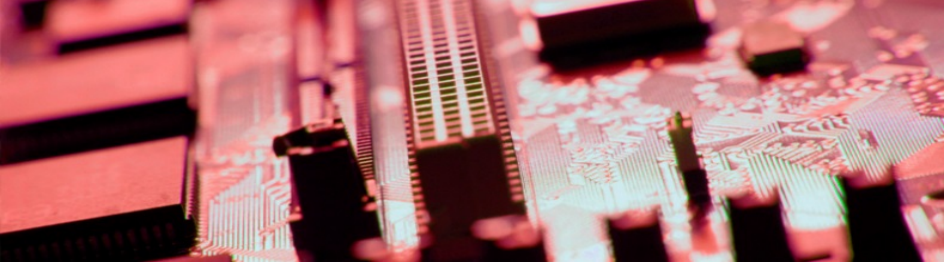


LCI workshop
on high performance clustered computing

Linux Clusters Institute: ZFS Concepts, Features, Administration

Georgia Tech, August 15th – 18th 2017

J.D. Maloney | Storage Engineer
National Center for Supercomputing Applications (NCSA)
malone12@illinois.edu

NCSA

Quick History of ZFS

- Developed by Sun Microsystems
  - Released initially in late 2005 with OpenSolaris
  - Oracle (owner of Sun), stopped release of source code in 2010
  - This prompted the formation of OpenZFS which is prevalent to this day
- Sometimes called the Zettabyte File System
- Focus on data integrity and reliability

OpenZFS

Why ZFS?

- Popular within the HPC community
- Building block for other file systems
  - Some large parallel file systems (Lustre, Gluster, others) can run on top of ZFS
  - Leverage it to combine disks to create LUNs without expensive controllers
- Fairly mature file system
- Has strengths in data integrity and validity that are important for reliable file systems
- Flexibility and fault tolerance

ZFS Concepts

Functionality

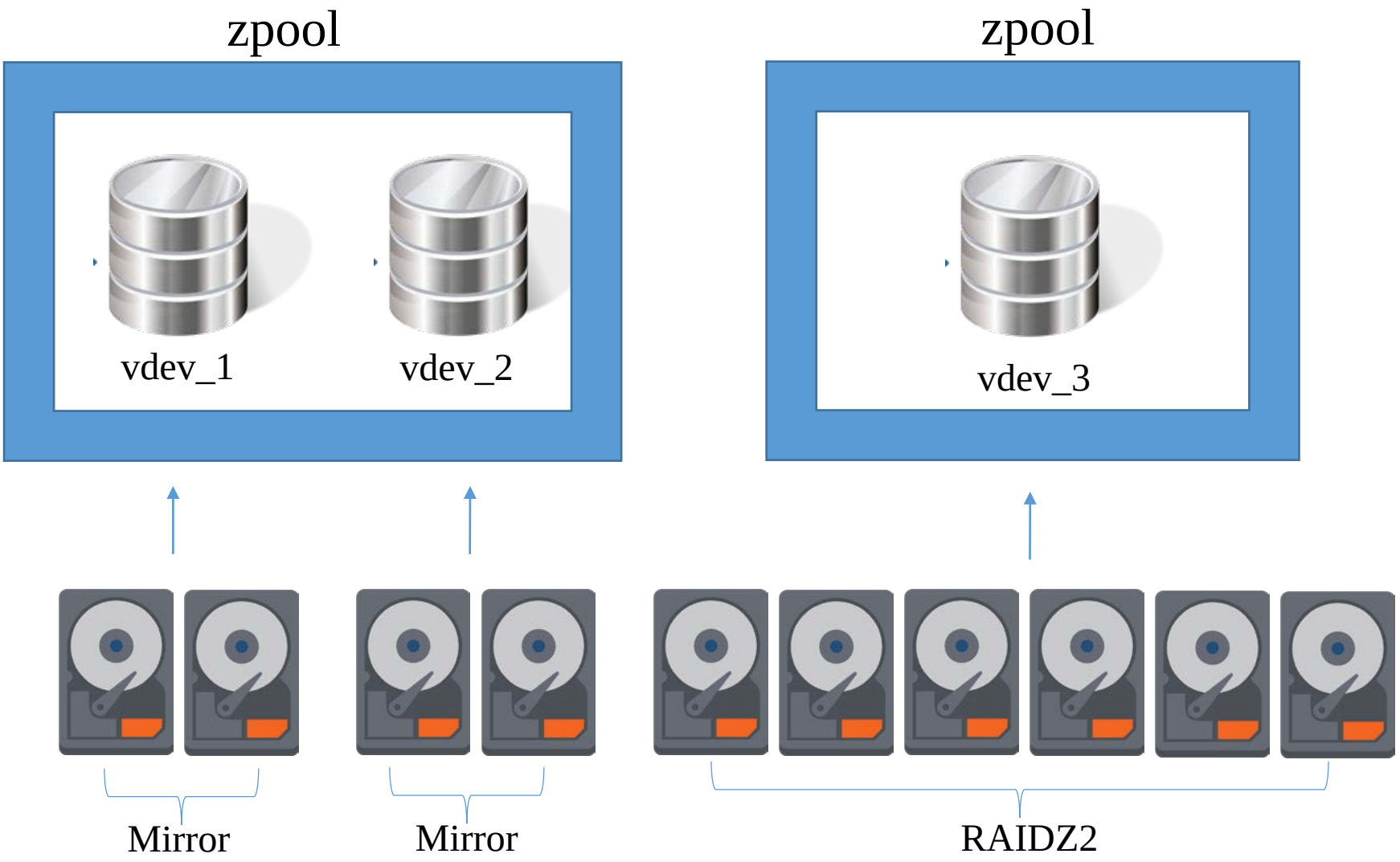
- Software RAID (similar in function to mdam, btrfs, etc.)
  - Combines individual drives within a single system
- Adds redundancy to gain tolerance to individual drive failures while maintaining data availability and integrity
- Requires no additional hardware to implement
  - RAID controller cards are heavily discouraged, use HBA's instead
  - Many RAID controllers have "IT Mode" or JBOD mode to effectively turn them into an HBA
- Supported on many Operating Systems
  - RHEL, CentOS, Ubuntu, Debian, MacOS, freeBSD, Illumos

Basic Architecture

- Individual drives are combined together to create a ”vdev” (virtual device)
- These vdevs form zpools or zvols that are presented to the operating system
- If multiple vdevs are in the same zpool, data is always striped across all of them
- There are multiple ways to combine disks to form a vdev
  - Stripe
  - Mirror
  - raidz,raidz2,raidz3
- Supports datasets – “child file systems” that get some of their own properties

ZFS vdev Layouts

- Stripe
  - Simple as it sounds, data striped across drives
- Mirror
  - Mirroring of two (or more) drives together
- RAIDZ/RAIDZ2/RAIDZ3
  - Equivalent to RAID 5, RAID 6, and a triple parity version of RAID (RAID 7 is actually something else) respectively
- Combining these vdev layouts creates other common RAID types
  - Multiple mirrors in a zpool effectively creates a RAID 10 equivalent
  - Multiple RAIDZ or RAIDZ2 vdevs in a single pool is effectively RAID 50 or RAID 60


zpool
zpool
vdev_1
vdev_2
vdev_3
Mirror
Mirror
RAIDZ2

Architecture Limitations

- Planning for expandability is important
- When adding disks to a zpool, you add them by vdev
- Having uniform vdevs (same redundancy type, drive count) while not required is usually recommended
- No ability to force balance of data across vdevs when new ones are added
  - However, ZFS will favor emptier vdevs for I/O priority. Helps rebalance, but hurts performance
- vdev geometry can not be changed post-creation

Architecture Benefits

- Writes headers to drives in the pool with pool information
  - This allows you to take drives out of one system, mix them up, put them in another and the zpool will import just fine
- Fast resynchronization for zfs offline/online or temporarily removed drive
  - ZFS will be able to detect the drive has mostly all the data it needs and will get it caught back up without having to fully rebuild drive
- Takes random I/O from application and can turn it into synchronous writes to disk thanks to ARC
- No lock-in with proprietary RAID cards, any system where drives can attach and ZFS can be installed can read the pool

Caching Architecture: Writes

- ZFS allows for dedicated caching drives for synchronous writes
  - ZIL/SLOG (ZFS Intent Log)
  - Caches data on files that are smaller than 64KB, larger flushed to disk
  - Best to use use two (mirrored) very fast SSDs (SATA/SAS/NVME), needs to have power-loss protection
  - Only need low capacity, 5GB would suffice
  - Not used by all applications: most databases, NFS, and ISCSI targets do use the ZIL, plain file movement (rsync, cp, scp, etc.) will NOT use the ZIL

Caching Architecture: Reads

- ZFS is known to be memory hungry as it uses half of RAM for ARC (Adjustable Replacement Cache)
  - This memory usage footprint can be limited via configuration tunable
  - Size of ARC does respond to kernel requests so it grows/shrinks as needed on its own
- ZFS allows for dedicated caching drives for reads in addition to ARC, non as L2ARC (Level 2 ARC)
  - Should be put on fast SSD
  - Does NOT need to be mirrored or have power loss protection as it is flushed on reboot anyway

Caching Architecture: Diagram

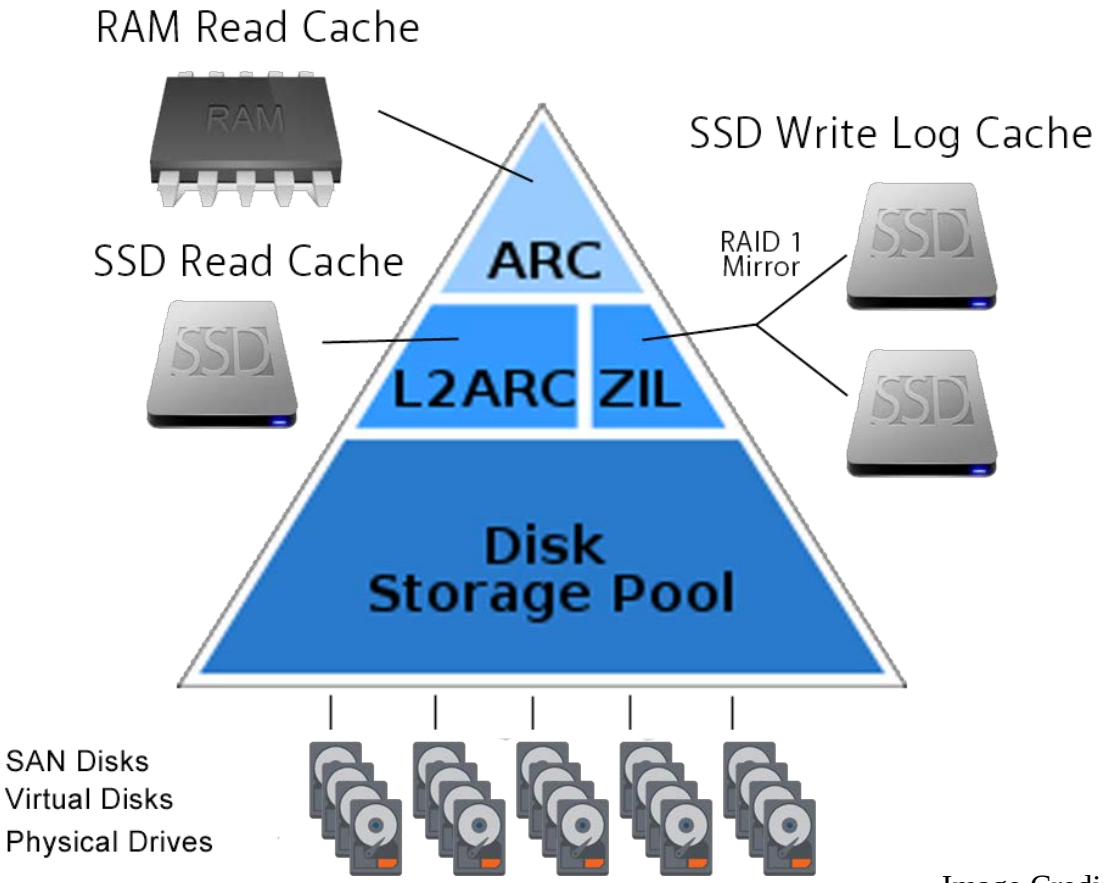


Image Credit: softnas.com

Recovering from Drive Failure

- When drive failure occurs a vdev goes into a degraded state
- Hot Spares are supported in ZFS if you want to have a drive ready in case you can't get to server physically
- Once the new drive is in place, a command is run and all drives in that vdev, no matter the vdev geometry, will participate in providing data to the new drive to rebuild the data
  - This is called a "resilver" and it's progress can be monitored via ZFS commands

Scrubbing the Pool

- To protect against bit-rot, or corruption of other kinds ZFS has a built in mechanism called a “scrub”
  - This walks the zpool looking for corruption
  - Can be scheduled through cron to run
- Scrubs on multiple zpools can be run simultaneously
- Occur while the file system is online and mounted so there is no downtime while scrubs are running on the file system
  - This is a unique power ZFS has in relation to other journaled file systems
- Scrubs will heal any inconsistency it finds to ensure data is correct

ZFS Features

Copy-on-Write (COW)

- When modifying a block, instead of modifying the live block, a new one is created and a pointer updated once the new block is written. Old block is then released back to the file system
  - This keeps the file system consistent in case of sudden system power loss
  - Great for data integrity and stability
  - Allows for additional file system features (eg. snapshots)
- Down side of this is it can lead to fragmentation on the disks
  - This manifests itself as the file system fills and the file system has to spend more time looking for available blocks and they're less contiguous
  - Leads to slower performance with file systems close to full
  - No easy way to defragment the file system

Snapshots

- Functions just as it sounds, takes a snapshot at a moment in time on the file system
- This allows a pool to get rolled back to the moment in time when the snapshot was taken
  - Protects against accidental file deletion or modification
  - Helpful if ransomware strikes (eg. file-server scenario with zpool NFS/SMB exported)
  - Enables the zfs send/receive feature
- Command run out of cron, multiple zpools can be snapshotted at the same time

ZFS Send/Receive

- Huge feature of ZFS
- Takes an already created snapshot and can send it to a file, or even another server running zfs
- Better than dd or other tools as the snapshot is unchanging and consistent so the file system stays online
- Great for making and sending backups of a local file system to an offsite location
  - zfs send over ssh to a machine with zfs receive that will take the data in
  - Very efficient as the only thing sent is the snapshot, so size of transfer will only be the changes

Compression

- ZFS has built in compression that occurs on the fly, behind the scenes
- Files appear normally on disk and their compressed state is invisible to the application working on the file
- There are a few different compression algorithms to chose from, the most effective one in terms of compression ratio/performance-penalty trade-off usually being LZ4 compression
  - Other compression types: LZJB, GZIP (1 through 9), ZLE (Zero Length Encoding)
  - LZ4 is the default compression when setting compression=on
  - Live compression performance can be view live on a per zpool basis

Deduplication

- ZFS supports block level deduplication
  - More useful than just file level deduplication (especially for large files)
- Stores deduplication tables in ARC so there is a large memory requirement when deduplication is enabled
  - If deduplication tables get too big they spill over into L2ARC or into the zpool disks themselves if no L2ARC is present
  - If table spills over into L2ARC there will be some performance impact, but much less of a hit than if it spilled into the actual pool itself
- Due to the high memory requirements of deduplication and the nature of HPC data, this feature is rarely useful for HPC
  - no firm ratio of disk capacity to RAM for dedup, but estimates are 1-4GB RAM/TB of usable space

Quota

- ZFS has built in quota support
- Quotas can be set for users and groups at the zpool level or at the dataset level
- Useful in situations where the ZFS file system is a standalone instance
  - eg. the zpool isn't the backend device for another file system layered on top
  - For situations where the zpool is just the backend device, quota management at the higher level is definitely best
- Quota support can be either enabled or disabled, default is disabled

Datasets within zpools

- ZFS allows for the creation of datasets which act like folders within a zpool, but can have independent qualities
  - Independent quotas, snapshots, compression, permissions
  - This allows for logical break outs of structure where it makes sense functionally
  - Don’t have to split up vdevs
- Allows for increased flexibility in managing the storage pool
  - Different groups can have different datasets with permissions and characteristics that match their needs
  - Different datasets for different file types (eg. logs are highly compressible so they could go into a compressed dataset)

ZFS Administration

Laying the Groundwork

- As stated earlier ZFS is compatible with many OS's; we'll be using CentOS 7 for the examples today
  - Many of these commands and steps are easy to translate over to debian-based OS's and there are great resources available online for additional examples
- Like many areas, few things are absolutes
  - Not everything translates to every environment
  - Hardware you have may act differently, there are too many variables to address them all
  - Don't have time to go through all scenarios, but we'll hit as many as we can give resources for the rest

Installing ZFS

- Grab the zfs repo & install gpg key

```
[root@zfs-demo ~]# wget http://download.zfsonlinux.org/epel/zfs-release.el7_3.noarch.rpm

[root@zfs-demo ~]# gpg --quiet --with-fingerprint /etc/pki/rpm-gpg/RPM-GPG-KEY-zfsonlinux
gpg: new configuration file `/root/.gnupg/gpg.conf' created
```

- Install the repo

```
[root@zfs-demo ~]# rpm -ivh zfs-release.el7_3.noarch.rpm
Preparing...                          ################################# [100%]
Updating / installing...
   1:zfs-release-1-4.el7_3.centos     ################################# [100%]
[root@zfs-demo ~]#
```

- Install zfs and kernel-devel

```
[root@zfs-demo ~]# yum install zfs kernel-devel
```

Post-Install Configuration

- Load the kernel module & Enable module on boot

```
[root@zfs-demo ~]# /sbin/modprobe zfs
[root@zfs-demo ~]# systemctl enable zfs-import-cache zfs-import-scan zfs-mount zfs-share zfs-zed zfs.target
```

- Check to make sure all is happy

```
[root@zfs-demo ~]# zpool status
no pools available
```

- Module will now load on boot, and mount your pools

Vdev Geometry Considerations

- Mirror vdev layout
  - Benefits
    - Faster vdev rebuild (no parity math)
    - Higher IOPs for the pool (great for pools backing databases, VMs)
    - Best for expandability (can add drives 2 at a time to expand pool)
  - Downsides
    - Poor space efficiency (50% space efficiency)
    - Depending on drive count less resiliency
- RAIDZ vdev layout
  - Benefits
    - Good space efficiency (80% space efficiency…or more depending on risk)
    - Good for big streaming I/O
  - Downsides
    - Lower IOPs performance
    - Expandability requires larger quantity of disk buy in
    - Slower rebuilds

Creating a zpool: Considerations

- Choose your vdev layout
- Select ashift (alignment shift) if necessary
  - Aligns vdev to block size of your media (12 for 4K block devices, 13 for 8k devices, no ashift for 512B devices*)
  - Making sure this is correct can impact performance and space overhead on disk
  - Once a vdev is created the ashift can not be changed
- Mapping of devices to vdevs
  - Look at your hardware's topology to see if you can divide up the failure domains across the vdevs to increase protection

Creating a zpool: Failure Domains

- Balance vdevs between failure domains to help withstand failure of other components
  - Server with 2 SAS attached JBODs, with mirrored stripes put one drive from each mirror in a separate JBOD
  - Have multiple SAS HBA - balance mirrors across HBA's to withstand their failure
  - Server with many JBODs (archive box) – spread raidz(2,3) vdevs across JBODs to withstand enclosure failure
- Other failure domains possible, very dependent on exact hardware configuration
  - If possible spread individual vdevs such that their redundancy protects them from more than just drive failure

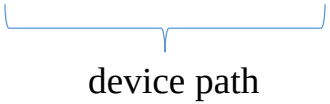
Creating a zpool: Preparation

- Identifying the disks in your system
  - Using lsscsi or fdisk

```
[root@zfs-demo ~]# fdisk -l | grep vd
Disk /dev/vda: 5368 MB, 5368709120 bytes, 10485760 sectors
Disk /dev/vdb: 5368 MB, 5368709120 bytes, 10485760 sectors
Disk /dev/vdc: 5368 MB, 5368709120 bytes, 10485760 sectors
Disk /dev/vdd: 5368 MB, 5368709120 bytes, 10485760 sectors
```

- Instead of building vdevs with device names, let's get their full path which will stay consistent
  - Multiple ways to do this, one example is below

```
[root@zfs-demo ~]# udevadm info --query=property /dev/vda | grep ID_PATH=
ID_PATH=virtio-pci-0000:00:0b.0
```



- Identify where these devices map to physically (try using dd on the device to get activity light)

Creating a zpool: vdev_id.conf

- Building a vdev_id.conf file allows us to assign aliases to disks for easier identification
  - ex. slot_0 or En_1_Bay_04
  - Comes in very handy when a drive fails or has issues
- Sample vdev_id.conf files (stored in /etc/zfs/)

```
[root@zfs-demo ~]# cat /etc/zfs/vdev_id.conf
multipath              no
topology               sas_direct
alias    slot_0  /dev/disk/by-path/virtio-pci-0000:00:0b.0
alias    slot_1  /dev/disk/by-path/virtio-pci-0000:00:0c.0
alias    slot_2  /dev/disk/by-path/virtio-pci-0000:00:0d.0
alias    slot_3  /dev/disk/by-path/virtio-pci-0000:00:0e.0
```

From a VM

or

```
multipath              no
topology               sas_direct
alias    E4_D4_B00      /dev/disk/by-path/pci-0000:83:00.0-sas-0x5b0bd6d0a104b7c8-lun-0
alias    E4_D4_B01      /dev/disk/by-path/pci-0000:83:00.0-sas-0x5b0bd6d0a104b7c9-lun-0
alias    E4_D4_B02      /dev/disk/by-path/pci-0000:83:00.0-sas-0x5b0bd6d0a104b7ca-lun-0
alias    E4_D4_B03      /dev/disk/by-path/pci-0000:83:00.0-sas-0x5b0bd6d0a104b7cb-lun-0
alias    E4_D4_B04      /dev/disk/by-path/pci-0000:83:00.0-sas-0x5b0bd6d0a104b7cc-lun-0
```

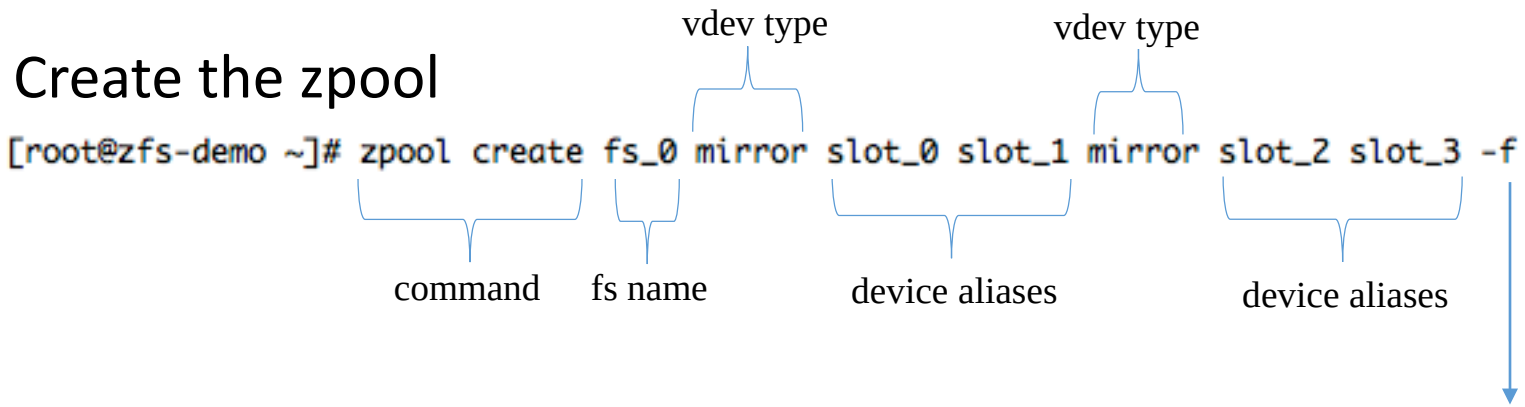
From a physical machine

Creating a zpool: Command

- After alias file is ready run udevadm trigger to put paths in place

```
[root@zfs-demo ~]# udevadm trigger
```

- Create the zpool



- ZFS attempts to detect if there is another partition on disk, the –f flag will force the creation of the pool

Checking zpool

- Command you'll run most often: zpool status

```
[root@zfs-demo ~]# zpool status
  pool: fs_0
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        fs_0        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            slot_0  ONLINE       0     0     0
            slot_1  ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            slot_2  ONLINE       0     0     0
            slot_3  ONLINE       0     0     0

errors: No known data errors
```

Getting zpool Attributes

- Zpools have a lot of tunable attributes

```
[root@zfs-demo ~]# zfs get all fs_0
NAME  PROPERTY              VALUE                  SOURCE
fs_0  type                  filesystem             -
fs_0  creation              Mon Jun 26 10:25 2017  -
fs_0  used                  56.5K                  -
fs_0  available             9.63G                  -
fs_0  referenced            19K                    -
fs_0  compressratio         1.00x                  -
fs_0  mounted               yes                    -
fs_0  quota                 none                   default
fs_0  reservation           none                   default
fs_0  recordsize            128K                   default
fs_0  mountpoint            /fs_0                  default
```

Cut off output…it's long

- Most attributes can be modified post FS create

```
[root@zfs-demo ~]# zfs set $property=$value $pool_name
```

Identifying Drive Failure

- Drive will show up as UNAVAIL and the pool is degraded

```
[root@zfs-demo fs_0]# zpool status
  pool: fs_0
 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: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        fs_0        DEGRADED     0     0     0
          mirror-0  ONLINE       0     0     0
            slot_0  ONLINE       0     0     0
            slot_1  ONLINE       0     0     0
          mirror-1  DEGRADED     0     0     0
            slot_2  UNAVAIL      0    72     0  corrupted data
            slot_3  ONLINE       0     0     0

errors: No known data errors
```

Handling Drive Failure

- Swap drive for new one and update /etc/zfs/vdev_if.conf with path changes

```
[root@zfs-demo ~]# cat /etc/zfs/vdev_id.conf
multipath               no
topology                sas_direct
alias    slot_0  /dev/disk/by-path/virtio-pci-0000:00:0b.0
alias    slot_1  /dev/disk/by-path/virtio-pci-0000:00:0c.0
alias    slot_2  /dev/disk/by-path/virtio-pci-0000:00:0f.0
alias    slot_3  /dev/disk/by-path/virtio-pci-0000:00:0e.0
```

- Run: “udevadm trigger” again; then run the replace command

```
[root@zfs-demo ~]# udevadm trigger
```

```
[root@zfs-demo fs_0]# zpool replace fs_0 slot_2 slot_2 -f
```

- The pool will begin to resilver to restore redundancy
- If pool set up with hot spare, that can be used for replace also

Scrubbing the Pool

- One line command run out of cron
  - Frequency depends on the size of pool and the desired impact on performance from scrub
  - Small disk (or really fast all SSD) pools can usually handle weekly scrubs, big ones usually monthly; scrub duration dependent on data stored
  - Ours kick off late Saturday night, but obviously set it for the lowest usage period in your environment

```
[root@zfs-demo ~]# zpool scrub fs_0
[root@zfs-demo ~]# zpool status
  pool: fs_0
 state: ONLINE
  scan: scrub in progress since Mon Jun 26 14:08:08 2017
    308M scanned out of 7.77G at 19.3M/s, 0h6m to go
    0 repaired, 3.88% done
```

Taking Snapshots

- Another ZFS task run out of cron
- Schedule with frequency desired (hourly, daily, etc.)
- Script that fires the snapshot should also handle snapshot retention
  - Snapshots will take up space as files deleted after the snapshot is taken will not truly be removed so pointers in the snapshot are still valid
  - Figure out how long snapshots should be kept around in your environment and based on the system's role

```
[root@zfs-demo ~]# zfs snapshot fs_0@20170626_1417

[root@zfs-demo ~]# zfs list -t snapshot
NAME                  USED  AVAIL  REFER  MOUNTPOINT
fs_0@20170626_1417   3.53G      -  7.77G  -
fs_0@20170626_1418       0      -  4.23G  -
```

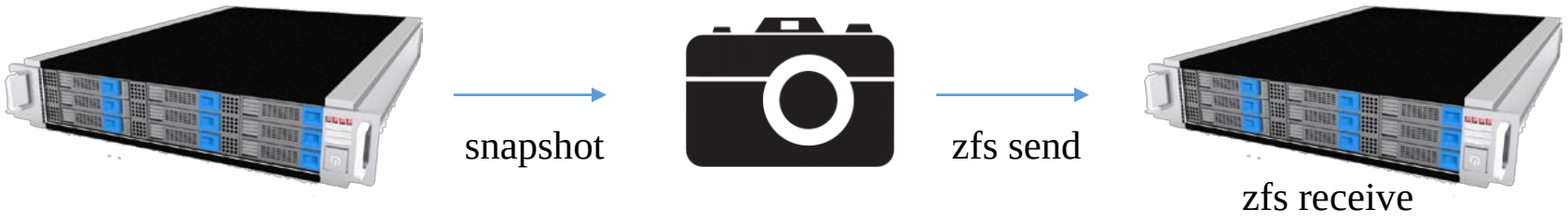
ZFS Send/Receive

- Killer feature of ZFS, live replicating systems from snapshots across WAN or LAN
- Send snapshot to image file

```
[root@zfs-demo ~]# zfs send fs_0@20170626_1418 > /diff_pool/fs_0_last_snap.img
```

- Send snapshot to another machine with zfs receive

```
[root@zfs-demo ~]# zfs send fs_0@20170626_1418 | ssh user@zfs-demo-2 "zfs receive fs_0"
```



Monitoring Health: ZED

- Built in ZFS alerts for critical events
  - Example events below, you can configure which classes get reported

```
[root@zfs-demo zed.d]# zpool events
TIME                           CLASS
Jun 26 2017 11:22:18.001000000 ereport.fs.zfs.resilver.start
Jun 26 2017 11:22:18.642000000 ereport.fs.zfs.resilver.finish
Jun 26 2017 11:22:19.302000000 ereport.fs.zfs.config.sync
Jun 26 2017 11:22:19.302000000 ereport.fs.zfs.vdev.remove
Jun 26 2017 11:22:19.835000000 ereport.fs.zfs.config.sync
Jun 26 2017 14:08:08.601000000 ereport.fs.zfs.scrub.start
Jun 26 2017 14:14:09.046000000 ereport.fs.zfs.scrub.finish
```

- Configured in /etc/zfs/zed.d/zed.rc
  - Put in your email address
  - Uncomment the ZED_NOTIFY_INTERVAL_SECS=3600 line so you don't spam yourself
  - Can also configure automatic zpool replace if you have hot spares

Monitoring Health: Scripts

- You can write your own scripts to monitor ZFS as well
- Command output is consistent and thus friendly to common shell script regex (awk, sed, grep, etc.)
- Running automatic S.M.A.R.T. tests on disks isn't a bad idea, short or long versions
- Using a utility such as the hddtemp package to monitor drive temperatures can also be handy especially if machine is not in a well cooled environment
- Lots of already written scripts and plugins for monitoring (eg. Nagios) are out there and available for download

Quick Administration Notes

- ECC Memory encouraged
  - Less likely to not have in data center environment, but strongly suggested
- Each vdev is only as fast as its slowest drive
  - Why mirrors are favored for high IOPs loads
  - Write ACK won't be sent until all disks have written out their part, which is longer with RAIDZ/RAIDZ2/RAIDZ3
- Use "zfs export" and "zfs import" when migrating pools from one machine to the other
  - Run the export command before shutdown of source machine to flush all information to disk; move drives over; run the import command; done
- Sleep well knowing your data is safe ☺

Resources

- http://zfsonlinux.org
- https://pthree.org/2012/04/17/install-zfs-on-debian-gnulinux/
- https://github.com/zfsonlinux/zfs/wiki/Mailing-Lists
- Man pages (man zpool; man zfs)

Acknowledgements

- Members of the SET group at NCSA for slide review
- Members of the steering committee for slide review

Questions