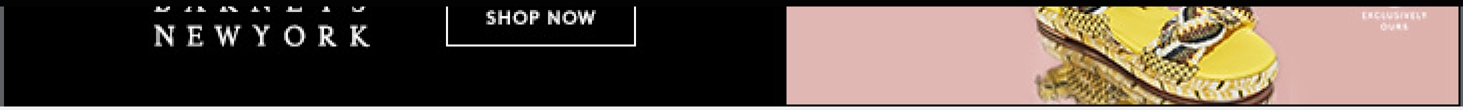


BIZ & IT —

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

Replication

At this point, you have a zpool. That zpool has at least one nice, redundant, self-healing vdev with parity in it. You know how to take snapshots, so now let's look at how to *replicate* those snapshots to *another* machine which is also running ZFS.

Set up SSH keys

This isn't strictly a ZFS step, but you'll need it in order to handle replication the easy way, so we'll go ahead and cover it here. Let's assume you have box1 and box2; your data is on box1 and you want to back it up to box2. Further, let's assume you want to push the backups from box1 to box2, rather than pulling them the other way around. First, generate yourself a root SSH key on box1:

```
me@box:~$ sudo ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/root/.ssh/id_dsa):
Created directory '/root/.ssh'.
```

```
Your identification has been saved in /root/.ssh/id_dsa.
Your public key has been saved in /root/.ssh/id_dsa.pub.
The key fingerprint is:
8f:67:61:ab:4d:be:99:9f:b9:4f:68:25:37:e5:82:ed root@box1
The key's randomart image is:
+--[ DSA 1024]----+
|                 |
|                 |
|                .|
|             o o |
|        S o o * .|
|         + o * o |
|        . * o E  |
|         B + +   |
|        . *o=o.  |
+-----------------+
```

You'll be asked if you want to save your key to the default location /root/.ssh (you do) and if you want to use a passkey (for this example, you don't). Once you're done, it's time to copy your new public key off to box2:

```
me@box1:~$ sudo scp /root/.ssh/id_dsa.pub me@box2:/tmp/
me@box2's password:
id_dsa.pub
```

Now it's time to add box1's public key to the root authorized keys file on box2 and make sure that we allow the use of keys on box2.

```
me@box2:~$ sudo -s
root@box2:~# cat /tmp/id_dsa.pub >> /root/.ssh/authorized_keys
root@box2:~# echo AuthorizedKeysFile %h/.ssh/authorized_keys >> /etc/ssh/sshd_config
```

Now we'll be able to SSH as root with no password from box1 to box2, which is necessary for our next step.

As of right now, box2 has its own zpool (which we named technica, and which *does not* have to be composed of the same number, type, or arrangement of vdevs as our original zpool on box1) but has no actual filesystems on it. I now have a gigabyte of data on ars/jpegs, and I want to replicate that data to box2. Keep in mind that we don't replicate the filesystem itself, we replicate *snapshots.* Let's take a snapshot:

```
me@box1:~$ sudo -s
root@box1:~# zfs snapshot ars/jpegs@1
root@box1:~# zfs list -rt all ars/jpegs
NAME          USED  AVAIL  REFER  MOUNTPOINT
ars/jpegs    1024M   199G  1024M  /ars/jpegs
ars/jpegs@1      0      -  1024M  -
```

Now let's replicate it:

```
root@box1:~# zfs send ars/jpegs@1 | ssh box2 zfs receive technica
```

It's that easy. After the 1GB of data gets done moving across the network, you now have a replicated copy of ars/jpegs on box1 at technica/jpegs on box2:

```
root@box2:~# zfs list -rt all technica/jpegs
NAME               USED  AVAIL  REFER  MOUNTPOINT
technica/jpegs    1024M  1.95T  1024M  /technica/jpegs
technica/jpegs@1      0      -  1024M  -
```

Incremental replication

What about the *next* time we replicate? Well, as long as we haven't gotten rid of snapshot ars/jpegs@1 on box1, we can use it as a *parent* snapshot and do *incremental* replication the next time, which will go much quicker. Let's make a silly little file, take another snapshot, and replicate incrementally:

```
root@box1:~# echo lolz > /ars/jpegs/lolz.txt
root@box1:~# ls -l /ars/jpegs
```

```
-rw-r--r-- 1 root root          5 Jan 23 16:32 lolz.txt

root@box1:~# zfs snapshot ars/jpegs@2
root@box1:~# zfs send -i ars/jpegs@1 ars/jpegs@2 | ssh box2 zfs receive technica/jpegs
```

Notice that this time, we used the -i argument and specified both snapshots. We also used the full path to the *existing* filesystem technica/jpegs in our receive command, since we're receiving an incremental to an existing filesystem, not a full replication to create a new filesystem with. This replication happened pretty much instantaneously—lolz.txt is just a silly little file, after all—and ZFS *already knows* what has or hasn't changed from snapshot @1 to snapshot @2. Since it doesn't have to grovel over the disk looking for changes, it can just immediately start sending them when asked.

Does everything look as we'd expect it to, over on box2?

```
root@box2:~# zfs list -rt all technica/jpegs
NAME               USED  AVAIL  REFER  MOUNTPOINT
technica/jpegs    1024M  1.95T  1024M  /technica/jpegs
technica/jpegs@1   117K      -  1024M  -
technica/jpegs@2      0      -  1024M  -

root@box2:~# ls -lh /technica/jpegs
total 1.0G
-rw-r--r-- 1 root root 1.0G Jan 23 16:23 1G.bin
-rw-r--r-- 1 root root    5 Jan 23 16:32 lolz.txt
```

Exactly as we'd expect: not only a full copy of the original filesystem, but a full copy of the original filesystem *and all of its snapshots* as we replicated them over.

At this point, you can safely get rid of snapshot ars/jpegs@1 on box1 if you'd like to. The next time you replicate to box2, you'll use @2 as a parent for whatever your next snapshot is and so on. This allows you to do some pretty cool stuff, like make a "main" server with expensive, fast storage (but not much of it) and a "backup" server with cheap, slow storage (with plenty of it). You can even keep *lots* of snapshots on your backup server, while destroying them pretty quickly from your "main" server. Pretty powerful stuff.

I have to be honest, the only reason I'm even *mentioning* dedup is I know there'll be a furor in the comments if I don't. There may be one anyway, because the next thing I have to tell you is something you don't want to hear:

You probably don't want to use dedup. Full stop.

Deduplication *sounds* exciting. Stop caring when your users blindly make a copy of a folder with 15G of stuff in it! Don't write more stuff than you have to! Keep more stuff on the same drive! Reap some performance benefits, sometimes, depending! But the problem is, the way ZFS implements dedup, it takes up a *lot* of RAM; unless you have a very specialized machine and a very specialized workload, almost certainly more RAM than you'll be willing to feed it.

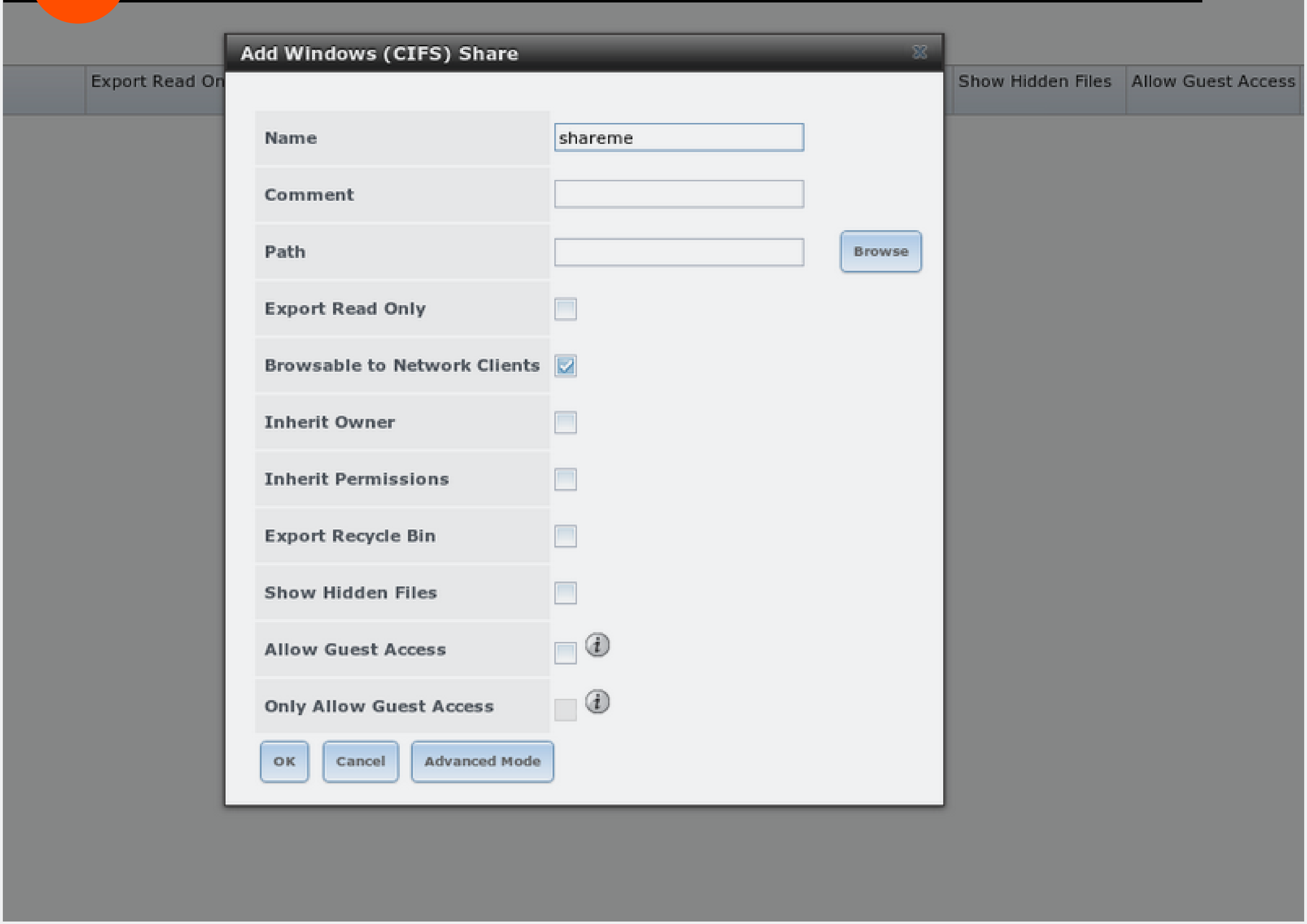
The bottom line: for every 1TB of deduplicated storage, you're going to need roughly 5GB of RAM. And that's for dedup tablespace *alone.* That doesn't count ZFS' normal memory consumption. I've tested this personally. After copying about 6TB of data to a ZFS filesystem with dedup turned on, my RAM consumption went *up* roughly 32GB. This was a special server that has 128GB of RAM, so luckily it could handle it. Even so, I disabled dedup immediately after the test because I wasn't happy with the result.

In most cases, for most users... it's just not worth it. Sorry.

The final takeaway

We've still really only scratched the surface of what ZFS can do. But hopefully, you've seen enough to get you half as interested in ZFS as I am. I've been using ZFS professionally and in production for over five years, and I can honestly say that it's both changed the course of my career and my business. I wouldn't *dream* of going back to the way I did things before ZFS.

For you Windows and Mac users out there (or any Linux users who are allergic to the command line), don't despair and stay tuned! Next in this series, I'll be covering FreeNAS, which is essentially "ZFS on easy mode." It's a ready-to-download, ready-to-use distribution that lets you set up, manage, and configure your own ZFS-powered Network Attached Storage device out of a generic PC with a bunch of hard drives; no command line required.



Add Windows (CIFS) Share

Name	shareme
Comment	
Path	
Export Read Only	□
Browsable to Network Clients	☑
Inherit Owner	□
Inherit Permissions	□
Export Recycle Bin	□
Show Hidden Files	□
Allow Guest Access	□ ⓘ
Only Allow Guest Access	□ ⓘ

ars TECHNICA

y in that
d he's been
a fierce advocate of FOSS ever since. He also also created and maintains http://freebsdwiki.net andhttp://ubuntuwiki.net.

Page: 1 2 3

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

READER COMMENTS 254

SHARE THIS STORY

← PREVIOUS STORY

NEXT STORY →

Related Stories

Today on Ars

Apple's iOS 12 strategy: Take more time to squash the bugs

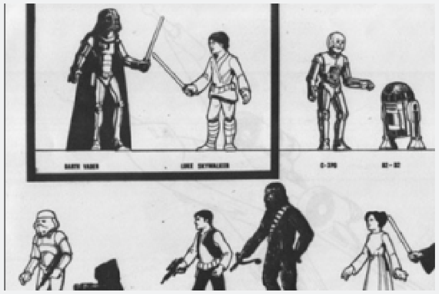
Teaser: Our Apollo series finale is coming tomorrow

Proposed NASA budget takes one small step toward the Moon

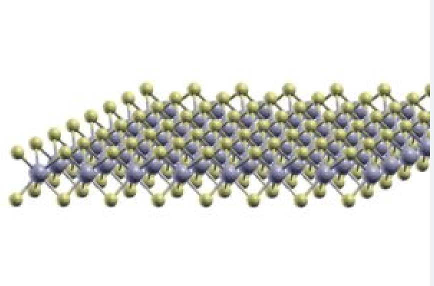
Daylight Saving Time isn't worth it, European Parliament members say

Altered Carbon somehow nails the sci-fi book-to-TV landing on Netfl

Comments on Twitter could now lead to punishment on Twitch



The Toys That Made Us: To make a great toy documentary, embrace Jackie Chan



Scientists identify hundreds of atomically thin materials

ars TECHNICA

RSS FEEDS
VIEW MOBILE SITE
ABOUT US
SUBSCRIBE

CONTACT US
STAFF
ADVERTISE WITH US
REPRINTS

CONDÉ NAST

CNMN Collection
WIRED Media Group