Guide to setting up a probox RAID-Z (RAID 5) system in Ubuntu 14.04
Luke Sjulson, June 2015

These instructions describe how to make a probox raid array using the raid-z functionality of zfs, which is a next-generation filesystem with a lot of features superior to ext4fs, particularly with respect to protecting your data against corruption. zfs takes the disks in your probox and assembles them into something called a "zpool," which looks and acts like one giant hard drive, except that it has transparent features such as error checking, redundancy against drive failure, and on-the-fly compression. I had this running under Xubuntu 12.04 before, and these instructions should work for that as well, but I've only tested these current directions in Xubuntu 14.04.

1) install zfs on your machine (http://serverascode.com/2014/07/01/zfs-ubuntu-trusty.html)

install zfs and run the module:

```
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:zfs-native/stable
sudo apt-get update
sudo apt-get install -y ubuntu-zfs
sudo modprobe zfs
```

tell zfs not to use more than 1 GB of RAM for cache, and also to automatically mount your drives upon boot:

```
sudo bash
echo "options zfs zfs_arc_max=1073741824" >> /etc/modprobe.d/zfs.conf
echo "zfs mount -a" >> /etc/rc.local
exit
```

2) assemble the probox and configure your zpool

Put the drives in Probox and use gparted to make partitions. Use "gpt" as the type of partition table. Make the primary partition the full size of the drive minus 1% of the total capacity (e.g. 30 GB for a 3 TB drive). Make only one partition per drive, and leave it unformatted. By discarding a small percentage of the drive's capacity, you can replace a broken 2.999 TB HD with a new 2.998 TB HD (e.g. two drives from different manufacturers with nominal 3 TB capacity). Before closing gparted, write down which partitions you are going to use (e.g. /dev/sdb1, /dev/sdc1, etc.). The drive letters will be assigned in order from top to bottom in the probox.

Next, find your partitions by UUID, the "universal unique identifier" of each partition:

```
ls -l /dev/disk/by-uuid
```

You can know which drive corresponds to which UUID because the device name (in terms of /dev/sda1 etc.) will go in order from top to bottom. So you'll see four drives in a row, and

then create the RAID array (a "zpool") by entering this very long command, all in one line:

```
sudo zpool create -o ashift=12 probox_zpool raidz1
/dev/disk/by-uuid/117097d4-96f8-4600-9b20-ef0d8952817e
/dev/disk/by-uuid/3f980c2c-d5be-4797-a38a-a1f50b60bb4f
```

etc. where you're cutting and pasting the UUID of each partition you want to be part of your RAID array. It's a good idea to enter this into a text editor first, making sure this is all one long line, then cutting and pasting into the terminal.

This creates a zpool called probox_zpool, which is mounted in your root directory as /probox_zpool. You will want to make it user writable

```
sudo chmod a+rwx /probox_zpool
```

so you can put your data there. However, before you copy data onto the RAID array you should:

```
sudo zfs set compression=gzip-1 probox_zpool
```

this enables on-the-fly gzip1 compression, which is the lowest level of gzip. I tested different compression settings and found that this one provides decent compression of .dat files with minimal overhead. Compression supposedly makes reading the data slightly faster but at the cost of increased CPU usage, which for our purposes is usually a good tradeoff. The data compression actually occurs when the data is copied onto the zpool - if the data is already there and you enable compression, it stays uncompressed. To check your compression ratio, use:

```
sudo zfs get compressratio
```

3) install maintenance scripts to run tests routinely and notify you of any errors

make a directory to hold the scripts
```
mkdir ~/scripts
```

then copy the scripts **run_once** and **zfscheck** to the scripts folder and make them executable
```
chmod u+rwx zfscheck run_once
```

edit the top two lines of zfscheck to insert your username and home directory.

Then make sure run_once is run every time a shell is opened. This just saves your screen's identifier to disk so that the script can make on-screen notifications if it detects errors
`echo ~/scripts/run_once >> ~/.bashrc`

then add some lines to root's crontab so the disks get checked regularly. start with
`sudo crontab -e`

then paste these lines into the crontab (editing the first line to give the path to where you saved the scripts):

```
SHELL=/bin/bash
0 3 * * * /home/luke/scripts/zfscheck # check for errors at 3AM daily
* 0 * * 6 /sbin/zpool scrub probox_zpool # do full drive scans weekly
```

4) OPTIONAL: configuring zfs "datasets" for features such as snapshots

ZFS takes your disks and puts them together into a big virtual hard drive called a zpool. You can split the zpool up further into "datasets," which act like directories on the zpool. The advantage of this is that various features can be enabled or disabled at the dataset level rather than the zpool level. For example, you can have one dataset be compressed, and another one isn't.

One useful feature for our purposes is called "snapshots," which is the same as the "Time Machine" feature on OS X. The idea is that the system takes regular "snapshots" of the state of your zpool and saves some of them. If you accidentally delete a file or screw something up, you can revert the state of your drive back to the relevant snapshot. This doesn't involve duplicate data being stored - every time you take a snapshot, ZFS actually stores only the changes between the current files and the prior snapshots. However, if you delete a bunch of huge files to free up disk space, that space doesn't actually become available for use until the snapshots holding that data for potential recovery get deleted. You can either delete the snapshots manually, or use a utility like zfs-auto-snapshot, which saves snapshots every 15 mins, keeping 4 snapshots; hourly snapshots every hour, keeping 24 snapshots; daily snapshots every day, keeping 31 snapshots; weekly snapshots every week, keeping 7 snapshots; monthly snapshots every month, keeping 12. To install this:

```
sudo apt-get install zfs-auto-snapshot
```

Next, configure your datasets. You can have as many or as few as you want (down to one). What I did was create three datasets, which in retrospect was probably unnecessary:

```
sudo zfs create probox_zpool/preprocessing
sudo zfs create probox_zpool/processed
sudo zfs create probox_zpool/rawdata
```

Now, enable snapshots on your datasets:

```
sudo zfs set com.sun:auto-snapshot=false probox_zpool
sudo zfs set com.sun:auto-snapshot=true probox_zpool/preprocessing
sudo zfs set com.sun:auto-snapshot=true probox_zpool/processed
sudo zfs set com.sun:auto-snapshot=true probox_zpool/rawdata
```

to list all existing snapshots:

```
zfs list -t snapshot
```

To manually delete shapshots to free up space, use the `snapshot_cleaner` script

Misc ZFS notes

How to replace a failing disk

First, find the id (or UUID) of the disk/partition. If you built your zpool using UUIDs like I recommended above, use ls -l /dev/disk/by-UUID

```
luke@box:~/temp$ ls -l /dev/disk/by-id/
total 0
lrwxrwxrwx 1 root root  9 Jul 10 09:53 ata-Crucial_CT512MX100SSD1_15020E572648 -> ../../sdb
lrwxrwxrwx 1 root root 10 Jul 10 09:53 ata-Crucial_CT512MX100SSD1_15020E572648-part1 -> ../../sdb1
lrwxrwxrwx 1 root root 10 Jul 10 09:53 ata-Crucial_CT512MX100SSD1_15020E572648-part2 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Jul 10 09:53 ata-Crucial_CT512MX100SSD1_15020E572648-part3 -> ../../sdb3
lrwxrwxrwx 1 root root  9 Jul 10 10:16 ata-WDC_WD40EZRX-00SPEB0_WD-WCC4E5CY5FKZ -> ../../sdg
lrwxrwxrwx 1 root root 10 Jul 10 10:16 ata-WDC_WD40EZRX-00SPEB0_WD-WCC4E5CY5FKZ-part1 -> ../../sdg1
lrwxrwxrwx 1 root root  9 Jul 10 10:16 ata-WDC_WD40EZRX-00SPEB0_WD-WCC4E5DXD81J -> ../../sda
lrwxrwxrwx 1 root root 10 Jul 10 09:53 ata-WDC_WD40EZRX-00SPEB0_WD-WCC4E5DXD81J-part1 -> ../../sda1
lrwxrwxrwx 1 root root  9 Jul 10 10:16 usb-ST3000DM_001-1ER166_152D00539000-0:0 -> ../../sdc
lrwxrwxrwx 1 root root 10 Jul 10 09:53 usb-ST3000DM_001-1ER166_152D00539000-0:0-part1 -> ../../sdc1
lrwxrwxrwx 1 root root 10 Jul 10 09:53 usb-ST3000DM_001-1ER166_152D00539000-0:0-part2 -> ../../sdc2
lrwxrwxrwx 1 root root  9 Jul 10 10:16 usb-ST3000DM_001-1ER166_152D00539000-0:1 -> ../../sdd
lrwxrwxrwx 1 root root 10 Jul 10 09:53 usb-ST3000DM_001-1ER166_152D00539000-0:1-part1 -> ../../sdd1
lrwxrwxrwx 1 root root 10 Jul 10 09:53 usb-ST3000DM_001-1ER166_152D00539000-0:1-part2 -> ../../sdd2
lrwxrwxrwx 1 root root  9 Jul 10 10:16 usb-ST3000DM_001-1ER166_152D00539000-0:2 -> ../../sde
lrwxrwxrwx 1 root root 10 Jul 10 09:53 usb-ST3000DM_001-1ER166_152D00539000-0:2-part1 -> ../../sde1
lrwxrwxrwx 1 root root 10 Jul 10 09:53 usb-ST3000DM_001-1ER166_152D00539000-0:2-part2 -> ../../sde2
lrwxrwxrwx 1 root root  9 Jul 10 10:16 usb-ST3000DM_001-1ER166_152D00539000-0:3 -> ../../sdf
lrwxrwxrwx 1 root root 10 Jul 10 09:53 usb-ST3000DM_001-1ER166_152D00539000-0:3-part1 -> ../../sdf1
lrwxrwxrwx 1 root root 10 Jul 10 09:53 usb-ST3000DM_001-1ER166_152D00539000-0:3-part2 -> ../../sdf2
```

```
lrwxrwxrwx 1 root root  9 Jul 10 10:16 wwn-0x50014ee20bbd9e97 -> ../../sdg
lrwxrwxrwx 1 root root 10 Jul 10 10:16 wwn-0x50014ee20bbd9e97-part1 -> ../../sdg1
lrwxrwxrwx 1 root root  9 Jul 10 10:16 wwn-0x50014ee26112cf79 -> ../../sda
lrwxrwxrwx 1 root root 10 Jul 10 09:53 wwn-0x50014ee26112cf79-part1 -> ../../sda1
lrwxrwxrwx 1 root root  9 Jul 10 09:53 wwn-0x500a07510e572648 -> ../../sdb
lrwxrwxrwx 1 root root 10 Jul 10 09:53 wwn-0x500a07510e572648-part1 -> ../../sdb1
lrwxrwxrwx 1 root root 10 Jul 10 09:53 wwn-0x500a07510e572648-part2 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Jul 10 09:53 wwn-0x500a07510e572648-part3 -> ../../sdb3
```

Next, check your zpool to make sure you can find the failing one:

```
luke@box:~/temp$ sudo zpool status
  pool: probox_zpool
 state: ONLINE
  scan: scrub repaired 0 in 17h1m with 0 errors on Sat Jul  4 05:56:45 2015
config:

        NAME                                            STATE     READ WRITE CKSUM
        probox_zpool                                      ONLINE       0     0     0
          raidz1-0                                      ONLINE       0     0     0
            usb-ST3000DM_001-1ER166_152D00539000-0:0-part1  ONLINE       0     0     0
            usb-ST3000DM_001-1ER166_152D00539000-0:1-part1  ONLINE       0     0     0
            usb-ST3000DM_001-1ER166_152D00539000-0:2-part1  ONLINE       0     0     0
            usb-ST3000DM_001-1ER166_152D00539000-0:3-part1  ONLINE       0     0     0

errors: No known data errors
```

In my case, this is the 0:0-part1 disk that's failing. Then I plugged in a new disk into a disk caddy. I made a full-disk partition on it so it would show up in /dev/disk/by-id. Then I used this command:

```
luke@box:~/temp$ sudo zpool replace probox_zpool usb-ST3000DM_001-1ER166_152D00539000-0:0-part1
ata-WDC_WD40EZRX-00SPEB0_WD-WCC4E5CY5FKZ
```

It's sudo zpool replace <zpoolname> <ID of failing partition> <ID of replacement DISK (not partition)>

The resilvering takes many hours. When it is done,

```
sudo zfs umount -a
```

to unmount everything, turn off your probox, and replace the failing drive.

How to change a zpool so that it is based on ID rather than device name (e.g. “/dev/sda1”)

```
# zpool export probox_zpool
# zpool import -d /dev/disk/by-id probox_zpool
```

This worked for me. The problem is that plugging in a second probox might change the by-id designation. However, the problem is that if you use UUIDs, then you can only make a zpool out of partitions, not whole disks, which reduces performance.