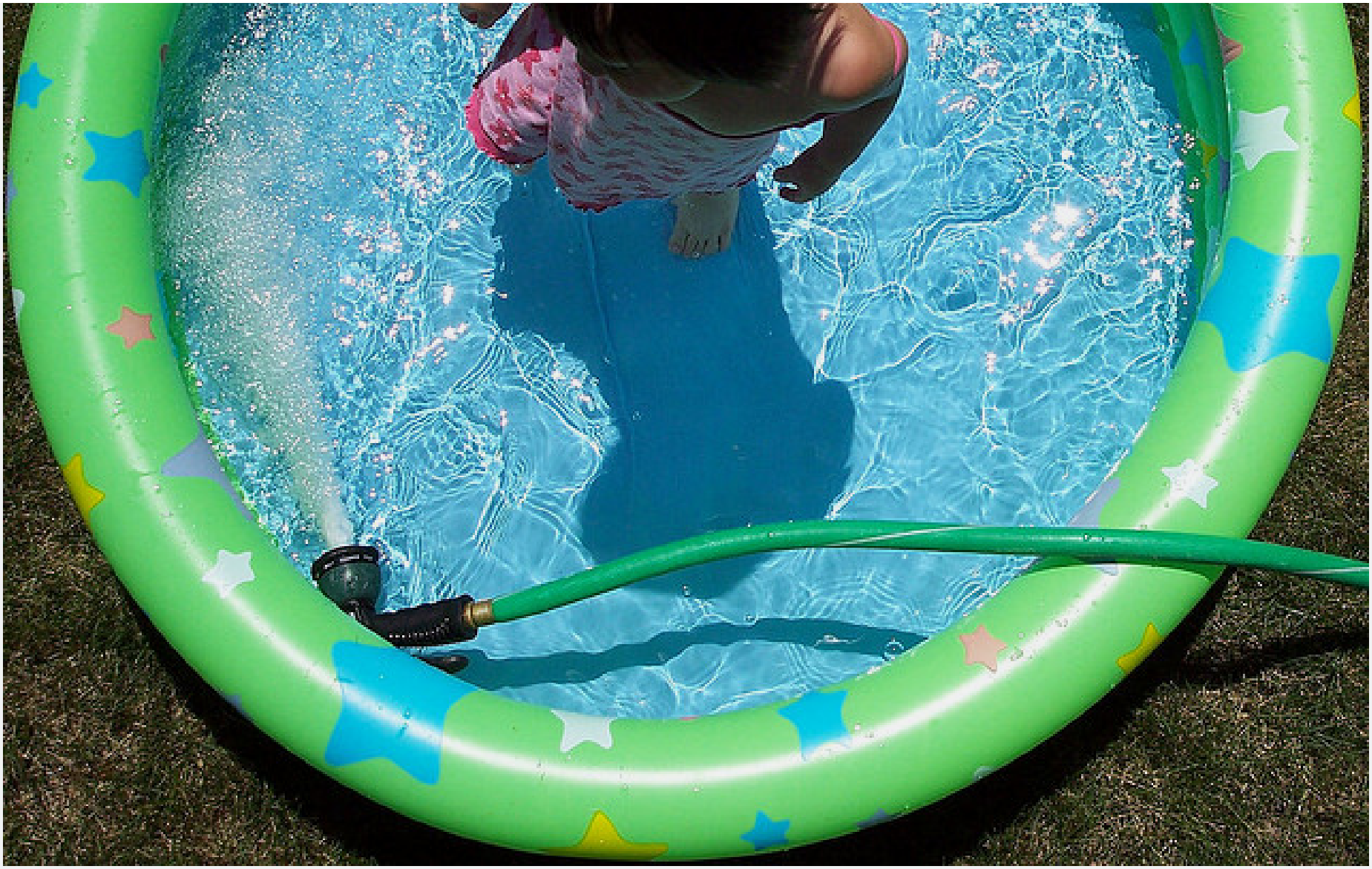
*BIZ & IT —*

# Ars walkthrough: Using the ZFS next-gen filesystem on Linux

If btrfs interested you, start your next-gen trip with a step-by-step guide to ZFS.

JIM SALTER - 2/23/2014, 12:00 PM



zpool is equally exciting.

## Baby's first zpool



[redacted] or not, it's p[redacted]asy. We're going to create a relatively simple zpool here, consisting of a single raidz1 vdev with three drives in it. We're

going to devote the entire drives to ZFS, so we don't need to partition them first. First step is positively identifying the drives, using /dev/disk/by-id:

```
me@box:~$ ls -l /dev/disk/by-id
total 0
lrwxrwxrwx 1 root root  9 Aug  1 22:09 ata-WDC_WD2002FAEX-007BA0_WD-WCAY01065572 -> ../../sda
lrwxrwxrwx 1 root root 10 Aug  1 22:09 ata-WDC_WD2002FAEX-007BA0_WD-WCAY01065572-part1 -> ../../sda1
lrwxrwxrwx 1 root root  9 Aug  1 22:09 ata-WDC_WD2002FAEX-007BA0_WD-WCAY01065731 -> ../../sdb
lrwxrwxrwx 1 root root  9 Aug  1 22:09 ata-WDC_WD2002FAEX-007BA0_WD-WCAY01127972 -> ../../sdc
lrwxrwxrwx 1 root root  9 Aug  1 22:09 ata-WDC_WD2002FAEX-007BA0_WD-WCAY01133538 -> ../../sdd
lrwxrwxrwx 1 root root  9 Aug  1 22:09 scsi-SATA_WDC_WD2002FAEX-_WD-WCAY01065572 -> ../../sda
lrwxrwxrwx 1 root root 10 Aug  1 22:09 scsi-SATA_WDC_WD2002FAEX-_WD-WCAY01065572-part1 -> ../../sda1
lrwxrwxrwx 1 root root  9 Aug  1 22:09 scsi-SATA_WDC_WD2002FAEX-_WD-WCAY01065731 -> ../../sdb
lrwxrwxrwx 1 root root  9 Aug  1 22:09 scsi-SATA_WDC_WD2002FAEX-_WD-WCAY01127972 -> ../../sdc
lrwxrwxrwx 1 root root  9 Aug  1 22:09 scsi-SATA_WDC_WD2002FAEX-_WD-WCAY01133538 -> ../../sdd
lrwxrwxrwx 1 root root  9 Aug  1 22:09 wwn-0x50014ee2080259c8 -> ../../sdd
lrwxrwxrwx 1 root root  9 Aug  1 22:09 wwn-0x50014ee2080268b2 -> ../../sdc
lrwxrwxrwx 1 root root  9 Aug  1 22:09 wwn-0x50014ee25d4cdecd -> ../../sda
lrwxrwxrwx 1 root root 10 Aug  1 22:09 wwn-0x50014ee25d4cdecd-part1 -> ../../sda1
lrwxrwxrwx 1 root root  9 Aug  1 22:09 wwn-0x50014ee25d4ce711 -> ../../sdb
```

Wheee! How do we interpret this? Well, we have four physical drives, beginning with /dev/sda and ending with /dev/sdd. We can see that /dev/sda has a partition table on it, which confirms what we'd already assume—this is our existing system drive. The other three drives are bare, and we'll use them for our raidz1 vdev.

We can see each drive listed multiple times because they can be referred to multiple ways: by their wwn ID, by their model and serial number as connected to the ATA bus, or by their model and serial number as connected to the (virtual, in this case) SCSI bus. Which one should you pick? Well, any of them will work, including the super simple devicename (like /dev/sdb) itself, but you want to pick one that you can also see on the label on the physical drive. This is so you can be **absolutely** certain that you pull and replace the correct drive later, if one fails.


D printed
ng to use

that.

```
me@box:~$ sudo zpool create -o ashift=12 ars raidz1 /dev/disk/by-id/wwn-0x50014ee25d4ce711 /dev/disk/by-id/w
```

Whew. Dissecting the pieces: **-o ashift=12** means "use 4K blocksizes instead of the default 512 byte blocksizes," which is appropriate on almost all modern drives. Ars is the name of our new zpool. Raidz1 means we want a striped array with a single block of parity per stripe. And, finally, we have the device identifiers themselves. Voila:

```
me@box:~$ sudo zpool status
  pool: ars
 state: ONLINE
  scan: none requested
config:

        NAME                        STATE     READ WRITE CKSUM
        ars                         ONLINE       0     0     0
          raidz1-0                  ONLINE       0     0     0
            wwn-0x50014ee25d4ce711  ONLINE       0     0     0
            wwn-0x50014ee2080268b2  ONLINE       0     0     0
            wwn-0x50014ee2080259c8  ONLINE       0     0     0

errors: No known data errors

me@box:~$ sudo zpool list
NAME  SIZE   ALLOC  FREE     CAP  DEDUP  HEALTH  ALTROOT
ars   2.98T  1008K  2.98T     0%  1.00x  ONLINE  -
```

Yay! Our first zpool! First thing we should notice here is that we're seeing the full 3T capacity even though we said we wanted raidz1, which means that we're using 1TB of those 3TB for parity. That's because the **zpool** command shows us *raw* capacity, not *usable* capacity. We can see the *usable* capacity by using the **zfs** command, which queries filesystems rather than zpools (or by

```
me@box:~$ sudo zfs list
NAME   USED  AVAIL  REFER  MOUNTPOINT
ars    709K  1.96T   181K  /ars

me@box:~$ df -h /ars
Filesystem      Size  Used Avail Use% Mounted on
ars             2.0T  128K  2.0T   1% /ars
```

There we go. Now we are seeing the 2TB of *usable* capacity we'd expect out of a single-parity array with three 1TB drives.

# More ZFS lingo

Now that we've created our first zpool, let's look at the purely *logical* constructs we can (and should) create underneath them: filesystems, zvols, snapshots, and clones.

## Filesystems

This doesn't mean what you probably think it means. Under ZFS, a filesystem is sort of like a partition that comes already formatted for you, only it's really easy to create them, modify them, resize them, and otherwise play around with them instantly and whenever you'd like. Let's create a couple now.

```
me@box:~$ sudo zfs create ars/textfiles
me@box:~$ sudo zfs create ars/jpegs

me@box:~$ sudo zfs list
NAME            USED  AVAIL  REFER  MOUNTPOINT
ars            1.18M  1.96T   192K  /ars
ars/jpegs       181K  1.96T   181K  /ars/jpegs
ars/textfiles   181K  1.96T   181K  /ars/textfiles

me@box:~$ ls -l /ars
```

```
drwxr-xr-x 2 me me 2 Jan 23 15:10 jpegs
drwxr-xr-x 2 me me 2 Jan 23 15:10 textfiles
```

Snazzy! But why would we want to create filesystems instead of just making folders, since they look just like folders? Lots of reasons. You can take snapshots of a filesystem, not a folder, but you can also set *properties* on a filesystem. Since one of these folders is for Lee's awesome ANSI art, we know it'll be highly compressible. Let's go ahead and set compression on it. And just to make sure our jpeg hoarding problem won't consume our entire storage pool, let's go ahead and set a quota of 200G on it:

```
me@box:~$ sudo zfs set compression=on ars/textfiles
me@box:~$ sudo zfs set quota=200G ars/jpegs
me@box:~$ sudo zfs list
NAME            USED  AVAIL  REFER  MOUNTPOINT
ars            1.19M  1.96T   202K  /ars
ars/jpegs       181K   200G   181K  /ars/jpegs
ars/textfiles   181K  1.96T   181K  /ars/textfiles
```

Nice! We can see now that /ars/jpegs only shows 200G of space available. We can just as easily set the quota from 200G to 1T now or shrink it from 200G down to, say, 50M. It's all instantaneous and easy—no scary partition sizing, no weird "filesystem grow/shrink operations" afterward, just (re)set the quota and you're done. But what about the compression we set on /ars/textfiles? I don't actually have a significant amount of Lee's ANSI art on hand, but we can check it out by writing a whole bunch of 00s into a big file:

```
me@box:~$ dd if=/dev/zero bs=128M count=128 of=/ars/textfiles/zeroes.bin
128+0 records in
128+0 records out
17179869184 bytes (17 GB) copied, 12.1335 s, 1.4 GB/s
```

OK, that's 16GB worth of 00s written to disk. Wait a minute, did that say that it wrote at *1.4 GBps*? Sure did—highly compressible data can be compressed in memory faster than it can be written to disk, so in some cases (like textfiles, or even more so, ridiculously large numbers of zeroes) having compression on can be a *huge* performance win. (Compression will slow performance down on already-compressed or otherwise incompressible data, like most images, movies, executables, etc.)

Now let's look at our textfiles filesystem. We saw a giant performance win, will we see a corresponding storage win?

```
me@box:~$ sudo zfs list ars/textfiles
NAME            USED  AVAIL  REFER  MOUNTPOINT
ars/textfiles   181K  1.96T   181K  /ars/textfiles

me@box:~$ ls -lh /ars/textfiles
total 512
-rw-r--r-- 1 me me 16G Jan 23 15:24 zeroes.bin
```

Looks *too* good to be true; there isn't *any* more storage space taken up by our 16GB of zeroes! In reality, this is just a very extreme case. Infinite zeroes are nearly infinitely compressible; normal text (or Lee's ANSI art) would still be very compressible—frequently up to 90 percent—but not near infinite. And as we can see, a simple **ls** shows that yes, all 16GB of our zeroes are safely stored.

There are *lots* more properties that can be played with on ZFS filesystems, but we can't possibly cover them all today.

## zvols

A zvol is basically a ZFS filesystem "without the filesystem." Logically, it's presented to the system as a raw block device, directly accessible through an entry in /dev. Why would you want a zvol? Well, honestly, you probably don't. If you *did* want one, you'd want it to format with another filesystem entirely, and therefore be able to use ZFS features like snapshots, compression, dynamic resizing, and replication on it. You might not want one even then, since zvols can be a little quirky with how they handle snapshots, but that's beyond the scope of what we're trying to do today, which is just get a good beginner's handle on the basic care and feeding of ZFS.

## Snapshots

A snapshot is an instantaneously created copy of *every single block of data in a filesystem* at the exact point in time the snapshot was created. Once you have a snapshot, you can mount it, you can look through its folders and files and what have you just like you could in the original filesystem, you can copy bits and pieces out of the snapshot and into the "real world," and you can even roll the entire filesystem itself *back* to the snapshot. Let's play:

```
me@box:~$ echo lolz > /ars/textfiles/lolz.txt
```

```
NAME                      USED  AVAIL  REFER  MOUNTPOINT
ars/textfiles@snapshot1   133K      -   186K  -
```

OK, we've added a new file to /ars/textfiles. I felt a terminal case of the stupids coming on, so I took a snapshot of the filesystem and there it is, ars/textfiles@snapshot1. Notice how sometimes I use a leading slash and sometimes I don't? To the filesystem, everything is relative to root, so everything has a leading slash. To ZFS, though, "ars" is the actual pool. When we use the **zfs** command, we don't put a leading slash behind "ars." (It's a little confusing at first, but you get used to it.)

```
me@box:~$ rm /ars/textfiles/lolz.txt
me@box:~$ ls /ars/textfiles
zeroes.bin
```

Oh no! I *knew* I felt a case of the dumb coming on. My incredibly valuable lolz.txt file is gone! No worries, though, I took a snapshot... let me go ahead and mount ars/textfiles@snapshot1 and see if my missing file is there:

```
me@box:~$ mkdir /tmp/textfiles@snapshot1
me@box:~$ sudo mount -t zfs ars/textfiles@snapshot1 /tmp/textfiles@snapshot1
me@box:~$ ls /tmp/textfiles@snapshot1
lolz.txt  zeroes.bin
```

Whew! lolz.txt is safe and sound in my snapshot, which I've mounted under /tmp. I *could* just copy the file out of the mounted snapshot and put it back where I want it. But what if I'd made lots and *lots* of changes, and I wasn't sure what had or hadn't been changed? I could still put *everything* back the way it was by *rolling back* to my former snapshot.

```
me@box:~$ sudo umount /tmp/textfiles@snapshot1
me@box:~$ sudo zfs rollback ars/textfiles@snapshot1
me@box:~$ ls /ars/textfiles/
lolz.txt  zeroes.bin
```

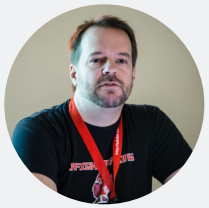
Super easy. Everything's just like it was. No fuss, no muss.

Clo

A clone is a copy of a filesystem (actually, a copy of a *snapshot* of a filesystem) that initially doesn't take up any more space on disk. As the clone diverges from its parent, it uses actual space to store the blocks that differ. There are a few interesting use cases for clones. For example, if you want to do something experimental but *really* don't want to commit to it happening in your "real" filesystem, you can instead create a clone, perform your experiments there, and then destroy the clone when you're done.

I personally find clones most valuable when using virtual machines. You can clone an older snapshot of a VM, boot it up, and then look for files, data, or programs in it without disturbing the "real" VM. Or, you can clone a *fresh* snapshot and try something risky on it. Want to see what happens when you do an in-place upgrade of that old, creaky Windows Small Business Server? Clone it and test-upgrade away.

Page: 1 2 3 Next →



JIM SALTER

Jim Salter (@jrssnet) is an author, public speaker, small business owner, mercenary sysadmin, and father of three—not necessarily in that order.

**TWITTER** @jrssnet



← PREVIOUS STORY

NEXT STORY →

## Today on Ars