klara

ZFS Powered Magic Upgrades

Using boot environments for atomic in-place upgrades

Summary & Introductions

1

Allan Jude
FreeBSD Core Team
OpenZFS Developer

2

Klara Inc.
FreeBSD Professional Services and Support

klara

Covered in this presentation

- What is a boot environment? How do they work?
- Creating and managing boot environments
- Deploying boot environments remotely
- Doing it even better next time

What we had before: NanoBSD

- Divides the disk into 2 partitions (firmware images)
- Install the stock image to both
- At upgrade time, overwrite the inactive image
- Boot-once to the newer image. If it fails, or is otherwise unserviceable, reboot to good image
- If the new image is accepted, configure it as the new default
- Repeat process for next upgrade

ZFS Boot Environments

- ZFS takes this concept further, allows more flexibility
- You can have many filesystems, no need to partition your disk
- Separate the OS (root FS) from user data (home dirs, logs, etc)
- ZFS has instantaneous snapshots and clones
- Snapshot and clone the root filesystem before you make changes or upgrade
- Keep every “working” system image you have ever had

klara

How does it work?

- Now you have multiple different ‘versions’ of your root filesystem to choose from, or revert back to
- Modern FreeBSD boot loader allows you to choose from the different root filesystems at boot with a nice menu
- Now you can ‘revert’ an upgrade without losing changes to home directories, logs, databases or other filesystems, further separating the ‘OS’ from the ‘Data’

Regaining Control

- The flexibility of ZFS puts you in control
- Any files in the filesystem mounted as / are treated as part of the operating system, kept separate from ‘data’
- Any files in other filesystems, are retained, no matter what ‘version’ of the OS you choose to boot from
- Packages (/usr/local) and the pkg database (/var/db/pkg) are included in /. This allows you to ‘undo’ a pkg upgrade too

```
NAME                    USED        REFER   MOUNTPOINT
zroot                    19.5G        88K   /zroot
zroot/ROOT               1.67G        88K   none
zroot/ROOT/default       1.67G      1.67G   /
zroot/tmp                  88K        88K   /tmp
zroot/usr                12.3G        88K   /usr
zroot/usr/obj            12.3G      8.03G   /usr/obj
zroot/usr/home          140M       140M   /usr/home
zroot/var                 153M        88K   /var
zroot/var/audit            88K        88K   /var/audit
zroot/var/crash         152M       152M   /var/crash
zroot/var/log           352K       352K   /var/log
zroot/var/mail          132K       132K   /var/mail
zroot/var/tmp              88K        88K   /var/tmp
```

Going Further

- When upgrading a system, we wanted to replace the entire OS with a newer version, or merging, no mistakes
- So we just install a new boot environment
- But what about /etc? My machine needs to have a configured network for puppet to replace the rest of the configuration
- Let's make /etc its own filesystem, it can persist through the upgrade this way...

klara

What Could Possibly Go Wrong?

Not So Fast...

- Lots of boot things depend on /etc being there
- No /etc/fstab, no /etc/rc, no /etc/rc.conf, no /etc/ttys
- Don’t want to have to run etcupdate or mergemaster
- Steal from NanoBSD? A read-only /etc recreated from /cfg
- Then learned about loader.conf variable: init_script see loader(8)
- Use init_script to mount /cfg very early during boot
- Replace persistent files in /etc with symlinks to /cfg

klara

What is this init_script?

```
mount -p | while read _dev _mp _type _rest;
do
    [ $_mp = "/" ] || continue
    if [ $_type = "zfs" ] ; then
        pool=${_dev%%/*}
        zfs mount ${pool}/cfg
    fi
    break
done
```

How does that work?

- /cfg populated with ~10 files that persist thru upgrade
- Configure network (rc.conf.*), sysctls, SSHd keys, fstab, etc
- Rest of /etc can be replaced with stock files
- Never have to merge /etc/rc.d files again
- Never get <<< === >>> marks in .conf files again
- Originally manually recreated symlinks over stock installs
- Used a VM and a script to make new BEs to ship

How do you deploy a BE?

- Create an image:
  - zfs snapshot img/ROOT/be@snap
  - zfs send -pec img/ROOT/be@snap | xz -9 > bename.zfs.xz
- Apply the image:
  - fetch -o - https://svr/bename.zfs.xz | unxz | zfs recv zroot/ROOT/newbe
- Boot Once:
  - zfsbootcfg zfs:zroot/ROOT/newbe:

Shortcomings

- We were still doing pkg upgrade -f in a chroot for the base system boot environment plus each jail
- Building images was painfully manual
- Missing a step or file almost every time
- Bootstrapping a fresh install was still a bunch of manual work, over slow IPMI
- Not usable by anyone else, too many sharp edges

Using BEs at Scale

- Over 100 servers, 38 DCs, 11 countries
- Only myself and 1 full time sysadmin
- Mix of versions, 11.1, 11.2, 12.0, 13-CURRENT
- freebsd-update upgrade too manual
- zfs recv; zfsbootcfg; reboot takes less than a minute
- Failure is gracefully, cycle power and returns to working config
- Remote upgrades with confidence, without console access

Not Just for Packages Anymore

- Poudriere is used to build official FreeBSD binary packages
- Uses Jails, and optionally ZFS and TMPFS for performance
- Starts 1 jail per core, builds one package in each jail, only dependencies installed (clean env), no network connection
- Ensures you don’t introduce undeclared dependancies
- You can use it to build your own customized package
    - ports tree * freebsd version * arch * set

klara

A Better Way to Build

- During the development of this upgrade procedure, I happened to be talking with Baptiste Daroussin (bapt@) who informed me of his previous work on ‘poudriere image’
- Designed to create customized VM or USB images. Used at Gandi to build FreeBSD images for their Public Cloud
- Supports overlays and preinstalled packages
- Targets: iso, iso+(z)mfs, usb, usb+(z)mfs, rawdisk, zfsrawdisk, tar, firmware, embedded

klara

Poudriere Image ZFS BE Support

- After discussion it was decided that zfs send should be added as an output format to ‘poudriere image’
- New targets: zfssend (full pool) and zfssend+be (just the BE)
- Modified overlay support to handle symlinks better
- Added support for a ‘ZFS Layout’ config file, in the same format used by bsdinstall, to datasets to create
- Control what files are part of the Boot Environment
- You can spin out /usr/local on its own filesystem if you wish

New Problem: Fresh Installs

- Previously, we used IPMI Remote Media feature to run bootonly.iso on each machine using ‘bsdinstall’
- No PXEBOOT with only 1-3 servers per DC
- Now we make our own iso+mfs image
- Prompts for some config details (no DHCP)
- Partition disks and create an empty pool
- Then zfs recv a full pool image on to it

Poudriere Image for Everyone

- Many recent enhancements upstreamed
- Work-in-Progress at https://github.com/allanjude/poudriere
- Use it to create your own custom images
- Build from poudriere jails you already have to build packages
- Or create from releases without having to compile!
- New Image Formats? vmdk, qcow2, vhd, MBR (CSM & EFI), GPT (CSM, EFI, both), <yours>

Enhancing Poudriere Image

- Needs better naming for image types
- Should support many more combinations
- Use it to replace tools/boot/rootgen.sh
- Should integrate various ‘Cloudware’
- Replicate features of ‘release’ building bits
- Support for post-build scripts (chroot)
- More appliance building features - (talk to me if interested)
- What features do you need?

klara

QUESTIONS

???

klarasystems.com

klara

More Resources

- Want to know more about ZFS?
  - “FreeBSD Mastery: ZFS”
  - “FreeBSD Mastery: Advanced ZFS”
  - Not just for FreeBSD
  - DRM-Free ebooks ZFSBook.com
- BSDNow.tv - Weekly video podcast on BSD & ZFS
  - More questions? feedback@bsdnow.tv
- @allanjude on twitter