Introduction to ZFS

2019-03 Edition Revision 1d)

Maintained by Ericloewe of the FreeNAS forums

ZFS is designed to be the last filesystem you will ever need.

Besides making life easier for Marketing, it also means that old knowledge of both filesystems and volume managers often does not apply.

The goal of this document is to provide a brief introduction to ZFS, for users unfamiliar with it. Important concepts will be discussed, but ZFS has much more to offer than this document aims to explain. In particular, this document will focus on ZFS as a FreeNAS user might experience it. At the end of this document, the Further Reading section contains references to recommended material.

This document is bound to change over time, as ZFS evolves. The updated version is available in the FreeNAS forums' Resources section, where changelogs can also be found.

Distribution of this document is discouraged, in order to avoid the proliferation of outdated versions. Please link to the Resource page instead of distributing this document. The most recent version can always be found there.

Table of Contents

What is ZFS?

ZFS is a filesystem. It is also a volume manager. That means that it handles storage all the way between disk and the OS, replacing both traditional filesystems like FAT, NTFS, UFS or ext4 *and* traditional RAID solutions like Hardware RAID cards, Windows Home Server's Drive Extender[1], Windows 10's Storage Spaces, FreeBSD's GEOM or Linux's Logical Volume Manager.

You might have heard about Oracle ZFS. ZFS was developed at Sun Microsystems by Matt Ahrens and Jeff Bonwick. It was open-sourced and ported to other operating systems. After acquiring Sun, Oracle took the unusual step of closing its internal development of the OpenSolaris operating system, including ZFS. The existing code was still open-source, so most of the ZFS team left Oracle and the OpenZFS[2] project was created, to continue open-source development of ZFS.

As of this writing, OpenZFS is available and mature on the following operating systems:

- Illumos (the open-source continuation of OpenSolaris)
- FreeBSD
- macOS
- Linux

Additionally, NetBSD is working towards updating its version of ZFS to catch up with OpenZFS development. A Windows port is also in the early stages of work, potentially turning ZFS into a proper universal file system.

[1] As a Windows Home Server exile from both Vanilla and 2011, I understand your pain.

[2] Though officially "OpenZFS", this document uses "ZFS" and "OpenZFS" interchangeably. Oracle ZFS is mentioned explicitly, if necessary.

What can ZFS do?

Above all, ZFS protects your data. To quote Allan Jude, coauthor of FreeBSD Mastery: ZFS:

*Disks are the physical manifestation of storage. Disks are evil. They lie about their characteristics and layout, they hide errors, and they fail in unexpected ways. ZFS means no longer having to fear that your disks are secretly plotting against you. Yes, your disks **are** plotting against you, but ZFS exposes their treachery and puts a stop to it.*

To this end, ZFS is completely Copy-on-Write (CoW) and checksums all data and metadata. Checksums are kept separate from their blocks, so ZFS can verify that data is valid *and* that is it the correct data and not something else that your evil disks or storage controller sent your way.

When ZFS detects an error, it immediately corrects it if possible. This can be achieved using mirrors (arbitrarily wide) or RAIDZ1, RAIDZ2 and RAIDZ3. In traditional RAID terminology, mirrors would be RAID1, RAIDZ1 is roughly equivalent to RAID5, RAIDZ2 to RAID6. RAIDZ3 extends this concept beyond what hardware RAID offers by one more disk. ZFS pools can be made arbitrarily large by striping across these internally-redundant groups of disks.

ZFS also attempts to make an administrator's life easy, with features such as snapshots, replication, transparent compression, quotas and reservations, and much more. This also extends to compatibility – ZFS pools created on any system can be imported on any other system[3] and they are not tied to any disk controller[4].

Repair tools, such as fsck, do not exist for ZFS as it is supposed to always be consistent.

[3] As long as all features used by the pool are supported. See the Feature Flags subsection for more details.
[4] As long as the disk controller didn't tie the disk to itself, as is typical of hardware RAID controllers

Structure of ZFS

From the user's perspective, ZFS has two layers – the Dataset and Snapshot Layer and the underlying pool, which provides the actual storage. We'll discuss the pool first:

ZFS pools

A ZFS volume is called a pool. Some people like to needlessly add the character 'z' at the beginning of ZFS-related nouns that do not start with 'z'. It is just a pool – save your time and do not say "my zpool".

A pool contains one or more *vdevs*, short for *virtual device*, and is striped across them. That means that data does not overlap across them and the failure of one vdev implies data loss. Again, **if one vdev is lost, the pool fails**.

So, it is necessary to ensure that no vdev fails. In other words, ZFS implements redundancy at the vdev layer[5].

Types of vdev

ZFS currently supports three types of vdev: single disk, mirror and RAIDZ1/2/3. Disks can be added or removed from mirrors, so it is possible to turn a single disk into a mirror and vice-versa. RAIDZ vdevs are immutable.

Single disks

Running a single disk as a vdev is a very bad idea in most situations. Data is at risk. Fortunately, it can easily be transformed into a mirror vdev.

Mirrors

Mirrors store the same data on multiple disks, much like RAID1. They can be made arbitrarily wide, but 2-wide is typical. 3-wide is sometimes seen, when RAIDZ2 or 3 are not desirable, but a two-disk redundancy is desired. In an n-wide mirror, n-1 disks can fail with no data loss.

RAIDZ1, RAIDZ2 and RAIDZ3

RAIDZ is similar in concept to hardware RAID5 and RAID6, but solves many of the flaws inherent in such setups. In particular, RAIDZ does not have a write hole.

A RAIDZn vdev can tolerate up to n disk failures without data loss.

RAIDZ vdevs do not impose any strict limitations on vdev width, but performance is abysmal for extremely wide RAIDZ vdevs, as the whole vdev has approximately the same IOPS as a single constituent disk. Generally, 10 disks is a good rule of thumb for the maximum width of a vdev.

For many workloads, RAIDZ is slower than mirrors due to the limited IOPS potential. CPU complexity also means that RAIDZ3 is the slowest of the three and RAIDZ1 the fastest, though this is not much of a concern on modern hardware.

As a rule of thumb, file sharing applications are fine with RAIDZ performance and block device applications (such as iSCSI) tend to require sets of mirrors.

[5] Further redundancy can be set at the dataset layer, using copies=2 or copies=3. If possible, copies will be written to different vdevs. This is not a replacement for reliable vdevs.

The most important limitation of RAIDZ is that vdevs are **immutable**. You cannot add or remove disks from a RAIDZ vdev, though work is ongoing to allow for expanding RAIDZ vdevs (but *not* removing disks). It is best to plan ahead with this limitation in mind.

Furthermore, RAIDZ1 is often considered too unreliable, as it can only survive one disk failure.

Mixing vdev types

It is possible to mix vdev types in one pool, but not recommended. Performance may suffer, and such configurations often see less testing.

When using RAIDZ vdevs, it is also a good idea to keep them at the same width and of the same type.

Practical examples

Good:

- One two-way mirror
- Two two-way mirrors
- One RAIDZ2, four-wide
- Two RAIDZ2, six-wide
- Eight three-way mirrors

Not horrible:

- RAIDZ1, five-wide
- RAIDZ2, four-wide + RAIDZ2, eight-wide
- RAIDZ3, seven-wide + mirror pair
- 24-wide mirror
- RAIDZ2, 14-wide

Not good at all:

- Single disk
- RAIDZ3, 11-wide + single disk
- 12 single disks
- RAIDZ3, 100-wide

Datasets

A dataset is a ZFS filesystem. It can be viewed as a construct that combines the advantages of a traditional directory (datasets form a tree, share space from the underlying pool, etc.) with the advantages of a disk partition (datasets can be treated separately and can be used to partition storage) and adds unique management capabilities.

Every ZFS pool has a top-level dataset, named after the pool. From there, an arbitrary number of datasets can be created as children of an existing dataset. Datasets are typically the unit of management – in other words, ZFS properties apply to datasets. These include features such as compression, checksums, quotas and reservations[6]. Snapshots and ZFS replication, explained below, also operate on entire datasets.

The rule of thumb is to use datasets instead of plain directories for data that is treated differently: Different owner, different snapshot schedule, different compression, different quota, etc. When in doubt, it is typically better to use more datasets rather than fewer.

Zvols

A zvol is a special type of dataset: instead of presenting a POSIX filesystem, it presents a virtual block device, backed by ZFS. Zvols generally provide storage for some other filesystem or application that requires direct disk access, notably VMs.

Most dataset properties also apply to zvols, and snapshots and replication operate identically.

[6] This also applies to encryption, on versions of ZFS that support it.

Using ZFS

ZFS has a number of features that distinguish it from traditional filesystems. These will be presented in this section.

Scrub

An important part of ZFS' data integrity guarantee is the scrub – this is the process whereby ZFS reads all data on a pool and checks that it is still correct, correcting it if necessary and possible, either from parity (mirrors and RAIDZ) or copies[7].

It is extremely important to regularly scrub a ZFS pool. To this end, FreeNAS allows setting up a scrub schedule for this to happen automatically. A scrub is a fairly intensive and prolonged operation, which should be scheduled to minimize the impact from the resulting loss of performance.

A good rule of thumb is to scrub a pool every two weeks or so. Error counts, including those found during scrubs, are logged by ZFS (although not persistently) and should be used to identify disks which are beginning to fail.

Resilver

ZFS calls the process of replacing a disk "resilver". It is fundamentally the same process as a scrub.

It is possible to replace a disk without first removing the old disk, thereby maintaining redundancy during the resilver. This is preferable, even if the old disk has some errors, as it minimizes risk.

It is also possible to resilver multiple disks at once – this is particularly useful when growing a vdev by replacing its disks with larger ones. This is limited only by the available disk connections.

Snapshots

One of ZFS' most useful features are snapshots. A snapshot is the state of a dataset at the time the snapshot was taken. Since ZFS is copy-on-write, snapshots are "free", in the sense that it is possible to have essentially arbitrary numbers of snapshots with zero performance implication, aside from the space taken up by the snapshots[8].

Once a snapshot exists, it is possible to view the contents of the dataset at the time the snapshot was taken, as well as rollback to the snapshot to revert any changes made to the dataset since it was taken. Snapshots can also be cloned. This process creates a new dataset based off of the snapshot, allowing for multiple datasets derived from a single one, storing only the changes since the clone was made. Snapshots and clones are used to handle boot environments on systems that boot from ZFS, such as FreeNAS, allowing for easy rollback of updates that go wrong.

Crucially, snapshots effectively mitigate both ransomware attacks and many forms of user error. If an important file is deleted, it can simply be restored from the snapshot. If the whole dataset

[7] Critical metadata is stored with three copies by default. File metadata is stored with two copies by default. User data is stored in a single copy by default. In the latter cases, these can be overridden for up to three copies.

[8] Listing thousands of snapshots and manually managing them tends to be a slow process, imposing a limit on what is a reasonable number of snapshots to have.

is encrypted by ransomware through a client machine connected over the network, it can simply be rolled back to the last snapshot before the attack.

For snapshots to be truly useful, they must happen regularly. FreeNAS allows for snapshot tasks to be configured and will regularly take snapshots and delete them after their set expiration date. One common way of optimizing this schedule is to have several tasks for one dataset – for instance, a task that snapshots a dataset every five minutes but whose snapshots expire after a day and a second task that takes a snapshot every hour, but which will be kept for a month.

Snapshots are also the basis of ZFS replication, discussed below.

ZFS replication

Replication, often called ZFS send/receive, after the `zfs send` and `zfs receive` commands, is an excellent tool for many situations. It allows sending a snapshot of a dataset, including its properties and child datasets, to another ZFS dataset. On the receiving side, the received snapshot of a dataset is effectively a copy of the original dataset as of the snapshot that was sent.

Replication is extremely efficient, as ZFS knows exactly what data changed between snapshots. This means that there is no need to compare both sides to determine what is new (as tools like rsync do) and only these changes need to be sent.

In practice, the most frequent use of replication is to maintain backups. These are generally done over the network, to a second machine. After initially replicating the desired dataset and its snapshots, it is possible to only send new snapshots to the backup server, keeping network usage low.

However, replication is useful in many other scenarios. Some dataset properties, such as compression, only apply to data written after the setting was changed. By replicating the data to a new dataset, it is possible to re-write existing data with the new properties. Replication can also be used to migrate data to a new pool on the same server, or to "synchronize" data between machines – the use of quotes is deliberate, as replication is not bidirectional, making this setup more complicated and not very suitable to daily synchronization.

On OpenZFS implementations that support encryption, it is possible to send an encrypted dataset to a different server. Since basic metadata and checksums are not encrypted, the receiving server can keep the data safe by scrubbing it regularly and restoring (the encrypted version) from parity – all without being able to read the data[9].

Compression

ZFS allows for transparent compression. On most modern systems, compression is effectively free and may provide significant reductions in data size. FreeNAS enables LZ4 compression by default, but other algorithms are available for specific requirements. Disabling compression for uncompressible data, such as media (particularly lossy formats), is possible, but not strictly necessary.

Recent versions of ZFS use compressed ARC[10], which keeps blocks compressed in memory instead of decompressing them. With compressible data, this means that significantly more data

[9] Some metadata may leak information, so care must still be taken. Notably, dataset names are not encrypted.

[10] The Adaptive Replacement Cache is ZFS' read cache.

can be kept in RAM, improving performance – decompressing data is still faster than reading from disk, and a small buffer of uncompressed blocks is kept. This makes the use of compression even more advantageous.

Free space and performance

ZFS' features come at a cost: Free space and performance are tied together. ZFS' performance drops as free space disappears, so it is necessary to keep a decent chunk of the pool empty. Beyond a certain point (around 95% full), ZFS switches from the fast allocator to the efficient allocator, which is much slower. If a pool is completely filled, it may become completely unusable.

For file storage, a good rule of thumb is keep pools below 80% full. For block storage, this figure can be 50% – and that already allows for a significant reduction in performance versus an empty pool.

Hardware for ZFS

ZFS features rely on direct access to disks. This means that running ZFS on top of hardware RAID is counterproductive and may be actively harmful to data integrity and performance. On the other hand, ZFS does not care how disks are attached[11] – it is possible to freely move disks between controllers; have pools made up of disks attached to multiple, different controllers; freely swap disks around on the same controller or to a different controller; etc.

ZFS actively leverages system RAM to improve performance – more RAM allows for more data and metadata to be cached in memory. ECC memory is highly recommended. Although ZFS is not more susceptible to damage caused by memory errors than traditional filesystems, ECC RAM prevents these errors from damaging data and is standard practice on server hardware.

Details on what hardware is appropriate for use with FreeNAS can be found in the documents referenced in the Further Reading section. Although they are written with FreeNAS in mind, the same general principles apply to any server running ZFS.

[11] As long as the Operating System supports the disk and/or controller reliably.

Advanced Topics

ZFS is far too complex to fully describe in this document. This section focuses on briefly presenting some important topics that users may want to take into account when dealing with more complex scenarios.

More information on all these topics can be found in the Further Reading section.

Write Caching

ZFS caches writes in memory. For sync writes[12], which demand that data be written to nonvolatile storage before completing, ZFS uses the so-called ZFS Intent Log (ZIL), stored on the pool. The ZIL can be slow, so it can be offloaded to a dedicated Separate LOG device (SLOG), which should be a very low-latency SSD with power loss protection.

Read Caching

For reads, ZFS uses the ARC, an in-memory cache. It is a combination most recently used and most frequently used cache. Although more physical memory is often preferred to improve caching, a second option is available: the Level 2 Arc (L2ARC), which is a cache on dedicated SSDs. L2ARC is particularly useful when it is not viable to add physical memory.

It is important to note that L2ARC is not a magical, universal performance booster. The contents of the L2ARC are referenced in L1ARC and thus take up precious RAM, at around 25 MB of ARC per 1 GB of L2ARC. The practical implication of this is that adding an L2ARC to a system with a less-than-ideal amount of RAM can *decrease* performance, as potential ARC hits that could have been served from RAM must now be served from disk – regardless of how fast the SSD is, it is still slower than DRAM.

The ARC will automatically expand to use as much free system memory as possible, as empty RAM is wasted RAM, and automatically resize down if asked to by the Operating System.

Feature Flags

Instead of traditional version numbers, ZFS uses so-called Feature Flags to determine compatibility with certain features. These do not apply to all features, as some features (like compressed ARC) do not require changes to ZFS' on-disk format. Those that do receive a feature flag. Some feature flags maintain compatibility when not in use, even after enabling them, some allow for read-only backwards compatibility and some permanently break backwards compatibility.

Deduplication

ZFS allows for transparent deduplication of user data. While this sounds like a fantastic feature, it is not. It is very slow, very RAM-hungry (a minimum of 5 GB of RAM per TB of storage should be provided[13]; insufficient memory will make pool import impossible until enough memory is provided) and not really *that* useful for most users.

[12] Sync writes are frequently encountered when dealing with block storage, databases and when sharing datasets via SMB with macOS clients.

[13] Details on this requirement can be found in *FreeBSD Mastery: Advanced ZFS*, as listed in the Further Reading section.

The `recordsize` property

ZFS uses the `recordsize` property to define the maximum size of blocks. It applies per dataset, so different datasets can have different values for this property. Since ZFS uses variable-sized blocks, this setting only affects the maximum size of blocks.

Although a larger `recordsize` may seem appealing at first, it is not appropriate for all situations. Larger blocks mean that fewer blocks need to be written, in turn reducing the amount of metadata required. Larger blocks, however, are more prone to write amplification, reducing performance and taking up more space in snapshots. Therefore, this property should be tuned down when dealing with databases, block storage and similar workloads; and tuned up when dealing with large, immutable files, such as video.

Further Reading

The FreeNAS User Guide
The ZFS man pages, `zfs(8)` and `zpool(8)`

For users of FreeNAS, the User Guide is the go-to reference for basically all operations. Users interacting directly with ZFS on other systems should consult the man pages on their system, which will match their installed version.

FreeBSD Mastery: ZFS and *FreeBSD Mastery: Advanced ZFS*,
by Allan Jude and Michael W. Lucas

These two books are essential to anyone wanting to learn more about ZFS and are highly recommended by pretty much anyone who has read them. Matt Ahrens, co-creator of ZFS praises these books by saying "Thanks for making ZFS *knowable* by everyone." Jeff Bonwick, also co-creator of ZFS, states "Thank you for doing this… now I do not have to."

The FreeBSD Handbook section on ZFS

The FreeBSD Handbook is an excellent resource for learning how to directly interact with ZFS. Although this is not necessary when using FreeNAS, it is always useful to understand what is being done under the hood. It is also very useful for other systems, as ZFS itself has the same administrative interface everywhere.

ZFS Feature Flags in FreeNAS, by Ericloewe

This resource explains the details of ZFS Feature Flags and keeps track of what Feature Flags are supported in each recent FreeNAS version.

FreeNAS Hardware Recommendations Guide, by Ericloewe
A Complete Guide to FreeNAS Hardware Design Part I, Part II, Part III and Part IV,
by Josh Paetzel

Although only tangentially related to ZFS, it is important to choose appropriate hardware for a server. While the recommendations made in these resources focus on FreeNAS, they are mostly applicable to other systems running ZFS.

Sync writes, or: Why is my ESXi NFS so slow, and why is iSCSI faster? By jgreco

This resource explains important details about sync writes and how ZFS deals with them, from the user's perspective, as well as how to deal with them.