Building servers with NanoBSD, ZFS and jails

Paul Schenkeveld <bsdcon@psconsult.nl>
PSconsult ICT Services BV, The Netherlands
$Id: paper.me,v 1.2 2010/02/08 06:42:56 paul Exp paul $

Disclaimer

There are more ways to configure a server than there are system administrators around. This paper does not pretend to provide the best solution for every situation, it just describes the way the author currently builds and maintains FreeBSD servers.

Introduction

System administrators want to keep their servers up to date. When upgrading the operating system or installed applications, the server will be unavailable for the users for some time. If all software maintenance can be scheduled after office hours when nobody uses your server, you are lucky but if your servers are in use 24/7 or maintenance cannot be scheduled after office hours for some reason, you want to minimize system downtime.

If, after upgrading a server, unexpected problems arise and you are forced to rollback the last upgrade, your problem is even bigger, users are faced with another downtime which is often longer than the downtime needed for the upgrade because rollbacks are usually not as straight-forward as upgrades.

FreeBSD offers software upgrades in source and binary forms. The advantage of binary upgrades, using freebsd-update for the base system and a tool like portupgrade or portmaster for installed packages has the advantage of quick upgrades but is not as versatile as upgrading from source because you have no access to compile time options with pre-compiled binaries. Upgrades from software, using `make buildworld`, `installworld`, `buildkernel` and `installkernel` for the base system and a tool for upgrading ports from source such as `portupgrade` or `portmaster` gives you maximum control over compile time options but makes upgrading a lenghty process during which your system is not or only partially available for users.

Administering over a hundred UNIX computers, the author got inspired by the way appliances and embedded systems can be easily administered using NanoBSD, a toolkit that comes with the FreeBSD base system. Key features of NanoBSD are the offline build of complete system images, dual system images that allow for easy rollback to the previous software version if an upgrade is not satisfactory and the separate storage of all modified configuration files in a filesystem which is shared between the two system images. The result is a method of building servers that can be upgraded with minimal downtime, and that can easily be rolled back to the previous state if needed.

By isolating applications in jails, a lightweight form of server virtualisation, applications and the operating system image they live in can be upgraded seperately. The jails these applications run in are in fact snapshots of one or more prototype jails. In these prototype jails we can upgrade the (userland part of) the operating system and install and upgrade ports. Upgrading a production jail only involves stopping the jail and starting it again with a different snapshot of the prototype, and takes only a matter of seconds. Roll-back in case the new snapshot causes unexpected problems is as easy as upgrading, re-start the jail with the previous snapshot version.

Although the whole system could be built using UFS filesystems, here we will use ZFS as it makes storage management and snapshots so much easier.

1. Server overview

This paper describes a way to set up a FreeBSD server that can be upgraded with minimal downtime. Rolling back unsuccesful upgrades cause the same minimal downtime as upgrades.

To ease software maintenance, all user-visible applications will run in jails. Underneath these jails we run a minimal FreeBSD operating system built using NanoBSD. It holds the kernel, some low level services and the development tools for building new system images. The NanoBSD system image can be put at the front of one (or more) of the disk drives but the author prefers to put the system on a flash drive so that failing disk drives do not challenge the startup of the server or the task of the system administrator. NanoBSD was specifically designed to run on flash drives and uses the flash drive in read-only mode (except when saving changes to configuration files).

1.1. NanoBSD

The `nanobsd.sh` script, which comes standard with the FreeBSD source distribution, was originally written by Poul Henning Kamp to easily make compact flash images for embedded systems. Two features of NanoBSD make it a good basis for our servers too. NanoBSD builds a complete system image off-line and NanoBSD (by default) divides your (flash) disk in two slices to hold two different copies of your system image so when running from one image the other can be upgraded without affecting users.

NanoBSD uses a `md(4)` memory disk for `/etc` and `/var`. Changes to `/etc` and `/usr/local/etc` (which is symlinked to `/etc/local`) must be copied to a separate configuration partition (mountable under `/cfg`). The script `/root/save_cfg` that comes with NanonBSD can be used for this purpose, a more versatile script called `cfgsync` written by yours truly can be found here[1]

So if `/var` is a memory disk and NanoBSD uses the (flash) disk in read-only mode, what about logfiles? Well, it is possible to put `/var` on a harddisk instead of using a memory disk (by including a line for `/var` in `/etc/fstab`) but the author prefers to make the NanoBSD image completely independent of the harddisks. So `syslog` can be told to log to a syslog daemon on another host or in one of the virtual hosts inside one of the jails. To not lose boot messages generated before the syslog daemon in the jail is up, `msyslog` (found in ports) allows to log over TCP and keep log messages in a memory buffer until the TCP connection to the syslog daemon in the jail (also `msyslog`) can be established.

1.2. Jails

Since version 4, FreeBSD offers jails, a facility that allows for light-weight virtual servers, see `jail(2)` and `jail(8)`. A process and all of its descendants are restricted to a chrooted directory tree, and a one or more IP adresses. Since jails do not really virtualize the hardware or the operating system, the performance penalty of using jails can be neglected.

Jails can be started and stopped automatically by `/etc/rc.d/jail` as configured in `/etc/rc.conf`. Two specific features that we use here are the automatic mounting and unmounting of filesystems when starting or stopping a jail and pre-start and pre-stop script execution. We use these features to set up the directory tree for each jail by combining a read-only snapshot of the prototype jail with read-write filesystems for the application(s) that run inside the jail and for creating and initializing a memory filesystem for the `/etc` directory inside the jail.

1.3. ZFS

Since version 7, FreeBSD offers the ZFS filesystem ported from Suns OpenSolaris operating system by Pawel Jakub Dawidek. It offers too many new features compared to the UFS filesystem to describe here. The main features we need to buid this server are support for large number of filesystems (ZFS calls them datasets), easy to use low-overhead snapshots and the automatic mounting of filesystems during boot without using `/etc/fstab`. The latter is important as `/etc/fstab` needs to be part of the read-only root filesystem of out NanoBSD image and we do not want to be forced to create a new image every time we want to add a new filesystem.

The initial port of ZFS to FreeBSD has stability problems, later versions (imported to FreeBSD 8 before 8.0 RELEASE and to FreeBSD 7 just after 7.2 RELEASE) seem very stable. The author recommends using a recent FreeBSD 8-STABLE.

2. How it all works

Combining these three technologies, we get the following system. In all examples below the ZFS storage pool will be called `zpool`, the prototype jail will be called `proto` and the production jail will be called `vhost1`.

2.1. Filesystem layout

A typical server will have the following global filesystems:

Filesystem	Mountpoint	Comments
/dev/ad0s1a	/	R/O NanoBSD image, including /usr and /usr/local.
devfs	/dev	
/dev/md0	/etc	Malloc backed memory disk.
/dev/md1	/var	Malloc backed memory disk.
zpool/FreeBSD	/FreeBSD	Contains CVS repository, source and ports dirs, ports
		distribution files and backup of NanoBSD images.
zpool/jail	/jail	Toplevel for all jails.
zpool/src	/usr/src	FreeBSD source tree, for building NanoBSD
		images and the initial prototype jail.
zpool/obj	/usr/obj	Object trees during initial build of the first prototype
		jail and when building new NanoBSD images.

The filesystem layout for the prototype jail looks like this:

Filesystem	Mountpoint	Comments
zpool/proto	/jail/proto	Jailed root, /usr and /usr/local
zpool/proto	/jail/proto/etc	Jailed /etc with /etc/local which is relocated
		from /usr/local/etc using a symlink.
zpool/proto_var	/jail/proto/var	Jailed /var.
zpool/proto_src	/jail/proto/usr/src	FreeBSD source
zpool/proto_ports	/jail/proto/usr/ports	FreeBSD port skeletons
zpool/proto_obj	/jail/proto/usr/obj	Used to hold object tree when upgrading the jail
localhost:/FreeBSD	/jail/proto/FreeBSD	R/W NFS mount to get CVS repository,
		ports and port distribution files.

Note that `/jail/proto` and `/jail/proto/etc` must always be mounted, otherwise it would not be possible to mount their snapshots for production jails. We use NFS instead of nullfs to mount `/FreeBSD` under `/jail/proto/FreeBSD` because we need to be able to write to de ports distribution files cache and nullfs is only stable for r/o mounts.

The filesystem layout for our `vhost1` looks like:

Filesystem	Mountpoint	Comments
zpool/vhost1	-	Toplevel filesystem from which descendants inherit
		properties and for doing recursive snapshots and backups.
/jail/proto/.zfs/snapshot/1	/jail/vhost1	R/O nullfs mount of a snapshot of the prototype root.
/jail/proto/etc/.zfs/snapshot/1	/jail/vhost1/conf/base/etc	R/O nullfs mount of a snapshot of the prototype /etc.
zpool/vhost1/cfg	/jail/vhost1/cfg	Persistent configuration files copied from /etc.
zpool/vhost1/var	/jail/vhost1/var	Jailed /var.
zpool/vhost1/home	/jail/vhost1/home	Jailed /home containing user data.
devfs	/jail/vhost1/dev	
/dev/md2	/jail/vhost1/etc	Malloc backed memory disk.

The filesystem `/jail/vhost1/etc` is created and populated during jail startup and destroyed during jail shutdown. This is done by two scripts, `/etc/jail_etc_init` and `/etc/jail_etc_cleanup` which can be found at [1].

They get invoked by `/etc/rc.d/jail` because of the following two lines in `/etc/rc.conf`:

```
jail_vhost1_exec_prestart0="/etc/jail_etc_init vhost1"
jail_vhost1_exec_prestop0="/etc/jail_etc_cleanup vhost1"
```

`/etc/jail_etc_init` first mounts a swap based memory disk at `/jail/vhost1/etc`. Then it copies over all original configuration files from `/jail/vhost1/conf/base/etc`, the snapshot of `/jail/proto/etc`. Finally it copies all saved, altered configuration files from `/jail/vhost1/cfg`. See how it resembles the way NanoBSD handles configuration files.

The script `/etc/jail_etc_cleanup` unmounts and destroys the memory disk when the jail is shut down. If you have downloaded `cfgsync`[1] and installed it in `/etc/admin`, all changes to `/jail/vhost1/etc` will be saved to `/jail/vhost1/cfg` automatically.

2.2. Automatic jail startup and shutdown

The prototype jail and this jail get automatically started by `/etc/rc` when the following lines are present in `/etc`:

```
 1  jail_enable="YES"
 2  jail_list="proto vhost1"

 4  jail_proto_rootdir="/jail/proto"
 5  jail_proto_hostname="proto.domain.local"
 6  jail_proto_ip="192.168.0.10"              # needed for fetch()
 7  jail_proto_interface="igb0"
 8  jail_proto_devfs_enable="YES"
 9  jail_proto_mount_enable="YES"
10  jail_proto_fstab="/etc/fstab.proto"

12  jail_vhost1_rootdir="/jail/vhost1"
13  jail_vhost1_hostname="vhost1.domain.local"
14  jail_vhost1_ip="192.168.0.11"
15  jail_vhost1_interface="igb0"
16  jail_vhost1_devfs_enable="YES"
17  jail_vhost1_mount_enable="YES"
18  jail_vhost1_fstab="/etc/fstab.vhost1"
19  jail_vhost1_exec_prestart0="/etc/admin/jail_etc_init vhost1"
20  jail_vhost1_exec_prestop0="/etc/admin/jail_etc_cleanup vhost1"
```

Lines 1 and 2 tell FreeBSD to automatically start up jails and which jails to start. `proto` is the name of our prototype server, where we perform software installation and upgrades, `vhost1` is name of the jail running our application.

Lines 4 to 10 configure our prototype jail, it will be chrooted in `/jail/proto` and needs a hostname and also an IP address because building ports requires network access for fetching the source files. This IP address will be aliased on interface `igb0` when the jail starts and the address will also be removed when the jail gets stopped. Also, before starting up the prototype jail, all filesystems from `/etc/fstab.proto` will be mounted and a `/dev` will be mounted under the jail root.

Lines 11 to 20 configure the application jail similar to the prototype jail but with the addition of two scripts to take care of setting up the `etc` directory under the jail root by combining the `/etc` from the prototype (unconfigured jail) and merging in all locally made changes from `/cfg`.

The file `/etc/fstab.proto` would look like this:

```
localhost:/FreeBSD /jail/proto/FreeBSD nfs rw 0 0
```

Note that only one NFS filesystem needs to be mounted through this fstab file when starting the prototype jail, `/dev` is mounted by the `rc.conf` line `jail_proto_devfs_enable="YES"` and all other filesystems were automatically mounted when ZFS was started.

The file /etc/fstab.vhost1 would look like this:

```
/jail/proto/.zfs/snapshot/1 /jail/vhost1 nullfs ro 0 0
/jail/proto/etc/.zfs/snapshot/1 /jail/vhost1/conf/base/etc nullfs ro 0 0
zpool/vhost1/cfg /jail/vhost1/cfg zfs rw 0 0
zpool/vhost1/var /jail/vhost1/var zfs rw 0 0
zpool/vhost1/home /jail/vhost1/home zfs rw 0 0
```

2.3. Manually starting and restarting application jails

Jails listed in jail_list in /etc/rc.conf get started automatically when FreeBSD enters multi-user mode. This is done using the /etc/rc.d/jail script.

To manually start, stop or restart jails use one of these commands:

```
# /etc/rc.d/jail start vhost1
# /etc/rc.d/jail stop vhost1
# /etc/rc.d/jail restart vhost1
```

2.4. Upgrading (or downgrading) a production jail

After upgrading or installing additional software under /jail/proto
we take a new snapshot:

```
# zfs snapshot -r zpool/proto@2
```

Next, we edit /etc/fstab.vhost1 and change the two references to snapshot
1 into references to snapshot 2:

```
/jail/proto/.zfs/snapshot/2 /jail/vhost1 nullfs ro 0 0
/jail/proto/etc/.zfs/snapshot/2 /jail/vhost1/conf/base/etc nullfs ro 0 0
/jail/proto/etc/.zfs/snapshot/2 /jail/vhost1/conf/base/etc nullfs ro 0 0
```

Now when the service window has arrived and all users have left the application in jail vhost1, we just restart it:

```
# /etc/rc.d/jail restart vhost1
```

and we are done, usually within seconds.

A downgrade, in case unforseen problems arise from the upgrade is as simple too, change /etc/fstab.vhost1 back and do another restart.

If you want to test out the upgraded prototype before exposing users, just create an extra jail. Now make ZFS clones of all filesystems under /jail/vhost1. Create the new fstab for the second jail using the new snapshot of /jail/proto plus the clones of the other filesystems and start this extra jail and do your testing.

2.5. Upgrading (or downgrading) NanoBSD

NanoBSD images can be built off-line or on this system itself. Because the nanobsd.sh script needs to create a memory disk and mount it in order to create a complete filesystem image, we do not build NanoBSD in a jail (mounts are not allowed there) but directly under /FreeBSD.

Assuming we are running from the first FreeBSD slice of our flash disk, use the following command to install newly created NanoBSD image into the second slice and tell the boot loader use the second slice next time we boot the system, execute the next command:

```
# sh /root/upgradep2 < _.disk.image
```

Then, when the system is quiet and users have left, reboot the server and we are done. Upgrading slice 1 if we are running from slice two is done by the script updatep1.

Switching back from the new image in slice 2 to the previous version in slice 1 is also very simple:

```
# boot0cfg -v -s 1 ad0
# reboot
```

3. Building the server

Building this server from scratch involves the following steps:

- Creating a NanoBSD image and putting it on the boot device
- Basic system configuration
- Setting up ZFS
- Setting up the development system
- Creating the protoype jail and taking the first snapshot
- Creating the production jails

We'll now look at these steps in turn.

3.1. Creating a NanoBSD image and putting it on the boot device

Here we have a typical chicken-and-egg situation. To build the initial NanoBSD image for your system you need a FreeBSD system with a complete FreeBSD source tree, once your server is running it can build its own NanoBSD images for upgrading. If you don't have a system at hand already, you can grab a FreeBSD installation ISO image from `www.freebsd.org` and install it on a PC or inside a virtual system like VMware, VirtualBox, Parallels or Qemu.

The next two things you need are a NanoBSD configuration file that describes the image to build (we will call it `nanoserver.conf` from now) and a kernel configuration file (which we will call `NANOSERVER`). Example configuration files can be found at [1].

Although the NanoBSD script and documentation suggest that it is only available for the `i386` architecture, it works perfectly well for `amd64` architecture too which is almost a requirement for modern servers and to get a stable ZFS. To build for `amd64`, you need to set `NANO_ARCH=amd64` and have the following function defined in `nanoserver.conf`:

```
create_amd64_diskimage() {            # to allow NANO_ARCH=amd64
    create_i386_diskimage
}
```

The file `nanoserver.conf` assumes that we are building for `amd64` architecture. It also assumes that the image will be put on a SanDisk compact flash of 1GB, adjust this to your situation. For easy development and testing a USB flash drive can be very handy here.

Now you can build the image by executing:

```
sh /usr/src/tools/tools/nanobsd/nanobsd.sh -c nanoserver.conf
```

If all is well, it will produce two image files, `_.disk.full` and `_.disk.image` in the directory `/usr/obj/nanobsd.NANOSERVER` (assuming `NANO_NAME` is set to `NANOSERVER` as in the example files).

The file `_.disk.image` is intended for upgrades, `_.disk.full` is a complete system image ready to be put on a flash disk. Write this to your flash disk using `dd(1)`:

```
dd < _.disk.full > /dev/<flashdevice> bs=64k
```

If all is well, your server can now boot from NanoBSD after installing the flash disk.

3.2. Basic system configuration

As soon as your server is running NanoBSD (congratulations if this is your first NanoBSD installation!) you can do your normal system configuration (root password, timezone, `/etc/rc.conf`, `/etc/resolv.conf`) but do not forget to mount `/cfg` and copy all changed files from `/etc` to `/cfg`. You can use save_cfg and other scripts in `/root` to help you or use the cfgsync script available at [1].

3.3. Setting up ZFS

First add the following line to `/etc/rc.conf` (and copy to `/cfg`!)

```
zfs_enable="YES"
```

Now start ZFS:

```
# /etc/rc.d/zfs start
# /etc/rc.d/zvol start
```

The next step is to create a ZFS storage pool and some global filesystems. Assume we have three SATA disks, `/dev/ad4`, `/dev/ad6` and `/dev/ad8` and want basic redundancy. We now create the storage pool with the command:

```
# zpool create zpool raidz ad4 ad6 ad8
```

First we create a swap device:

```
# zfs create -o org.freebsd:swap=on -V 8G zpool/swap
```

Tell FreeBSD to start using this new swap space:

```
# swapon /dev/zvol/
```

Now we create some global filesystems that will always stay mounted:

```
# zfs create -o mountpoint=/usr/src zpool/src
# zfs create -o mountpoint=/usr/obj zpool/obj
# zfs create -o mountpoint=/FreeBSD zpool/FreeBSD
# zfs create -o mountpoint=/jail zpool/jail
```

`/usr/src` and `/usr/obj` are needed for building NanoBSD images, `/FreeBSD` is where we will keep a local mirror of the FreeBSD CVS repository, files needed by `csup(1)` and our NanoBSD configuration files. `/jail` is the directory under which all jails will be created.

For our prototype jail we need some more filesystems. More than one prototype jail can be created if you don't want all production jails to be the same. The entire prototype that will be exported to other jails will be in two filesystems:

```
# zfs create -o mountpoint=/jail/proto zpool/proto
# zfs create zpool/proto/etc
```

Note that `zpool/proto/etc` is a child of `zpool/proto`, this way we can recursively create a snapshot covering the complete system.

The prototype has a private `/var` which is not a child of its toplevel filesystem (hence `zpool/proto_var` and not `zpool/proto/var`). It will not be exported to production jails, they have a `/var` of their own:

```
# zfs create -o mountpoint=/jail/proto/var zpool/proto_var
```

Like `/var`, the prototype jail will also have a private `/usr/obj` filesystem which is not exported to production jails:

```
# zfs create -o mountpoint=/jail/proto/usr/src zpool/proto_src
# zfs create -o mountpoint=/jail/proto/usr/ports zpool/proto_ports
# zfs create -o mountpoint=/jail/proto/usr/obj zpool/proto_obj
```

Because we also need the contents of `/FreeBSD` under our prototype jail and need both read and write access, we set up NFS. Add the following lines to `/etc/rc.conf` (and save to `/cfg`!):

```
rpcbind_enable="YES"
mountd_enable="YES"
nfs_server_enable="YES"
```

Make `/FreeBSD` NFS exported:

```
# zfs set sharenfs="-maproot=0 localhost" zpool/FreeBSD
```

We will use the FreeBSD automatic jail startup to get it mounted. Create the file `/etc/fstab.proto` (and save it to `/cfg`!) with this line in it:

```
localhost:/FreeBSD /jail/proto/FreeBSD nfs rw 0 0
```

In /etc/rc.conf add:

```
jail_enable="YES"
jail_list="proto"

jail_proto_rootdir="/jail/proto"
jail_proto_hostname="proto.domain.local"
jail_proto_ip="192.168.0.10"
jail_proto_devfs_enable="YES"
jail_proto_mount_enable="YES"
jail_proto_fstab="/etc/fstab.proto"
```

Be sure to set a valid ip address in `jail_proto_ip`, you need it when building ports inside the prototype jail. To get NFS running and `/jail/proto/FreeBSD` now execute the following:

```
# /etc/rc.d/rpcbind start
# /etc/rc.d/mountd start
# /etc/rc.d/nfsd start
# /etc/rc.d/jail start proto
```

We now have our prototype jail in place, let's put FreeBSD into it.

3.4. Setting up the development system

To build new NanoBSD images and the FreeBSD userland for our jails, we need sources. Although it's easy to get a source tree from a FreeBSD installation CD or DVD, having a local mirror if the FreeBSD CVS repository is has the advantage of having all FreeBSD releases and branches to choose from. The repository can be created an updated regularly using `csup(1)`. To set up your local CVS mirror do the following:

```
# mkdir /FreeBSD/cvs /FreeBSD/sup
# cat << ! > /FreeBSD/sup/supfile
*default host=cvsup.FreeBSD.org
*default prefix=/FreeBSD/cvs
*default base=/FreeBSD
*default release=cvs delete use-rel-suffix compress
cvs-all
!
```

Instead of `cvsup.FreeBSD.org`, you can use `cvsup.<country>.freebsd.org` to connect to a server close by, `<country>` is the iso3166 two-letter abbreviation of your country e.g. `jp` for Japan.

Now use `csup` to get your initial repository, this can take several hours depending on your Internet connection and how busy the cvsup server is:

```
# csup -z /FreeBSD/sup/supfile
```

The same command can be user on a regular base to efficiently synchronize the newest commits. When your repository is complete, check out the desired version of the sources:

```
# cd /usr
# cvs -Q -R -d /FreeBSD/cvs co -r RELENG_8 src
```

Your NanoBSD configuration files can be kept in `/FreeBSD/NanoBSD`:

```
# mkdir /FreeBSD/NanoBSD
```

3.5. Creating the protoype jail and taking the first snapshot

Before compiling the first incarnation of your prototype jail, create `/etc/make.conf` and `/etc/src.conf` and tune them to your linking. Many subsystems like `BOOT`, `I4B`, `IPFILTER`, `PF` to name a few are normally not needed inside a jail. By excluding them from the build, you can save yourself some time waiting for make buildworld to finish. Don't forget to copy these files to `/cfg`!

Now build your system:

```
# cd /usr/src
# make -j 3 buildworld
```

Vary `-j` (number of parallel make jobs) depending on CPU resources that are available for your build.

When the system is built, install it in the prototype jail with:

```
# make installworld DESTDIR=/jail/proto
# make distribution DESTDIR=/jail/proto
```

Now we relocate the directory `/usr/local/etc` to `/etc/local` so that all configuration files installed by ports will be under `/etc` and can be made private per jail using `/cfg`:

```
# mkdir -p /jail/proto/usr/local/etc
# mv /jail/proto/usr/local/etc /jail/proto/etc/local
# ln -s ../../etc/local /jail/proto/usr/local/etc
```

Note that this is the same construction as the `/usr/local/etc` relocation of your NanoBSD image!

Add a minimal `/etc/rc.conf` containing:

```
hostname="proto.domain.local"
```

create additional directories needed in your production jails:

```
# mkdir /jail/proto/home
# mkdir /jail/proto/var/db/mysql
...
```

and copy `/etc/resolv.conf`, `/etc/localtime`, `/etc/make.conf` and `/etc/src.conf` from your NanoBSD system:

```
# cp -p /etc/resolv.conf /jail/proto/etc
# cp -p /etc/localtime /jail/proto/etc
# cp -p /etc/make.conf /jail/proto/etc
# cp -p /etc/src.conf /jail/proto/etc
```

These file will automatically end up in every jail unless overridden with private copies.

Make any changes to configuration files that you want all jails to share, a root password or `"*"` to deny root logins is recommended.

Finally, create the first snapshot of the prototype jail:

```
# zfs snapshot -r zpool/proto@1
```

3.6. Creating the production jails

To create a production jail named vhost1 do the following. First create private filesystems and a `fstab` file:

```
# zfs create -o legacy zpool/vhost1
# zfs create -o legacy zpool/vhost1/cfg
# zfs create -o legacy zpool/vhost1/var
# zfs create -o legacy zpool/vhost1/home
# cat << ! > /etc/fstab.vhost1
/jail/proto/.zfs/snapshot/1 /jail/vhost1 nullfs ro 0 0
/jail/proto/etc/.zfs/snapshot/1 /jail/vhost1/conf/base/etc nullfs ro 0 0
zpool/vhost1/cfg /jail/vhost1/cfg zfs rw 0 0
zpool/vhost1/var /jail/vhost1/var zfs rw 0 0
zpool/vhost1/home /jail/vhost1/home zfs rw 0 0
!
```

Add the new jail to `/etc/rc.conf`:

```
jail_vhost1_rootdir="/jail/vhost1"
jail_vhost1_hostname="vhost1.domain.local"
jail_vhost1_ip="192.168.0.11"
jail_vhost1_interface="igb0"
jail_vhost1_devfs_enable="YES"
jail_vhost1_mount_enable="YES"
jail_vhost1_fstab="/etc/fstab.vhost1"
jail_vhost1_exec_prestart0="/etc/admin/jail_etc_init vhost1"
jail_vhost1_exec_prestop0="/etc/admin/jail_etc_cleanup vhost1"
```

and add the name of the jail to jail_list in /etc/rc.conf. Again, do not forget to copy the new fstab.vhost1 and rc.conf to /cfg.

Mount all vhost1 filesystems manually:

```
# mount -aF /etc/fstab.vhost1
```

You can now tune the configuration of the new jail by copying configuration files from /jail/vhost1/conf/base/etc to /jail/vhost1/cfg or creating completely new configuration files directly under /jail/vhost1/cfg. Remember that file you expect to be in /usr/local/etc now live in /jail/vhost1/conf/base/etc/local and /jail/vhost1/cfg/local.

When you are done configuring, unmount and start your jail:

```
# mount -aF /etc/fstab.vhost1
# /etc/rc.d/jail start vhost1
```

3.7. Uprading the prototype and installing ports

To upgrade the FreeBSD userland of the prototype jail and install ports, we have to get a shell inside the jail. The command jexec can give us that shell but it needs to know the jail id (jid) of the prototype jail. You can find the jail id by looking at /var/run/jail_proto.id or using the jls command:

```
# jls
   JID  IP Address      Hostname                      Path
     4  192.168.0.10    proto.domain.local            /jail/proto
```

To get a shell inside the prototype jail execute:

```
# jexec 4 /bin/sh
# PS1='proto# '
proto#
```

The first time we have to get source and ports trees under /jail/proto:

```
proto# cd /usr
proto# cvs -Q -R -d /FreeBSD/cvs co -r RELENG_8 src
proto# cvs -Q -R -d /FreeBSD/cvs co ports
```

Later, we can update our source and ports trees (after running csup to synchronize the local CVS mirror) with:

```
proto# cd /usr/src
proto# cvs -Q -R up -d -P
proto# cd /usr/ports
proto# cvs -Q -R up -d -P
```

Upgrading FreeBSD userland is easy now:

```
proto# cd /usr/src
proto# make -j 3 buildworld
proto# make installworld
proto# mergemaster -i
```

To install ports, we'd better set some environment variables and make two directories:

```
proto# export WRKDIRPREFIX=/usr/obj/ports
proto# export DISTDIR=/FreeBSD/distfiles
proto# mkdir /usr/obj/ports /FreeBSD/distfiles
```

These will prevent port build from placing downloaded sources under /usr/ports/distfiles and work directories under all port skeleton directories so /usr/ports remains a clean checkout from the CVS repository.

For upgrading your installed ports, a ports management tool like portupgrade or portmaster is recommended.

4. Advanced topics

As long as you do not forget to copy all changed configuration files to the appropriate /cfg directory, system administration is much the same as on a normal FreeBSD system except for software maintenance.

Here are some miscellaneous tips on administering these servers.

4.1. Maintaining multiple servers

Multiple servers can be built and maintained from a single build server. On all but the first server, omit all filesystems under `/jail/proto` except for `/jail/proto/etc` and `/jail/proto` itself. Use `zfs send` and `zfs receive` to replicate `/jail/proto` and `/jail/proto/etc` from the master server to all slaves:

```
master# zfs send -R zpool/proto@1 > proto.zstream
slave# zfs receive -dF zpool < proto.zstream
```

Later upgrades can be transferred incrementally:

```
master# zfs send -R -I @1 zpool/proto@2 > proto.zstream
slave# zfs receive -dF zpool < proto.zstream
```

Of course the `zfs send` and `zfs receive` can be connected by some network plumbing, i.e. with `nc(1)` or `ssh(1)`.

4.2. NanoBSD logging using msyslog

To save all system logs from your NanoBSD system to the logfiles on `vhost1`, build a NanoBSD image that includes the port `sysutils/msyslog` and install `msyslog` in your prototype jail. Bump the snapshot version of `vhost1` to the version that includes `msyslog`.

Now set up msyslog in NanoBSD. In `/etc/rc.conf`:

```
syslogd_enable="NO"
msyslogd_enable="YES"
```

and in `/usr/local/etc/syslog.conf`:

```
*.*          %tcp -h 192.168.0.11 -p 601 -s 131072
```

(and save to `/cfg`).

Msyslog will use TCP to connect to the syslog server on `vhost1` and buffer up to 128k (`-s` option) of logging until the connection succeeds. The buffering preserves boot messages until `vhost1` gets started.

Set up `msyslog` in jail `vhost1`. In `/jail/vhost1/etc/rc.conf`:

```
syslogd_enable="NO"
msyslogd_enable="YES"
msyslogd_flags="-i 'tcp -p 601 -q'"
```

and configure `/jail/vhost1/usr/local/etc/syslog.conf` the way you like, use `/etc/syslog.conf` as an example.

Did you remember to save the configuration to `/jail/vhost1/cfg`?

4.3. Prevent NanoBSD from sending email

Because NanoBSD has `/var` in memory, it's not a good idea for it to send email. The only email sent from FreeBSD out-of-the-box are the results of daily, weekly, monthly and security periodic runs and reports of lost `vi(1)` buffers after a crash.

Replacing sendmail by a dummy that directly delivers to a real SMTP server without queueing the message (such as `/usr/ports/mail/ssmtp`) can do that trick. Alternatively, `periodic(8)` can be instructed to write logs to plain files instead of emailing or you could turn off periodic runs completely.

4.4. Snapshot and backup strategies

ZFS has very powerful snapshot and backup/restore features (through `zfs send` and `zfs receive`). By having an unused parent directory per jail it is very easy to do recursive snapshots and backups per jail. If you want to make frequent snapshots of your data to protect your users against human error, perform these snapshots recursively on these parent filesystems or directly on their children but be aware not to make frequent snapshots of the prototype jail as these may interfere with the snapshots we use for upgrading jails and replication to other servers. Also, do not make frequent snapshots of `/usr/obj`, `/usr/src`, `/usr/ports` and `/FreeBSD`. If you do, you'll find out soon why this is not such a brilliant idea.

4.5. Relocating files to /etc

Sometimes we'd like to change files that are part of the read-only jail root or keep persistent copies of files under `/var`. Examples are `/root/.ssh/known_hosts` and files under `/var/cron/tabs`. The easiest solution is to create symlinks for these files or directories to `/etc`. This needs to be done when generating the NanoBSD image by defining as customization function in `nanoserver.conf`.

Similarly, we can symlink `/jail/vhost1/root/.ssh` to something like `/etc/root.dot.ssh`. You have to do this from within the prototype jail, then create a new snapshot and restart the production jail:

```
proto# ln -s ../../etc/root.dot.ssh /root/.ssh
proto# mkdir /etc/root.dot.ssh
```

Experiences so far

After installing the first two servers this way during the summer of 2009, the author now has about ten of these servers running at 5 different sites. The system administrator must be well aware of the mostly read-only nature of the operating system and not forget to copy changed configuration files to stable storage (a cron job that warns about unsaved changes from time to time helps a lot). On the other hand, it helps a lot in clearly separating the operating system and ports or packages from locally grown applications, utilities and scripts.

Most applications can easily be run inside a jail, some however need special treatment and then some really don't want to run in a jail at all. Samba for example only works on broadcast interfaces. FreeBSD sets the netmask of an alias address to 255.255.255.255. This makes samba ignore that interface. To succesfully run samba in a jail, the jail must be assigned the primary IP address of every interface that samba must provide services on.

References

Up-to-date information on building these servers, example configuration files and all tools mentioned above can be found at [1].

[1] `http://www.psconsult.nl/talks/AsiaBSDcon2010-Servers`.

You may find the following manual pages relevant to maintaining these servers:

build(7), csup(1), dd(1), jail(8), jexec(8), jls(8), make.conf(5), nanobsd(8), periodic(8), ports(7), rc.conf(5), src.conf(5), zfs(8), zpool(8)