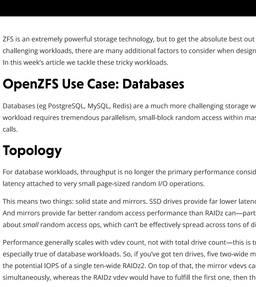
OpenZFS Storage Best Practices and Use Cases – Part 3: Databases and VMs

DECEMBER 12, 2023

In the conclusion of our ZFS Best Practices series we're covering two of the trickiest use cases, databases and virtual machine hosting.



ZFS is an extremely powerful storage technology, but to get the absolute best out of it, especially under the most challenging workloads, there are many additional factors to consider when designing and configuring your storage pool. In this week's article we tackle these tricky workloads.

OpenZFS Use Case: Databases

Databases (eg PostgreSQL, MySQL, Redis) are a much more challenging storage workload than generic file service. This workload requires tremendous parallelism, small-block random access within massive files, and constant explicit fsync calls.

Topology

For database workloads, throughput is no longer the primary performance consideration—latency is. Specifically, the latency attached to very small page-sized random I/O operations.

This means two things: solid state and mirrors. SSD drives provide far lower latency than conventional drives possibly can. And mirrors provide far better random access performance than RAIDz can—particularly when we're talking about *small random* access ops, which can't be effectively spread across tons of disks in a wide striped vdev.

Performance generally scales with vdev count, not with total drive count—this is true of almost all workloads, but it's especially true of database workloads. So, if you've got ten drives, five two-wide mirror vdevs provide roughly five times the potential IOPS of a single ten-wide RAIDZ2. On top of that, the mirror vdevs can each fulfill two read requests simultaneously, whereas the RAIDz vdev would have to fulfill the first one, then the other.

For database workloads, striped RAID—including RAIDz—generally performs *worse* than a single disk. We're really not kidding about the importance of mirrors for this workload!

Tunables

The recordsize property should be matched to the page size of your database engine. For MySQL/MariaDB defaults, that's 16KiB. For PostgreSQL defaults, it's 8KiB. It's very much worth doing your research here and finding the actual page size used by your application so you can get this setting tuned properly.

If you set up a PostgreSQL database on an untuned OpenZFS dataset, the recordsize will be 128KiB, and every read you issue will pull in a whopping sixteen times more data than it needs. Worse, every *write* you issue will actually be a read-modify-write—you'll have to read in 128KiB, modify 8KiB of it, then write the entire 128KiB out again.

Don't do that.

Most database workloads are highly compressible and will see both space *and* performance benefits from setting compress=lz4. Be careful with compress=zstd: it achieves higher compression ratios, but they can frequently come at the expense of performance, whereas lz4 is generally a massive win across the board. A Klara ZFS Performance Audit can help you find the best possible configuration for your database to ensure you get maximum performance.

xattr=sa is not worth it for datasets used to store database files, because there are only a few relatively massive files. But you may want to set atime=off, to make sure you aren't unnecessarily blipping the access time metadata on the database every time you read or write a row!

Support vdevs

Most database workloads are extraordinarily sync-heavy, which in turn means that a nice, fast **LOG** vdev can offer major performance benefits. Be careful to choose an SSD that offers high sustained write performance, high write endurance, and low and predictable latency.

The best choice for **LOG** vdevs, by far, is Intel Optane technology. Optane offers incredibly low latency, and utterly UFO-class endurance. Unfortunately, it's getting difficult to find Optane, so you may be stuck with conventional NAND-based SSDs.

If you're stuck with NAND flash, you want a fairly large size (to improve write endurance) and, ideally, hardware QoS and onboard non-volatile cache. This means enterprise-targeted drives, like Kingston's fairly inexpensive DC500M line.

CACHE vdevs are rarely useful in database environments, as it will just occupy more RAM that could be used for primary cache. The DBMS also has its own cache which will need to be tuned to not interfere with the ARC, and to ensure RAM is not wasted double caching the same data.

Every use case is different, but we do not generally recommend implementing a **SPECIAL** vdev on a system primarily intended for database hosting—there's not enough file-level metadata to read or write to make the added point of failure worthwhile.

OpenZFS Use Case: Virtual Machine Hosting

In many ways, VM hosting strongly resembles database hosting—you've got random access within much larger files, with potentially a lot of synchronous write requests, and this challenging workload benefits strongly from careful, knowledgeable tuning.

Unlike database hosting, a single VM can provide several effectively different workloads—which makes tuning for it a bit less of a walk through the docs, and a bit more of an applied art.

VMs may or may not issue a lot of fsync calls—for the most part, the outside of the VM won't fsync any more frequently than the inside of it does. So a VM hosting databases will issue endless fsyncs, while one doing general office file serving over SMB will probably issue almost none.

One enormous caveat here: if you're doing VM hosting with separate storage and compute siloes, you need to keep your network transport in mind. If you use synchronous NFS as a network transport, your VM will issue 100% synchronous writes, even if its internal workload is completely asynchronous.

Topology

We strongly recommend mirrors for VM hosting, for many of the same reasons we recommended them for databases. Mirrors provide extreme performance, they resilver very rapidly when a drive must be replaced, and in smaller systems, they provide additional flexibility.

In a ten-bay system with a single RAIDz2 vdev, filling that vdev generally means panic—there aren't enough bays to add a second matching vdev, and expanding the original vdev means laboriously replacing each disk and waiting for it to resilver, which could mean weeks of daily fiddling before you can turn a 50TB vdev into a 100TB vdev.

In the same ten-bay system, you might start with a single mirror vdev. When that vdev gets to around 70% full, you might then decide to add a second mirror vdev—which pops right into your pool with no hassle, and no downtime, instantly doubling your original storage space.

Even once your smaller system is completely full, you only need to replace *two* drives total before seeing additional free space—a system with one 20TB mirror vdev and three 10TB mirror vdevs provides 50TB, not 40TB!

With that said, larger systems or less challenging workloads may be fine with fairly narrow RAIDz vdevs, which offer higher storage efficiency and/or fault tolerance.

Three-wide RAIDz1 offers 67% efficient storage and excellent performance (due to the very narrow stripe size, as well as the implied higher total vdev count).

Four-wide RAIDZ2 offers the same 50% storage efficiency as mirrors do, and considerably lower performance—but they offer dual fault tolerance, which some admins may find worth it.

Finally, six-wide RAIDz2 offers the 67% storage efficiency that three-wide RAIDz1 did—*and* dual fault tolerance.

We do not generally recommend hosting VMs on a single RAIDz vdev—if you don't have sufficient bays for multiple vdevs at a given width, you should almost certainly choose a narrower topology.

Tunables

Record size is, as always, the single most important tunable for this workload. Unfortunately, finding the best recordsize is a bit more complex—in addition to the application workload's patterns, you now need to worry about your disk virtualization format.

For VMs running on the Linux Kernel Virtual Machine (KVM), the QCOW2 file format is a very common storage back end. By default, QCOW2 files use **cluster_size=64K**. This means that all access to the VM's virtual drives is done in 64KiB pieces, which in turn means **recordsize=64K** is the optimal setting for datasets hosting those QCOW2 files.

With that said, you may not want to leave that cluster_size set to default! If possible, you want the same operation page size all the way up the stack: from application-level requests inside the VM, to storage calls made by the host to the storage back-end, all the way up to OpenZFS itself.

As an example, if you're setting up a KVM virtual machine to serve a MySQL database, you want to create its qcow2 file using **cluster_size=16K**, and the dataset it's stored on with **recordsize=16K**.

Again, please keep in mind that databases themselves are usually configurable—the odds are very good that any random database-backed application will use that DB's default page size, but the more complex the application, the more likely it is that there's been custom hand-tuning that you'll want to match directly for optimal performance.

With VMWare, the best configuration will depend on if you are using iSCSI or NFS. When using iSCSI is it a single volume, or a large VMFS? For NFS, what is the NFS client block size? Many factors will impact what settings will provide the best performance outcome.

If you're using a ZVOL as your VM's storage back end, you should be aware that the same issues apply to its **volblocksize** tunable, which defaults to 8KiB. This is ideal or near-ideal for PostgreSQL, but a very bad choice for nearly any other workload.

As usual, **atime=off** is usually a free and easy way to conserve a few IOPS. But like databases and unlike general purpose file service, there aren't likely to be enough individual files for **xattr=sa** to make a positive difference.

Support vdevs

It's hard to give concrete advice about support vdevs for VM hosting environments because the workload inside the VMs is generally what determines the utility of support vdevs.

If you're using NFS transport, or your VMs have highly synchronous workloads of their own, a **LOG** vdev is a very good idea, and you should follow the same selection process as we covered in the databases section. In the absence of NFS transport or sync-heavy workloads *inside* the VMs, you probably won't have enough synchronous write requests to justify the **LOG** vdev, though.

As usual, we're lukewarm on **CACHE** vdevs. They're more likely to be useful on VM servers than most other workloads, but that's not the same thing as "likely to be useful." In general, truly massive amounts of storage with extremely repetitive access rates will benefit the most from a **CACHE**—but adding more system RAM is generally a much better idea than adding a **CACHE**.

a **SPECIAL** to a VM hosting server can reduce the indirection penalty in ZFS, where you must first read a metadata block to find the address of the data block that needs to be read. Only in OpenZFS 2.2 and later can ZVOLs make full use of the **SPECIAL** vdev.

Conclusions

ZFS provides an extremely reliable and performant storage backend for any workload, but with the right configuration and tuning it can perform even better.

| *For more information on how to avoid common mistakes when benchmarking ZFS performance, refer to our article on* ***common mistakes in ZFS storage benchmarks.***

Consider a ZFS Subscription to access the Klara team's expertise and advice to get the most out of OpenZFS for your workload.

In this series, we have examined universal best practices, recommendations about backups and data safety, and optimizing ZFS for all kinds of workloads from file serving to databases. If you have any questions, please reach out to us, we'd be glad to help.

Topics / Tags

openzfs

Back to Articles