Home

The Zettabyte Filesystem

ZFS Workshop Space

Navigate space

Recently Updated

- **Home**
  updated by David Gabriel
  (view change)
  less than a minute ago
- **Step 1 - Access workshop documentation, log on to ZFS server**
  updated by David Gabriel
  (view change)
  about 2 hours ago
- **Step 8 - Zero cost clones**
  updated by David Gabriel
  (view change)
  about 3 hours ago
- **How to get it**
  created by David Gabriel
  about 3 hours ago
- **Cache Layers (ARC, ZIL)**
  updated by David Gabriel
  (view change)
  Jun 14, 2011
- **Zero-Cost Clones**
  updated by David Gabriel
  (view change)
  Jun 14, 2011
- **clone1.png**
  attached by David Gabriel
  Jun 14, 2011
- **clone1**
  attached by David Gabriel
  Jun 14, 2011
- **What is ZFS and what can(t) it do**
  updated by David Gabriel
  (view change)
  Jun 14, 2011
Pres

Pretence - what is this all about

This is...

- about ZFS, its features and limitations
- just about as much about Solaris as necessary

This is not...

- about Solaris or any of its forks
- about Solaris setup and administration
- about dtrace

What is ZFS and what can('t) it do
Features

- Stable and mature (in widespread use on sun/oracle enterprise appliances for years)
- Safe storage (up to triple-parity)
- Failure resilient (checksums, self healing)
- Copy on write (snapshots, never overwrite data)
- Integrated volume management (flexibility)
- Performance ([L2]ARC and ZIL cache)
- Online dedup and compression (save space)
- Zero cost clones (quick provisioning)
- Replication (desaster recovery / archive)
- Easily expandable (just add new arrays to the 'pool')
- Unified storage
  - NAS: CIFS and NFS (yes, even AD integration and nfsv4)
  - SAN: iSCSI and FC(oE) (COMSTAR iSCSI Target)
- Insanely high limits for... like everything! (256 zettabytes (2^78 bytes) in any zpool, 2^48 entries in any individual directory etc.)

Current limitations

- a vdev (disk) can not be removed from a pool
- vdevs can not be added to existing arrays (growing a RAID array is not possible, only adding of a new array to a pool)

How to get it

- At the moment several native implementations exist.
  - Oracle (formerly Sun) native in Solaris or Solaris Express
  - any Solaris fork: Openindiana, Illumos, Schilix, Belenix, ...
  - Nexenta

- and there are also some more or less production-ready ports and re-implementations:
  - ZFS on Linux - LLNLs continued work on a native port after KQ Infotechs death
  - Debian/GNU/kFreeBSD - Debian with a BSD Kernel, old zpool version though
  - zfs-fuse - ZFS via fuse port. good features, lacking performance

How it works
Simplified design representation

- Read Cache: (L2)ARC, speed up access to 'hot' data
- Write Cache: ZIL (ZFS Intent Log), speed up SYNCHRONOUS writes
RAID-Levels and Parity
RAID-0 (stripe)

- No RAID because not redundant
- Use this for L2 ARC cache
  - ARC-Cache speeds up reads, you won't lose data if a device fails

RAID-1 (mirror)

- Use this for ZIL cache
  - ZIL cache speeds up synchronous writes, ALWAYS use redundant drives or your data will be at risk!

RAID-10 (stripe+mirror)

RAID-Z(2,3) (data+parity)

- RAID-Z2 and RAID-Z3 add more parity-disks
- Now why is RAID-Z bad? I have a parity disk so all is fine!? Well...

SATA drives are specified with an uncorrectable read error rate (URE) of \(10^{14}\). Which means that once every 100,000,000,000,000 bits an error will occur. One hundred trillion bits is about 12 terabytes

With a 7 drive RAID-5 disk failure, you'll have 6 remaining 2 TB drives. As the RAID controller is reading through those 6 disks to reconstruct the data from the failed drive, it is almost certain it will see an URE.
Not exactly redundant...

⚠️ Use at least at RAID-Z2 or RAID-1(0) for production systems

Copy On Write
Block (Recordsize) Checksums
ZFS has a flexible recordsize for each FS you create.

- **default recordsize is 128k**
- **modify recordsize in SAN scenarios (LUN, Virtual Machine Images, Databases etc.)**
- **modify recordsize via zfs set command, i.e. to set recordsize to 32k:**
  ```
  ~# zfs set recordsize=32k pool/fs
  ```
- **recordsize is adjustable between 512b and 128k in magnitudes of 2**

- ZFS creates checksums (Fletcher, AES) for every record.
- No need for RAID-rebuild if a record is corrupted
- Can speed up hardware troubleshooting (disk, controller or cable error?)

### Selfhealing

![Diagram of Selfhealing process](image)

### Snapshots

#### Snapshot Features

- Point-In-Time representation of the FS
- Snapshots initially take up no space
- Snapshots have no performance impact
- Snapshots are accessible via the FS, just change to the hidden ".zfs" folder and copy contents from there to the active FS
- Snapshots grow as the active FS changes
- The content of a snapshot can be sent to another server via "zfs send | ssh root@server zfs receive..." command
  - Scripts@Infrageek
- Use this for off-site backups and/or archiving
- Snapshots are the base for zero-cost clones
Snapshots, simplified graphical representation

Active Filesystem
R/W

Snapshot@10:00h
R/O
Holds only records changed between 09:00-10:00

Snapshot@09:00h
R/O
Holds only records changed between 08:00-09:00

Snapshot@08:00h
R/O
Holds only records changed between hh:mm-08:00

....
Snapshot@hh:mm
R/O
keep any number of snapshots you want!

Zero-Cost Clones
Cache Layers (ARC, ZIL)

ARC (Adjustable Replacement Cache)

ARC Explained

- L1 ARC resides in memory, per default uses up all available memory
  - *limit ARC cache size via parameter in /etc/system*
    
    ```sh
    # limit ARC cache to 1 GB
    set zfs:zfs_arc_max = 1610612736
    ```

- L2 ARC devices can be added to pools (think [PCI-E based] SSD storage) like this:
  
  ```sh
  # zpool add <pool> cache <device1> <device2> ...
  ```

- Keeps tables for most recently used and most frequently used blocks to optimize cache hits

ZIL (ZFS Intent Log)

ZIL Explained

- Speeds up SYNCHRONOUS writes
  - *NFS and OLTP workloads are big sync() fans*
• find out if your pool would benefit from ZIL: `zilstat`

• Did I mention these log devices should always be mirrored?

• ZIL devices can be added to pools like this:

  ```bash
  zpool add <pool> log mirror <device1> <device2> ...
  ```

---

**ZFS Pool and Volume Management**

• Traditional LVM and ZFS models:

  ![Diagram of LVM and ZFS models]

  **Actual Model:**
  LVM and file system

  **New Model:**
  Storage Pool with ZFS

• Some Pool Management Options

  • `zpool` command manages all pool operations (create, destroy, repair/resilver, scrub, etc.)

  • `format` command shows available disks

  ```bash
  # format
  AVAILABLE DISK SELECTIONS:
  0. c1t1d0 <DEFAULT cty 60798 alt 2 hd 255 sec 63>
     /pci0,0/pci0043,2a10c,4/pci1043,824f0/disk0,0
  1. c2t0d0 <ATA-OCZ-VERTEX2-1.28-107.13GB>
     /pci0,0/pci1043,82d40/disk0,0
  2. c2t1d0 <ATA-OCZ-VERTEX2-1.28-107.13GB>
     /pci0,0/pci1043,82d40/disk1,0
  3. c2t2d0 <ATA-OCZ-VERTEX2-1.28-107.13GB>
     /pci0,0/pci1043,82d40/disk2,0
  4. c2t3d0 <ATA-OCZ-VERTEX2-1.28-107.13GB>
     /pci0,0/pci1043,82d40/disk3,0
  5. c2t4d0 <ATA-OCZ-VERTEX2-1.28-107.13GB>
     /pci0,0/pci1043,82d40/disk4,0
  6. c2t5d0 <ATA-OCZ-VERTEX2-1.28-107.13GB>
     /pci0,0/pci1043,82d40/disk5,0
  ```
• Create a striped (RAID-0) pool

```
~# zpool create testpool c2t0d0 c2t1d0
```

• Create a mirrored (RAID-1) pool

```
~# zpool create testpool mirror c2t0d0 c2t1d0
```

• Create a striped+mirrored (RAID-10) pool

```
~# zpool create testpool mirror c2t0d0 c2t1d0 mirror c2t2d0 c2t3d0
```

• Create a single / double / triple parity (RAID-Z[2,3]) pool

```
~# zpool create testpool raidz c2t0d0 c2t1d0 c2t2d0 c2t3d0
~# zpool create testpool raidz2 c2t0d0 c2t1d0 c2t2d0 c2t3d0
~# zpool create testpool raidz3 c2t0d0 c2t1d0 c2t2d0 c2t3d0
```

Some Filesystem Options

• Create new ZFS on testpool

```
~# zfs create testpool/export1
```

• Put quota on it:

```
~# zfs set quota=10g testpool/export1
```

• Dedup / Compression?

```
~# zfs set dedup=on testpool/export1
~# zfs set compression=on testpool/export1
```

• Export via NFS

  • To access the volume on the client with 'root', you have to set it explicitly!

```
~# zfs set sharenfs=ro=@net/24 testpool/export1
~# zfs set sharenfs=rw=@host/32,root=@host/32,sec=sys testpool/export1
```

• Show all FS options
No-Brainers

- Use ECC RAM (for christ's sake, shell out those 10 bucks more)
- Like your data? Use RAID-Z2 (old but true post on storagemojo)
- Again: ZIL: always use a mirrored device (with battery backup if possible)
- Scrub your ZFS regularly (SATA every week, SCSI every month)
- Snapshots are no backup
- One backup is no backup 😞
- Fear of disaster? Use "zfs send" to offsite location

Setup

Hardware

- Asus Rampage2 GENE microATX (ICH10R 6 x SATA, JMicron SATA+PATA, RealTek GBit LAN)
- Core i7-960 LGA1366 4 x 3.2GHz
- 2 x 2GB Corsair ECC DDR3 RAM
- Intel E1000 PCI NIC
- some cheap PCI-E GPU
- Seagate 500GB SATA HDD (rootvol) (to be mirrored)
- 6 x OCZ Vertex2 120GB SSD
- ChiefTec GPS-450 PSU

to come:
LianLi Cube, redundant rootvol-disk, fanless PSU
Performance

IOZone Performance Tests.

6xOCZ SSDs in RAIDZ2 config, recordsize=128k, compression=off, dedup=off, copies=1. BS means blocksize. yep, those are hundreds of mb/s.
Resources

- ZFS Wiki
- ZFS Administration Guide
- Blogs
  - http://constantin.glez.de
  - http://www.zfsbuild.com
- Presentations
- Misc
  - http://www.datadisk.co.uk/html_docs/sun/sun_zfs_cs.htm

Work

Step 2 - Nexenta, Solaris basics
Although Nexenta utilizes apt/dpkg and ported lots of packages from Ubuntu to Solaris, it is NOT binary compatible with GNU/Linux or Ubuntu.

Nexenta brings some useability to the Solaris userspace though, like fully fledged bash, vim, grep etc.

- apt-clone for secure upgrades / package installation (uses zfs snapshots)
- Do not install services that interfere with Solaris system services like cron, syslog...

- little command comparison:

<table>
<thead>
<tr>
<th>Linux</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>apt-get</td>
<td>apt-clone</td>
</tr>
<tr>
<td>distro package management</td>
<td>pkg*</td>
</tr>
<tr>
<td>top</td>
<td>prstat</td>
</tr>
<tr>
<td>iotop</td>
<td>fsstat, iostat</td>
</tr>
<tr>
<td>/etc/init.d/...</td>
<td>svcs, svcadm, cfgadm</td>
</tr>
<tr>
<td>fdisk</td>
<td>format, iostat -E</td>
</tr>
<tr>
<td>lspci</td>
<td>prtconv</td>
</tr>
<tr>
<td>strace &amp; co.</td>
<td>mdb, dtrace</td>
</tr>
</tbody>
</table>

Step 3 - Examine server and disks

- Use the commands iostat and format to list installed disks. ‘^C’ exits from format utility.

```
  # iostat -E
  # format
```

- Notice the different device notation.

Teams: each team is assigned two disks. Please stick to them for the duration of the course.

- Team 1: c2t0d0 c2t1d0
- Team 2: c2t2d0 c2t3d0
- Team 3: c2t4d0 c2t5d0

The disk c1t1d0 contains the syspool (rootfs). Please do not use it during the course of this workshop.
Step 4 - Create zpools

- In this step we learn about the basics of the ZFS pool administration and the ‘zpool’ command

- Create a zpool of two devices

Only one Team member can create a zpool at a time. You might however destroy the pool and create a new one to test different configurations (stripe, mirror, raidz). Please work together so anyone can do all the steps.

```
create a pool

# pool can be any given name
# type can be either empty (stripe), raidz or mirror
# devices are the disks assigned to your team
~# zpool create <pool> <type> <device1> <device2>
```

- Notice how zpool creation takes up virtually no time as no blocks need to be zero’ed out because of the zfs internal end-to-end checksumming method. Your data is always consistent and only used blocks need to be checked.

```
list pools, check status and mountpoint

~# zpool list
~# zpool status
~# df -h
```

```
destroy a pool

~# zpool destroy <pool>
```

- Note that a newly create pool is automatically mounted in the rootfs

- Display all extended attributes of a zpool. We will learn about some of these attributes at a later point.

```
display attributes

~# zpool get all <pool>
```

- Devices in a pool can be taken offline to replace them. This does not take the whole pool offline. Note that this only works for mirror and raidz pools!

```
offline device

~# zpool offline <pool> <device>
```
• Check the status of the pool again. It should be 'DEGRADED'

• Now change into your pool directory and create a file

```
create file

~# cd /<pool>
/<pool> mkdir 1g <filename>
```

• Re-add the device

```
re-add device

~# zpool online <pool> <device>
~# zpool status
~# zpool scrub <pool>
~# zpool status
```

⚠️ Use this technique to incrementally swap devices of lower capacity with devices of higher capacity. After swapping all drives, the whole pool size will grow!

• After the scrub 1g should have been resilvered (rebuild), all devices should be online and the pool status should have changed back to 'ONLINE'

• When all team members have finished please destroy the pool and give notice.

**Step 5 - Working with ZFS**

• In this step we learn about ZFS file system creation and ZFS properties

⚠️ One team member should create a two disk raidz zpool for all team members to work with. Please use the devices you chose at the beginning of the course. Choose ZFS names to your liking, however avoid duplicate names.

```
create pool

~# zpool create <pool> raidz <device1> <device2>
```

• Creating a new ZFS in a zpool is as simple as that:

```
create new zfs

~# zfs create <pool>/<zfs-name>
~# df -h
```
• Notice the newly created ZFS is automatically mounted and accessible.

• Check ZFS properties

```
list properties
~# zfs get all <pool>/<zfs-name>
```

• Most options are fine by default, notice the following options as they can have an impact on performance or general 'behaviour' of the ZFS.

<table>
<thead>
<tr>
<th>Option</th>
<th>What it does</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>quota</td>
<td>limit capacity of this ZFS</td>
<td>-</td>
</tr>
<tr>
<td>reservation</td>
<td>reserve space in the zpool for this ZFS</td>
<td>-</td>
</tr>
<tr>
<td>recordsize</td>
<td>define the recordsize ZFS stores data with</td>
<td>may impact performance if ZFS is used for containers</td>
</tr>
<tr>
<td></td>
<td>must be set at ZFS creation time!</td>
<td>like virtual machines, databases etc.</td>
</tr>
<tr>
<td>compression</td>
<td>turn compression on/off, gzip or lzjb</td>
<td>CPU, save space</td>
</tr>
<tr>
<td>atime</td>
<td>control update of file access times</td>
<td>turn off to speed up certain workload, check tuning guide</td>
</tr>
<tr>
<td>snapdir</td>
<td>show/hide .zf snapshot directory</td>
<td>-</td>
</tr>
<tr>
<td>copies</td>
<td>store data more than once on disk helps with bad blocks, no backup though</td>
<td>write performance, data needs to be written more than once</td>
</tr>
<tr>
<td>usedby*</td>
<td>get stats of snapshots etc.</td>
<td>-</td>
</tr>
<tr>
<td>logbias</td>
<td>tune ZFS for latency or throughput</td>
<td>latency is fine, use throughput for Oracle DBs</td>
</tr>
<tr>
<td>dedup</td>
<td>eliminate blocks with same checksum, store them only once, happens online must be set at ZFS creation time!</td>
<td>CPU, RAM, save space</td>
</tr>
</tbody>
</table>

• Now create a total of 3 ZFS to work with in the latter steps and give notice.

```
create 3 ZFS
~# zfs create <pool>/<my-zfs1>
~# zfs create <pool>/<my-zfs2>
~# zfs create <pool>/<my-zfs3>
```

Step 6 - The options dedup, compression and copies

• In this step we test dedup, compression or both and compare the results. Also we will look into the 'copies' parameter.

  • Team 1: compression
  • Team 2: dedup
  • Team 3: compression+dedup
A tarball containing a servertemplate has been provided for you in /opt/workshop. Please copy the tarball to the root of your zpool for all team members to use.

- everyone please open a second shell (if you have not already done so) and examine disk and cpu stats while doing the further steps:

  **examine server performance**

  `~# iostat -dmcl7 1`
  or
  `~# zpool iostat <pool> 1`

- Team 1 turn on compression:

  **compression**

  `~# zfs set compression=on <pool>/<zfs>`

- Team 2 turn on dedup:

  **dedup**

  `~# zfs set dedup=on <pool>/<zfs>`

- Team 3 turn on dedup and compression:

  **dedup and compression**

  `~# zfs set dedup=on <pool>/<zfs>`
  `~# zfs set compression=on <pool>/<zfs>`

- Now extract the tarball to your ZFS which you have modified before.

  `~# time tar xjpf /<pool>/lucid_template.tar.bz2 -C /<pool>/<zfs>;/ time sync`

- Now examine dedup and compression ratios

  **examine ZFS**

  `~# zpool list`
  `~# zfs get compressratio <pool>/<zfs>`

- Notice how 'DEDUP' is a pool-wide property while compression is a ZFS-wide property.

- Now all teams set the 'copies' parameter to '3' and extract the tarball again. Compare pool and ZFS sizes.
Examining how dedup and compression ratios develop now. With `copies=3` all data will be stored 3 times in the ZFS, obviously dedup has to honor this. Compression helps, however 😊. Please give notice when finished.

**Step 7 - Snapshots**

- In this step we look into the workings of snapshots, probably the most prominent feature of ZFS.

- Start by taking a fresh snapshot of a previously created and still empty ZFS. Snapshots always have to be prefixed with a `@`-character. After creation you can list all snapshots with a modified version of the `zfs list` command

```bash
- # zfs snapshot <pool>/<zfs>@$(date +%Y-%m-%d_%H:%M)
- # zfs list -t snapshot
or
- # zfs list -t snapshot -r <pool>
```

- You should see the freshly created snapshot. Now modify the ZFS by expanding the template server archive into the empty ZFS. Take a fresh snapshot afterwards.

```bash
- # tar xjpf /<pool>/lucid_template.tar.bz2 -C /<pool>/<zfs>/
- # zfs snapshot <pool>/<zfs>@$(date +%Y-%m-%d_%H:%M)
- # zfs list -t snapshot
or
- # zfs list -t snapshot -r <pool>
```

- Let's say 'Th3 N00b' has entered the stage and rm'd the '/etc/' directory of our server template while an automatic script has taken a fresh snapshot afterwards.

```bash
- # rm -rf /<pool>/<zfs>/etc
- # zfs snapshot <pool>/<zfs>@$(date +%Y-%m-%d_%H:%M)
- # zfs list -t snapshot
or
- # zfs list -t snapshot -r <pool>
```

- Notice how the 'USED' column changes to reflect the deleted files. You can also 'get' stats about snapshot usage via the 'usedby' parameters:

```bash
- # zfs get usedbysnapshots
```
• The trusty sysop (you) has spotted 'Th3 N00b’s fault by now. To correct it, enter the ZFS snapshot directory and copy the deleted files from a previous snapshot to the active filesystem.

```bash
# cd /<pool>/<zfs>/.zfs/snapshot/<snapshot-before-deletion-of-etc>
# rsync -avP etc /<pool>/<zfs>/
```

• Done! All is well again thanks to this cool feature. Another option would have been to revert the whole ZFS back to a previous state, however this is an irreversible action. To do so:

```bash
# zfs rollback -f -r <snapshot-before-deletion-of-etc>
```

• When finished with the above steps please give notice.

**Step 8 - Zero cost clones**

• Another handy feature are snapshot based clones, which can be promoted to a standalone ZFS if necessary.

Please work together as no duplicate names of clones are permitted in a ZFS.

• Let’s start with a new snapshot of our recently ‘repaired’ server template. We’ll create 2 clones of it:

```bash
# zfs clone <pool>/<zfs>@<snapshot> <pool>/clone1
# zfs clone <pool>/<zfs>@<snapshot> <pool>/clone2
```

• You can find out whether a ZFS is an original or a clone via the ‘origin’ parameter. Also notice the discrepancy between ‘df -h’ and ‘zfs list’.

```bash
# zfs get origin
# zfs list
# df -h
```

• As the clones are merely copies of the root-inode of the snapshot, they take up no space initially. Only if data is modified in the clone it starts to use up space.

• Clone ZFS parameters can be modified like ‘normal’ ZFS, meaning you can turn on dedup, compression, quota etc.

• Please modify (add, delete, modify data) the clones to your liking and watch the output of ‘zfs list’ as they start to take up space!
Clones are a much better approach to save space than dedup or compression, as clones use neither more CPU nor RAM. Both features offer rich possibilities but one should only use them if clones are not an option.

- Clone everything! Servers, databases, whole environments for test/dev purpose etc.
- Clones are cheap, save space and have no performance impact.

Promote a clone file system so it's no longer dependent on its "origin" snapshot. The clone parent-child dependency relationship is now reversed, so that the origin file system becomes a clone of the specified file system.

**promote a clone**

```bash
~# zfs promote <pool>/clone1
~# zfs get origin
~# zfs list
~# df -h
```

To get rid of a clone just 'destroy' it:

**destroy a clone**

```bash
~# zfs destroy <pool>/clone2
```

Now try to destroy clone1

```bash
~# zfs destroy <pool>/clone1
~# zfs get origin
```

Notice the error. It's not possible to destroy the ZFS 'clone1' because it is not a clone anymore but has taken the role of the original now.

When finished with this task please give notice.

**Step 9 - Network sharing options**

- This chapter sadly can only give an overview as it would be too exhausting to delve into all the options and possibilities.
  - ZFS has a pretty good CIFS implementation that also is known to work well in an Active Directory environment, being able to store NTFS ACLs and SIDs without having to rely on user or permission mappings. The 'sharesmb' command controls the service while 'zfs set sharesmb' creates CIFS shares. Consider the manpage of 'sharemgr' too.
  - As the UNIX' weapon of choice however we will concentrate on NFS in this workshop.

Turn on NFS sharing on a ZFS:

```bash
~# zfs set sharenfs=on <pool>/<zfs>
~# zfs get sharenfs <pool>
```

This is great for auto-exporting home directories of users, as it does not allow root access to the share. Hook it into your 'useradd'-script, put some quota on it and you are ready to go!
To turn on root access to a share you have to explicitly allow it.

```
~# zfs set sharenfs="sec=sys,root=@host1/32,rw=@host1/32" <pool>/<zfs>
```

That pretty much sums it up. For a more complex setup one could rely on 'netgroups' stored in LDAP or NIS or on NFSv4.

**Step 10 - Use ZFS features for rapid server deployment**

* In this final step we use what we have learned about ZFS to create a template based, PXE-booted server.

* Before you start please examine the following setup:
• One server is prepared for each team, so please work together on this task.
• Please follow the task description exactly, otherwise the preconfigured DNS, DHCP and TFTP settings will not work and you will not be able to access the server via the Internet.

Teams please proceed to your corresponding sub page:
- Team 1
- Team 2
- Team 3

Step 10 - Team 1

• First, clean up the existing zpool and start over.
  
  ```
  $ zpool destroy <pool>
  ```

• Create a new pool and a template ZFS.
  
  ```
  $ zpool create testpool1 raidz c2t0d0 c2t1d0
  $ zfs create testpool1/template
  ```

• Untar the templateserver and take a snapshot of it
  
  ```
  $ tar xjpf /opt/workshop/lucid_template.tar.bz2 -C /testpool1/template; sync
  $ zfs snapshot testpool1/template@$(date +%Y-%m-%d_%H:%M)
  $ zfs list -t snapshot
  ```

• Create a new clone of the template. This will be the server we boot in a minute!
  
  ```
  $ zfs clone testpool1/template@<date> testpool1/zfstest1
  $ zfs list -r testpool1
  ```

• Modify the templateserver so we can identify it later
  
  ```
  $ vim /testpool1/zfstest1/etc/hostname
  ```

• Export the clone ZFS via NFS so the kernel can mount the rootfs at boot time. Also set a quota for the server.
  
  ```
  $ zfs set sharenfs="sec=sys,root@10.1.51.101/32,rw@10.1.51.101/32" testpool1/zfstest1
  $ zfs set quota=10g testpool1/zfstest1
  ```
Please give notice that you have finished preparing the server and the VM should be powered on.

- **Wait for your server to finish booting and log on to it:**

  ```
  # ping zfstest1
  # ssh root@zfstest1
  ```

- **Congratulations! Welcome to your new server. Now install some stuff, watch the clone grow on the ZFS server etc. To finish the course install a webserver and an app (whatever you like...) and access it via:**

  http://zfstest1.dest-unreachable.net:8080

- **Notice that you can access the `.zfs` snapshot folder even from the server.**

  ```
  # cd /.zfs/snapshot
  ```

That’s it! Thank you for your time and patience, hope you liked it!