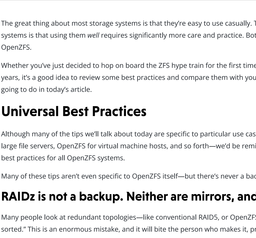
OpenZFS Storage Best Practices and Use Cases – Part 1: Snapshots and Backups

NOVEMBER 14, 2023

In a new series of articles on OpenZFS, we'll go over some universal best practices for OpenZFS storage, and then dig into several common use cases along with configuration tips and best practices specific to those use cases.



The great thing about most storage systems is that they're easy to use casually. The terrible thing about most storage systems is that using them *well* requires significantly more care and practice. Both of these canards hold true for OpenZFS.

Whether you've just decided to hop on board the ZFS hype train for the first time, or you've been successfully using it for years, it's a good idea to review some best practices and compare them with your own—and that's exactly what we're going to do in today's article.

Universal Best Practices

Although many of the tips we'll talk about today are specific to particular use cases—OpenZFS for databases, OpenZFS for large file servers, OpenZFS for virtual machine hosts, and so forth—we'd be remiss if we didn't open with some tips and best practices for all OpenZFS systems.

Many of these tips aren't even specific to OpenZFS itself—but there's never a bad time to go over Storage 101!

RAIDz is not a backup. Neither are mirrors, and neither is DRAID.

Many people look at redundant topologies—like conventional RAID5, or OpenZFS RAIDz2—and think "well, that's backup sorted." This is an enormous mistake, and it will bite the person who makes it, probably much sooner than they expected.

Redundant topologies are important for two things in OpenZFS: improving system uptime and allowing the system to automatically heal (self-repair) corrupt data and metadata.

Although this does allow the system to survive specific types of hardware failure (most notably, individual drive failures) it does not protect the system from common failure modes that proper backup methods should: catastrophic hardware failure, catastrophic *environment* failure, human user error, and human *administrator* error.

If your server catches on fire, RAIDz won't save you. If a tornado destroys your facility, RAIDz won't help. If a user deletes a file, RAIDz won't bring it back. If an administrator destroys the entire pool, RAIDz *absolutely* won't help you.

Snapshots are only *part* of a good backup strategy.

OpenZFS snapshots are, frankly, amazing. They can be taken instantaneously, capture the entire contents of one or several datasets with atomic precision, and do not negatively impact system performance like LVM snapshots do.

By themselves, however, snapshots are still not a complete backup. Taking a snapshot can protect your system from human user error, and even user malice. A snapshot still cannot *entirely* protect your system from human administrator error, however, and it's no protection whatsoever from administrator-level *malice*.

Snapshots also do not protect a system from catastrophic hardware or environmental failures—at least, locally stored snapshots don't. That's where OpenZFS snapshot *replication* comes in. Using OpenZFS replication, an administrator can not only preserve the local storage system's state but replicate It to an entirely separate machine.

The policies and procedures surrounding both replication itself, and the replication target, are what can extend OpenZFS snapshotting into a viable, full-service backup strategy. If the replication target is on separate hardware, the system's data is protected from catastrophic hardware failure. If the replication target is offsite, the system's data is protected from catastrophic environmental failure.

Protecting the system from human *administrator*-level error or malice is, of course, trickier. In a sufficiently paranoid environment, the replication target can be kept firewalled entirely away from the production environment, and from any other security concerns. In particularly high-security environments, the backup system might even require direct physical access to retrieve its data.

Snapshots are only helpful if they've already been taken.

By the time you need a snapshot, it's too late to take a snapshot. While it's great to get in the habit of manually taking snapshots before doing risky things—eg **zfs snapshot mypool/myset@beforeidoadumbthing**—it's easy to forget to take a preliminary snapshot, and it's unfortunately easy not to realize you're doing something dumb before you've already done it.

For this reason, *every* OpenZFS storage system should have an automated snapshot orchestration system. Such a system can be configured to automatically take snapshots at regular intervals, and also destroy the oldest snapshots at regular intervals (to keep the system from becoming choked with immutable data).

Two particularly common snapshot systems in use today are Sanoid and pyznap. Both systems allow the administrator to define snapshot policies, which in turn control how frequently snapshots are taken and under what conditions they become "stale" (meaning they can and should be automatically destroyed to reclaim storage free space).

If this sounds intimidating, it shouldn't. Here is a sample config stanza from **sanoid.conf** on system:

```
    [rpool]
            use_template = production
            recursive = zfs
```

This recursively snapshots an entire ZFS-on-root system (**rpool** is the pool name, which means it's also the pool's root dataset) using the production template. What does that mean? Well, you can scroll down a bit further to see the included template named **production**:

```
[template_production]
        frequently = 0
        hourly = 36
        daily = 30
        monthly = 3
        yearly = 0
        autosnap = yes
        autoprune = yes
```

We take hourly, daily, and monthly snapshots. We keep 36 hourlies, 30 dailies, and 3 monthlies at all times. When we have more of any of these types of snapshots than policy allows—*and* when the oldest snapshots are as old as you'd expect them to be, when you have that many—the system automatically destroys the stale snapshots for you.

That template is included by default, along with a couple others—but you can also define your own templates, override template settings directly in the module config stanza, or edit the templates themselves.

In short, it's *easy* to have your system automatically manage snapshots for you, which in turn makes it near-criminally irresponsible not to!

Did we mention backup? Backup is important.

Without reliable, high performance filesystem-level replication, backups are incredibly difficult to get right. You need to worry about backing up all the right things, about the consistency of the backup (if it takes an hour to run your backup, the files *in* the backup should not be changing while that backup runs), about the impact on system performance while the backup is happening, and—most crucial of all—about whether you can actually *restore* your backup later, should you need it.

Luckily, this is an article about OpenZFS best practices specifically—which means backups are extremely easy. Just replicate the entire dataset or group of datasets that your important stuff lives in off to another machine, far enough separated from the first to make it less vulnerable to catastrophes which might wipe out your source system.

The sanoid snapshot orchestration system we mentioned in the last section includes a companion app, syncoid, which makes OpenZFS replication as simple as *rsync*. For example, to back up the entire ZFS on root system we referenced in the last section, we can use a single command:

```
root@backupsystem:~# syncoid -r root@sourcesystem:rpool
backuppool/sourcessystem/rpool
```

That's it. This single command will replicate every last bit of data on the source system, *including individual snapshots*, to the backup system. Running the same command again will bring any newer snapshots from the source into the target, and do so extremely rapidly—in some cases, hundreds or even thousands of times faster than *rsync* can, and with drastically lower performance impact on the system as well.

We don't have to worry about consistency of the replicated data, because it's actually *all* snapshot-based: each snapshot covers an entire dataset (or datasets) and is point-in-time consistent. It's also cryptographically verified for correctness all the way down at the individual block level. It's also guaranteed to be *complete*—if replication breaks halfway through a snapshot, the partial snapshot will not show up in a **zfs list** command on the target system.

Put all this together, and it means backup verification just got incredibly simple: if **zfs list -t snapshot backuppool/sourcesystem/rpool** shows a snapshot named **@2023-01-01_23:59:00_daily**, you know for a fact that you've got the complete and consistent set of data from that exact moment in time on the source system.

Universal best practices that *aren't* about backups

At this point, you're probably wondering if we've got any universal best practices for you that don't amount to "back up your system." Of course! They're just not as important as the backup-related ones, in much the same way that choosing what to order for dinner is less important than making it to the restaurant in one piece.

First, you should enable compression, which is off by default. The right compression algorithm for your workload will not only save space, it will significantly improve performance. OpenZFS offers several compression algorithms, so let's talk about the typical top three choices:

LZ4 compression offers moderate compression ratios with very little CPU utilization. ZLE offers compression of zeroes and padding only—it won't even try to compress actual data—which makes it an excellent choice for otherwise-incompressible data, like photos and movies. Finally, the new ZSTD algorithm offers better compression ratios than LZ4 at the expense of somewhat higher CPU utilization.

Next, we don't recommend ZFS deduplication. It rarely provides much in the way of actual space savings, and it can have catastrophically bad impact on write performance—especially after several years of use.

> *For a detailed look at deduplication and recent advances in ZFS deduplication, check out our article on OpenZFS Fast Dedup.*

If you're considering support vdevs—meaning **CACHE**, **LOG**, or **SPECIAL**—don't try to partition one large SSD to serve multiple roles. This almost always results in *decreased* performance, as in "worse than you'd have with no support vdevs at all," and it's one of the most common mistakes we see newbies make.

Finally, whatever your workload is, *understand* your workload. Storage performance is not "one size fits all," and a system which is blindingly fast at one workload can be *cripplingly slow* on another. Trying to "kitchen sink" a storage system—blindly applying every technique you've ever heard of someone using—is a path to unnecessary expenses, frustration, and failure.

Conclusions

In this first part of our series of OpenZFS Best Practices, we have covered universal considerations that apply to every pool. In the remaining two parts we will delve into specific considerations and tuning for fileservers and SANs, and then databases and VMs. Be sure to subscribe so you don't miss the continuation of this series.

If your head is spinning after reading this guide, don't feel bad—you aren't the first person to learn that doing storage right is difficult, and you won't be the last. Klara's ZFS support subscription provides continuous access to expert advice and assistance with your critical storage infrastructure to make sure you always have access to the support you need.

Topics / Tags

openzfs

Back to Articles