FreeBSD® guides

# An Introduction to ZFS

BY DREW GURKOWSKI

ZFS combines the roles of volume manager and independent file system into one, giving multiple advantages over a stand-alone file system. It is renowned for speed, flexibility, and, most importantly, taking great care to prevent data loss. While many traditional file systems had to exist on a single disk at a time, ZFS is aware of the underlying structure of the disks and creates a pool of available storage, even on multiple disks. The existing file system will grow automatically when extra disks are added to the pool, immediately becoming available to the file system.


## Getting Started

FreeBSD can mount ZFS pools and datasets during system initialization. To enable it, add this line to /etc/rc.conf:

```
zfs_enable="YES"
```

Then start the service:

```
# service zfs start
```

## Identify Hardware

Before setting up ZFS, identify the device names of the disk associated with the system. A quick way of doing this is with:

```
# egrep 'da[0-9]|cd[0-9]' /var/run/dmesg.boot
```

The output should identify the device names, examples throughout the rest of this guide will use the default SCSI names: da0, da1, and da2. If the hardware differs, make sure to use the correct device names instead.

## Creating a Single Disk Pool

To create a simple, non-redundant pool using a single disk device:

```
# zpool create example /dev/da0
```

To create files for users to browse within the pool:

```
# cd /example
# ls
# touch testfile
```

The new file can be viewed using ls:

```
# ls -al
```

We can already start using more advanced ZFS features and properties. To create a dataset within the pool with compression enabled:

```
# zfs create example/compressed
# zfs set compression=on example/compressed
```

The example/compressed dataset is now a ZFS compressed file system.
Disable compression with:

```
# zfs set compression=off example/compressed
```

To unmount a file system, use zfs umount and then verify with df:

```
# zfs umount example/compressed
# df
```

Verify that example/compressed is not included as a mounted file under the output.
The file system can be re-mounted with zfs:

```
# zfs mount example/compressed
# df
```

With the file system mounted, the output should include a line similar to the one below:

```
example/compressed 17547008 0 17547008 0% /example/compressed
```

ZFS datasets are created just like any other file system, the following example creates a new file system called data:

```
# zfs create example/data
```

Use df to see the data and space usage (some of the output has been removed for clarity).

```
# df
                                    . . .
example/compressed   17547008        0 17547008     0%    /example/compressed
example/data         17547008        0 17547008     0%    /example/data
```

Because these file systems are built on ZFS, they draw from the same pool for storage. This eliminates the need for volumes and partitions that other file systems rely on.

To destroy the file systems and then the pool that is no longer needed:

```
# zfs destroy example/compressed
# zfs destroy example/data
# zpool destroy example
```

## RAID-Z

RAID-Z pools require three or more disks but offer protection from data loss if a disk were to fail. Because the ZFS pools can use multiple disks, support for RAID is inherent in the design of the file system

To create a RAID-Z pool, specifying the disks to add to the pool:

```
# zpool create storage raidz da0 da1 da2
```

With the zpool created, a new file system can be made in that pool:

```
# zfs create storage/home
```

Enable compression and store an extra copy of directories and files:

```
# zfs set copies=2 storage/home
# zfs set compression=gzip storage/home
```

A RAID-Z pool is a great place to store crucial system files, such as the home directory for users.

```
# cp -rp /home/* /storage/home
# rm -rf /home /usr/home
# ln -s /storage/home /home
# ln -s /storage/home /usr/home
```

File system snapshots can be created to roll back to later, the snapshot name is marked in yellow and can be whatever you want:

```
# zfs snapshot storage/home@11-01-22
```

ZFS creates snapshots of a dataset, allowing users to back up a file system for roll backs or data recovery in the future.

```
# zfs rollback storage/home@11-01-22
```

To list all available snapshots, zfs list can be used:

```
# zfs list -t snapshot storage/home
```

## Recovering RAID-Z

Every software RAID has a method of monitoring its state. View the status of RAID-Z devices using:

```
# zpool status -x
```

If all pools are Online and everything is normal, the message shows:

```
all pools are healthy
```

If there is a problem, perhaps a disk being in the Offline state, the pool state will look like this:

```
  pool: storage
 state: DEGRADED
status: One or more devices has been taken offline by the administrator.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Online the device using ‘zpool online’ or replace the device with
        'zpool replace'.
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        storage     DEGRADED     0     0     0
          raidz1    DEGRADED     0     0     0
            da0     ONLINE       0     0     0
            da1     OFFLINE      0     0     0
            da2     ONLINE       0     0     0

errors: No known data errors
```

"OFFLINE" shows the administrator took da1 offline using:

```
# zpool offline storage da1
```

Power down the computer now and replace da1. Power up the computer and return da1 to the pool:

```
# zpool replace storage da1
```

Next, check the status again, this time without -x to display all pools:

```
# zpool status storage
 pool: storage
 state: ONLINE
 scrub: resilver completed with 0 errors on Fri Nov 4 11:12:03 2022
config:

  NAME        STATE     READ WRITE CKSUM
  storage     ONLINE       0     0     0
    raidz1    ONLINE       0     0     0
      da0     ONLINE       0     0     0
      da1     ONLINE       0     0     0
      da2     ONLINE       0     0     0

errors: No known data errors
```

## Data Verification

ZFS uses checksums to verify the integrity of stored data, these data checksums can be verified (which is called scrubbing) to ensure integrity of the storage pool:

```
# zpool scrub storage
```

Only one scrub can be run at a time due to the heavy input/output requirements. The length of the scrub depends on how much data is stored in the pool. After scrubbing completes, view the status with zpool status:

```
# zpool status storage
  pool: storage
 state: ONLINE
  scrub: scrub completed with 0 errors on Fri Nov 4 11:19:52 2022
config:


   NAME         STATE     READ WRITE CKSUM
   storage      ONLINE       0     0     0
     raidz1     ONLINE       0     0     0
       da0      ONLINE       0     0     0
       da1      ONLINE       0     0     0
       da2      ONLINE       0     0     0


errors: No known data errors
```

Displaying the completion date of the last scrubbing helps decide when to start another. Routine scrubs help protect data from silent corruption and ensure the integrity of the pool.

## ZFS Administration

ZFS has two main utilities for administration: The zpool utility controls the operation of the pool and allows adding, removing, replacing, and managing disks. The zfs utility allows creating, destroying, and managing datasets, both file systems and volumes.

While this introductory guide won't cover ZFS administration, you can refer to zfs(8) and zpool(8) for other ZFS options.

## Further Resources

While both the non-redundant and RAID-Z pools created using this guide will work in most use cases, more complex or specialized systems may require further ZFS management and setup. This guide barely scrapes the surface of what can be done using ZFS as it is an extremely powerful and customizable file system. The OpenZFS wiki has expansive documentation on installation, ZFS system administration, and manual pages. If tuning is required due to system architecture, ZFS tuning guides can be found on both the OpenZFS and FreeBSD wiki pages.

---

**DREW GURKOWSKI**, FreeBSD Foundation