# ZFS

The Last Word in File Systems

RITLUG

ritlug.com

Josh Bicking

# A brief history of disk filesystems

- Purpose
  - Manage disk space
  - Provide a user-friendly abstraction
    - Files, folders
- Originally prioritized minimal overhead (ex: FAT, early ext*)
  - Usability limitations: Short filenames, case insensitivity, file size limits, FS size limits, no permissions
  - Worst of all: volatile!
- Eventually added new features (ext3+, NTFS)
  - journaling: write changes to a log, which will then be committed to the FS
  - ACLs: advanced permission rules for files and directories
  - Modern FSes work pretty well!

# ZFS history: A tale of legal spaghetti

- Developed by Sun Microsystems for Solaris
- *2001* - Development starts
- *2004* - ZFS announced publicly
- *2005* - OpenSolaris comes to fruition: ZFS code included, under the Common Development and Distribution License (CDDL)
- *2005-2010* - Sun continues to update ZFS, provide new features
  - And everyone wanted em. OSes developed their own ZFS implementation, or included the code
  - Linux: The CDDL and GPLv2 don't get along
    - The slow and safe solution: FUSE (filesystem in user space)
  - FreeBSD
  - Mac OS X (later discontinued, and developed as MacZFS)

# ZFS history: A tale of legal spaghetti

- *Early 2010* - Acquisition of Sun Microsystems by Oracle
- *Late 2010* - The illumos project launches
  - Shortly after, OpenSolaris is discontinued. Yikes.
  - illumos devs continue ZFS development
  - Check out [Fork Yeah! The rise and development of illumos]
- *2013* - The OpenZFS project was founded
  - All the cool kids are using ZFS now
  - Goal of coordinated open-source development of ZFS

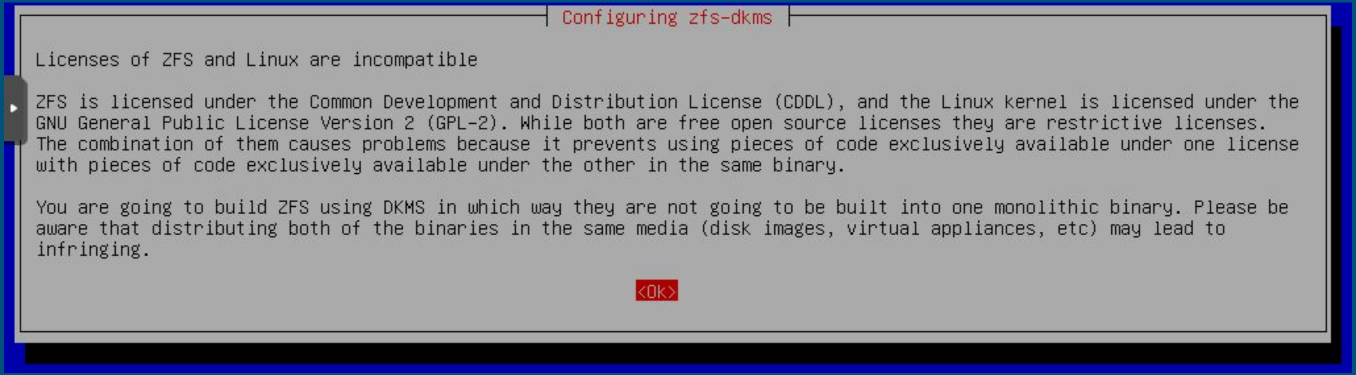
illumos

ZFS ON LINUX

OpenZFS

# ZFS on Linux, as of 2016+

- Ubuntu 16.04 bundled ZFS as a kernel module, claimed license compatibility
  - FSF was disapproving
  - Original CDDL proposal to the OSI stated *“the CDDL is not expected to be compatible with the GPL, since it contains requirements that are not in the GPL”*
- Nowadays: most think it’s fine if they’re bundled separately
  - Ex: GPL’d Linux using the CDDL’d ZFS library



```
Configuring zfs-dkms

Licenses of ZFS and Linux are incompatible

ZFS is licensed under the Common Development and Distribution License (CDDL), and the Linux kernel is licensed under the
GNU General Public License Version 2 (GPL-2). While both are free open source licenses they are restrictive licenses.
The combination of them causes problems because it prevents using pieces of code exclusively available under one license
with pieces of code exclusively available under the other in the same binary.

You are going to build ZFS using DKMS in which way they are not going to be built into one monolithic binary. Please be
aware that distributing both of the binaries in the same media (disk images, virtual appliances, etc) may lead to
infringing.

<Ok>
```

# ZFS on Linux: Installing

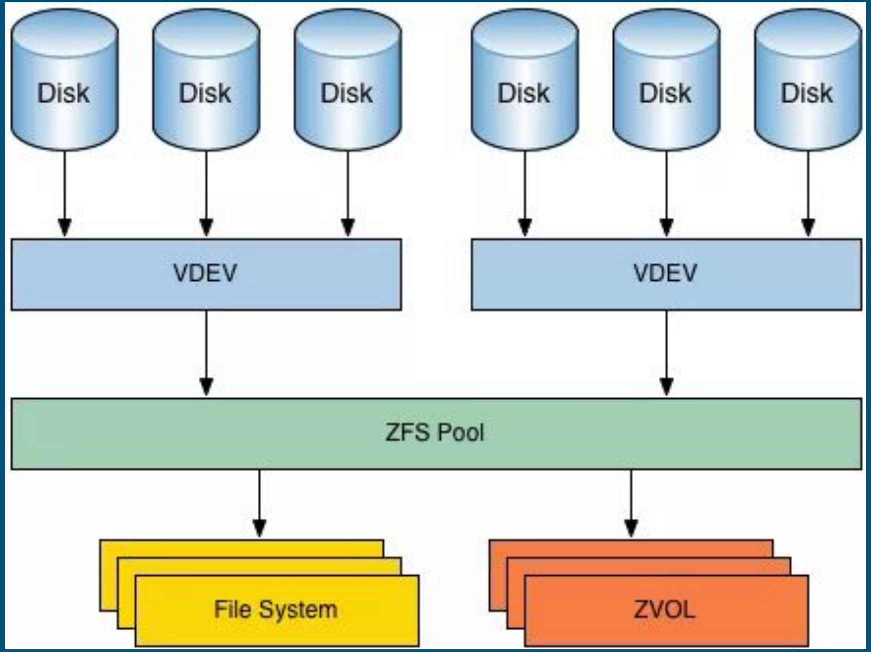
- Debian Stretch, in `contrib` (Jessie, in `backports`):
  - `apt install linux-headers-$(uname -r) zfs-dkms zfsutils-linux [zfs-initramfs]`
- Ubuntu 16.04+:
  - `apt install zfsutils-linux`
- Fedora/RHEL/CentOS:
  - Repo from http://download.zfsonlinux.org/ must be added
  - See Getting Started on the `zfsonlinux/zfs` GitHub wiki
  - RHEL/CentOS: Optional kABI-tracking kmod (no recompiling with each kernel update!)

# ZFS features: what do it do

- Not only a file system, but a volume manager too
  - Can have *complete knowledge* of both physical disks and the filesystem
- Max *16 Exabytes* file size, Max *256 Quadrillion Zettabytes* storage
- Snapshots
  - With very little overhead, thanks to a copy-on-write (COW) transactional model
- Native data deduplication (!)
- Data integrity verification and automatic repair
  - Hierarchical checksumming of all data and metadata
- Native handling of tiered storage and cache devices
- Smart caching decisions and cache use
- Native data compression
- Easy transmission of volumes, or volume changes

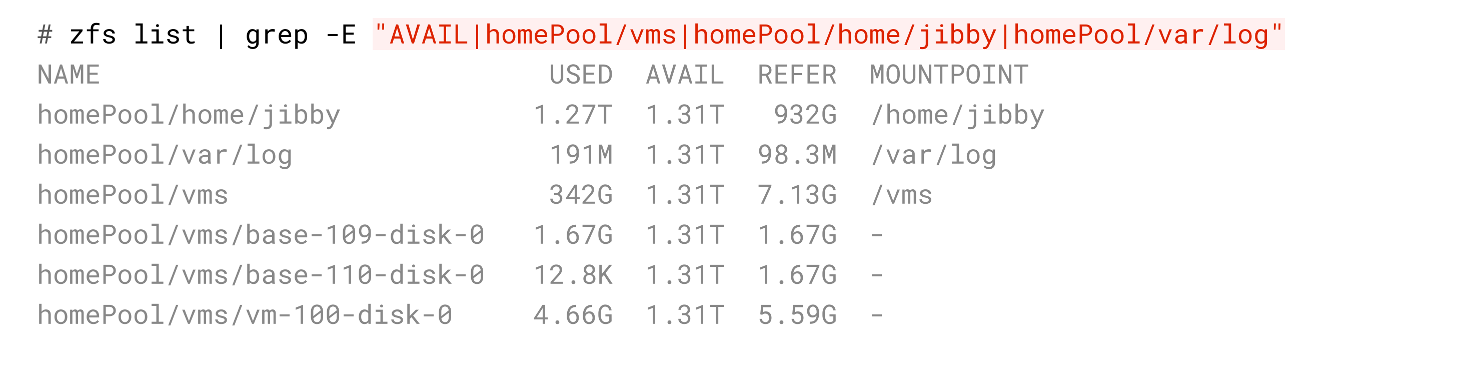
# ZFS Terminology

- *vdev*
  - Hardware RAID, physical disk, etc.
- *pool (or zpool)*
  - One or more vdevs
- *raidz, mirror,* etc.
  - ZFS controlled RAID levels
  - Some combination of vdevs
  - Specified at pool creation



# ZFS Terminology

- *dataset*
  - “Containers” for the filesystem part of ZFS
  - Mount point specified with the `mountpoint` configuration option
  - Nestable to fine-tune configuration
- *volume or (zvol)*
  - A block device stored in a pool



```
# zfs list | grep -E "AVAIL|homePool/vms|homePool/home/jibby|homePool/var/log"
NAME                            USED  AVAIL  REFER  MOUNTPOINT
homePool/home/jibby            1.27T  1.31T   932G  /home/jibby
homePool/var/log                191M  1.31T  98.3M  /var/log
homePool/vms                    342G  1.31T  7.13G  /vms
homePool/vms/base-109-disk-0   1.67G  1.31T  1.67G  -
homePool/vms/base-110-disk-0   12.8K  1.31T  1.67G  -
homePool/vms/vm-100-disk-0     4.66G  1.31T  5.59G  -
```

# ZFS Commands: starting a pool

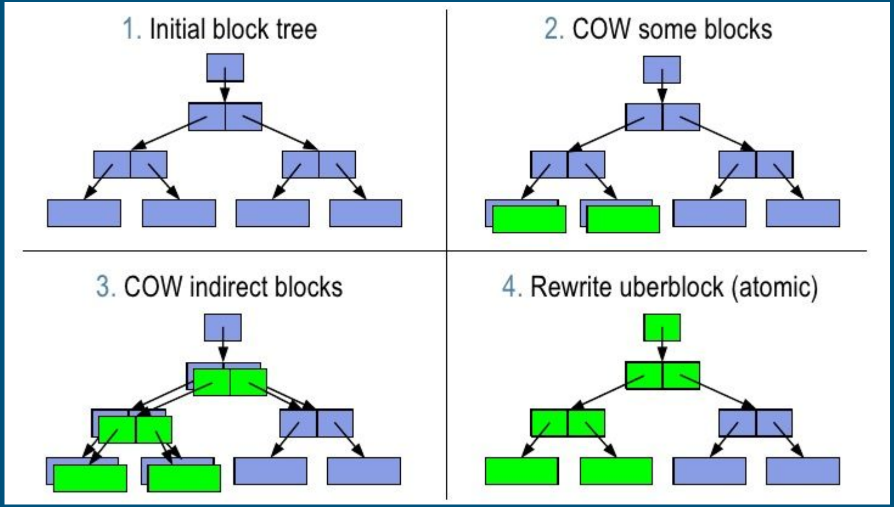
```
# zpool create myPool /dev/vdb /dev/vdc
# zpool list
NAME     SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
myPool  99.5G   111K  99.5G         -     0%     0%  1.00x  ONLINE  -
# zpool destroy myPool
# zpool create myMirrorPool mirror /dev/vdb /dev/vdc
# zpool list
NAME           SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
myMirrorPool  49.8G   678K  49.7G         -     0%     0%  1.00x  ONLINE  -
```

# ZFS Commands: datasets and volumes

```
# zfs create myMirrorPool/myDataset
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
myMirrorPool             111K  48.2G    24K  /myMirrorPool
myMirrorPool/myDataset    24K  48.2G    24K  /myMirrorPool/myDataset
# zfs create -V 3G myMirrorPool/myVol
# mkfs.ext4 /dev/zvol/myMirrorPool/myVol
# mkdir /myVol
# mount /dev/zvol/myMirrorPool/myVol /myVol/
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
myMirrorPool            3.10G  45.1G    24K  /myMirrorPool
myMirrorPool/myDataset    24K  45.1G    24K  /myMirrorPool/myDataset
myMirrorPool/myVol      3.10G  48.1G  81.8M  -
```

# Copy-on-write (COW)

- Moooove aside, journaled filesystems
- Write process
  - Write new data to a new location on disk
  - For any data that points to that data, write new data as well (indirect blocks)
  - In one (atomic) action, update the *uberblock*: the “entrypoint” to all data in the FS
  - TLDR: Either the whole write occurs, or none of it does
- Results in a “Snapshots for free” system



# Snapshots

- Simple implementation
    - What if we didn’t discard the old data during COW?
- Label that instance of data
    - Future changes are stacked on top of the snapshot
- Essentially a list of changes between then and now

```
# zfs list -t snapshot -r homePool/home | grep -E "AVAIL|monthly"
NAME                                                        USED  AVAIL  REFER  MOUNTPOINT
homePool/home@zfs-auto-snap_monthly-2019-02-01-0500        12.8K      -  72.6M  -
homePool/home@zfs-auto-snap_monthly-2019-03-01-0500           0B      -  72.6M  -
homePool/home/jibby@zfs-auto-snap_monthly-2019-02-01-0500  33.4G      -  1.27T  -
homePool/home/jibby@zfs-auto-snap_monthly-2019-03-01-0500     0B      -   932G  -
homePool/home/root@zfs-auto-snap_monthly-2019-02-01-0500    511K      -   152M  -
homePool/home/root@zfs-auto-snap_monthly-2019-03-01-0500      0B      -   152M  -
```

# ZFS commands: snapshots

```
# dd if=/dev/zero bs=1M count=1000 of=/myMirrorPool/myDataset/file
# zfs list
NAME                      USED  AVAIL  REFER  MOUNTPOINT
myMirrorPool             4.07G  44.1G    24K  /myMirrorPool
myMirrorPool/myDataset   1000M  44.1G  1000M  /myMirrorPool/myDataset
myMirrorPool/myVol       3.10G  47.1G   114M  -
# zfs snapshot myMirrorPool/myDataset@newfile
# rm /myMirrorPool/myDataset/file
# zfs list
NAME                      USED  AVAIL  REFER  MOUNTPOINT
myMirrorPool             4.07G  44.1G    24K  /myMirrorPool
myMirrorPool/myDataset   1000M  44.1G    24K  /myMirrorPool/myDataset
myMirrorPool/myVol       3.10G  47.1G   114M  -
```

# ZFS commands: snapshots

```
# zfs list -t snapshot
NAME                             USED  AVAIL  REFER  MOUNTPOINT
myMirrorPool/myDataset@newfile  1000M      -  1000M  -
# zfs snapshot myMirrorPool/myDataset@deletedfile
# zfs list -t snapshot
NAME                                 USED  AVAIL  REFER  MOUNTPOINT
myMirrorPool/myDataset@newfile      1000M      -  1000M  -
myMirrorPool/myDataset@deletedfile     0B      -    24K  -
# zfs destroy myMirrorPool/myDataset@deletedfile
# ls /myMirrorPool/myDataset/
# zfs rollback -r myMirrorPool/myDataset@newfile
# ls /myMirrorPool/myDataset/
file
```

# ZFS pitfalls

- Moderately steep learning curve
  - Not really a “set and forget” FS, more of “set, configure, and monitor performance”
- If configured wrong, performance can suffer
  - And there’s *a lot* to be configured
- More overhead than your average FS
  - While snapshots are nice, might not be worth running on your daily driver
- No good, long term solution to fragmentation
  - Leading idea is *block pointer rewrite*, which an OpenZFS member described as *“like changing your pants while you're running”*

# Demo time!

- Make a pool
- Make a volume
- Look at configurables
- Play around with compression
- Try out snapshots

```
Demo commands, for future generations to follow along:

fdisk -l
zpool create tank -f /dev/vdb /dev/vdc
fdisk -l
zfs create -o compression=off -o mountpoint=/dataset1
tank/dataset1
cd /dataset1
zfs list
dd if=/dev/zero bs=1M count=2000 | pv | dd
of=/dataset1/outfile bs=1M
ls -lh outfile
zfs get all tank/dataset1

rm outfile
zfs set compression=zle tank/dataset1
dd if=/dev/zero bs=1M count=2000 | pv | dd
of=/dataset1/outfile bs=1M
ls -lh outfile
zfs get all tank/dataset1

zfs snapshot tank/dataset1@add_outfile
zfs list -t snapshot
cd .zfs
tree
cd snapshot/add_outfile/
ls -lh outfile
cd /dataset1
rm outfile
tree .zfs
ls
zfs list -t snapshot
zfs create -V 10G tank/vol1
```

# More Info & References

Are the GPLv2 and CDDL incompatible?

Sun's Common Development and Distribution License Request to the OSI

zfsonlinux/zfs GitHub wiki: Getting Started

Github issue: ZFS Fragmentation: Long-term Solutions

Fork Yeah! The rise and development of illumos

OpenZFS User Documentation

Aaron Toponce's "Getting Started with ZFS" Guide