ZFS

In ~ 30 Minutes

bill.hathaway@gmail.com

What is ZFS?

A modern take on storage

Safe - transaction based, checksums

Simple - 2 main commands

Efficient - share resources well

Dynamic - change on the fly

Zettabyte is 2^70 bytes

ZFS was developed in response to dealing with 20+ years of complexity and limitations around storage. Todays disk drives are a roughly a million times bigger than the first hard disks. File systems have all sorts of whacky limitations like the number of inodes or items in a directory. Sysadmins have to make choices about how much space to assign to volumes and file systems and then later shrink or add space if they were wrong. Troubleshooting physical hardware, logical volume managers, and file systems increases complexity. Fsck times mean outages can that take hours or days.

With filesystems, safety is a hard constraint. ZFS provides safety through two mechanisms a Copy-on-Write architecture that uses transactions to keep the on-disk format consistent and checksums that can tell if any data is corrupted. If you have redundancy, then the problem can be fixed. No FSCK is needed. Ever.

ZFS is easy to administer because it simplifies the administration model. There are only 2 main commands. Each of these have sub-commands, but the usage is consistent and tries to be intuitive.

ZFS is efficient in that it uses a concept called pooled storage. This means that disk space and disk bandwidth can be easily shared among multiple file systems.

Dynamic - storage can be added, new file systems created, or properties changed all while running

Where can I use ZFS?

Nexenta

zfs-fuse.net

code.google.com/p/maczfs

The operating systems on the left all support ZFS natively.

Apple originally was porting ZFS to OS X, but apparently there was an unsolved licensing dispute and that is no longer the case. A group of enthusiasts are continuing to work with the code to make it available to Mac users.

KQ Infotech announced they would be porting ZFS to Linux and maintaining the port.

Blog at kqinfotech.wordpress.com

ZFS Basic Terms

Volume - block device

File system - standard POSIX FS layer

Snapshot - read-only copy of a FS

Clone - read-write copy of a FS

Dataset - any of the 4 terms above

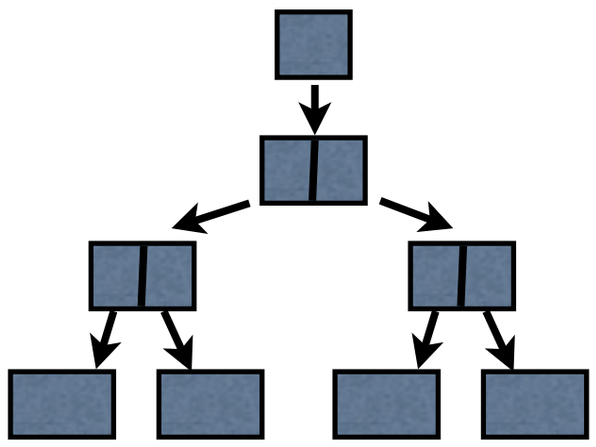
Pool - logical set of vdevs

VDev - block storage (redundancy done here)

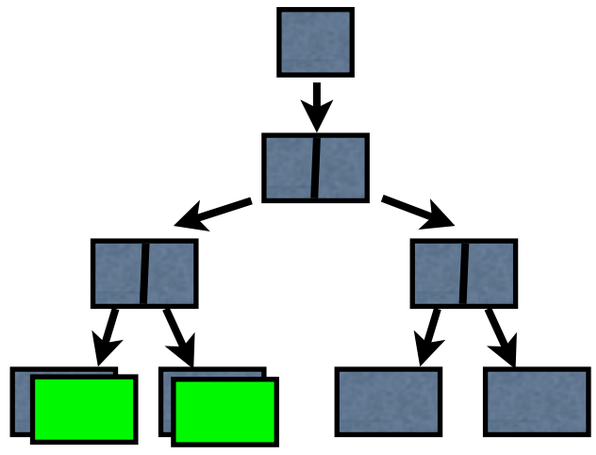
ZFS volumes are typically used to support iSCSI luns or for swap devices

Copy-on-Write

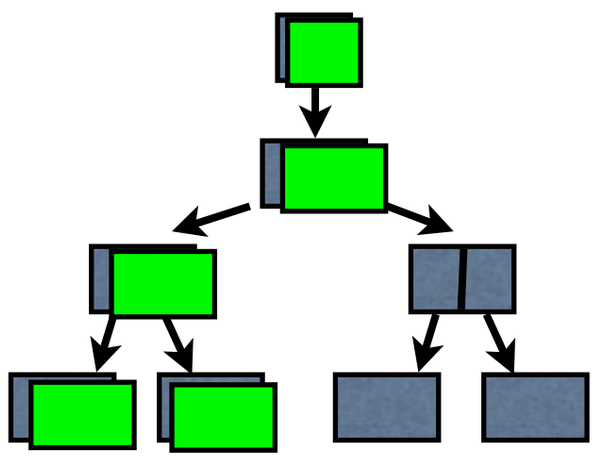
1. Initial tree blocks



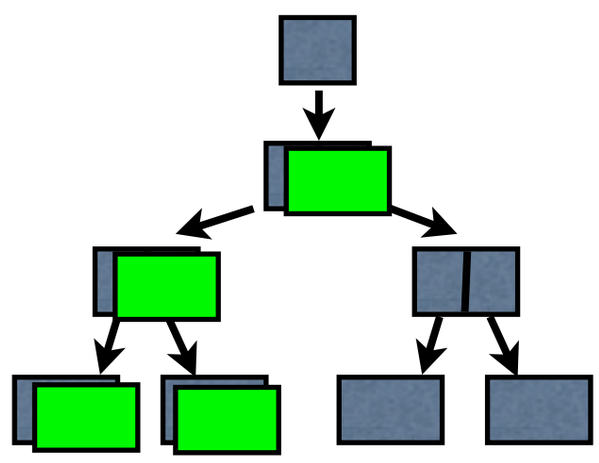
2. CoW some blocks



3. CoW indirect blocks



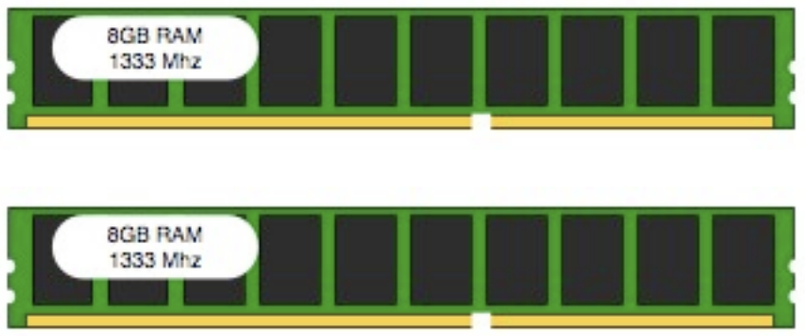
4. Rewrite uberblock



Using the copy-on-write mechanism allows ZFS to operate very safely.

In this example, the original data blocks are blue, and new data is green.

Pooled Storage



Manage disks more like RAM

Pooled storage means that as you add storage to ZFS you don't need to worry about micro-managing it. The storage is made available to all the file systems using the pool, similar to how when you add DIMMs to a system you don't need to reconfigure anything.

No dimmconfig

No /etc/dimmtab

No fdimm

You can create multiple pools of storage per server.

Pooled Storage

Underlying storage is manipulated via **zpool**

zpool **create**

zpool **list**

zpool **status**

zpool **add**

There are also additional sub-commands for tasks such as replacing drives or scrubbing data.

Redundancy is handled at the pool level. When you create a pool you can add drives in a striped fashion, as mirrors, or in a parity configuration similar to RAID5 or 6.

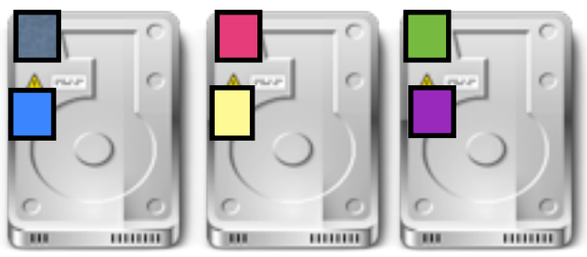
You can also have multiple pools per a machine. An example is where you have a pool mirrored storage to support a high-performance database using 15k RPM disks and another pool that uses slower SATA based storage in a RAID5 like configuration holding archived data.

Vdev Types

Non-redundant

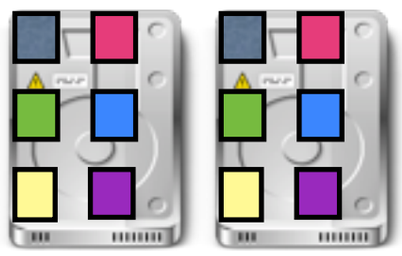
Single disk

Striped



Redundant

Mirror



RAIDZ

Here we are showing the types of vdevs that can be used as building blocks for ZFS pools.

If we start with a single disk, we can expand it later either to a mirrored vdev by attaching a disk or we could change to a striped configuration by adding a disk.

One common complaint with ZFS is that you can't remove disks from striped or RAIDZ configurations, even if you have plenty of available space. This will be remedied when a feature called "block pointer rewrite" gets integrated.

There are double and triple parity versions of RAIDZ called RAIDZ2 and RAIDZ3.

Pooled Storage

zpool **create** data **mirror** c1t0d0 c2t0d0

zpool **list** data

NAME	SIZE	ALLOC	FREE	CAP	DEDUP	HEALTH	ALTROOT
data	496G	164K	496G	0%	1.00x	ONLINE	-

df -h /data

Filesystem	size	used	avail	capacity	Mounted on
data	488G	24K	488G	1%	/data

Here we will create a pool consisting of mirrored storage and then run a zpool list command to see how much space is available.

A file system mounted on the name of the pool will be available by default

Pooled Storage

zpool **status**

```
pool: data
state: ONLINE
scrub: none requested
config:
        NAME          STATE     READ WRITE CKSUM
        data          ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            c1t0d0    ONLINE       0     0     0
            c2t0d0    ONLINE       0     0     0

errors: No known data errors
```


The status command tells us which drives are part of the pool and if they have had any errors

Pooled Storage

zpool **create** data **mirror** c1t0d0 c2t0d0

zpool **list** data

```
NAME     SIZE   ALLOC    FREE     CAP   DEDUP   HEALTH   ALTROOT
data     496G    164K    496G      0%   1.00x   ONLINE   -
```

zpool **add** data **mirror** c3t0d0 c4t0d0

zpool **list** data

```
NAME     SIZE   ALLOC    FREE     CAP   DEDUP   HEALTH   ALTROOT
data     992G    164K    992G      0%   1.00x   ONLINE   -
```

Pooled Storage

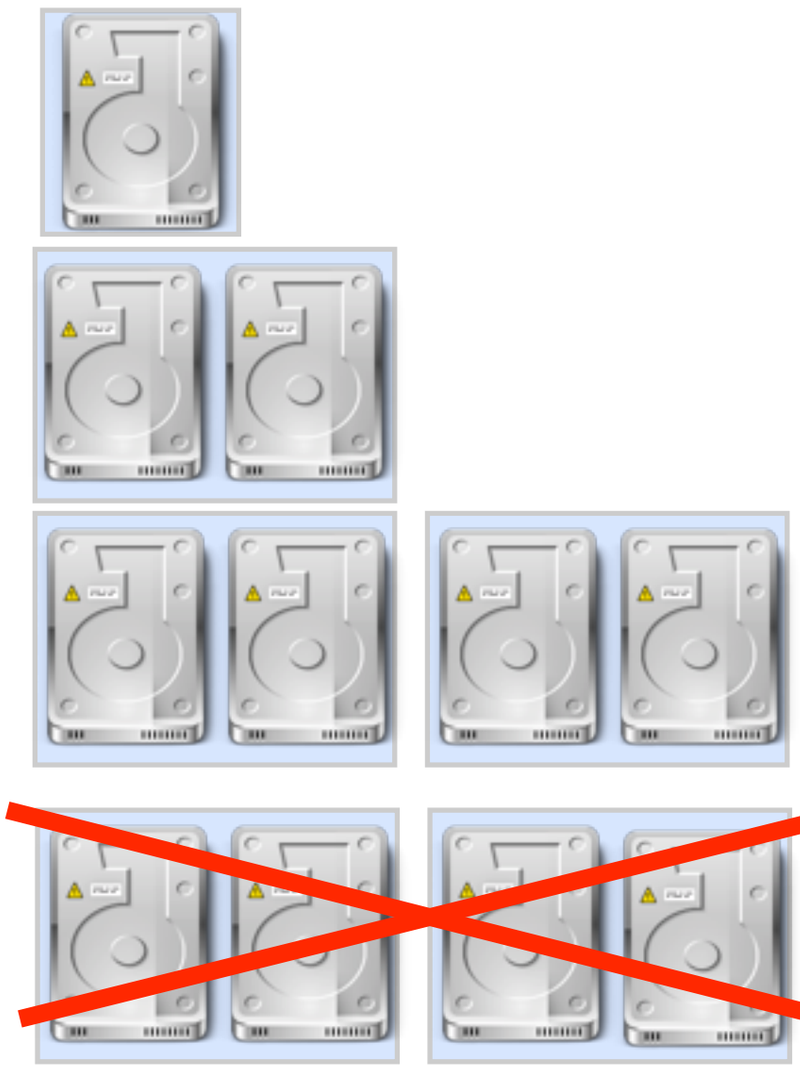
zpool **status**

```
pool: data
state: ONLINE
scrub: none requested
config:
        NAME          STATE     READ WRITE CKSUM
        data          ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            c1t0d0    ONLINE       0     0     0
            c2t0d0    ONLINE       0     0     0
          mirror-1    ONLINE       0     0     0
            c3t0d0    ONLINE       0     0     0
            c4t0d0    ONLINE       0     0     0

errors: No known data errors
```


The status command tells us which drives are part of the pool and if they have had any errors

Pool Evolution



zpool create data c1t0d0

zpool attach data c2t0d0

zpool add data mirror \
c3t0d0 c4t0d0

zpool destroy data

Here is an example where we start with a single disk.

Using the zpool attach command we can add a mirror to the original disk.

If we want more space or better performance, we can also add another pair of mirrored disks.

If we decide we are all done with the pool and want to remove all the data we can use the zpool destroy command to free up the disks

File Systems

FS manipulation done via **zfs** command

```
# zfs create data/web
# zfs set compression=on data/web
# zfs snapshot data/web@before_upgrade
# zfs create data/home
# zfs create data/home/slaney
```


zfs has several sub commands, the most common used are create, set, get, and list

ZFS Properties

All objects have props that change behavior

Properties are typically inherited

zfs set property=value $dataset

zfs get <$propname|all> $dataset

zfs **set mountpoint=/apache** data/web

All ZFS objects have properties that can control their behavior.

Most inherit from their parent.

ZFS Properties

```
# zfs get all data/web
```

NAME	PROPERTY	VALUE	SOURCE
data/web	type	filesystem	-
data/web	used	329M	-
data/web	available	992G	-
data/web	**compressratio**	**1.75x**	-
data/web	quota	none	default
data/web	**mountpoint**	**/apache**	**local**
data/web	checksum	on	default
data/web	**compression**	**on**	**local**
data/web	**atime**	**off**	**inherited from data**

This example shows some properties for one of our file systems.
The first few lines contain read-only properties and can't be set.
The compress ratio shows how much space is being saved by compression if it is active

We can see that the mountpoint is set to /apache and the source is local, meaning it is due to a setting we did explicitly to this file system

Lower we can see that the atime property is disabled and that the source shows it is inherited from a parent object

File Systems

```
# zfs list
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
data	131K	488G	23K	/data
data/home	21K	488G	21K	**/data/home**
data/home/slaney	21K	488G	21K	**/data/home/slaney**
data/web	21K	488G	21K	/apache

```
# zfs set mountpoint=/home data/home
# zfs list | grep home
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
data/home	21K	488G	21K	**/home**
data/home/slaney	21K	488G	21K	**/home/slaney**

ZFS File Systems

zfs set quota=10g data/home

zfs set compression=on data/home

zfs create data/postgres

zfs set reservation=50g data/postgres

zfs set atime=off data/home/slaney

Snapshots

A read-only copy of a filesystem

Can be used to roll-back to a previous state

```
# zfs snapshot $fs@$name

# zfs snapshot data/web@pre_upgrade

# zfs snapshot data/web@post_upgrade

# zfs snapshot data/home@`date +%F`
```

Managing Snapshots

zfs list -t snapshot

```
NAME                                  USED  AVAIL  REFER  MOUNTPOINT

data/web@pre_upgrade                  19K      -    26K          -

data/web@post_upgrade                  0       -   213M         -

data/home@2010-01-03                  18k      -    17M         -
```

zfs rollback data/web@post_upgrade

zfs destroy data/home@2010-01-03

Clones

A clone is a read-write copy based on a snapshot.

```
# zfs snapshot data/postgres@for_test

# zfs clone data/postgres@for_test  data/pgtest

# zfs set mountpoint=/postgres2 data/pg_test
```


Clones have a dependency on the snapshot they are built from.

Data Replication

1. Take a snapshot

   # zfs snapshot data/postgres@2009-12-31

2. Use zfs send/recieve

   # zfs send data/posgres@2009-12-31 | \

   ssh $remote_host zfs receive $dataset

3. Later .... use optional incremental update

   # zfs snapshot data/postgres@2010-01-02

   # zfs send -i data/postgres@2009-12-31 data/postgres@2010-01-02 | \

   ssh $remote_host zfs receive $dataset

The zfs send/receive stream can be piped over SSH or your transport of choice

You can obviously also use rsync

De-Duplication

PAIN WARNING: Still in heavy development as of 2010/01/12

To activate:

```
# zfs set dedup=on $dataset
```

To view space savings:

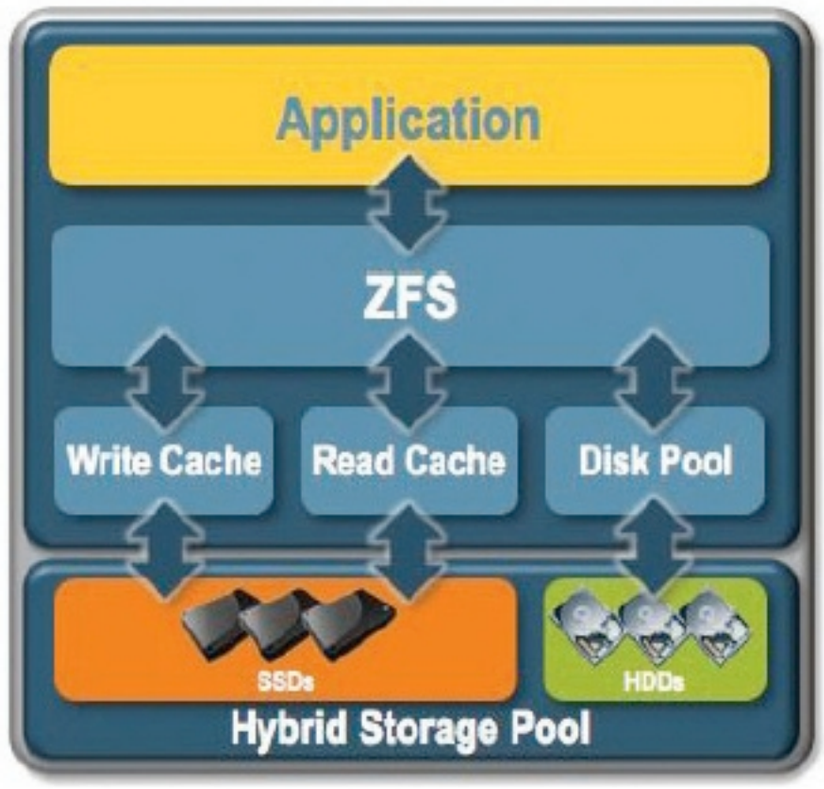
```
# zpool list $dataset
```

```
NAME    SIZE   ALLOC    FREE     CAP  DEDUP   HEALTH   ALTROOT
tpool   136G   33.7G    102G     24%  3.16x   ONLINE   -
```

Solid State Disk

Leverages SSD + standard drives

SSD can be used as either (or both)
- Write accelerator for intent log (ZIL)
- 2nd level cache for reads (L2ARC)



ZFS uses the best parts of hard disk drives (large capacity) and the best parts of SSDs (fast access speed)

A SSD can also be partitioned so that a small portion is used for the intent log and a larger section is used for L2ARC

ZFS in OpenSolaris

- Packaging system is ZFS aware
- OS upgrades works on a clone of /
- Adds extra grub entry
- Reboot into new OS rev
- If trouble, just reboot and pick previous rev

ZFS Resources

- **LMGTFY "ZFS best practice guide"**
- **LMGTFY "ZFS evil tuning guide"**
- **zfs-discuss @ opensolaris.org**
- **http://tinyurl.com/zfshome**