ZFS/Virtual disks

< ZFS

This article covers some basic tasks and usage of ZFS. It differs from the main article **ZFS** somewhat in that the examples herein are demonstrated on a zpool built from virtual disks. So long as users do not place any critical data on the resulting zpool, they are free to experiment without fear of actual data loss.

Related articles

ZFS

Installing Arch Linux on ZFS

The examples in this article are shown with a set of virtual discs known in ZFS terms as VDEVs. Users may create their VDEVs either on an existing physical disk or in tmpfs (RAMdisk) depending on the amount of free memory on the system.

Note: Using a file as a VDEV is a great method to play with ZFS but isn't viable strategy for storing "real" data.

Contents

Install the ZFS Family of Packages

Due to differences in licencing, ZFS bins and kernel modules are easily distributed from source, but no-so-easily packaged as pre-compiled sets. The requisite packages are available in the AUR and in an unofficial repo. Details are provided on the **ZFS#Installation** article.

Creating and Destroying Zpools

Management of ZFS is pretty simplistic with only two utils needed:

- `/usr/bin/zpool`
- `/usr/bin/zfs`

Mirror

For zpools with just two drives with redundancy, it is recommended to use ZFS in *mirror* mode which functions like a RAID1 mirroring the data. Mirroring can also be used as an alternative to Raidz setups with surprising results. See more on vdev mirroring **here (http://jrs-s.net/2015/02/06/zfs-you-should-use-mirror-vdevs-not-raidz/)**.

RAIDZ1

The minimum number of drives for a RAIDZ1 is three. It's best to follow the "power of two plus parity" recommendation. This is for storage space efficiency and hitting the "sweet spot" in performance. For RAIDZ-1, use three (2+1), five (4+1), or nine (8+1) disks. This example

will use the most simplistic set of (2+1).

Create three x 2G files to serve as virtual hardrives:

```
$ for i in {1..3}; do truncate -s 2G /scratch/$i.img; done
```

Assemble the RAIDZ1:

```
# zpool create zpool raidz1 /scratch/1.img /scratch/2.img /scratch/3.img
```

Notice that a 3.91G zpool has been created and mounted for us:

```
# zfs list
```

```
NAME   USED  AVAIL  REFER  MOUNTPOINT
test   139K  3.91G  38.6K  /zpool
```

The status of the device can be queried:

```
# zpool status zpool
```

```
  pool: zpool
 state: ONLINE
  scan: none requested
config:

        NAME                STATE     READ WRITE CKSUM
           zpool            ONLINE       0     0     0
          raidz1-0          ONLINE       0     0     0
            /scratch/1.img  ONLINE       0     0     0
            /scratch/2.img  ONLINE       0     0     0
            /scratch/3.img  ONLINE       0     0     0
```

```
errors: No known data errors
```

To destroy a zpool:

```
# zpool destroy zpool
```

RAIDZ2 and RAIDZ3

Higher level ZRAIDs can be assembled in a like fashion by adjusting the for statement to create the image files, by specifying "raidz2" or "raidz3" in the creation step, and by appending the additional image files to the creation step.

Summarizing Toponce's guidance:

- RAIDZ2 should use four (2+2), six (4+2), ten (8+2), or eighteen (16+2) disks.
- RAIDZ3 should use five (2+3), seven (4+3), eleven (8+3), or nineteen (16+3) disks.

Linear Span

This setup is for a JBOD, good for 3 or less drives normally, where space is still a concern and your not ready to move to full features of ZFS yet because of it. RaidZ will be your better bet once you achieve enough space to satisfy, since this setup is NOT taking advantage of the

full features of ZFS, but has its roots safely set in a beginning array that will suffice for years until you build up your hard drive collection.

Assemble the Linear Span:

```
# zpool create zpool san /dev/sdd /dev/sde /dev/sdf
```

```
# zpool status zpool
```

```
  pool: zpool
 state: ONLINE
  scan: scrub repaired 0 in 4h22m with 0 errors on Fri Aug 28 23:52:55 2015
config:

        NAME        STATE     READ WRITE CKSUM
        zpool       ONLINE       0     0     0
          sde       ONLINE       0     0     0
          sdd       ONLINE       0     0     0
          sdf       ONLINE       0     0     0

errors: No known data errors
```

Creating and Destroying Datasets

An example creating child datasets and using compression:

- create the datasets

```
# zfs create -p -o compression=on san/vault/falcon/snapshots
# zfs create -o compression=on san/vault/falcon/version
# zfs create -p -o compression=on san/vault/redtail/c/Users
```

- now list the datasets (this was a linear span)

```
$ zfs list
```

Note, there is a huge advantage(file deletion) for making a 3 level dataset. If you have large amounts of data, by separating by datasets, its easier to destroy a dataset than to try and wait for recursive file removal to complete.

Displaying and Setting Properties

Without specifying them in the creation step, users can set properties of their zpools at any time after its creation using `/usr/bin/zfs`.

Show Properties

To see the current properties of a given zpool:

```
# zfs get all zpool
```

Modify properties

Disable the recording of access time in the zpool:

```
# zfs set atime=off zpool
```

Verify that the property has been set on the zpool:

```
# zfs get atime

NAME  PROPERTY     VALUE     SOURCE
zpool  atime        off       local
```

Tip: This option like many others can be toggled off when creating the zpool as well by appending the following to the creation step: -O atime-off

Add Content to the Zpool and Query Compression Performance

Fill the zpool with files. For this example, first enable compression. ZFS uses many compression types, including, lzjb, gzip, gzip-N, zle, and lz4. Using a setting of simply 'on' will call the default algorithm (lzjb) but lz4 is a nice alternative. See the zfs man page for more.

```
# zfs set compression=lz4 zpool
```

In this example, the linux source tarball is copied over and since lz4 compression has been enabled on the zpool, the corresponding compression ratio can be queried as well.

```
$ wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.11.tar.xz
$ tar xJf linux-3.11.tar.xz -C /zpool
```

To see the compression ratio achieved:

```
# zfs get compressratio
```

```
NAME      PROPERTY       VALUE  SOURCE
zpool  compressratio  2.32x  -
```

Simulate a Disk Failure and Rebuild the Zpool

To simulate catastrophic disk failure (i.e. one of the HDDs in the zpool stops functioning), zero out one of the VDEVs.

```
$ dd if=/dev/zero of=/scratch/2.img bs=4M count=1 2>/dev/null
```

Since we used a blocksize (bs) of 4M, the once 2G image file is now a mere 4M:

```
$ ls -lh /scratch
```

```
total 317M
-rw-r--r-- 1 facade users 2.0G Oct 20 09:13 1.img
-rw-r--r-- 1 facade users 4.0M Oct 20 09:09 2.img
-rw-r--r-- 1 facade users 2.0G Oct 20 09:13 3.img
```

The zpool remains online despite the corruption. Note that if a physical disc does fail, dmesg and related logs would be full of errors. To detect when damage occurs, users must execute a scrub operation.

```
# zpool scrub zpool
```

Depending on the size and speed of the underlying media as well as the amount of data in the zpool, the scrub may take hours to complete. The status of the scrub can be queried:

```
# zpool status zpool
```

```
  pool: zpool
 state: DEGRADED
status: One or more devices could not be used because the label is missing or
        invalid.  Sufficient replicas exist for the pool to continue
        functioning in a degraded state.
action: Replace the device using 'zpool replace'.
   see: http://zfsonlinux.org/msg/ZFS-8000-4J
  scan: scrub repaired 0 in 0h0m with 0 errors on Sun Oct 20 09:13:39 2013
config:

        NAME                STATE     READ WRITE CKSUM
           zpool            DEGRADED     0     0     0
          raidz1-0          DEGRADED     0     0     0
            /scratch/1.img  ONLINE       0     0     0
            /scratch/2.img  UNAVAIL      0     0     0  corrupted data
            /scratch/3.img  ONLINE       0     0     0

errors: No known data errors
```

Since we zeroed out one of our VDEVs, let's simulate adding a new 2G HDD by creating a new image file and adding it to the zpool:

```
$ truncate -s 2G /scratch/new.img
# zpool replace zpool /scratch/2.img /scratch/new.img
```

Upon replacing the VDEV with a new one, zpool rebuilds the data from the data and parity info in the remaining two good VDEVs. Check the status of this process:

```
# zpool status zpool
```

```
  pool: zpool
 state: ONLINE
  scan: resilvered 117M in 0h0m with 0 errors on Sun Oct 20 09:21:22 2013
config:

        NAME                  STATE     READ WRITE CKSUM
           zpool              ONLINE       0     0     0
          raidz1-0            ONLINE       0     0     0
            /scratch/1.img    ONLINE       0     0     0
            /scratch/new.img  ONLINE       0     0     0
            /scratch/3.img    ONLINE       0     0     0

errors: No known data errors
```

Snapshots and Recovering Deleted Files

Since ZFS is a copy-on-write filesystem, every file exists the second it is written. Saving changes to the very same file actually creates another copy of that file (plus the changes made). Snapshots can take advantage of this fact and allow users access to older versions of files provided a snapshot has been taken.

Note: When using snapshots, many Linux programs that report on filesystem space such as **df** will report inaccurate results due to the unique way snapshots are used on ZFS. The

output of `/usr/bin/zfs list` will deliver an accurate report of the amount of available and free space on the zpool.

To keep this simple, we will create a dataset within the zpool and snapshot it. Snapshots can be taken either of the entire zpool or of a dataset within the pool. They differ only in their naming conventions:

Snapshot Target	Snapshot Name
Entire zpool	zpool@snapshot-name
Dataset	zpool/dataset@snapshot-name

Make a new data set and take ownership of it.

```
# zfs create zpool/docs
# chown facade:users /zpool/docs
```

Note: The lack of a proceeding / in the create command is intentional, not a typo!

Time 0

Add some files to the new dataset (/zpool/docs):

```
$ wget -O /zpool/docs/Moby_Dick.txt  http://www.gutenberg.org/ebooks/2701.txt.utf-8
$ wget -O /zpool/docs/War_and_Peace.txt http://www.gutenberg.org/ebooks/2600.txt.utf-8
$ wget -O /zpool/docs/Beowulf.txt http://www.gutenberg.org/ebooks/16328.txt.utf-8
```

```
# zfs list
```

```
NAME           USED  AVAIL  REFER  MOUNTPOINT
zpool         5.06M  3.91G  40.0K  /zpool
zpool/docs    4.92M  3.91G  4.92M  /zpool/docs
```

This is showing that we have 4.92M of data used by our books in /zpool/docs.

Time +1

Now take a snapshot of the dataset:

```
# zfs snapshot zpool/docs@001
```

Again run the list command:

```
# zfs list
```

```
NAME           USED  AVAIL  REFER  MOUNTPOINT
zpool         5.07M  3.91G  40.0K  /zpool
zpool/docs    4.92M  3.91G  4.92M  /zpool/docs
```

Note that the size in the USED col did not change showing that the snapshot take up no space in the zpool since nothing has changed in these three files.

We can list out the snapshots like so and again confirm the snapshot is taking up no space, but instead **refers to** files from the originals that take up, 4.92M (their original size):

```
# zfs list -t snapshot
```

```
NAME               USED  AVAIL  REFER  MOUNTPOINT
zpool/docs@001      0      -  4.92M  -
```

Time +2

Now let's add some additional content and create a new snapshot:

```
$ wget -O /zpool/docs/Les_Mis.txt http://www.gutenberg.org/ebooks/135.txt.utf-8
# zfs snapshot zpool/docs@002
```

Generate the new list to see how the space has changed:

```
# zfs list -t snapshot
```

```
NAME               USED  AVAIL  REFER  MOUNTPOINT
zpool/docs@001  25.3K      -  4.92M  -
zpool/docs@002      0      -  8.17M  -
```

Here we can see that the 001 snapshot takes up 25.3K of metadata and still points to the original 4.92M of data, and the new snapshot takes-up no space and refers to a total of 8.17M.

Time +3

Now let's simulate an accidental overwrite of a file and subsequent data loss:

```
$ echo "this book sucks" > /zpool/docs/War_and_Peace.txt
```

Again, take another snapshot:

```
# zfs snapshot zpool/docs@003
```

Now list out the snapshots and notice the amount of referred to decreased by about 3.1M:

```
# zfs list -t snapshot
```

```
NAME              USED  AVAIL  REFER  MOUNTPOINT
zpool/docs@001  25.3K      -  4.92M  -
zpool/docs@002  25.5K      -  8.17M  -
zpool/docs@003      0      -  5.04M  -
```

We can easily recover from this situation by looking inside one or both of our older snapshots for good copy of the file. ZFS stores its snapshots in a hidden directory under the zpool: `/zpool/files/.zfs/snapshot`:

```
$ ls -l /zpool/docs/.zfs/snapshot
```

```
total 0
dr-xr-xr-x 1 root root 0 Oct 20 16:09 001
dr-xr-xr-x 1 root root 0 Oct 20 16:09 002
dr-xr-xr-x 1 root root 0 Oct 20 16:09 003
```

We can copy a good version of the book back out from any of our snapshots to any location on or off the zpool:

```
% cp /zpool/docs/.zfs/snapshot/002/War_and_Peace.txt /zpool/docs
```

Note: Using <TAB> for autocompletion will not work by default but can be changed by modifying the *snapdir* property on the pool or dataset.

```
# zfs set snapdir=visible zpool/docs
```

Now enter a snapshot dir or two:

```
$ cd /zpool/docs/.zfs/snapshot/001
$ cd /zpool/docs/.zfs/snapshot/002
```

Repeat the df command:

```
$ df -h | grep zpool
zpool           4.0G     0  4.0G   0% /zpool
zpool/docs      4.0G  5.0M  4.0G   1% /zpool/docs
zpool/docs@001  4.0G  4.9M  4.0G   1% /zpool/docs/.zfs/snapshot/001
zpool/docs@002  4.0G  8.2M  4.0G   1% /zpool/docs/.zfs/snapshot/002
```

Note: Seeing each dir under .zfs the user enters is reversible if the zpool is taken offline and then remounted or if the server is rebooted.

For example:

```
# zpool export zpool
# zpool import -d /scratch/ zpool
$ df -h | grep zpool
zpool         4.0G     0  4.0G   0% /zpool
zpool/docs    4.0G  5.0M  4.0G   1% /zpool/docs
```

Time +4

Now that everything is back to normal, we can create another snapshot of this state:

```
# zfs snapshot zpool/docs@004
```

And the list now becomes:

```
# zfs list -t snapshot
```

```
NAME              USED  AVAIL  REFER  MOUNTPOINT
zpool/docs@001  25.3K      -  4.92M  -
zpool/docs@002  25.5K      -  8.17M  -
zpool/docs@003   155K      -  5.04M  -
zpool/docs@004     0      -  8.17M  -
```

Listing Snapshots

Note, this simple but important command is missing frequently from other articles on the subject, so its worth mention.

To list any snapshots on your system, run the following command

```
$ zfs list -t snapshot
```

Deleting Snapshots

The limit to the number of snapshots users can save is 2^64. User can delete a snapshot like so:

```
# zfs destroy zpool/docs@001
```

```
# zfs list -t snapshot
```

```
NAME               USED  AVAIL  REFER  MOUNTPOINT
zpool/docs@002  3.28M      -  8.17M  -
zpool/docs@003   155K      -  5.04M  -
zpool/docs@004      0      -  8.17M  -
```

Troubleshooting

If your system is not configured to load the zfs pool upon boot, or for whatever reason you want to manually remove and add back the pool, or if you have lost your pool completely, a convenient way is to use import/export.

If your pool was named <pool>

```
# zpool import pool
```

If you have any problems accessing your pool at any time, try export and reimport.

```
# zfs export pool
# zfs import pool
```