Big, fast, and stable storage on a budget

Build and maintain a fast storage server with ZFS and RAID-Z

Skill Level: Advanced

Alfredo Deza (alfredodeza@gmail.com)
Software Engineer

28 Jun 2011

Create flexible and scalable storage without sacrificing your budget or speed. Using the ZFS file system and RAID-Z, you can easily recover from a corrupt operating system and deal with common hard drive failures.

Introduction to ZFS and RAID-Z

ZFS is one of the most advanced file systems currently available. The acronym stands for zettabyte file system, which is a direct reference to its storage capabilities: up to 256 quadrillion zettabytes. To put it another way, a single zettabyte equals 1,073,741,824 terabytes!

This unique and powerful tool is multi-faceted. We will concentrate on portions relevant to building a scalable, redundant storage server. This takes us to RAID-Z, which, as the name implies, is similar to the RAID technology (more specifically to RAID-5).

Atomic data writes
RAID-Z and the ZFS file system use *atomic data writes*. Data is either fully written to disc or not at all. There is no in-between. This almost always guarantees that the writes will not be corrupted by failures related to network connectivity, power, or operating system crashes.

Those familiar with a RAID are probably wondering what device (or controller) will be handling the disks and data. With RAID-Z, everything is handled by the file system

itself. However, unlike similar technologies, it will self-heal corrupted blocks and fix them on the fly without user interaction. RAID-Z is constantly verifying data checksums for integrity and is able to identify blocks needing to be rebuilt. It does this prior to the requested data reaching the user.

Because several operating systems support both RAID-Z and ZFS, the OS is your preference. Most of the examples and techniques discussed in this article should apply and work correctly regardless of the OS.

Storage design

How much space do you need now? How much space do you anticipate needing in a year? What is the appropriate number of hard drives for this setup? These are all good questions to consider before beginning. I will discuss each of these design decisions for optimal preparation.

> **Testing sandbox**
> For demonstration purposes, the examples are done with five devices at two gigabytes each. I strongly suggest you set up a testing sandbox in a virtual environment if you are unfamiliar with RAID-Z or ZFS.

Operating system

It is highly recommended to use an operating system that natively supports the ZFS filesystem to take advantage of all the features and performance. All of the examples that will follow use OpenIndiana as the operating system (a community maintained fork of Open Solaris), so the commands and terminal interaction will reflect that environment. (See Resources for more about OpenIndiana.)

However, if you are more comfortable with Linux® and know the compatible commands to get the same results this article achieves then you should see if your distribution of choice supports *zfs-fuse*, a port of the ZFS filesystem.

Storage space

Because they will directly affect the rest of your choices, it is vital to plan for storage space needs at the beginning of the design phase. The space will also be affected by the type of RAID-Z configuration required (more on this in the RAID-Z Options section).

Do not count on the OS as part of the total storage. I recommend installing the OS on a separate hard drive as this will make it easier to reinstall or repair the system without affecting the share.

Make sure whatever number you have in mind matches your RAID-Z needs (e.g., at least five hard drives for this demonstration). Although RAID-Z setups can have hard drives that are different in size (for example, a 5 x 2 terabyte drive and a 1 x 1 terabyte drive), ZFS will effectively define the hard drives by the smallest one in the pool.

Optimal performance is derived from same brand, same-sized drives.

Overall storage size
When deciding on total storage, remember that the final size will vary depending upon the RAID-Z configuration. For example, two terabytes in a mirror setup requires four terabytes total.

RAID-Z options

There are numerous configurations available in a RAID-Z setup. I will cover two, then go into more detail on the final choice for this project.

RAID-Z is the least fault tolerant. It offers a one disk parity in the pool. If you have issues with more than one drive, you will have a corrupted, non-recoverable storage pool.

RAID-Z2 uses two disks for parity. Two disks provide safety even with issues such as a corrupted or malfunctioning disk. Although the RAID-Z2 takes two extra drives to set up properly, it is more fault tolerant than a plain RAID-Z setup.

At a minimum, a RAID-Z2 setup should be configured with five drives total (two used for parity) which will leave three drives to be counted as overall storage. For example, two terabyte drives would result in about six terabytes of storage.

Hard drives

As I mentioned before, it is best to go with same brand, same-sized drives. It is safer to stay with a pool of drives that are the most similar.

Optimally, purchase at least one extra hard drive when building the storage server to reduce potential delays. It is not uncommon to receive a hard drive with manufacturing problems.

Growing and scalability

Additional drives means speed increases as data is spread across the space, therefore, it is important to make sure the total size matches the target storage space. There are some *SATA* controllers that allow expansion of the motherboard, but may create a bottleneck if not planned for in advance.

There are three key elements for scalable and growable storage:

1. Number of hard drives
2. Overall storage capacity
3. Server capacity of hard drives

Just recently, I built a sixteen terabyte storage server, that includes the ability to grow to twice the space. This allowed for unexpected growth and helped control the budget.

Creating a pool

Initially, we decided to install the operating system on a separate drive, and we have five other devices available. We need to take note of the labels for each device to specify them later on when we create the storage pool.

Listing 1. Find the device labels

```
root@raidz:~# format
Searching for disks...done

AVAILABLE DISK SELECTIONS:
0. c7d0 DEFAULT cyl 4092 alt 2 hd 128 sec 32
/pci@0,0/pci-ide@1,1/ide@0/cmdk@0,0
1. c9t0d0 DEFAULT cyl 1022 alt 2 hd 128 sec 32
/pci@0,0/pci8086,2829@d/disk@0,0
2. c9t1d0 DEFAULT cyl 1022 alt 2 hd 128 sec 32
/pci@0,0/pci8086,2829@d/disk@1,0
3. c9t2d0 DEFAULT cyl 1022 alt 2 hd 128 sec 32
/pci@0,0/pci8086,2829@d/disk@2,0
4. c9t3d0 DEFAULT cyl 1022 alt 2 hd 128 sec 32
/pci@0,0/pci8086,2829@d/disk@3,0
5. c9t4d0 DEFAULT cyl 1022 alt 2 hd 128 sec 32
/pci@0,0/pci8086,2829@d/disk@4,0

Specify disk (enter its number):
```

We issue `format` command to discover the labels (or IDs) but we do not enter a number as a selection. Hit `Ctrl-C` to exit from the prompt.

I installed the OS on device 0 and later decided to add the other five devices, so the following labels are the interesting for the storage pool:

- c9t0d0
- c9t1d0

- c9t2d0
- c9t3d0
- c9t4d0

A fascinating feature of ZFS is that is quickly creates the file system itself. Formatting or reformatting hard drives is often time-consuming. The creation of the file system in this case took about two seconds. On average, this should not take more than a few seconds, regardless of the size of the hard drives.

Listing 2. Create the storage pool

```
zpool create tank raidz2 c9t0d0 c9t1d0 c9t2d0 c9t3d0 c9t4d0
```

`Zpool create tank` command tells the file system to build and format some drives, and label this storage pool as *tank*. I specified `raidz2` as the RAID-Z type, but you could select `raidz` or `raidz3`. The IDs for the five devices I selected are in Listing 1.

We have accomplished another significant step: the storage pool is already mounted and ready to use!

Listing 3. Verifying pool creation

```
root@raidz:~# df -h
Filesystem             size   used  avail capacity  Mounted on
swap                   642M    16K   642M     1%    /tmp
swap                   642M    44K   642M     1%    /var/run
rpool/export/home      7.8G    21K   4.1G     1%    /export/home
rpool                  7.8G    77K   4.1G     1%    /rpool
tank                   5.8G    34K   5.8G     1%    /tank
```

The command output was shortened for brevity, but you can see that *tank* is displayed with almost six gigabytes and already mounted at the root level of the file system. Because we chose a two disk parity setup, the effective storage size will be close to the total of the other three devices. Now that we know the storage pool is mounted and ready, it let's check the health status of the disks and the overall array.

Listing 4. Storage pool status

```
root@raidz:~# zpool status tank
pool: tank
state: ONLINE
scrub: none requested
config:

NAME        STATE     READ WRITE CKSUM
```

```
tank        ONLINE       0     0     0
raidz2      ONLINE       0     0     0
c9t0d0      ONLINE       0     0     0
c9t1d0      ONLINE       0     0     0
c9t2d0      ONLINE       0     0     0
c9t3d0      ONLINE       0     0     0
c9t4d0      ONLINE       0     0     0

errors: No known data errors
```

If so desired, nested filesystems can be created within the storage pool *tank*. This is very useful if you wish to set specific storage needs that are isolated and permissions are tied to the share itself. If you only have one share, this can be complicated.

Listing 5. Adding shares

```
root@raidz:/tank# zfs create tank/bar
root@raidz:/tank# zfs list
NAME                      USED  AVAIL  REFER  MOUNTPOINT
rpool                    3.74G  4.07G  77.5K  /rpool
rpool/ROOT               2.99G  4.07G    19K  legacy
rpool/swap                512M  4.43G   137M  -
tank                      160K  5.83G  34.8K  /tank
tank/bar                 34.0K  5.83G  34.0K  /tank/bar
```

We created a share named *bar* that shows up like a separate share. Great! However, don't let the almost six gigabytes displayed confuse you. We haven't doubled the total share space, rather it is useful to know how much we have available from the total storage pool.

Recovery

Now that the storage pool is up, we need to be aware that data might get corrupted or hard drives can be faulty. Fortunately, ZFS and RAID-Z allow easy administration for these tasks. This section provides a simulation of a bad hard drive and a total system failure where the operating system cannot recover.

Dead hard drive

In this scenario, I have deliberately made a hard drive inaccessible to the server. Let's see what ZFS reports when this happens.

Listing 6. Dead drive status

```
root@raidz:~# zpool status tank
pool: tank
state: DEGRADED
```

```
status: One or more devices could not be opened.
Sufficient replicas exist for the pool to continue functioning in
a degraded state.
action: Attach the missing device and online it using 'zpool online'.
scrub: none requested
config:

NAME        STATE     READ WRITE CKSUM
tank        DEGRADED     0     0     0
raidz2      DEGRADED     0     0     0
c9t0d0      ONLINE       0     0     0
c9t1d0      ONLINE       0     0     0
c9t2d0      ONLINE       0     0     0
c9t3d0      ONLINE       0     0     0
c9t4d0      UNAVAIL      0     0     0  cannot open

errors: No known data errors
```

Our storage pool just became degraded. The status command shows what is happening in the pool and which hard drive seems to be failing. It is also important to note that the storage is fully usable. Remember, I selected *raidz2* that allowed up to two simultaneous hard drive failures. To make this scenario a bit more realistic, I copied some log files into the storage to simulate a degraded state.

After replacing the defective hard drive, I started the server again to observe the results.

Listing 7. Recovering from a degraded state

```
root@raidz:~# zpool status tank
pool: tank
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.'
scrub: resilver completed after 0h0m with 0 errors on Tue Nov  9 07:25:23 2010
config:

NAME        STATE     READ WRITE CKSUM
tank        ONLINE       0     0     0
raidz2      ONLINE       0     0     0
c9t0d0      ONLINE       0     0     0
c9t1d0      ONLINE       0     0     0
c9t2d0      ONLINE       0     0     0
c9t3d0      ONLINE       0     0     0
c9t4d0      ONLINE       0     0     2  31.5K resilvered

errors: No known data errors
```

Not only has the share fully recovered without any interaction from us, the output provides detail about what happened, what actions were taken, and possible options if you are not satisfied with the results (for example, corrupted files).

The **resilvered** tag means ZFS reconstructed that amount of data for that specific drive.

Unrecoverable operating system

In this section, I replicate a system error with an inability to boot. Since the system was installed on a separate hard drive away from precious storage, the system can be reinstalled on the same hard drive without touching the devices associated with the share.

Just as before, we rely on `zpool` command to diagnose the storage and see if we can attach it again.

Listing 8. Pool status after re-installation

```
root@reinstalled:~# zpool status
pool: rpool
state: ONLINE
scrub: none requested
config:

NAME        STATE     READ WRITE CKSUM
rpool       ONLINE       0     0     0
c7d0s0      ONLINE       0     0     0

errors: No known data errors
```

This verifies that the installation worked, but it still doesn't see the storage.

Listing 9. Finding the pool

```
root@reinstalled:~# zpool import
pool: tank
id: 11026414298329032494
state: ONLINE
status: The pool was last accessed by another system.
action: The pool can be imported using its name or numeric identifier and
the '-f' flag.
see: http://www.sun.com/msg/ZFS-8000-EY
config:

tank        ONLINE
raidz2      ONLINE
c9t0d0      ONLINE
c9t1d0      ONLINE
c9t2d0      ONLINE
c9t3d0      ONLINE
c9t4d0      ONLINE
```

Great! That is the pool. It's in perfect shape and ready to be imported.

Listing 10. Importing and verification of the pool

```
root@reinstalled:~# zpool import tank
cannot import 'tank': pool may be in use from other system, it was
```

```
last accessed by raidz (hostid: 0x4013af) on Tue Nov  9 07:31:33 2010
use '-f' to import anyway
root@reinstalled:~# zpool import -f tank
root@reinstalled:~# zpool status tank
pool: tank
state: ONLINE
scrub: none requested
config:

NAME        STATE     READ WRITE CKSUM
tank        ONLINE       0     0     0
raidz2      ONLINE       0     0     0
c9t0d0      ONLINE       0     0     0
c9t1d0      ONLINE       0     0     0
c9t2d0      ONLINE       0     0     0
c9t3d0      ONLINE       0     0     0
c9t4d0      ONLINE       0     0     0

errors: No known data errors
root@reinstalled:~# df -h
Filesystem             size   used  avail capacity  Mounted on
/                      7.8G   2.8G   4.2G    41%    /
/devices                 0K     0K     0K     0%    /devices
swap                   800M   368K   800M     1%    /etc/svc/volatile
rpool/export           7.8G    21K   4.2G     1%    /export
rpool                  7.8G    79K   4.2G     1%    /rpool
tank                   5.8G   104K   5.8G     1%    /tank
```

I attempted to import the pool, but the output reports that it might be in use by another system, in this case, the one previously installed. I forced the import and verified the status, which returns clean with no errors. Finally, I checked the mounting and location, and confirmed that it is exactly the same as before.

We have run through common scenarios and have recovered the storage with little effort.

Sharing

Sharing is straightforward. The next section covers sharing with NFS as it is supported in several operating systems, including various flavors of Linux and UNIX.

As ZFS has native NFS support, you only need to know the file system, type of sharing and the location.

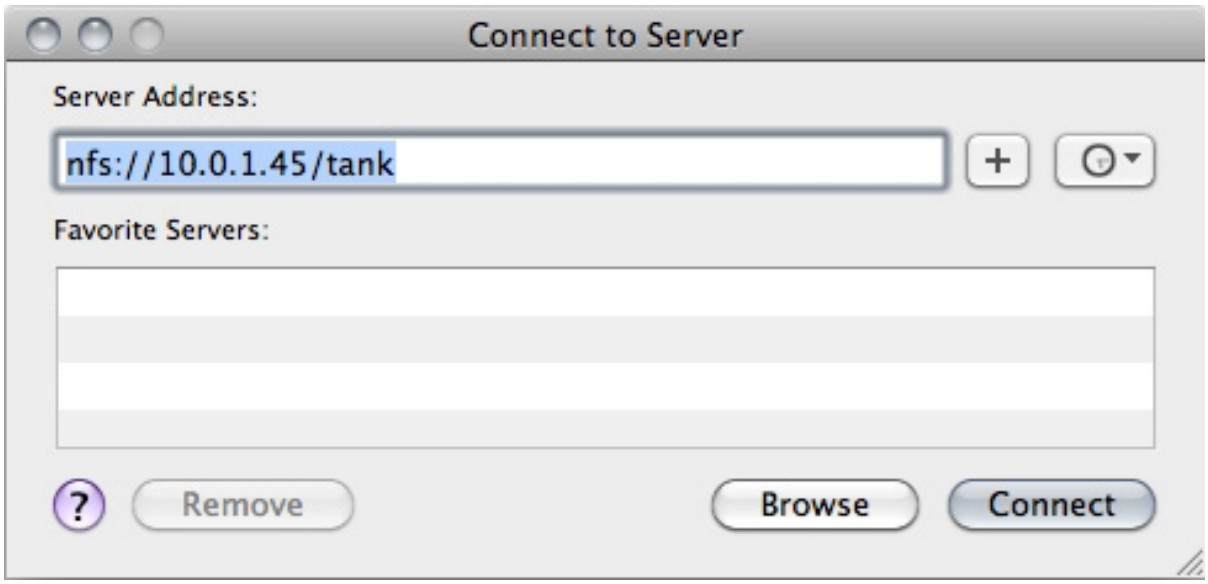
Listing 11. Sharing through NFS

```
root@reinstalled:~# zfs set sharenfs=on tank
```

In this case, *tank* is the share and it is instantly shared over NFS in the local network. Also, you can set permissions on specific shares, but it is worth noting that if you share the root of the share, you are automatically sharing any present child

shares, or those created in the future.

Figure 1. Connecting to NFS share in Mac OSX



It is usually time-consuming to set up shares on servers, but the ability of the file system itself to enable sharing through a specific protocol such as NFS is helpful.

The previous share didn't have protection or permissions set, so let's see how to modify that share so it allows *read and write* actions to a specific subnet in the network.

Listing 12. Network-based permissions in ZFS share

```
zfs set sharenfs=rw=@10.0.0.0/8 tank
```

This type of network restriction works without additional configuration. As soon as this command is sent, access is restricted and only the specified subnets will have read and write access to the share.

Permissions can be finely tuned to specific users or groups, and implementation and configuration may vary from system to system.

Conclusion

ZFS has been ported to work on many operating systems and is currently available in many UNIX and Linux versions.

While there is significantly more information to discuss about ZFS and RAID-Z, this article has looked at the most important items for building a share, from design

decisions to sharing the storage over the network.

RAID-Z is not the only type of disk array available to ZFS. Mirrors are usually faster and less memory-consuming, but require equal numbers of hard drives to use it. In this case, with five hard drives ready for the share, we would have not been able to use them all.

ZFS is ideal when trying to build storage in a server that can hold multiple devices and then share it over the network. However, you would not be able to enjoy all the benefits if you do not have at least two separate devices. With more devices, you receive better performance.

I hope you are ready to explore this file system and find it as useful as I have.

Resources

Learn

- OpenIndiana A community developed, open source UNIX operating system that fully supports ZFS.
- Illumos A fully open community fork of the OpenSolaris operating system
- Nexenta OS Combines the original Unix kernel for ZFS and the Debian/Ubuntu package manager (aptitude).
- ZFS in Ubuntu, a wiki guide that describes working in Ubuntu.
- ZFS Best Practices
- FreeBSD A complete guide about ZFS in FreeBSD.
- History and features of ZFS
- ZFS vs other RAIDs
- Events of interest: Check out upcoming conferences, trade shows, and webcasts that are of interest to IBM open source developers.
- developerWorks Open source zone: Find extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- IBM trial software: Innovate your next open source development project using trial software, available for download or on DVD.

Discuss

- developerWorks community: Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis. Help build the Real world open source group in the developerWorks community.

About the author

Alfredo Deza

Alfredo Deza is a software engineer, former professional athlete, and Olympian with a strong background in system administration. He is passionate about open source software and is a regular speaker to local technology groups and international conferences such as PyCon. In his free time, he tries to improve his photography skills and enjoys contributing to open source projects.