Jail Preparation/Setup

Filesystem

We'll want a dataset to store data that will exist outside the jails. Yay for ZFS (for reasons I'm glossing over...).

```
zfs create -o compress=lz4 -o atime=off -o mountpoint=/usr/local/data zroot/data
```

(Did I have to create `/usr/local/data` before doing the above? I don't recall... but I'm pretty sure no.)

We expect to have a BookStack jail, which has a database.

```
zfs create -o compress=lz4 -o atime=off zroot/data/dbs
```

And we can have a dataset for the BookStack db, specifically.

```
zfs create -o compress=lz4 -o atime=off zroot/data/dbs/bookstack
```

We'll get to this later, but now (later) we can `nullfs`-mount the dataset inside the jail (in its `fstab`) like so:

```
# Device                  Mountpoint                                FStype  Options Dump  Pass#
/usr/local/data/dbs/bookstack /usr/local/bastille/jails/bs_jail/root/var/db/mysql nullfs  rw,late 0     0
```

And in case you jump ahead, you'll also need to know that `mysql` needs to own the directory.

```
cd /usr/local/data/dbs/
chown -R 88:88 bookstack/
```

Speaking of jumping ahead... I had issues when using a newer version of MariaDB a few months later. The MariaDB setup might need more to get it working. It may have to do with changes to Mysql.

Jail Management

It's tempting to manage jails by hand, but I'll leave that exercise to my local server. We'll use this script-based tool instead.

```
pkg install bastille
```

We want the Bastille jails to start up upon system reboot, so we add it to the `rc.conf` file.

```
sysrc bastille_enable=YES
```

And we can hop right into the Bastille configuration that defines the jails' default parameters.

```
vim /usr/local/etc/bastille/bastille.conf
```

The notable changes are:

```
bastille_zfs_enable="YES"
bastille_zfs_zpool="zroot"


bastille_jail_addr="10.101.10.10"  # not sure if this is even used or makes sense
```

Something of note is it uses a particular loopback device that must be created (added to `rc.conf)`.

```
sysrc cloned_interfaces+=lo1
```

```
sysrc ifconfig_lo1_name="bastille0"
```

And since the jails are on a separate loopback network and need to be NAT'd, we probably need this:

```
sysrc gateway_enable="YES"
```

And then the cloned interface can be brought up.

```
service netif cloneup
```

We would then update `pf.conf` accordingly to allow jail traffic if the example we started with didn't already have this.

```
table <jails> persist


{...}
```

```
nat on $ext_if from <jails> to any -> $ext_if_ip


## static port forwarding for sending http/https to [reverse proxy] jail
## rdr pass inet proto tcp from any to any port {80, 443} -> 10.17.89.45
```

> When searching online, you may find that NAT rules end with `-> ($ext_if)`, but that will include all aliases, which will make the NAT-ting most likely not behave as intended. You want to NAT on the external IP (`-> $ext_if_ip`).

Base Jail

(For creating quickly update-able thin jails later)

It's pretty simple to create the base jail. This will download a fresh base install, basically.

```
bastille bootstrap 12.1-RELEASE
```

You'll want to occasionally update this with:

```
bastille bootstrap 12.1-RELEASE update
```

We should now be ready to create jails.

(On a 25G instance, `ZFS list` currently reflects there is 16.8G remaining space.)

Initial Jail Creation

More foreshadowing!

`Usage: bastille create [option] name release ip [interface].`

`Options` - Empty, Thick, VNET (none of these)

`Interface` - vtnet (no!), bastille0 (yes, but implied)

```
bastille create caddy_jail 12.1-RELEASE 10.101.10.100
```

```
bastille create bs_jail 12.1-RELEASE 10.101.10.110
```

```
bastille create bw_jail 12.1-RELEASE 10.101.10.120
```

```
bastille create thelounge_jail 12.1-RELEASE 10.101.10.130
```

```
bastille create website_jail 12.1-RELEASE 10.101.10.140
```

And after creating five thin jails, the remaining space is still 16.8G. Yay, ZFS, again!

Quick quality of life improvement in the jails

Let's create a `.cshrc` for copying into the jails. It's the same as the regular one, but it uses different prompt colors.

```
# $FreeBSD: releng/12.1/bin/csh/dot.cshrc 338374 2018-08-29 16:59:19Z brd $
#
# .cshrc - csh resource script, read at beginning of execution by each shell
#
# see also csh(1), environ(7).
# more examples available at /usr/share/examples/csh/
#

alias h         history 25
alias j         jobs -l
alias la        ls -aF
alias lf        ls -FA
alias ll        ls -lA



# A righteous umask
umask 22

set path = (/sbin /bin /usr/sbin /usr/bin /usr/local/sbin /usr/local/bin $HOME/bin)

setenv  EDITOR  vi
setenv  PAGER   less
setenv  BLOCKSIZE       K

if ($?prompt) then
        # An interactive shell -- set some stuff up
```

```
# colors for prompt (0 for regular and 1 for bold, or use %B%b for bold)
set     red="%{\033[0;31m%}"
set   green="%{\033[0;32m%}"
set  yellow="%{\033[0;33m%}"
set    blue="%{\033[0;34m%}"
set magenta="%{\033[0;35m%}"
set    cyan="%{\033[0;36m%}"
set   white="%{\033[0;37m%}"
set     end="%{\033[0m%}" # This is needed at the end... :(

# prompt vars
set name = "${red}%B%n%b${end}"
set host = "${red}%m${end}"
set dir  = "${cyan}%~${end}"

set prompt = "[${name}@${host}:${dir}]%# "

#set prompt = "%N@%m:%~ %# "
set promptchars = "%#"

set complete = enhance

set filec
set history = 1000
set savehist = (1000 merge)
set autolist = ambiguous
# Use history to aid expansion
set autoexpand
set autorehash
set mail = (/var/mail/$USER)
if ( $?tcsh ) then
        bindkey "^W" backward-delete-word
        bindkey -k up history-search-backward
        bindkey -k down history-search-forward
endif

# Clean up...
unset red green yellow blue magenta cyan white end
unset name host dir
```

```
endif

# color in autocomplete
set color
# color in ls
alias ls        ls -G

# LS colors, made with https://geoff.greer.fm/lscolors/
setenv LSCOLORS         gxfxcxdxbxegedabagacad
```

Then `mv` each jail's `.cshrc` as `.cshrc.orig`, and then `cp` the `.cshrc.jail` as each jails's new `/root/root/.cshrc`. See below for a script to do this quickly and easily.

Other Bits

It may be a good time to reboot the server. You've made several changes to the system, and you'll want to make sure they stuck and are working correctly.

Changes to `/etc/pf.conf` require `pfctl -f /etc/pf.conf`**. Changes to `/etc/rc.conf` require... something. Changing the `.cshrc` requires sourcing it or logging in fresh. The jails need to be started. Rebooting will do all this, including starting the jails.

** Just make sure you at least have already run `pfctl -vnf /etc/pf.conf` to make sure the config works.

Common Initial Jail Setup

The beginning steps are mostly the same across the jails. Before jumping in, if you haven't already, remember to `mv` the jail's `.cshrc` as `.cshrc.orig`, and then `cp` the host's `.cshrc.jail` as the jails's new `/root/root/.cshrc`.

In fact, here's a script (that magically worked perfectly the first time I ran it), that I just saved in `/usr/local/scripts`.

```
#!/bin/sh

# Copies custom .cshrc from /root/.cshrc.jail in place of the
# jail's default .cshrc, and renames the default as .cshrc.orig.
```

```
# Exit script if error (non-zero return code)
set -e

# check for a single arg (the name of the jail)
if [ "$#" -ne 1 ]; then
  echo "Usage: $0 JAIL_NAME" >&2
  exit 1
fi

# Variables to be used
jail_name="$1"
jails_dir="/usr/local/bastille/jails"
jail_dir="${jails_dir}/${jail_name}/root/root"

# check that the directory exists
if [ ! -d "${jail_dir}" ]; then
	echo "Directory ${jail_dir} doesn't exist." >&2
	exit 1
fi

# check that the original .cshrc exists
if [ ! -f "${jail_dir}/.cshrc" ]; then
	echo "File ${jail_dir}/.cshrc doesn't exist." >&2
	exit 1
fi

# check that the custom .cshrc exists
if [ ! -f "/root/.cshrc.jail" ]; then
	echo "Custom .cshrc.jail in /root doesn't exist." >&2
	exit 1
fi

mv ${jail_dir}/.cshrc ${jail_dir}/.cshrc.orig

cp /root/.cshrc.jail ${jail_dir}/.cshrc

# Write to log briefly what happened
echo "Added custom .cshrc to ${jail_name}."

exit 0
```

Don't forget to `chmod +x` it. Then you just run it with `/usr/local/scripts/custom_cshrc.sh <jail_name>`.

Misc

Another initial jail setup task may be to set up the timezone. You can (unlikely, but possible) have weird internet problems if your time is off. The host time being right is the most important, but feel free to check the current date and time with the `date` command. If you need to update things, run `tzsetup` and choose your timezone.

Also, you may update the jail. If you just created or updated your base jail, or if this is a thin jail, then there is actually no reason for this. But if you do need/want to do an update...

> Updates cannot be installed when the system securelevel (`jail.conf` setting) is greater than zero.

So we must first edit `.../jails/$jail/jail.conf` to change `securelevel` from `2` to `0`, then restart the jail.

Then the updating can happen.

```
bastille cmd $jail freebsd-update fetch install
bastille pkg $jail update
bastille pkg $jail upgrade -y
```

Then we edit `.../jails/$jail/jail.conf` again to change `securelevel` from `0` to `2`, then restart the jail again.

And now you have a current, clean slate upon which to build.

Resources

The Bastille docs are great. https://bastillebsd.org/

Revision #3
Created 4 August 2020 07:22:15 by scoob
Updated 24 September 2020 06:23:02 by scoob