ZFS

ZFS

ZFS support was added to Ubuntu Wily 15.10 as a technology preview and comes fully supported in Ubuntu Xenial 16.04. Note that ZFS is only supported on 64 bit architectures. Also note that ZFS is only supported for data storage, not the root filesystem.

A minimum of 2GB of free memory is required to run ZFS, however it is recommended to use ZFS on a system with at least 8GB of memory.

To install ZFS, use:

```
sudo apt install zfsutils-linux
```

Below is a quick overview of ZFS, this is intended as a getting started primer. For further information on ZFS, please refer to some excellent documentation written by Aaron Toponce.

NOTE

For the save of brevity, devices in this document are referred to as /dev/sda /dev/sdb etc. One should avoid this and use a full device name path using /dev/disk/by-uuid to uniquely identify drives to avoid boot time failures if device name mappings change.

ZFS Virtual Devices (ZFS VDEVs)

A VDEV is a meta-device that can represent one or more devices. ZFS supports 7 different types of VDEV:

- File - a pre-allocated file
- Physical Drive (HDD, SDD, PCIe NVME, etc)
- Mirror - a standard RAID1 mirror
- ZFS software raidz1, raidz2, raidz3 'distributed' parity based RAID
- Hot Spare - hot spare for ZFS software raid.
- Cache - a device for level 2 adaptive read cache (ZFS L2ARC)
- Log - ZFS Intent Log (ZFS ZIL)

VDEVS are dynamically striped by ZFS. A device can be added to a VDEV, but cannot be removed from it.

ZFS Pools

A zpool is a pool of storage made from a collection of VDEVS. One or more ZFS file systems can be created from a ZFS pool.

In the following example, a pool named "pool-test" is created from 3 physical drives:

```
$ sudo zpool create pool-test /dev/sdb /dev/sdc /dev/sdd
```

Striping is performed dynamically, so this creates a zero redundancy RAID-0 pool.

Notice: If you are managing many devices, it can be easy to confuse them, so you should probably prefer /dev/disk/by-id/ names, which often use serial numbers of drives. The examples here should not suggest that 'sd_' names are preferred. They merely make examples herein easier to read.

One can see the status of the pool using the following command:

```
$ sudo zpool status pool-test
```

...and destroy it using:

```
$ sudo zpool destroy pool-test
```

A 2 x 2 mirrored zpool example

The following example, we create a zpool containing a VDEV of 2 drives in a mirror:

```
$ sudo zpool create mypool mirror /dev/sdc /dev/sdd
```

next, we add another VDEV of 2 drives in a mirror to the pool:

```
$ sudo zpool add mypool mirror /dev/sde /dev/sdf -f

$ sudo zpool status
  pool: mypool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            sdc     ONLINE       0     0     0
            sdd     ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            sde     ONLINE       0     0     0
            sdf     ONLINE       0     0     0
```

In this example:

- /dev/sdc, /dev/sdd, /dev/sde, /dev/sdf are the physical devices
- mirror-0, mirror-1 are the VDEVs
- mypool is the pool

There are plenty of other ways to arrange VDEVs to create a zpool.

A single file based zpool example

In the following example, we use a single 2GB file as a VDEV and make a zpool from just this one VDEV:

```
$ dd if=/dev/zero of=example.img bs=1M count=2048
$ sudo zpool create pool-test /home/user/example.img
$ sudo zpool status
  pool: pool-test
 state: ONLINE
  scan: none requested
config:

        NAME                      STATE     READ WRITE CKSUM
        pool-test                 ONLINE       0     0     0
          /home/user/example.img  ONLINE       0     0     0
```

In this example:

- /home/user/example.img is a file based VDEV
- pool-test is the pool

RAID

ZFS offers different RAID options:

Striped VDEVS

This is equivalent to RAID0. This has no parity and no mirroring to rebuild the data. This is not recommended because of the risk of losing data if a drive fails. Example, creating a striped pool using 4 VDEVs:

```
$ sudo zpool create example /dev/sdb /dev/sdc /dev/sdd /dev/sde
```

Mirrored VDEVs

Much like RAID1, one can use 2 or more VDEVs. For N VDEVs, one will have to have N-1 fail before data is lost. Example, creating mirrored pool with 2 VDEVs

$ sudo zpool create example mirror /dev/sdb /dev/sdc

Striped Mirrored VDEVs

Much like RAID10, great for small random read I/O. Create mirrored pairs and then stripe data over the mirrors. Example, creating striped 2 x 2 mirrored pool:

```
sudo zpool create example mirror /dev/sdb /dev/sdc mirror /dev/sdd /dev/sde
```

or:

```
sudo zpool create example mirror /dev/sdb /dev/sdc
sudo zpool add example mirror /dev/sdd /dev/sde
```

RAIDZ

Like RAID5, this uses a variable width strip for parity. Allows one to get the most capacity out of a bunch of disks with parity checking with a sacrifice to some performance. Allows a single disk failure without losing data. Example, creating a 4 VDEV RAIDZ:

```
$ sudo zpool create example raidz /dev/sdb /dev/sdc /dev/sdd /dev/sde
```

RAIDZ2

Like RAID6, with double the parity for 2 disk failures with performance similar to RAIDZ. Example, create a 2 parity 5 VDEV pool:

```
$ sudo zpool create example raidz2 /dev/sdb /dev/sdc /dev/sdd /dev/sde /dev/sdf
```

RAIDZ3

3 parity bits, allowing for 3 disk failures before losing data with performance like RAIDZ2 and RAIDZ. Example, create a 3 parity 6 VDEV pool:

```
$ sudo zpool create example raidz3 /dev/sdb /dev/sdc /dev/sdd /dev/sde /dev/sdf /dev/sdg
```

Nested RAIDZ

Like RAID50, RAID60, striped RAIDZ volumes. This is better performing than RAIDZ but at the cost of reducing capacity. Example, 2 x RAIDZ:

```
$ sudo zpool create example raidz /dev/sdb /dev/sdc /dev/sdd /dev/sde
$ sudo zpool add example raidz /dev/sdf /dev/sdg /dev/sdh /dev/sdi
```

ZFS Intent Logs

ZIL (ZFS Intent Log) drives can be added to a ZFS pool to speed up the write capabilities of any level of ZFS RAID. One normally would use a fast SSD for the ZIL. Conceptually, ZIL is a logging mechanism where data and metadata to be the written is stored, then later flushed as a transactional write. In reality, the ZIL is more complex than this and described in detail here. One or more drives can be used for the ZIL.

For example, to add a SSDs to the pool 'mypool', use:

```
$ sudo zpool add mypool log /dev/sdg -f
```

ZFS Cache Drives

Cache devices provide an additional layer of caching between main memory and disk. They are especially useful to improve random-read performance of mainly static data.

Fox example, to add a cache drive /dev/sdh to the pool 'mypool', use:

```
$ sudo zpool add mypool cache /dev/sdh -f
```

ZFS file systems

ZFS allows one to create a maximum of 2^64 file systems per pool. In the following example, we create two file systems in the pool 'mypool':

```
sudo zfs create mypool/tmp
sudo zfs create mypool/projects
```

and to destroy a file system, use:

```
sudo zfs destroy mypool/tmp
```

Each ZFS file systems can have properties set, for example, setting a maximum quota of 10 gigabytes:

```
sudo zfs set quota=10G mypool/projects
```

or adding using compression:

```
sudo zfs set compression=on mypool/projects
```

ZFS Snapshots

A ZFS snapshot is a read-only copy of ZFS file system or volume. It can be used to save the state of a ZFS file system at a point of time, and one can roll back to this state at a later date. One can even extract files from a snapshot and not need to perform a complete roll back.

In the following example, we snapshot the mypool/projects file system:

```
$ sudo zfs snapshot -r mypool/projects@snap1
```

..and we can see the collection of snapshots using:

```
$ sudo zfs list -t snapshot
NAME                      USED  AVAIL  REFER  MOUNTPOINT
mypool/projects@snap1    8.80G      -  8.80G  -
```

Now lets 'accidentally' destroy all the files and then roll back:

```
$ rm -rf /mypool/projects
$ sudo zfs rollback mypool/projects@snap1
```

One can remove a snapshot using the following:

```
$ sudo zfs destroy mypool/projects@snap1
```

ZFS Clones

A ZFS clone is a writeable copy of a file system with the initial content of the clone being identical to the original file system. A ZFS clone can only be created from a ZFS snapshot and the snapshot cannot be destroyed until the clones created from it are also destroyed.

For example, to clone mypool/projects, first make a snapshot and then clone:

```
$ sudo zfs snapshot -r mypool/projects@snap1
$ sudo zfs clone mypool/projects@snap1 mypool/projects-clone
```

ZFS Send and Receive

ZFS send sends a snapshot of a filesystem that can be streamed to a file or to another machine. ZFS receive takes this stream and will write out the copy of the snapshot back as a ZFS filesystem. This is great for backups or sending copies over the network (e.g. using ssh) to copy a file system.

For example, make a snapshot and save it to a file:

```
sudo zfs snapshot -r mypool/projects@snap2
sudo zfs send mypool/projects@snap2 > ~/projects-snap.zfs
```

..and receive it back:

```
sudo zfs receive -F mypool/projects-copy < ~/projects-snap.zfs
```

ZFS Ditto Blocks

Ditto blocks create more redundant copies of data to copy, just for more added redundancy. With a storage pool of just one device, ditto blocks are spread across the device, trying to place the blocks at least 1/8 of the disk apart. With multiple devices in a pool, ZFS tries to spread ditto blocks across separate VDEVs. 1 to 3 copies can be can be set. For example, setting 3 copies on mypool/projects:

```
$ sudo zfs set copies=3 mypool/projects
```

ZFS Deduplication

ZFS dedup will discard blocks that are identical to existing blocks and will instead use a reference to the existing block. This saves space on the device but comes at a large cost to memory. The dedup in-memory table uses ~320 bytes per block. The greater the table is in size, the slower write performance becomes.

For example, enable dedup on mypool/projects, use:

```
$ sudo zfs set dedup=on mypool/projects
```

For more pros/cons of deduping, refer to http://constantin.glez.de/blog/2011/07/zfs-dedupe-or-not-dedupe. Deduplication is almost never worth the performance penalty.

ZFS Pool Scrubbing

To initiate an explicit data integrity check on a pool one uses the zfs scrub command. For example, to scrub pool 'mypool':

```
$ sudo zpool scrub mypool
```

one can check the status of the scrub using zpool status, for example:

```
$ sudo zpool status -v mypool
```

Data recovery, a simple example

Let's assume we have a 2 x 2 mirror'd zpool:

```
$ sudo zpool create mypool mirror /dev/sdc /dev/sdd mirror /dev/sde /dev/sdf -f
$ sudo zpool status
  pool: mypool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            sdc     ONLINE       0     0     0
            sdd     ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            sde     ONLINE       0     0     0
            sdf     ONLINE       0     0     0
```

Now populate it with some data and check sum the data:

```
$ dd if=/dev/urandom of=/mypool/random.dat bs=1M count=4096
$ md5sum /mypool/random.dat
f0ca5a6e2718b8c98c2e0fdabd83d943  /mypool/random.dat
```

Now we simulate catastrophic data loss by overwriting one of the VDEV devices with zeros:

```
$ sudo dd if=/dev/zero of=/dev/sde bs=1M count=8192
```

And now initiate a scrub:

```
$ sudo zpool scrub mypool
```

And check the status:

```
$ sudo zpool status
  pool: mypool
 state: ONLINE
status: One or more devices has experienced an unrecoverable error.  An
        attempt was made to correct the error.  Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the errors
        using 'zpool clear' or replace the device with 'zpool replace'.
   see: http://zfsonlinux.org/msg/ZFS-8000-9P
  scan: scrub in progress since Tue May 12 17:34:53 2015
    244M scanned out of 1.91G at 61.0M/s, 0h0m to go
    115M repaired, 12.46% done
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            sdc     ONLINE       0     0     0
            sdd     ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            sde     ONLINE       0     0   948  (repairing)
            sdf     ONLINE       0     0     0
```

...now let us remove the drive from the pool:

```
$ sudo zpool detach mypool /dev/sde
```

..hot swap it out and add a new one back:

```
$ sudo zpool attach mypool /dev/sdf /dev/sde  -f
```

..and initiate a scrub to repair the 2 x 2 mirror:

```
$ sudo zpool scrub mypool
```

ZFS compression

As mentioned earlier, one can compress data automatically with ZFS. With the speed of modern CPUs this is a useful option as reduced data size means less data to physically read and write and hence faster I/O. ZFS offers a comprehensive range of compression methods. The default is lz4 (a high performance replacement of lzjb) that offers faster compression/decompression to lzjb and slightly higher compression too. One can change the compression level, e.g.

```
sudo zfs set compression=gzip-9 mypool
```

or even the compression type:

```
sudo zfs set compression=lz4 mypool
```

and check on the compression level using:

```
sudo zfs get compressratio
```

lz4 is significantly faster than the other options while still performing well; lz4 is the safest choice.

Kernel/Reference/ZFS (last edited 2017-09-20 16:45:06 by colin-king @ localhost[127.0.0.1]:colin-king)