Using ZFS on A Souped-Up Raspberry Pi 5

Mark G.

May 9, 2026

This presentation gives an overview of the Zettabyte File System (ZFS) on a Raspberry Pi 5. The Pi 5 has 16GB of memory, helpful since ZFS is memory hungry. The operating system is 64-bit Raspberry Pi OS (Trixie). Topics covered are how to enable ZFS on Raspberry Pi OS, how to create a ZFS storage system, and some examples of ZFS in use.

Of note is the `Pironman 5 Max` case used to hold the Pi 5 and its built-in ability to allow for two NVMe drives to be attached directly to the Pi's PCIe controller. ZFS works much better with directly attached drives rather than USB attached drives.

Contents

List of Figures

1 Hardware and Apparatus

Technologies used in this demonstration are:

- Raspberry Pi 5 16GB RAM (Raspberry Pi OS - Trixie)
- Pironman 5 Max case ($132.99)
- 2 Pironman HAT attached M.2 NVMe drives

I bought the Pi 5 as a iRasptek Starter Kit[1] last year for $279.99, before the prices went insane, as it is now $499.99.

Figure 1: `iRasptek Pi 5 Starter Kit`

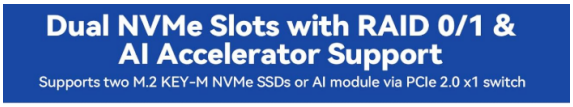
The SD card came with the previous version (bookworm) of Raspberry Pi OS, which I replaced with the current version (Trixie) for this demonstration. I used the 64-bit full desktop version from the Raspberry Pi Imager selector.

The Pironman case can be found on amazon at Pironman 5 Max[2].

The NVMe drives are attached to this adapter board, see figure 3, that is part of the Pironman case.

[1] https://www.amazon.ca/dp/B0DSSQ8C53
[2] https://www.amazon.ca/dp/B0F8J5HW21

Figure 2: `Pironman 5 Max`



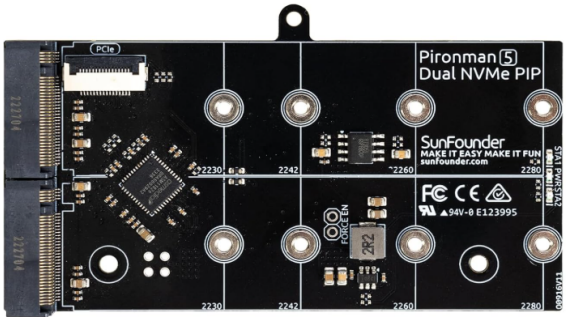


Figure 3: `Pironman NVMe adapter board`

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

The video is out-of-date and specifics are noted below.

2.1 System Identification

We are using:

```
pi@pironman:~ $ uname -a
Linux pironman 6.12.75+rpt-rpi-2712 #1 SMP PREEMPT Debian
 1:6.12.75-1+rpt1 (2026-03-11) aarch64 GNU/Linux

pi@pironman:~ $ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 13 (trixie)"
NAME="Debian GNU/Linux"
VERSION_ID="13"
VERSION="13 (trixie)"
VERSION_CODENAME=trixie
DEBIAN_VERSION_FULL=13.4
ID=debian
```

2.2 System Update / Upgrade

We start with the `apt update / upgrade` steps to ensure we are fully up to date:

```
pi@pironman:~ $ sudo apt update
[sudo] password for pi:
Hit:1 http://deb.debian.org/debian trixie InRelease
Hit:2 http://deb.debian.org/debian trixie-updates InRelease
Hit:3 http://deb.debian.org/debian-security trixie-security InRelease
Hit:4 http://archive.raspberrypi.com/debian trixie InRelease
63 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Apply the upgrades:

```
pi@pironman:~ $ sudo apt upgrade
Upgrading:
  firefox   ...

Summary:
  Upgrading: 63, Installing: 0, Removing: 0, Not Upgrading: 0
```

```
  Download size: 243 MB
  Space needed: 1,060 kB / 231 GB available

Continue? [Y/n] y
...
Processing triggers for initramfs-tools (0.148.3+rpt2) ...
update-initramfs: Generating /boot/initrd.img-6.12.75+rpt-rpi-v8
'/boot/initrd.img-6.12.75+rpt-rpi-v8' -> '/boot/firmware/initramfs8'
update-initramfs: Generating /boot/initrd.img-6.12.75+rpt-rpi-2712
'/boot/initrd.img-6.12.75+rpt-rpi-2712' -> '/boot/firmware/initramfs_2712'
Processing triggers for hicolor-icon-theme (0.18-2) ...
Processing triggers for libc-bin (2.41-12+rpt1+deb13u2) ...
Processing triggers for man-db (2.13.1-1) ...
Processing triggers for ca-certificates-java (20240118) ...
done.
Setting up openjdk-21-jre:arm64 (21.0.11+10-1~deb13u2) ...
```

2.3 ZFS Tools Installation

Now we can install the required packages from the video mentioned above. The installation of three packages (`raspberrypi-kernel-headers zfs-dkms zfsutils-linux`) were listed. Unfortunately, it seems the video is out of date and we get this error:

```
pi@pironman:~ $ sudo apt install raspberrypi-kernel-headers zfs-dkms zfsutils-linux
Error: Unable to locate package raspberrypi-kernel-headers
```

So we just remove the headers part and see what happens:

```
i@pironman:~ $ sudo apt install zfs-dkms zfsutils-linux
Installing:
  zfs-dkms  zfsutils-linux

Installing dependencies:
  dkms  gawk  libnvpair3linux  libsigsegv2  libuutil3linux
  libzfs7linux  libzpool7linux  zfs-zed

Suggested packages:
  menu  gawk-doc  debhelper  nfs-kernel-server  samba-common-bin
  fs-initramfs  | zfs-dracut

Recommended packages:
  linux-libc-dev

Summary:
```

```
  Upgrading: 0, Installing: 10, Removing: 0, Not Upgrading: 0
  Download size: 7,296 kB
  Space needed: 31.1 MB / 231 GB available

Continue? [Y/n] y
```

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.

This looks like:

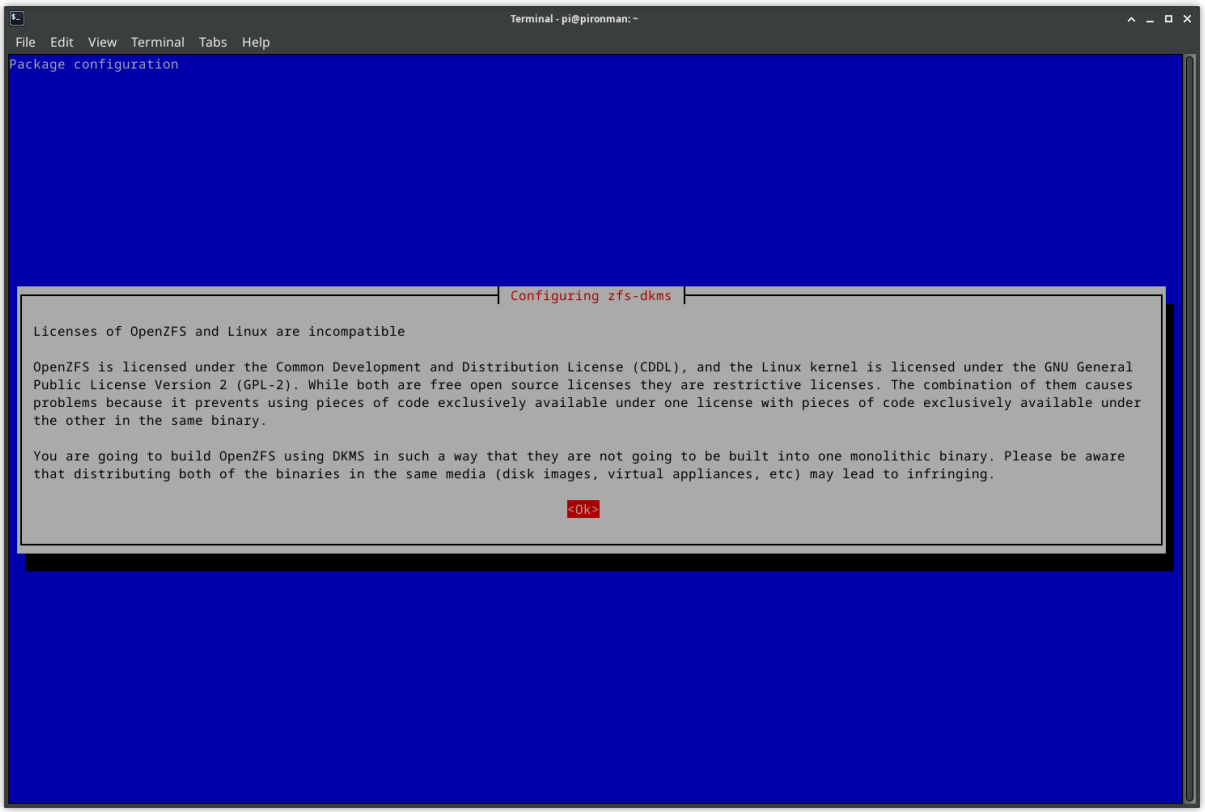


Figure 4: `License for OpenZFS on Linux`

The ensuing build / installation will take some time (about 20 minutes or so for this installation) and completed successfully. See the appendix for a complete transcript of the installation.

Now we make sure the system is fully upgraded, and then reboot.

```
pi@pironman:~ $ sudo apt full-upgrade
[sudo] password for pi:
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
pi@pironman:~ $ shutdown -r now
Call to Reboot failed: Interactive authentication required.
pi@pironman:~ $ sudo shutdown -r now
pi@pironman:~ $
```

3 Drive Preparation

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.

Here's a list of the drives / partitions on the system:

```
pi@pironman:~ $ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0         7:0    0     2G  0 loop
mmcblk0     179:0    0 238.3G  0 disk
-mmcblk0p1 179:1    0   512M  0 part /boot/firmware
-mmcblk0p2 179:2    0 237.8G  0 part /
zram0       254:0    0     2G  0 disk [SWAP]
nvme0n1     259:0    0 931.5G  0 disk
-nvme0n1p1 259:1    0   260M  0 part
-nvme0n1p2 259:2    0   512K  0 part
-nvme0n1p3 259:3    0     4G  0 part
-nvme0n1p4 259:4    0 927.3G  0 part
nvme1n1     259:5    0 931.5G  0 disk
-nvme1n1p1 259:6    0   260M  0 part
-nvme1n1p2 259:7    0   512K  0 part
-nvme1n1p3 259:8    0     4G  0 part
-nvme1n1p4 259:9    0 927.3G  0 part
```

The NVMe drives already have some partitions, which we'll remove and replace.

3.1 Partitioning Tool `gdisk`

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

```
pi@pironman:~ $ gdisk
GPT fdisk (gdisk) version 1.0.10

Type device filename, or press <Enter> to exit:
```

We simply press `<Enter>` and exit.

Now let's use the first NVMe drive `nvme0n1`. We need to be `root`, so we just become the superuser in order to eliminate all the `sudo` typing.

```
pi@pironman:~ $ sudo su -
[sudo] password for pi:
root@pironman:~# gdisk /dev/nvme0n1
GPT fdisk (gdisk) version 1.0.10

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

Found valid GPT with protective MBR; using GPT.

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 one megabyte (1MB) boundaries, which `gdisk` uses as a default (first sector = 2048 below).

We use the default value for the first sector, and `900G` for the last sector:

```
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): L
Type search string, or <Enter> to show all codes: zfs
a504 FreeBSD ZFS                        a584 Midnight BSD ZFS
bf01 Solaris /usr & Mac ZFS
Hex code or GUID (L to show codes, Enter = 8300): L
Type search string, or <Enter> to show all codes: ZFS
a504 FreeBSD ZFS                        a584 Midnight BSD ZFS
bf01 Solaris /usr & Mac ZFS
Hex code or GUID (L to show codes, Enter = 8300): a504
Changed type of partition to 'FreeBSD ZFS'
```

The magic number `a504` was found using the list partition types command `L`, as seen above. The two searches for lowercase `zfs` and uppercase `ZFS` were done to make sure that we found all ZFS types. The choice of `a504`, a.k.a. `FreeBSD ZFS` seems sensible since it is a mature implementation and may be compatible with more systems.

Now we can print using the `p` command the partition table:

```
Command (? for help): p
Disk /dev/nvme0n1: 1953525168 sectors, 931.5 GiB
Model: WD_BLACK SN770 1TB
Sector size (logical/physical): 512/512 bytes
```

```
Disk identifier (GUID): 9E3EC082-1560-4263-927A-FFB03A912E43
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  FreeBSD ZFS

Command (? for help):
```

We will also name the partition so we can use a handy `/dev/disk/by-partlabel/` device name, rather than `/dev/nvme0n1` 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/nvme0n1: 1953525168 sectors, 931.5 GiB
Model: WD_BLACK SN770 1TB
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): 9E3EC082-1560-4263-927A-FFB03A912E43
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/nvme0n1.
The operation has completed successfully.
```

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

```
root@pironman:~# ls -la /dev/disk/by-partlabel/
total 0
drwxr-xr-x  2 root root 140 May  8 14:30 .
drwxr-xr-x 12 root root 240 May  7 21:05 ..
...
lrwxrwxrwx  1 root root  15 May  8 14:30 pipool0 -> ../../nvme0n1p1
```

Now we can do the same partitioning for the second NVMe device using the same commands as above, except using `/dev/nvme1n1` and a label of `pipool1`.

```
root@pironman:~# gdisk /dev/nvme1n1
GPT fdisk (gdisk) version 1.0.10

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

Found valid GPT with protective MBR; using GPT.

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): p
Disk /dev/nvme1n1: 1953525168 sectors, 931.5 GiB
Model: WD_BLACK SN770 1TB
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): 9E4E45FB-A10F-4103-AFA5-81A9BD54E22A
```

```
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  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/nvme1n1.
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@pironman:~# ls -la /dev/disk/by-partlabel/
total 0
drwxr-xr-x  2 root root  80 May  8 14:33 .
drwxr-xr-x 12 root root 240 May  7 21:05 ..
lrwxrwxrwx  1 root root  15 May  8 14:30 pipool0 -> ../../nvme0n1p1
lrwxrwxrwx  1 root root  15 May  8 14:33 pipool1 -> ../../nvme1n1p1
```

4 Creating the ZFS Storage Pool

There are two commands that operate with ZFS: `zpool` and `zfs`. Pool management is done using the `zpool` commands. Dataset management is done using the `zfs` commands.

Use `man zpool` to get an overview of pool commands. Use man `zpool-create`, as an example, to get more detailed information about the `zpool create` subcommand.

Let's issue the `zpool create` command using the `/dev/disk/by-partlabel/` entries we created specifically for this purpose:

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

Let's see the results:

```
root@pironman:~# 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
```

We can get a list of our active pools:

```
root@pironman:~# zpool list
NAME     SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
pipool   896G   165K   896G        -         -     0%     0%  1.00x    ONLINE  -
```

4.1 `zfs export`

We can `export` a pool if we want to remove it from use by the current system:

```
root@pironman:~# zpool export pipool

root@pironman:~# zpool list
no pools available
```

It is important to remember that any datasets on the pool should be un-mounted and not in use when trying to export a pool. The `zfs unmount` command may help with this, as well as closing any programs using the pool's datasets.

4.2 `zfs import`

We can `import` pools to allow them to be used by the active system. This is often used when you have a ZFS pool on a USB device, for example. To list importable pools, just issue the `zpool import` command with no arguments:

```
root@pironman:~# zpool import
  pool: pipool
    id: 18363470157193984878
 state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

 pipool       ONLINE
```

```
  mirror-0   ONLINE
    pipool0  ONLINE
    pipool1  ONLINE
```

We'll see available pools for import. We can use the import command with the `-a` switch to import all available pools or we can name the pool explicitly:

```
root@pironman:~# zpool import pipool
root@pironman:~# zpool list
NAME     SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
pipool   896G   172K   896G        -         -     0%     0%  1.00x    ONLINE  -
```

5 Creating the ZFS Datasets

Dataset management is done using the `zfs` commands.

Use `man zfs` to get an overview of dataset configuration commands. Use man `zfs-create`, as an example, to get more detailed information about the `zfs create` subcommand.

Let's put a dataset on our new `pipool`:

```
root@pironman:~# zfs create pipool/data
```

This new dataset inherits its mountpoint from the pool's default mountpoint (as shown further below) and appends the dataset's name to the mountpoint, i.e. `/pipool/data`.

Use our `zfs list` command to show datasets and their mountpoints:

```
root@pironman:~# zfs list
NAME          USED  AVAIL  REFER  MOUNTPOINT
pipool        174K   868G    24K  /pipool
pipool/data    24K   868G    24K  /pipool/data
```

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

```
root@pironman:~# zfs snap pipool/data@empty
```

Let's list everything in the pool using the type switch `-t` and the value `all` to show both datasets and snapshots:

```
root@pironman:~# zfs list -t all
NAME                 USED  AVAIL  REFER  MOUNTPOINT
pipool               194K   868G    24K  /pipool
pipool/data           24K   868G    24K  /pipool/data
pipool/data@empty      0B      -    24K  -
```

Notice that the `USED` value for the `pipool/data@empty` snapshot is `0B`.

When we want to interact with files in this dataset (`pipool/data`), we have to change directory to within the dataset's mountpoint (`/pipool/data`).

```
root@pironman:~# cd /pipool/data
```

At this point, we create files and directories as we would any other file system.

Put two megabytes of zeros into a file in the `/pipool/data` directory:

```
root@pironman:/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, 0.000992721 s, 2.1 GB/s

root@pironman:/pipool/data# ls -la
total 2
drwxr-xr-x 2 root root       3 May  8 15:09 .
drwxr-xr-x 3 root root       3 May  8 15:05 ..
-rw-r--r-- 1 root root 2097152 May  8 15:09 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  -
```

Remember, our snapshot was made before the `zeros` file was created, and so does not have that file.

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@pironman:/pipool/data# zfs rollback pipool/data@empty
```

We can look for our `zeros` file, noting it is gone.

```
root@pironman:/pipool/data# ls -la
total 1
drwxr-xr-x 2 root root 2 May  8 15:05 .
drwxr-xr-x 3 root root 3 May  8 15:05 ..
```

5.1 Another Dataset (pi's home)

A second dataset is easily created on the pool. We'll use this opportunity to move our pi user's home directory to ZFS, giving us snapshot and other ZFS benefits for our user data.

```
root@pironman:~# zfs create pipool/home
```

Moving a home directory is tricky when logged in as that user, so we'll have to make sure to be logged in as only root, or some other user and then root.

The simplest way to do this is to logout the user `pi` from the GUI, then create a new temporary user to which we'll give `sudo` privileges.

The `adduser` options are as follows:

- `--ingroup sudo` - place the new user in the `sudo` group so it can become the superuser and allow us to move the `pi` user directory.
- `--no-create-home` - we don't want to bother creating a home directory for this user, since we'll delete it soon.
- `mark` - the user we will create, we'll be prompted for a password and some other optional information (which we'll leave blank).

```
root@pironman:~# adduser --ingroup sudo --no-create-home mark
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for mark
Enter the new value, or press ENTER for the default
 Full Name []:
 Room Number []:
 Work Phone []:
 Home Phone []:
 Other []:
Is the information correct? [Y/n] Y
```

Now we must logout, since we used sudo to become root via the pi user.

```
root@pironman:~# ^D
logout
pi@pironman:~ $ ^D
logout
Connection to 192.168.60.189 closed.
```

This should close all open files used by the pi user.

We'll login as our temp user now:

```
% ssh mark@192.168.60.189
mark@192.168.60.189's password:
Linux pironman 6.12.75+rpt-rpi-2712 #1 SMP PREEMPT Debian 1:6.12.75-1+rpt1
 (2026-03-11) aarch64

Could not chdir to home directory /home/mark: No such file or directory

mark@pironman:/$
```

Make sure we can become root:

```
mark@pironman:/$ sudo su -
root@pironman:~#
```

Now try and move `pi`'s home directory:

```
root@pironman:~# cd /home
root@pironman:/home# mv pi /pipool/home
root@pironman:/home# ls -la
total 8
drwxr-xr-x  2 root root 4096 May  8 19:33 .
drwxr-xr-x 19 root root 4096 May  8 18:25 ..
```

Create a symbolic link so everything can find the new home directory (very important):

```
root@pironman:/home# ln -s /pipool/home/pi pi

root@pironman:/home# ls pi
Desktop  Documents  Downloads  Music  Pictures Public Templates  Videos
```

We should reboot now and login as the `pi` user to make sure everything is working properly.

```
root@pironman:~# shutdown -r now
root@pironman:~# Read from remote host 192.168.60.189: Connection reset by peer
Connection to 192.168.60.189 closed.
```

Clean-up the user database by deleting the temporary user:

```
% ssh pi@192.168.60.189

pi@192.168.60.189's password:
...

pi@pironman:~ $ sudo deluser mark
```

```
[sudo] password for pi:
pi@pironman:~ $
```

What we've done here is created a ZFS dataset that will contain all our `pi` user's settings and files. We can transfer / backup this dataset to other systems, make snapshots and have all the ZFS benefits with two NVMe drives in a redundant mirror underneath.

6 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`.

6.1 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.

6.1.1 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
poupool  1.81T  76.3G  1.74T        -         -    31%     4%  1.00x    ONLINE  -

  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
----------  -----  -----  -----  -----  -----  -----
poupool     76.3G  1.74T      2     32  56.9K  1.95M
----------  -----  -----  -----  -----  -----  -----
History for 'poupool':
2023-01-02.18:06:34 zpool create poupool mirror ada0p1.eli ada1p1.eli ada3p1.eli
...
```

6.2 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, if our `pipool` was on USB drives, after exporting it, 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.

6.3 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
   ```

6.3.1 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
```

6.4 Snapshot / Rollback

A snapshot is a point in time reference to 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.

6.5 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.

A typical `zfs send` command looks like:

```
# zfs snapshot jpool/jails/minimal@for_local_jails
# 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.

SSH can be used with send and receive to allow for dataset transfers directly across the network. Very useful.

7 ZFS Trivia

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.

7.1 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.

7.2 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.

7.3 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.

7.4 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.

7.5 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`.

7.5.1 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.

7.6 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 `ext4`, to be created on a ZFS volume. These volumes benefit from snapshots and other ZFS features.

8 Appendix

The full text of the `zfs-dkms` and `zfsutils-linux` installation are shown here for completeness.

```
Get:1 http://deb.debian.org/debian trixie/main arm64 libsigsegv2 arm64 2.14-1+b2 [34.3 kB]
Get:2 http://deb.debian.org/debian trixie/main arm64 gawk arm64 1:5.2.1-2+b2 [644 kB]
Get:3 http://archive.raspberrypi.com/debian trixie/main arm64 zfs-dkms all 2.4.1-1~bpo13+1~rpt1 [3,684 kB]
Get:4 http://deb.debian.org/debian trixie/main arm64 dkms all 3.2.2-1~deb13u1 [60.1 kB]
Get:5 http://archive.raspberrypi.com/debian trixie/main arm64 libnvpair3linux arm64 2.4.1-1~bpo13+1~rpt1 [73.3 kB]
Get:6 http://archive.raspberrypi.com/debian trixie/main arm64 libuutil3linux arm64 2.4.1-1~bpo13+1~rpt1 [63.2 kB]
Get:7 http://archive.raspberrypi.com/debian trixie/main arm64 libzfs7linux arm64 2.4.1-1~bpo13+1~rpt1 [289 kB]
Get:8 http://archive.raspberrypi.com/debian trixie/main arm64 libzpool7linux arm64 2.4.1-1~bpo13+1~rpt1 [1,624 kB]
Get:9 http://archive.raspberrypi.com/debian trixie/main arm64 zfsutils-linux arm64 2.4.1-1~bpo13+1~rpt1 [727 kB]
Get:10 http://archive.raspberrypi.com/debian trixie/main arm64 zfs-zed arm64 2.4.1-1~bpo13+1~rpt1 [97.4 kB]
Fetched 7,296 kB in 2s (3,188 kB/s)
Preconfiguring packages ...
Selecting previously unselected package libsigsegv2:arm64.
(Reading database ... 165192 files and directories currently installed.)
Preparing to unpack .../libsigsegv2_2.14-1+b2_arm64.deb ...
Unpacking libsigsegv2:arm64 (2.14-1+b2) ...
Setting up libsigsegv2:arm64 (2.14-1+b2) ...
Selecting previously unselected package gawk.
(Reading database ... 165202 files and directories currently installed.)
Preparing to unpack .../0-gawk_1%3a5.2.1-2+b2_arm64.deb ...
Unpacking gawk (1:5.2.1-2+b2) ...
Selecting previously unselected package dkms.
Preparing to unpack .../1-dkms_3.2.2-1~deb13u1_all.deb ...
Unpacking dkms (3.2.2-1~deb13u1) ...
Selecting previously unselected package zfs-dkms.
Preparing to unpack .../2-zfs-dkms_2.4.1-1~bpo13+1~rpt1_all.deb ...
```

```
Unpacking zfs-dkms (2.4.1-1~bpo13+1~rpt1) ...
Selecting previously unselected package libnvpair3linux:arm64.
Preparing to unpack .../3-libnvpair3linux_2.4.1-1~bpo13+1~rpt1_arm64.deb ...
Unpacking libnvpair3linux:arm64 (2.4.1-1~bpo13+1~rpt1) ...
Selecting previously unselected package libuutil3linux:arm64.
Preparing to unpack .../4-libuutil3linux_2.4.1-1~bpo13+1~rpt1_arm64.deb ...
Unpacking libuutil3linux:arm64 (2.4.1-1~bpo13+1~rpt1) ...
Selecting previously unselected package libzfs7linux:arm64.
Preparing to unpack .../5-libzfs7linux_2.4.1-1~bpo13+1~rpt1_arm64.deb ...
Unpacking libzfs7linux:arm64 (2.4.1-1~bpo13+1~rpt1) ...
Selecting previously unselected package libzpool7linux:arm64.
Preparing to unpack .../6-libzpool7linux_2.4.1-1~bpo13+1~rpt1_arm64.deb ...
Unpacking libzpool7linux:arm64 (2.4.1-1~bpo13+1~rpt1) ...
Selecting previously unselected package zfsutils-linux.
Preparing to unpack .../7-zfsutils-linux_2.4.1-1~bpo13+1~rpt1_arm64.deb ...
Unpacking zfsutils-linux (2.4.1-1~bpo13+1~rpt1) ...
Selecting previously unselected package zfs-zed.
Preparing to unpack .../8-zfs-zed_2.4.1-1~bpo13+1~rpt1_arm64.deb ...
Unpacking zfs-zed (2.4.1-1~bpo13+1~rpt1) ...
Setting up libnvpair3linux:arm64 (2.4.1-1~bpo13+1~rpt1) ...
Setting up gawk (1:5.2.1-2+b2) ...
Setting up dkms (3.2.2-1~deb13u1) ...
Setting up zfs-dkms (2.4.1-1~bpo13+1~rpt1) ...
Loading new zfs/2.4.1 DKMS files...
Building for 6.12.75+rpt-rpi-2712 and 6.12.75+rpt-rpi-v8

Building initial module zfs/2.4.1 for 6.12.75+rpt-rpi-2712
The kernel is built without module signing facility, modules won't be signed

Running the pre_build script..................... done.
Building module(s)................................. done.
Running the post_build script... done.
Installing /lib/modules/6.12.75+rpt-rpi-2712/updates/dkms/zfs.ko.xz
Installing /lib/modules/6.12.75+rpt-rpi-2712/updates/dkms/spl.ko.xz
Running depmod... done.

Building initial module zfs/2.4.1 for 6.12.75+rpt-rpi-v8
The kernel is built without module signing facility, modules won't be signed

Running the pre_build script..................... done.
Building module(s)................................. done.
Running the post_build script... done.
Installing /lib/modules/6.12.75+rpt-rpi-v8/updates/dkms/zfs.ko.xz
Installing /lib/modules/6.12.75+rpt-rpi-v8/updates/dkms/spl.ko.xz
Running depmod... done.
Setting up libuutil3linux:arm64 (2.4.1-1~bpo13+1~rpt1) ...
Setting up libzpool7linux:arm64 (2.4.1-1~bpo13+1~rpt1) ...
Setting up libzfs7linux:arm64 (2.4.1-1~bpo13+1~rpt1) ...
No diversion 'diversion of /lib/aarch64-linux-gnu/libzfs_core.so.3
 to /lib/aarch64-linux-gnu/libzfs_core.so.3.usr-is-merged by libzfs6linux', non
e removed.
No diversion 'diversion of /lib/aarch64-linux-gnu/libzfs_core.so.3.0.0
 to /lib/aarch64-linux-gnu/libzfs_core.so.3.0.0.usr-is-merged by libzfs6lin
ux', none removed.
Setting up zfsutils-linux (2.4.1-1~bpo13+1~rpt1) ...
insmod /lib/modules/6.12.75+rpt-rpi-2712/updates/dkms/spl.ko.xz
insmod /lib/modules/6.12.75+rpt-rpi-2712/updates/dkms/zfs.ko.xz
Created symlink '/etc/systemd/system/zfs-import.target.wants/zfs-import-cache.service' →
 '/usr/lib/systemd/system/zfs-import-cache.service'.
Created symlink '/etc/systemd/system/zfs.target.wants/zfs-import.target' →
 '/usr/lib/systemd/system/zfs-import.target'.
Created symlink '/etc/systemd/system/zfs-mount.service.wants/zfs-load-module.service' →
```

```
 '/usr/lib/systemd/system/zfs-load-module.service'.
Created symlink '/etc/systemd/system/zfs.target.wants/zfs-load-module.service' →
 '/usr/lib/systemd/system/zfs-load-module.service'.
Created symlink '/etc/systemd/system/zfs.target.wants/zfs-mount.service' →
 '/usr/lib/systemd/system/zfs-mount.service'.
Created symlink '/etc/systemd/system/zfs.target.wants/zfs-share.service' →
 '/usr/lib/systemd/system/zfs-share.service'.
Created symlink '/etc/systemd/system/zfs-volumes.target.wants/zfs-volume-wait.service' →
 '/usr/lib/systemd/system/zfs-volume-wait.service'.
Created symlink '/etc/systemd/system/zfs.target.wants/zfs-volumes.target' →
 '/usr/lib/systemd/system/zfs-volumes.target'.
Created symlink '/etc/systemd/system/multi-user.target.wants/zfs.target' →
 '/usr/lib/systemd/system/zfs.target'.
zfs-import-scan.service is a disabled or a static unit, not starting it.
Processing triggers for initramfs-tools (0.148.3+rpt2) ...
update-initramfs: Generating /boot/initrd.img-6.12.75+rpt-rpi-v8
'/boot/initrd.img-6.12.75+rpt-rpi-v8' -> '/boot/firmware/initramfs8'
update-initramfs: Generating /boot/initrd.img-6.12.75+rpt-rpi-2712
'/boot/initrd.img-6.12.75+rpt-rpi-2712' -> '/boot/firmware/initramfs_2712'
Processing triggers for libc-bin (2.41-12+rpt1+deb13u2) ...
Processing triggers for man-db (2.13.1-1) ...
Setting up zfs-zed (2.4.1-1~bpo13+1~rpt1) ...
Created symlink '/etc/systemd/system/zed.service' →
 '/usr/lib/systemd/system/zfs-zed.service'.
Created symlink '/etc/systemd/system/zfs.target.wants/zfs-zed.service' →
 '/usr/lib/systemd/system/zfs-zed.service'.
```