 [illegible] 되 [illegible] 지 [illegible] 않 [illegible] 게 [illegible] 메 [illegible] 트 [illegible] 입 [illegible] 니 [illegible] 다. [illegible] KB의 데이터를 참조합니다. 이것은 루트 파일 시스템에서는 이상하게도 사용되지 않는 것으

세 [illegible] 인 **/ROOT/default**는 현재 루트 파일시스템입니다. [illegible] 예 [illegible] 때 [illegible] 문 [illegible] 의 [illegible] 데 [illegible] 외 [illegible] 에 [illegible] 세 [illegible] 트 [illegible] 그 정도의 데이터가 있다는 뜻입니다.

ZFS가 이 데이터를 루트 파일 시스템에서 분리하는 이유는 무엇일까요? ZFS를 사용하면 여러 루트 파일시스템 중에서 쉽게 선택할 수 있습니다. 이 호스트는 FreeBSD 10.1을 실행하
zroot/ROOT/10.1-p1과 같은 새로운 데이터 세트를 생성하고 [illegible] 루트 [illegible] 으로 [illegible] 사용하 [illegible] 록 [illegible] 를 [illegible] 마 [illegible] 해 [illegible] 지 [illegible] 않 [illegible] 은 [illegible] 외 [illegible] 는 [illegible] 다른 위치에 마운트합니다. 업그레이드가 제대로

[illegible] **zroot/tmp**[illegible] **/tmp**[illegible].

ZFS [illegible] (ZFS partitions and properties)

ZFS[illegible]. [illegible], [illegible] LBA([illegible])[illegible]. [illegible].

[illegible].

ZFS[illegible]. [illegible], [illegible] **/var/log**[illegible]. ZFS [illegible], [illegible] ZFS [illegible]

[illegible] **quota** [illegible] ZFS [illegible]. [illegible] **zfs(8)** [illegible] ZFS[illegible]. ZFS [illegible]

```
$ zfs set quota=2G zroot/var/log
```

zfs get [illegible].

```
$ zfs get quota zroot/var/log
NAME           PROPERTY  VALUE  SOURCE
zroot/var/log  quota        2G  local
```

zfs get all[illegible] ZFS [illegible].
4[illegible] ZFS [illegible], 6[illegible].

ZFS [illegible](ZFS Limits)

[illegible]. [illegible] FAT [illegible] 32MB[illegible], [illegible] 2GB, 4GB[illegible]

ZFS [illegible] ZFS[illegible], [illegible]. ZFS [illegible] 48 [illegible] 78 [illegible] 64 [illegible] 256 [illegible] 128 [illegible] 64 [illegible] 2 [illegible] 2 [illegible]

[illegible]. [illegible]. [illegible] ZFS[illegible]

[illegible]

ZFS[illegible] **zpool(8)**[illegible], [illegible] FreeBSD [illegible]. [illegible]

```
$ zpool status
  pool: zroot
state: ONLINE
  scan: none requested
config:

NAME       STATE   READ WRITE CKSUM
zroot      ONLINE     0     0     0
 gpt/zfs0 ONLINE      0     0     0
```

```
errors: No known data errors
```

먼저 풀의 이름과 상태가 표시됩니다. 시스템에는 둘 이상의 ZFS 풀이 있을 수 있으며, 수십 개의 하드 드라이브가 있는 대규모 시스템에는 여러 개의 풀이 있는 경우가 많습니다. 이 호스트에 여러 개

ZFS는 스토리지 풀 [illegible] 가장 최근 검사 결과를 보여줍니다.

풀 목록의 마지막 부분에는 풀에 있는 가상 장치의 레이아웃이 표시됩니다.

가상 장치 (Virtual Devices)

스토리지 풀에는 하나 이상의 가상 장치, 즉 VDEV가 포함되어 있습니다. VDEV는 기존 RAID 장치와 유사합니다. 시스템 관리자가 실제로는 여러 개의 작은 디스크라는 것을 알고 있지만

가상 디바이스는 ZFS의 모든 마법이 일어나는 곳입니다. RAID 방식의 중복성을 위해 풀을 배열할 수 있습니다. 공급자를 전용 읽기 및 쓰기 캐시로 사용하여 가상 장치의 성능을 향상시킬 수 있습

ZFS의 데이터 중복성 및 자동 오류 수정도 VDEV 수준에서 이루어집니다. ZFS의 모든 데이터는 무결성 검증을 위해 체크섬 처리됩니다. 풀에 충분한 중복성이 있는 경우, ZFS는 자가 복

> ZFS는 백업의 필요성을 없애지 않습니다. 백업을 없애는 유일한 방법은 절대적인 무관심뿐입니다.

풀의 [illegible] 명령은 해당 풀의 가상 장치도 표시 [illegible] 예를 들어 [illegible] 공급자입니다. 2장에서 설명한 대로 ZFS FreeBSD는 암호화와 같은 기능을 지원하기 위해 GEOM 공급자를 사용합니다.

블록과 이노드 (Blocks and Inodes)

전통적인 파일시스템은 거의 항상 데이터를 저장하기 위해 다양한 데이터 블록을 사용하고 이러한 블록의 내용을 인덱스 노드로 매핑합니다. BSD의 UFS와 Linux의 extfs는 이러한 블록과

이러한 파일시스템과 마찬가지로 ZFS는 인덱스 블록과 데이터 블록을 사용합니다. 그러나 이전 파일시스템과 달리 ZFS는 필요에 따라 인덱스 노드를 생성합니다. ZFS는 가능할 때마다 데이터에

UFS 슈퍼블록과 달리 동적으로 생성된 인덱스 블록은 디스크의 알려진 위치에 배치할 수 없습니다. ZFS는 인덱스 블록이 손상될 가능성에 어떻게 대처할 수 있나요? ZFS는 중요한 인덱스 블록

이제 ZFS의 기본을 알았으니, 이 책의 나머지 부분에서는 수백 가지의 세부 사항을 설명합니다. 스택의 가장 아래쪽인 가상 장치부터 시작하겠습니다.

챕터 2 : 가상 장치 (Virtual Devices)

이 장에서는 소시지가 어떻게 만들어지는지 살펴보겠습니다. 이것은... 돼지, 즉 디스크입니다. 디스크는 스토리지의 물리적 표현입니다. 디스크는 사악합니다. 디스크는 그 특성과 레이아웃에 대해 거짓말

ZFS에서 사용 가능한 디스크를 가장 효과적으로 사용하려면 운영 체제가 ZFS에 디스크를 제공하는 방식과 ZFS가 해당 디스크의 데이터를 정렬하는 방식에 대한 기본적인 이해가 필요합니다.

디스크 및 기타 저장 매체 (Disks and Other Storage Media)

ZFS는 디스크가 아닌 다른 저장 매체에서도 실행할 수 있습니다. FreeBSD GEOM 저장소 제공자라면 무엇이든 ZFS 저장 매체가 될 수 있습니다. ZFS는 파일을 백업 스토리지로

원시 디스크 스토리 (Raw Disk Storage)

전체 물리적 디스크를 사용하면 복잡성이 줄어듭니다. 또한, 파티셔닝에 대해 걱정할 필요가 없으며, ZFS와 물리적 디스크 사이에 소프트웨어나 구성이 필요하지 않습니다. 그러나 일반적으로 이러한 장

디스크에서 부팅하려면 디스크에 부트 로더가 있어야 합니다. 부트 로더는 파티션된 디스크에서만 사용할 수 있습니다. 원시 디스크는 부팅할 수 없습니다. FreeBSD는 디스크에 유용한 레이블을 지

더 나쁜 점은, 대체 디스크는 원본 디스크와 정확히 같은 크기이거나 더 커야 한다는 것입니다. 6TB 디스크의 크기가 모두 같은 것은 아니며, 공급업체마다 디스크 크기가 몇 메가바이트씩 다릅니다.

파티션 스토리지

전체 원시 디스크를 사용하는 대신 디스크를 파티션한 다음 그 파티션 중 하나를 ZFS용으로 사용할 수 있습니다. 이 방법의 가장 큰 장점은 별도의 부팅 장치가 필요 없이 작은 부팅 파티션을 만들어 Z

많은 원본 Solaris ZFS 관리 가이드에서는 성능상의 이유로 파티션(또는 Solaris 용어로는 슬라이스)을 사용하지 말 것을 권장합니다. Solaris에서 파일 시스템에 파티션을 사용하면

FreeBSD는 여러 가지 파티셔닝 체계를 지원하지만, GPT를 강력히 권장합니다. 이전 파티셔닝 시스템인 MBR은 기본 파티션의 수를 4개로 제한한 반면, GPT는 최대 128개의 파
Mastery: Storage Essentials에서는 두 가지 파티셔닝 방법에 대한 FreeBSD의 지원을 다룹니다.

> 클레이 태블릿에 데이터를 저장하는 경우, bsdlabe(8) 파티션을 사용할 수 있습니다.

파티션 사용의 단점은 ZFS가 제공하는 일부 이식성을 잃을 수 있다는 것입니다. 한 시스템에서 다른 시스템으로 디스크를 이동하는 경우, 대상 시스템이 디스크 파티션을 인식할 수 있어야 합니다.

GEOM 장치 스토리지 (GEOM Device Storage)

ZFS는 다양한 FreeBSD GEOM 클래스를 백업 스토리지로 사용할 수도 있습니다. 이러한 클래스는 파일 시스템과 물리적 장치 사이에 위치하며 다양한 기능을 수행합니다. GEOM

암호화된 ZFS 풀을 생성하는 가장 좋은 방법은 GELI(FreeBSD Disk Encryption Subsystem)입니다. GELI는 ZFS와 물리적 디스크 사이를 오갈 때 블
NI(AES New Instructions)를 지원하는 최신 CPU에서 최상의 성능을 제공합니다. 이 하드웨어 오프로드 기능의 도움으로 GELI는 초당 1GB 이상의 속도로 데이터를 암
Authentification Code, HMAC)를 저장하는 데이터 인증(무결성 검증) 기능을 선택적으로 제공할 수 있습니다. 이 HMAC를 사용하여 데이터의 무결성(데이터가 변조되지

고가용성 스토리지 기술(High Availability Storage Technology, HAST)은 FreeBSD의 분산 스토리지 솔루션입니다. 네트워크를 통해 컴퓨터 간에 블록
호스트에 쓰여질 때까지 쓰여진 것으로 간주되지 않습니다. 반면에 ZFS 복제는 주기적인 스냅샷 동기화를 기반으로 합니다. 그러나 HAST를 사용하면 두 번째 머신은 첫 번째 머신과 동시에 풀을

GEOM 레이블은 각 디스크나 파티션에 의미 있는 메모를 첨부할 수 있는 편리한 방법을 제공합니다. disk ident, gptid, GPT 레이블과 같은 표준과 GEOM-specific

GEOM은 고가용성을 위한 다중 경로도 지원합니다. 디스크뿐만 아니라 컨트롤러 카드, 백플레인 또는 케이블도 고장이 나는 경우가 있습니다. 다중 경로를 사용하면 '듀얼 포트'인 엔터프라이즈 드라 대한 여러 경로로 뒷받침됩니다. 다중 경로에 대해서는 FreeBSD Mastery: Advanced ZFS에서 다중 경로에 대해 설명합니다.

관리자는 GEOM 스케줄러 모듈을 사용하여 더 나은 성능을 달성하기 위해 다양한 I/O 스 gsche 8) 을 참조하세요. (이 [illegible]

파일 백업 스토리지 (File-Backed Storage)

파일 백업 가상 디스크를 ZFS 저장 장치로 사용할 수 있습니다. 프로덕션 환경에서는 권장하지 않지만, 파일 백업 디스크는 테스트 및 실험용으로 유용할 수 있습니다.

프로바이더와 디스크(Providers vs. Disks)

"프로바이더"는 FreeBSD의 기술 용어입니다. GEOM 스토리지 제공자는 데이터 스토리지를 제공하는 것을 말합니다. 디스크일 수도 있습니다. 어떤 식으로든 저장소를 변환하는 GEOM

> FreeBSD의 유연한 스토리지 시스템은 멍청한 짓을 할 수 있는 힘을 제공합니다. 하지 마세요.

이 책에서 "디스크"에 대해 설명하는 것은 "디스크 위에 있는 일종의 공급자"를 의미합니다. 디스크에 스왑 파티션과 ZFS 파티션이 있어도 문제없지만, 이 디스크가 전적으로 ZFS 전용일 필요는

VDEV: 가상 장치 (VDEVs: Virtual Devices)

가상 장치 또는 VDEV는 ZFS의 논리적 저장 장치입니다. 각 VDEV는 하나 이상의 GEOM 공급자로 구성됩니다. ZFS는 여러 가지 유형의 VDEV를 지원하며, 이는 VDEV RAID가 있습니다.

이러한 VDEV는 3장에서 설명한 스토리지 풀로 배열됩니다. 실제 데이터는 4장에서 다루는 것처럼 풀 위에 저장됩니다. 하지만 가상 디바이스의 배열에 따라 풀의 성능과 물리적 손상에 대한 저항력

스토리지 풀은 하나 이상의 VDEV로 구성되며, 풀 데이터는 중복성 없이 해당 VDEV에 분산됩니다. (4장에서 설명한 대로 복사 속성을 사용하여 중복성을 추가할 수 있지만 전체 디스크 장애에

한 풀에 여러 개의 VDEV를 사용하면 고급 RAID 어레이와 유사한 시스템이 만들어집니다. RAID-Z2 어레이는 RAID-6과 유사하지만, 두 개의 RAID-Z2 VDEV가 있는 ZFS는 중복성 없이 각 VDEV에 걸쳐 데이터를 스트라이핑합니다. 개별 VDEV가 중복성을 제공합니다.

VDEV 중복성 (VDEVs Redundancy)

둘 이상의 디스크가 포함된 VDEV는 여러 가지 중복성 체계를 사용하여 내결함성을 제공할 수 있습니다. 단일 디스크를 단독으로 이중화할 수 있는 방법은 없습니다. ZFS는 미러 디스크와 여러 패 ZFS는 중복성을 사용하여 자가 복구합니다. copi 없 속성 [illegible] 가 복 지 하 [illegible] 문제를 해결할 수 있지만(

Stripe (1개 공급자)

단일 디스크로 구성된 VDEV를 스트라이프라고 하며, 중복성이 없습니다. 예상할 수 있듯이 단일 공급자가 손실되면 디스크의 모든 데이터가 사라집니다. 스트라이프 풀에는 단일 디스크 VDEV만

ZFS 풀은 풀의 모든 VDEV에 걸쳐 데이터를 스트라이핑하고 중복성을 제공하기 위해 VDEV에 의존합니다. 하나의 스트라이프 디바이스에 장애가 발생하면 전체 풀이 장애가 발생합니다. 풀에

Mirrors (2개 이상의 공급자)

미러 VDEV는 모든 디스크에 모든 데이터의 전체 사본을 저장합니다. 공급자의 드라이브 중 하나를 제외한 모든 드라이브를 잃어버려도 데이터에 액세스할 수 있습니다. 미러에는 원하는 수의 디스크를

미러는 모든 디스크에서 데이터를 한 번에 읽을 수 있기 때문에 랜덤 및 순차 읽기 속도가 매우 우수합니다. 모든 데이터를 모든 디스크에 써야 하고 가장 느린 디스크가 끝날 때까지 작업이 완료되지 않기

RAID-Z1(3개 이상의 공급자)

ZFS에는 RAID-Z라고 하는 3개의 최신 RAID 스타일 중복 VDEV가 포함되어 있습니다. RAID-Z는 RAID-5와 비슷하지만 파일 무결성을 보장하기 위해 체크섬을 포함합니

RAID-Z는 데이터 및 패리티 정보를 모든 디스크에 분산시킵니다. RAID-Z의 공급자가 죽거나 손상된 데이터를 제공하기 시작하면 RAID-Z는 패리티 정보를 사용하여 누락된 데이터를 다

RAID-Z1 VDEV는 단일 스토리지 공급업체의 장애를 견딜 수 있습니다. 첫 번째 장애가 발생한 드라이브가 교체되기 전에 두 번째 공급자가 장애를 일으키면 모든 데이터가 손실됩니다. 패리티

RAID-Z2(4개 이상의 공급자

RAID-Z2는 RAID-Z1과 유사하지만, VDEV당 2개의 패리티 디스크가 있습니다. RAID-6과 마찬가지로 RAID-Z2는 두 개의 공급자가 실패하더라도 계속 작동할 수 있습

RAID-Z3(5개 이상의 공급자)

RAID-Z의 가장 편집증적인 형태인 RAID-Z3는 VDEV당 3개의 패리티 디스크를 사용합니다. 즉, 5개의 디스크 어레이에 3개의 장애 디스크를 사용할 수 있습니다. RAID-Z2

RAID-Z 디스크 구성 (RAID-Z Disk Configurations)

모든 버전의 RAID-Z를 사용할 때 기억해야 할 중요한 사항 중 하나는 RAID-Z의 공급자 수가 완전히 고정되어 있다는 것입니다. RAID-Z VDEV에 드라이브를 추가하여 확장할 수

20개의 하드 드라이브를 수용할 수 있는 호스트가 있다고 가정해 보겠습니다. 나중에 필요에 따라 드라이브를 풀에 추가할 수 있다고 생각하고 12개의 드라이브를 설치하여 단일 RAID-Z2로 사
생각입니다.

미리 계획하세요. 물리적 장비, 시작해야 하는 드라이브 수, 스토리지를 확장하는 방법을 고려하세요. 예제 서버는 하나의 RAID-Z2 VDEV를 포함하는 온풀과 나머지 8개의 디스크를 포함하

RAID-Z 2의 규칙 (The RAID-Z Rule of 2s)

일반적으로 논의되는 구성 중 하나는 2의 배수만큼의 데이터 디스크 수와 주어진 RAID-Z 레벨에 필요한 패리티 디스크를 더하는 것입니다. 즉, 이 규칙에 따르면 RAID-Z1은 2n+1 [
등)을 사용해야 합니다. 이 규칙은 데이터가 2의 거듭제곱인 크기의 작은 블록으로 구성된 경우에만 유효합니다. 하지만 다른 요인들이 훨씬 더 큰 차이를 만듭니다. 일반적으로 압축이 훨씬 더 효과적인 [

VDEV 복구 (Repairing VDEVs)

중복 VDEV에 속한 공급자가 장애가 발생하면, 해당 공급자가 속한 VDEV는 "degraded(성능 저하)" 상태가 됩니다. 성능이 저하된 VDEV는 여전히 모든 데이터를 보유하지만

공급자가 교체된 후에는 시스템이 새 공급자에 데이터를 저장해야 합니다. 미러를 사용하면 남은 디스크에서 데이터를 읽고 교체된 디스크에 쓰면 됩니다. RAID-Z의 경우, 데이터를 패리티에서 다시 [

ZFS가 RAID와 파일 시스템을 결합하는 방식은 ZFS가 어떤 블록에 데이터가 들어 있고 어떤 블록이 비어 있는지를 알고 있다는 것을 의미합니다. 새 드라이브에 모든 바이트를 기록할 필요 없

RAID-Z와 기존 RAID 비교(RAID-Z versus Traditional RAID)

RAID-Z는 기존 RAID에 비해 여러 가지 장점이 있지만, 가장 큰 장점은 ZFS가 디스크 중복성 계층 외에 볼륨 관리자이자 파일 시스템이라는 사실에서 비롯됩니다.

과거에는 파일시스템이 하나의 디스크에서만 작동할 수 있었습니다. 디스크가 두 개라면 두 개의 파일시스템이 필요했습니다. 기존 RAID를 사용하면 여러 디스크를 하나의 가상 디스크로 결합하여 10

파일 시스템과 볼륨 관리자를 결합함으로써 ZFS는 모든 데이터가 어디에 있는지, 스토리지 계층과 데이터가 어떻게 상호 작용하는지 정확히 파악할 수 있습니다. 이를 통해 ZFS는 디토 블록(3장)[

기존 RAID는 2단계 작업이 중간에 중단되는 "쓰기 구멍(write hole)"이라는 단점이 있습니다. RAID 5 및 6 장치는 모든 데이터 디스크에 기록할 데이터를 청크 단위로 묶습니다.

쓰기 구멍 문제는 고장난 디스크를 교체하기 전까지는 발견되지 않습니다. 패리티가 잘못되었거나 미러가 잘못되면 RAID 장치가 파일시스템에 가비지 데이터를 반환하게 됩니다. 기존 파일시스템은 이 [

ZFS는 쓰기 시 복사 및 체크섬으로 이러한 문제를 해결합니다. 쓰기 시 덮어쓰기(Copy-on-write, 7장)는 데이터를 제자리에서 덮어쓰지 않는다는 의미입니다. 각 업데이트는 트랜잭션 [

특수 VDEV (Special VDEVs)

풀은 풀의 성능을 향상시키기 위해 특수 목적의 VDEV를 사용할 수 있습니다. 이러한 특수 VDEV 유형은 데이터를 영구적으로 저장하는 데 사용되지 않고, 대신 더 빠른 장치에 데이터의 추가 [

별도의 인텐트 로그(Seperate Intent Log; SLOG, ZIL)

ZFS는 풀의 일부로 ZFS 인텐트 로그(ZIL)를 유지합니다. 다른 파일시스템의 저널과 마찬가지로 진행 중인 작업을 기록하는 곳으로, 시스템 충돌이나 정전 시 작업을 완료하거나 롤백할 수 있습

성능을 향상시키는 한 가지 방법은 ZIL을 일반 풀 작업에서 분리하는 것입니다. 풀의 일반 부분을 사용하는 대신 전용 장치를 별도의 인텐트 로그(SLOG)로 사용할 수 있습니다. 전용 장치는 일반

ZFS는 데이터를 수신한 SLOG를 순서대로 풀의 메인 스토리지에 복사하는 대신 데이터를 합리적인 그룹으로 일괄 처리하여 보다 효율적으로 쓸 수 있습니다.

특정 소프트웨어는 디스크에 쓰는 데이터가 실제로 디스크에 있는지 확인을 받은 후에 작업을 진행합니다. 데이터베이스는 시스템 충돌이나 정전 시 손상을 방지하기 위해 이 작업을 수행하는 경우가 많습니

ZIL을 미러링하여 데이터 손실을 방지할 수 있습니다.

캐시(Cache: L2ARC)

디스크에서 파일을 읽을 때 시스템은 다른 용도로 메모리가 필요할 때까지 파일을 메모리에 보관합니다. 이것은 초기 BSD 시절에도 사용된 오래된 기술입니다. UFS 기반 BSD 시스템의 상단(1

하지만 전통적인 버퍼 캐시는 수십 년 전에 설계되었습니다. ZFS에는 최신 하드웨어용으로 설계된 적응형 대체 캐시(ARC)가 있어 속도가 향상됩니다. ARC는 가장 최근에 자주 액세스한 파일

하지만 최신 시스템 중 원하는 만큼 캐시를 저장할 수 있는 충분한 RAM을 갖춘 시스템은 거의 없습니다. ZFS가 SLOG를 사용해 쓰기를 가속화하는 것처럼, 매우 빠른 디스크를 사용해 읽기

개체가 캐싱의 이점을 누릴 만큼 자주 사용되지만 RAM에 저장할 만큼 자주 사용되지는 않는 경우 ZFS는 캐시 장치에 개체를 저장할 수 있습니다. L2ARC는 일반적으로 매우 빠르고 내구성이

VDEV가 성능에 미치는 영향 (How VDEVs Affect Performance)

각기 다른 유형의 VDEV는 서로 다른 성능을 발휘합니다. 디스크 성능을 벤치마킹하고 분석하는 것은 매우 복잡한 주제이므로, 누군가 귀찮아서 읽지 않는다면 커다란 교과서 한 권이 필요할 것입니다.

일반적인 측정 방법 중 하나는 초당 입출력 횟수, 즉 드라이브가 초당 수행할 수 있는 고유한 작업의 수를 나타내는 초당 입출력 횟수(IOPS)입니다. 회전식 드라이브의 IOPS는 일반적으로 읽기/[

비패리티 스핀들의 수는 손상되지 않은 풀의 스트리밍 읽기 및 쓰기 성능을 제한합니다. "스트리밍" 성능은 드라이브가 읽거나 쓸 수 있는 초당 메가바이트(MB/s)의 수로 요약됩니다. 드라이브가 데

한 번에 디스크의 더 많은 다른 위치에서 읽거나 쓸 수 있다는 것을 의미하며, 그 결과 IOPS 성능이 향상됩니다. 스핀들이 많다는 것은 더 많은 디스크가 최대한 빠르게 작동하여 데이터를 읽고 쓸 수

다른 일반적인 측정 항목으로는 읽기 대역폭, 쓰기 대역폭, 공간 효율성, 스트리밍 성능 등이 있습니다.

일반적으로 미러는 더 나은 IOPS와 읽기 대역폭을 제공할 수 있지만, RAID-Z는 더 나은 쓰기 대역폭과 훨씬 더 나은 공간 효율성을 제공할 수 있습니다.

여러 개의 VDEV가 있는 풀은 모든 VDEV에 걸쳐 데이터를 스트라이핑합니다. 이렇게 하면 성능이 향상되지만, 각 VDEV마다 중복 디스크가 있기 때문에 공간을 많이 차지할 수 있습니다.

몇 가지 일반적인 VDEV 구성을 살펴보고 가능한 다양한 배열이 성능과 용량에 어떤 영향을 미치는지 알아보겠습니다. 일반적인 1TB 스피닝 디스크 세트를 사용한다고 가정해 보겠습니다. 각 디스[

디스크 하나 (One Disk)

디스크가 하나뿐인 경우 가능한 구성은 단일 ZFS 스트라이프 VDEV 하나뿐입니다. 이 구성은 가장 기본적인 구성이며 내결함성을 제공하지 않습니다. 디스크 하나가 죽으면 모든 데이터가 사라집니

Table 1. Single Disk Virtual Device Configurations

Disks	Config	Read IOPS	Write IOPS	Read MB/s	Write MB/s	Usable Space	Fault Tolerance
1	Stripe	250	250	100	100	1 TB (100%)	none

단일 디스크 스트라이프 장치의 성능 특성은 기본 디스크의 특성과 의심스러울 정도로 비슷합니다. 이상하지 않나요?

두 개의 디스크 (Two Disks)

시스템에 두 개의 디스크가 있는 경우, 두 개의 스트라이프 VDEV 또는 하나의 미러 VDEV로 풀을 구성할 수 있습니다.

스트라이프 VDEV는 사용 가능한 스토리지와 대역폭을 두 배로 늘리지만 장애 위험도 두 배로 증가합니다. ZFS는 각 파일의 블록을 두 개의 디스크에 분산시킵니다. 디스크 중 하나에 장애가 발생

미러링된 단일 VDEV를 사용하면 각 데이터 블록이 두 디스크에 모두 저장됩니다. 이렇게 하면 두 디스크에서 한 번에 블록을 읽을 수 있으므로 읽기 성능이 향상됩니다. 하지만 가장 작은 디스크의 [

Table 2: Two-Disk Virtual Device Configurations

Disks	Config	Read IOPS	Write IOPS	Read MB/s	Write MB/s	Usable Space	Fault Tolerance
2	2 x Stripe	500	500	200	200	2 TB (100%)	none
2	1 x 2 disk Mirror	500	250	200	100	1 TB (50%)	1

표에서 볼 수 있듯이 미러 풀은 스트라이프 풀의 쓰기 성능은 절반이고 공간은 절반입니다.

디스크 3개 (Three Disks)

디스크 3개는 더 깊은 미러와 RAID-Z를 포함하여 더 많은 옵션을 의미합니다. 스트라이프 디스크 3개로 구성된 풀을 사용할 수도 있지만 실패 확률이 훨씬 높습니다.

더 깊은 미러에는 더 많은 디스크가 있으므로 더 많은 내결함성과 향상된 읽기 성능을 제공합니다. 스핀들과 헤드가 많다는 것은 VDEV가 세 개의 드라이브 중 가장 사용량이 적은 드라이브에서 데이터

RAID-Z1은 내결함성을 위해 VDEV의 디스크 중 하나만 필요하므로 공간 효율성이 더 뛰어납니다. 데이터는 모든 드라이브에 분산되어 있으므로 읽기 및 쓰기를 수행하려면 함께 작동해야 합니[

Table 3: Three-Disk Virtual Device Configurations

Disks	Config	Read IOPS	Write IOPS	Read MB/s	Write MB/s	Usable Space	Fault Tolerance
3	1 x 3 disk Mirror	750	250	300	100	1 TB (33%)	2

3	1 x 3 disk RAID-Z1	250	250	200	200	2 TB (66%)	1

여기서 주의할 점은 IOPS가 반드시 실제 읽기/쓰기 성능으로 확장되는 것은 아니라는 점입니다. 미러 VDEV는 각 드라이브의 헤드가 독립적으로 작동할 수 있는 반면, RAID-Z에서는 헤드[
RAID-Z1은 모든 디스크에 쓰기를 분산할 수 있지만 패리티로 인해 처리량이 일부 손실됩니다.

디스크 4개 또는 5개 디스크 (Four or Five Disks)

4개 또는 5개의 디스크를 사용하면 더 많은 옵션을 사용할 수 있습니다.

다중 미러 VDEV(기존 RAID 10과 유사)는 데이터베이스와 같은 랜덤 I/O 워크로드에 최상의 성능을 제공합니다. 4개의 디스크를 각각 2개의 디스크가 있는 2개의 미러 VDEV로 [

디스크가 4개이면 RAID-Z2가 옵션이 됩니다. RAID-Z2의 패리티 디스크 2개는 디스크 2개가 손실되더라도 VDEV가 계속 작동할 수 있음을 의미합니다. 디스크 수가 같은 미러와 [

5개의 디스크로 RAID-Z3를 구축할 수 있습니다. RAID-Z3 VDEV는 디스크 3개가 모두 손실되더라도 살아남을 수 있습니다. RAID-Z3는 성능과 내결함성을 교환합니다.

물론 RAID-Z1은 여전히 옵션으로 남아 있습니다.

Table 4: Four- or Five-Disk Virtual Device Configurations

Disks	Config	Read IOPS	Write IOPS	Read MB/s	Write MB/s	Usable Space	Fault Tolerance
4	2 x 2 disk Mirror	1000	500	400	200	2 TB (50%)	2 (1/VDEV)
4	1 x 4 disk RAIDZ-Z1	250	250	300	300	3 TB (75%)	1
4	1 x 4 disk RAIDZ-Z2	250	250	200	200	2 TB (50%)	2
5	1 x 5 disk RAIDZ-Z1	250	250	400	400	4 TB (80%)	1
5	1 x 5 disk RAIDZ-Z2	250	250	300	300	3 TB (60%)	2
5	1 x 5 disk RAIDZ-Z3	250	250	200	200	2 TB (40%)	3

RAID-Z1의 스트리밍(MB/s) 읽기 및 쓰기 성능이 RAID-Z2와 어떻게 비교되는지, RAID-Z3의 성능은 둘 다와 어떻게 비교되는지 참고하세요. 패리티 디스크를 추가하면 해당 [

다중 미러 VDEV의 내결함성은 약간 까다롭습니다. 중복성은 풀 단위가 아니라 VDEV 단위라는 점을 기억하세요. 각 미러 VDEV는 여전히 n - 1의 내결함성을 제공합니다. 각 미러 VI

6~12개의 디스크 (Six to Twelve Disks)

디스크 수가 많으면 내결함성, 공간 효율성, 성능의 균형을 맞추는 것이 중요합니다.

6개의 디스크를 3개의 2 디스크 미러 VDEV로 구성하면 상당한 공간과 우수한 쓰기 성능을 확보할 수 있습니다. 한 쌍의 3-디스크 미러 VDEV를 선택하면 공간은 더 적지만 데이터 손실 [

디스크를 6개 이상 확보하면 한 풀에 여러 개의 RAID-Z VDEV를 사용할 수 있습니다. 12개의 디스크를 단일 VDEV로 함께 운영하여 가용 공간을 최대한 확보하거나, 두 개의 개별 \

Table 5: Six- to Twelve-Disk Virtual Device Configurations

Disks	Config	Read IOPS	Write IOPS	Read MB/s	Write MB/s	Usable Space	Fault Tolerance
6	3 x 2 disk Mirror	1500	750	600	300	3 TB (50%)	3 (1/VDEV)
6	2 x 3 disk Mirror	1500	500	600	200	2 TB (33%)	4 (2/VDEV)
6	1 x 6 disk RAIDZ-Z1	250	250	500	500	5 TB (83%)	1
6	1 x 6 disk RAIDZ-Z2	250	250	400	400	4 TB (66%)	2
6	1 x 6 disk RAIDZ-Z3	250	250	300	300	3 TB (50%)	3
12	6 x 2 disk Mirror	3000	1500	1200	600	6 TB (50%)	6 (1/VDEV)
12	4 x 3 disk Mirror	3000	1000	1200	400	4 TB (33%)	8 (2/VDEV)
12	1 x 12 disk RAIDZ-Z1	250	250	1100	1100	11 TB (92%)	1
12	2 x 6 disk RAIDZ-Z1	500	500	1000	1000	10 TB (83%)	2 (1/VDEV)
12	3 x 4 disk RAIDZ-Z1	750	750	900	900	9 TB (75%)	3 (1/VDEV)
12	1 x 12 disk RAIDZ-Z2	250	250	1000	1000	10 TB (83%)	2
12	2 x 6 disk RAIDZ-Z2	500	500	800	800	8 TB (66%)	4 (2/VDEV)
12	1 x 12 disk RAIDZ-Z3	250	250	900	900	9 TB (75%)	3
12	2 x 6 disk RAIDZ-Z3	500	500	600	600	6 TB (50%)	6 (3/VDEV)

풀에서 여러 RAID-Z 장치를 사용하는 것은 풀에서 여러 미러 장치를 사용하는 것과 매우 유사합니다. 디스크 장애에 대한 허용 오차는 풀이 아닌 VDEV별로 적용됩니다. 6개의 디스크 RAI

많은 디스크 (Many Disks)

일반적으로 VDEV당 9~12개 이하의 디스크를 사용하는 것이 좋습니다. 더 많이 사용할 수도 있지만 ZFS는 그렇게 설계되지 않았습니다. 36개의 디스크를 배열하여 몇 가지 가능한 배열과

Table 5: Six- to Twelve-Disk Virtual Device Configurations

Disks	Config	Read IOPS	Write IOPS	Read MB/s	Write MB/s	Usable Space	Fault Tolerance

36	18 x 2 disk Mirror	9000	4500	3600	1800	18 TB (50%)	18 (1/VDEV)
36	12 x 3 disk Mirror	9000	3000	3600	1200	12 TB (33%)	24 (2/VDEV)
36	1 x 36 disk RAID-Z2	250	250	3400	3400	34 TB (94%)	2
36	2 x 18 disk RAID-Z2	500	500	3200	3200	32 TB (89%)	4 (2/VDEV)
36	4 x 9 disk RAID-Z2	1000	1000	2800	2800	28 TB (78%)	8 (2/VDEV)
36	6 x 6 disk RAID-Z2	1500	1500	2400	2400	24 TB (66%)	12 (2/VDEV)

더 많은 VDEV를 사용하면 매우 빠른 풀을 만들 수 있습니다. 18개의 디스크 미러 VDEV로 구성된 풀은 그 어떤 것보다 더 빠르게 데이터를 읽을 수 있으며, 장애가 발생하기 전에 18개의

각 VDEV에 패리티 또는 미러를 추가하면 안정성이 향상됩니다. VDEV의 수가 많을수록 성능이 향상됩니다. 여러분의 역할은 이 두 가지 특성을 적절히 조합하여 환경을 지원하는 것입니다.

각 VDEV는 가장 느린 디스크의 랜덤 읽기/쓰기 성능으로 제한되므로 하나의 VDEV에 디스크가 너무 많으면 공간 효율성을 조금만 높이기 위해 성능을 포기하는 것입니다. 성능을 개선하기 위해

그렇다면 VDEV가 많으면 많을수록 좋은데, 왜 6 x 6 디스크 RAID-Z2 풀이 1 x 36 디스크 RAID-Z2 풀에 비해 읽기 및 쓰기 속도가 훨씬 느린 것일까요? 답은 내결함성 | 패리티에 사용된 디스크는 더 이상 스트리밍 성능에 기여하지 않습니다. 디스크를 더 추가하여 해당 성능을 복원할 수 있습니다. 6 x 8 디스크 RAID-Z2 풀은 36개의 데이터 디스크와 12[

VDEV에 대해 알고 있는 내용을 바탕으로 실제 풀을 만들어 보겠습니다.

챕터 3. 풀(Pools)

ZFS 풀 또는 zpool은 ZFS 스택의 중간을 형성하며, 하위 수준의 가상 장치를 사용자가 볼 수 있는 파일 시스템에 연결합니다. 풀 보기 시스템의 모든 풀을 보려면 스토리지 블록의 zpool

ZFS 블록

UFS 및 extfs와 같은 기존 파일시스템은 디스크에 데이터를 고정 크기 블록으로 저장합니다. 파일 시스템에는 아이노드(inode)라고 하는 특수 블록이 있으며, 이 블록은 어떤 블록이 어떤 [

ZFS는 특별한 인덱스 블록을 미리 구성하지 않습니다. 스트라이프라고도 하는 스토리지 블록만 사용합니다. 각 블록에는 블록을 디스크의 다른 블록과 트리로 연결하기 위한 인덱스 정보가 포함되어 있습

전용 인덱스 블록이 없어도 좋지만, ZFS는 분명 어딘가에서 시작해야 합니다! 모든 데이터 트리에는 루트가 필요합니다. ZFS는 우버블록이라는 특수 블록을 사용해 파일 시스템 루트에 대한 포인터[

우버블록만이 중요한 블록은 아닙니다. ZFS는 파일시스템 메타데이터 및 풀 데이터와 같은 중요한 정보가 포함된 블록을 여러 개의 동일한 블록에 복사합니다. 메인 블록이 손상되면 ZFS는 디토 블

ZFS는 트랜잭션 그룹 또는 txg의 스토리지 미디어에 변경 내용을 커밋합니다. 트랜잭션 그룹은 여러 개의 일괄 변경 내용을 포함하며 64비트 숫자가 증가합니다. 각 트랜잭션 그룹은 라인의 다음

ZFS는 인덱싱을 위해 일부 블록을 사용하지만, 이러한 z노드와 d노드는 풀에 있는 모든 스토리지 블록을 사용할 수 있습니다. 파일시스템을 만들 때 할당되는 UFS2 또는 extfs 인덱스 노드

스트라이프, RAID 및 풀 (Stripes, RAID, and Pools)

스토리지와 관련하여 스트라이프라는 단어를 여러 번 들어보셨을 것입니다. ZFS 풀은 가상 장치에 걸쳐 데이터를 "스트라이프"합니다. 전통적인 RAID는 물리적 장치에 걸쳐 데이터를 "스트라이프"

스트라이프는 단일 장치에 기록되는 데이터 덩어리입니다. 대부분의 기존 RAID는 128KB 스트라이프 크기를 사용합니다. 기존 RAID 장치에 파일을 쓸 때 RAID 소프트웨어는 보통 병렬

스트라이프는 중복성을 제공하지 않습니다. 기존 RAID는 패리티 및/또는 미러링을 통해 중복성을 확보합니다. ZFS 풀은 기본 VDEV에서 중복성을 확보합니다.

ZFS는 로켓으로 구동되는 롤러스케이트에 줄무늬를 넣습니다. ZFS 데이터 세트는 기본 스트라이프 크기인 128KB를 사용하지만, ZFS는 장비와 워크로드에 맞게 이 스트라이프 크기를 동

사용할 수 있으며, FreeBSD 10.2 이상에 포함될 예정입니다.

ZFS 풀은 기존 RAID보다 훨씬 더 유연합니다. 전통적인 RAID는 고정적이고 유연하지 않은 데이터 레이아웃을 가지고 있습니다(일부 하드웨어 벤더는 유연성이 더 높은 자체 독점 RAID [

ZFS 풀은 변경을 허용할 뿐만 아니라 추가도 쉽게 허용하도록 설계되었습니다. 다섯 개의 VDEV가 있는 ZFS 풀이 있는데 여섯 번째를 추가하려는 경우 괜찮습니다. ZFS는 해당 VDEV

하지만 ZFS를 사용하면 해당 가상 장치는 ZFS가 지원하는 모든 유형의 VDEV가 될 수 있습니다. 미러 쌍인 두 개의 VDEV를 예로 들어보겠습니다. 하나의 zpool에 넣습니다. ZF

VDEV를 추가할 수 있습니다. RAID-10을 확장한다는 것은 데이터를 백업하고, RAID 어레이에 디스크를 추가하고, 데이터를 복원하는 것을 의미합니다. zpool을 확장한다는 것은 풀

하지만 풀은 중복성을 제공하지 않는다는 점을 기억하세요. 모든 ZFS 중복성은 기본 VDEV에서 제공됩니다.

풀 보기 (Viewing Pools)

시스템의 zpool을 list를 실행합니다.

```
$ zpool list
NAME   SIZE  ALLOC  FREE  EXPANDSZ  FRAG  CAP  DEDUP  HEALTH  ALTROOT
db    2.72T  1.16G  2.72T       -    0%   0%  1.00x  ONLINE  -
zroot  920G  17.3G  903G        -    2%   1%  1.00x  ONLINE  -
```

첫 번째 열 db 와 zroot 라는 두 개의 풀이 있습니다.

다음 세 열은 각 풀의 크기와 사용량 정보를 제공합니다. 크기, 사용된 공간의 양, 사용 가능한 공간의 양이 표시됩니다.

EXPANDSZ 열은 기본 스토리지 공급업체에 여유 공간이 있는지 여부를 보여줍니다. 5장에서 설명한 대로 이 풀의 공간을 확장할 수 있습니다. 이 공간에는 패리티 정보로 이동하는 블록이 [

FRAG 아래에서 이 풀의 조각화 양을 볼 수 있습니다. 조각화는 파일 시스템 성능을 저하시킵니다.

CAP 열은 사용 가능한 공간의 몇 퍼센트가 사용되었는지 보여줍니다.

DEDUP 항목은 파일 시스템에서 발생한 중복 제거의 양을 보여줍니다. 6장에서는 중복 제거에 대해 다룹니다.

HEALTH 열은 기본 VDEV의 상태를 반영합니다. 스토리지 공급업체에 장애가 발생하면 첫 번째 힌트는 온라인 이외의 모든 상태입니다. 5장에서는 풀 상태에 대해 설명합니다.

마지막 **ALTROOT**는 이 풀이 마운트된 위치, 즉 "대체 루트"를 보여줍니다.

4장에서는 대체 루트에 대해 다룹니다. 특정 풀에 대한 정보를 알고 싶으면 zpool list 뒤에 풀 이름을 나열합니다.

이 명령은 prod 와 test 의 출력만 표시합니다.

```
$ zpool list prod test
```

기본 드라이브 -V 사용률을 포함하여 풀에 [illegible] 합니다.

```
$ zpool list -v zroot
```

-p 플래그는 사람이 [illegible] 관리 스크립트에 유용합니다.

기본 VDEV 레이아웃 zpool status 를 사용하여 풀을 만들 때 zpool 상태의 많은 예를 보게 될 것입니다.

풀을 간단히 확인하려면 zpool status -x를 실행합니다.

```
$ zpool status -x
all pools are healthy
```

때로는 이것만으로 충분할 때도 있습니다.

여러 개의 VDEV (Multiple VDEVs)

풀에는 여러 개의 VDEV가 포함될 수 있습니다. VDEV를 추가하면 풀에서 사용 가능한 공간이 늘어날 뿐만 아니라 성능도 향상됩니다. 풀은 모든 쓰기를 VDEV 간에 분할합니다. 작은 파

2장에서는 다양한 VDEV 유형의 성능에 대해 설명합니다. 이러한 성능은 풀 레벨로 올라갑니다. 여러 VDEV에서 대용량 파일을 읽는 경우, 파일 읽기는 마지막(일반적으로 가장 느린) 드라이

모범 사례에서는 풀에 동일한 스토리지 VDEV만 사용하는 것이 좋습니다. 풀에 미러링된 VDEV가 여러 개 있는 경우 풀에 RAID-Z3 장치를 추가하지 마세요. 혼합 스토리지 VDEV[

> ZFS는 유닉스의 전통에 따라 멍청한 일을 하는 것을 막지 않는데, 이는 영리한 일을 하는 것도 막는 것이기 때문입니다.

VDEV [illegible] (Removing VDEVs)

[illegible] VDEV[illegible]. [illegible] VDEV[illegible]. [illegible] VDEV[illegible], VDEV [illegible]. OpenZFS[illegible]. RAID-Z [illegible]. [illegible], [illegible].

[illegible].

[illegible] (Pools Alignment and Disk Sector Size)

ZFS[illegible]. [illegible] ZFS[illegible]

[illegible] ZFS [illegible].

[illegible] (Partition Alignment)

[illegible]. [illegible] 512[illegible], [illegible] 4096[illegible](4K) [illegible]

[illegible](MBR)[illegible]. GPT(GUI

[illegible] **gpart(8)**[illegible], [illegible]. [illegible]. [illegible]

[illegible] SSD[illegible] 128KB [illegible] 1MB [illegible].

[illegible] GPT [illegible] **-a 1m** [illegible]. gpart [illegible] -

ZFS [illegible] (ZFS Sector Size)

ZFS[illegible] 512[illegible]. [illegible] 512[illegible] 512[illegible]. 4K-[illegible]. [illegible]. [illegible]. 512[illegible] ZFS[illegible] 4K [illegible], [illegible], [illegible]. 1KB [illegible].

ZFS [illegible]. [illegible].

[illegible], [illegible] ZFS[illegible] 4K-[illegible]. [illegible] ZFS [illegible]

ZFS[illegible]. [illegible] 512[illegible] 51

4K-[illegible]. [illegible] 4K-[illegible] ZFS[illegible].

ashift[illegible]. ashift[illegible] 9[illegible] ZFS[illegible] 512[illegible] 2^9[illegible] 512[illegible] ashift[illegible] 12[illegible] 2^{12}[illegible] 4096[illegible] ZFS[illegible] 4096[illegible] Fr

> [illegible] '9[illegible] 2^9[illegible]?

FreeBSD 10.1 [illegible] Ashift (FreeBSD 10.1 and Newer Ashift)

/etc/sysctl.conf [illegible] **sysctl vfs.zfs.min_auto_ashift**[illegible] ashift[illegible].

```
$ sysctl vfs.zfs.min_auto_ashift=12
```

/etc/sysctl.conf

FreeBSD 10.1

sysctl ashift

FreeBSD Ashift (Older FreeBSD Ashift)

10.1 FreeBSD FreeBSD ashift sysctl ZFS

gnop(8) FreeBSD GEOM gnop **/dev/gpt/zfs0** gnop

```
$ gnop create -S 4096 /dev/gpt/zfs0
```

/dev/gpt/zfs0.nop VDEV ZFS VDEV

```
$ zpool create compost mirror gpt/zfs0.nop gpt/zfs1
```

gnop(8) **gnop(8)** ZFS

(Creating Pools and VDEVs)

zpool(8) **zpool(8)** VDEV 5 RAID-Z

ashift

(Sample Drives)

0 **gpart(8)** ZFS 6 1TB

```
$ gpart create -s gpt da0
$ gpart add -a 1m -s1g -l sw0 -t freebsd-swap da0
$ gpart add -a 1m -1 zfs0 -t freebsd-zfs da0
```

```
$ gpart show -l da0
=>     40  1953525088  da0  GPT  (932G)
       40       2008    -  free -  (1.0M)
     2048    2097152    1  sw0  (1.0G)
  2099200  1951424512    2  zfs0  (931G)
1953523712       1416    -  free -  (708K)
```

[illegible] GPT [illegible] ZFS [illegible] gpt/zfs0[illegible] gpt/zfs5[illegible]. [illegible]

[illegible] (Striped Pools)

[illegible] **zpool create** [illegible]. [illegible] 5[illegible]

```
$ sysctl vfs.zfs.min_auto_ashift=12
$ zpool create compost gpt/zfs0 gpt/zfs1 gpt/zfs2 gpt/zfs4 gpt/zfs4
```

[illegible] **zpool status**[illegible].

```
$ zpool status
  pool: compost
 state: ONLINE
  scan: none requested
config:

NAME       STATE READ WRITE CKSUM
compost    ONLINE   0     0     0
 gpt/zfs0  ONLINE   0     0     0
 gpt/zfs1  ONLINE   0     0     0
 gpt/zfs2  ONLINE   0     0     0
 gpt/zfs3  ONLINE   0     0     0
 gpt/zfs4  ONLINE   0     0     0
```

5[illegible]. [illegible] VDEV[illegible].

[illegible]. [illegible] VDEV[illegible], VDEV[illegible]. [illegible]

[illegible] (Mirrored Pools)

[illegible]. [illegible]. [illegible], [illegible]

[illegible] **zpool create** [illegible] **mirror** [illegible]. [illegible]. [illegible] ashift[illegible].

```
$ sysctl vfs.zfs.min_auto_ashift=12
$ zpool create reflect mirror gpt/zfs0 gpt/zfs1
```

zpool status[illegible].

```
$ zpool status
  pool: reflect
 state: ONLINE
```

```
  scan: none requested
config:

NAME        STATE READ WRITE CKSUM
reflect     ONLINE   0     0     0
 mirror-0   ONLINE   0     0     0
  gpt/zfs0  ONLINE   0     0     0
  gpt/zfs1  ONLINE   0     0     0

errors: No known data errors
```

zpool [illegible] mirror-0[illegible]. mirror-0 [illegible] **gpt/zfs0** [illegible] **gpt/zfs1** [illegible] VDEV [illegible].[illegible]

```
$ zpool create reflect mirror gpt/zfs0 gpt/zfs1 gpt/zfs2 gpt/zfs3
```

[illegible](FreeBSD Mastery: Advanced ZFS[illegible]).

RAID-Z Pools

[illegible], [illegible] **zpool create** [illegible] RAID-Z, [illegible] RAID-Z [illegible]. [illegible] zpool RAID-Z([illegible]

> [illegible] '[illegible]'[illegible].

```
$ sysctl vfs.zfs.min_auto_ashift=12
$ zpool create raidz1 gpt/zfs0 gpt/zfs1 gpt/zfs2
```

[illegible] raidz1-0[illegible] VDEV[illegible].

```
$ zpool status bucket
  pool: bucket
 state: ONLINE
  scan: none requested
config:

NAME        STATE READ WRITE CKSUM
bucket      ONLINE   0     0     0
 raidz1-0   ONLINE   0     0     0
  gpt/zfs0  ONLINE   0     0     0
  gpt/zfs1  ONLINE   0     0     0
  gpt/zfs2  ONLINE   0     0     0
```

□ □□ □□ □□□ □ □□□ □□□ □□□□ □□□□ □□□ □□□□□. □□ RAID-Z □□□□ □□ □ □□ □□□□ □□□□. □□□□ 6□□ □□□□ raidz3-Z3□□ □□□□□

```
$ zpool create bucket raidz3 gpt/zfs0 gpt/zfs1 gpt/zfs2 gpt/zfs3 gpt/zfs4 gpt/zfs5
```

□□□□□ □□□ □ □□□□, □□ □□□□ raidz3-0□□□ □ □□□ □□□□□.

```
$ zpool status
  pool: bucket
 state: ONLINE
  scan: none requested
config:

NAME        STATE  READ WRITE CKSUM
bucket      ONLINE    0     0     0
 raidz3-0   ONLINE    0     0     0
  gpt/zfs0  ONLINE    0     0     0
  gpt/zfs1  ONLINE    0     0     0
...
```

□□□ □□□ □□ □□ VDEV□ □□□□. □□□ □□ □□ VDEV□ □□□□ □□□ □□ □□□?

□□ VDEV □ (Multi-VDEV Pools)

□□ □□ VDEV□ *mirror, raidz, raidz2, raidz3* □□ **zpool(8)**□ □ VDEV□ □□□□□ □□□□□. □□□ □□ **zpool(8)**□□ VDEV□ □□□

□ □□ □□ □□□□□ □□ RAID-10 □□□ □□□□□□□ □□ □□□ □□ □□□□□□□ □□□ □□□□□□. □□□□ □□□ □□□ □□□.

```
$ sysctl vfs.zfs.min_auto_ashift=12
$ zpool create barrel mirror gpt/zfs0 gpt/zfs1 mirror gpt/zfs2 gpt/zfs3
```

□□ **zpool create barrel**□ **zpool(8)**□ barrel□□□ **mirror** □ □□ □□□□"□□□ **gpt/zfs0 gpt/zfs1** □□□□□. **mirror** □□□□□
VDEV□ □□□□□ □ **zpool(8)**□□□□□. □ □□ VDEV□ **gpt/zfs2 gpt/zfs3** □□, □□. □ □□ □□□ □□□ □□□ □□□ □□□□.

```
$ zpool status barrel
  pool: barrel
 state: ONLINE
  scan: none requested
config:

NAME       STATE READ WRITE CKSUM
barrel     ONLINE   0     0     0
 mirror-0  ONLINE   0     0     0
```

```
  gpt/zfs0 ONLINE   0     0     0
  gpt/zfs1 ONLINE   0     0     0
 mirror-1  ONLINE   0     0     0
  gpt/zfs2 ONLINE   0     0     0
  gpt/zfs3 ONLINE   0     0     0
```

□□□ mirror-0□ mirror-1□□□ □ □□ VDEV□ □□□□. □ VDEV□□ □ □□ □□ □□□ □□□□□. ZFS□ □□ VDEV□ □□ □□□□ □□□□□□□□. RAID □□□□ □□□□ □□□ RAID□ □□□ □ □□□, □□□□ RAID □□□□□ □□□ □□□ □□ □ □□□□. □□□□ □ □□ RAID-Z1 VDEV□ □□ □□□□ □

```
$ zpool create vat raidz1 gpt/zfs0 gpt/zfs1 gpt/zfs2 raidz1 gpt/zfs3 gpt/zfs4 gpt/zfs5
```

□ □□ RAID-Z1 VDEV□ **gpt/zfs0, gpt/zfs1, gpt/zfs2**□ **gpt/zfs3, gpt/zfs4, gpt/zfs5**□ □□□□□. zpool□ □ RAID-Z □□□ □□□ □□ □□□□□.

```
$ zpool status vat
...
config:

NAME       STATE READ WRITE CKSUM
vat        ONLINE   0     0     0
 raidz1-0  ONLINE   0     0     0
  gpt/zfs0 ONLINE   0     0     0
  gpt/zfs1 ONLINE   0     0     0
  gpt/zfs2 ONLINE   0     0     0
 raidz1-1  ONLINE   0     0     0
  gpt/zfs3 ONLINE   0     0     0
  gpt/zfs4 ONLINE   0     0     0
  gpt/zfs5 ONLINE   0     0     0
```

□ VDEV□□ □□□ □□□□ □□□□.

□□□ RAIDZ□□ □□□ □□□, □□ □□ VDEV□ □□□□ □□□ □□□ □ RAIDZ □□ □□ □□□□□ □□□ □□□ □□ □ □□ □□□ □□□ □ □□□□. □□ □□□ □

□□ □□ □□ □□□ □□ □□ VDEV □□ □□□□□ □□ □□□□□. □□□ □□ □□□ □□□ VDEV□□ □□□ □ □□□, □□□□□□ □□ □□□ VDEV □□ □□□□□. □

□□ □□ □□ (Using Log Devices)

2□□□ □□□ □□, ZFS□ □□ □□ □□ □□ □□ □□ **zpool(8)** □□ **log**, **cache** □□ **log cache** □□□□ □□□□□□□□□□□□□□□. SSD□□□.

```
$ zpool create scratch gpt/zfs0 log gtp/zlog0 cache gpt/zcache1
```

□□ □□□ □□ □□□ □□□□□.

```
$ zpool status scratch
...
config:

NAME         STATE READ WRITE CKSUM
scratch      ONLINE   0     0     0
 gpt/zfs0    ONLINE   0     0     0
logs
 gpt/zlog0   ONLINE   0     0     0
cache
 gpt/zcache1 ONLINE   0     0     0
```

[illegible] gpt/zfs0 gpt/zfs3 [illegible] gpt/zlog0 [illegible].

```
$ zpool create db mirror gpt/zfs0 gpt/zfs1 mirror gpt/zfs2 gpt/zfs3 log mirror gpt/zlog0 gpt/zlog1
```

[illegible] SSD [illegible]

[illegible] VDEV (Mismatched VDEVs)

[illegible] VDEV [illegible] **zpool(8)** [illegible].

```
$ zpool create daftie raidz gpt/zfs0 gpt/zfs1 gpt/zfs2 mirror gpt/zfs3 gpt/zfs4 gpt/zfs5
invalid vdev specification
use '-f' to override the following errors:
mismatched replication level: both raidz and mirror vdevs are present
```

zpool(8) [illegible] **zpool create -f** [illegible] ZFS [illegible].

ZFS [illegible] **-f** [illegible] ZFS [illegible].

[illegible] (Reusing Providers)

[illegible].

```
$ zpool create db gpt/zfs1 gpt/zfs2 gpt/zfs3 gpt/zfs4
invalid vdev specification
use '-f' to override the following errors:
/dev/gpt/zfs3 is part of exported pool 'db'
```

[illegible] (5[illegible]). [illegible] ZFS [illegible].

[illegible] **-f** [illegible].

```
$ zpool create -f db gpt/zfs1 gpt/zfs2 gpt/zfs3 gpt/zfs4
```

ZFS [illegible] **-f** [illegible] create [illegible]

[illegible] (Pool Integrity)

ZFS[illegible] **fsck(8)**[illegible] **fsck(8)**[illegible] ZFS[illegible]

ZFS [illegible] (ZFS Integrity)

[illegible]

ZFS[illegible]

ZFS[illegible] **self-healing**[illegible] ZFS[illegible]

[illegible] VDEV[illegible] ZFS[illegible] RAID-Z[illegible]

[illegible] VDEV[illegible]

ZFS[illegible] **fsck(8)**[illegible] **fsck(8)** ZFS[illegible] ZFS[illegible]

[illegible]

Scrubbing ZFS

ZFS [illegible](scrub)[illegible]

[illegible] zpool[illegible]

```
...
scan: scrub repaired 0 in 15h57m with 0 errors on Sun Feb  8 15:57:55 2015
...
errors: No known data errors
...
```

[illegible]

```
$ zpool scrub zroot
```

[illegible] **zpool status** [illegible]

```
$ zpool status
...
```

```
scan: scrub in progress since Tue Feb 24 11:52:23 2015
12.8G scanned out of 17.3G at 23.0M/s, 0h3m to go
0 repaired, 74.08% done
...
```

스토리지를 스크러빙하는 ZFS 풀은 평소 zpool 실 scrub -s 옵션을 사용하세요. 진행 중인 스크럽을 취소해야 하는 경우,

```
$ zpool scrub -s zroot
```

돌아가서 가능한 한 빨리 시스템에서 스크럽을 완료하도록 하세요.

스크럽 빈도 (Scrub Frequency)

ZFS에 내장된 무결성 테스트와 복원력 덕분에 대부분의 오류는 복원력이 작동할 수 있을 만큼 조기에 발견된다면 수정할 수 있습니다. 즉, 하드웨어의 품질에 따라 호스트의 풀을 얼마나 자주 스크러빙

이 장의 뒷부분에 있는 "ZFS 유지 관리 자동화"에서 설명한 대로 FreeBSD는 정기적인 스크럽을 수행할 수 있습니다.

풀 속성 (Pool Properties)

ZFS는 속성을 사용하여 풀의 특성을 표현합니다. zpool 속성은 데이터 집합의 속성과 매우 유사하게 보이고 작동하며 많은 속성이 둘 사이에 겹치는 것처럼 보이지만, 데이터 집합 속성은 풀 속성

풀의 속성은 전체 풀에 영향을 줍니다. 풀의 일부에만 속성을 설정하려면 필요에 맞는 데이터 세트별 속성을 확인하세요.

풀 속성 보기 (Viewing Pool Properties)

시스템에 zpool 풀 get all 을 실행합니다. 특 zroot 풀에 관한 몇 가지 줄을 추가하면 됩니다. 여기서는

```
$ zpool get all zroot
NAME   PROPERTY  VALUE   SOURCE
zroot  size      920G    -
zroot  capacity  1%      -
zroot  altroot   -       default
zroot  health    ONLINE  -
...
```

처음 두 열은 풀 이름과 속성 이름을 나타냅니다.

세 번째 열에는 속성 값이 나열됩니다. 이 값은 사용 또는 사용 안 함, 켜짐 또는 꺼짐, 활성 또는 비활성 등과 같은 값일 수도 있고 값일 수도 있습니다. 이 풀의 크기 속성은 920G로, 이 풀에는

SOURCE 열에는 이 속성이 설정된 위치가 표시됩니다. 이 열은 대시 하나 또는 기본 또는 로컬이라는 단어로 표시될 수 있습니다. 대시는 이 속성이 자체적으로 설정된 것이 아니라 어떻게든 풀에

단일 속 zpool 려 get 을 실행합니다.

```
$ zpool get size
NAME   PROPERTY  VALUE  SOURCE
db     size      2.72T  -
zroot  size      920G   -
```

마지막에 풀 이름을 지정하여 범위를 좁히세요.

풀 속성 변경 (Changing Pool Properties)

이 책 전체에 **zpool set** 기능을 사용하여 **comment** 속성을 변경하고 설정했습니다. 풀의

```
$ zpool set comment="Main OS files" zroot
```

이제 이 코멘트가 속성 목록에 표시됩니다.

```
$ zpool get comment
NAME   PROPERTY  VALUE         SOURCE
db     comment   -             default
zroot  comment   Main OS files  local
```

여기서 SOURCE 열에 주목하세요. 기본적으로 풀에는 주석이 없습니다. 하지만 코멘트를 설정했으므로 소스가 로컬로 변경됩니다. 속성의 소스가 기본값에서 로컬로 변경되면 영원히 로컬로 유지됩

```
$ zpool set comment="-" zroot
# zpool get comment
NAME   PROPERTY  VALUE  SOURCE
db     comment   -      default
zroot  comment   -      local
```

주석을 기본값으로 로컬 설정하므로 값의 소스는 로컬로 유지됩니다.

풀을 **-C** 를 때 용하여 풀의 속 **C** 을 설정할 수 있습니다. 풀의 루트 데이터 세트에 대한 속성은

```
$ zpool create -o altroot=/mnt -O canmount=off -m none zroot /dev/gpt/disk0
```

이 **altroot** */mnt* 로 설정되어 **canmount** 속 **off** 로 설정되어 있습니다. 속성으로 데이터 기록 방식이 변경되면 속성 변경 후 기록된 데이터만 영향을 받습니다. ZFS는 속성 변

풀 히스토리 (Pool History)

모든 zpool은 풀이 생성된 시점부터 지금까지 풀에 적용된 모든 변경 사항의 사본을 보관합니다. 이 기록에는 시스템 전원 켜기 및 끄기와 같은 일상적인 이벤트는 포함되지 않지만, 속성 설정, 풀 업

기 **zpool history** 를 실행하고 풀 이름을 입력합니다.

```
$ zpool history zroot
History for 'zroot':
2014-01-07.04:12:05 zpool create -o altroot=/mnt -O canmount=off -m none zroot mirror /
dev/gpt/disk0.nop /dev/gpt/disk1.nop
2014-01-07.04:12:50 zfs set checksum=fletcher4 zroot
2014-01-07.04:13:00 zfs set atime=off zroot
...
```

[illegible] FreeBSD [illegible] FreeBSD [illegible] ZFS [illegible].

[illegible]:

```
...
2015-03-12.14:36:35 zpool set comment=Main OS files zroot
2015-03-12.14:43:45 zpool set comment=- zroot
```

comment [illegible]. [illegible].

[illegible], [illegible].

Zpool [illegible] (Zpool Maintenance Automation)

FreeBSD[illegible] **periodic(8)**[illegible] *periodic.conf*[illegible] **daily_status_zfs_enable**[illegible] ZFS [illegible].

```
daily_status_zfs_enable="YES"
```

[illegible] **periodic(8)** [illegible] "all pools are healthy" [illegible] **zpool status -x**[illegible].

[illegible] **daily_status_zfs_zpool_list yes** [illegible] *periodic.conf*[illegible] **daily_status_zpool** [illegible]

FreeBSD[illegible]. [illegible] *periodic.conf*[illegible] FreeBSD **daily_scrub_zfs_enable YES**[illegible]. 35

```
daily_scrub_zfs_enable="YES"
```

FreeBSD[illegible] **daily_scrub_zfs_pools**[illegible]. [illegible].

```
daily_scrub_zfs_pools="zroot prod test"
```

[illegible] **daily_scrub_zfs_default_threshold**[illegible].

```
daily_scrub_zfs_default_threshold="10"
```

[illegible] **daily_scrub_zfs_${poolname}_threshold**[illegible]. [illegible] 7[illegible].

```
daily_scrub_zfs_prod_threshold="7"
```

자체 임계값이 없는 풀은 기본 임계값을 사용합니다.

풀 제거 (Removing Pools)

풀을 제거하려면 **destroy** 명령과 풀 이름을 사용합니다.

```
$ zpool destroy test
```

파괴는 풀의 기본 공급자를 파괴된 풀의 일부로 표시하여 다른 풀에 재사용할 수 있도록 합니다. 디스크를 지우지는 않으며, 5장을 읽은 사람이라면 누구나 풀을 복원하고 데이터에 액세스할 수 있습니다.

공급업체의 데이터를 안전하게 삭제하거나 덮어써야 하는 경우 디스크 덮어쓰기 또는 파쇄 프로그램이 필요합니다.

Zpool 기능 플래그 (Zpool Feature Flags)

ZFS와 풀에는 원래 풀에서 지원하는 기능을 나타내는 버전 번호가 있습니다. 시스템에서 익숙하지 않은 스토리지 공급업체를 보고 "이 풀은 ZFS 버전 20이므로 중복 제거나 기본 암호화를 지원

그 후 오라클은 ZFS 소스를 폐쇄했고, 다양한 사람들이 마지막 오픈 소스 ZFS 릴리스를 가져와서 직접 유지 관리하도록 했습니다. 마지막 오픈 소스 오라클 ZFS 버전은 28이었습니다. 다양

OpenZFS 팀은 버전 번호로 기능을 추적하는 방식에서 벗어나는 것이 가장 좋은 방법이라고 판단했습니다. 이들은 OpenZFS 버전을 최대 5000개까지 늘려서 오라클이 새로운 버전을

FreeBSD를 포함하여 OpenZFS를 [redacted] 정식 [redacted] 예상될 때까지 [redacted] 해야 합니다. 최신 버전의 FreeBSD는 아마도 새로운 풀 기능을 지원할 [

기능을 사용하면 일반적으로 어떤 식으로든 온디스크 형식이 변경됩니다. 예를 들어, 스냅샷 지원을 추가한다는 것은 "이것은 스냅샷입니다."라는 새 필드와 메타데이터를 추가하는 것을 의미합니다. 해당 [

기능 플래그 보기 (Viewing Feature Flags)

풀에서 지원하는 기능 플래그 [redacted] 풀 속성을 찾습니다.

```
$ zpool get all zroot | grep feature
zroot  feature@async_destroy  enabled  local
zroot  feature@empty_bpobj    active   local
zroot  feature@lz4_compress   active   local
...
```

활성화된 풀 기능은 사용할 수 있지만 실제로는 아직 사용되지 않습니다. 시스템이 새로운 유형의 압축을 지원할 수 있지만 실제로 새 알고리즘을 사용하여 풀에 데이터를 쓰지 않은 경우입니다. 온디스크 [

비활성화된 풀 기능은 운영 체제에서 사용할 수 있지만 활성화되어 있지 않습니다. 풀에 이러한 기능을 사용할 수 있다고 표시되어 있지 않습니다. 비활성화된 기능이 있다는 것은 운영 체제에서 해당 기능

이 기능이 활성화되어 있으면 해당 기능이 사용 중이므로 디스크 형식이 변경된 것입니다. 대부분의 경우 이 기능을 지원하지 않는 시스템에서는 이 풀을 가져올 수 없습니다. 기능이 활성화되어 있지만 해

일부 기능은 "read-only compatible"입니다. 해당 기능을 사용 중인 경우, 해당 기능을 지원하지 않는 시스템으로 풀을 부분적으로 가져올 수 있습니다. 새 호스트는 풀에 있는 일부

풀을 만들면 해당 운영 [redacted] 모든 기능을 활성화합니다. 풀을 만들 때 풀의 모든 기능을 비활성화한 다음 기능을 보다 선택적으로 활성화할 수 있습니다.

[illegible].

[illegible] 4. ZFS [illegible] (ZFS Datasets)

[illegible] *home* [illegible] *usr* [illegible]
/usr/local/var [illegible] */var* [illegible].

ZFS [illegible]. [illegible] ZFS [illegible]

[illegible] */var* [illegible] */home* [illegible] **tunefs(8)** [illegible]. UFS [illegible]

[illegible]. ZFS [illegible].

[illegible] (Datasets)

[illegible]

ZFS [illegible] */usr* [illegible] */home* [illegible]? [illegible]. iSCSI [illegible]? [illegible].

[illegible]

[illegible] **zfs(8)** [illegible].

[illegible] (Dataset Types)

[illegible] ZFS [illegible] (filesystems), [illegible] (volumes), [illegible] (snapshots), [illegible] (clones), [illegible] (bookmarks) [illegible] 5 [illegible]

[illegible]. ZFS [illegible], setuid [illegible]

ZFS [illegible] **zvol** [illegible]. iSCSI [illegible] UFS [illegible]. ZFS [illegible]

[illegible]

[illegible]

[illegible]? (Why Do I Want Datasets?)

[illegible] */usr* */var* [illegible]. ZFS [illegible]

[illegible] ZFS [illegible]

[illegible]

[illegible] ZFS [illegible]. [illegible] ZFS [illegible].

[illegible] */webserver* [illegible] **sudo(8)** [illegible]

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 오

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 있습니

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 있

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 아니

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. ZFS의 놀라운 유연성과 효율성에 대한 한 가지 단점은 디스크 공간 사용량에 대한

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

마지막으로 마운트포인트는 데이터세트를 마운[illegible]zfs mount[illegible]줄사용하세요.데이터 세트가 마운트된다는 의미는 아니며, 단지 마운트될 경우 이 위치로 이동한다는 의미일 뿐입니다. 마운트된 모든 Z

데이[illegible]zfs[illegible]list[illegible]에[illegible]정 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.

데이터세트 생성, 이동 및 삭제하기 (Creating, Moving, and Destroying Dataset

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

파일시스템 만들기 (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 파

```
$ 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.

볼륨 만들기 (Creating Volumes)

-V 플래그와 zfs 크 create 볼 이터세트의 전체 경로를 입력합니다.

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

Zvols는 다른 데 -t volume 옵 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 /dev/zvol 트 제 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
```

newfs(8) [illegible]

[illegible] (Renaming Datasets)

[illegible] **zfs rename** [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

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

[illegible] **-f** [illegible].

> [illegible].

[illegible] (Moving Datasets)

[illegible] ZFS [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] *zroot/var/db* [illegible].

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

[illegible] **rename** [illegible] ZFS [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] 7[illegible].

[illegible] (Destroying Datasets)

[illegible] **zfs destroy** [illegible].

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

-r [illegible] **-R** [illegible].

[illegible] **-v** **-n** [illegible] **zfs(8)** [illegible].

ZFS [illegible] (ZFS Properties)

ZFS [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

속성 보기 (Viewing Properties)

zfs(8) 도구는 특정 속성 또는 **zfs get** 명령과 모든 속성을 검색할 수 있을 때 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. [

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.

데이터 **zfs** 이름 **get** 및 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 **name** 때 을 사용합니다. 여러 데이터세트의 여러 속성을 보려는 경우에 가장 적합합니다. 데이터세트의 이름을 표시하려면 특수 속성

```
$ 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
...
```

[illegible].

[illegible] (Changing Properties)

zfs set [illegible] **compression**, **off** [illegible].

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

zfs get[illegible].

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

[illegible] **compression** [illegible] ZFS[illegible]. [illegible] 6[illegible]. [illegible]

[illegible] (Read-Only Properties)

ZFS[illegible] **creation** [illegible]. "[illegible]." [illegible]

[illegible] (Filesystem Properties)

[illegible] **noexec** [illegible]. ZFS[illegible].

atime

[illegible] atime[illegible] **atime** [illegible]. ZFS[illegible] **atime** [illegible]. [illegible] on[illegible] atime. [illegible].

[illegible](off) [illegible].

atime[illegible]. [illegible]. [illegible], [illegible]

exec

exec [illegible] **on**[illegible]. [illegible] **exec** [illegible] **off** [illegible]. [illegible].

[illegible] **exec** [illegible] */bin/sh* [illegible] */bin/sh /home/mydir/script.sh*[illegible]. [illegible], [illegible].

readonly

[illegible] **readonly** [illegible] **on** [illegible] **off** [illegible].

setuid

[illegible] setuid [illegible] **passwd(1)**, **login(1)** [illegible] setuid [illegible] */home*[illegible] */tmp* [illegible] setuid [illegible].

> [illegible] setuid [illegible]. [illegible] setuid [illegible].

ZFS[illegible] **setuid** [illegible] setuid [illegible] **on** [illegible]. [illegible] **off** [illegible] setuid [illegible] setuid [illegible].

[illegible] (User-Defined Properties)

ZFS [illegible]? [illegible].

[illegible] **org.freebsd:swap**) [illegible]. [illegible] **org.illumos:swap** [illegible].

[illegible], Jude[illegible] **com.allanjude** [illegible] **backup_ignore** [illegible].

> [illegible] ZFS [illegible]. [illegible].

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

Jude[illegible]. true[illegible].

[illegible]/[illegible] (Parent/Child Relationships)

[illegible] **-r** [illegible] **zfs(8)** [illegible].

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

[illegible] *mypool/lamb* [illegible]. [illegible].

zfs inherit [illegible].

```
$ 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
```

[illegible] **compression** [illegible] **compression** [illegible].

[illegible].

```
$ 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 압축을 사용하도록 지시했습니다. 이는 자식에게까지 영향을 미쳤습니다.

상속 및 이름 변경 (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 데이터 집합은 압축을 사용합니다. *lamb*에서 *baz*로, 그리고 *second*의 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 압축됩

속성 제거 (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 재귀적으로 상속합니다. 내부에 설정하거나 사용자 정의 속성을 완전히 제거하려면 풀의 루트 데이터세트에서

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

[illegible].

ZFS [illegible] (Mounting ZFS Filesystems)

[illegible] */etc/fstab* [illegible] CD-ROM [illegible]. ZFS[illegible]

[illegible] ZFS [illegible] **mountpoint** [illegible] **mountpoint** [illegible] **mountpoint** [illegible].

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

[illegible] */usr/home* [illegible].

[illegible] FreeBSD *zroot* [illegible] *zroot* [illegible] */usr* [illegible] *zroot* [illegible].

[illegible] *db* [illegible] */db* [illegible].

[illegible] **mountpoint** [illegible] *legacy* [illegible] *none* [illegible].

[illegible] **canmount** [illegible] ZFS [illegible] **canmount yes** [illegible] **-a** [illegible] **zfs mount -a** [illegible] */etc/rc.conf* [illegible] ZFS[illegible] FreeBSD[illegible] **zfs mount -a**[illegible].

canmount [illegible] **noauto**[illegible] **zfs mount -a** [illegible] **zfs unmount -a** [illegible]

canmount[illegible] **off**[illegible].

[illegible] **canmount** [illegible].

canmount [illegible].

[illegible] (Datasets without Mount Points)

ZFS [illegible]. FreeBSD 10.1 [illegible]

```
$ 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 [illegible] */usr* [illegible]. [illegible]?

zfs list[illegible] */usr* [illegible] *zroot/usr* [illegible] **canmount** [illegible].

```
$ 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[illegible] *off*[illegible] *zroot/usr* [illegible] *usr/bin* [illegible] */usr/local*[illegible] */usr*[illegible] *usr/src* [illegible], [illegible]

[illegible] */var* [illegible].

[illegible] (Multiple Datasets with the Same Mount Point)

canmount[illegible] *off*[illegible] **canmount** *off* [illegible].

FreeBSD[illegible] zroot[illegible]. [illegible].

[illegible] **mountpoint** [illegible] *zroot* [illegible] */* [illegible], **canmount** [illegible] *off*[illegible] **mountpoint** [illegible].

[illegible] */opt* [illegible]. [illegible]. [illegible]. [illegible]

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

[illegible] */opt* [illegible].

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

[illegible].

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

db/data [illegible] **exec** [illegible] **setuid** [illegible]. [illegible].

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

[illegible] *db/programs* [illegible]. [illegible] *db/data* [illegible], [illegible].

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

이제 /opt에 바이너리용 2개와 데이터용 2개, 총 4개의 데이터세트가 마운트되어 있습니다. 사용자가 알다시피 이 디렉터리는 정상적인 디렉터리입니다. 하지만 파일 권한이 어떻게 되어 있든 이 디렉터

마운트 포인트가 없는 풀 (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.

수동으로 파일 시스템 마운트 및 마운트 해제하기 (Manually Mounting and Unmounting Filesyster

파일시스템을 수동으로 마운트하거나 마운트 해제하려면 zfs mount 명령을 사용합니다. 기본값 또는 mountpoint으로 설정된 파일시스템에 가장 일반적으로 사용됩니다.

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

파일시스템과 그 하위 파일시스템을 마운트하려면

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

데이터 세트를 마운트할 때 사용할 마운트 지점을 지정합니다. 이 마운트 지점은 데이터 세트를 마운트 해제할 때까지만 지속됩니다.

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

데이터세트에 mountpoint가 정의되어 있는 경우에만 데이터세트를 마운트할 수 있습니다. 데이터세트에 마운트 지점이 없는데 임시 마운트 지점을 정의하면 오류가 발생합니다.

ZFS 와 /etc/fstab (ZFS and /etc/fstab)

원래는 /etc/fstab으로 ZFS 파일시스템 마운트의 일부 또는 전부를 제어하도록 구현할 수 있습니다. 그러면 부팅 시점에 마운트 해제됩니다.

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

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

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

[illegible] */etc/fstab* [illegible] ZFS [illegible] *zfs* [illegible] *noatime, noexec, readonly, ro, nosuid* [illegible] *atime, exec, rw, suid* [illegible] ZFS [illegible]). [illegible] */tmp* [illegible] *scratch/junk* [illegible] nosuid [illegible] */etc/fstab* [illegible].

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

[illegible] */etc/fstab* [illegible] ZFS [illegible].

ZFS [illegible] (Tweaking ZFS Volumes)

Zvol [illegible].

[illegible] (Space Reservations)

zvol [illegible] **volsize** [illegible]. **volsize** [illegible] **volblocksize** [illegible]. (6 [illegible] 0 [illegible]

[illegible], [illegible]

Zvol [illegible] *(thin provisioning)* [illegible] *(sparse volumes)* [illegible].

[illegible] '[illegible]' [illegible].

zfs create -V -s [illegible].

Zvol [illegible] (Zvol Mode)

FreeBSD [illegible] **geom(4)** [illegible] zvol [illegible] **volmode** [illegible].

[illegible] **volmode** [illegible] *dev* [illegible] */dev* [illegible]. [illegible], RAID [illegible] GEO

volmode [illegible] *none* [illegible] ZFS [illegible], [illegible].

volmode [illegible] *default* [illegible] **sysctl vfs.zfs.vol.mode** [illegible]. [illegible] zvol [illegible]. [illegible] 1 [illegible] geor

[illegible] **zfs rename** [illegible] zvol [illegible].

[illegible] (Dataset Integrity)

ZFS [illegible] VDEV [illegible]. [illegible]. RAID-Z [illegible]

[illegible] (Checksums)

ZFS [illegible]. *on, fletcher2, fletcher4, sha256, off, noparity* [illegible] ZFS [illegible].

on [illegible] OpenZFS [illegible] *fletcher4* [illegible] 2015 [illegible].

[illegible] *fletcher4* [illegible] *fletcher4* [illegible] *fletcher4* [illegible] *on* [illegible] ZFS [illegible].

off [illegible].

noparity [illegible]. [illegible] RAID-Z [illegible]

[illegible] *fletcher2* [illegible] *sha256* [illegible] *fletcher4* [illegible] ZFS [illegible].

[illegible] (deduplication) [illegible] *sha256* [illegible].

[illegible] (Copies)

ZFS [illegible] **copies** [illegible] ZFS [illegible] **copies** [illegible] 3 [illegible]

[illegible] 2 [illegible] **copies** [illegible] 3 [illegible], **dd(1)** [illegible] 6 [illegible].

[illegible] **copies** [illegible] 1 [illegible] 2 [illegible]. 10MB [illegible]:

```
$ 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
```

[illegible]. [illegible] ZFS [illegible].

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

[illegible] 10MB [illegible]. [illegible] 2 [illegible]

```
$ 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)
```

[illegible].

```
$ zfs list mypool/lamb
NAME         USED  AVAIL  REFER  MOUNTPOINT
mypool/lamb  30.2M  13.7G  30.1M  /lamb
```

[illegible] 30MB [illegible] **ls(1)** [illegible] 10MB [illegible] 2 [illegible] 20 [illegible].

```
$ 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.

메타데이터 중복성 (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: 미

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

redundant_metadata=*all*(기본값)로 설정하면 ZFS는 모든 메타데이터의 추가 복사본을 저장합니다. 하나의 디스크 블록이 손상되면 최악의 경우 사용자 데이터의 단일 블록

redundant_metadata=*most*로 설정하면 ZFS는 대부분의 메타데이터 유형에 대해서만 추가 복사본을 저장합니다. 이렇게 하면 쓰기해야 하는 메타데이터가 줄어들기 때문

redundant_metadata=*most*로 **copies**를 3으로 설정하고 데이터 세트가 미러 풀에 있는 경우, ZFS는 대부분의 메타데이터 사본 6개와 데이터 및 일부 메타데

이 속성은 데이터베이스와 같이 메타데이터를 자주 업데이트하는 특정 사용 사례를 위해 설계되었습니다. 데이터가 이미 충분히 강력한 내결함성으로 보호되고 있다면 데이터베이스가 변경될 때마다 작성해야

이제 데이터 집합에 대해 이해했으니 풀 유지 관리에 대해 알아보겠습니다.