Things To Like About ZFS

Mark G.

June 22, 2024

This presentation gives an overview of the Zettabyte File System (ZFS). The hosting platforms for this demonstration are FreeBSD on a Minisforum mini PC and an implementation of ZFS on a Raspberry Pi 4. Topics covered are an introduction to ZFS, how to create a ZFS storage system, some examples of ZFS in use, and descriptions of some of the more useful features of ZFS.

What is a zettabyte? Wikipedia provides a nice chart at the page Binary prefix[1]:

Prefixes for decimal and binary multiples

Decimal				Binary				
Value		SI		Value		IEC		JEDEC
1000	10^3	k	kilo	1024	2^{10}	Ki	kibi	K kilo
1000^2	10^6	M	mega	1024^2	2^{20}	Mi	mebi	M mega
1000^3	10^9	G	giga	1024^3	2^{30}	Gi	gibi	G giga
1000^4	10^{12}	T	tera	1024^4	2^{40}	Ti	tebi	T tera
1000^5	10^{15}	P	peta	1024^5	2^{50}	Pi	pebi	—
1000^6	10^{18}	E	exa	1024^6	2^{60}	Ei	exbi	—
1000^7	10^{21}	Z	zetta	1024^7	2^{70}	Zi	zebi	—
1000^8	10^{24}	Y	yotta	1024^8	2^{80}	Yi	yobi	—
1000^9	10^{27}	R	ronna				—	
1000^{10}	10^{30}	Q	quetta				—	

V • T • E

Figure 1: Chart of number multipliers/prefixes

[1] https://en.wikipedia.org/wiki/Binary_prefix

Contents

List of Figures

1 Introduction

The Zettabyte File System has a Wikipedia Page, which is well worth reading.

Important notes about ZFS:

1. Pooled storage: ZFS combines physical devices, and their associated partitions if used, with data and file storage (datasets), giving ZFS an all encompassing view of storage and its management. These pools are shared by the datasets (file systems) and eliminate the fixed partition size / mounted file system problems of the past. You can still run out of space, of course, but pools can be expanded without affecting individual datasets.

2. Data integrity: ZFS has built in check-summing of files, which ensures data integrity. Redundancy is achieved via multiple physical devices or multiple on-disk copies of files.

3. Performance: ZFS can use aggressive caching via the Adaptive Replacement Cache (ARC) which is stored in system memory. It can also leverages fast-disk based read caches, and write caches.

1.1 How Much Can it Hold?

The designed maximum capacity of ZFS, given sufficient physical devices, is 2^{128} bytes. That's an IPv6 address space worth of storage! I am unsure why this amount is not 10^{21} based on the Zetta- prefix it is named after.

1.2 It Needs Lots of Memory

There exists a caching subsystem called the Adaptive Replacement Cache (ARC) which gobbles up memory, so using ZFS on anything less than 8GB will work, but is not recommended. Memory use by ARC can be administratively limited, but memory is important so the more, the better.

1.3 Data Storage Members (Disks / Partitions)

Usually, the lowest level of any file system was the physical device, be it the entire hard drive, or a partition thereon, or even a hardware/software RAID configuration. This isn't the case with a ZFS dataset (file system equivalent) since a dataset resides within a ZFS pool.

Partitions or even full disks are used to build a pool's total storage. These partitions or full disks are referred to as `vdevs` (virtual devices) in most ZFS documentation. In some cases, `vdevs` can be encrypted before being added to a pool, thereby allowing for

a fully encrypted pool, on top of which sit the datasets.

1.4 Pools

Pools are a logical collection of storage devices called `vdevs` that provide a large amount (pool) of storage.

Pools are created using the `zpool create` command. and can be configured in various ways:

1. Single disk: Can be useful for USB attached media, but does not provide redundancy, in general, unless '`copies=2`' is set on a dataset. I use these as backups.

   The command below creates a single device pool using the full USB disk.

   ```
   # zpool create usbpool /dev/da0
   ```

2. Mirror n-way: Two or more `vdevs` can be used to create a mirrored pool, and this is the most common form of redundancy. All my server systems use at least 2-way mirrors for the OS and supporting services.

   The command below creates a 2-way system mirror pool (`syspool`) using two full NVMe drives. One drive can fail and the system will still be functional.

   ```
   # zpool create syspool mirror /dev/nvd0 /dev/nvd1
   ```

3. Raid-Z: This requires three (3) or more disks, but does provide more storage than a mirrored pool. There are three types `raidz=raidz1`, `raidz2` and `raidz3`. The z-number corresponds to the number of parity `vdevs` (disks) used. Of course, higher raid levels have a minimum number of `vdevs`. I use these as large pools for automated network backups.

   The command below creates a `raidz2` backup pool (`buppool`) using five full SATA drives. Two of the drives can fail before the pool itself fails.

   ```
   # zpool create buppool raidz2 /dev/ada0 /dev/ada1 /dev/ada2 /dev/ada3 /dev/ada4
   ```

A pool is often mounted on a mountpoint anchored at the root of the file system with a directory of the same name as the pool. That is, `buppool` would be defaulted to mount on `/buppool`. This mountpoint is also inherited as the base of any datasets created on the pool. This is not the most useful way to mount a pool's datasets. We'll see a better example when we look at how a FreeBSD installation sets up mount points and datasets.

1.5 Datasets

A dataset is the closest equivalent to a typical file system. When creating a dataset, the pool it lives on is specified as the first part of the dataset's path. For example:

```
# zfs create buppool/backup1
```

We use the `buppool` pool and create a dataset on it named `backup1`. We can put a copy of our important data into this dataset. We can create `backup2` and so on as needed. They will have mount points relative to `/buppool` and use storage space from the pool in which they were created.

The nice thing about datasets is their many configurable properties. Some of these are:

1. Compression: A dataset can be setup to use compression by default, such as `LZ4` or `gzip`.

2. Access time `atime` updates: When files are accessed, a property called `atime` is often updated. This is usually undesirable in most use cases, as it causes file changes in the dataset, which are propagated to snapshots, thereby increasing storage space requirements. Incremental backup strategies can also be impacted.

   Therefore, it is common to set `noatime` on a dataset, which inhibits this access time update.

3. `exec` or `noexec`: Determines if files can be set as executable on the dataset, a common Unix file system attribute.

4. `setuid` or `nosetuid`: Another common Unix file system attribute which determines if an executable can be run as a specific user, usually `root`.

A dataset is created directly on a pool and usually inherits the pool's mountpoint as the leading part of the dataset's final mountpoint. Sometimes this is not what you want and you need to create the dataset using some options like so:

```
# zfs create -o mountpoint=/data buppool/data
```

The above command creates a new dataset that is mounted on `/data`, rather than the inherited value of `/buppool/data`.

1.5.1 Datasets are Hierarchical

Datasets inherit some of their properties from their parent. In some cases this is the base pool dataset, but in most useful cases it is a dataset created to specify a set of features that all child datasets will share. A good example of this would be the typical `/usr` directory structure:

```
# zfs create -o mountpoint=/usr -o canmount=off syspool/usr
```

Note that `canmount` is not an inheritable property, but acts in this case to help with the propagation of `/usr` as a base mount point for further child datasets:

```
# zfs create syspool/usr/local
# zfs create syspool/usr/src
# zfs create syspool/usr/home
```

Each of these datasets now get mounted at `/usr/local`, `/usr/src`, and `/usr/home`. They can also be individually managed and given separate sets of properties if desired.

1.6 ZFS Volumes (zvols)

Apart from datasets, there are also volumes called `zvols` (not covered here), which act like disk devices. This allows for other files systems, like UFS, to be created on a ZFS volume. These volumes benefit from snapshots and other ZFS features.

2 Hardware and Apparatus

Enough of the boring stuff, let's see some ZFS in action. Technologies used in this demonstration are:

- Miniforum UM690Pro Mini PC 32GB, dual NVMe (FreeBSD)
- Raspberry Pi 4B 8GB RAM (Raspberry Pi OS)
- 2 USB attached M.2 NVMe drives
- 2 USB attached M.2 SATA drives
- Network switch and cabling
- USB HDMI capture device
- HDMI switch / splitter and cabling

2.1 ZFS On Root (FreeBSD)

First, I'd like to introduce the FreeBSD documentation on ZFS [2]. It is excellent.

We'll do a quick install of FreeBSD to show one method of using ZFS as a root file system on a 2-way disk mirror. In this document, some steps of the installation are omitted, since they have no relevance to ZFS.

Choose to setup FreeBSD using Auto (ZFS):

[2] https://docs.freebsd.org/en/books/handbook/zfs/

Figure 2: FreeBSD Installation - Partition Layout Selection

We are greeted with the initial un-configured ZFS configuration screen:

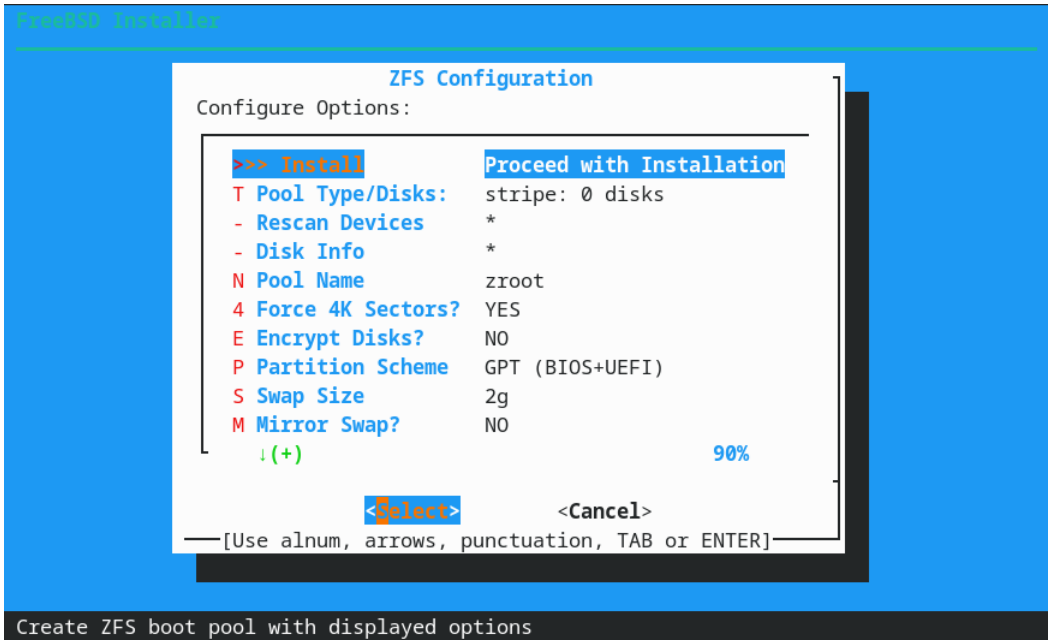

Figure 3: FreeBSD Installation - Initial ZFS Configuration

Press `T` to choose the pool type option and hit the `ENTER` key to initiate the action:

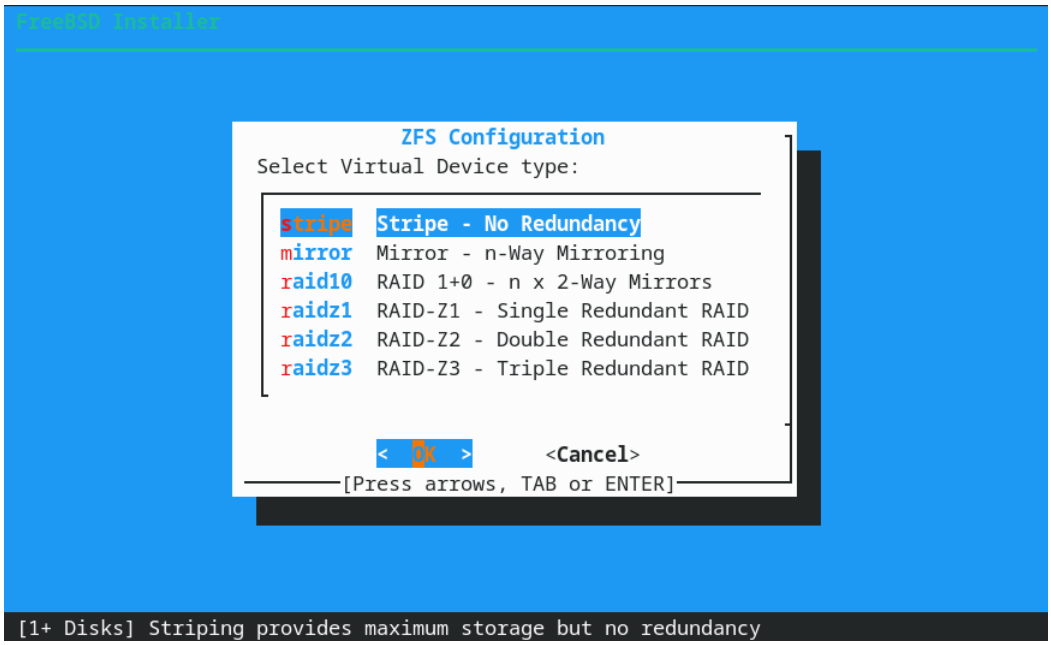

Figure 4: FreeBSD Installation - List of pool types

Press `m` to choose the mirror option and hit the `ENTER` key to initiate the action:

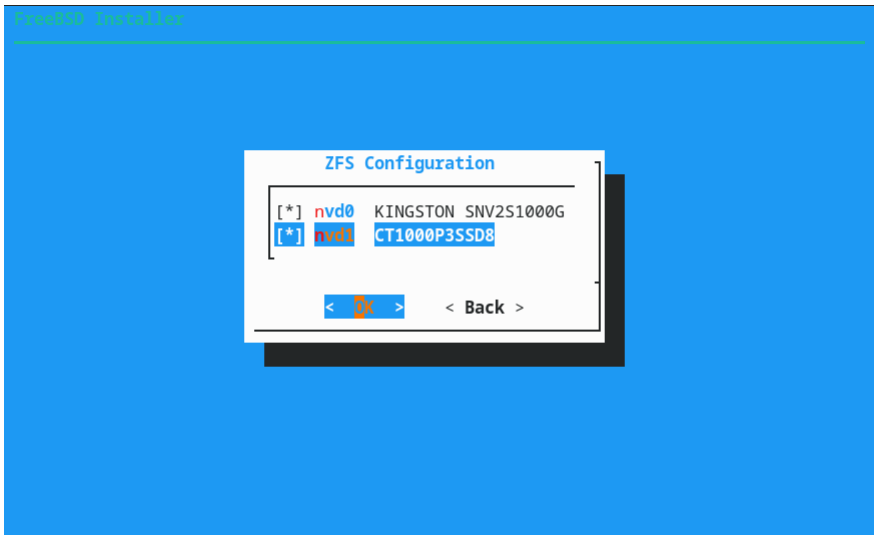

Figure 5: FreeBSD Installation - Select drives for mirror

We've selected both drives for a two way mirror.

We are returned to the ZFS Configuration screen, where we still have some options to set.

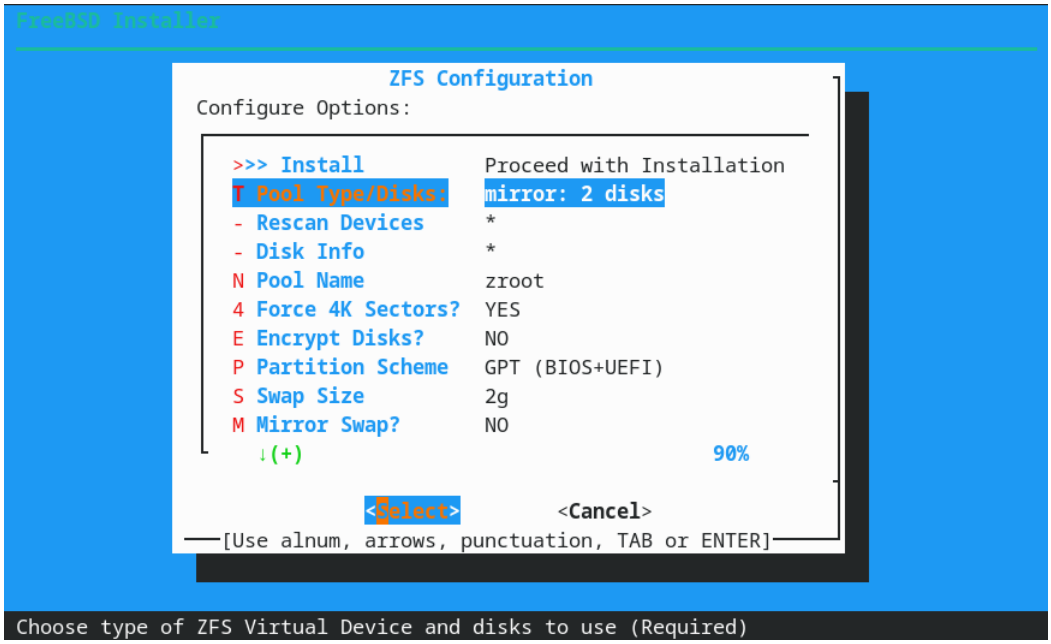

Figure 6: FreeBSD Installation - Return to ZFS Configuration screen

For the rest of the options, I usually setup a FreeBSD system as follows:

1. Use `N` to set the pool name so it is unique to the system. This helps with backups and keeping track of various systems.

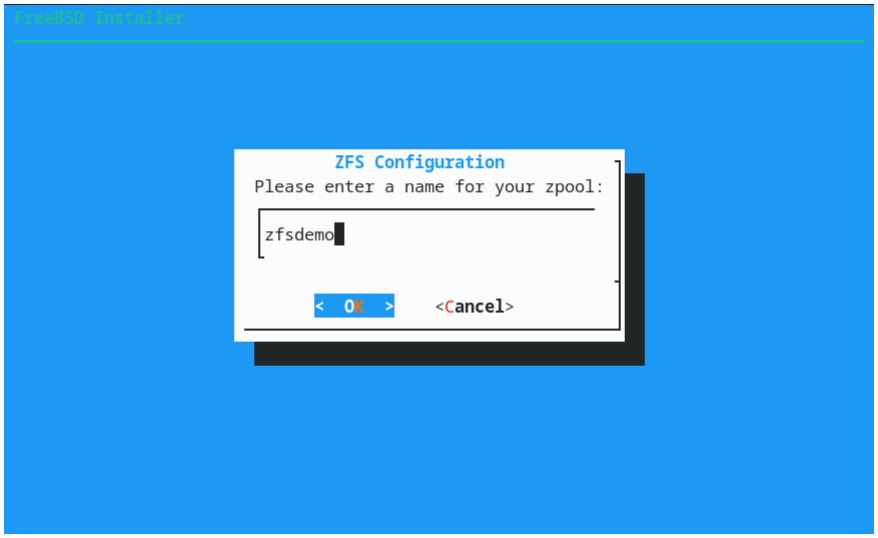

Figure 7: FreeBSD Installation - Enter a new pool name

2. Force 4K sectors is always `YES`.

3. Use `E` to select Encrypt disks and press the `SPACE` bar to toggle the setting to `YES`. We will be asked for a passphrase in a moment.

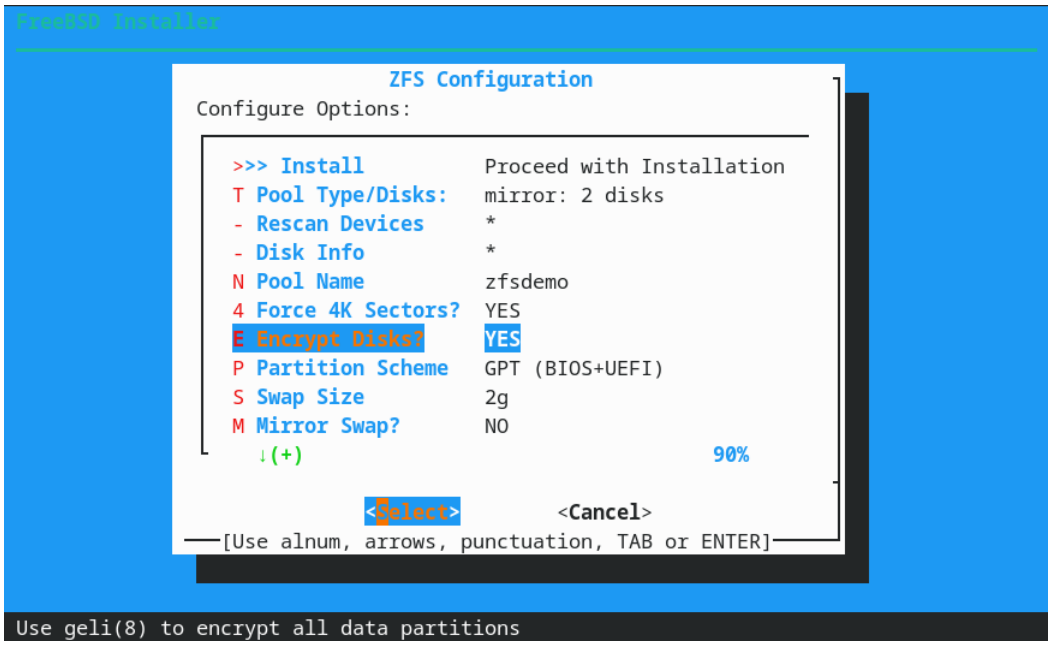

Figure 8: FreeBSD Installation - Encrypt Disks set to YES

4. Partition scheme is usually fine to leave at GPT (BIOS and UEFI) compatible.

5. Use `S` to select Swap Size and press `ENTER` to set a swap size. I usually set a minimum of 4GB per drive in the mirror.

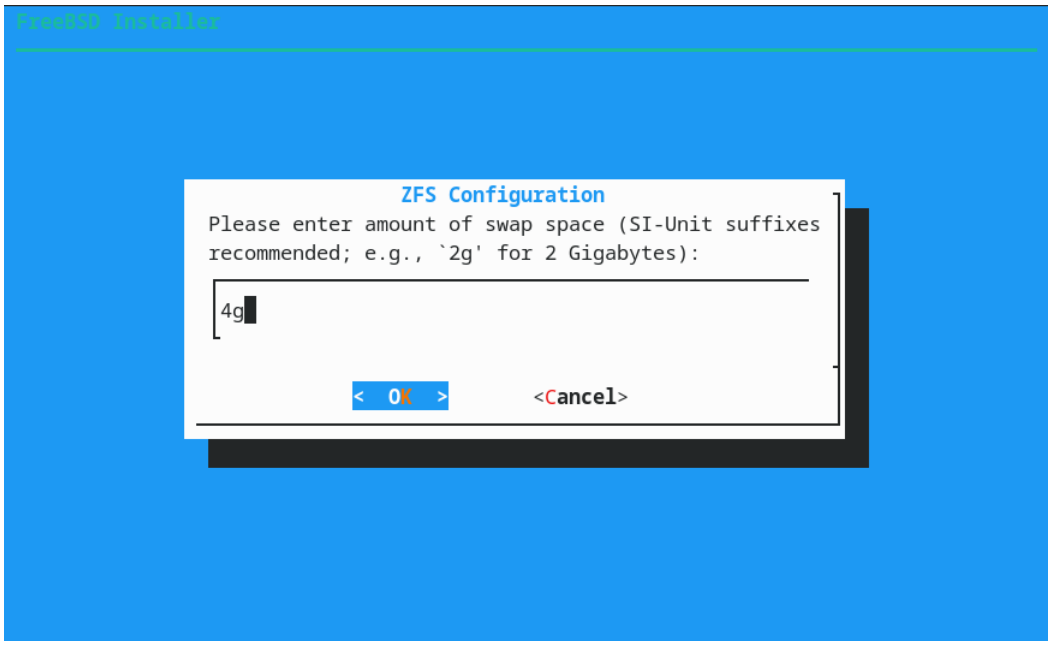

Figure 9: FreeBSD Installation - Set swap size to 4G per drive

6. Use `M` to select the Mirror Swap option and press the `SPACE` bar until `NO` is selected. No need to mirror swap.

7. Not all options are shown on the main ZFS Configuration screen, since it is small in our example, so the down arrow key will bring the remaining option(s) into view.

8. Use `W` to select the Encrypt Swap option and press the `SPACE` bar until `YES` is selected. It's a good idea to keep temporary memory protected on the disk, since there could be passwords stored in application memory.

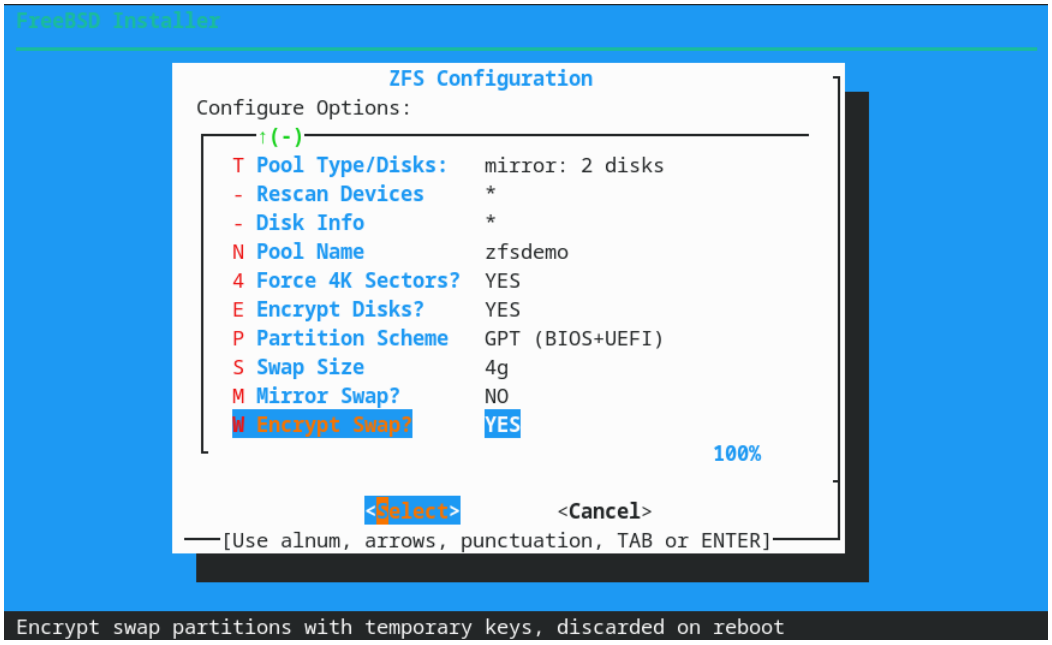

Figure 10: FreeBSD Installation - Set swap to be encrypted

That concludes the ZFS Configuration, so we press the up arrow until we reach the `Install - Proceed with Installation` option and press `ENTER` to select it.

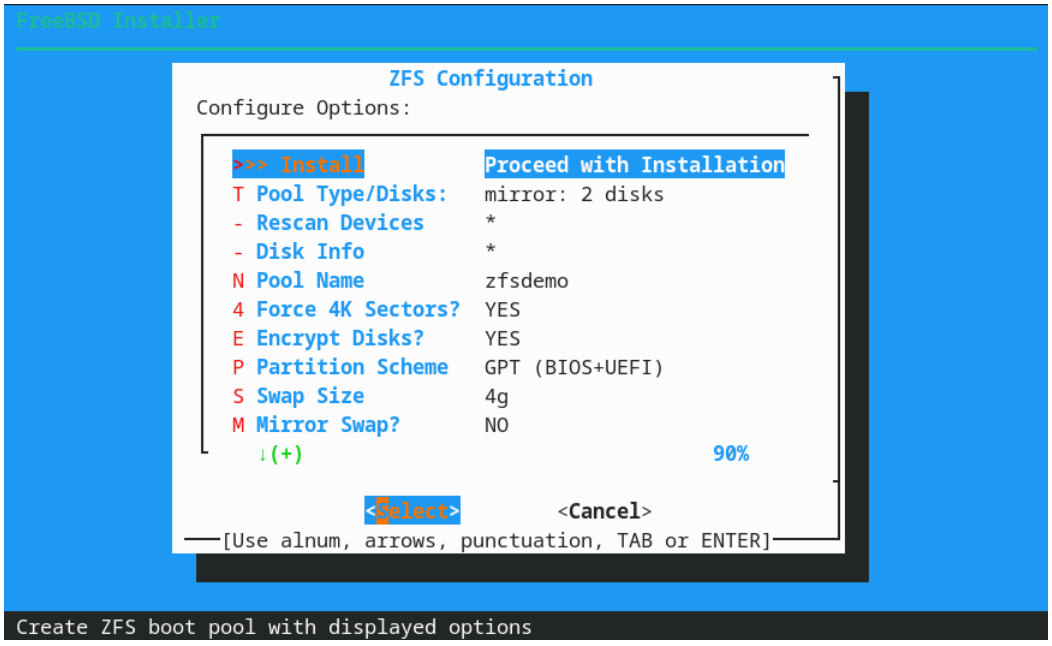

Figure 11: FreeBSD Installation - Proceed with Installation

The next step is to begin setting up the drives, so we are warned that proceeding will destroy our disks:

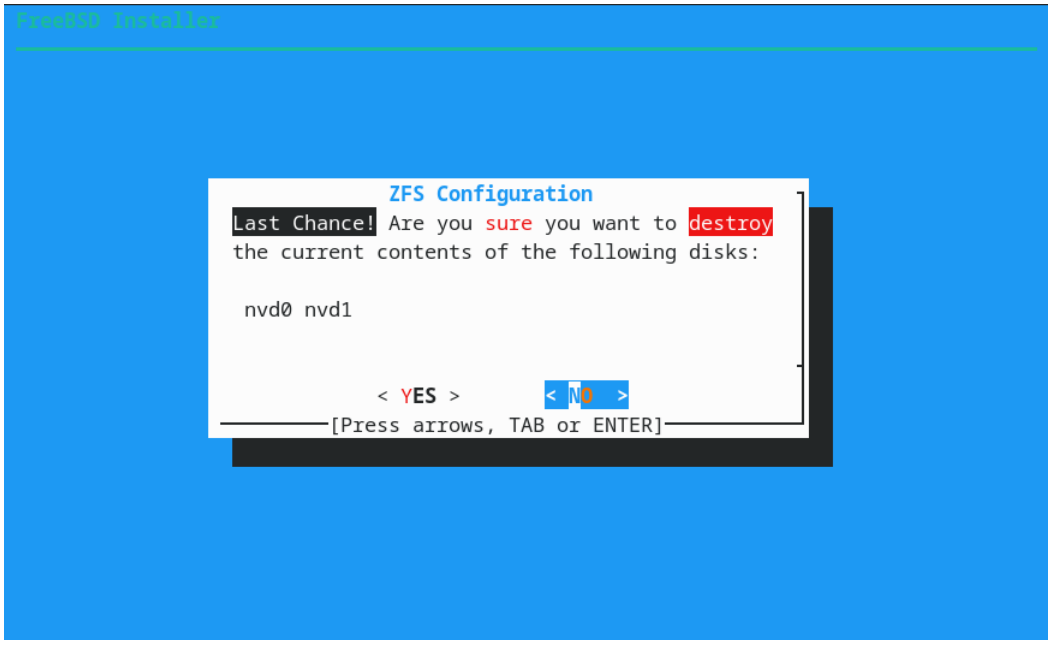

Figure 12: FreeBSD Installation - Disk destruction warning

The rest of the installation is just standard FreeBSD, so we will look at the results after `bsdinstall` has completed. As shown in appendix 4.1, there is much useful information to be extracted from the installation log found at `/var/log/bsdinstall_log`, but we'll just look at the currently running system's ZFS configuration.

2.1.1 Partitions

It is useful to examine how the `bsdinstall` program partitioned the drives and used them as `vdevs`. The two drives are named `nvd0` and `nvd1`. The `gpart show -p` command displays the full partition table on each drive with the full partition name (i.e. `nvd0p4`) listed.

```
mv@zfsdemo:~ % gpart show -p
=>         40  1953525088    nvd0  GPT  (932G)
           40      532480  nvd0p1  efi  (260M)
       532520        1024  nvd0p2  freebsd-boot  (512K)
       533544         984          - free -  (492K)
       534528     8388608  nvd0p3  freebsd-swap  (4.0G)
      8923136  1944600576  nvd0p4  freebsd-zfs  (927G)
   1953523712        1416          - free -  (708K)
```

```
=>        40  1953525088    nvd1  GPT  (932G)
          40      532480  nvd1p1  efi  (260M)
      532520        1024  nvd1p2  freebsd-boot  (512K)
      533544         984          - free -  (492K)
      534528     8388608  nvd1p3  freebsd-swap  (4.0G)
     8923136  1944600576  nvd1p4  freebsd-zfs  (927G)
  1953523712        1416          - free -  (708K)
```

The partition names can be seen in the `/dev` directory:

```
mv@zfsdemo:~ % ls /dev/nvd*
/dev/nvd0       /dev/nvd0p3     /dev/nvd0p4.eli /dev/nvd1p2     /dev/nvd1p4
/dev/nvd0p1     /dev/nvd0p3.eli /dev/nvd1       /dev/nvd1p3     /dev/nvd1p4.eli
/dev/nvd0p2     /dev/nvd0p4     /dev/nvd1p1     /dev/nvd1p3.eli
```

The encryption can be seen on the devices with the `.eli` extension. Indeed, `/dev/nvd0p3.eli` is one of our encrypted swap devices, shown here in the `/etc/fstab` file:

```
mv@zfsdemo:~ % cat /etc/fstab
# Device                Mountpoint      FStype  Options         Dump    Pass#
/dev/gpt/efiboot0       /boot/efi       msdosfs rw              2       2
/dev/nvd0p3.eli         none            swap    sw              0       0
/dev/nvd1p3.eli         none            swap    sw              0       0
```

The `/dev/nvd0p4.eli` device is part of the encrypted ZFS mirror pool as shown in our `zpool status` command:

```
mv@zfsdemo:~ % zpool status
  pool: fpool
 state: ONLINE
config:

        NAME            STATE     READ WRITE CKSUM
        fpool           ONLINE       0     0     0
          mirror-0      ONLINE       0     0     0
            nvd0p4.eli  ONLINE       0     0     0
            nvd1p4.eli  ONLINE       0     0     0

errors: No known data errors
```

Note the discrepancy in the pool name (`fpool` rather than `zfsdemo`), as I recorded the setup process after the fact.

2.1.2 ZFS Datasets

We can see the ZFS datasets chosen by the installer in several ways. The first is to look at the `zpool history` command:

```
root@zfsdemo:~ # zpool history
History for 'fpool':
2024-06-14.19:49:25 zpool create -o altroot=/mnt -O compress=lz4 -O atime=off
 -m none -f fpool mirror nvd0p4.eli nvd1p4.eli

2024-06-14.19:49:25 zfs create -o mountpoint=none fpool/ROOT
2024-06-14.19:49:25 zfs create -o mountpoint=/ fpool/ROOT/default
2024-06-14.19:49:25 zfs create -o mountpoint=/tmp -o exec=on -o setuid=off fpool/tmp
2024-06-14.19:49:25 zfs create -o mountpoint=/usr -o canmount=off fpool/usr
2024-06-14.19:49:25 zfs create fpool/usr/home
2024-06-14.19:49:25 zfs create -o setuid=off fpool/usr/ports
2024-06-14.19:49:25 zfs create fpool/usr/src
2024-06-14.19:49:25 zfs create -o mountpoint=/var -o canmount=off fpool/var
2024-06-14.19:49:25 zfs create -o exec=off -o setuid=off fpool/var/audit
2024-06-14.19:49:25 zfs create -o exec=off -o setuid=off fpool/var/crash
2024-06-14.19:49:25 zfs create -o exec=off -o setuid=off fpool/var/log
2024-06-14.19:49:25 zfs create -o atime=on fpool/var/mail
2024-06-14.19:49:25 zfs create -o setuid=off fpool/var/tmp
2024-06-14.19:49:25 zfs set mountpoint=/fpool fpool
2024-06-14.19:49:25 zpool set bootfs=fpool/ROOT/default fpool
2024-06-14.19:49:25 zpool set cachefile=/mnt/boot/zfs/zpool.cache fpool
2024-06-14.19:49:25 zfs set canmount=noauto fpool/ROOT/default
```

The second is to use `zfs list`:

```
root@zfsdemo:~ # zfs list
NAME                                        USED  AVAIL     REFER  MOUNTPOINT
fpool                                      1.50G   890G       96K  /fpool
fpool/ROOT                                 1.50G   890G       96K  none
fpool/ROOT/13.2-RELEASE_2024-06-14_195559     8K   890G     1.27G  /
fpool/ROOT/default                         1.50G   890G     1.29G  /
fpool/tmp                                   172K   890G      172K  /tmp
fpool/usr                                   456K   890G       96K  /usr
fpool/usr/home                              168K   890G      168K  /usr/home
fpool/usr/ports                              96K   890G       96K  /usr/ports
fpool/usr/src                                96K   890G       96K  /usr/src
fpool/var                                   704K   890G       96K  /var
fpool/var/audit                              96K   890G       96K  /var/audit
fpool/var/crash                              96K   890G       96K  /var/crash
fpool/var/log                               224K   890G      224K  /var/log
```

```
fpool/var/mail                                  96K   890G       96K  /var/mail
fpool/var/tmp                                   96K   890G       96K  /var/tmp
```

Another is to check the `mount` command:

```
root@zfsdemo:~ # mount
fpool/ROOT/default on / (zfs, local, noatime, nfsv4acls)
devfs on /dev (devfs)
/dev/gpt/efiboot0 on /boot/efi (msdosfs, local)
fpool/tmp on /tmp (zfs, local, noatime, nosuid, nfsv4acls)
fpool on /fpool (zfs, local, noatime, nfsv4acls)
fpool/usr/ports on /usr/ports (zfs, local, noatime, nosuid, nfsv4acls)
fpool/usr/src on /usr/src (zfs, local, noatime, nfsv4acls)
fpool/var/log on /var/log (zfs, local, noatime, noexec, nosuid, nfsv4acls)
fpool/usr/home on /usr/home (zfs, local, noatime, nfsv4acls)
fpool/var/tmp on /var/tmp (zfs, local, noatime, nosuid, nfsv4acls)
fpool/var/crash on /var/crash (zfs, local, noatime, noexec, nosuid, nfsv4acls)
fpool/var/mail on /var/mail (zfs, local, nfsv4acls)
fpool/var/audit on /var/audit (zfs, local, noatime, noexec, nosuid, nfsv4acls)
```

2.2 ZFS On Linux (Raspberry Pi OS)

ZFS on Linux (ZoL) is currently supported on Raspberry Pi OS, and an example of how to set it up is shown below.

We are using:

```
pi@pi4-8G:~ $ uname -a
Linux pi4-8G 6.6.31+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.31-1+rpt1
  (2024-05-29) aarch64 GNU/Linux

root@pi4-8G:~# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 12 (bookworm)"
NAME="Debian GNU/Linux"
VERSION_ID="12"
VERSION="12 (bookworm)"
VERSION_CODENAME=bookworm
ID=debian
```

I learned most of this from a youtube video at: https://www.youtube.com/watch?v=5vQRGxTTqPw

Install some required packages, making sure the package database is up to date first. The installation of the three packages (`raspberrypi-kernel-headers zfs-dkms zfsutils-linux`) will take some time (almost an hour and a half for this installation).

```
$ sudo apt update
$ sudo apt install raspberrypi-kernel-headers zfs-dkms zfsutils-linux
```

You will get a license popup noting that the OpenZFS license is not compatible with the GPL license. We are not using the system in a manner that will cause this license incompatibility to be an issue.

Make sure the system is fully upgraded, and then reboot.

```
$ sudo apt full-upgrade
$ sudo reboot
```

Now that the system has all the tools and kernel modules for ZFS, we can begin to make a ZFS storage pool.

You have to manage the disks you'll use. In many cases, you can just use the full disk in the pool creation command, but we'll do something a little different.

We use the `gdisk` command, since it supports ZFS partitions, whereas others such as `parted` or `fdisk` do not.

```
root@pi4-8G:~# gdisk /dev/sda
GPT fdisk (gdisk) version 1.0.9

Partition table scan:
  MBR: not present
  BSD: not present
  APM: not present
  GPT: not present

Creating new GPT entries in memory.

Command (? for help):
```

It is useful to get a list of commands:

```
Command (? for help): ?
b       back up GPT data to a file
c       change a partition's name
d       delete a partition
i       show detailed information on a partition
l       list known partition types
n       add a new partition
o       create a new empty GUID partition table (GPT)
p       print the partition table
q       quit without saving changes
```

```
r       recovery and transformation options (experts only)
s       sort partitions
t       change a partition's type code
v       verify disk
w       write table to disk and exit
x       extra functionality (experts only)
?       print this menu
```

We use `o` to create a new GPT table.

```
Command (? for help): o
This option deletes all partitions and creates a new protective MBR.
Proceed? (Y/N): y

Command (? for help):
```

Now we add a ZFS partition type on most of the disk. We specify 900G, which is not the whole disk, but this allows us to easily create a replacement `vdev` in the case of failure. Sometimes using the whole disk will result in slightly different sizes from different manufacturers, and this can cause problems when a new disk is a few bytes too small.

The default alignment for partitions is now recommended to land on 1MB boundaries, which `gdisk` uses as a default (first sector = 2048 below).

```
Command (? for help): n
Partition number (1-128, default 1):
First sector (34-1953525134, default = 2048) or {+-}size{KMGTP}:
Last sector (2048-1953525134, default = 1953523711) or {+-}size{KMGTP}: 900G
Current type is 8300 (Linux filesystem)
Hex code or GUID (L to show codes, Enter = 8300): a504
Changed type of partition to 'FreeBSD ZFS'
```

The magic number `a504` can be found using the list partition types command `l`.

```
Command (? for help): l
Type search string, or <Enter> to show all codes: zfs
a504 FreeBSD ZFS                        a584 Midnight BSD ZFS
bf01 Solaris /usr & Mac ZFS
```

We will also name the partition so we can use a handy `/dev/disk/by-partlabel/` device name, rather than `/dev/sda1` and so on.

```
Command (? for help): c
Using 1
Enter name: pipool0
```

Now we print and then save the partition table:

```
Command (? for help): p
Disk /dev/sda: 1953525168 sectors, 931.5 GiB
Model: Tech
Sector size (logical/physical): 512/4096 bytes
Disk identifier (GUID): C1B210AD-3108-40FF-9536-D69AA53A5C09
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 1953525134
Partitions will be aligned on 2048-sector boundaries
Total free space is 66090348 sectors (31.5 GiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048      1887436800   900.0 GiB   A504  pipool0
```

Saving:

```
Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? (Y/N): y
OK; writing new GUID partition table (GPT) to /dev/sda.
The operation has completed successfully.
```

Now if we list the `/dev/disk/by-partlabel`, we should see our new label:

```
oot@pi4-8G:~# ls -la /dev/disk/by-partlabel/
total 0
drwxr-xr-x 2 root root  60 Jun 21 19:15 .
drwxr-xr-x 9 root root 180 Jun 21 19:15 ..
lrwxrwxrwx 1 root root  10 Jun 21 19:15 pipool0 -> ../../sda1
```

Now we can do the same partitioning for the second USB device using the same commands as above, except using `/dev/sdb` and a label of `pipool1`.

```
root@pi4-8G:~# gdisk /dev/sdb
GPT fdisk (gdisk) version 1.0.9

Partition table scan:
  MBR: not present
  BSD: not present
  APM: not present
  GPT: not present
```

```
Creating new GPT entries in memory.

Command (? for help): o
This option deletes all partitions and creates a new protective MBR.
Proceed? (Y/N): y

Command (? for help): n
Partition number (1-128, default 1):
First sector (34-1953525134, default = 2048) or {+-}size{KMGTP}:
Last sector (2048-1953525134, default = 1953523711) or {+-}size{KMGTP}: 900G
Current type is 8300 (Linux filesystem)
Hex code or GUID (L to show codes, Enter = 8300): a504
Changed type of partition to 'FreeBSD ZFS'

Command (? for help): c
Using 1
Enter name: pipool1

Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? (Y/N): y
OK; writing new GUID partition table (GPT) to /dev/sdb.
The operation has completed successfully.
```

We have two new 900GiB `vdevs` which we can use to create a ZFS mirror pool. They are listed by partition label:

```
root@pi4-8G:~# ls -la /dev/disk/by-partlabel/
total 0
drwxr-xr-x 2 root root  80 Jun 21 19:19 .
drwxr-xr-x 9 root root 180 Jun 21 19:15 ..
lrwxrwxrwx 1 root root  10 Jun 21 19:15 pipool0 -> ../../sda1
lrwxrwxrwx 1 root root  10 Jun 21 19:19 pipool1 -> ../../sdb1
```

Let's issue the `zpool` command:

```
root@pi4-8G:~# zpool create mirror \
  /dev/disk/by-partlabel/pipool0 /dev/disk/by-partlabel/pipool1
cannot create 'mirror': name is reserved
pool name may have been omitted
```

Whoops, forgot the pool name:

```
root@pi4-8G:~# zpool create pipool mirror \
  /dev/disk/by-partlabel/pipool0 /dev/disk/by-partlabel/pipool1
```

Let's see the results:

```
root@pi4-8G:~# zpool status
  pool: pipool
 state: ONLINE
config:

        NAME         STATE     READ WRITE CKSUM
        pipool       ONLINE       0     0     0
          mirror-0   ONLINE       0     0     0
            pipool0  ONLINE       0     0     0
            pipool1  ONLINE       0     0     0

errors: No known data errors
```

Let's put a dataset on it:

```
root@pi4-8G:~# zfs create pipool/data
```

Use our `zfs list` command:

```
root@pi4-8G:~# zfs list
NAME          USED  AVAIL  REFER  MOUNTPOINT
pipool        516K   868G    96K  /pipool
pipool/data    96K   868G    96K  /pipool/data
```

We can take a snapshot of our new empty `data` dataset, so we can demonstrate a `rollback` command.

```
oot@pi4-8G:~# zfs snap pipool/data@empty
```

Let's list everything in the pool:

```
root@pi4-8G:~# zfs list -t all
NAME                USED  AVAIL  REFER  MOUNTPOINT
pipool              600K   868G    96K  /pipool
pipool/data          96K   868G    96K  /pipool/data
pipool/data@empty     0B      -    96K  -
```

Put some random noise into a file on the `pipool/data` dataset:

```
root@pi4-8G:~# cd /pipool/data
```

```
root@pi4-8G:/pipool/data# dd if=/dev/zero of=zeros bs=1M count=2
2+0 records in
2+0 records out
2097152 bytes (2.1 MB, 2.0 MiB) copied, 0.00777732 s, 270 MB/s

root@pi4-8G:/pipool/data# ls -lah
total 1.5K
drwxr-xr-x 2 root root    3 Jun 21 19:33 .
drwxr-xr-x 3 root root    3 Jun 21 19:26 ..
-rw-r--r-- 1 root root 2.0M Jun 21 19:33 zeros
```

Our snapshot now has some data used (64K):

```
root@pi4-8G:/pipool/data# zfs list -t all
NAME                USED  AVAIL  REFER  MOUNTPOINT
pipool              760K   868G    96K  /pipool
pipool/data         160K   868G    96K  /pipool/data
pipool/data@empty    64K      -    96K  -
```

So our snapshot does not have the `zeros` file in it. Since a file with nothing but zeros is not usually useful, we'll `rollback` our dataset as a demonstration. When we `rollback`, all changes in the dataset since the snapshot are discarded. Be careful when doing this.

```
root@pi4-8G:/pipool/data# zfs rollback pipool/data@empty
```

We can check our `zfs list` and look for our `zeros` file, noting it is gone.

```
root@pi4-8G:/pipool/data# zfs list -t all
NAME                USED  AVAIL  REFER  MOUNTPOINT
pipool              684K   868G    96K  /pipool
pipool/data          96K   868G    96K  /pipool/data
pipool/data@empty     0B      -    96K  -

root@pi4-8G:/pipool/data# ls -la
total 1
drwxr-xr-x 2 root root 2 Jun 21 19:26 .
drwxr-xr-x 3 root root 3 Jun 21 19:26 ..
```

Here are some other useful commands:

```
root@pi4-8G:/pipool/data# zpool list
NAME     SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
pipool   896G   752K   896G        -         -     0%     0%  1.00x    ONLINE  -

root@pi4-8G:/pipool/data# zpool history
History for 'pipool':
2024-06-21.19:23:33 zpool create pipool mirror /dev/disk/by-partlabel/pipool0 /dev/disk/by-partlabel/pipool1
2024-06-21.19:26:53 zfs create pipool/data
2024-06-21.19:30:01 zfs snap pipool/data@empty
```

```
2024-06-21.19:39:14 zfs rollback pipool/data@empty
```

We can `export` the `pipool` in order to remove the drives:

```
root@pi4-8G:~# zpool export pipool
root@pi4-8G:~# zpool status
no pools available
```

Pull the drives out.

3 Useful Features And Examples

This section is divided into feature/example subsections, so as a feature is introduced, we can see an example of it immediately. Features that we'll examine:

1. Useful pool commands.
2. Exporting and importing pools for transferring data between systems.
3. Useful dataset commands.
4. Using snapshot / rollback.
5. Mentioning `zfs send` and `zfs receive` with optional SSH tunneling.

3.1 Useful Pool Commands

These are the `zpool` commands that I use fairly regularly.

1. `zpool history`
   This is a great command for reviewing and reproducing how a pool was initial created, as well as all the operations taken on datasets. Truly fantastic.
2. `zpool status`
   This is the second best command in that it shows the current state of the pool and its structure. You can find out whether disks have failed and the status of scrubs or resilvers.
3. `zpool scrub`
   This command causes the ZFS system to validate all file system checksums for the entire pool. It will automatically repair damage if you have a redundant pool and alert you to problems with single device pools. It is often run as a periodic `cron` job.
4. `zpool list`
   If you have more than one pool, this command will show a nice list of the pools,

their space statistics and other information.

3.1.1 Useful Pool Scripts

Here is a `cron` script that I run daily called `/root/bin/cron-zpool.sh`:

```
#!/bin/sh
#
# Check pool status daily
#
zpool list
zpool status -v
zpool iostat
zpool history
```

Its `/etc/crontab` entry looks like this:

```
# Make sure to keep an eye on ZFS pools
1       7       *       *       *       root    /root/bin/cron-zpool.sh
```

The output of this command comes via an email and looks like this:

```
NAME      SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
jpool    1.81T   220G  1.60T        -         -     7%    11%  1.00x    ONLINE  -
mpool    1.80T   121G  1.68T        -         -     0%     6%  1.00x    ONLINE  -
poupool  1.81T  76.3G  1.74T        -         -    31%     4%  1.00x    ONLINE  -
  pool: jpool
 state: ONLINE
  scan: scrub repaired 0B in 00:49:23 with 0 errors on Tue Jun 11 04:17:14 2024
config:

 NAME            STATE     READ WRITE CKSUM
 jpool           ONLINE       0     0     0
   mirror-0      ONLINE       0     0     0
     ada0p1.eli  ONLINE       0     0     0
     ada2p1.eli  ONLINE       0     0     0
     ada4p1.eli  ONLINE       0     0     0

errors: No known data errors

  pool: mpool
 state: ONLINE
  scan: scrub repaired 0B in 00:04:29 with 0 errors on Sun Jun  9 03:06:38 2024
config:

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

```
     ada3p4.eli  ONLINE       0     0     0
     ada1p4.eli  ONLINE       0     0     0

errors: No known data errors

  pool: poupool
 state: ONLINE
  scan: scrub repaired 0B in 00:01:16 with 0 errors on Tue Jun 11 03:29:11 2024
config:

 NAME        STATE     READ WRITE CKSUM
 poupool     ONLINE       0     0     0
   nvd0      ONLINE       0     0     0
   nvd1      ONLINE       0     0     0

errors: No known data errors
              capacity     operations     bandwidth
pool        alloc   free   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
jpool        220G  1.60T      1     17   281K   395K
mpool        121G  1.68T      1     16   153K   244K
poupool     76.3G  1.74T      2     32  56.9K  1.95M
----------  -----  -----  -----  -----  -----  -----
History for 'jpool':
2023-01-02.18:06:34 zpool create jpool mirror ada0p1.eli ada1p1.eli ada3p1.eli
2023-01-02.21:49:21 zfs create -o mountpoint=/usr/jails -o canmount=off jpool/jails
2023-01-02.21:52:20 zfs create jpool/jails/dns
2023-01-02.22:18:40 zfs create jpool/jails/pkg
2023-01-06.17:12:07 zfs rename jpool/jails/dns jpool/jails/dnsprimary
2023-01-09.20:22:34 zfs create jpool/jails/minimal
...
```

3.2 Export / Import

If your storage pool drives are removable, you must `zpool export` the pool before disconnecting the drives. All datasets are automatically un-mounted during the `export`. When you plug them in again, you must `zpool import` the pool.

When you `import` a (possibly unknown) pool for the first time on a system, always use the alternative root option so as to not clobber any already mounted file systems. For example, after exporting `pipool`, we can import it again with an alternate root mount point:

```
# zpool import -R /mnt pipool

root@pi4-8G:~# zfs list
NAME          USED  AVAIL  REFER  MOUNTPOINT
```

```
pipool        776K   868G    96K  /mnt/pipool
pipool/data   152K   868G    96K  /mnt/pipool/data
```

The `-R` (altroot option) will force any datasets to be mounted relative to the `/mnt` mount point. This is critical if you are importing a backup that has a copy of your root file system dataset on it. You don't want your running root file system (`/`) suddenly changing on you.

We saw an example of `zfs export` in our Raspberry Pi demonstration.

```
# zpool export mypool
```

3.3 Useful Dataset Commands

These commands help you examine or manipulate datasets.

1. `zfs list`
   Straightforward, shows datasets. Can use the `-t` option to specify `snap` or `all` to get snapshots only or all datasets including their snapshots, respectively.

   ```
   # zfs list -t all
   ```

2. `zfs rename`
   It can be useful to rename a dataset or snapshot. I use this if I've made an error in naming.

   ```
   # zfs rename jpool/jails/dns jpool/jails/dnsprimary
   ```

3. `zfs mount / unmount`
   These allow you to mount and un-mount datasets as needed. Most of this is done automatically.

   ```
   # zfs unmount pipool/data
   ```

3.3.1 Useful Dataset Scripts

Here is a script to keep up to seven days of daily snapshots (run as a root cron job).

```
#!/bin/sh
#
# A script to create a full copy of the local zfs pool
# for transfer to the das backup.
#
# The 7 day rolling snap was extracted from a man zfs example.
# Should be run daily from root's crontab.
#
```

```
zfsroot="jpool"

zfs destroy -r ${zfsroot}@7daysago
zfs rename -r ${zfsroot}@6daysago @7daysago
zfs rename -r ${zfsroot}@5daysago @6daysago
zfs rename -r ${zfsroot}@4daysago @5daysago
zfs rename -r ${zfsroot}@3daysago @4daysago
zfs rename -r ${zfsroot}@2daysago @3daysago
zfs rename -r ${zfsroot}@yesterday @2daysago
zfs rename -r ${zfsroot}@today @yesterday
zfs snapshot -r ${zfsroot}@today
```

3.4 Snapshot / Rollback

A snapshot is a point in time reference of the state of the file data stored on a dataset. Snapshots are instant and initially take up no space. After a snapshot is taken, new files and modifications of files cause new storage blocks to be used on the live file system. This is known as copy-on-write. The snapshot will reference the old non-modified blocks and will have no relation with the new file blocks.

Always take a recursive snapshot (using the `-r` option) before doing system updates/upgrades, so you can easily `rollback` to the previous state if the update/upgrades fails to please you.

The Raspberry Pi demo showed a snapshot and rollback example.

3.5 Send / Receive

The `zfs send` and `zfs recv` commands allow for the movement and temporary copying of datasets, either into any pool, or a standalone file.

I use this to make copies of jails and other business data. This is a powerful feature that can be tunneled over SSH. I don't have time to cover it today, but your own research into the possibilities will reward you.

You can easily use a template dataset that you make a copy of using `zfs send` piped into `zfs recv`. I used this to create a template jail for my services and then made multiple copies, as needed.

Here is some pool history showing the process:

```
023-07-30.22:19:59 zfs snap jpool/jails/minimal@for_local_jails
2023-07-30.22:20:37 zfs recv -Fv jpool/jails/dnsprimary
2023-07-30.22:20:58 zfs recv -Fv jpool/jails/ldap
```

```
2023-07-31.22:55:41 zfs recv -Fv jpool/jails/sccs
```

You don't see the `zfs send` command, but a typical one looks like:

```
# zfs send -vR jpool/jails/minimal@for_local_jails | zfs recv -Fv jpool/jails/service1
```

You end up with an exact copy of the `minimal` jail for use as a new service.

4 Appendicies

For completeness, these appendix entries may contain useful reference material.

4.1 FreeBSD Installation Log

The installation of FreeBSD using the `bsdinstall` program leaves a fairly detailed log file at `/var/log/bsdinstall_log`. This file can be mined for the actual commands used during the installation process. I wrote a (very ugly) script in python to extract the commands associated with disk partitioning and ZFS configuration. The results for our example FreeBSD install follow.

NOTE: Don't run any of these scripts on your own system. Very bad things will happen.

4.1.1 Creating Disk Partitions

This is a shell script created from the section of the `bsdinstall` log file reporting on the partitioning of disks. There seem to be gaps with respect to commands regarding the `efiboot` partition, but that is out of scope at this time.

```
mv@think:~/tmp % cat zfs_create_diskpart.sh
#!/bin/sh
#zfs_create_diskpart

gpart destroy -F "nvd0"
graid delete "nvd0"
zpool labelclear -f "/dev/nvd0"
gpart create -s gpt "nvd0"
gpart destroy -F "nvd0"
gpart create -s gpt "nvd0"
gpart add -a 4k -l efiboot0 -t efi -s 260M "nvd0"
newfs_msdos "/dev/gpt/efiboot0"
printf "$FSTAB_FMT" "/dev/gpt/efiboot0" "/boot/efi" "msdosfs" "rw" "2" "2" >> "/tmp/bsdinstall_etc/fstab"
gpart add -a 4k -l gptboot0 -t freebsd-boot -s 512k "nvd0"
gpart bootcode -b "/boot/pmbr" -p "/boot/gptzfsboot" -i 2 "nvd0"
gpart add -a 1m -l swap0 -t freebsd-swap -s 4294967296b "nvd0"
zpool labelclear -f "/dev/nvd0p3"
gpart add -a 1m -l zfs0 -t freebsd-zfs "nvd0"
zpool labelclear -f "/dev/nvd0p4"
printf "$FSTAB_FMT" "/dev/nvd0p3.eli" "none" "swap" "sw" "0" "0" >> "/tmp/bsdinstall_etc/fstab"

gpart destroy -F "nvd1"
```

```
graid delete "nvd1"
zpool labelclear -f "/dev/nvd1"
gpart create -s gpt "nvd1"
gpart destroy -F "nvd1"
gpart create -s gpt "nvd1"
gpart add -a 4k -l efiboot1 -t efi -s 260M "nvd1"
gpart add -a 4k -l gptboot1 -t freebsd-boot -s 512k "nvd1"
gpart bootcode -b "/boot/pmbr" -p "/boot/gptzfsboot" -i 2 "nvd1"
gpart add -a 1m -l swap1 -t freebsd-swap -s 4294967296b "nvd1"
zpool labelclear -f "/dev/nvd1p3"
gpart add -a 1m -l zfs1 -t freebsd-zfs "nvd1"
zpool labelclear -f "/dev/nvd1p4"
printf "$FSTAB_FMT" "/dev/nvd1p3.eli" "none" "swap" "sw" "0" "0" >> "/tmp/bsdinstall_etc/fstab"
```

4.1.2 Creating Encryption and the ZFS Pool / Datasets

Next is a shell script created from the section of the `bsdinstall` log file reporting on the configuration of the ZFS details.

```
mv@think:~/tmp % cat zfs_create_boot.sh
#!/bin/sh
#zfs_create_boot

:> "/tmp/bsdinstall_etc/fstab"
printf "$FSTAB_FMT" "# Device" "Mountpoint" "FStype" "Options" "Dump" "Pass#" >> "/tmp/bsdinstall_etc/fstab"
zpool destroy "fpool"
sysctl vfs.zfs.min_auto_ashift=12
#!/bin/sh
#zfs_create_boot

kldload aesni
geli init -bg -e AES-XTS -J - -l 256 -s 4096 "nvd0p4"
geli attach -j - "nvd0p4"
geli init -bg -e AES-XTS -J - -l 256 -s 4096 "nvd1p4"
geli attach -j - "nvd1p4"

zpool create -o altroot=/mnt -O compress=lz4 -O atime=off -m none -f "fpool" mirror  nvd0p4.eli nvd1p4.eli
zfs create -o mountpoint=none "fpool/ROOT"
zfs create -o mountpoint=/ "fpool/ROOT/default"
zfs create -o mountpoint=/tmp -o exec=on -o setuid=off "fpool/tmp"
zfs create -o mountpoint=/usr -o canmount=off "fpool/usr"
zfs create  "fpool/usr/home"
zfs create -o setuid=off "fpool/usr/ports"
zfs create  "fpool/usr/src"
zfs create -o mountpoint=/var -o canmount=off "fpool/var"
zfs create -o exec=off -o setuid=off "fpool/var/audit"
zfs create -o exec=off -o setuid=off "fpool/var/crash"
zfs create -o exec=off -o setuid=off "fpool/var/log"
zfs create -o atime=on "fpool/var/mail"
zfs create -o setuid=off "fpool/var/tmp"
zfs set "mountpoint=/fpool" "fpool"

mkdir -p "/mnt/tmp"
chmod 1777 "/mnt/tmp"
mkdir -p "/mnt/var/tmp"
chmod 1777 "/mnt/var/tmp"
zpool set bootfs="fpool/ROOT/default" "fpool"
mkdir -p "/mnt/boot/zfs"
zpool set cachefile="/mnt/boot/zfs/zpool.cache" "fpool"
zfs set "canmount=noauto" "fpool/ROOT/default"
```

```
echo "zfs_enable=\"YES\"" >> "/tmp/bsdinstall_etc/rc.conf.zfs"
echo "kern.geom.label.disk_ident.enable=\"0\"" >> "/tmp/bsdinstall_boot/loader.conf.zfs"
echo "kern.geom.label.gptid.enable=\"0\"" >> "/tmp/bsdinstall_boot/loader.conf.zfs"
echo "vfs.zfs.min_auto_ashift=12" >> "/tmp/bsdinstall_etc/sysctl.conf.zfs"
echo "aesni_load=\"YES\"" >> "/tmp/bsdinstall_boot/loader.conf.aesni"
echo "geom_eli_load=\"YES\"" >> "/tmp/bsdinstall_boot/loader.conf.geli"
```