ZFS Quick Guide

The ZFS file system is used widely on BSD, and is coming into more use on Linux. Following are some of my notes on it.

Initial Setup (FreeBSD)

Start ZFS service

FreeBSD comes with the ZFS service installed, but not active. We need to start the service, and also tell the system to start it when the system reboots.

```
echo 'zfs_enable="YES"' >> /etc/rc.conf
service zfs start
```

Create a zpool

Now that we have ZFS running, we'll create a zpool, the basic container for all of our stuff. In this case, I want to use the raidz2 for redundancy (two drives are used for checksumming). Since I don't know the correct names for everything, I'll use geom to find them.

Warning: The following will find all drives on the system, including your boot drives, so remove the boot drives before proceeding with the zpool creation.

Note: I've shown three ways to find the drives on the system. The first three commands give redundant information. If you have smartctl on your system, that is probably the easiest, since it has a scan function built in, but geom is FreeBSD's way to do it.

```
# find the drives on the system. Choose ONE of the following
geom disk list | grep Geom | rev | cut -d' ' -f1 | rev | sort
egrep 'da[0-9]|cd[0-9]' /var/run/dmesg.boot | sort
smartctl --scan | cut -d' ' -f1
# we want RAID-6, name it storage, and us /dev/da0 through 7
zpool create -f storage raidz2 /dev/da[01234567]
```

You can add extra functionality by creating *intent*, *dedup* and *cache* vdev's at the same time. Following example shows adding a vdev (mirror) to a pool.

```
zpool create -f -m /storage storage raidz2 da4 da5 da6 \
                      da7 da8 da9 dedup mirror da2 da3
```

This will create a pool named storage, mounted (-m) at /storage, forced to ignore most drive errors. The pool will be a raidz2 (aka RAID 6) with 6 drives (4-9), and have a dedup vdev consisting of a mirror from da2 and 3.

Set dataset defaults

Your new dataset may not have the default values you want. It is simple enough to set defaults for all child datasets at this point, then any datasets/volumes you create will default to the following values.

This is the way I have most set up. Modify it for your own use. Pay particular attention to the dedup=on and compress=gzip-9.

- dedup will use more memory and cpu, but will reduce the amount of disk space required if your data contains duplicates (think 10 Devuan Daedalus installations taking up the same amount of space as 1 for the common blocks)
- compress is fine, but gzip-9 will tear up your processor. I use gzip-9 for backup servers, but for servers actually serving all the time (like iSCSI or NFS), I use ztsd or even lz4 to decrease my server load. See the excellent article at https://www.zfshandbook.com/docs/advanced-zfs/compression/ for a comparison.
- volmode=full should really be the default (grumble). It simply takes the block of space and exports it for iSCSI.

```
zfs set atime=off dedup=on compress=gzip-9 volmode=full storage
```

Create a Dataset

Datasets are just allocations in the file system for specific purposes. Unlike traditional disks and volume managers, space in ZFS is not preallocated. You can create an allocation with various options (using the -o flag). Anything not set will be inherited from the parent.

```
zfs create -o quota=150G -o atime=off -o compression=lz4
storage/backups/client1/server.client1
# or
zfs create -o quota=150G -o atime=off -o compression=on
storage/backups/client1/server.client1
```

This allocates 150G of space in the zpool storage, turning off atime, but setting compression to lz4. It will be mounted wherever client1 is mounted. Everything else is inherited from client1, which in turn inherits from backup, which inherits from storage.

Use a ZFS Volume for swap space

It is perfectly fine to set up swap on a ZFS volume, but we do want to turn of checksumming. Here, we'll create a 2G ZFS volume named swap (in storage, so storage/swap), add an entry to fstab, and turn it on.

```
zfs create -V 2G -o checksum=off storage/swap
echo '/dev/zvol/storage/swap none swap sw 0 0' >> /etc/fstab
swapon /dev/zvol/storage/swap
```

NOTE: you can add space to this swap area in real time, if you do it at a time when the swap is not

necessary to the continued operating of the machine. Simply turn off swap, increase the volume size, then turn swap back on again.

```
# turn off swap
swapoff /dev/zvol/storage/swap
# increase volume size to 4G
zfs set volsize=4096M storage/swap
# check that it worked
zfs get volsize,reservation storage/swap
# turn swap back on (could also use swapon /dev/zvol/storage/swap, but I'm
lazy)
swapon -aL
```

Using a file for swap space

Instead of using a swap partition or zvol, we can simply use a file. In this case, we can add swap space by deleting/recreating the swap file (after turning swap off), but a simpler way if you need more swap space is to simply add a second swap file.

Following code creates an 8G swap file. Note that after reading https://www.cyberciti.biz/faq/create-a-freebsd-swap-file/, I modified my old way of doing this.

```
# create an 8G swap file
dd if=/dev/zero of=/swap bs=1G count=8
# set permissions
chmod 0600 /swap
# look for an unused md device (ie, not listed)
mdconfig -lv
cp /etc/fstab /etc/fstab.save
# edit /etc/fstab and add the following line, using the correct md##
joe /etc/fstab
md42    none    swap    sw,file=/swap    0    0
# save your file
# turn on swap
swapon -aq
# look at swap information
swapinfo -k
```

iSCSI considerations

On FreeBSD, the iSCSI config is /etc/ctl.conf, and the service is ctld

```
service ctld reload # reread iscsi exports on FreeBSD
```

iSCSI generally uses volumes which are then exported by the target. I have found that it is useful, from a management perspective, to place them under a dataset strictly for them, since I back up iSCSI volumes on a different timeline than I do other stuff.

A lot of time, my iSCSI targets are just the operating system, and maybe log files. If I also use it for dynamic data (e-mail repo, databases, web sites), I back up through a different means, though a lot of time I use NFS to store this.

I generally create something like storage/iscsi, then create the volumes under that.

```
zfs create storage/iscsi
zfs create -V 10G storage/iscsi/server1.disk0
zfs create -V 10G storage/iscsi/server1.disk1
```

This allows me to snapshot my iSCSI volumes with one command, then send them with one:

```
zfs snapshot -r storage/iscsi@2022-04-20_22-30
zfs send -Ri storage/iscsi@2022-04-20_22-30 | ssh someplace 'zfs recv -Fduv storage/backups/iscsi'
```

Your iSCSI paths will be the same as you would expect; in this case, stored under /dev/zvol/storage/iscsi

Note: If you have an existing system and want to change, zfs rename is your friend.

1. On initiator
    1. Shut down access to volume on the iSCSI initiator
    2. detach from iSCSI initiator (not sure if this is required)
2. On iSCSI target
    1. Rename the volume

        ```
        zfs rename storage/volumename storage/iscsi/volumename
        ```

    2. Edit the config to point to new location (**/etc/ctl.conf** on FreeBSD)
    3. reload iscsi service

        ```
        service ctld reload # on FreeBSD
        ```

3. On initiator
    1. rescan target
    2. allow access to volume

Getting and setting properties

ZFS has properties that allow you to modify the way the file system works. The following example sets a quota of 100 Megabytes for storage/varlog so our logs do not fill up the system.

Note the last line showing you can set multiple properties at one time by separating the properties by spaces

```
zfs get all storage/varlog
zfs get quota storage/varlog
zfs set quota=100M storage/varlog
zfs set exec=off checksum=off storage/varlog
```

If you want to return to the default settings, where a property is inherited from the container, use the following code

```
zfs inherit -r quota storage/varlog
```

Deduplication

If you want to see what the effect of deduplication would be, you can use zdb to calculate it. **NOTE** This will take a long time as it must open every block on the disk and build a deduplication table from it. However, it is well worth doing if you are considering adding the additional complexity of deduplication.

```
zdb -S -U /path/to/some/cache/file poolname
```

You can find the name of the cache file (if configured) with

```
zpool get cachefile poolname
```

Dedup requires 320 bytes per block of memory, so you can take the output of the above command, multiply by 320, and see the amount of RAM which will be required to run dedup. NOTE: This can grow as more unique blocks are allocated.

Find differences between two snapshots

So, you have zfs running, and you have some automated process running every morning at 4am. Now, you want to see what changed in the 24 hour period.

```
# get a list of snapshots so we know exactly what to ask for
zfs list -r -t snapshot storage/someplace
# find the differences. First one should be the most recent
zfs diff storage/someplace/subdir@20181209_041339
storage/someplace/subdir@20181209_041339
# same thing, but looking only for one subdir named joe
zfs diff storage/someplace/subdir@20181209_041339
storage/someplace/subdir@20181208_041339 | grep '/joe/'
```

output is similar to Subversion, ie a 'M' indicates it was modified, a - indicates it was removed, and a + indicates it was added. Note that your snapshots do not have to be consecutive; you can look at the diff between your oldest and newest snapshot, or even your current copy, as this shows:

```
# compare snapshot taken 20181209_041339 with the current copy
zfs diff storage/someplace/subdir@20181209_041339 storage/someplace/subdir
```

Useful commands

- Create a snapshot of **path/to/base** named **snapshotname**

```
zfs snapshot path/to/base@snapshotname
```

- List all snapshots in a particular tree. gives USED (space used by snapshot) and REFER (data referred to in original set)

```
zfs list -r -t snapshot /storage/varlog
```

- Remove an existing snapshot (use above command to find the correct name)

```
zfs destroy -r tank/storage/varlog/@20181026_054020
```

- Get a nice list of stats on every dataset in a tree (does the whole tree). Gives AVAIL, ie amount of space available, USED, USEDSNAP (space used by snapshots), USEDDS (space used by the dataset exclusive of snapshots, ie actual data), USEDREFRESERV (whatever that is) and USEDCHILD (used by children of the dataset).

```
zfs list -o space -r storage/varlog
```

References

- https://www.freebsd.org/doc/handbook/zfs.html
- https://www.freebsd.org/doc/handbook/zfs-quickstart.html
- https://forums.freebsd.org/threads/is-swap-on-zfs-safe.27855/
- https://www.freebsd.org/doc/handbook/adding-swap-space.html
- https://www.cyberciti.biz/faq/freebsd-hard-disk-information/
- https://www.thegeekdiary.com/how-to-find-the-space-consumed-by-zfs-snapshots/
- https://docs.oracle.com/cd/E23824_01/html/E24456/filesystem-6.html
- https://wordpress.morningside.edu/meyersh/2009/11/30/zfs-deduplication/
- https://linuxhint.com/zfs-deduplication/
- https://www.cyberciti.biz/faq/create-a-freebsd-swap-file/

From:
https://kb.unixservertech.com/ - **Unix Server Tech Knowledge Base**

Permanent link:
https://kb.unixservertech.com/quickreference/zfs

Last update: **2025/03/22 04:39**