Aaron Toponce

{ 2012 12 17 }

ZFS Administration, Part X- Creating Filesystems

Table of Contents

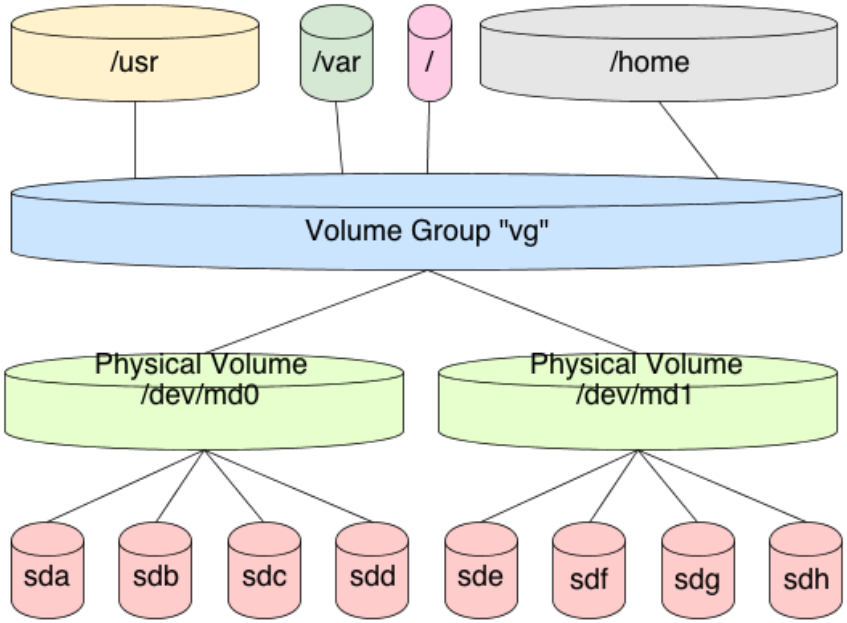
We now begin down the path that is the "bread and butter" of ZFS, known as "ZFS datasets", or filesystems. Previously, up to this point, we've been discussing how to manage our storage pools. But storage pools are not

meant to store data directly. Instead, we should create filesystems that share the same storage system. We'll refer to these filesystems from now as datasets.

Background

First, we need to understand how traditional filesystems and volume management work in GNU/Linux before we can get a thorough understanding of ZFS datasets. To treat this fairly, we need to assemble Linux software RAID, LVM, and ext4 or another Linux kernel supported filesystem together.

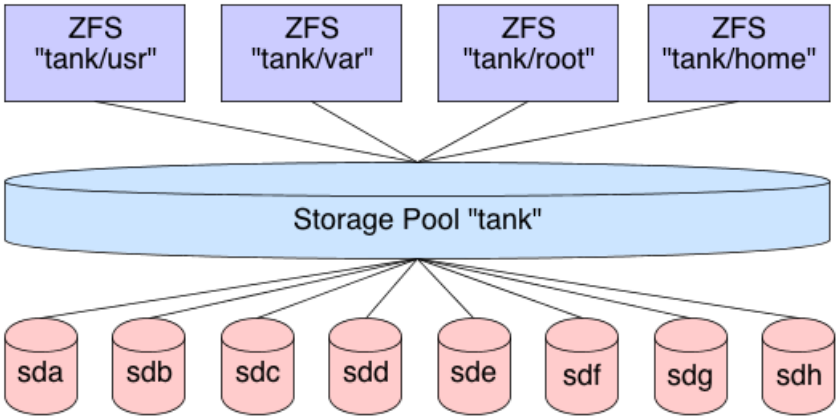
This is done by creating a redundant array of disks, and exporting a block device to represent that array. Then, we format that exported block device using LVM. If we have multiple RAID arrays, we format each of those as well. We then add all these exported block devices to a "volume group" which represents my pooled storage. If I had five exported RAID arrays, of 1 TB each, then I would have 5 TB of pooled storage in this volume group. Now, I need to decide how to divide up the volume, to create logical volumes of a specific size. If this was for an Ubuntu or Debian installation, maybe I would give 100 GB to one logical volume for the root filesystem. That 100 GB is now marked as occupied by the volume group. I then give 500 GB to my home directory, and so forth. Each operation exports a block device, representing my logical volume. It's these block devices that I format with ex4 or a filesystem of my choosing.



Linux RAID, LVM, and filesystem stack. Each filesystem is limited in size.

In this scenario, each logical volume is a fixed size in the volume group. It cannot address the full pool. So, when formatting the logical volume block device, the filesystem is a fixed size. When that device fills, you must resize the logical volume and the filesystem together. This typically requires a myriad of commands, and it's tricky to get just right without losing data.

ZFS handles filesystems a bit differently. First, there is no need to create this stacked approach to storage. We've already covered how to pool the storage, now we well cover how to use it. This is done by creating a dataset in the filesystem. By default, this dataset will have full access to the entire storage pool. If our storage pool is 5 TB in size, as previously mentioned, then our first dataset will have access to all 5 TB in the pool. If I create a second dataset, it too will have full access to all 5 TB in the pool. And so on and so forth.



Each ZFS dataset can use the full underlying storage.

Now, as files are placed in the dataset, the pool marks that storage as unavailable to all datasets. This means that each dataset is aware of what is available in the pool and what is not by all other datasets in the pool. There is no need to create logical volumes of limited size. Each dataset will continue to place files in the pool, until the pool is filled. As the cards fall, they fall. You can, of course, put quotas on datasets, limiting their size, or export ZVOLs, topics we'll cover later.

So, let's create some datasets.

Basic Creation

In these examples, we will assume our ZFS shared storage is named "tank". Further, we will assume that the pool is created with 4 preallocated files of 1 GB in size each, in a RAIDZ-1 array. Let's create some datasets.

```
# zfs create tank/test
# zfs list
NAME         USED  AVAIL  REFER  MOUNTPOINT
tank         175K  2.92G  43.4K  /tank
tank/test   41.9K  2.92G  41.9K  /tank/test
```

Notice that the dataset "tank/test" is mounted to "/tank/test" by default, and that it has full access to the entire pool. Also notice that it is occupying only 41.9 KB of the pool. Let's create 4 more datasets, then look at the output:

```
# zfs create tank/test2
# zfs create tank/test3
# zfs create tank/test4
# zfs create tank/test5
# zfs list
NAME         USED  AVAIL  REFER  MOUNTPOINT
tank         392K  2.92G  47.9K  /tank
tank/test   41.9K  2.92G  41.9K  /tank/test
tank/test2  41.9K  2.92G  41.9K  /tank/test2
tank/test3  41.9K  2.92G  41.9K  /tank/test3
tank/test4  41.9K  2.92G  41.9K  /tank/test4
tank/test5  41.9K  2.92G  41.9K  /tank/test5
```

Each dataset is automatically mounted to its respective mount point, and each dataset has full unfettered access to the storage pool. Let's fill up some data in one of the datasets, and see how that affects the underlying storage:

```
# cd /tank/test3
# for i in {1..10}; do dd if=/dev/urandom of=file$i.img bs=1024 count=$RANDOM &> /dev/null; done
# zfs list
NAME         USED  AVAIL  REFER  MOUNTPOINT
tank         159M  2.77G  49.4K  /tank
tank/test   41.9K  2.77G  41.9K  /tank/test
tank/test2  41.9K  2.77G  41.9K  /tank/test2
tank/test3   158M  2.77G   158M  /tank/test3
tank/test4  41.9K  2.77G  41.9K  /tank/test4
tank/test5  41.9K  2.77G  41.9K  /tank/test5
```

Notice that in my case, "tank/test3" is occupying 158 MB of disk, so according to the rest of the datasets, there is only 2.77 GB available in the pool, where previously there was 2.92 GB. So as you can see, the big advantage here

is that I do not need to worry about preallocated block devices, as I would with LVM. Instead, ZFS manages the entire stack, so it understands how much data has been occupied, and how much is available.

Mounting Datasets

It's important to understand that when creating datasets, you aren't creating exportable block devices by default. This means you don't have something directly to mount. In conclusion, there is nothing to add to your /etc/fstab file for persistence across reboots.

So, if there is nothing to add do the /etc/fstab file, how do the filesystems get mounted? This is done by importing the pool, if necessary, then running the "zfs mount" command. Similarly, we have a "zfs unmount" command to unmount datasets, or we can use the standard "umount" utility:

```
# umount /tank/test5
# mount | grep tank
tank/test on /tank/test type zfs (rw,relatime,xattr)
tank/test2 on /tank/test2 type zfs (rw,relatime,xattr)
tank/test3 on /tank/test3 type zfs (rw,relatime,xattr)
tank/test4 on /tank/test4 type zfs (rw,relatime,xattr)
# zfs mount tank/test5
# mount | grep tank
tank/test on /tank/test type zfs (rw,relatime,xattr)
tank/test2 on /tank/test2 type zfs (rw,relatime,xattr)
tank/test3 on /tank/test3 type zfs (rw,relatime,xattr)
tank/test4 on /tank/test4 type zfs (rw,relatime,xattr)
tank/test5 on /tank/test5 type zfs (rw,relatime,xattr)
```

By default, the mount point for the dataset is "/<pool-name>/<dataset-name>". This can be changed, by changing the dataset property. Just as storage pools have properties that can be tuned, so do datasets. We'll dedicate a full post to dataset properties later. We only need to change the "mountpoint" property, as follows:

```
# zfs set mountpoint=/mnt/test tank/test
# mount | grep tank
tank on /tank type zfs (rw,relatime,xattr)
tank/test2 on /tank/test2 type zfs (rw,relatime,xattr)
tank/test3 on /tank/test3 type zfs (rw,relatime,xattr)
tank/test4 on /tank/test4 type zfs (rw,relatime,xattr)
tank/test5 on /tank/test5 type zfs (rw,relatime,xattr)
tank/test on /mnt/test type zfs (rw,relatime,xattr)
```

Nested Datasets

Datasets don't need to be isolated. You can create nested datasets within each other. This allows you to create namespaces, while tuning a nested directory structure, without affecting the other. For example, maybe you want compression on /var/log, but not on the parent /var. there are other benefits as well, with some caveats that we will look at later.

To create a nested dataset, create it like you would any other, by providing the parent storage pool *and* dataset. In this case we will create a nested log dataset in the test dataset:

```
# zfs create tank/test/log
# zfs list
NAME            USED  AVAIL  REFER  MOUNTPOINT
tank            159M  2.77G  47.9K  /tank
tank/test      85.3K  2.77G  43.4K  /mnt/test
tank/test/log  41.9K  2.77G  41.9K  /mnt/test/log
tank/test2     41.9K  2.77G  41.9K  /tank/test2
tank/test3      158M  2.77G   158M  /tank/test3
tank/test4     41.9K  2.77G  41.9K  /tank/test4
tank/test5     41.9K  2.77G  41.9K  /tank/test5
```

Additional Dataset Administration

Along with creating datasets, when you no longer need them, you can destroy them. This frees up the blocks for use by other datasets, and cannot be reverted without a previous snapshot, which we'll cover later. To destroy a dataset:

```
# zfs destroy tank/test5
# zfs list
NAME            USED  AVAIL  REFER  MOUNTPOINT
tank            159M  2.77G  49.4K  /tank
tank/test      41.9K  2.77G  41.9K  /mnt/test
tank/test/log  41.9K  2.77G  41.9K  /mnt/test/log
tank/test2     41.9K  2.77G  41.9K  /tank/test2
tank/test3      158M  2.77G   158M  /tank/test3
tank/test4     41.9K  2.77G  41.9K  /tank/test4
```

We can also rename a dataset if needed. This is handy when the purpose of the dataset changes, and you want the name to reflect that purpose. The arguments take a dataset source as the first argument and the new name as the last argument. To rename the tank/test3 dataset to music:

```
# zfs rename tank/test3 tank/music
# zfs list
NAME            USED  AVAIL  REFER  MOUNTPOINT
tank            159M  2.77G  49.4K  /tank
tank/music      158M  2.77G   158M  /tank/music
tank/test      41.9K  2.77G  41.9K  /mnt/test
tank/test/log  41.9K  2.77G  41.9K  /mnt/test/log
tank/test2     41.9K  2.77G  41.9K  /tank/test2
tank/test4     41.9K  2.77G  41.9K  /tank/test4
```

Conclusion

This will get you started with understanding ZFS datasets. There are many more subcommands with the "zfs" command that are available, with a number of different switches. Check the manpage for the full listing. However,

even though this isn't a deeply thorough examination of datasets, many more principles and concepts will surface as we work through the series. By the end, you should be familiar enough with datasets that you will be able to manage your entire storage infrastructure with minimal effort.

Posted by Aaron Toponce on Monday, December 17, 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.

{ 15 } Comments

1. Michael | March 20, 2013 at 12:09 pm | Permalink

   Great articles. I may have found a minor typo you may want to fix (unless I read it wrong)... In the part X post I think "my" should be "by" or "by my"... My brain filled it in as "by" and I almost missed the typo... For reference, the portion I am referring to is "... how do the filesystems get mounted? This is done my importing the pool, ..."

   Thanks again !!!

2. Aaron Toponce | March 20, 2013 at 12:37 pm | Permalink

   Fxed! Thanks!

3. Warren Downs | July 1, 2013 at 4:17 pm | Permalink