

FreeBSD Mastery: ZFS

Michael W. Lucas, Allan Jude, FreeBSD Mastery: ZFS, ,
FreeBSD, , , , , , , , .
, , , , , , .

- 0 : 0
- 1: ZFS 0
- 2 : 0 0 (Virtual Devices)
- 3. 0 (Pools)
- 4. ZFS 0000 (ZFS Datasets)

?? 0 : ??

[illegible]

通常，在 Linux 中，文件系统（如 extfs、UFS2、NTFS）的挂载点通常位于 `/mnt` 目录下。例如，要将一个 UFS2 文件系统挂载到 `/mnt/ufs2`，可以使用以下命令：

ZFS? ?????

[illegible]

1. 在 30 秒内，将 100 个文件写入 100 个文件，每个文件 100 字节。ZFS 的吞吐量应该比 ext4 高。ZFS 的吞吐量应该比 ext4 高。ZFS 的吞吐量应该比 ext4 高。

1980년대 초, IBM은 "FAT"이라는 파일 시스템을 도입했다. 이 시스템은 5, 10, 20MB의 파티션을 지원하며, 파일의 위치를 추적하는 데 사용되었다. 이 시스템은 이후의 파일 시스템에 큰 영향을 미쳤다.

[illegible]

但是 ZFS 的 设计 非常 复杂 . 因此 在 使用 之前 必须 了解 其 原理 . 本文 将 介绍 ZFS 的 设计 原理 , 并 讨论 其 在 不同 平台 上的 实现 . ZFS 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . ZFS 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . FreeBSD 的 实现 : ZFS 的 实现 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 .

ZFS ??

本文 将 介绍 ZFS 的 设计 原理 . Sun 的 ZFS 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . Sun 的 ZFS 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . Sun 的 ZFS 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 .

本文 将 介绍 ZFS 的 设计 原理 . MySQL 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . ZFS 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 .

ZFS 的 设计 原理 非常 复杂 , CDDL 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . ZFS 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 .

OpenZFS 的 设计 原理 非常 复杂 (http://open-zfs.org) 的 设计 原理 非常 复杂 . OpenZFS 的 设计 原理 非常 复杂 , Linux, OS X, Illumos, FreeBSD 的 设计 原理 非常 复杂 . ZFS 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 .

FreeBSD 的 设计 原理 非常 复杂 , ZFS 的 设计 原理 非常 复杂 . CDDL 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . ZFS 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . FreeBSD 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 .

?? ?? ??

ZFS 的 设计 原理 非常 复杂 , FreeBSD 的 设计 原理 非常 复杂 . GEOM 的 设计 原理 非常 复杂 , FreeBSD 的 设计 原理 非常 复杂 . "ZFS 的 设计 原理 非常 复杂" 的 设计 原理 非常 复杂 . Absolute FreeBSD(No Starch Press, 2007) 的 设计 原理 非常 复杂 . FreeBSD 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 .

FreeBSD 的 设计 原理 非常 复杂 , OpenZFS 的 设计 原理 非常 复杂 . ZFS 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . FreeBSD 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 .

ZFS 的 设计 原理 非常 复杂 . ZFS 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 . ZFS 的 设计 原理 非常 复杂 , 本文 将 介绍 其 在 不同 平台 上的 实现 .

[[[[[[] [[] [[. FreeBSD ZFS [[GEM [[[[[[, [[
[[[[[[[[[[. RAID [[ZFS [[[[[[[[.

RAID 0 0000 00 00 0000 0000 00 0 00 00 . 0 ZFS 0 000 0 00
00 00 00 00 . 0 00 0000 00 00 0000 00 ZFS 00000000
00 ! 00 00 ZFS 00 00 0 00 00 0000 . 00 00 00 ZFS 00 0
00 0 00 00 0 00 ZFS 00 00 00 . ZFS 0000 , 00 0000 00
0000 00 0 00 00 00 00 00 0000 0000 .

ZFS? ???? ??????

ZFS                        .

我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。 Linux
 KVM 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 ， 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。
 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。
 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。
 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。
 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。 我们 使用 了 文件系统 ZFS 的 特性 来 实现 数据 的 冗余 和 容错 。

ZFS 11 11111111 11111111 . 111 1 11111111 RAM 111 111 . 11111111
111 11 1111 1111 UFS2 11 11 11111111 1 11111111 .

1. 检查 /etc/fstab 文件，确认是否配置了正确的文件系统类型和挂载选项。

ZFS ????

00 0000 ZFS 0000 0000 0000 . 000 0000 0000 0000 . 000 .
 00 0000 00 0000 00 000000 ZFS 0 0000 . 0000 00 0 00
 0000 ZFS 0000 00 ZFS 0000 000 FreeBSD 0000 00 00 000 0000
 0000 .

RAM

Sun 0000 ZFS 0 000000 ECC RAM 000000 00 00 00 00 0000 . 00
0000 00 0000 000000 . 000 00 0000 0000 , "ECC 00 0000 ZFS ECC
00 0000 00 00 00000 0 0000 00 " 000 . ZFS 000 00 00 000 0000
00 000 000 000000 00 000 00000 0000 .

[illegible]

RAID ????

RAID 0 0000 0000 0000 0000 . 0000 . 0000 RAID 0 000 ZFS 0 000 RAID 0 000 ZFS 0 000 000 000 000000 . 000 000000 RAID 0 00 HBA(000 00 000) 000000 .

RAID 0 0000 RAID 000 . 0000 RAID 0000 000 00 00 000 0000 RAID 000 0000 , 0 0000 00 0000 000000 0000 . 00 0000 000 00 000 00 000 00 000 000 0000 00 0000 000000 . Windows 3.1 00 OS 000 000000 RAID 0 0000 00 00 300 000 00 0000 RAID 0 0 000 000000 000 00 00 0000 .

00 000 000000 . 000000 000 0 0 000000 . 000 000000 .

ZFS 000000 00 0000 0 000 0000000 . 000000 000 000000 00 0000 000000 . 000 000 0000 000 000 0000 000 0000 000 0000 . 000 RAID 000 00 0000 0 00 000000 00 000 000 ZFS 000 000 0 00 000 000000 . 0000 RAID 0 00 000 0000 .

ZFS 0000 000000 00 RAID 0000 000000 000 00 000 , 00 00 0000 0000 000 ZFS 0 00 000000 . 0000 RAID 0 0000 0000 000 000 000 00 000 .

000 00 RAID 000000 000 RAID 0 00 0000 000 0 0000 . 000000 "0000 00 0 ", 0 JBOD 0 000000 000 000000 000 0 000000 00 000 RAID-0 00 000000 . 000 00 00 0000 00 00 0 00 000 000 00 00 000 000000 . 0 00 00 000 000000 0000 0000 0 0000 00000000 00 0 000 , 000 0 000 RAID 00000000 00 0 000 0000 ! ZFS 000 RAID 000000 0000 0 000 000 000 RAID 000 00 000 0000 00 000000 00 0000 000 00 0000 00 0 0000 . 000 00 ZFS 0 00 0 0000 000000 .

00 0000 RAID 000 000000 JBOD 000000 000 0 0000 . 00 0000 RAID 000 0000 0 0000 . 0 00 00 0000 RAID 0 00 0000 00 0000 0 000 000000 .

000 00 00 000000 000 0000 RAID 0 000 000 000 00 000 00 0000 000000 .

RAID 000000 0 0000 RAID 0 00 0000 000 0000 000 000000 . 000000 "00 00 " 000 00000000 . 000 000 000000 00 000 00 00 0000 000 0 0000 . 000 00 , 00 00 , 00 000 000000 . 00 0 00 000 0000 0 "00 0000 "00 000 0000 0000 000000 0000 0 000 0 00 00 000000 0000 000000 .

000 00 0000 RAID 0000 000 000 00 0000 00 000000 . 000000 .

SATA ? SAS ? SSD

000 00 000 000 0000 000000 . ZFS 0 SAS 0000 , SATA 0000 , 000 000 , SSD 00 00 00 0000 0000 00 00 00 000 000 0 0000 .

0000 0000 00 00 0 00 00 0000 00000 . 0000 0000 0000 00 00
 000 000 , 000 0000 00 000 0000 . 000 00 000 000 000 000 0 000
 00000 . 000 00 0000 "00 00 0 00 00 000 00 000 00 00 "0 00 000
 000 0000 . 0000 000 'f' , 0000 000 'b' 00000 0000 000 00 000 0
 0000 .

[illegible][illegible]

在 2024 年 10 月 15 日，我们发布了关于 GPT 模型在 FreeBSD 系统上的性能测试报告。该报告详细分析了 GPT 模型在 FreeBSD 系统上的性能表现，并提供了相关的测试数据和结论。

`WD-WCAW36477223 /dev/gpt/s2d9-AW36477223`

1. 在 `FreeBSD` 4.x 版本中，`gptid` 是一个系统变量，用于标识磁盘分区。它由 `gptid` 和 `partition` 两部分组成。例如，`gptid=1` 表示第一个分区。

□□ □□□ □□□ □□ □□□□ □□ □□□ □□ □□□□□ . □□ □□ □□□□□ .































REFER 00 ZFS00 00 000 0000 .000 000 0000 0000 0 00 0000 000 ,
000 000 00 000 000 0000 .0000 00 00 ZFS000 00 00 0000 00000
zroot 000 429MB0 "00 "000 96KB0 0000 00000 .0 0000 13GB0 00 000
000 ,0 00 000 000 00 96KB0 0000 0 0000 .000 00 00 0000 .000
000 0 000 000 0000 00 00000 .60000 ZFS 000 000 00 000 00000
000 000 0000 0 0 0000 0 0 000 000 ,00 0 00 000 000 00000 .

```

|_|_|_|_|      |_|   |_|_|   |_|_|   |_|_|   |_|_|   . zroot ZFS | |_|_|_|_|      |_|_|_|_|      .

```

[illegible]

```

root@ROOT/default# ls -ld /dev/dmz
drwxr-xr-x 1 root root 4096 Jan 10 12:00 /dev/dmz
root@ROOT/default# ls -ld /dev/dmz
drwxr-xr-x 1 root root 4096 Jan 10 12:00 /dev/dmz
root@ROOT/default#

```

[illegible]

`zroot/tmp`

ZFS ??? ? ?? (ZFS partitions and properties)

[illegible][illegible]

ZFS       .  ,    

我们 **/var/log** 我们 在 我们 我们 我们 我们 我们 我们 我们 我们 我们 ,
我们 ZFS 我们 我们 我们 我们 我们 .

我们 我们 我们 在 我们 我们 ZFS 我们 在 我们 . 我们 我们 , **quota** 我们 我们 我们
我们 我们 在 我们 我们 在 ZFS 我们 我们 我们 我们 我们 我们 . ZFS 我们
我们 **zfs(8)** 我们 .

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

zfs get 我们 我们 我们 .

```
$ zfs get quota zroot/var/log
```

NAME	PROPERTY	VALUE	SOURCE
zroot/var/log	quota	2G	local

zfs get all 我们 ZFS 我们 我们 我们 我们 我们 在 我们 .
4我们 ZFS 我们 我们 我们 , 6我们 我们 我们 我们 我们 我们 .

ZFS ??(ZFS Limits)

我们我们 我们 我们 我们 我们 我们 . 我们 我们 我们 我们 FAT 我们 我们 我们
32MB 我们 我们 我们 我们 我们 , 我们 2GB, 4GB 我们 . 我们 FAT32
2TB 我们 我们 我们 我们 . UFS 我们 ext2/3/4fs 我们 我们 我们 我们
. 我们 我们 我们 我们 我们 我们 我们 , 我们 在 我们 我们 我们
我们 我们 我们 我们 我们 . 我们 我们 我们 我们 我们 我们 我们
我们 我们 我们 我们 我们 我们 我们 我们 我们 .

ZFS 我们 ZFS 我们 我们 我们 我们 我们 我们 , 我们 我们 我们 . ZFS
在 我们 我们 我们 128我们 我们 我们 我们 我们 我们 我们
我们 我们 在 我们 我们 我们 . 我们 我们 我们 16我们 2^{48}
我们 我们 在 我们 . 我们 我们 256我们 , 我们 2^{78} 我们 我们 . 我们 我们
我们 2^{64} 我们 我们 我们 在 我们 , 我们 我们 我们 2^{64} 我们 我们 我们 在 我们 .

我们 我们 我们 我们 我们 我们 我们 我们 我们 我们 我们 . 我们 我们 我们 我们
在 我们 我们 我们 我们 我们 我们 我们 . 我们 ZFS 我们 我们
我们 在 我们 我们 我们 我们 我们 我们 . 我们 我们 我们
FAT/UFS/extfs 我们 在 我们 我们 .

???? ?

ZFS 我们 我们 我们 我们 我们 . 我们 我们 我们 我们 我们 我们
我们 , 我们 我们 在 我们 我们 在 我们 我们 我们 我们 我们 我们 .
zpool(8) 我们 我们 我们 我们 我们 我们 在 我们 . 我们 我们 FreeBSD 我们
我们 .

\$ 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

在 ZFS 中，数据块被存储在池中的设备。ZFS 池由一个或多个设备组成，这些设备可以是物理磁盘、虚拟设备（VDEV）或 RAID。ZFS 池的状态可以是 ONLINE、DEGRADED 或 OFFLINE。如果池中的任何设备出现故障，池的状态将变为 DEGRADED。如果池中的所有设备都出现故障，池的状态将变为 OFFLINE。

ZFS 池的扫描（scan）用于检查数据块的完整性。如果池中的任何设备出现故障，ZFS 将自动启动扫描。扫描完成后，ZFS 将报告任何数据损坏。如果扫描发现数据损坏，ZFS 将尝试修复数据。如果无法修复数据，ZFS 将报告数据损坏。

在 ZFS 中，数据块被存储在池中的设备。ZFS 池由一个或多个设备组成，这些设备可以是物理磁盘、虚拟设备（VDEV）或 RAID。ZFS 池的状态可以是 ONLINE、DEGRADED 或 OFFLINE。如果池中的任何设备出现故障，池的状态将变为 DEGRADED。如果池中的所有设备都出现故障，池的状态将变为 OFFLINE。

?? ?? (Virtual Devices)

在 ZFS 中，虚拟设备（VDEV）是由一个或多个物理设备组成的逻辑设备。VDEV 可以是 RAID、镜像或条带化。VDEV 的状态可以是 ONLINE、DEGRADED 或 OFFLINE。如果 VDEV 中的任何物理设备出现故障，VDEV 的状态将变为 DEGRADED。如果 VDEV 中的所有物理设备都出现故障，VDEV 的状态将变为 OFFLINE。

在 ZFS 中，RAID 是由两个或多个 VDEV 组成的逻辑设备。RAID 可以是 RAID-Z、RAID-1 或 RAID-10。RAID 的状态可以是 ONLINE、DEGRADED 或 OFFLINE。如果 RAID 中的任何 VDEV 出现故障，RAID 的状态将变为 DEGRADED。如果 RAID 中的所有 VDEV 都出现故障，RAID 的状态将变为 OFFLINE。

ZFS 池的扫描（scan）用于检查数据块的完整性。如果池中的任何设备出现故障，ZFS 将自动启动扫描。扫描完成后，ZFS 将报告任何数据损坏。如果扫描发现数据损坏，ZFS 将尝试修复数据。如果无法修复数据，ZFS 将报告数据损坏。

ZFS 池的扫描（scan）用于检查数据块的完整性。如果池中的任何设备出现故障，ZFS 将自动启动扫描。扫描完成后，ZFS 将报告任何数据损坏。如果扫描发现数据损坏，ZFS 将尝试修复数据。如果无法修复数据，ZFS 将报告数据损坏。

在 ZFS 中，zpool status 命令用于检查 ZFS 池的状态。zpool status 命令将输出池的名称、状态、扫描状态、配置和错误信息。如果池中的任何设备出现故障，zpool status 命令将报告错误信息。如果池中的所有设备都出现故障，zpool status 命令将报告池的状态为 OFFLINE。

??? ??? (Blocks and Inodes)

[illegible]

1. 在 ZFS 中，`zfs create` 命令用于创建新的文件系统。例如，`zfs create tank/myfs` 会在名为 `tank` 的池下创建一个名为 `myfs` 的文件系统。

2. 文件系统创建后，可以使用 `zfs mount` 命令将其挂载到指定的目录。例如，`zfs mount tank/myfs` 会将 `myfs` 挂载到 `/mnt/tank/myfs`。

3. 文件系统支持快照功能，可以通过 `zfs snapshot` 命令创建快照。例如，`zfs snapshot tank/myfs@now` 会创建名为 `now` 的快照。

4. 快照可以用于备份和恢复。要恢复文件系统，可以使用 `zfs rollback` 命令。例如，`zfs rollback tank/myfs@now` 会将文件系统恢复到 `now` 快照的状态。

5. ZFS 还支持数据加密，可以通过 `zfs encryption` 子命令进行配置。

[illegible][illegible]

?? 2 : ?? ?? (Virtual Devices)

在 虚拟机 中 创建 虚拟 磁盘 设备 时，... 我们 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备。 例如，我们可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，... ZFS 文件系统 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，... ZFS 文件系统 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，...

ZFS 文件系统 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，... ZFS 文件系统 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，...

??? ? ?? ?? ?? (Disks and Other Storage Media)

ZFS 文件系统 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，... FreeBSD GEOM 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，... ZFS 文件系统 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，...

?? ??? ??? (Raw Disk Storage)

在 虚拟机 中 创建 虚拟 磁盘 设备 时，... ZFS 文件系统 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，...

在 虚拟机 中 创建 虚拟 磁盘 设备 时，... FreeBSD 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，...

在 虚拟机 中 创建 虚拟 磁盘 设备 时，... 6TB 虚拟 磁盘 设备 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，... 512 字节 的 虚拟 磁盘 设备 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，... 4096 字节 (4K, 字节) 的 虚拟 磁盘 设备 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，... 8 字节 的 虚拟 磁盘 设备 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，...

??? ????

在 虚拟机 中 创建 虚拟 磁盘 设备 时，... ZFS 文件系统 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，... ZFS 文件系统 可以 使用 不同的 方法 来 创建 虚拟 磁盘 设备，...

本文將探討在 FreeBSD 系統中，如何透過 ZFS 實現數據的高可用性和容錯能力。

在 Solaris 中，ZFS 是作為一個獨立的分區系統實現的，而在 FreeBSD 中，ZFS 則是作為一個用戶空間的庫來實現的。Solaris 的 ZFS 實現是基於 UFS 的，而 FreeBSD 的 ZFS 實現則是基於 VFS 的。Solaris 的 ZFS 實現是基於 UFS 的，而 FreeBSD 的 ZFS 實現則是基於 VFS 的。FreeBSD 的 ZFS 實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。FreeBSD 的 ZFS 實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。

FreeBSD 的 ZFS 實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。FreeBSD 的 ZFS 實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。FreeBSD 的 ZFS 實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。FreeBSD 的 ZFS 實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。FreeBSD 的 ZFS 實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。

在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。

在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。

GEOM 設備存儲 (GEOM Device Storage)

ZFS 在 FreeBSD 中是通過 GEOM 設備存儲來實現的。GEOM 設備存儲是 FreeBSD 的一個子系統，它允許用戶將多個物理設備組合成一個邏輯設備。ZFS 可以將數據存儲在這些邏輯設備中，從而實現數據的高可用性和容錯能力。GEOM 設備存儲支持多種設備類型，包括硬盤、SSD 和 RAID。ZFS 可以將數據存儲在這些設備中，從而實現數據的高可用性和容錯能力。

在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。

在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。在 FreeBSD 中，ZFS 的實現是基於 VFS 的，而 Solaris 的 ZFS 實現則是基於 UFS 的。

GEOM 提供了 一个 抽象 的 磁盘 接口 。 disk ident, gptid, GPT 分区 表 的 GEOM-specific glabel 标识符 是 唯一 的 。 这 使得 磁盘 的 管理 更加 简单 。

GEOM 的 设计 旨在 提供 一个 统一 的 磁盘 接口 。 它 允许 用户 通过 一个 简单的 接口 来 访问 各种 类型的 磁盘 设备 。 例如 ， 用户 可以 通过 一个 简单的 命令 来 创建 一个 新的 磁盘 分区 ， 而 不需要 了解 底层 的 磁盘 格式 或 分区 表 的 细节 。 这 使得 磁盘 的 管理 更加 简单 。

此外 ， GEOM 还支持 各种 类型的 磁盘 设备 ， 包括 硬盘 驱动器 、 固态硬盘 (SSD) 、 网络 存储 设备 (如 iSCSI) 和 虚拟 磁盘 设备 (如 VMFS) 。 这 使得 GEOM 成为 一个 非常 灵活 的 磁盘 管理 工具 。

在 FreeBSD 中 ， GEOM 的 使用 通常 涉及 创建 一个 新的 磁盘 分区 或 对 现有 的 磁盘 分区 进行 管理 。 这 可以通过 使用 一些 简单的 命令 来完成 ， 如 `gpart` 和 `geom` 。

关于 GEOM 的 更多 信息 ， 请 参考 [FreeBSD Mastery: Advanced ZFS](#) 。

GEOM 的 设计 旨在 提供 一个 统一 的 磁盘 接口 。 它 允许 用户 通过 一个 简单的 接口 来 访问 各种 类型的 磁盘 设备 。 例如 ， 用户 可以 通过 一个 简单的 命令 来 创建 一个 新的 磁盘 分区 ， 而 不需要 了解 底层 的 磁盘 格式 或 分区 表 的 细节 。 这 使得 磁盘 的 管理 更加 简单 。

此外 ， GEOM 还支持 各种 类型的 磁盘 设备 ， 包括 硬盘 驱动器 、 固态硬盘 (SSD) 、 网络 存储 设备 (如 iSCSI) 和 虚拟 磁盘 设备 (如 VMFS) 。 这 使得 GEOM 成为 一个 非常 灵活 的 磁盘 管理 工具 。

在 FreeBSD 中 ， GEOM 的 使用 通常 涉及 创建 一个 新的 磁盘 分区 或 对 现有 的 磁盘 分区 进行 管理 。 这 可以通过 使用 一些 简单的 命令 来完成 ， 如 `gpart` 和 `geom` 。

关于 GEOM 的 更多 信息 ， 请 参考 [FreeBSD Mastery: Advanced ZFS](#) 。

?? ?? (File-Backed Storage)

在 某些 情况下 ， ZFS 可以 使用 文件 系统 来 存储 数据 。 这 通常 是 通过 使用 一个 文件 系统 来 存储 数据 的 块 来 实现 的 。 这 使得 ZFS 可以 使用 任何 支持 文件 系统 的 存储 设备 。

????? (Providers vs. Disks)

在 FreeBSD 中 ， GEOM 的 使用 通常 涉及 创建 一个 新的 磁盘 分区 或 对 现有 的 磁盘 分区 进行 管理 。 这 可以通过 使用 一些 简单的 命令 来完成 ， 如 `gpart` 和 `geom` 。

此外 ， GEOM 还支持 各种 类型的 磁盘 设备 ， 包括 硬盘 驱动器 、 固态硬盘 (SSD) 、 网络 存储 设备 (如 iSCSI) 和 虚拟 磁盘 设备 (如 VMFS) 。 这 使得 GEOM 成为 一个 非常 灵活 的 磁盘 管理 工具 。

在 FreeBSD 中 ， GEOM 的 使用 通常 涉及 创建 一个 新的 磁盘 分区 或 对 现有 的 磁盘 分区 进行 管理 。 这 可以通过 使用 一些 简单的 命令 来完成 ， 如 `gpart` 和 `geom` 。

关于 GEOM 的 更多 信息 ， 请 参考 [FreeBSD Mastery: Advanced ZFS](#) 。

FreeBSD 的 磁盘 管理 工具 包括 `gpart` 和 `geom` 。

在 某些 情况下 ， ZFS 可以 使用 文件 系统 来 存储 数据 。 这 通常 是 通过 使用 一个 文件 系统 来 存储 数据 的 块 来 实现 的 。 这 使得 ZFS 可以 使用 任何 支持 文件 系统 的 存储 设备 。

VDEV: ?? ?? (VDEVs: Virtual Devices)

在 FreeBSD 中 ， VDEV 的 使用 通常 涉及 创建 一个 新的 虚拟 设备 或 对 现有 的 虚拟 设备 进行 管理 。 这 可以通过 使用 一些 简单的 命令 来完成 ， 如 `zfs` 和 `vdev` 。

此外 ， VDEV 还支持 各种 类型的 虚拟 设备 ， 包括 硬盘 驱动器 、 固态硬盘 (SSD) 、 网络 存储 设备 (如 iSCSI) 和 虚拟 磁盘 设备 (如 VMFS) 。 这 使得 VDEV 成为 一个 非常 灵活 的 虚拟 设备 管理 工具 。

在 FreeBSD 中 ， VDEV 的 使用 通常 涉及 创建 一个 新的 虚拟 设备 或 对 现有 的 虚拟 设备 进行 管理 。 这 可以通过 使用 一些 简单的 命令 来完成 ， 如 `zfs` 和 `vdev` 。

关于 VDEV 的 更多 信息 ， 请 参考 [FreeBSD Mastery: Advanced ZFS](#) 。

RAID ☐ ☐ ☐ ☐ .

VDEV 3 4 . ZFS

[illegible]

VDEV을 RAID로 구성하여 ZFS을 구성하는 방법

RAID-6을 구성하고, RAID-Z2 VDEV을 구성하여 ZFS을 RAID-60으로 구성하는 방법

RAID-1과 RAID-10을 구성하고, ZFS을 구성하여 VDEV을 구성하는 방법

RAID-Z2를 구성하고, VDEV을 구성하는 방법

VDEV ??? (VDEVs Redundancy)

[illegible]

Stripe (1? ???)

\square $\square\square\square$ $\square\square$ VDEV $\square\square\square\square\square$ \square , $\square\square\square$ $\square\square\square$. $\square\square$ \square $\square\square$ \square $\square\square\square$

$\square\square\square$ $\square\square\square$ \square $\square\square\square$ $\square\square\square\square$. $\square\square\square\square$ $\square\square$ \square $\square\square$ VDEV $\square\square\square\square$.

ZFS 11 11 11 VDEV 11 1111 11111111 1111 1111 11 VDEV 111111 .
 1111 111111 111111 1111 1111 11 11 1111 111111 . 11 1111 11 1111
 111111 . 111111 111111 11 1 1111 111111 111111 111111 111111 111111 VDEV
 1111 111111 .

Mirrors (2? ??? ???)

[illegible]

RAID-Z1(3? ??? ???)

$2n+3(5, 7, 9 \dots)$ 的 奇数 个 数据 副本 . 每 个 数据 副本 2 个 奇数 的 数据 副本 的 奇数 个 数据 副本 . 数据 副本 的 奇数 个 数据 副本 . 数据 副本 的 奇数 个 数据 副本 . 数据 副本 的 奇数 个 数据 副本 .

VDEV ?? (Repairing VDEVs)

当 VDEV 的 数据 副本 的 奇数 个 数据 副本 , 当 数据 副本 的 VDEV 的 "degraded(数据 副本)" 数据 副本 . 数据 副本 的 VDEV 的 数据 副本 的 奇数 个 数据 副本 . 5 数据 副本 的 数据 副本 的 数据 副本 .

数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 . RAID-Z 的 数据 副本 , 数据 副本 的 数据 副本 的 数据 副本 .

ZFS 的 RAID 的 数据 副本 的 数据 副本 的 ZFS 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 ZFS 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 RAID 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . RAID 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . RAID 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 ZFS RAID-Z 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 5 数据 副本 的 ZFS 的 数据 副本 的 数据 副本 .

RAID-Z? ?? RAID ??(RAID-Z versus Traditional RAID)

RAID-Z 的 数据 副本 的 RAID 的 数据 副本 的 数据 副本 的 数据 副本 , 数据 副本 的 ZFS 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 .

数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 100MB 数据 副本 的 数据 副本 的 数据 副本 ! 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 RAID 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 RAID 的 数据 副本 的 数据 副本 的 数据 副本 .

数据 副本 的 数据 副本 的 数据 副本 的 ZFS 的 数据 副本 的 数据 副本 , 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 ZFS 的 数据 副本 的 数据 副本 (3) 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . ZFS 的 数据 副本 的 数据 副本 的 数据 副本 , 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 .

数据 副本 的 RAID 的 2 数据 副本 的 数据 副本 的 数据 副本 "数据 副本 (write hole)" 数据 副本 的 数据 副本 . RAID 5 的 6 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 .

数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . RAID 的 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 . 数据 副本 的 数据 副本 的 数据 副本 的 数据 副本 .

寫入 讀取 寫入 讀取 .

ZFS 的 寫 入 是 採用 寫 入 緩 存 (Copy-on-write, 7) 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

?? VDEV (Special VDEVs)

寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

??? ??? ??(Seperate Intent Log; SLOG, ZIL)

ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

ZIL 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

??(Cache: L2ARC)

寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . 寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

寫 入 緩 存 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . RAM 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 . ZFS 的 寫 入 是 採用 寫 入 緩 存 的 方式 來 處理 的 .

2 ARC L2ARC

RAM ZFS L2ARC SSD NVMe SSD ZFS

VDEV? ??? ?? (How VDEVs Affect Performance)

VDEV

IOPS

12 1 (12 x 1TB) 6 2 (6 x 2TB) ZFS IOPS

RAID-Z

VDEV

250 IOPS 100MB/s

One Disk

1. Single Disk Virtual Device Configurations
 2. ZFS VDEV
 3.
 4.
 5.
 6.
 7.
 8.
 9.
 10.
 11.
 12.
 13.
 14.
 15.
 16.
 17.
 18.
 19.
 20.
 21.
 22.
 23.
 24.
 25.
 26.
 27.
 28.
 29.
 30.
 31.
 32.
 33.
 34.
 35.
 36.
 37.
 38.
 39.
 40.
 41.
 42.
 43.
 44.
 45.
 46.
 47.
 48.
 49.
 50.
 51.
 52.
 53.
 54.
 55.
 56.
 57.
 58.
 59.
 60.
 61.
 62.
 63.
 64.
 65.
 66.
 67.
 68.
 69.
 70.
 71.
 72.
 73.
 74.
 75.
 76.
 77.
 78.
 79.
 80.
 81.
 82.
 83.
 84.
 85.
 86.
 87.
 88.
 89.
 90.
 91.
 92.
 93.
 94.
 95.
 96.
 97.
 98.
 99.
 100.

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

1. Single Disk Virtual Device Configurations
 2. ZFS VDEV
 3.
 4.
 5.
 6.
 7.
 8.
 9.
 10.
 11.
 12.
 13.
 14.
 15.
 16.
 17.
 18.
 19.
 20.
 21.
 22.
 23.
 24.
 25.
 26.
 27.
 28.
 29.
 30.
 31.
 32.
 33.
 34.
 35.
 36.
 37.
 38.
 39.
 40.
 41.
 42.
 43.
 44.
 45.
 46.
 47.
 48.
 49.
 50.
 51.
 52.
 53.
 54.
 55.
 56.
 57.
 58.
 59.
 60.
 61.
 62.
 63.
 64.
 65.
 66.
 67.
 68.
 69.
 70.
 71.
 72.
 73.
 74.
 75.
 76.
 77.
 78.
 79.
 80.
 81.
 82.
 83.
 84.
 85.
 86.
 87.
 88.
 89.
 90.
 91.
 92.
 93.
 94.
 95.
 96.
 97.
 98.
 99.
 100.

Two Disks

1. Single Disk Virtual Device Configurations
 2. ZFS VDEV
 3.
 4.
 5.
 6.
 7.
 8.
 9.
 10.
 11.
 12.
 13.
 14.
 15.
 16.
 17.
 18.
 19.
 20.
 21.
 22.
 23.
 24.
 25.
 26.
 27.
 28.
 29.
 30.
 31.
 32.
 33.
 34.
 35.
 36.
 37.
 38.
 39.
 40.
 41.
 42.
 43.
 44.
 45.
 46.
 47.
 48.
 49.
 50.
 51.
 52.
 53.
 54.
 55.
 56.
 57.
 58.
 59.
 60.
 61.
 62.
 63.
 64.
 65.
 66.
 67.
 68.
 69.
 70.
 71.
 72.
 73.
 74.
 75.
 76.
 77.
 78.
 79.
 80.
 81.
 82.
 83.
 84.
 85.
 86.
 87.
 88.
 89.
 90.
 91.
 92.
 93.
 94.
 95.
 96.
 97.
 98.
 99.
 100.

1. Single Disk Virtual Device Configurations
 2. ZFS VDEV
 3.
 4.
 5.
 6.
 7.
 8.
 9.
 10.
 11.
 12.
 13.
 14.
 15.
 16.
 17.
 18.
 19.
 20.
 21.
 22.
 23.
 24.
 25.
 26.
 27.
 28.
 29.
 30.
 31.
 32.
 33.
 34.
 35.
 36.
 37.
 38.
 39.
 40.
 41.
 42.
 43.
 44.
 45.
 46.
 47.
 48.
 49.
 50.
 51.
 52.
 53.
 54.
 55.
 56.
 57.
 58.
 59.
 60.
 61.
 62.
 63.
 64.
 65.
 66.
 67.
 68.
 69.
 70.
 71.
 72.
 73.
 74.
 75.
 76.
 77.
 78.
 79.
 80.
 81.
 82.
 83.
 84.
 85.
 86.
 87.
 88.
 89.
 90.
 91.
 92.
 93.
 94.
 95.
 96.
 97.
 98.
 99.
 100.

1. Single Disk Virtual Device Configurations
 2. ZFS VDEV
 3.
 4.
 5.
 6.
 7.
 8.
 9.
 10.
 11.
 12.
 13.
 14.
 15.
 16.
 17.
 18.
 19.
 20.
 21.
 22.
 23.
 24.
 25.
 26.
 27.
 28.
 29.
 30.
 31.
 32.
 33.
 34.
 35.
 36.
 37.
 38.
 39.
 40.
 41.
 42.
 43.
 44.
 45.
 46.
 47.
 48.
 49.
 50.
 51.
 52.
 53.
 54.
 55.
 56.
 57.
 58.
 59.
 60.
 61.
 62.
 63.
 64.
 65.
 66.
 67.
 68.
 69.
 70.
 71.
 72.
 73.
 74.
 75.
 76.
 77.
 78.
 79.
 80.
 81.
 82.
 83.
 84.
 85.
 86.
 87.
 88.
 89.
 90.
 91.
 92.
 93.
 94.
 95.
 96.
 97.
 98.
 99.
 100.

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

1. Single Disk Virtual Device Configurations
 2. ZFS VDEV
 3.
 4.
 5.
 6.
 7.
 8.
 9.
 10.
 11.
 12.
 13.
 14.
 15.
 16.
 17.
 18.
 19.
 20.
 21.
 22.
 23.
 24.
 25.
 26.
 27.
 28.
 29.
 30.
 31.
 32.
 33.
 34.
 35.
 36.
 37.
 38.
 39.
 40.
 41.
 42.
 43.
 44.
 45.
 46.
 47.
 48.
 49.
 50.
 51.
 52.
 53.
 54.
 55.
 56.
 57.
 58.
 59.
 60.
 61.
 62.
 63.
 64.
 65.
 66.
 67.
 68.
 69.
 70.
 71.
 72.
 73.
 74.
 75.
 76.
 77.
 78.
 79.
 80.
 81.
 82.
 83.
 84.
 85.
 86.
 87.
 88.
 89.
 90.
 91.
 92.
 93.
 94.
 95.
 96.
 97.
 98.
 99.
 100.

Three Disks

1. Single Disk Virtual Device Configurations
 2. ZFS VDEV
 3.
 4.
 5.
 6.
 7.
 8.
 9.
 10.
 11.
 12.
 13.
 14.
 15.
 16.
 17.
 18.
 19.
 20.
 21.
 22.
 23.
 24.
 25.
 26.
 27.
 28.
 29.
 30.
 31.
 32.
 33.
 34.
 35.
 36.
 37.
 38.
 39.
 40.
 41.
 42.
 43.
 44.
 45.
 46.
 47.
 48.
 49.
 50.
 51.
 52.
 53.
 54.
 55.
 56.
 57.
 58.
 59.
 60.
 61.
 62.
 63.
 64.
 65.
 66.
 67.
 68.
 69.
 70.
 71.
 72.
 73.
 74.
 75.
 76.
 77.
 78.
 79.
 80.
 81.
 82.
 83.
 84.
 85.
 86.
 87.
 88.
 89.
 90.
 91.
 92.
 93.
 94.
 95.
 96.
 97.
 98.
 99.
 100.

1. Single Disk Virtual Device Configurations
 2. ZFS VDEV
 3.
 4.
 5.
 6.
 7.
 8.
 9.
 10.
 11.
 12.
 13.
 14.
 15.
 16.
 17.
 18.
 19.
 20.
 21.
 22.
 23.
 24.
 25.
 26.
 27.
 28.
 29.
 30.
 31.
 32.
 33.
 34.
 35.
 36.
 37.
 38.
 39.
 40.
 41.
 42.
 43.
 44.
 45.
 46.
 47.
 48.
 49.
 50.
 51.
 52.
 53.
 54.
 55.
 56.
 57.
 58.
 59.
 60.
 61.
 62.
 63.
 64.
 65.
 66.
 67.
 68.
 69.
 70.
 71.
 72.
 73.
 74.
 75.
 76.
 77.
 78.
 79.
 80.
 81.
 82.
 83.
 84.
 85.
 86.
 87.
 88.
 89.
 90.
 91.
 92.
 93.
 94.
 95.
 96.
 97.
 98.
 99.
 100.

RAID-Z1 的 配置 在 VDEV 中 是 通过 指定 的 磁盘 数量 来 实现 的 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。

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

在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。

Four or Five Disks

4 或 5 的 配置 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。

在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。

在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。

在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。 在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。

在 配置 时 需要 指定 磁盘 数量 和 冗余 因子 。

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 읽기 속도 (MB/s)는 쓰기 속도보다 빠르며, RAID-Z2의 읽기 속도는 쓰기 속도의 절반이며, RAID-Z3의 읽기 속도는 쓰기 속도의三分之一입니다.

각각의 VDEV은 여러 개의 디스크로 구성됩니다. 각 VDEV은 n-1개의 디스크로 구성되며, n개의 디스크가 있는 VDEV은 n-1개의 디스크를 사용하여 데이터를 저장합니다. 각 VDEV은 n개의 디스크를 사용하여 데이터를 저장하며, n개의 디스크가 있는 VDEV은 n-1개의 디스크를 사용하여 데이터를 저장합니다. 각 VDEV은 n개의 디스크를 사용하여 데이터를 저장하며, n개의 디스크가 있는 VDEV은 n-1개의 디스크를 사용하여 데이터를 저장합니다.

6~12??? ??? (Six to Twelve Disks)

각각의 디스크는 여러 개의 VDEV에 사용될 수 있으며, 각 VDEV은 여러 개의 디스크를 사용하여 데이터를 저장합니다.

6개의 디스크를 사용하여 3개의 2개의 디스크로 구성된 VDEV을 구성할 수 있으며, 각 VDEV은 3개의 디스크를 사용하여 데이터를 저장합니다. 각 VDEV은 3개의 디스크를 사용하여 데이터를 저장하며, 3개의 VDEV은 9개의 디스크를 사용하여 데이터를 저장합니다.

각각의 6개의 디스크를 사용하여 2개의 3개의 디스크로 구성된 RAID-Z VDEV을 구성할 수 있으며, 12개의 디스크를 사용하여 데이터를 저장합니다. 각 VDEV은 3개의 디스크를 사용하여 데이터를 저장하며, 2개의 VDEV은 6개의 디스크를 사용하여 데이터를 저장합니다.

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

?? 3. ?(Pools)

ZFS 的 池 (pool) 是 zpool 命令 创建的 , 池 是 存储 数据 的 地方 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 .

ZFS ??

UFS 的 extfs 是 一个 文件系统 . 池 是 一个 存储 数据 的 地方 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 .

ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 .

池 的 名称 是 由 用户 指定 的 , ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 .

池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 .

ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 .

ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 .

?????, RAID ? ? (Stripes, RAID, and Pools)

池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 . ZFS 的 池 是 由 一个 或多个 物理 存储 设备 组成 的 . 池 的 名称 是 由 用户 指定 的 .

RAID 1 的寫入速度會比 RAID 0 慢，因為 RAID 1 需要將資料寫入兩個磁碟。RAID 10 則是 RAID 0 和 RAID 1 的組合，它提供了較高的讀取速度，但寫入速度會比 RAID 0 慢。RAID 10 的磁碟配置如下：

RAID 10 的磁碟配置如下：

ZFS 的磁碟配置如下：

ZFS 的磁碟配置如下：

ZFS 的磁碟配置如下：

ZFS 的磁碟配置如下：

ZFS 的磁碟配置如下：

?? (Viewing Pools)

zpool list

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

的 的 的 的 的 的 . 的 的 的 **db** **zroot** 的 的 的 的 的 .
 的 的 的 的 的 的 的 的 的 的 . 的 , 的 的 的 , 的 的 的 的 的
 的 .
EXPANDSZ 的 的 的 的 的 的 的 的 的 的 的 . 5 的 的 的 的
 的 的 的 的 的 . 的 的 的 的 的 的 的 的 的 .
 的 的 的 的 的 的 的 .
FRAG 的 的 的 的 的 的 的 . 的 的 的 的 的 .
CAP 的 的 的 的 的 的 的 .
DEDUP 的 的 的 的 的 的 的 . 6 的 的 的 的 的 .
 .
 的 **HEALTH** 的 的 VDEV 的 的 的 . 的 的 的 的 的 的 的 的
 的 的 的 的 . 5 的 的 的 的 .
 的 , **ALTROOT** 的 的 的 的 , 的 " 的 的 " 的 的 .

4 的 的 的 的 . 的 的 的 的 的 的 **zpool list** 的 的 的
 的 .
 的 的 的 **prod** 的 **test** 的 的 .

```
$ zpool list prod test
```

的 的 的 的 的 的 的 的 的 **-v** 的 的 . 的 的 的
 的 的 的 的 .

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

-p 的 的 的 的 的 的 的 的 的 , **-H** 的 的 的 .
 的 的 的 的 的 .

的 VDEV 的 的 的 的 的 **zpool status** 的 的 . 的 的 的
 zpool 的 的 的 的 的 .

的 的 的 **zpool status -x** 的 .

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

的 的 的 的 的 .

?? ?? VDEV (Multiple VDEVs)

的 的 的 VDEV 的 的 的 . VDEV 的 的 的 的 的 的
 的 的 的 . 的 的 的 VDEV 的 的 . 的 的 的 的
 的 , 的 的 的 的 的 **ZFS** VDEV 的 的 的 .

2 的 的 VDEV 的 的 的 . 的 的 的 的 . 的 VDEV
 的 的 的 的 , 的 的 的 (的 的) 的 的 的 的


```
/etc/sysctl.conf  sysctl vfs.zfs.min_auto_ashift  ashift
```

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

`/etc/sysctl.conf`

```
# FreeBSD 10.1 #root@ashift:~# sysctl kern.cam
```

?? FreeBSD Ashift (Older FreeBSD Ashift)

10.1 10.1 FreeBSD 10.1 10.1 FreeBSD 10.1 10.1 ashift sysctl 10.1 10.1 , ZFS 10.1 10.1 10.1 10.1 10.1 . 10.1 10.1 10.1 10.1 , 10.1 10.1 10.1 10.1 .

```
# 建立 檔案 目錄 目錄 目錄 目錄 . FreeBSD 的 GEOM 的 gnop(8)
# 目錄 目錄 目錄 目錄 目錄 . gnop 目錄 目錄 目錄 目錄 目錄
# 目錄 目錄 (目錄 目錄 目錄 目錄 目錄 ). "目錄 目錄 目錄 目錄 , 4096 目錄 目錄
# 目錄 目錄 ." 目錄 gnop 目錄 目錄 . 目錄 目錄 zpool 目錄 目錄 . 目錄
/dev/gpt/zfs0 目錄 目錄 gnop 目錄 目錄 .
```

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

```

$ ls -ld /dev/gpt/zfs0.nop
lrwxrwxrwx. 1 root root 10 2023-08-24 10:00 /dev/gpt/zfs0.nop -> /dev/zfs
$ ls -ld /dev/zfs
lrwxrwxrwx. 1 root root 10 2023-08-24 10:00 /dev/zfs -> /dev/zfs
$ ls -ld /dev/zfs
lrwxrwxrwx. 1 root root 10 2023-08-24 10:00 /dev/zfs -> /dev/zfs

```

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

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

?? ?? ???? ???? (Creating Pools and VDEVs)

[illegible]

```

00000000  00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000  00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000  00000000 00000000 00000000 00000000 00000000 00000000 00000000





```

?? ???? (Sample Drives)

```
0####      ##   #####    #   ##   #   ##   ##   ##   ##   ##   ##   ##   ##   ##   ##
#####     . #####       ##   #####     . ####   #####   ##   ##   ##   ##   ##   ##
#####          zfs#   ##   # GPT #####         . # #####   ## 1GB ##   #####   gpart(8)#
##   ## ZFS #####   ## 6## 1TB #####   #####   .
```

```
$ gpart create -s gpt da0
$ gpart add -a 1m -slg -l sw0 -t freebsd-swap da0
$ gpart add -a 1m -l 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)
```

 GPT ZFS  gpt/zfs0  gpt/zfs5  . 

????? ? (Striped Pools)

```

00  0000  00  0000  0000  000  00  000  00000
0000  0000  000  000  00000  0000  00  00000
00  00  000  00000  . 00  000  00  000000  0000  00  000  . 0000  500
0000  0000  000  00  00000  .

```

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

```

[ ] [ ] [ ] [ ] [ ] [ ] . [ ] zpool status [ ] [ ] [ ] [ ] [ ] [ ] [ ]

```

```
$ 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個 物理 磁 磁頭 . 磁 磁頭 VDEV磁 磁 .

磁 磁 磁 磁頭 磁 磁 磁頭 . 磁 磁 磁 磁 VDEV磁 磁 磁頭
磁頭磁頭 , VDEV磁 磁頭 磁頭 . 磁頭 磁 磁頭磁頭 磁頭 磁頭 .
磁 磁 磁 磁頭 磁頭 .

?? ? (Mirrored Pools)

磁頭 磁 磁 磁頭 磁 磁頭 磁頭 磁頭 . 磁 磁 磁頭 磁頭
磁頭 磁 磁 磁 磁 磁頭 磁頭 磁頭 . 磁 磁頭 磁 磁 磁頭 磁頭 , 磁 磁頭
磁頭 .

磁 **zpool create** 磁 磁 磁頭 . **mirror** 磁頭 磁頭 磁頭 磁 磁頭
磁頭 . 磁 磁頭 磁 磁頭 ashift磁 磁頭 .

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

zpool status 磁 磁 磁頭 .

```
$ 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 磁 磁 mirror-0磁 磁 磁頭 . mirror-0 磁 VDEV磁 磁 . 磁 VDEV磁
磁 磁 磁 , 磁 **gpt/zfs0** **gpt/zfs1** 磁 磁頭 磁頭 . 磁 磁 磁 磁 磁頭 磁頭 磁頭
磁 磁頭 . 磁 磁 磁 磁 磁頭 磁頭 .

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

我们使用 `zpool create` 命令来创建 RAID-Z 池。在创建 RAID-Z 池时，我们使用 `reflect` 选项来指定 RAID 级别为 RAID-Z。
 (FreeBSD Mastery: Advanced ZFS 第 10 章 第 10.1 节)。

RAID-Z Pools

在创建 RAID-Z 池时，我们使用 `zpool create` 命令。RAID-Z 池的创建命令如下所示：
 `zpool create` 命令的选项包括 `reflect`、`mirror`、`raidz` 等。
 我们使用 `zpool create` 命令来创建 RAID-Z 池。
 RAID-Z 池的创建命令如下所示：
 `zpool create` 命令的选项包括 `reflect`、`mirror`、`raidz` 等。
 RAID-Z 池的创建命令如下所示：
 `zpool create` 命令的选项包括 `reflect`、`mirror`、`raidz` 等。

在创建 RAID-Z 池时，我们使用 `zpool create` 命令。RAID-Z 池的创建命令如下所示：
 `zpool create` 命令的选项包括 `reflect`、`mirror`、`raidz` 等。

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

我们使用 `zpool status` 命令来查看 RAID-Z 池的状态。
 我们使用 `zpool status` 命令来查看 RAID-Z 池的状态。

```
$ 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
```

我们使用 `zpool create` 命令来创建 RAID-Z 池。RAID-Z 池的创建命令如下所示：
 `zpool create` 命令的选项包括 `reflect`、`mirror`、`raidz` 等。
 RAID-Z 池的创建命令如下所示：
 `zpool create` 命令的选项包括 `reflect`、`mirror`、`raidz` 等。

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

我们使用 `zpool status` 命令来查看 RAID-Z 池的状态。
 我们使用 `zpool status` 命令来查看 RAID-Z 池的状态。

```
$ 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 0 is mirrored by VDEV 1 and 2.
# VDEV 1 is mirrored by VDEV 0 and 2.
# VDEV 2 is mirrored by VDEV 0 and 1.
# VDEV 3 is mirrored by VDEV 4 and 5.
# VDEV 4 is mirrored by VDEV 3 and 5.
# VDEV 5 is mirrored by VDEV 3 and 4.
# VDEV 6 is mirrored by VDEV 7 and 8.
# VDEV 7 is mirrored by VDEV 6 and 8.
# VDEV 8 is mirrored by VDEV 6 and 7.
```

[illegible]

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

```
# pool create barrel zpool(8) barrel # pool mirror
# . mirror " " . # gpt/zfs0
# gpt/zfs1 # . # mirror #
# VDEV 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  RAID-10 .
.ZFS VDEV  RAID  VDEV  . FreeBSD  GEOM
RAID  VDEV  RAID  , RAID
RAID-Z1 VDEV
.
```

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

1. RAID-Z1 VDEV 3 创建 3 个 , 1 **gpt/zfs0, gpt/zfs1, gpt/zfs2** 创建 .
 2. 创建 **gpt/zfs3, gpt/zfs4, gpt/zfs5** . zpool 1 创建 1 个 .
 3. 创建 . 1 个 1 个 RAID-Z 1 个 1 个 .

```
$ 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

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

00 00 00 00 00 00 VDEV 00 0000 00 0000 . 00 00 00 00 VDEV
 00 00 0 00 , 0000 00 00 VDEV 00 0000 . 00 00 VDEV 00 0000 IOPS
 0 00 0000 0000 .

?? ?? ?? (Using Log Devices)

2個 1個 1個 , ZFS 1個 1個 1個 1個 /1個 1個 1個 1個 1個 1個 1個 1個
1個 1個 . 1個 1個 1個 1個 1個 1個 1個 1個 SSD 1個 . **zpool(8)** 1個 1個
1個 **log**, 1個 1個 **cache** 1個 1個 . 1個 1個 1個 **log** 1個 **cache** 1個 1個 1個
1個 1個 . 1個 1個 1個 1個 1個 1個 1個 1個 1個 1個 1個 1個
1個 .

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

1個 1個 1個 1個 1個 .

```
$ 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

1個 1個 1個 1個 1個 1個 1個 1個 1個 1個 1個 1個 . 1個 1個 1個 1個 1個 1個
1個 1個 . 1個 1個 1個 1個 ZFS 1個 1個 1個 1個 1個 1個 . 1個 ZIL 1個
1個 1個 1個 1個 1個 1個 1個 1個 1個 1個 . 1個 *gpt/zfs0* 1個
gpt/zfs3 1個 1個 1個 1個 1個 1個 1個 , 1個 1個 1個 *gpt/zlog0*
gpt/zlog1 1個 1個 .

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

1個 1個 1個 1個 1個 1個 1個 1個 1個 1個 1個 1個 . 1個 1個 1個 1個 1個 1個
1個 1個 1個 1個 , 1個 1個 1個 1個 1個 . 1個 1個 SSD 1個 1個 1個 1個 1個
1個 1個 1個 1個 !

???? ?? VDEV (Mismatched VDEVs)

1個 1個 1個 1個 VDEV 1個 1個 1個 1個 , **zpool(8)** 1個 1個 1個 1個
1個 .

```
$ 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) 提供了一種簡單的方法來創建和管理 ZFS 池。要創建一個池，請使用 **zpool create** 命令。例如，要創建一個名為 **db** 的池，並使用四個 VDEV 設備（**gpt/zfs1**、**gpt/zfs2**、**gpt/zfs3** 和 **gpt/zfs4**），請執行以下命令：

zpool create db gpt/zfs1 gpt/zfs2 gpt/zfs3 gpt/zfs4

如果設備名稱包含空格，請使用引號將其括起來。例如，要創建一個名為 **db** 的池，並使用四個 VDEV 設備（**gpt/zfs1**、**gpt/zfs2**、**gpt/zfs3** 和 **gpt/zfs4**），請執行以下命令：

??? ??? (Reusing Providers)

在創建池時，如果指定的 VDEV 設備已經被另一個池使用，則會收到錯誤消息。要解決這個問題，請使用 **-f** 選項來覆蓋錯誤。例如，要創建一個名為 **db** 的池，並使用四個 VDEV 設備（**gpt/zfs1**、**gpt/zfs2**、**gpt/zfs3** 和 **gpt/zfs4**），請執行以下命令：

```
$ 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'
```

要創建一個池，並使用四個 VDEV 設備（**gpt/zfs1**、**gpt/zfs2**、**gpt/zfs3** 和 **gpt/zfs4**），請執行以下命令：

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

如果設備名稱包含空格，請使用引號將其括起來。例如，要創建一個名為 **db** 的池，並使用四個 VDEV 設備（**gpt/zfs1**、**gpt/zfs2**、**gpt/zfs3** 和 **gpt/zfs4**），請執行以下命令：

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

ZFS 池創建後，您可以使用 **zpool status** 命令來查看池的狀態。例如，要查看 **db** 池的狀態，請執行以下命令：

zpool status db

? ??? (Pool Integrity)

ZFS 池創建後，您可以使用 **fsck(8)** 命令來檢查池的完整性。例如，要檢查 **db** 池的完整性，請執行以下命令：

fsck(8) db

如果池的完整性檢查通過，則會顯示 **OK**。如果池的完整性檢查失敗，則會顯示錯誤消息。要解決這個問題，請使用 **-f** 選項來覆蓋錯誤。例如，要檢查 **db** 池的完整性，請執行以下命令：

ZFS ??? (ZFS Integrity)

要檢查 ZFS 池的完整性，您可以使用 **zfs** 命令。例如，要檢查 **db** 池的完整性，請執行以下命令：

zfs

如果池的完整性檢查通過，則會顯示 **OK**。如果池的完整性檢查失敗，則會顯示錯誤消息。要解決這個問題，請使用 **-f** 選項來覆蓋錯誤。例如，要檢查 **db** 池的完整性，請執行以下命令：


```
$ zpool scrub zroot
```

zpool status 命令用于检查 zpool 池的状态。它显示了池的扫描进度、扫描速度、扫描时间以及扫描结果。

```
$ 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
...
```

zpool status 命令的输出显示了池的扫描进度、扫描速度、扫描时间以及扫描结果。如果池处于扫描状态，输出会显示扫描的进度和预计完成时间。如果池处于空闲状态，输出会显示池的健康状态。

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

zpool scrub -s 命令用于手动启动池的扫描。它会将池置于扫描状态，并开始扫描池中的数据。

??? ?? (Scrub Frequency)

ZFS 池的扫描频率是由池的属性 scrub 控制的。该属性决定了池的扫描频率，可以是 on (每次写入数据时) 或 onerror (仅在检测到错误时)。

要查看池的 scrub 属性，可以使用 zpool get scrub 命令。要更改池的 scrub 属性，可以使用 zpool set scrub 命令。

? ?? (Pool Properties)

ZFS 池的属性可以分为池属性、数据集属性和文件系统属性。池属性是池级别的属性，数据集属性是数据集级别的属性，文件系统属性是文件系统级别的属性。

要查看池的属性，可以使用 zpool get 命令。要更改池的属性，可以使用 zpool set 命令。

? ?? ?? (Viewing Pool Properties)

zpool get all 命令用于查看池的所有属性。它会将池的所有属性及其值输出到终端。

```
$ zpool get all zroot
NAME      PROPERTY  VALUE      SOURCE
zroot     ONLINE    ONLINE    
```



```
zroot  size      920G  -
zroot  capacity  1%    -
zroot  altroot   -      default
zroot  health    ONLINE -
...
```

□ □ □ □ □ . □ □ □ □ □ □ , □ □ □ □ □ □ , □ □ □ □ □ □
□ □ □ □ □ □ □ □ □ □ . □ □ □ □ □ □ 920G□ , □ □ □ 920GB□ □ □ □ □ □ .

SOURCE [redacted] [redacted] [redacted] [redacted] [redacted] . [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted]

[redacted] [redacted] [redacted] . [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted]

[redacted] . [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] . FreeBSD [redacted] [redacted] [redacted] [redacted]

[redacted] . [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] , [redacted] [redacted] [redacted] [redacted]

[redacted] [redacted] [redacted] .

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ **zpool get** □ □ □ □ □ .

```
$ zpool get size
```

NAME	PROPERTY	VALUE	SOURCE
db	size	2.72T	-
zroot	size	920G	-

?? ?? (Changing Pool Properties)

```

| | |||| | || |||| | || |||||| . zpool set ||| ||| | ||
|||||. |||| | comment || |||||.

```

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

```
$ zpool get comment
```

NAME	PROPERTY	VALUE	SOURCE
db	comment	-	default
zroot	comment	Main OS files	local

[illegible]

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

zpool 命令使用 -o 选项来设置 zpool 的属性。

zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。

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

zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。

? (Pool History)

zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。

zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。

```
$ 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
...
```

FreeBSD 系统使用 zpool 命令来管理 zfs 池。FreeBSD 系统使用 zpool 命令来管理 zfs 池。FreeBSD 系统使用 zpool 命令来管理 zfs 池。FreeBSD 系统使用 zpool 命令来管理 zfs 池。

zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。zpool 命令使用 -o 选项来设置 zpool 的属性。

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

comment 属性用于设置 zpool 的注释。comment 属性用于设置 zpool 的注释。comment 属性用于设置 zpool 的注释。comment 属性用于设置 zpool 的注释。

zpool 是 ZFS 的池，是 ZFS 文件系统的数据容器。zpool 是 ZFS 的池，是 ZFS 文件系统的数据容器。zpool 是 ZFS 的池，是 ZFS 文件系统的数据容器。

Zpool ?? ?? ??? (Zpool Maintenance Automation)

FreeBSD 的 `periodic(8)` 是用于定期执行任务的工具。它可以根据系统的时间表来执行任务。在 `periodic.conf` 文件中，我们可以配置 `daily_status_zfs_enable` 来启用 ZFS 的日常状态检查。

```
daily_status_zfs_enable="YES"
```

通过 `periodic(8)` 配置，我们可以让系统定期执行 `zpool status -x` 命令，以检查所有池的健康状态。

此外，我们还可以配置 `daily_status_zfs_zpool_list` 来指定要检查的池列表。在 `periodic.conf` 文件中，我们可以配置 `daily_status_zpool` 来启用 ZFS 的日常池状态检查。

FreeBSD 的 `periodic(8)` 是用于定期执行任务的工具。它可以根据系统的时间表来执行任务。在 `periodic.conf` 文件中，我们可以配置 `daily_scrub_zfs_enable` 来启用 ZFS 的日常数据完整性检查。

```
daily_scrub_zfs_enable="YES"
```

此外，我们还可以配置 `daily_scrub_zfs_pools` 来指定要检查的池列表。在 `periodic.conf` 文件中，我们可以配置 `daily_scrub_zfs_pools` 来启用 ZFS 的日常池数据完整性检查。

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

此外，我们还可以配置 `daily_scrub_zfs_default_threshold` 来指定默认的阈值。

```
daily_scrub_zfs_default_threshold="10"
```

此外，我们还可以配置 `daily_scrub_zfs_${poolname}_threshold` 来指定特定池的阈值。

```
daily_scrub_zfs_prod_threshold="7"
```

通过配置 `periodic(8)`，我们可以实现 ZFS 池的日常维护自动化。

? ?? (Removing Pools)

要删除 ZFS 池，我们可以使用 `zpool destroy` 命令。

```
$ zpool destroy test
```

zpool 的 destroy 命令会强制删除 zpool 及其所有数据。如果 zpool 正在使用中，则会先将其中的所有数据写入到备用设备中，然后才会删除。这可能会导致数据丢失，因此在使用前请务必备份重要数据。

zpool 的 destroy 命令会强制删除 zpool 及其所有数据。如果 zpool 正在使用中，则会先将其中的所有数据写入到备用设备中，然后才会删除。这可能会导致数据丢失，因此在使用前请务必备份重要数据。

Zpool ?? ??? (Zpool Feature Flags)

ZFS 的 feature 标志用于控制某些功能的行为。这些标志可以分为三类：enabled、active 和 disabled。enabled 标志表示该功能已启用，但尚未生效；active 标志表示该功能已生效；disabled 标志表示该功能已禁用。可以通过 zpool get 命令查看 zpool 的 feature 标志。

zpool 的 feature 标志可以通过 zpool get 命令查看。例如，要查看 zpool 的 feature 标志，可以运行以下命令：
zpool get feature zroot
这将输出 zpool 的 feature 标志及其状态。如果 feature 标志为 enabled，则表示该功能已启用，但尚未生效；如果为 active，则表示该功能已生效；如果为 disabled，则表示该功能已禁用。

OpenZFS 的 feature 标志可以通过 zpool get 命令查看。例如，要查看 OpenZFS 的 feature 标志，可以运行以下命令：
zpool get feature zroot
这将输出 OpenZFS 的 feature 标志及其状态。如果 feature 标志为 enabled，则表示该功能已启用，但尚未生效；如果为 active，则表示该功能已生效；如果为 disabled，则表示该功能已禁用。

FreeBSD 的 zpool-features(7) 手册页提供了有关 zpool 的 feature 标志的详细信息。可以通过 man zpool-features(7) 查看该手册页。该手册页列出了所有可用的 feature 标志及其默认状态。

zpool 的 feature 标志可以通过 zpool get 命令查看。例如，要查看 zpool 的 feature 标志，可以运行以下命令：
zpool get feature zroot
这将输出 zpool 的 feature 标志及其状态。如果 feature 标志为 enabled，则表示该功能已启用，但尚未生效；如果为 active，则表示该功能已生效；如果为 disabled，则表示该功能已禁用。

?? ??? ?? (Viewing Feature Flags)

zpool 的 feature 标志可以通过 zpool get 命令查看。例如，要查看 zpool 的 feature 标志，可以运行以下命令：
zpool get feature zroot

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

如果 在 系统 启动 时 遇到 问题 ， 请 检查 系统 日志 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。

如果 在 系统 启动 时 遇到 问题 ， 请 检查 系统 日志 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。

在 系统 启动 时 ， 请 检查 系统 日志 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。

在 系统 启动 时 ， 请 检查 系统 日志 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。

在 系统 启动 时 ， 请 检查 系统 日志 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。

在 系统 启动 时 ， 请 检查 系统 日志 。 如果 问题 仍然存在 ， 请 联系 技术支持 团队 。

4. ZFS (ZFS Datasets)

1. 在 /etc/fstab 文件中添加以下配置，以启用 NTFS 文件系统支持：


```

# 在 /etc/fstab 文件中添加以下配置
ntfs-3g /mnt/ntfs /dev/sdb1 defaults,uid=1000,gid=1000,umask=022 0

```

ZFS 11 111 1111 1111 111 11 11111 1111 1111 1111 111111 1
 111 11111 . 1111 1 ZFS 111 111 1 11 111 1111 1 111 111 11111 .
 1 111 111 11 11 11 111 1111 1 1111 111 111 11 111 1 1 1111 . 6
 111 111 11 11111 111 111 111 11111 111 11 1111 111 111 1
 1111 .

```

# 检查 /var 分区是否挂载
if [ -d /var ]; then
    echo " /var 分区已挂载。"
else
    echo " /var 分区未挂载。"
fi

# 检查 /home 分区是否挂载
if [ -d /home ]; then
    echo " /home 分区已挂载。"
else
    echo " /home 分区未挂载。"
fi

# 检查 UFS 文件系统是否挂载
if [ -d /tunefs(8) ]; then
    echo " UFS 文件系统已挂载。"
else
    echo " UFS 文件系统未挂载。"
fi

```




















 . ZFS 













 .



????? (Datasets)



ZFS 的 目录 结构 是 扁平 的 。 在 传统 的 文件 系统 中 ， 我们 通常 会 看到 类似 的 目录 结构 。

/home 的 目录 结构 是 扁平 的 。 在 传统 的 文件 系统 中 ， 我们 通常 会 看到 类似 的 目录 结构 。

在 传统 的 文件 系统 中 ， 我们 通常 会 看到 类似 的 目录 结构 。

在 传统 的 文件 系统 中 ， 我们 通常 会 看到 类似 的 目录 结构 。

在 传统 的 文件 系统 中 ， 我们 通常 会 看到 类似 的 目录 结构 。

zfs(8)

????? ?? (Dataset Types)

而 ZFS 的 文件系统 (filesystems), 卷 (volumes), 快照 (snapshots), 克隆 (clones), 书签 (bookmarks) 这 5 个 概念 都是 相互 关联 的 。

而 我们 通常 使用 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 ZFS 文件系统 的 设计 思想 是 基于 卷 (volumes) 的 文件系统 , setuid 权限 的 使用 也是 基于 卷 (volumes) 的 设计 思想 。 通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 , 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 文件系统 , NFSv4 的 设计 思想 也是 基于 卷 (volumes) 的 设计 思想 , chflags(2) 的 设计 思想 也是 基于 卷 (volumes) 的 设计 思想 。

ZFS 的 设计 思想 是 **zvol** 的 设计 思想 的 延伸 。 而 我们 通常 使用 的 iSCSI 的 设计 思想 也是 基于 卷 (volumes) 的 设计 思想 。 UFS 的 设计 思想 也是 基于 卷 (volumes) 的 设计 思想 。 ZFS 的 设计 思想 也是 基于 卷 (volumes) 的 设计 思想 。 Zvol 的 设计 思想 也是 基于 卷 (volumes) 的 设计 思想 。 FreeBSD 的 设计 思想 也是 基于 卷 (volumes) 的 设计 思想 。

通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。 通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。 通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。

通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。 通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。 通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。

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

通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。 通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。 通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。

而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。 通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。

通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。 通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。

通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。

通常 的 文件系统 都是 基于 块 (blocks) 的 文件系统 。 而 ZFS 的 设计 思想 是 基于 卷 (volumes) 的 设计 思想 。

如何 实现 ?

我们 在 这个 项目 中 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

如何 实现 (Viewing Datasets)

zfs list 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

```
$ zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
mypool                             420M  17.9G   96K    none
mypool/R00T                        418M  17.9G   96K    none
mypool/R00T/default                418M  17.9G  418M  /
...
```

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .


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

이제 우리는 'lamb' / 'baby' 데이터셋을 생성했습니다. , 데이터셋은 'lamb' 데이터셋의 하위 데이터셋입니다.

?? ??? (Creating Volumes)

-V 옵션은 데이터셋의 크기를 지정합니다. **zfs create** 명령을 사용하여 데이터셋을 생성합니다. 이 데이터셋은 4GB의 크기를 가집니다.

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

Zvols은 데이터셋을 ZFS 데이터셋으로 변환합니다. **-t volume** 옵션을 사용하여 ZFS 데이터셋을 Zvol로 변환합니다. **zfs list** 명령을 사용하여 Zvol을 확인합니다.

```
$ zfs list mypool/avolume
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
mypool/avolume	4.13G	17.9G	64K	-

Zvol은 ZFS 데이터셋을 Zvol로 변환합니다. 이 Zvol은 4GB의 크기를 가집니다. Zvol은 4.13GB의 크기를 가집니다.

Zvol은 /dev/zvol/mypool/avolume로 마운트됩니다. 이 Zvol은 /dev/zvol/mypool/avolume로 마운트됩니다.

```
$ ls -al /dev/zvol/mypool/avolume
```

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

newfs(8)은 Zvol을 파일 시스템으로 변환합니다. 이 Zvol은 /dev/zvol/mypool/avolume로 마운트됩니다.

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

zfs rename 명령을 사용하여 데이터셋을 재명명합니다. 이 데이터셋은 db/production로 재명명됩니다.

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

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

-f 옵션은 데이터셋을 재명명합니다. 이 데이터셋은 db/production로 재명명됩니다. **-f** 옵션은 데이터셋을 재명명합니다. 이 데이터셋은 db/production로 재명명됩니다.

이제 우리는 db/production 데이터셋을 생성했습니다.

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

00000000 ZFS 0000 0000 00 0000 0000 00000000 0 000 00 0000 00 0
 0000 . 000 00000 000 00000 000000 00 000 000 0 0000 .
 00000000 000 000 000 0000 0000 .

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

`uu` `uu` `uuuu` , `uu` `uu` `uu` `uu` `uu` `uu` `uu` `uu` . **rename** `uu` -
u `uuuu` `uuuu` ZFS `uu` `uu` `uu` `uuuu` `uuuu` `uuuu` `uu` `uu` `uuuu`
`uu` `uu` `uuuu` . `uuuu` `uu` `uuuuu` `uuuuuu` `uuuu` `uu` `uuuuuu` `uu` `uu` `uu`
`uuuuuu` `uu` `uuuuuu` .





















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

 ?

zfs destroy

 .

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

[illegible]

00000000 0000 0 00 00 000000 0000 000000 -v 0 -n 0000 0000 0 0000 . -v
 0000 0000 0000 00 0000 0000 0000 , -n 0 zfs(8) 000000 000000 000000 .
 0 0 0000 0000 0000 00 0 0000 0000 0000 000000 000000 .

ZFS ?? (ZFS Properties)

ZFS 是 一个 文件 系统 和 卷 管理 系统 的 集合 体 。 它 是 由 开放 存储 基金会 开发 的 ， 旨在 提供 高 性能 、 高 可靠 性 和 高 容错 性 的 数据 存储 方案 。 ZFS 是 一个 跨 平台 的 系统 ， 支持 多种 操作 系统 ， 包括 太阳 微系统 公司 的 太阳 微系统 操作系统 、 惠普 公司 的 惠普 操作系统 、 红帽 公司 的 红帽 企业 Linux 操作系统 等 。 ZFS 是 一个 跨 平台 的 系统 ， 支持 多种 操作 系统 ， 包括 太阳 微系统 公司 的 太阳 微系统 操作系统 、 惠普 公司 的 惠普 操作系统 、 红帽 公司 的 红帽 企业 Linux 操作系统 等 。



?? ?? (Viewing Properties)

```
zfs(8)  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 10
```

NAME	PROPERTY	VALUE	SOURCE
mypool/lamb	compression	lz4	inherited from mypool

SOURCE 00 0 000000 . 00 000 0 000 ZFS 000000 0000 000 000000 . 00
000 0000 0 000000 0 000 00000 00000 00000 . 00 000 000000
0000 0 000000 , 000000 000 0000 0 000 0000 000 000000 . 000
000 0 00 0000 00 "00 /00 00 "00 0000 00 00 000000 00000 .

zfs get

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

```
$ zfs get quota,reservation zroot/home
```

NAME	PROPERTY	VALUE	SOURCE
zroot/home	quota	none	local
zroot/home	reservation	none	default

NAME	QUOTA	RESERV
db	none	none
zroot	none	none
zroot/R00T	none	none
zroot/R00T/default	none	none

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

zfs 的 属性 分为 本地属性 和 全局属性 两种。本地属性 是指 只针对 某个 zfs 实例 生效 的属性。全局属性 是指 针对 整个 zfs 实例 生效 的属性。

?? ?? (Changing Properties)

zfs set 命令 用于 设置 本地属性。例如，将 压缩 属性 设置为 **off**，命令如下：

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

zfs get 命令 用于 获取 本地属性。例如，获取 压缩 属性，命令如下：

```
$ zfs get compression mypool/lamb/baby

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

zfs 的 属性 分为 本地属性 和 全局属性 两种。本地属性 是指 只针对 某个 zfs 实例 生效 的属性。全局属性 是指 针对 整个 zfs 实例 生效 的属性。例如，将 压缩 属性 设置为 **off**，命令如下：

zfs 的 属性 分为 本地属性 和 全局属性 两种。本地属性 是指 只针对 某个 zfs 实例 生效 的属性。全局属性 是指 针对 整个 zfs 实例 生效 的属性。例如，将 压缩 属性 设置为 **off**，命令如下：

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

ZFS 的 属性 分为 本地属性 和 全局属性 两种。本地属性 是指 只针对 某个 zfs 实例 生效 的属性。全局属性 是指 针对 整个 zfs 实例 生效 的属性。例如，将 压缩 属性 设置为 **off**，命令如下：

ZFS 的 属性 分为 本地属性 和 全局属性 两种。本地属性 是指 只针对 某个 zfs 实例 生效 的属性。全局属性 是指 针对 整个 zfs 实例 生效 的属性。例如，将 压缩 属性 设置为 **off**，命令如下：

?? ??? ?? (Filesystem Properties)

zfs 的 属性 分为 本地属性 和 全局属性 两种。本地属性 是指 只针对 某个 zfs 实例 生效 的属性。全局属性 是指 针对 整个 zfs 实例 生效 的属性。例如，将 压缩 属性 设置为 **off**，命令如下：

zfs 的 属性 分为 本地属性 和 全局属性 两种。本地属性 是指 只针对 某个 zfs 实例 生效 的属性。全局属性 是指 针对 整个 zfs 实例 生效 的属性。例如，将 压缩 属性 设置为 **off**，命令如下：

atime

zfs 的 属性 分为 本地属性 和 全局属性 两种。本地属性 是指 只针对 某个 zfs 实例 生效 的属性。全局属性 是指 针对 整个 zfs 实例 生效 的属性。例如，将 压缩 属性 设置为 **off**，命令如下：

zfs 的 属性 分为 本地属性 和 全局属性 两种。本地属性 是指 只针对 某个 zfs 实例 生效 的属性。全局属性 是指 针对 整个 zfs 实例 生效 的属性。例如，将 压缩 属性 设置为 **off**，命令如下：

我们使用 `com.allanjude:backup_ignore` 属性来配置备份忽略。

在 `com.allanjude:backup_ignore` 属性中，我们可以设置 `on` 或 `off`。

```

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

现在，当我们运行 `zfs get` 命令时，我们可以看到 `com.allanjude:backup_ignore` 属性的值为 `true`。

文件系统关系 (Parent/Child Relationships)

在 ZFS 中，文件系统之间的关系是非常重要的。我们可以通过 `zfs(8)` 手册页来了解更多信息。

```

$ zfs get -r compression mypool/lamb
```

NAME	PROPERTY	VALUE	SOURCE
mypool/lamb	compression	lz4	inherited from mypool
mypool/lamb/baby	compression	off	local

从上面的输出中，我们可以看到 `mypool/lamb` 的 `compression` 属性是从 `mypool` 继承的，而 `mypool/lamb/baby` 的 `compression` 属性是本地设置的。

zfs inherit 命令用于将属性从父文件系统继承到子文件系统。

```

$ 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

现在，`mypool/lamb/baby` 的 `compression` 属性已经继承自 `mypool`，其值为 `lz4`。

我们可以使用 `zfs set` 命令来设置文件系统的属性。

```

$ 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

1. 我们使用 `gzip-9` 属性来压缩文件。

2. 继承和重命名 (Inheritance and Renaming)

我们使用 `zfs create` 命令来创建新的文件系统。

我们使用 `zfs get` 命令来查看文件系统的属性。

```

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

我们使用 `zfs rename` 命令来重命名文件系统。

```

$ 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
```

我们使用 `zfs inherit` 命令来继承属性。

我们使用 `zfs set` 命令来设置属性。

3. 移除属性 (Removing Properties)

我们使用 `zfs unset` 命令来移除属性。

我们使用 `zfs inherit` 命令来继承属性。

```

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

我们使用 `zfs get` 命令来查看文件系统的属性。

zfs inherit 命令 用于 设置 文件系统 的 属性 继承 策略。 默认 情况下， 子 文件系统 会 继承 父 文件系统的 属性。

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

zfs inherit 命令 还可以 用于 设置 文件系统 的 其他 属性， 例如 压缩 策略。

ZFS 挂载与卸载 (Mounting ZFS Filesystems)

在 FreeBSD 中， ZFS 文件系统的 挂载 和 卸载 操作 是通过 使用 `mount` 命令 来完成的。 挂载 操作 通常 是在 系统 启动 时 通过 `/etc/fstab` 文件 来配置 的。 对于 CD-ROM 文件系统， 挂载 操作 通常 是在 用户 手动 插入 光盘 后 进行的。

对于 ZFS 文件系统， 挂载 操作 通常 是在 用户 手动 插入 磁盘 后 进行的。 挂载 点 通常 是 `/usr/home`。 挂载 命令 的 格式 如下：

```
$ zfs get mountpoint zroot/usr/home
```

NAME	PROPERTY	VALUE	SOURCE
zroot/usr/home	mountpoint	/usr/home	inherited from zroot/usr

上述命令 的输出 显示， `zroot/usr/home` 的 挂载 点是 `/usr/home`。 这 表明， 该 文件系统 是 通过 继承 的方式 来 设置 挂载 点的。

在 FreeBSD 中， ZFS 文件系统的 挂载 和 卸载 操作 通常 是在 用户 手动 插入 磁盘 后 进行的。 挂载 操作 通常 是在 用户 手动 插入 磁盘 后 进行的。 挂载 点 通常 是 `/usr/home`。 挂载 命令 的 格式 如下：

对于 ZFS 文件系统， 挂载 操作 通常 是在 用户 手动 插入 磁盘 后 进行的。 挂载 点 通常 是 `/usr/home`。 挂载 命令 的 格式 如下：

对于 ZFS 文件系统， 挂载 操作 通常 是在 用户 手动 插入 磁盘 后 进行的。 挂载 点 通常 是 `/usr/home`。 挂载 命令 的 格式 如下：

对于 ZFS 文件系统， 挂载 操作 通常 是在 用户 手动 插入 磁盘 后 进行的。 挂载 点 通常 是 `/usr/home`。 挂载 命令 的 格式 如下：

对于 ZFS 文件系统， 挂载 操作 通常 是在 用户 手动 插入 磁盘 后 进行的。 挂载 点 通常 是 `/usr/home`。 挂载 命令 的 格式 如下：

canmount off 表示 数据 集 可以 被 挂载 到 任何 目录 下 。 而 如果 是 on 则 表示 数据 集 只能 被 挂载 到 指定 的 目录 下 。 在 这里 我们 看到 数据 集 只能 被 挂载 到 /usr 目录 下 。

我们 可以 通过 **canmount** 属性 来 查看 数据 集 的 挂载 属性 。

canmount 属性 可以 通过 以下 命令 来 查看 。

```
zfs list -o name,canmount,mountpoint -r zroot/usr
```

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

ZFS 数据 集 可以 被 挂载 到 任何 目录 下 。 而 如果 是 on 则 表示 数据 集 只能 被 挂载 到 指定 的 目录 下 。 FreeBSD 10.1 版本 中 我们 可以 看到 数据 集 只能 被 挂载 到 /usr 目录 下 。

```
$ zfs mount
zroot/R00T/default  /
zroot/tmp          /tmp
zroot/usr/home     /usr/home
zroot/usr/ports    /usr/ports z
root/usr/src       /usr/src
...
```

/usr 目录 下 的 数据 集 只能 被 挂载 到 /usr 目录 下 。

zfs list 命令 可以 通过 以下 命令 来 查看 数据 集 的 挂载 属性 。

```
zfs list -o name,canmount,mountpoint -r zroot/usr
```

```
$ 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 表示 数据 集 可以 被 挂载 到 任何 目录 下 。 而 如果 是 on 则 表示 数据 集 只能 被 挂载 到 指定 的 目录 下 。 在 这里 我们 看到 数据 集 只能 被 挂载 到 /usr 目录 下 。

我们 可以 通过 **canmount** 属性 来 查看 数据 集 的 挂载 属性 。

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

canmount off 表示 数据 集 可以 被 挂载 到 任何 目录 下 。 而 如果 是 on 则 表示 数据 集 只能 被 挂载 到 指定 的 目录 下 。

canmount off 表示 数据 集 可以 被 挂载 到 任何 目录 下 。

```
FreeBSD  1  1  11111  11  11  zroot  111  111  1111  .  1  111111  11  11  111
1111  1111  111  .
```

0 00 000 000 00000 **mountpoint** 000 00 000 , *zroot* 00 /0 00
 000 000 **canmount** *off* 00 0 000 . 00 0 00000 00 0 00
mountpoint 00 00 . 00 00 00 000 00 0 00000 000 00 00
 000 .

0 00 00 0000 000 00 /opt 0000 000 000 0000 . 000 0000 0 000
 0000 0000 0000 00 000 000 00 0 000 . 00 000000 0000 00
 0000 . 00 000 0000 0000 0000 000 000 000 .

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

□□ □ □ □□□□ □□ /opt□ □□ □□ □□ □□ □ □ □□□□ .

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

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

[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
```

.db/programs , db/data

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

□□ /opt□ □□□□ 2□□ □□□□ 2□ , □ 4□□ □□□□□ □□□□ □□□□ . □□□□
 □□□□ □ □□□□□ □□□□ □□□□□□□□ . □□□ □□ □□□ □□□ □□ □□ □ □□□□ □□ □□
 □□ □□□ □□ □□□□□ . □□□□□ □□ □□□□ □□□□ □□□□ □□□ □□□□□ □□ □□□□□ □□ □□

mount(8)

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

在 `/etc/fstab` 中 ZFS 文件系统 应该 在 哪个 位置 添加 挂载 点 和 选项 。 选项 包括 `noatime, noexec, readonly` 等 。 例如 ， 在 `scratch/junk` 目录下 挂载 `nosuid` 选项 的 文件系统 。 (选项 包括 `atime, exec, rw, suid` 等)。 在 `/tmp` 目录下 挂载 `scratch/junk` 文件系统 时 ， 应该 在 `/etc/fstab` 中添加 以下 内容 。

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

在 挂载 点 上 挂载 文件系统 时 ， 应该 在 `/etc/fstab` 中 添加 挂载 点 和 选项 。

ZFS ?? ?? (Tweaking ZFS Volumes)

Zvol 文件系统 应该 在 哪个 位置 添加 挂载 点 和 选项 。 选项 包括 `noatime, noexec, readonly` 等 。 例如 ， 在 `scratch/junk` 目录下 挂载 `nosuid` 选项 的 文件系统 。

?? ?? (Space Reservations)

`zvol` 的 **volsize** 选项 应该 在 哪个 位置 添加 挂载 点 和 选项 。 选项 包括 `noatime, noexec, readonly` 等 。 例如 ， 在 `scratch/junk` 目录下 挂载 `nosuid` 选项 的 文件系统 。

在 挂载 点 上 挂载 文件系统 时 ， 应该 在 `/etc/fstab` 中 添加 挂载 点 和 选项 。

Zvol 文件系统 应该 在 哪个 位置 添加 挂载 点 和 选项 。 选项 包括 `noatime, noexec, readonly` 等 。 例如 ， 在 `scratch/junk` 目录下 挂载 `nosuid` 选项 的 文件系统 。

在 挂载 点 上 挂载 文件系统 时 ， 应该 在 `/etc/fstab` 中 添加 挂载 点 和 选项 。

zfs create -V 选项 应该 在 哪个 位置 添加 挂载 点 和 选项 。 选项 包括 `noatime, noexec, readonly` 等 。

Zvol ?? (Zvol Mode)

FreeBSD 应该 在 哪个 位置 添加 挂载 点 和 选项 。 选项 包括 `noatime, noexec, readonly` 等 。

通过 **volmode** 指定 `dev` 设备名称，如 `/dev/sda`。通过 `geom` 指定 GEOM 设备名称，如 `geom`。通过 `dev` 指定设备名称，如 `dev`。通过 `none` 指定无设备。

volmode 指定 `none` 表示无设备。ZFS 通过 `volmode` 指定设备名称，如 `geom`，通过 `dev` 指定设备名称，如 `dev`。

volmode 指定 `default` 表示默认设备名称。通过 `sysctl vfs.zfs.vol.mode` 指定设备名称。通过 `zvol` 指定设备名称，如 `1`，通过 `geom`，`2`，`dev`，`3`，`none` 指定设备名称。

通过 `zfs rename` 指定设备名称，如 `zvol`。通过 `zfs rename` 指定设备名称，如 `zvol`。

Dataset Integrity

ZFS 通过 `VDEV` 指定设备名称，如 `geom`。通过 `RAID-Z` 指定设备名称，如 `geom`。通过 `RAID-Z` 指定设备名称，如 `geom`。通过 `RAID-Z` 指定设备名称，如 `geom`。

Checksums

ZFS 通过 `on` 指定设备名称，如 `on`。通过 `on` 指定设备名称，如 `on`。通过 `on` 指定设备名称，如 `on`。

通过 `on` 指定设备名称，如 `on`。通过 `on` 指定设备名称，如 `on`。通过 `on` 指定设备名称，如 `on`。

通过 `fletcher4` 指定设备名称，如 `fletcher4`。通过 `fletcher4` 指定设备名称，如 `fletcher4`。通过 `fletcher4` 指定设备名称，如 `fletcher4`。

`off` 指定设备名称，如 `off`。

`noparity` 指定设备名称，如 `noparity`。通过 `noparity` 指定设备名称，如 `noparity`。通过 `noparity` 指定设备名称，如 `noparity`。

通过 `ZFS` 指定设备名称，如 `ZFS`。通过 `ZFS` 指定设备名称，如 `ZFS`。通过 `ZFS` 指定设备名称，如 `ZFS`。


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











□ □ □ □ 30MB □ , □ □ □ □ □ □ □ □ 10, □ □ 10MB □ □ 2□ 20□ □ .

ls(1) □ □ □ □ □ □ □ □ :

```
$ 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
```






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

이러한 구성을 사용하면, VDEV 구성을 RAID-Z로 구성하여, 데이터의 중복성을 높일 수 있습니다. 이 구성은 3개의 VDEV를 사용하여, 데이터의 중복성을 높일 수 있습니다. 이 구성은 3개의 VDEV를 사용하여, 데이터의 중복성을 높일 수 있습니다.

redundant_metadata           .

redundant_metadata all() ZFS .
 .

redundant_metadata most ZFS

redundant_metadata *most*  **copies** 3       ,
ZFS    6      4  .

[illegible]