Skip to main content.

BSD Now

A Weekly BSD Podcast - News, Interviews and Tutorials

Subscribe on Youtube « | Subscribe with iTunes « | RSS: MP3 | Video | HD Video | HD Torrent Feed
Navigation: Home | Episodes | About | Live | Shirt | Contact Us | Tutorials

A crash course on ZFS

2013-12-04

Live demo in BSD Now Episode 014 | Originally written by TJ and Allan for bsdnow.tv | Last updated: 2014/03/08

NOTE: the author/maintainer of the tutorial(s) is no longer with the show, so the information below may be outdated or incorrect.

Originally a proprietary feature of Sun's operating system, ZFS has become the most powerful and flexible filesystem to date - and completely open source. Open source development has since moved to the OpenZFS project after Oracle closed the source of versions past v28. ZFS is actually not just a filesystem; it's a suite of tools and a volume manager with RAID capabilities. ZFS boasts data integrity features that other filesystems lack entirely. Unless specifically disabled by the administrator, all data written to disk has a checksum associated with it. By doing this, ZFS is able to detect silent data corruption and provide a level of reliability that is unmatched. This reliability is increased when you couple it with a solid, mature implementation such as

FreeBSD's or illumos'. Work is going on to improve Mac OS X's version, but it's not production-ready quite yet. ZFS on Linux is still in its infancy and should be avoided in production environments too.

Requirements

ZFS likes plenty of RAM. It *loves* RAM. At the bare minimum, your system should have 1GB. For each 1TB of total storage, it's recommended that you add another 1GB. It never hurts to add more though, and ZFS is much faster with a lot of memory. This is because it does adaptive caching of the data, keeping the most often-accessed files in RAM. If you're using deduplication, the memory requirements skyrocket even higher - 5GB of RAM for every 1TB of storage. It's possible to run ZFS with a lower amount of memory through specific tuning, but you're walking a thin line. You should always run ZFS on a 64bit platform for best performance and stability. ZFS can work with a hardware RAID controller, but it's best to just give it the raw disks.

Installation

You can have your whole OS be installed to a zpool, or you can just use ZFS for the data storage. Prior to FreeBSD 10.0, you had to do some manual work to install the OS to a zpool. There were third party scripts and installers that made the process easier, but as of 10.0 they're no longer required. We have a ZFS-on-root option in bsdinstall now. If you're running a version before that, see these instructions. Having your whole OS be protected by ZFS is a really great thing. You can sleep easy knowing your data is safe.

Creation and Initial Setup

To enable ZFS on FreeBSD, add a line to your /etc/rc.conf file like so:

```
# echo 'zfs_enable="YES"' >> /etc/rc.conf
```

Now all zpools you create will be automatically mounted on startup. The next step is to create the storage pool. In ZFS, all storage is combined into a common pool that is then used to create one or more datasets. A ZFS pool can be backed by whole disks, partitions or regular files. In this example we'll use regular files. If you are testing in a virtual machine, you can create multiple virtual disks instead. **It's highly recommended to use full disks (not files) on real hardware.**

```
# truncate -s 2G file1
# truncate -s 2G file2
# truncate -s 2G file3
# truncate -s 2G file4
```

We'll create the ZFS pool now. Valid types include stripe, mirror, raidz1, raidz2 and raidz3. Each of them have their own advantages and disadvantages. Based on the number of disks you have and your intended usage, you'll want to read about each of them. For this example, we will do a RAID-Z.

```
# zpool create mypool raidz1 /tutorial/file1 /tutorial/file2 /tutorial/file3 /tutorial/file4
```

This will create /mypool and will be 6GB in size. We have 4 pseudo-drives of 2GB each, minus 1 for data redundancy. You can check the list of pools and get a brief overview of them by issuing:

```
# zfs list

NAME     USED  AVAIL  REFER  MOUNTPOINT
mypool   144K  5.84G  43.4K  /mypool
```

You can change the mount point of your zpool by doing something like this during creation:

```
# zpool create -m /mnt mypool /tutorial/file1 /tutorial/file2 /tutorial/file3 /tutorial/file4
```

One of the most powerful features of ZFS is the ability to create multiple separate file systems with different settings from a common storage pool. Here we'll create a "subdirectory" dataset in the root pool and enable LZ4 compression on it. We can then make another subdirectory and disable compression on it. We'll be using the ports tree here, since it's just a bunch of text files. Those compress well, and this will speed up portsnap and things like that by quite a bit. On the other hand, distfiles (source code tarballs) are already compressed, so we don't want to waste time trying to recompress them. Specific information about compression is mentioned later in this tutorial.

```
# zfs create mypool/usr
# zfs create -o compression=lz4 mypool/usr/ports
# zfs create -o compression=off mypool/usr/ports/distfiles
# zfs list

NAME                              USED  AVAIL  REFER  MOUNTPOINT
mypool                            315K  5.84G  44.9K  /mypool
mypool/usr                        133K  5.84G  44.9K  /mypool/usr
mypool/usr/ports                 88.3K  5.84G  44.9K  /mypool/usr/ports
mypool/usr/ports/distfiles       43.4K  5.84G  43.4K  /mypool/usr/ports/distfiles
```

If we look at the compression property, we can see the default applied to /usr, then our specific settings applied to /usr/ports and /usr/ports/distfiles

```
# zfs get -r compression mypool

NAME                             PROPERTY     VALUE     SOURCE
mypool                           compression  off       default
mypool/usr                       compression  off       default
mypool/usr/ports                 compression  lz4       local
mypool/usr/ports/distfiles       compression  off       local
```

Note that changing the compression property, and most other properties, only affects data written after the setting is applied. Enabling or disabling compression does not change data that was written previously.

To delete a zpool:

```
# zpool destroy mypool
```

Or just a specific dataset:

```
# zfs destroy mypool/dumbstuff
```

And it's gone.

Snapshots and Clones

One of the other powerful features of ZFS is the ability to take snapshots, which allow you to preserve "point in time" versions of the filesystem. Let's create a 10MB file in the ports dataset:

```
# dd if=/dev/random of=/mypool/usr/ports/somefile bs=1m count=10
# cd /mypool/usr/ports
# ls -lh

total 10245
-rw-r--r--  1 root  wheel    10M Nov 29 14:55 somefile
```

Now we create a snapshot to preserve this version of the filesystem. If you specify the -r (recursive) flag, it will also create a snapshot of each sub-dataset using the same snapshot name.

```
# zfs snapshot -r mypool/usr/ports@firstsnapshot
# zfs list -t all -o name,used,refer,written

NAME                                                USED  REFER  WRITTEN
mypool                                             10.5M  44.9K    44.9K
```

```
mypool/usr                                     10.1M  44.9K    44.9K
mypool/usr/ports                               10.1M  10.0M        0
mypool/usr/ports@firstsnapshot                     0  10.0M    10.0M
mypool/usr/ports/distfiles                     43.4K  43.4K        0
mypool/usr/ports/distfiles@firstsnapshot           0  43.4K    43.4K
```

As you can see, the snapshots initially take no additional space, as they only contain the data that already exists in the dataset the snapshot belongs to. If some or all of this data is overwritten, this changes. In this case we'll overwrite the last 5 megabytes of data in the file and add an additional 5MB of data to the file:

```
# dd if=/dev/random of=/mypool/usr/ports/somefile bs=1m oseek=5 count=10
# zfs list -t all -o name,used,refer,written

NAME                                            USED  REFER  WRITTEN
mypool                                         20.4M  44.9K    44.9K
mypool/usr                                     20.2M  44.9K    44.9K
mypool/usr/ports                               20.1M  15.0M    10.0M
mypool/usr/ports@firstsnapshot                 5.03M  10.0M    10.0M
mypool/usr/ports/distfiles                     43.4K  43.4K        0
mypool/usr/ports/distfiles@firstsnapshot           0  43.4K    43.4K
```

The file is now 15MB and the snapshot has grown to use 5MB of space. The 5MB of data that was overwritten has actually be preserved by the snapshot. In total, 20MB of space has been consumed - the 15MB for the current file and the 5MB of preserved overwritten data. 5MB of storage has been saved by not having to store the unmodified part of the original file twice.

ZFS snapshots are read-only, but they can be accessed via a hidden ".zfs" directory. This allows you to easily restore a single file that was accidentally modified or deleted:

```
# ls -lh /mypool/usr/ports/.zfs/snapshot/firstsnapshot/
```

```
total 10247
-rw-r--r--  1 root  wheel    10M Nov 29 15:01 somefile
```

If you wanted to reverse all of the files in a dataset back to how they were in a snapshot, rather than copying all of the files from the snapshot back to the dataset (which would consume double the space), ZFS has the "rollback" operation, which reverts all changes written since the snapshot:

```
# zfs rollback -r mypool/usr/ports@firstsnapshot
# zfs list -t all -o name,used,refer,written -r mypool

NAME                                              USED  REFER  WRITTEN
mypool                                           10.3M  44.9K    44.9K
mypool/usr                                       10.1M  44.9K    44.9K
mypool/usr/ports                                 10.1M  10.0M    1.50K
mypool/usr/ports@firstsnapshot                   1.50K  10.0M    10.0M
mypool/usr/ports/distfiles                       43.4K  43.4K        0
mypool/usr/ports/distfiles@firstsnapshot             0  43.4K    43.4K
```

Now it's back to the original 10MB version, and the freed space has been reclaimed.

Sending and Receiving Snapshots

ZFS lets you send snapshots of your pool or dataset and output it to a file. You can also pipe it to other commands. This can be used to send datasets over the internet, using SSH, and receive them on a remote host. Offsite backups are a great use case for this. First, let's take a snapshot of a dataset and redirect it to a regular file. This is a local backup.

```
# zfs snapshot mypool/myfiles@backup
# zfs send mypool/myfiles@backup > /mnt/filesystem-backup
```

To restore the backup from the file, we would do something like:

```
# zfs receive -v mypool/myfiles < /mnt/filesystem-backup
```

Now let's also copy that snapshot to a remote server. This is an offsite backup.

```
# zfs send mypool/myfiles@backup | ssh you@remoteserver zfs receive -v otherpool/myfiles
```

It's also possible to do incremental data (only what's changed since the last time) when you have multiple snapshots.

```
# zfs send -i mypool/myfiles@backup mypool/myfiles@laterbackup | \
  ssh you@remoteserver zfs receive -v otherpool/myfiles
```

You can get quite creative with these tools, piping them to other utilities (xz, for example) and using standard redirection techniques.

Data Integrity

One of the major selling features of ZFS is the safety it provides. All data and metadata written are checksummed to ensure that the data has not become corrupted over time. Every time data is read from a ZFS pool, the checksum is calculated and compared to the checksum that was calculated when the data was originally written. If they do not match, it means the data has become corrupted on the storage. If you have a redundant ZFS pool, the corruption will be automatically repaired and noted in the status screen. You can also initiate a manual scan of all data on the drive to check for corruption:

```
# zpool scrub mypool
# zpool status mypool
```

```
pool: mypool
state: ONLINE
scan: scrub in progress since Fri Nov 29 15:30:59 2013
      13.8M scanned out of 13.9M at 4.60M/s, 0h0m to go
0 repaired, 99.45% done
config:

NAME                            STATE     READ WRITE CKSUM
mypool                          ONLINE       0     0     0
 raidz1-0                       ONLINE       0     0     0
    /tutorial/file1             ONLINE       0     0     0
    /tutorial/file2             ONLINE       0     0     0
    /tutorial/file3             ONLINE       0     0     0
    /tutorial/file4             ONLINE       0     0     0

errors: No known data errors
```

We can also simulate the failure of a disk:

```
# rm /tutorial/file3
# zpool scrub mypool
# zpool status mypool

pool: mypool
state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
see: http://illumos.org/msg/ZFS-8000-2Q
scan: scrub repaired 0 in 0h0m with 0 errors on Fri Nov 29 15:36:45 2013
config:

NAME                            STATE     READ WRITE CKSUM
```

```
mypool                    DEGRADED     0     0     0
  raidz1-0                DEGRADED     0     0     0
    /tutorial/file1       ONLINE       0     0     0
    /tutorial/file2       ONLINE       0     0     0
    474744448393399570    UNAVAIL      0     0     0  was /tutorial/file3
    /tutorial/file4       ONLINE       0     0     0

errors: No known data errors
```

Simulate replacing the failed disk with a new one:

```
# truncate -s 2G /tutorial/file3
# zpool replace mypool 474744448393399570 /tutorial/file3
# zpool status mypool

pool: mypool
state: ONLINE
status: One or more devices is currently being resilvered.  The pool will
        continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
scan: resilver in progress since Fri Nov 29 15:38:42 2013
      13.8M scanned out of 14.1M at 3.46M/s, 0h0m to go
      3.45M resilvered, 98.17% done
config:

        NAME                          STATE     READ WRITE CKSUM
        mypool                        ONLINE       0     0     0
          raidz1-0                    ONLINE       0     0     0
            /tutorial/file1           ONLINE       0     0     0
            /tutorial/file2           ONLINE       0     0     0
            replacing-2               UNAVAIL      0     0     0
              474744448393399570      UNAVAIL      0     0     0  was /tutorial/file3/old
            /tutorial/file3           ONLINE       0     0     0  (resilvering)
            /tutorial/file4           ONLINE       0     0     0

errors: No known data errors
```

Once the resilver is finished the pool is back to normal:

```
# zpool status mypool

 pool: mypool
 state: ONLINE
 scan: resilvered 3.45M in 0h0m with 0 errors on Fri Nov 29 15:38:46 2013
 config:

        NAME                        STATE     READ WRITE CKSUM
        mypool                      ONLINE       0     0     0
          raidz1-0                  ONLINE       0     0     0
            /tutorial/file1         ONLINE       0     0     0
            /tutorial/file2         ONLINE       0     0     0
            /tutorial/file3         ONLINE       0     0     0
            /tutorial/file4         ONLINE       0     0     0

errors: No known data errors
```

To grow a mirror, expansion will need to be set on the pool.

```
# zpool set autoexpand=on mypool
```

For a two drive disk pool (mirrored) to increase in size these two drives will need to be replaced, one at a time. The new drives will need to be the same size, since they will be mirrored. Let's assume that the pool is on ad1 and ad2. Before we start, we can run a scrub on mypool to ensure the data integrity. Replace the first drive (ad1) with one of the larger new drives (ad3). Offline (remove) ad1 from mypool:

```
# zpool offline mypool ad1
```

It will now be in degraded mode. The pools state will change to **DEGRADED**. Next we replace ad1 with ad3:

```
# zpool replace mypool ad1 ad3
```

After replacing the drive, the pool will resilver itself. We need to wait until this operation completes. The size of mypool will still read the same as it did before, no increase in size will be visible. Replace the second drive (ad2) with the other larger new drives (ad4). Here we will follow the same steps that we did for the first drive. Offline (remove) ad2 from mypool:

```
# zpool offline mypool ad2
```

It will now be in **DEGRADED** mode. Finally we replace ad2 with ad4:

```
# zpool replace mypool ad2 ad4
```

After the drive has been replaced, mypool will resilver. After the resilver process completes, we can check the size of mypool with:

```
# zfs list
```

The size of mypool will now register the increased size of the drives. You can also choose what checksum algorithm is used with the "set" subcommand, like so:

```
# zfs set checksum=sha256 mypool
```

Other choices include "fletcher2," "fletcher4" and "none."

Boot Environments

Boot environments are a feature that lets you boot from multiple, different zpools. This lets you rollback from a failed update, among other situations. They're great for testing and experimentation. For FreeBSD 9.2, you need to install the sysutils/beadm port to be able to use them to their full potential. For this example, we'll take a snapshot of the main pool, break something, then reboot back into the snapshot where everything worked.

```
# beadm create sketchyupdate
```

So now I'm just casually running some commands...

```
# rm -rf /boot/*
```

Oh, that was a bad idea. At this point, your system will not come up anymore if it's rebooted. To fix this, let's tell the system to reboot from the "sketchyupdate" snapshot we took before making this foolish decision.

```
# beadm activate sketchyupdate
# reboot
```

Now it's like we never did anything! If you want to see your boot environments, run:

```
# beadm list
```

And that will show them all.

Compression

The ability to transparently compress data is more useful than one might initially realize. Not only does it save space, but in some cases it drastically improves performance. This is because the time it takes to compress or decompress the data is quicker than the time it takes to read or write the uncompressed data to disk. Currently, the following compression options are available:

- LZ4 (the latest and greatest - recommended)
- gzip (configurable between levels 0-9, uses 6 by default - not recommended)
- LZJB (still pretty fast and provides a good trade-off between speed and space)

You can enable or disable compression on each dataset, and check the ratio of space saved with:

```
# zfs get compressratio mypool

NAME                   PROPERTY       VALUE                  SOURCE
mypool                 compressratio  1.00x
```

Think about what types of files you'll be storing in which directories. Compression is a very powerful tool.

Deduplication

ZFS can potentially save a lot of disk space by using deduplication. Basically, deduplication allows you to store the same data multiple times, but only take up the space of a single copy. Depending on your environment and what kind of data you'll be storing, this can make a massive difference. An obvious example would be storing virtual machines, where not a lot is changing between them. It comes at a cost though: you need **a lot** of RAM. If you're planning to store multiple copies of the same file(s), consider the space-saving benefits of enabling deduplication on your pool or dataset. Data can be deduplicated on the file, block, or byte level. Let's try it out:

```
# zfs create mypool/vms
# zfs set dedup=on mypool/vms
```

If you're paranoid about hash collisions, you might want to use extra verification. This adds additional overhead and probably isn't needed at all with SHA256, but it's up to you and what you're comfortable with.

```
# zfs set dedup=verify mypool/vms
```

If you are using the collision verification, it might make sense to use a faster but weaker checksum instead of SHA256 to compensate. You can select which mix to use like so:

```
# zfs set checksum=sha256,verify mypool/vms
```

You'll have to find the balance that best suits your specific situation.

ARC, L2ARC and ZIL

Data stored on ZFS comes from the disks initially, but disks are slow. There needed to be a way to cache often-accessed data on a faster storage medium, and that's where the ARC and L2ARC come in. As previously mentioned, ZFS will store a copy of the most often-accessed files in RAM (in addition to having a copy in the pool). This is the ARC, or Adaptive Replacement Cache. When your data becomes larger than the amount of memory you have, however, it spills over into the disk and operations become much slower. You can use a fast storage device like an SSD for a "level 2" ARC, or L2ARC. The L2ARC is a caching "layer" between the RAM (very fast) and the disks (not so fast). To add an L2ARC to your existing zpool, we might do:

```
# zpool add mypool cache /tutorial/ssd
```

While it won't show an immediate speed benefit, over time it will start learning what to cache and you'll see improvement.

The ZIL, ZFS Intent Log, is the write cache. This is in contrast to the L2ARC, which is the read cache. It writes the file metadata to a faster device to increase the write throughput. A ZIL basically turns synchronous writes into asynchronous writes, improving overall performance. Let's add one to our pool.

```
# zpool add mypool log /tutorial/file7
```

It's also possible to add mirrored ZILs for even more protection.

```
# zpool add mypool log mirror /tutorial/file7 /tutorial/file8
```

SLC SSDs make a great choice for the ZIL as well.

Conclusion

ZFS is really the final word in filesystems. With a feature set longer than this tutorial, it can take a while to master. You can set many more options per dataset, enable disk usage quotes and much more. Once you've used it and seen the benefits, you'll probably never want to use anything else. Hopefully this has been helpful to get you on your way to becoming a FreeBSD ZFS master.

Some links for further reading:

- man zpool
- man zfs
- man zpool-features
- man zdb
- http://wiki.illumos.org/display/illumos/ZFS
- https://en.wikipedia.org/wiki/ZFS
- https://www.freebsd.org/doc/handbook/filesystems-zfs.html
- http://open-zfs.org/wiki/Main_Page
- https://wiki.freebsd.org/ZFS
- https://wiki.freebsd.org/RootOnZFS
- http://constantin.glez.de/blog/2011/07/zfs-dedupe-or-not-dedupe

---

Latest News

New announcement

2017-05-25

Hi, Mr. Dexter. Also, we understand that Brad Davis thinks there should be more real news....

Two Year Anniversary

2015-08-08

We're quickly approaching our two-year anniversary, which will be on episode 105. To celebrate, we've created a unique t-shirt design, available for purchase until the end of August. Shirts will be shipped out around September 1st. Most of the proceeds will support the show, and specifically allow us to buy...

New discussion segment

2015-01-17

We're thinking about adding a new segment to the show where we discuss a topic that the listeners suggest. It's meant to be informative like a tutorial, but more of a "free discussion" format. If you have any subjects you want us to explore, or even just a good name...

How did you get into BSD?

2014-11-26

We've got a fun idea for the holidays this year: just like we ask during the interviews, we want to hear how all the viewers and listeners first got into BSD. Email us your story, either written or a video version, and we'll read and play some of them for...

1 | 2 | 3 | 4 | 5 | Next >

Episode 228: The Spectre of Meltdown

2018-01-10

Direct Download:HD VideoMP3 AudioTorrent This episode was brought to you by Headlines Meltdown Spectre Official Site Kernel-memory-leaking Intel processor design flaw forces Linux, Windows redesign Intel’s official response The Register mocks intels response with pithy annotations Intel’s Analysis PDF XKCD Response from FreeBSD FreeBSD's patch WIP Why Raspberry Pi isn’t vulnerable to Spectre or Meltdown Xen mitigation patches Overview of affected FreeBSD Platforms/Architectures Groff's response We’ll...

Episode 227: The long core dump

2018-01-03

Direct Download:HD VideoMP3 AudioTorrent This episode was brought to you by Headlines NetBSD 7.1.1 released The NetBSD Project is pleased to announce NetBSD 7.1.1, the first security/critical update of the NetBSD 7.1 release branch. It represents a selected subset of fixes deemed important for security or stability reasons. Complete source and binaries for NetBSD 7.1.1...

Episode 226: SSL: Santa’s Syscall List

2017-12-27

Direct Download:HD VideoMP3 AudioTorrent This episode was brought to you by Headlines FreeBSD Q3 Status Report 2017 FreeBSD Team Reports FreeBSD Release Engineering Team Ports Collection The FreeBSD Core Team The FreeBSD Foundation Projects FreeBSD CI Kernel Intel 10G iflib Driver Update Intel iWARP Support pNFS Server Plan B Architectures AMD Zen (family 17h) support Userland Programs Updates to GDB Ports FreeBSDDesktop OpenJFX 8 Puppet Documentation Absolute FreeBSD, 3rd Edition Manual Pages Third-Party Projects The nosh Project FreeBSD...

Episode 225: The one true OS

2017-12-20

Direct Download:HD VideoMP3 AudioTorrent This episode was brought to you by Headlines TrueOS stable release 17.12 We are pleased to announce a new release of the 6-month STABLE version of TrueOS! This release cycle focused on lots of cleanup and stabilization of the distinguishing features of TrueOS: OpenRC, boot speed, removable-device...