Aaron Toponce

{ 2012 12 21 }

ZFS Administration, Part XIV- ZVOLS

Table of Contents

What is a ZVOL?

A ZVOL is a "ZFS volume" that has been exported to the system as a block device. So far, when dealing with the ZFS filesystem, other than creating our pool, we haven't dealt with block devices at all, even when mounting the

datasets. It's almost like ZFS is behaving like a userspace application more than a filesystem. I mean, on GNU/Linux, when working with filesystems, you're constantly working with block devices, whether they be full disks, partitions, RAID arrays or logical volumes. Yet somehow, we've managed to escape all that with ZFS. Well, not any longer. Now we get our hands dirty with ZVOLs.

A ZVOL is a ZFS block device that resides in your storage pool. This means that the single block device gets to take advantage of your underlying RAID array, such as mirrors or RAID-Z. It gets to take advantage of the copy-on-write benefits, such as snapshots. It gets to take advantage of online scrubbing, compression and data deduplication. It gets to take advantage of the ZIL and ARC. Because it's a legitimate block device, you can do some very interesting things with your ZVOL. We'll look at three of them here- swap, ext4, and VM storage. First, we need to learn how to create a ZVOL.

Creating a ZVOL

To create a ZVOL, we use the "-V" switch with our "zfs create" command, and give it a size. For example, if I wanted to create a 1 GB ZVOL, I could issue the following command. Notice further that there are a couple new symlinks that exist in /dev/zvol/tank/ and /dev/tank/ which points to a new block device in /dev/:

```
# zfs create -V 1G tank/disk1
# ls -l /dev/zvol/tank/disk1
lrwxrwxrwx 1 root root 11 Dec 20 22:10 /dev/zvol/tank/disk1 -> ../../zd144
# ls -l /dev/tank/disk1
lrwxrwxrwx 1 root root 8 Dec 20 22:10 /dev/tank/disk1 -> ../zd144
```

Because this is a full fledged, 100% bona fide block device that is 1 GB in size, we can do anything with it that we would do with any other block device, and we get all the benefits of ZFS underneath. Plus, creating a ZVOL is near instantaneous, regardless of size. Now, I could create a block device with GNU/Linux from a file on the filesystem. For example, if running ext4, I can create a 1 GB file, then make a block device out of it as follows:

```
# fallocate -l 1G /tmp/file.img
# losetup /dev/loop0 /tmp/file.img
```

I now have the block device /dev/loop0 that represents my 1 GB file. Just as with any other block device, I can format it, add it to swap, etc. But it's not as elegant, and it has severe limitations. First off, by default you only have 8 loopback devices for your exported block devices. You can change this number, however. With ZFS, you can create 2^64 ZVOLs by default. Also, it requires a preallocated image, on top of your filesystem. So, you are managing three layers of data: the block device, the file, and the blocks on the filesystem. With ZVOLs, the block device is exported right off the storage pool, just like any other dataset.

Let's look at some things we can do with this ZVOL.

Swap on a ZVOL

Personally, I'm not a big fan of swap. I understand that it's a physical extension of RAM, but swap is only used when RAM fills, spilling the cache. If this is happening regularly and consistently, then you should probably look into getting more RAM. It can act as part of a healthy system, keeping RAM dedicated to what the kernel actively needs. But, when active RAM starts spilling over to swap, then you have "the swap of death", as your disks thrash, trying to keep up with the demands of the kernel. So, depending on your system and needs, you may or may not need swap.

First, let's create 1 GB block device for our swap. We'll call the dataset "tank/swap" to make it easy to identify its intention. Before we begin, let's check out how much swap we currently have on our system with the "free" command:

```
# free
             total       used       free     shared    buffers     cached
Mem:      12327288    8637124    3690164          0     175264    1276812
-/+ buffers/cache:    7185048    5142240
Swap:            0          0          0
```

In this case, we do not have any swap enabled. So, let's create 1 GB of swap on a ZVOL, and add it to the kernel:

```
# zfs create -V 1G tank/swap
# mkswap /dev/zvol/tank/swap
# swapon /dev/zvol/tank/swap
```

```
# free
             total       used       free     shared    buffers     cached
Mem:      12327288    8667492    3659796          0     175268    1276804
-/+ buffers/cache:    7215420    5111868
Swap:      1048572          0    1048572
```

It worked! We have a legitimate Linux kernel swap device on top of ZFS. Sweet. As is typical with swap devices, they don't have a mountpoint. They are either enabled, or disabled, and this swap device is no different.

Ext4 on a ZVOL

This may sound wacky, but you could put another filesystem, and mount it, on top of a ZVOL. In other words, you could have an ext4 formatted ZVOL and mounted to /mnt. You could even partition your ZVOL, and put multiple filesystems on it. Let's do that!

```
# zfs create -V 100G tank/ext4
# fdisk /dev/tank/ext4
( follow the prompts to create 2 partitions- the first 1 GB in size, the second to fill the rest )
# fdisk -l /dev/tank/ext4

Disk /dev/tank/ext4: 107.4 GB, 107374182400 bytes
16 heads, 63 sectors/track, 208050 cylinders, total 209715200 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 8192 bytes
I/O size (minimum/optimal): 8192 bytes / 8192 bytes
Disk identifier: 0x000a0d54

          Device Boot      Start         End      Blocks   Id  System
/dev/tank/ext4p1            2048     2099199     1048576   83  Linux
/dev/tank/ext4p2         2099200   209715199   103808000   83  Linux
```

Let's create some filesystems, and mount them:

```
# mkfs.ext4 /dev/zd0p1
# mkfs.ext4 /dev/zd0p2
# mkdir /mnt/zd0p{1,2}
# mount /dev/zd0p1 /mnt/zd0p1
# mount /dev/zd0p2 /mnt/zd0p2
```

Enable compression on the ZVOL, copy over some data, then take a snapshot:

```
# zfs set compression=lzjb pool/ext4
# tar -cf /mnt/zd0p1/files.tar /etc/
# tar -cf /mnt/zd0p2/files.tar /etc /var/log/
# zfs snapshot tank/ext4@001
```

You probably didn't notice, but you just enabled transparent compression and took a snapshot of your ext4 filesystem. These are two things you can't do with ext4 natively. You also have all the benefits of ZFS that ext4 normally couldn't give you. So, now you regularly snapshot your data, you perform online scrubs, and send it offsite for backup. Most importantly, your data is consistent.

ZVOL storage for VMs

Lastly, you can use these block devices as the backend storage for VMs. It's not uncommon to create logical volume block devices as the backend for VM storage. After having the block device available for Qemu, you attach the block device to the virtual machine, and from its perspective, you have a "/dev/vda" or "/dev/sda" depending on the setup.

If using libvirt, you would have a /etc/libvirt/qemu/vm.xml file. In that file, you could have the following, where "/dev/zd0" is the ZVOL block device:

```
<disk type='block' device='disk'>
  <driver name='qemu' type='raw' cache='none'/>
  <source dev='/dev/zd0'/>
  <target dev='vda' bus='virtio'/>
  <alias name='virtio-disk0'/>
```

```
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
</disk>
```

At this point, your VM gets all the ZFS benefits underneath, such as snapshots, compression, deduplication, data integrity, drive redundancy, etc.

Conclusion

ZVOLs are a great way to get to block devices quickly while taking advantage of all of the underlying ZFS features. Using the ZVOLs as the VM backing storage is especially attractive. However, I should note that when using ZVOLs, you cannot replicate them across a cluster. ZFS is not a clustered filesystem. If you want data replication across a cluster, then you should not use ZVOLs, and use file images for your VM backing storage instead. Other than that, you get all of the amazing benefits of ZFS that we have been blogging about up to this point, and beyond, for whatever data resides on your ZVOL.

Posted by Aaron Toponce on Friday, December 21, 2012, at 6:00 am.

Filed under Debian, Linux, Ubuntu, ZFS. Follow any responses to this post with its comments RSS feed. You can post a comment or trackback from your blog. For IM, Email or Microblogs, here is the Shortlink.

{ 13 } Comments

1. Ahmed Kamal | July 22, 2013 at 2:56 pm | Permalink

   When you put ext4 on top of a ZVOL, and snapshot it .. You say it's "consistent" I guess it's only crash-consistent .. There is no FS/ZVOL integration to ensure better consistency, right