

Zfs

ZFS is a rethinking of the storage stack, combining traditional file systems as well as volume managers into one cohesive tool. ZFS has some specific terminology that sets it apart from more traditional storage systems, however it has a great set of features with a focus on usability for systems administrators.

ZFS Concepts

Virtual Devices

A VDEV is similar to a raid device presented by a RAID card, there are several different types of VDEV's that offer various advantages, including redundancy and speed. In general VDEV's offer better reliability and safety than a RAID card. It is discouraged to use a RAID setup with ZFS, as ZFS expects to directly manage the underlying disks.

Types of VDEV's * stripe (a single disk, no redundancy) * mirror (n-way mirrors supported) * raidz * raidz1 (1-disk parity, similar to RAID 5) * raidz2 (2-disk parity, similar to RAID 6) * raidz3 (3-disk parity, no RAID analog) * disk * file (not recommended for production due to another filesystem adding unnecessary layering)

Your data is striped across all the VDEV's present in your Storage Pool, so more VDEV's will increase your IOPS.

Storage Pools

ZFS uses Storage Pools as an abstraction over the lower level storage provider (VDEV), allow you to separate the user visible file system from the physical layout.

ZFS Dataset

ZFS datasets are analogous to traditional filesystems but with many more features. They provide many of ZFS's advantages. Datasets support Copy on Write snapshots, quota's, compression and deduplication.

Limits

One directory may contain up to 2^{48} files, up to 16 exabytes each. A single storage pool can contain up to 256 zettabytes (2^{78}) of space, and can be striped across 2^{64} devices. A single host can have 2^{64} storage pools. The limits are huge.

Commands

Storage Pools

Actions: * List * Status * Destroy * Get/Set properties

List zpools

```
## Create a raidz zpool
```

```
$ zpool create bucket raidz1 gpt/zfs0 gpt/zfs1 gpt/zfs2
```

```
## List ZPools
```

```
$ zpool list
```

```
NAME      SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
```

```
zroot    141G    106G    35.2G          -    43%    75%    1.00x    ONLINE    -
```

List detailed information about a specific zpool

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

NAME	SIZE	ALLOC	FREE	EXPANDSZ	FRAG	CAP	DEDUP	HEALTH	ALTROOT
zroot	141G	106G	35.2G	-	43%	75%	1.00x	ONLINE	-
gptid/c92a5ccf-a5bb-11e4-a77d-001b2172c655	141G	106G	35.2G		-	43%	75%		

Status of zpools

Get status information about zpools

```
$ zpool status
```

```
pool: zroot
```

```
state: ONLINE
```

```
scan: scrub repaired 0 in 2h51m with 0 errors on Thu Oct  1 07:08:31 2015
```

```
config:
```

NAME	STATE	READ	WRITE	CKSUM
zroot	ONLINE	0	0	0
gptid/c92a5ccf-a5bb-11e4-a77d-001b2172c655	ONLINE	0	0	0

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

Scrubbing a zpool to correct any errors

```
$ zpool scrub zroot
```

```
$ zpool status -v zroot
```

```
pool: zroot
```

```
state: ONLINE
```

```
scan: scrub in progress since Thu Oct 15 16:59:14 2015
```

```
39.1M scanned out of 106G at 1.45M/s, 20h47m to go
```

```
0 repaired, 0.04% done
```

```
config:
```

NAME	STATE	READ	WRITE	CKSUM
zroot	ONLINE	0	0	0
gptid/c92a5ccf-a5bb-11e4-a77d-001b2172c655	ONLINE	0	0	0

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

Properties of zpools

Getting properties from the pool properties can be user set or system provided.

```
$ zpool get all zroot
```

NAME	PROPERTY	VALUE	SOURCE
zroot	size	141G	-
zroot	capacity	75%	-
zroot	altroot	-	default
zroot	health	ONLINE	-
...			

Setting a zpool property

```
$ zpool set comment="Storage of mah stuff" zroot
```

```
$ zpool get comment
```

NAME	PROPERTY	VALUE	SOURCE
tank	comment	-	default

```
zroot comment Storage of mah stuff local
```

Remove zpool

```
$ zpool destroy test
```

Datasets

Actions: * Create * List * Rename * Delete * Get/Set properties

Create datasets

```
## Create dataset
```

```
$ zfs create tank/root/data
```

```
$ mount | grep data
```

```
tank/root/data on /data (zfs, local, nfsv4acls)
```

```
## Create child dataset
```

```
$ zfs create tank/root/data/stuff
```

```
$ mount | grep data
```

```
tank/root/data on /data (zfs, local, nfsv4acls)
```

```
tank/root/data/stuff on /data/stuff (zfs, local, nfsv4acls)
```

```
## Create Volume
```

```
$ zfs create -V zroot/win_vm
```

```
$ zfs list zroot/win_vm
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
tank/win_vm	4.13G	17.9G	64K	-

List datasets

```
## List all datasets
```

```
$ zfs list
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
zroot	106G	30.8G	144K	none
zroot/ROOT	18.5G	30.8G	144K	none
zroot/ROOT/10.1	8K	30.8G	9.63G	/
zroot/ROOT/default	18.5G	30.8G	11.2G	/
zroot/backup	5.23G	30.8G	144K	none
zroot/home	288K	30.8G	144K	none
...				

```
## List a specific dataset
```

```
$ zfs list zroot/home
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
zroot/home	288K	30.8G	144K	none

```
## List snapshots
```

```
$ zfs list -t snapshot
```

zroot@daily-2015-10-15	0	-	144K
zroot/ROOT@daily-2015-10-15	0	-	144K
zroot/ROOT/default@daily-2015-10-15	0	-	24.2G
zroot/tmp@daily-2015-10-15	124K	-	708M
zroot/usr@daily-2015-10-15	0	-	144K
zroot/home@daily-2015-10-15	0	-	11.9G
zroot/var@daily-2015-10-15	704K	-	1.42G

zroot/var/log@daily-2015-10-15	192K	-	828K
zroot/var/tmp@daily-2015-10-15	0	-	152K

Rename datasets

```
$ zfs rename tank/root/home tank/root/old_home
```

```
$ zfs rename tank/root/new_home tank/root/home
```

Delete dataset

Datasets cannot be deleted if they have any snapshots

```
zfs destroy tank/root/home
```

Get / set properties of a dataset

Get all properties

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

NAME	PROPERTY	VALUE	SOURCE
zroot/home	type	filesystem	-
zroot/home	creation	Mon Oct 20 14:44 2014	-
zroot/home	used	11.9G	-
zroot/home	available	94.1G	-
zroot/home	referenced	11.9G	-
zroot/home	mounted	yes	-
...			

Get property from dataset

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

NAME	PROPERTY	VALUE	SOURCE
zroot/home	compression	off	default

Set property on dataset

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

Get a set of properties from all datasets

```
$ zfs list -o name,quota,reservation
```

NAME	QUOTA	RESERV
zroot	none	none
zroot/ROOT	none	none
zroot/ROOT/default	none	none
zroot/tmp	none	none
zroot/usr	none	none
zroot/home	none	none
zroot/var	none	none
...		

Snapshots

ZFS snapshots are one of the things about zfs that are a really big deal

- The space they take up is equal to the difference in data between the filesystem and its snapshot
- Creation time is only seconds
- Recovery is as fast as you can write data.
- They are easy to automate.

Actions: * Create * Delete * Rename * Access snapshots * Send / Receive * Clone

Create snapshots

“`bash ## Create a snapshot of a single dataset zfs snapshot tank/home/sarlalian@now`

Create a snapshot of a dataset and its children

```
$ zfs snapshot -r tank/home@now $ zfs list -t snapshot NAME USED AVAIL REFER MOUNTPOINT
tank/home@now 0 - 26K - tank/home/sarlalian@now 0 - 259M - tank/home/alice@now 0 - 156M -
tank/home/bob@now 0 - 156M - ...
```

Destroy snapshots

```
## How to destroy a snapshot
```

```
$ zfs destroy tank/home/sarlalian@now
```

```
## Delete a snapshot on a parent dataset and its children
```

```
$ zfs destroy -r tank/home/sarlalian@now
```

Renaming Snapshots

```
## Rename a snapshot
```

```
$ zfs rename tank/home/sarlalian@now tank/home/sarlalian@today
```

```
$ zfs rename tank/home/sarlalian@now today
```

```
## zfs rename -r tank/home@now @yesterday
```

Accessing snapshots

```
## CD Into a snapshot directory
```

```
$ cd /home/.zfs/snapshot/
```

Sending and Receiving

```
## Backup a snapshot to a file
```

```
$ zfs send tank/home/sarlalian@now | gzip > backup_file.gz
```

```
## Send a snapshot to another dataset
```

```
$ zfs send tank/home/sarlalian@now | zfs recv backups/home/sarlalian
```

```
## Send a snapshot to a remote host
```

```
$ zfs send tank/home/sarlalian@now | ssh root@backup_server 'zfs recv tank/home/sarlalian'
```

```
## Send full dataset with snapshots to new host
```

```
$ zfs send -v -R tank/home@now | ssh root@backup_server 'zfs recv tank/home'
```

Cloning Snapshots

```
## Clone a snapshot
```

```
$ zfs clone tank/home/sarlalian@now tank/home/sarlalian_new
```

```
## Promoting the clone so it is no longer dependent on the snapshot
```

```
$ zfs promote tank/home/sarlalian_new
```

Putting it all together

This following a script utilizing FreeBSD, jails and ZFS to automate provisioning a clean copy of a mysql staging database from a live replication slave.

```
#!/bin/sh
```

```
echo "==== Stopping the staging database server ===="
```

```

jail -r staging

echo "==== Cleaning up existing staging server and snapshot ===="
zfs destroy -r zroot/jails/staging
zfs destroy zroot/jails/slave@staging

echo "==== Quiescing the slave database ===="
echo "FLUSH TABLES WITH READ LOCK;" | /usr/local/bin/mysql -u root -pmyrootpassword -h slave

echo "==== Snapshotting the slave db filesystem as zroot/jails/slave@staging ===="
zfs snapshot zroot/jails/slave@staging

echo "==== Starting the slave database server ===="
jail -c slave

echo "==== Cloning the slave snapshot to the staging server ===="
zfs clone zroot/jails/slave@staging zroot/jails/staging

echo "==== Installing the staging mysql config ===="
mv /jails/staging/usr/local/etc/my.cnf /jails/staging/usr/local/etc/my.cnf.slave
cp /jails/staging/usr/local/etc/my.cnf.staging /jails/staging/usr/local/etc/my.cnf

echo "==== Setting up the staging rc.conf file ===="
mv /jails/staging/etc/rc.conf.local /jails/staging/etc/rc.conf.slave
mv /jails/staging/etc/rc.conf.staging /jails/staging/etc/rc.conf.local

echo "==== Starting the staging db server ===="
jail -c staging

echo "==== Make sthe staging database not pull from the master ===="
echo "STOP SLAVE;" | /usr/local/bin/mysql -u root -pmyrootpassword -h staging
echo "RESET SLAVE;" | /usr/local/bin/mysql -u root -pmyrootpassword -h staging

```

Additional Reading

- BSDNow's Crash Course on ZFS
- FreeBSD Handbook on ZFS
- BSDNow's Crash Course on ZFS
- Oracle's Tuning Guide
- OpenZFS Tuning Guide
- FreeBSD ZFS Tuning Guide