LINUX™ JOURNAL

ZFS for Linux

Feb 12, 2018 By Charles Fisher (/users/charles-fisher)

in

G+

Presenting the Solaris ZFS filesystem, as implemented in Linux FUSE, native kernel modules and the Antergos Linux installer.

ZFS remains one of the most technically advanced and feature-complete filesystems since it appeared in October 2005. Code for **Sun's original Zettabyte File System** (http://web.archive.org/web/20060428092023/http://www.sun.com/2004-0914/feature) was released under the CDDL open-source license, and it has since become a standard component of FreeBSD and slowly migrated to various BSD brethren, while maintaining a strong hold over the descendants of OpenSolaris, including OpenIndiana and SmartOS.

Oracle is the owner and custodian of ZFS, and it's in a peculiar position with respect to Linux filesystems. **Btrfs** (https://docs.oracle.com/cd/E37670_01/E37355/html/ol_btrfs.html) , the main challenger to ZFS, began development at Oracle, where it is a core component of Oracle Linux, despite **stability issues** (https://www.suse.com/communities/blog/butter-bei-die-fische) Red Hat's recent decision to **deprecate Btrfs** (https://www.theregister.co.uk/2017/08/16/red_hat_banishes_btrfs_from_rhel) likely introduces compatibility and support challenges for Oracle's Linux road map. Oracle obviously has deep familiarity with the Linux filesystem landscape, having recently released **"dedup" patches for XFS** (https://blogs.oracle.com/linuxkernel/upcoming-xfs-work-in-linux-v48-v49-and-v410%2c-by-darrick-wong) . ZFS is the only filesystem option that is stable, protects your data, is proven to survive in most hostile environments and has a lengthy usage history with well understood strengths and weaknesses.

ZFS has been (mostly) kept out of Linux due to **CDDL incompatibility** (https://sfconservancy.org/blog/2016/feb/25/zfs-and-linux) with Linux's GPL license. It is the clear hope of the Linux community that Oracle will re-license ZFS in a form that can be included in Linux, and we should all gently cajole Oracle to do so. Obviously, a re-license of ZFS will have a clear impact on Btrfs and the rest of Linux, and we should work to understand Oracle's position as the holder of these tools. However, Oracle continues to gift large software projects for independent leadership. Incomplete examples of Oracle's largesse include

OpenOffice (http://www.zdnet.com/article/oracle-gives-openoffice-to-apache) and recently **Java Enterprise Edition** (https://adtmag.com/articles/2017/09/12/java-ee-moving-to-eclipse.aspx), so it is not inconceivable that Oracle's generosity may at some point extend additionally to ZFS.

To further this conversation, I want to investigate the various versions of ZFS for Linux. Starting within an RPM-centric environment, I first describe how to install the minimally invasive FUSE implementation, then proceed with a native install of ZFS modules from source. Finally, leaving RPM behind, I proceed to the Antergos distribution that implements native ZFS as a supported installation option.

ZFS Technical Background

ZFS is similar to other storage management approaches, but in some ways, it's radically different. ZFS does not normally use the Linux Logical Volume Manager (LVM) or disk partitions, and it's usually convenient to delete partitions and LVM structures prior to preparing media for a zpool.

The zpool is the analog of the LVM. A zpool spans one or more storage devices, and members of a zpool may be of several various types. The basic storage elements are single devices, mirrors and raidz. All of these storage elements are called vdevs.

Mirrored vdevs in a zpool present storage that's the size of the smallest physical drive. A mirrored vdev can be upgraded (that is, increased in size) by **attaching** (http://docs.oracle.com/cd/E19253-01/819-5461/gcfhe/index.html) larger drives to the mirrorset and "resilvering" (synchronizing the mirrors), then detaching the smaller drives from the set Resilvering a mirror will involve copying only used blocks to the target device—unused blocks are not touched, which can make resilvering much faster than hardware-maintained disk mirroring (which copies unused storage).

ZFS also can maintain RAID devices, and unlike most storage controllers, it can do so without battery-backed cache (as long as the physical drives honor "write barriers"). ZFS can create a raidz vdev with multiple levels of redundancy, allowing the failure of up to three physical drives while maintaining array availability. Resilvering a raidz also involves only used blocks and can be much faster than a storage controller that copies all disk blocks during a RAID rebuild. A raidz vdev should normally compose 8–12 drives (larger raidz vdevs are not recommended). Note that the **number of drives in a raidz cannot be expanded** (http://louwrentius.com/the-hidden-cost-of-using-zfs-for-your-home-nas.html) .

ZFS greatly prefers to manage raw disks. RAID controllers should be configured to present the raw devices, never a hardware RAID array. ZFS is able to enforce storage integrity far better than any RAID controller, as it has intimate knowledge of the structure of the filesystem. All controllers should be configured to present "Just a Bunch Of Disks" (JBOD) for best results in ZFS.

Data safety is an important design feature of ZFS. All blocks written in a zpool are aggressively checksummed to ensure the data's consistency and correctness. You can select the checksum algorithm from sha256, fletcher2 or fletcher4. You also can disable the checksum on user data, which is *specifically never recommended* (this setting might be useful on a scratch/tmp filesystem where speed is critical, while consistency and recovery are irrelevant; however, `sync=disabled` is the **recommended setting** (https://wiki.archlinux.org/index.php/ZFS) for temporary filesystems in ZFS.

You can change the checksum algorithm at any time, and new blocks will use the updated algorithm. A checksum is stored separately from the data block, with the parent block, in the hope that localized block damage can be detected. If a block is found to disagree with the parent's checksum, an alternate copy of the block is retrieved from either a mirror or raidz device, rewritten over the bad block, then the I/O is completed without incident. ZFS filesystems can use these techniques to "self-heal" and protect themselves from "bitrot" data changes on hard drive platters that are caused by controller errors, power loss/fluctuations in the read/write heads, and even the bombardment of cosmic rays.

ZFS can implement "deduplication" by maintaining a searchable index of block checksums and their locations. If a new block to be written matches an existing block within the index, the existing block is used instead, and space is saved. In this way, multiple files may share content by maintaining single copies of common blocks, from which they will diverge if any of their content changes. The documentation states that a "dedup-capable checksum" must be set before dedup can be enabled, and sha256 is offered as an example—the checksum must be "collision-resistant" to identify a block uniquely to assure the safety of dedup. Be warned that memory requirements for ZFS expand drastically when deduplication is enabled, which quickly can overwhelm a system lacking sufficient resources.

The zpool can hold datasets, snapshots, clones and volumes. A "dataset" is a standard ZFS filesystem that has a mountpoint and can be modified. A "snapshot" is a point-in-time copy of a filesystem, and as the parent dataset is changed, the snapshot will collect the original blocks to maintain a consistent past image. A "clone" can be built upon a snapshot and allows a different set of changes to be applied to the past image, effectively allowing a filesystem to branch—the clone and original dataset will continue to share unchanged blocks, but otherwise will diverge. A "volume" is similar to a block device, and can be loopback-mounted with a filesystem of any type, or perhaps presented as an iscsi target. Checksums are enforced on volumes. Note that, unlike partitions or logical volumes, elements in a zpool can be intermingled. ZFS knows that the outside edge of a disk is faster than the interior, and it may decide to mix blocks from multiple objects in a zpool at these locations to increase performance. Due to this commingling of filesystems, **forensic analysis of zpools** (https://forums.freenas.org/index.php?threads/ecc-vs-non-ecc-ram-and-zfs.15449) is difficult and expensive:

> ***But, no matter how much searching you do, there is [sic] no ZFS recovery tools out there. You are welcome to call companies like Ontrack for data recovery. I know one person that did, and they spent $3k just to find out if their data was recoverable. Then they spent another $15k to get just 200GB of data back.***

There are no fsck or defrag tools for ZFS datasets. The boot process never will be delayed because a dataset was not cleanly unmounted. There is a "scrub" tool that will walk a dataset and verify the checksum of every used block on all vdevs, but the scrub takes place on mounted and active datasets. ZFS can recover very well from power losses or otherwise dirty dismounts.

Fragmentation in ZFS is a larger question, and it appears related more to remaining storage capacity than rapid file growth and reduction. Performance of a heavily used dataset will begin to degrade when it is 50% full, and it will dramatically drop over 80% usage when ZFS begins to use "best-fit" rather than "first-fit" to store new blocks. Regaining performance after dropping below 50% usage can involve dropping and resilvering physical disks in the containing vdev until all of the dataset's blocks have migrated. Otherwise, the dataset should be completely unloaded and erased, then reloaded with content that does not exceed 50% usage (the **zfs send and receive utilities** (https://docs.oracle.com/cd/E18752_01/html/819-5461/gbchx.html) are useful for this purpose). It is important to provide ample free disk space to datasets that will see heavy use.

It is strongly encouraged to use ECC memory with ZFS. Error-correcting memory is advised as critical for the correct processing of checksums that maintain zpool consistency. Memory can be altered by system errors and cosmic rays—ECC memory can correct single-bit errors, and panic/halt the system when multi-bit errors are detected. ECC memory is normally found in servers, but becomes somewhat rare with desktops and laptops. Some warn of the **"scrub of death"** (http://jrs-s.net/2015/02/03/will-zfs-and-non-ecc-ram-kill-your-data) and describe actual lost data from non-ECC RAM. However, one of the creators of ZFS **says that all filesystems are vulnerable** (https://arstechnica.com/civis/viewtopic.php?f=2&t=1235679&p=26303271#p26303271) when non-ECC memory is in use, and ZFS is actually more graceful in failure than most, and further describes undocumented settings that force ZFS to recompute checksums in memory repeatedly, which minimizes dangers from non-ECC RAM. A lengthy configuration guide addresses ZFS safety in a non-ECC environment with these undocumented settings (**https://www.csparks.com/ZFS%20Without%20Tears.html** (https://www.csparks.com/ZFS%20Without%20Tears.html)), but the guide does not appear to cover the FUSE implementation.

zfs-fuse

The Linux implementation of **FUSE** (https://github.com/libfuse/libfuse) received a ZFS port in 2006. FUSE is an interface that allows a filesystem to be implemented by a process that runs in userspace. Fedora has maintained zfs-fuse as an RPM package for some time, but this package does not appear in any of the Red Hat-based distributions, including Oracle Linux. Red Hat appears to have intentionally omitted any relevant RPM for ZFS support.

The FUSE implementation is likely the only way to (currently) use ZFS on Linux in a manner that is fully compliant with both the CDDL and the GPL.

The FUSE port is relatively slow compared to a kernel ZFS implementation. FUSE is not generally installed in a manner that is compatible with NFS, so a zfs-fuse filesystem cannot be exported over the network without preparing a FUSE version with NFS support (NFSv4 might be available if an fsid= is supplied). The zfs-fuse implementation is likely reasonable for local, archival and potentially compressed datasets. Some have **used Btrfs for ad-hoc compressed filesystems** (http://www.excamera.com/sphinx/article-btrfs.html) , and zfs-fuse is certainly an option for similar activity.

The last version of zfs-fuse that will work in Oracle Linux 7.4 is the RPM in **Fedora 25** (https://mirrors.lug.mtu.edu/fedora/linux/releases/25/Everything/x86_64/os/Packages/z) . A new ZFS release is in Fedora 26, but it fails to install on Oracle Linux 7.4 due to an OpenSSL dependency—Red Hat's

OpenSSL is now too old. The following shows installing the ZFS RPM:

```
# rpm -Uvh zfs-fuse-0.7.0-23.fc24.x86_64.rpm
Preparing...                     #################
Updating / installing...
   1:zfs-fuse-0.7.0-23.fc24  #################

# cat /etc/redhat-release /etc/oracle-release
Red Hat Enterprise Linux Server release 7.4 (M
Oracle Linux Server release 7.4
```

The zfs-fuse userspace agent must be executed before any zpools can be manipulated (note a systemd unit is included for this purpose):

```
# zfs-fuse
#
```

For an easy example, let's re-task a small hard drive containing a Windows 7 installation:

```
# fdisk -l /dev/sdb

Disk /dev/sdb: 160.0 GB, 160000000000 bytes, 3
Disk label type: dos
Disk identifier: 0x8d206763

   Device Boot      Start         End      Blo
/dev/sdb1   *        2048      206847      102
/dev/sdb2          206848   312496127   156144
```

It is usually most convenient to dedicate an entire disk to a zpool, so delete all the existing partitions:

```
# fdisk /dev/sdb
Welcome to fdisk (util-linux 2.23.2).
```

```
Changes will remain in memory only, until you
Be careful before using the write command.


Command (m for help): d
Partition number (1,2, default 2): 2
Partition 2 is deleted

Command (m for help): d
Selected partition 1
Partition 1 is deleted

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

Now a zpool can be added on the drive (note that creating a pool adds a dataset of the same name, which, as you see here, is automatically mounted):

```
# zpool create vault /dev/sdb

# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks     Used Availab
vault               153796557       21 1537965

# mount | grep vault
vault on /vault type fuse.zfs
```

Creating a zpool on non-redundant devices is informally known as **"hating your data"** (https://arstechnica.com/information-technology/2014/02/ars-walkthrough-using-the-zfs-next-gen-filesystem-on-linux) and should be contemplated only for demonstration purposes. However, **zpools on non-redundant** (http://docs.oracle.com/cd/E19253-01/819-5461/gevpg/index.html) media (for example, flash drives) have obvious data-consistency and compression advantages to VFAT, and the `copies` parameter can be adjusted for such a dataset to force all blocks to be recorded on the media multiple times (up to three) to increase recoverability.

Mirrored drives can be created with `zpool create vault mirror /dev/sdb /dev/sdc`. Additional drives can be added as mirrors to an existing drive with `zpool attach`. A simple RAIDset can be created with `zpool create vault raidz /dev/sdb /dev/sdc /dev/sdd`.

The standard `umount` command should (normally) not be used to unmount ZFS datasets—use the zpool/zfs tools instead (note the "unmount" rather than "umount" spelling):

```
# zfs unmount vault

# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks     Used Availab

# zfs mount vault
# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks     Used Availab
vault               153796557       21 1537965
```

A ZFS dataset can be mounted in a new location by altering the "mountpoint":

```
# zfs unmount vault

# mkdir /root/vault

# zfs set mountpoint=/root/vault vault

# zfs mount vault

# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks     Used Availab
vault               153796547       21 1537965
```

```
# zfs unmount vault

# zfs set mountpoint=/vault vault

# zfs mount vault

# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks      Used Availab
vault               153796547        21 1537965
```

The mountpoint is retained and is persistent across reboots.

Creating an additional dataset (and mounting it) is as easy as creating a directory (note this command can take some time):

```
# zfs create vault/tmpdir

# df | awk 'NR==1||/(vault|tmpdir)/'
```

```
Filesystem          1K-blocks     Used Availab
vault               153796496      800 1537956
vault/tmpdir        153795717       21 1537956

# cp /etc/yum.conf /vault/tmpdir/

# ls -l /vault/tmpdir/
-rw-r--r--. 1 root root 813 Sep 23 16:47 yum.c
```

ZFS supports several types of compression in a dataset. Gzip of varying degrees, zle and lzjb can all be present in a single mountpoint. The checksum algorithm also can be adjusted on the fly:

```
# zfs get compress vault/tmpdir
NAME          PROPERTY     VALUE      SOURCE
vault/tmpdir  compression  off        local

# zfs get checksum vault/tmpdir
```

```
NAME          PROPERTY  VALUE      SOURCE
vault/tmpdir  checksum  on         default

# zfs set compression=gzip vault/tmpdir

# zfs set checksum=fletcher2 vault/tmpdir

# cp /etc/redhat-release /vault/tmpdir

# zfs set compression=zle vault/tmpdir

# zfs set checksum=fletcher4 vault/tmpdir

# cp /etc/oracle-release /vault/tmpdir

# zfs set compression=lzjb vault/tmpdir

# zfs set checksum=sha256 vault/tmpdir

# cp /etc/os-release /vault/tmpdir
```

Note that the GZIP compression factor can be adjusted (the default is six, just as in the GNU GZIP utility). This will directly impact the speed and responsiveness of a dataset:

```
# zfs set compression=gzip-1 vault/tmpdir

# cp /etc/profile /vault/tmpdir

# zfs set compression=gzip-9 vault/tmpdir

# cp /etc/grub2.cfg /vault/tmpdir

# ls -l /vault/tmpdir
-rw-r--r--. 1 root root 6308 Sep 23 17:06 grub
-rw-r--r--. 1 root root   32 Sep 23 17:00 orac
-rw-r--r--. 1 root root  398 Sep 23 17:00 os-r
-rw-r--r--. 1 root root 1795 Sep 23 17:05 prof
-rw-r--r--. 1 root root   52 Sep 23 16:59 redh
-rw-r--r--. 1 root root  813 Sep 23 16:58 yum.
```

Should the dataset no longer be needed, it can be dropped:

```
# zfs destroy vault/tmpdir

# df | awk 'NR==1||/(vault|tmpdir)/'
Filesystem          1K-blocks     Used Availab
vault               153796523      800 1537957
```

You can demonstrate a recovery in ZFS by copying a few files and creating a snapshot:

```
# cp /etc/passwd /etc/group /etc/shadow /vault

# ls -l /vault
-rw-r--r--. 1 root root  965 Sep 23 14:41 grou
-rw-r--r--. 1 root root 2269 Sep 23 14:41 pass
----------. 1 root root 1255 Sep 23 14:41 shad
```

```
# zfs snapshot vault@goodver

# zfs list -t snapshot
NAME                USED  AVAIL  REFER  MOUNTPOINT
vault@goodver         0      -     27K  -
```

Then you can simulate more file manipulations that involve the loss of a critical file:

```
# rm /vault/shadow
rm: remove regular file '/vault/shadow'? y

# cp /etc/resolv.conf /etc/nsswitch.conf /etc/

# ls -l /vault
-rw-r--r--. 1 root root    965 Sep 23 14:41 gr
-rw-r--r--. 1 root root   1760 Sep 23 16:14 ns
-rw-r--r--. 1 root root   2269 Sep 23 14:41 pa
-rw-r--r--. 1 root root     98 Sep 23 16:14 re
```

```
-rw-r--r--. 1 root root 670311 Sep 23 16:14 se
```

Normally, snapshots are visible in the .zfs directory of the dataset. However, this functionality does not exist within the zfs-fuse implementation, so you are forced to create a clone to retrieve your lost file:

```
# zfs clone vault@goodver vault/history

# ls -l /vault/history
-rw-r--r--. 1 root root  965 Sep 23 14:41 grou
-rw-r--r--. 1 root root 2269 Sep 23 14:41 pass
----------. 1 root root 1255 Sep 23 14:41 shad
```

Note that the clone is not read-only, and you can modify it. The two mountpoints will maintain a common set of blocks, but are otherwise independent:

```
# cp /etc/fstab /vault/history

# ls -l /vault/history
-rw-r--r--. 1 root root  541 Sep 23 16:23 fsta
-rw-r--r--. 1 root root  965 Sep 23 14:41 grou
-rw-r--r--. 1 root root 2269 Sep 23 14:41 pass
----------. 1 root root 1255 Sep 23 14:41 shad
```

Assuming that you have completed your recovery activity, you can destroy the clone and snapshot. A scrub of the parent dataset to verify its integrity at that point might be wise, and then you can list your zpool history to see evidence of your session:

```
# zfs destroy vault/history

# zfs destroy vault@goodver

# zpool scrub vault
# zpool status vault
  pool: vault
 state: ONLINE
 scrub: scrub in progress for 0h1m, 30.93% don
config:

       NAME        STATE     READ WRITE CKSUM
       vault       ONLINE       0     0     0
         sdb       ONLINE       0     0     0

errors: No known data errors

# zpool history vault
```

For my final words on zfs-fuse, I'm going to list the software version history for zpool and zfs. Note: it is critical that you create your zpools with the lowest ZFS version that you wish to use, which in this case is zpool version 23 and zfs version 4:

```
# zpool upgrade -v
This system is currently running ZFS pool vers

The following versions are supported:

VER  DESCRIPTION
---  -----------------------------------------
 1   Initial ZFS version
 2   Ditto blocks (replicated metadata)
 3   Hot spares and double parity RAID-Z
 4   zpool history
 5   Compression using the gzip algorithm
 6   bootfs pool property
 7   Separate intent log devices
 8   Delegated administration
 9   refquota and refreservation properties
 10  Cache devices
 11  Improved scrub performance
 12  Snapshot properties
 13  snapused property
 14  passthrough-x aclinherit
 15  user/group space accounting
 16  stmf property support
 17  Triple-parity RAID-Z
 18  Snapshot user holds
 19  Log device removal
 20  Compression using zle (zero-length encodi
 21  Deduplication
 22  Received properties
 23  Slim ZIL


# zfs upgrade -v
The following filesystem versions are supporte

VER  DESCRIPTION
---  ----------------------------------------
 1   Initial ZFS filesystem version
 2   Enhanced directory entries
 3   Case insensitive and File system unique i
 4   userquota, groupquota properties
```

Native ZFS

You can obtain a zfs.ko kernel module from the **ZFS on Linux site** (http://zfsonlinux.org) and load into Linux, which will provide high-performance ZFS with full functionality. In order to install this package, you must remove the FUSE version of ZFS (assuming it was installed as in the previous section):

```
# rpm -e zfs-fuse
Removing files since we removed the last packa
```

After the FUSE removal, you need to install a new yum repository on the target system. ZFS on a Red Hat-derivative likely will require network access to the ZFS repository (standalone installations will be more difficult and are not covered here):

```
# yum install \
   http://download.zfsonlinux.org/epel/zfs-rel
...
==============================================
 Package                          Repository
==============================================
Installing:
 zfs-release                      /zfs-release

==============================================
Install  1 Package

Total size: 2.9 k
Installed size: 2.9 k
Is this ok [y/d/N]: y
...

Installed:
  zfs-release.noarch 0:1-5.el7_4

Complete!
```

After configuring the repository, load the GPG key:

```
# gpg --quiet --with-fingerprint /etc/pki/rpm-
pub  2048R/F14AB620 2013-03-21 ZFS on Linux
 Key fingerprint = C93A FFFD 9F3F 7B03 C310  C
sub  2048R/99685629 2013-03-21
```

At this point, you're are ready to proceed with a native ZFS installation.

The test system used here, Oracle Linux 7.4, normally can boot from one of two kernels. There is a "Red Hat-Compatible Kernel" and also an "Unbreakable Enterprise Kernel" (UEK). Although the FUSE version is completely functional under both kernels, the native ZFS installer does not work with the UEK (meaning further that Oracle Ksplice is precluded with the standard ZFS installation). If you are running Oracle Linux, you must be booted on the RHCK when manipulating a native ZFS configuration, and this includes the initial install. Do not attempt installation or any other native ZFS activity while running the UEK:

```
# rpm -qa | grep ^kernel | sort
kernel-3.10.0-693.2.2
kernel-devel-3.10.0-693.2.2
kernel-headers-3.10.0-693.2.2
kernel-tools-3.10.0-693.2.2
kernel-tools-libs-3.10.0-693.2.2
kernel-uek-4.1.12-103.3.8.1
kernel-uek-firmware-4.1.12-103.3.8.1
```

The ZFS installation actually uses yum to compile C source code in the default configuration (DKMS), then prepares an initrd with `dracut` (use `top` to monitor this during the install). This installation will take some time, and there are notes on using a pre-compiled zfs.ko collection in an alternate installation configuration (kABI). The test platform used here is Oracle Linux, and the Red Hat-Compatible Kernel may not be fully interoperable with the precompiled zfs.ko collection (not tested while preparing this article), so the default DKMS build was retained. Here's an example installation session:

```
# yum install kernel-devel zfs
...
==============================================
 Package
==============================================
Installing:
 zfs
Installing for dependencies:
 dkms
 libnvpair1
 libuutil1
 libzfs2
 libzpool2
 spl
 spl-dkms
 zfs-dkms

==============================================
Install  1 Package (+8 Dependent packages)


Total download size: 6.6 M
Installed size: 29 M
Is this ok [y/d/N]: y
...
   - Installing to /lib/modules/3.10.0-693.2.2
spl:
splat.ko:
zavl:
znvpair.ko:
zunicode.ko:
zcommon.ko:
zfs.ko:
zpios.ko:
icp.ko:

Installed:
  zfs.x86_64 0:0.7.1-1.el7_4

Complete!
```

After the yum session concludes, you can load the native zfs.ko into the "RHCK" Linux kernel, which will pull in a number of dependent modules:

```
# modprobe zfs

# lsmod | awk 'NR==1||/zfs/'
Module                  Size  Used by
zfs                  3517672  0
zunicode              331170  1 zfs
zavl                   15236  1 zfs
icp                   266091  1 zfs
zcommon                73440  1 zfs
znvpair                93227  2 zfs,zcommon
spl                   102592  4 icp,zfs,zcommo
```

At this point, the pool created by FUSE can be imported back into the system (note the error):

```
# /sbin/zpool import vault
cannot import 'vault': pool was previously in
Last accessed at Sun Sep 24 2017
The pool can be imported, use 'zpool import -f

# /sbin/zpool import vault -f
```

The import will mount the dataset automatically:

```
# ls -l /vault
-rw-r--r--. 1 root root    965 Sep 23 14:41 gr
-rw-r--r--. 1 root root   1760 Sep 23 16:14 ns
-rw-r--r--. 1 root root   2269 Sep 23 14:41 pa
-rw-r--r--. 1 root root     98 Sep 23 16:14 re
-rw-r--r--. 1 root root 670311 Sep 23 16:14 se
```

You can create a snapshot, then delete another critical file:

```
# /sbin/zfs snapshot vault@goodver

# rm /vault/group
rm: remove regular file '/vault/group'? y
```

At this point, you can search the /vault/.zfs directory for the missing file (note that `.zfs` does not appear with `ls -a`, but it is present nonetheless):

```
# ls -la /vault
drwxr-xr-x.  2 root root      6 Sep 25 17:47 .
dr-xr-xr-x. 19 root root   4096 Sep 25 17:17 .
-rw-r--r--.  1 root root   1760 Sep 23 16:14 n
-rw-r--r--.  1 root root   2269 Sep 23 14:41 p
-rw-r--r--.  1 root root     98 Sep 23 16:14 r
-rw-r--r--.  1 root root 670311 Sep 23 16:14 s
```

```
# ls -l /vault/.zfs
dr-xr-xr-x. 2 root root 2 Sep 23 13:54 shares
drwxrwxrwx. 2 root root 2 Sep 25 17:47 snapsho

# ls -l /vault/.zfs/snapshot/
drwxr-xr-x. 2 root root 7 Sep 24 18:58 goodver

# ls -l /vault/.zfs/snapshot/goodver
-rw-r--r--. 1 root root    965 Sep 23 14:41 gr
-rw-r--r--. 1 root root   1760 Sep 23 16:14 ns
-rw-r--r--. 1 root root   2269 Sep 23 14:41 pa
-rw-r--r--. 1 root root     98 Sep 23 16:14 re
-rw-r--r--. 1 root root 670311 Sep 23 16:14 se
```

Native ZFS implements newer software versions of zpool and zfs—remember, it is critical that you create your zpools with the lowest ZFS version that you ever intend to use, which in this case is zpool version 28, and zfs version 5. The FUSE version is far simpler to install on a fresh Red Hat

OS for recovery purposes, so consider carefully before upgrading to the native ZFS versions:

```
# /sbin/zpool upgrade -v
...

 23  Slim ZIL
 24  System attributes
 25  Improved scrub stats
 26  Improved snapshot deletion performance
 27  Improved snapshot creation performance
 28  Multiple vdev replacements


# /sbin/zfs upgrade -v
...

 4   userquota, groupquota properties
 5   System attributes
```

Strong words of warning should accompany the use of native ZFS on a Red Hat-derivative.

Kernel upgrades are a cause for concern. If the zfs.ko family of modules are not installed correctly, then no pools can be brought online. For this reason, it is far more imperative to retain known working kernels when upgraded kernels are installed. As I've noted previously, Oracle's UEK is not ZFS-capable when using the default native installation.

OS release upgrades also introduce even more rigorous warnings. Before attempting an upgrade, remove all of the ZFS software. Upon upgrade completion, repeat the ZFS software installation using a yum repository that is specific for the new OS release. The ZFS on Linux site currently lists repositories for Red Hat releases 6, 7.3 and 7.4. It is wise to stay current on patches and releases, and strongly consider upgrading a 7.0 – 7.2 Red Hat-derivative where native ZFS installation is contemplated or desired.

Note also that Solaris ZFS has encryption and Windows SMB capability—these are not functional in the Linux port.

Perhaps someday Oracle will permit the Red Hat family to bundle native ZFS by relaxing the license terms. That will be a very good day.

Antergos

Definite legal ambiguity remains with ZFS. Although Ubuntu recently announced support for the zfs.ko module for its container subsystem, its **legal analysis remains murky** (https://insights.ubuntu.com/2016/02/18/zfs-licensing-and-linux) . Unsurprisingly, none of the major enterprise Linux distributions have been willing to bundle ZFS as a first-class supported filesystem.

Into this void comes Antergos, a descendant of Arch Linux. The Antergos installer will download and compile ZFS source code into the installation kernel in a manner similar to the previous section.

Although the example installation detailed here did not proceed without incident, it did leave a working, mirrored zpool for the root filesystem running the same version release as the native RPM installs.

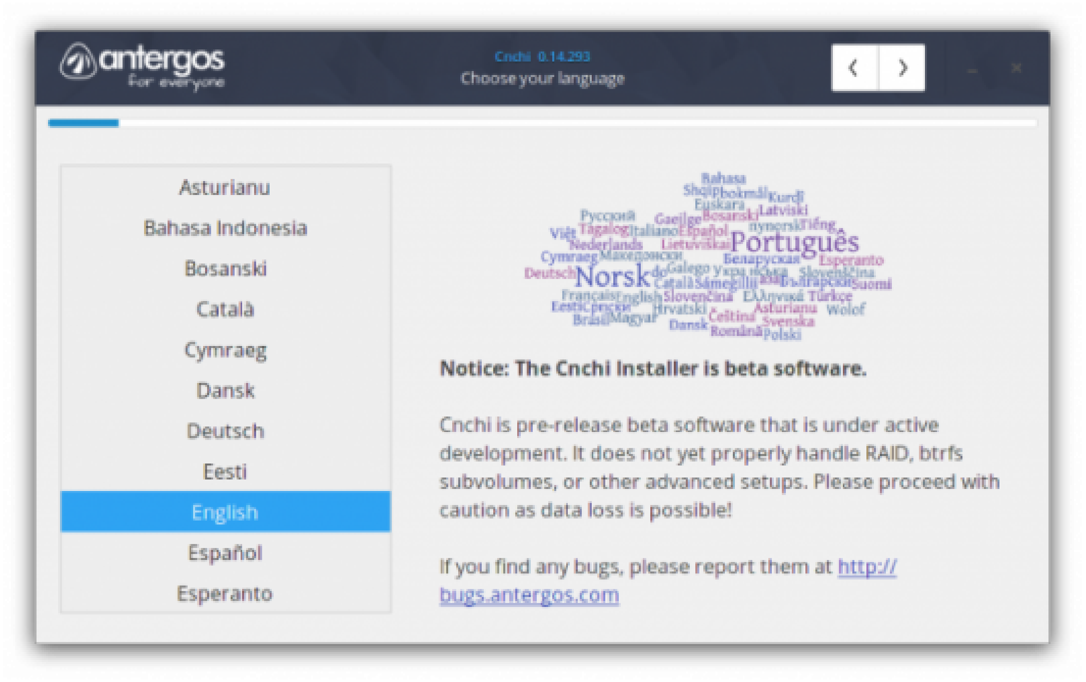
What Antergos did not do was install the Linux kernel itself to both drives. A separate ext4 partition was configured for /boot on only one drive, because **Grub2 does not support ZFS** (https://forums.freebsd.org/threads/25164) , and there appears to be a current lack of alternatives for booting Linux from a ZFS dataset. I had expected to see an installation similar to MirrorDisk/UX for HP-UX, where the firmware is configured with primary and alternate boot paths, and the OS is intelligent enough to manage identical copies of the boot and root filesystems on multiple drives. What I actually found was the root filesystem mirrored by ZFS, but the kernel in /boot is not, nor is the system bootable if the single ext4 /boot partition fails. A fault-tolerant Antergos installation will require RAID hardware—ZFS is not sufficient.

You can download the **Antergos Live ISO** (https://antergos.com/try-it) and write it as a bootable image to a flash drive with the command:

```
# dd bs=4M if=antergos-17.9-x86_64.iso of=/dev
```

Note that the Antergos Minimal ISO **does not support ZFS** (https://github.com/Antergos/Cnchi/issues/573) ; it's only in the Live ISO. Internet access is required while the installer is running. The latest packages will be downloaded in the installer session, and very little is pulled from the ISO media.

After booting your system on the live ISO, ensure that you are connected to the internet and activate the installer dialog. Note the warnings of beta software status—whether this refers to ZFS, Btrfs or other Linux RAID configurations is an open question.



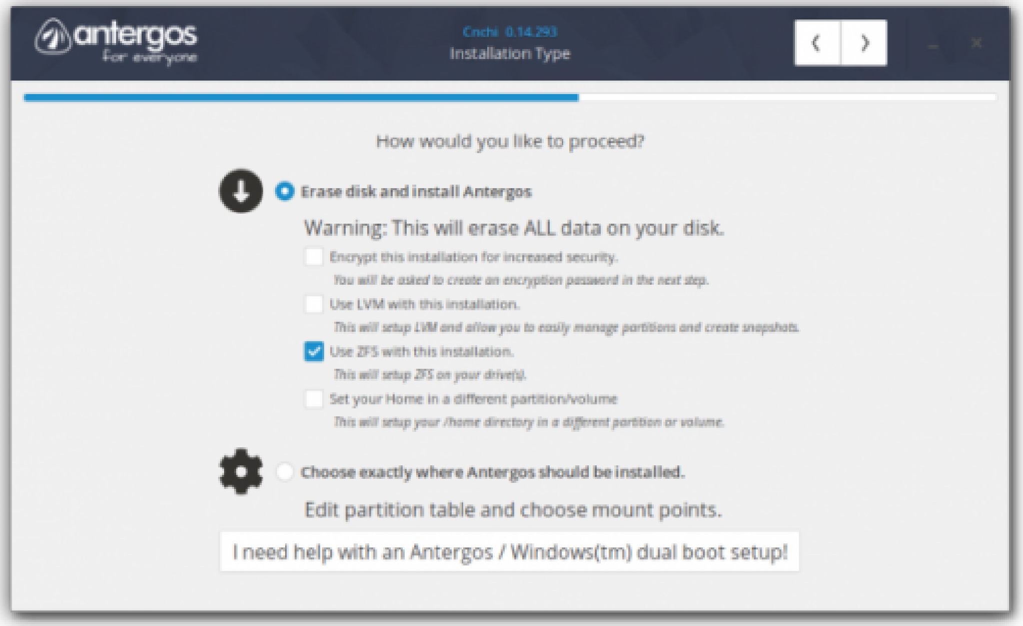
Notice: The Cnchi Installer is beta software.

Cnchi is pre-release beta software that is under active development. It does not yet properly handle RAID, btrfs subvolumes, or other advanced setups. Please proceed with caution as data loss is possible!

If you find any bugs, please report them at http://bugs.antergos.com

Figure 1. Installer Warning

Select your territory or locale, time zone, keyboard layout (I suggest the "euro on 5"), and choose your desktop environment. After I chose GNOME, I also added Firefox and the SSH Service. Finally, a ZFS option is presented—enable it (Figure 2).



How would you like to proceed?

[✓ Erase disk and install Antergos]

Warning: This will erase ALL data on your disk.

- [ ] Encrypt this installation for increased security.
  You will be asked to create an encryption password in the next step.
- [ ] Use LVM with this installation.
  This will setup LVM and allow you to easily manage partitions and create snapshots.
- [✓] Use ZFS with this installation.
  This will setup ZFS on your drive(s).
- [ ] Set your Home in a different partition/volume
  This will setup your /home directory in a different partition or volume.

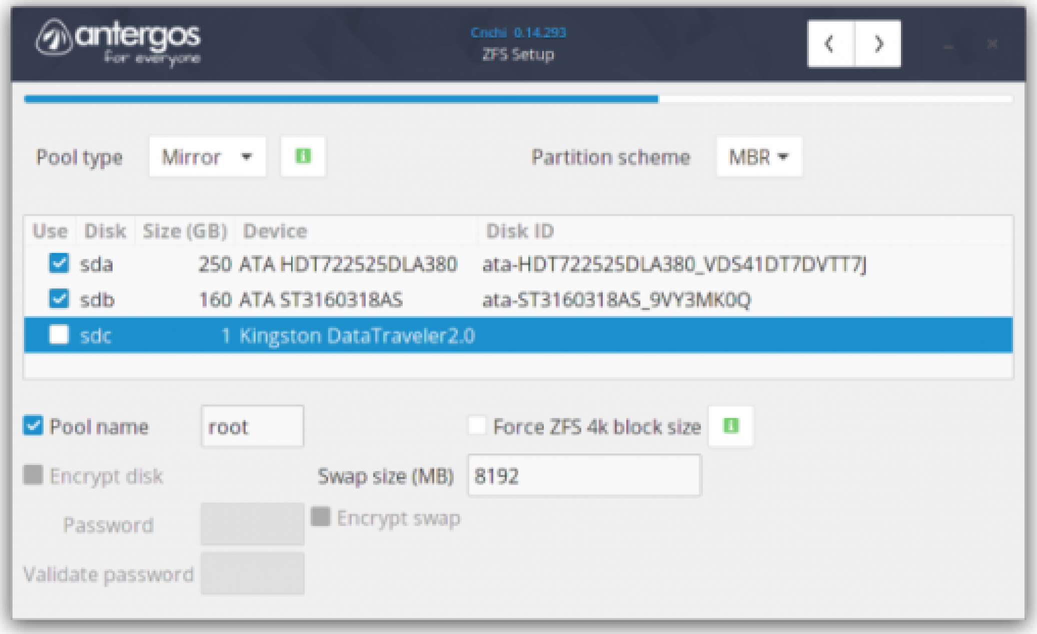
Choose exactly where Antergos should be installed.

Edit partition table and choose mount points.

I need help with an Antergos / Windows(tm) dual boot setup!

Figure 2. Toggle ZFS

As Figure 3 shows, I configured two SATA drives in a zpool mirror. I named the pool "root", which may have caused an error at first boot. Note also the 4k block size toggle—this is a performance-related setting that might be advisable for some configurations and usage patterns.



Pool type: Mirror
Partition scheme: MBR

Use	Disk	Size (GB)	Device	Disk ID
✓	sda	250	ATA HDT722525DLA380	ata-HDT722525DLA380_VDS41DT7DVTT7J
✓	sdb	160	ATA ST3160318AS	ata-ST3160318AS_9VY3MK0Q
	sdc	1	Kingston DataTraveler2.0	

✓ Pool name: root
Force ZFS 4k block size
Encrypt disk
Swap size (MB): 8192
Password
Encrypt swap
Validate password

Figure 3. Configure the zpool

The next pages prompt for the final confirmation before the selected drives are wiped, after which you will be prompted to create a default user.

While the installer is running, you can examine the zpool. After opening a terminal and running `sudo sh`, I found the following information about the ZFS configuration:

```
sh-4.4# zpool history
History for 'root': 2017-09-30 16:10:28
zpool create -f -m /install root mirror /dev/s
zpool set bootfs=root root
zpool set cachefile=/etc/zfs/zpool.cache root
zfs create -V 2G root/swap
zfs set com.sun:auto-snapshot=false root/swap
zfs set sync=always root/swap
zpool export -f root
zpool import -f -d /dev/disk/by-id -R /install
```

Note that /dev/sda2 has been mirrored to /dev/sdb, showing that Antergos has installed a zpool on an MBR partition. More important, these drives are not configured identically. This is not a true redundant mirror with the ability to boot from either drive.

After fetching and installing the installation packages, Antergos will build zfs.ko. You can see the calls to gcc if you run the `top` command in a terminal window.

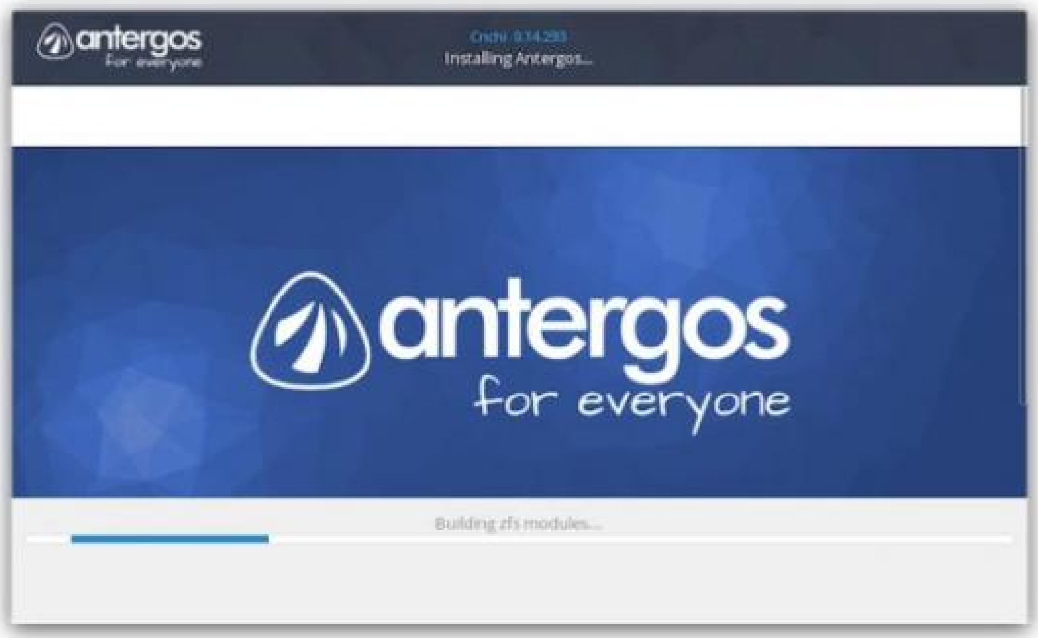


Figure 4. Building ZFS

My installation session completed normally, and the system rebooted. GRUB presented me with the Antergos boot splash, but after booting, I was thrown into single-user mode:

```
starting version 234
```

```
ERROR: resume: no device specified for hiberna
ZFS: Unable to import pool root.
cannot import 'root': pool was previously in u
Last accessed by <unknown> (hostid=0) at Tue O
The pool can be imported, use 'zpool import -f
ERROR: Failed to mount the real root device.
Bailing out, you are on your own. Good luck.

sh: can't access tty; job control turned off

[rootfs ]# zpool import -f root
cannot mount '/': directory is not empty
[rootfs ]# zfs create root/hold
[rootfs ]# cat /dev/vcs > /hold/vcs.txt
```

The zpool import error above also was encountered when the FUSE pool was imported by the native driver. I ran the force import (`zpool import -f root`), which succeeded, then created a new dataset and copied the terminal to it, so you can the session here. After a Ctrl-Alt-Delete, the system booted normally. Naming the zpool "root" in the installer may have caused this problem.

My test system does not have ECC memory, so I attempted to adjust the undocumented kernel parameter below, followed by a reboot:

```
echo options zfs zfs_flags=0x10 >> /etc/modpro
```

After the test system came up, I checked the flags and found that the ECC memory feature had not been set. I set it manually, then ran a scrub:

```
# cat /sys/module/zfs/parameters/zfs_flags
0

# echo 0x10 > /sys/module/zfs/parameters/zfs_f

# cat /sys/module/zfs/parameters/zfs_flags
16
```

```
# zpool scrub root

# zpool status root
  pool: root
 state: ONLINE
  scan: scrub in progress since Sun Oct  1 12:
        251M scanned out of 5.19G at 25.1M/s,
        0B repaired, 4.72% done
config:

        NAME                                  STAT
        root                                  ONLI
          mirror-0                            ONLI
            wwn-0x5000cca20cda462e-part2      ONLI
            wwn-0x5000c5001a0d9823            ONLI

errors: No known data errors
```

I also found that the kernel and initrd do not incorporate version numbers in their filenames, indicating that an upgrade may overwrite them. It likely will be wise to copy them to alternate locations within boot to ensure that a fallback kernel is available (this would need extra menu entries in GRUB):

```
# ls -l /boot
-rw-r--r-- 1 root root 26729353 Sep 30 17:25 i
-rw-r--r-- 1 root root  9225042 Sep 30 17:24 i
-rw-r--r-- 1 root root  5474064 Sep 21 13:34 v
```

You can continue your investigation into the Antergos zpool mirror by probing the drives with `fdisk`:

```
sh-4.4# fdisk -l /dev/sda
Disk /dev/sda: 232.9 GiB, 250059350016 bytes,
Disklabel type: dos
```

```
Device     Boot   Start       End   Sectors
/dev/sda1  *       2048   1048575   1046528
/dev/sda2       1048576 488397167 487348592 23
```

```
sh-4.4# fdisk -l /dev/sdb
Disk /dev/sdb: 149 GiB, 160000000000 bytes, 31
Disklabel type: gpt

Device         Start       End   Sectors  Size
/dev/sdb1       2048 312481791 312479744  149G
/dev/sdb9  312481792 312498175     16384    8M
```

Antergos appears to be playing fast and loose with the partition types. You also can see that the /boot partition is a non-redundant ext4:

```
# grep -v ^# /etc/fstab
UUID=f9fc... /boot ext4 defaults,relatime,data
/dev/zvol/root/swap swap swap defaults 0 0
```

```
# df|awk 'NR==1||/boot/'
Filesystem     1K-blocks    Used Available Use
/dev/sda1         498514   70732    418454  15
```

Antergos is not configuring a completely fault-tolerant drive mirror, and this is a **known problem** (https://github.com/Antergos/Cnchi/issues/569) . The ext4 partition holding the kernel is a single point of failure, apparently required for GRUB. In the event of the loss of /boot, the Live ISO could be used to access the zpool, but restoring full system availability would require much more effort. The same likely will apply to raidz.

Conclusion

ZFS is the filesystem that is "often imitated, never duplicated".

The main contenders for ZFS functionality appear to be Btrfs, Apple APFS and Microsoft's ReFS.

After many years of Btrfs development, it still **lacks performance and maturity** (https://blog.pgaddict.com/posts/friends-dont-let-friends-use-btrfs-for-oltp) ("we are still refusing to support 'Automatic Defragmentation', 'In-band Deduplication' and higher RAID levels, because the quality of these options is not where it ought to be"). Apple **very nearly bundled ZFS into OS X** (https://arstechnica.com/gadgets/2016/06/zfs-the-other-new-apple-file-system-that-almost-was-until-it-wasnt) , but backed out and produced **APFS** (https://en.wikipedia.org/wiki/Apple_File_System) instead. Microsoft is also trying to create a next-generation filesystem named **ReFS** (https://en.wikipedia.org/wiki/ReFS) , but in doing so it is once again proving Henry Spencer's famous quote, "Those who do not understand Unix are condemned to reinvent it, poorly." ReFS will lack compression, deduplication and copy-on-write snapshots.

All of us have critical data that we do not wish to lose. ZFS is the only filesystem option that is stable, protects our data, is proven to survive in most hostile environments and has a lengthy usage history with well understood strengths and weaknesses. Although many Linux administrators who need its features likely will load ZFS, the installation and maintenance tools have obvious shortcomings that can trap the unwary.

It is time once again to rely on Oracle's largesse and ask them to open the ZFS filesystem fully to Linux for the benefit of the community. This will solve many problems, including Oracle's, and it will engender goodwill in the Linux community that, at least from a filesystem perspective, is sorely lacking.

Disclaimer

The views and opinions expressed in this article are those of the author and do not necessarily reflect those of *Linux Journal*.