

# FreeBSD 12.1 ZFS On DigitalOcean

Basic setup for FreeBSD w/ jails using Bastille in a DigitalOcean droplet, including reverse proxy jail, gitea, bitwarden-rs, a website, and Bookstack.

- [Legwork](#)
  - [Initial Steps](#)
  - [Initial Login](#)
  - [Jail Preparation/Setup](#)
- [Fun Stuff](#)
  - [BookStack Jail](#)
  - [Website Jail](#)
  - [Caddy Jail](#)
- [More Fun Stuff](#)
  - [Bitwarden-rs Jail](#)
  - [Gitea Jail](#)
  - [Website Jail w/ Git Power-up](#)
  - [IRC!Radio by dsc\\_](#)
- [More Legwork - Backups](#)
  - [Backups Overall](#)
  - [BookStack Backup](#)
- [More Legwork - Upgrading Versions](#)
  - [Upgrade From 12.1- to 12.2-RELEASE](#)

# Legwork

# Initial Steps

## Domain

It was high time that I "owned" part of the internet, so I went on NameCheap.com and "bought" two URLs. I went with the WhoisGuard feature, and I set the domain registrations for auto-renewal.

## VPS

I then registered w/ DigitalOcean and provided my credit card. Now I am ready to roll.

I started a new project on DigitalOcean, and created a new droplet. I selected FreeBSD 12.1 w/ ZFS (could also choose 11.3, and could choose UFS or ZFS, but ZFS is great, and I know I'll wish I had it once I get a little more advanced). I went with the \$5/mo plan (1 vCPU, 1 GB RAM, 25GB SSD). I was not provided a root password. Instead, I had to provide an SSH pubkey. All the better.

## VPS Setup

It would be wise to pop into the Networking tab and create a basic firewall for the droplet. I chose to accept `https`, `http`, and `SSH`, all on default ports. FreeBSD doesn't have a firewall running out the box, so this step is probably wise.

## Domain Setup

Shortly after creating the droplet, I brought the newly acquired IP address over to NameCheap and created A records.

- Created an A record for the domain where the host is simply `@`, meaning `mydomain.tld` will work as is.
- Created an A record for the domain where the host is `www`, so [www.mydomain.tld](http://www.mydomain.tld) will also work.
- Left in place, for now, the CNAME record to the NameCheap parking page. (I should have probably deleted right away.)
- Also left in place, for now at least, the URL redirect for the host `@` to [www.mydomain.tld](http://www.mydomain.tld). I need to research this.

## All Done

Well, not so fast. DigitalOcean and NameCheap both have the ability to enable 2FA. Maybe you should get that out of the way before you get too far. I did.

It's almost time to get various jails set up and piped into a reverse proxy (another jail), and then have CNAME records created for pointing subdomains (a subdomain for each jail) at the droplet's IP. You do the CNAME records in the Advanced DNS screen of the NameCheap site.

At this point, there's not much left to do but log in to the server. I tried ssh [root@mydomain.tld](ssh://root@mydomain.tld) right away, and it didn't work, and I had to use the IP address. But after a little while, it worked fine. DNS is fun, right?

Legwork

# Initial Login

Upon SSH'ing into the server, I am greeted w/ a cold black and white terminal. Excellent.

## Housekeeping

Before moving forward, let's get the system up to date and grab a few packages we'll need momentarily.

```
freebsd-update fetch install
```

```
pkg update && pkg upgrade -y
```

```
pkg install -y git nano tmux vim-console lsblk
```

I was tempting to install `curl` and `wget`, but those are covered with `fetch`, which is part of the FreeBSD base system.

Get the clock up to date, just in case it's not.

```
tzsetup
```

For the host, you'll probably want to make sure you select the timezone where the VPS is located.

Enable the time service.

```
sysrc ntpd_enable="YES"
```

```
sysrc ntpd_sync_on_start="YES"
```

And then might as well start it now.

```
service ntpd start
```

What are we working with here (supposedly a 25G SSD, right?)... `lsblk` reveals:

DEVICE	MAJ:MIN	SIZE	TYPE	LABEL	MOUNT
vtbd0	0:69	25G	GPT	-	-
vtbd0p1	0:71	256K	freebsd-boot	gptid/fe84c375-529c-11ea-...	-
vtbd0p2	0:72	2.0G	freebsd-swap	gpt/swap0	-
vtbd0p3	0:89	23G	freebsd-zfs	gpt/disk0	<ZFS>

Running `swapinfo -h` confirms the swap space.

# Quality of Life

I prefer to freshen up the `.cshrc` file. Here's the original.

```
# $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 -lAF

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

```

set promptchars = "%#"

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

endif

```

And then I added some things, mostly inside the `if` block:

```

if ($?prompt) then

    #set prompt = "%N@%m:%~ %# " # comment this out, and replace it below

    # 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... :(

    # Set username color. (Preferably, unique)
    if ( $USER == "root" ) then
        set user_color="${yellow}"
    else
        set user_color="${magenta}"
    endif

```

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

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

set complete = enhance

# 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

# BSD colors for ls.  See https://geoff.greer.fm/lscolors/
setenv LSCOLORS gxfxcxdxbxegedabagacad
```

## Turn off the mail notifications

Add the following to `/etc/rc.conf`, whether pasting or by `sysrc`.

```
sendmail_enable="NO"
sendmail_submit_enable="NO"
sendmail_outbound_enable="NO"
sendmail_msp_queue_enable="NO"
```

# Security

## More remote user access

I've got two local computers. I pasted my other comp's `SSH` pubkey in a shared file where this comp could copy it, and then I pasted it into `.ssh/authorized_keys`. Now both computers can `SSH` in.



Sweet.

## SSH authentication

We may want to clean up the `/etc/ssh/sshd_config` by disallowing password logins. That particular setting is already set to 'no' but there are other things, like `ChallengeResponseAuthentication` and `UsePAM` which could also be set to 'no' (for a belt-and-suspenders approach, as they say).

## System users

The default user is the root user. To continue the belt-and-suspenders approach theme, we could create a user for me (in the wheel group and operator group) and then disable root `SSH` login as well. Smart. But I'm skipping that.

## Root password

More belt-and-suspenders - we could/should add a root password. Very smart. Why am I skipping this?

## Obscurity

We could change the default listening port. This will decrease the noise, but is it really worth it? I mean, once the server is set up, you can just block SSH via the DigitalOcean firewall. It's like 2FA in that regard... sorta. Also inconvenient though.

## Blocking bots

We could/should install Fail2Ban. Maybe later. For now (a bit below), we'll implement basic, built-in brute force protection.

## Firewall

Here's how we're starting off with `/etc/pf.conf`:

```
# macros -> tables -> options -> traffic normalization -> queueing -> translation (NAT) -> packet filtering

## MACROS

# the external network interface to the internet
ext_if="vtnet0"
ext_if_ip="YOUR.NEW.IP.HERE"

# port on which sshd is running
ssh_port = "22"
```

```
# allowed inbound ports (host services)
# don't include ports that'll be redirected to jails
#inbound_tcp_services = "{auth, " $ssh_port " }"
#inbound_udp_services = "{dhcpv6-client}"
web_services = "{http, https}"

# jail IP addresses
#caddy_ip = "10.101.10.100"
#bookstack_ip = "10.101.10.110"

## TABLES

table <bruteforce> persist
table <webcrawlers> persist
table <rfc6890> { 0.0.0.0/8 10.0.0.0/8 100.64.0.0/10 127.0.0.0/8 169.254.0.0/16 \
172.16.0.0/12 192.0.0.0/24 192.0.0.0/29 192.0.2.0/24 192.88.99.0/24 \
192.168.0.0/16 198.18.0.0/15 198.51.100.0/24 203.0.113.0/24 \
240.0.0.0/4 255.255.255.255/32 }
# jails table is used by Bastille
table <jails> persist

## OPTIONS

# politely send TCP RST for blocked packets. The alternative is
# "set block-policy drop", which will cause clients to wait for a timeout
# before giving up.
set block-policy return

# log only on the external interface
set loginterface $ext_if

# skip all filtering on localhost
set skip on lo

## TRAFFIC NORMALIZATION

# reassemble all fragmented packets before filtering them
scrub in on $ext_if all fragment reassemble max-mss 1440
```

## ## TRANSLATION

# NAT out jail traffic

nat pass on \$ext\_if from <jails> to any -> \$ext\_if\_ip

## static port forwarding for http/https traffic to [reverse proxy] jail

#rdr pass inet proto tcp from any to any port {80, 443} -> \$caddy\_ip

## ## PACKET FILTERING

# block forged client IPs (such as private addresses from WAN interface)

antispoof quick for \$ext\_if

# skip rfc6890 on external interface

block in quick on egress from <rfc6890>

block return out quick on egress to <rfc6890>

# default behavior: block all traffic

block all

# allow all icmp traffic (like ping)

pass quick on \$ext\_if proto icmp

pass quick on \$ext\_if proto icmp6

# special pass rules for SSH

pass in quick on \$ext\_if proto tcp to port \$ssh\_port \

keep state (max-src-conn 6, max-src-conn-rate 4/10, \

overload <bruteforce> )

# special pass rules for http/https (NOT SURE ABOUT THIS W/ REVERSE PROXY)

#pass in on \$vtnet0 proto tcp to port { 80 443 } \

# keep state (max-src-conn 45, max-src-conn-rate 9/1, \

# overload <webcrawlers> flush global)

# allow incoming traffic to services hosted by this machine (to the host; not jails)

#pass in quick on \$ext\_if proto tcp to port \$inbound\_tcp\_services

#pass in quick on \$ext\_if proto udp to port \$inbound\_udp\_services

# allow all outgoing traffic

pass out quick on \$ext\_if

There's some nice foreshadowing in there.

There's a lot going on in that file. Here's a quick rundown:

1. Macros are for expanding into the rules below. Additionally, they make it so you can have rules that don't change between systems because you just have to update the macros for the system-specific variables.
2. Some of the macros are commented out for now. We may add them later. For now, this is all we need.
3. A table is kind of like a macro, but not really. For one, they don't expand out in the shell's variable substitution (which you'll see with `pfctl -vnf /etc/pf.conf`). They behave that way though, where a rule applies to every IP address or range in the table. The `bruteforce` and `webcrawler` tables are for keeping track of IP's that abuse the server. The `rfc6890` table is for IP's that we should never encounter anyway. The `jails` table is populated by Bastille (the jail/container manager we'll be using, which we'll talk more about later), and it contains all the Bastille jail IP's.
4. The options generally are already commented (not commented out... literally commented, so go read them).
5. The NAT rule (in combination with `gateway_enable` in `/etc/rc.conf`) allows jail traffic to pass out through the external interface and appear to come from the server's IP rather than the otherwise un-routeable jail's IP.
6. The rest is either commented, self-explanatory (perhaps with research), or commented on below.

Double check that the syntax is good and appears correct.

```
pfctl -vnf /etc/pf.conf
```

Assuming no errors, now it can be enabled.

```
sysrc pf_enable="YES"
```

And then, finally, PF can be started, which will boot us from our `SSH` session(s).

```
service pf start
```

It is wise to enable logging too. Besides, we'd need to parse the logs for Fail2Ban. (Can it parse binary logs?)

```
sysrc pflog_enable="YES"
```

The data in the `pflog` file is written in binary, so one way to view it is:

```
tcpdump -ner /var/log/pflog
```

May also want to install `pftop`.

## PF Tables

We're getting brute force protection w/ a `bruteforce` table. This is already included above.

```
table <bruteforce> persist
table <webcrawlers> persist
table <rfc6890> { 0.0.0.0/8 10.0.0.0/8 100.64.0.0/10 127.0.0.0/8 169.254.0.0/16 \
172.16.0.0/12 192.0.0.0/24 192.0.0.0/29 192.0.2.0/24 192.88.99.0/24 \
192.168.0.0/16 198.18.0.0/15 198.51.100.0/24 203.0.113.0/24 \
240.0.0.0/4 255.255.255.255/32 }

{...}

# skip rfc6890 on external interface
block in quick on egress from <rfc6890>
block return out quick on egress to <rfc6890>

{...}

# special pass rules for SSH
pass in quick on $ext_if proto tcp to port $ssh_port \
keep state (max-src-conn 6, max-src-conn-rate 4/10, \
overload <bruteforce> )
# the very end could have 'flush global', but I'd risk ending my
# own connection if I accidentally triggered it

# could do a similar rule for http/https using <webcrawlers>
```

We'll periodically empty/clear the `bruteforce` table (since brute forcing tends to not happen from the same IP for long periods of time), so we'll write a shell script to do this for us.

```
vim /usr/local/bin/clear_overload.sh
```

In the script, we'll keep the IP's that have been around for 2 weeks or less, and flush/expire all else.

```
#!/bin/sh

pfctl -t bruteforce -T expire 1209600
```

It needs to be executable.

```
chmod 755 /usr/local/bin/clear_overload.sh
```

And then we put it in a `cron` job to run every first day of the week on the zeroth minute of 2am (with `crontab -e`).

```
# minute hour mday month wday command  
  
0 2 * * 1 /usr/local/bin/clear_overload.sh
```

## Other Config

It's worth looking at the `/etc/rc.conf`. It comes like this:

```
hostname="freebsd-hostname"  
cloudinit_enable="YES"  
sshd_enable="YES"  
ifconfig_vtnet0="DHCP"  
digitaloceanpre="YES"  
digitalocean="YES"  
zfs_enable="YES"  
  
# DigitalOcean Dynamic Configuration lines and the immediate line below it,  
# are removed each boot.  
  
# DigitalOcean Dynamic Configuration  
defaultrouter="YOUR.NEW.IP.HERE"  
# DigitalOcean Dynamic Configuration  
ifconfig_vtnet0="inet YOUR.NEW.IP.HERE netmask 255.255.240.0"  
# DigitalOcean Dynamic Configuration  
ifconfig_vtnet0_alias0="inet 10.10.0.5 netmask 255.255.0.0"  
# DigitalOcean Dynamic Configuration  
ifconfig_vtnet0_ipv6="inet6 2604:A880:0400:00D0:0000:0000:1B07:F001 prefixlen 64"  
# DigitalOcean Dynamic Configuration  
ipv6_defaultrouter="2604:A880:0400:00D0:0000:0000:0000:0001"  
# DigitalOcean Dynamic Configuration  
ipv6_activate_all_interfaces="yes"
```

And we have added:

```
ntpd_enable="YES"
ntpd_sync_on_start="YES"

sendmail_enable="NO"
sendmail_submit_enable="NO"
sendmail_outbound_enable="NO"
sendmail_msp_queue_enable="NO"

pf_enable="YES"
pflog_enable="YES"
```

And we'll soon be adding (more foreshadowing!):

```
bastille_enable="YES"
cloned_interfaces="lo1"
ifconfig_lo1_name="bastille0"
gateway_enable="YES"
```

## Bonus Round

I've been using `vim` more than usual during this process.

1. It would be helpful to begin to put together a config file (`.vimrc`).
2. It would be wise to add it to my `custom_cshrc.sh` script, so I have it in jails too. (More foreshadowing!)

Anywoo, this is still a todo.

# 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` - vnet (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 jail'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 jail'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/>

# Fun Stuff



# BookStack Jail

## Prerequisites

Have a jail called `bs_jail`

We already created a handful at once. Let's look (at the relevant output).

```
[root@freebsd:~]# bastille list
```

JID	IP Address	Hostname	Path
bs_jail	10.101.10.110	bs_jail	/usr/local/bastille/jails/bs_jail/root

## Initial Prep

You might as well make sure you have your custom `.cshrc` in the jail (see `custom_cshrc.sh` saved in `/usr/local/scripts`), and maybe run `tzsetup` as well.

(Most everything below is performed **outside** the jail.)

Install some initial necessary packages.

```
bastille pkg bs_jail install -y vim-console git sudo bash
```

## Advanced Prep (nullfs)

(Relocate database outside the jail)

If we ever have a problem with this jail and need to blow it away, it would be nice for the database to live on. We can do this! In fact, this is probably one of several steps that could/should be taken to ensure data not specific to the jail is saved outside the jail.

First, let's create a directory for it the db to live. You can `mkdir -p` this step. I have ZFS and `/zroot/data` already, so it'll be:

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

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

MariaDB (MySQL) stores the database in `/var/db/mysql`. In fact, it probably would store multiple databases in there if we were using it for something else in the jail. Luckily, we're not. So there's our directory where we'll mount the new directory.

In order to get a `/var/db/mysql` in the first place, `mysql` needs to be installed, so we'll do that now.

```
bastille pkg bs_jail install -y mariadb102-client mariadb102-server
```

As mentioned on a prior page, newer packages for `mariadb` seem to behave differently, so parts of this tutorial related to `mariadb` may need to be adjusted.

I'm leaving myself (and whoever else) a possible hint. A newer `mysql` has a different syntax. The following link talks about it midway down the page:

<https://arstechnica.com/gadgets/2020/05/caddy-offers-tls-https-and-more-in-one-dependency-free-go-web-server/> ... I suspect there's more to it though...

The user and group that own the BookStack db are both `88` (which is the `mysql` user and group, which you can see in the `stdout` from the previous command). We've gotta match that, and the permissions.

```
cd /usr/local/data/dbs/
```

```
chown 88:88 bookstack/
```

Next double check that the folder `/var/db/mysql` exists. It should. If it does, proceed with:

```
bastille stop bs_jail
```

Now it's time to set up the `fstab`. In the case of `bastille`, it's in `/usr/local/bastille/jails/$NAME`. For a thin jail, there will already be a line in the `fstab`, so this can be pasted in prior to it, or after, or just the relevant row.

#	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

While the jail is stopped, we need to ensure `mysql` (`mariadb`) has the powers it needs.

```
echo 'allow.raw_sockets = "1";' >> /usr/local/bastille/jails/bs_jail/jail.conf
```

Now you can restart the jail and finish the setup.

```
bastille start bs_jail
```

## Install PHP

Install PHP, as well as the necessary PHP extensions.

```
bastille pkg bs_jail install -y php72 php72-mbstring php72-tokenizer php72-pdo php72-pdo_mysql \
php72-openssl php72-hash php72-json php72-phar php72-filter php72-zlib php72-dom \
php72-xml php72-xmlwriter php72-xmlreader php72-pear php72-curl php72-session \
php72-ctype php72-iconv php72-gd php72-simplexml php72-zip php72-filter php72-tokenizer \
php72-calendar php72-fileinfo php72-intl php72-mysqli php72-phar php72-opcache php72-tidy
```

I guess we can check the version.

```
bastille cmd bs_jail php --version
```

Soft-link `php.ini-production` to `php.ini`.

```
bastille cmd bs_jail ln -s /usr/local/etc/php.ini-production /usr/local/etc/php.ini
```

Enable and start PHP-FPM.

```
bastille sysrc bs_jail php_fpm_enable=yes
```

```
bastille service bs_jail php-fpm start
```

## Install MariaDB

Install MariaDB. (Skip this one step if you already ran this command in anticipation of nullfs-mounting the db folder.)

```
bastille pkg bs_jail install -y mariadb102-client mariadb102-server
```

Might as well check the version.

```
bastille cmd bs_jail mysql --version
```

Enable and start MariaDB.

```
bastille sysrc bs_jail mysql_enable="yes"
```

```
bastille service bs_jail mysql-server start
```

Check if it's running, because we might have permissions issues or something:

```
bastille service bs_jail mysql-server status
```

If there's an issue, one possibility could be the inability to write to `/tmp`. A `bastille cmd bs_jail chmod 1777 /tmp` would solve that. But after `mariadb102`, there seems to be some other issue that I haven't figured out yet.

Assuming we're up and running, let's move on.

## Get MariaDB ready

Run the secure installation executable to lock things down. Note your root password you create.

```
bastille cmd bs_jail mysql_secure_installation
```

Log into MariaDB as the root user.

```
bastille cmd bs_jail mysql -u root -p
```

Create a database (or use an existing name, if you'll be importing, which I will).

```
CREATE DATABASE dbname; # substitute with your choice of name, though it does not matter if creating new
GRANT ALL ON dbname.* TO 'username' IDENTIFIED BY 'password'; # substitute any user and pass
FLUSH PRIVILEGES;
exit;
```

# Install Nginx

Install Nginx.

```
bastille pkg bs_jail install -y nginx
```

Check the version.

```
bastille cmd bs_jail nginx -v
```

Enable and start Nginx.

```
bastille sysrc bs_jail nginx_enable=yes
```

```
bastille service bs_jail nginx start
```

Set up Nginx for BookStack.

```
bastille cmd bs_jail vim /usr/local/etc/nginx/bookstack.conf
```

And we'll add:

```
server {
    listen 80;
    # listen [::]:80;      # you may need to comment this out
    server_name bookstack.mydomain.tld;    # substitute hostname.domain
    root /usr/local/www/bookstack/public;

    index index.php index.html;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ \.php$ {
        try_files $uri =404;
        include fastcgi_params;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_pass 127.0.0.1:9000;
    }
}
```

Now we need to include `bookstack.conf` in the main `nginx.conf` file.

```
bastille cmd bs_jail vim /usr/local/etc/nginx/nginx.conf
```

And add the following line to the `http {}` block.

```
include bookstack.conf;
```

Test the Nginx configuration changes.

```
bastille cmd bs_jail nginx -t
```

Good? Then reload Nginx.

```
bastille service bs_jail nginx reload
```

# Install Composer

Install Composer by running the script on their website. Note the final step is not on their website.

Go to their website for line 2 below: <https://getcomposer.org/download>. The remaining steps are the same (plus line 5).

This is going into the `bs_jail` console again briefly!

```
bastille console bs_jail
```

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('SHA384', 'composer-setup.php') === 'long_hash') { echo 'Installer verified'; } else { echo
'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
mv composer.phar /usr/local/bin/composer
exit
```

And we're back in the land of the host. Now we'll check this version.

```
bastille cmd bs_jail composer --version
```

# Install BookStack

Since `composer` is not intended to be run as root user, we're going to set up a user. We'll run most of these within the jail.

```
bastille console bs_jail
```

In the jail, we'll run `adduser`, with "username" name (whichever name you chose when you gave it privileges to write to the `mysql` db), add it to the `wheel` group, choose `bash` shell, add password, and done. Here's a head start:

```
adduser -s bash -G wheel
```

Great, but let's make this easier on ourselves. Run the `visudo` command and uncomment the `%wheel ALL=(ALL) ALL` line to allow members of the `wheel` group to execute any command.

```
visudo

# Uncomment by removing hash (#) sign
%wheel ALL=(ALL) ALL
```

Then `su - bs_user`, and let's get started already.

Let's create the document root folder and take ownership of it.

```
sudo mkdir -p /usr/local/www/bookstack
```

```
sudo chown -R username:username /usr/local/www/bookstack
```

Substitute with the user you just created (and whose shell you're in now).

Run the composer install command from the `/usr/local/www/bookstack` directory.

```
composer install
```

Copy the `.env.example` file to `.env` and populate it with your own database (and mail details?).

```
cp .env.example .env
```

```
vim .env
```

You can generally get away with just changing the db name, db user, and db password (per MariaDB steps above). You may need to put the user and password in double quotes. Come back to this step if `php artisan migrate` says `access denied`. If importing a database, be sure to use that `db` name. For a public web server, be sure to update `APP_URL` as well.

Optional: Ensure that the `storage`, `bootstrap/cache` and `public/uploads` folders are writable by the web server. (Prob can ignore given we've got a `chown` incoming.)

In the application root (where you should already be), run the following command.

```
php artisan key:generate
```

## Finish up!

To update the database:

```
php artisan migrate
```

If there was an error here, fix the problem, then run the following, and then jump back up three steps (to the `.env` file).

```
php artisan config:clear  
php artisan cache:clear
```

Change ownership of the `/usr/local/www/bookstack` directory to `www`.

```
sudo chown -R www:www /usr/local/www/bookstack
```

You can now login using the default admin details `admin@admin.com` with a password of `password` (or, if you've restored a db, then you can log in with those credentials). It is recommended to change these details directly after your first login. Create your user account as an admin user, log in with it, and then disable the default admin user.

## Are We Really Done?

As things stand, the BookStack webserver is listening on the jail's internal IP on port 80 (http). I would not recommend setting up `pf` to redirect http traffic to the jail. The jail will be waiting and ready when we can access it securely. We'll do that next in our second... err... third jail. We'll create a simple website in the second jail. Plus it'll be time for the following...



Also! In our initial legwork of getting the server set up, we touched on DNS records. Well, now is a good time (actually, these records don't seem to instantaneously populate, so *before* now would have been better) to create a CNAME record. Over in NameCheap, the 'hostname' is "bookstack", or "bs", or whatever you want... "docs"? ... and "mydomain.tld" is the 'value', and save, and you're done.

## Bonus

At the time of writing this, BookStack has not implemented a change requested by users (and even submitted). But it works! One notable item missing from BookStack is the ability to go to the next or previous pages. Well, if you add the following script to the custom header settings, it'll insert this into the `<head>` of the html, and bam, buttons.

The one thing you'll want to do is set your own `rgb` numbers in the two `.bnav-page-button:hover` CSS items, so you'll get whatever color you want, rather than the red that is currently used.

Check out the relevant PR for more info. <https://github.com/BookStackApp/BookStack/issues/1381>

```
<script>
function Button(type, hint, title, attributes){

const prevSVG = '<svg preserveAspectRatio="xMidYMid meet" height="1em" width="1em" fill="none"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" stroke-width="2" stroke-linecap="round" stroke-
linejoin="round" stroke="currentColor"><g><line x1="19" y1="12" x2="5" y2="12"></line><polyline
points="12 19 5 12 12 5"></polyline></g></svg>';
const nextSVG = '<svg preserveAspectRatio="xMidYMid meet" height="1em" width="1em" fill="none"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" stroke-width="2" stroke-linecap="round" stroke-
linejoin="round" stroke="currentColor"><g><line x1="5" y1="12" x2="19" y2="12"></line><polyline
points="12 5 19 12 12 19"></polyline></g></svg>';
var currentSVG = "";

if(type == "next"){
currentSVG = nextSVG
} else {
currentSVG = prevSVG
}

this.element = document.createElement("a");
this.element.classList.add("bnav-page-button");
```

```

this.element.classList.add(type);
var inner = '<div class="bnav-card-svg ' + type + '">' + currentSVG + '</div><div class="bnav-page-card ' +
type + '"><div class="bnav-card-hint"><span>' + hint + '</span></div><div class="bnav-card-
title"><span>' + title + '</span></div></div>'
attributes.innerHTML = inner

for (var i in attributes) {
this.element[i] = attributes[i];
}
return this.element;
}

document.addEventListener("DOMContentLoaded", function() {
if (window.location.pathname.indexOf("page")) {

var pages = document.querySelectorAll("a.page"),
current = document.querySelector("a.selected"),
currentIndex = Array.prototype.indexOf.call(pages, current);

var pageNavLinks = document.createElement("div");
pageNavLinks.classList.add("bnav-page-nav-links")
document.querySelector(".page-content").appendChild(pageNavLinks);

if (pages.item(currentIndex - 1) != null) {
var prevPageEl = pages.item(currentIndex - 1);
var prevButton = new Button('prev', 'Previous Article', prevPageEl.innerText, { href: prevPageEl.href })
document.querySelector(".bnav-page-nav-links").appendChild(prevButton);
}

if (pages.item(currentIndex + 1) != null) {
var nextPageEl = pages.item(currentIndex + 1);
var nextButton = new Button('next', 'Next Article', nextPageEl.innerText, { href: nextPageEl.href })
document.querySelector(".bnav-page-nav-links").appendChild(nextButton);
}
}
});
</script>

<style>

```

```
/* bottom page navigation */
```

```
.bnav-page-nav-links {  
width: auto;  
margin: 3em 0 0 0;  
display: grid;  
padding: 1.5em 0 0 0;  
column-gap: 24px;  
grid-template: "previous next" auto / 1fr 1fr;  
border-top: solid #EAEAEA 1px;  
}
```

```
.bnav-page-button {  
color: rgb(36, 42, 49) !important;  
display: flex;  
margin: 0;  
padding: 0;  
position: relative;  
flex-direction: row;  
align-items: center;  
text-decoration: none !important;  
border: 1px solid rgb(230,236,241);  
border-radius: 3px;  
box-shadow: rgba(116,129,141,0.1) 0px 3px 8px 0px;  
transition: border 250ms ease 0s;  
}
```

```
.bnav-page-button:hover{  
color: rgb(18, 124, 173) !important;  
border-color: rgb(18, 80, 173);  
cursor: pointer;  
}
```

```
.bnav-page-button:hover svg{  
color: rgb(18, 80, 173);  
}
```

```
.bnav-page-button.prev {  
grid-area: previous / previous / previous / previous;
```

```
}

.bnav-page-button.next {
grid-area: next / next / next / next;
}

.bnav-page-card {
flex: 1 1 0%;
margin: 0px;
display: block;
padding: 1em;
text-align: left;
}

.bnav-page-card.next {
text-align: left;
}

.bnav-page-card.prev {
text-align: right;
}

.bnav-card-svg {
padding-right: 0;
flex: 0 0 auto;
color: rgb(157, 170, 182);
margin: 0px;
display: block;
padding: 16px;
font-size: 24px;
}

.bnav-card-svg.prev {
order: 0
}

.bnav-card-svg.next {
order: 1
}
```

```
.bnav-card-svg > svg {  
width: 1em;  
height: 1em;  
vertical-align: middle;  
transition: color 250ms ease 0s;  
}
```

```
.bnav-card-hint {  
color: rgb(157, 170, 182);  
margin: 0;  
display: block;  
padding: 0;  
}
```

```
.bnav-card-hint > span {  
font-size: 12px;  
font-weight: 400;  
line-height: 1.2;  
}
```

```
.bnav-card-title {  
margin: 0px;  
display: block;  
padding: 0px;  
transition: color 250ms ease 0s;  
}
```

```
.bnav-card-title > span {  
font-size: 16px;  
font-weight: 500;  
line-height: 1.5;  
}
```

```
.bnav-card-icon {  
flex: 0 0 auto;  
color: rgb(157, 170, 182);  
margin: 0px;  
display: block;
```

```
padding: 16px;
font-size: 24px;
transition: color 250ms ease 0s;
}

/* end bottom page navigation */
</style>
```

## Bonus #2: Updating

According to BookStack site, this can be done very quickly in a single line. We'll try it.

```
git pull origin release && composer install --no-dev && php artisan migrate
```

It works! It warns you that you're doing this migration in production, and you say 'yes' and it's done.

## References

<https://www.vultr.com/docs/how-to-install-bookstack-on-freebsd-12>

Updating: <https://www.bookstackapp.com/docs/admin/updates/>

I skipped a few things, but it should work as I describe.

# Website Jail

Before this, I can't think of a time where I edited or wrote html. I can remember creating a basic `index.php` as a test for nginx and/or apache a couple times while tinkering with Nextcloud, but that might be it.

Accordingly, this will be a very basic start of a very simple website. I maybe look forward to doing "cool" complicated stuff in the future, but for now we'll have close to nothing on it. I'm creating the web page because I figure that I might as well have a landing page for the domain itself, but I'm more interested in setting up the reverse proxy work for the subdomains.

To set the expectations properly, the goal is to create an `html` file that renders in a browser by visiting `mydomain.tld`. We'll not be worrying about TLS/https (because `caddy` will eventually do that for us). We'll simply install a web server, create the `html` file, port forward (`rdr`) in `PF` to the jail, and visit in the browser. Someone who's done this a couple times - even if they're documenting it - might be done in under two minutes. It took me more than two minutes.

## Prep

Run the `custom_cshrc.sh` you created in `/usr/local/scripts` to put a custom `.cshrc` file in the jail. Remember, the script just takes the jail name as its only argument.

If desired, adjust the date and time with `tzsetup` or `bastille cmd website_jail tzsetup`.

## Web Server

We'll keep it simple and consistent (i.e., BookStack is served by `nginx`), so we'll install `nginx`.

```
bastille pkg website_jail install -y nginx vim-console
```

And then we'll enable it and start it.

```
bastille sysrc website_jail nginx_enable="YES"
```

```
bastille service website_jail nginx start
```

We'll configure it in a moment.

# Internet Content

That sure is a fancy title for a bare `html` file.

Let's just hop into the jail console for a few minutes.

```
bastille console website_jail
```

And we'll head to the usual FreeBSD spot, create a website directory, and then file.

```
cd /usr/local/www
```

```
mkdir mydomain.tld && cd mydomain.tld
```

```
vim index.html
```

And we will create our initial homepage.

```
<!DOCTYPE html>
<html>
<body>

<h1>We Did It!</h1>

<p>How exciting.</p>

<p>Be sure to check out all the great related services.  Links coming soon...</p>

</body>
</html>
```

# Configuration

Now we can create our configuration in `nginx` so it knows how to listen and what content to serve.

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



In theory, all we have to do is change `server_name localhost` to `server_name mydomain.tld` `www.mydomain.tld` and change `root /usr/local/www/nginx` to `root /usr/local/www/mydomain.tld`. With any luck, we can reload `nginx` and be ready to test (almost).

Before moving forward, `exit` out of the jail console.

First we test the config (even though the test is built into the reload).

```
bastille cmd website_jail nginx -t
```

If successful, we perform the reload.

```
bastille service website_jail nginx reload
```

## Testing It Out

You'll need the jail's IP for this, which you can get from `bastille list`.

Then there needs to be a redirect rule in `PF`, which is basically port forwarding. There's an example already in `/etc/pf.conf`, so it just needs to be uncommented, and updated with the website jail's internal IP.

```
# the macro
website_ip = "10.101.10.140"

# the port forward
rdr pass inet proto tcp from any to any port {80, 443} -> $website_ip
```

And it needs to be tested with:

```
pfctl -vnf /etc/pf.conf
```

## Hiccup

NameCheap.com provides a default CNAME record that redirects my internet traffic to their "parking page" and I hadn't deleted that yet, so I had to wait on it to die.

Visiting the IP address does successfully display the webpage, but it would have been nice to see DNS do what it's supposed to too. Of course, it worked via hostname eventually.

## Last Step

Remove those rules from `pf` and force reload `pf`. We will be using `https` in no time flat after the next jail is up.

# Caddy Jail

We will ultimately change `PF` to direct all web traffic to this jail. This jail will run `caddy` as a reverse proxy for the other jails. Web request SSL terminations happen at the `caddy` web server, and the traffic is then passed transparently to the respective jails. A great benefit of `caddy` is the built-in Let's Encrypt feature for initial certs and renewals.

## Preamble

The beginning steps are mostly the same across the jails. Before jumping in, if you haven't already, remember to run `custom_cshrc.sh caddy_jail`, and then probably/possibly run `bastille caddy_jail tzsetup` and choose your time zone. Actually, it would make the most sense for the reverse proxy to be on the host's time, which it should be already, so ignore that.

Next, we 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, refer to a prior page that talks about initial jail setup.

## Setup Specific to this Jail

We install what we need from `pkg`.

```
bastille pkg caddy_jail install -y caddy vim-console curl
```

You should read the message spit out by `pkg` because it tells you all you need to know, pretty much. In particular, pay attention to the version of `caddy`. This write-up centers around `v1`. This write-up will not work well with `v2`.

## Config for the Jail

We'll need to give `caddy` the ability to "authenticate" us with Let's Encrypt.

```
bastille sysrc caddy_jail caddy_cert_email="your.email@example.org"
```

And then we'll need the `Caddyfile`, which hopefully works how we think it will.

But wait! Save yourself some time and run this:

```
bastille cmd caddy_jail caddy -version
```

Your config/Caddyfile will be different depending on v1 or v2. The quarterly FreeBSD package is v1 right now (as of the time of this original write-up).

```
bastille console caddy_jail
```

```
cd /usr/local/
```

```
mkdir www && cd www
```

```
vim Caddyfile
```

Depending on V1 or V2, mind the `Caddyfile` location. V2 moves the `Caddyfile` location from `/usr/local/www` to `/usr/local/etc/caddy/Caddyfile`, so be sure its location matches the location listed in the `rc` file (and is preferably in the standard location according to V1 or V2).

For `v1`:

```
mydomain.tld, www.mydomain.tld {
  proxy / 10.101.10.140:80 {
    transparent
  }
}

bookstack.mydomain.tld {
  proxy / 10.101.10.110:80 {
    transparent
  }
}
```

For `v2`:

```
mydomain.tld, www.mydomain.tld {
  reverse_proxy 10.101.10.140
}
```

```
bookstack.mydomain.tld {  
  reverse_proxy 10.101.10.110  
}
```

Then `exit` out of the jail's console. And then we enable `caddy` and start it (almost).

```
bastille sysrc caddy_jail caddy_enable="YES"
```

# Grand Finale

Now we adjust `/etc/pf.conf` to forward `http` and `https` traffic to the `caddy` jail.

```
# the macro  
caddy_ip = "10.101.10.100"  
  
# and the port forward  
rdr pass inet proto tcp from any to any port {80, 443} -> $caddy_ip
```

And then we test that the config doesn't have an errors, and then reload `PF`. (Reload w/ just `-f`.)

```
pfctl -vnf /etc/pf.conf
```

And now let's start `caddy` and hope that it grabs certs and starts serving our two existing jails.

```
bastille service caddy_jail caddy start
```

And either check the URL in your browser, or also check:

```
bastille service caddy_jail caddy status
```

That was easy.

# More Fun Stuff

# Bitwarden-rs Jail

My company provides a password manager, so I don't *need* this. But what they provide is closed source. I may just switch over to bitwarden, but perhaps little by little. For now, I just want to get this working.

Like the password manager I'm provided for work, this is a zero-knowledge setup; i.e., the database is stored in an encrypted state, locked by my complex pass phrase. If someone gains control of the jail, or even the host system, the database of records/credentials does them no good. It's still scary to put up a publicly-accessible instance, but I'm doing it anyway. Besides, I'll only use it in limited capacity for now, and perhaps I'll put it behind a VPN (Wireguard?) at some point.

## Advanced Prep (nullfs)

### (Relocate database outside the jail)

If we ever have a problem with this jail and need to blow it away, it would be nice for the database to live on. We can do this! In fact, this is probably one of several steps that could/should be taken to ensure data not specific to the jail is saved outside the jail. We already did this for the BookStack jail (and the majority of this section is a straight copy). Carrying on:

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

Bitwarden-rs stores the database in a `/data` directory. The standard install creates the `/data` directory in `/home/bitwardenrs/bitwarden_rs_dist`. Due to the fact that this is a thin jail, the `/home` directory (a few directories deep) where the null mount would go cannot be used; so instead we'll mount the `/data` directory over top of the new data dir we'll create.

We will need to adjust the `.env` file accordingly.

```
bastille console bw_jail
```

```
cd /var && mkdir -p db/data
```

Then `exit` from the `su` and `exit` from the console.

```
bastille stop bw_jail
```

Adjust the jail's `fstab`.

# Device	Mountpoint	FStype	Options	Dump	Pass#
/usr/local/bastille/releases/12.1-RELEASE	/usr/local/bastille/jails/bw_jail/root/.bastille	nullfs	ro	0	0
/usr/local/data/dbs/bitwarden	/usr/local/bastille/jails/bw_jail/root/var/db/data	nullfs	rw,late	0	0

```
bastille start bw_jail
```

Below, you'll need to set ownership or permissions on this `/var/db/data`, otherwise bitwarden-rs can't write to it.

## On With It

First of all, let's **not** run `tzsetup` on this jail. That gave me problems with 2FA on another instance. Let's try without it.

Grab initial packages/dependencies.

```
bastille pkg bw_jail install -y sqlite3 nginx git sudo vim-console bash node npm python27-2.7.18
```

Adjust

```
# We've got to do a bit of work inside the jail (it'll be easier there)

bastille console bw_jail

# some npm dependency will need to have python2.7 and will fail with python3

cd /usr/local/bin/

# set the symlink
```



```
In -s /usr/local/bin/python2.7 python
```

```
cd -
```

## Set up user.

Add new bitwardenrs user to the jail. Set the user below to: bitwardenrs. Enter every line, no need for other configs, only your password

```
adduser -s bash
```

## Adjust priv's and log in:

```
# allow sudo, we will use it later
```

```
visudo
```

```
# ADD
```

```
bitwardenrs ALL=(ALL) ALL
```

```
# change to the new user to build and execute our service
```

```
su bitwardenrs
```

```
cd
```

```
id
```

```
# should look like: uid=1001(bitwardenrs) gid=1001(bitwardenrs) groups=1001(bitwardenrs)
```

## One add'l step needed: (maybe... try skipping it)

```
bitwardenrs@bw_jail:~ $ exit
```

```
root@bw_jail:~ # chmod 1777 /tmp
```

```
root@bw_jail:~ # su bitwardenrs
```

## Another add'l step, and you can't skip this:

Within jail, as root , `cd /var/db && chown -R bitwardenrs:bitwardenrs data` )

## Install rust

```
# install latest rust version, pkg version may be outdated and can't build bitwarden_rs
```

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

```
# include the rust env variables
```

```
source $HOME/.cargo/env
```

## Time to build.

```
# get back to the users home
```

```
cd ..
```

```
# checkout the latest bitwarden_rs release
```

```
git clone https://github.com/dani-garcia/bitwarden_rs/
```

```
cd bitwarden_rs/
```

```
git checkout "$(git tag --sort=v:refname | tail -n1)"
```

```
# and build it with sqlite support
```

```
cargo build --features sqlite --release
```

```
cargo install diesel_cli --no-default-features --features sqlite-bundled
```

```
cd ..
```

"If you need web-vault, we will build it here." Of course we need the web vault.

```
# WEB-VAULT
```

```
# clone the repository
```

```
git clone https://github.com/bitwarden/web.git web-vault
```

```
cd web-vault
```

```
# switch to the latest tag, is not working here, dani-garcia/bw_web_builds lacks v2.12.0 patch
```

```
# export WEB_VERSION="$(git tag --sort=v:refname | tail -n1)"
```

```
# lets use the last working version
```

```
# ^^ use that `export` command instead of the below
```

```
export WEB_VERSION=v2.14.0
```

```
git checkout ${WEB_VERSION}
```

```
# download and apply the bitwarden_rs patch
```

```
curl https://raw.githubusercontent.com/dani-garcia/bw_web_builds/master/patches/${WEB_VERSION}.patch
```

```
>${WEB_VERSION}.patch
```

```
git apply ${WEB_VERSION}.patch -v
```

"Install dependencies and fix some issues."

```
# there is no native freebsd version from node-sass 4.11, lets bump it to 4.12.0
```

```
cat package.json | sed -e 's/"node-sass": "^4.11.0"/"node-sass": "4.13.0"/' | tee package.json
```

```
# I deleted a ^ and changed to 13, from the original write-up I found
```

```
# download submodules
```

```
npm run sub:init
```

```
# manually install angular/compiler-cli
```

```
npm i @angular/compiler-cli
```

```
# install all the other dependencies
```

```
npm install
```

```
# sweetalert used to fail with the latest angular2, but it's been fixed
```

## Finally, Build the web-vault

```
npm run dist
```

A 1G RAM VPS instance will run out of memory. Interestingly, it acted incapable of using the swapfile, though I wonder if I could have forced it to.

I had to resize the droplet to get 2G of RAM, and then `export NODE_OPTIONS=--max_old_space_size=4096` (from within the bash shell). And then it works, right? Right.

"At this point we have every components and will have to put them together"

```
cd
```

```
# copy bitwarden_rs dist
```

```
cp -r ~/bitwarden_rs/target/release bitwarden_rs_dist
```

```
cd bitwarden_rs_dist
```

```
# and copy the web-vault files
```

```
cp -r ../web-vault/build web-vault
```

# Config

There are `.env` file settings to change. From bitwardenrs user's home dir:

```
cp bitwarden_rs/.env.template bitwarden_rs_dist/.env
```

Edit accordingly (remember, we chose a different data dir to null-fs mount).

```
## Main data folder
DATA_FOLDER=/var/db/data

{...}

## Domain settings
DOMAIN=https://vault.mydomain.tld
```

# Set up nginx.

```
su ☐# be root

bash

# create nginx.conf

cat << EOF >/usr/local/etc/nginx/nginx.conf
worker_processes auto;

events {
    worker_connections 1024;
}
```

```

http {
    include    mime.types;
    default_type  application/octet-stream;
    sendfile    on;
    keepalive_timeout 6h;

#server {
    #listen 80;

    #server_name vault.mydomain.tld;
    #return 301 https://$server_name$request_uri;
#}

server {

    listen 10.101.10.120:80;
    server_name vault.mydomain.tld;


    #ssl_session_cache builtin:1000 shared:SSL:10m;
    #ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    #ssl_ciphers HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
    #ssl_prefer_server_ciphers on;


    access_log      /var/log/nginx/bitwarden_rs_web_vault.log;


    location / {
        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Proto $scheme;


        proxy_pass          http://10.101.10.120:8000;
        proxy_read_timeout  90;


        #proxy_redirect      http://10.101.10.120:8000 https://10.101.10.120;
    }
}

}

EOF

# enable and start nginx

```

```
sysrc nginx_enable="YES"
```

```
nginx -t # test
```

```
service nginx start
```

That's right. Just port 80. It's behind Caddy, remember?

"Create the bitwardenrs init script"

```
mkdir -p /usr/local/etc/rc.conf.d/
```

```
# limit the rocket server only to localhost
```

```
echo "ROCKET_ADDRESS=10.101.10.120" >/usr/local/etc/rc.conf.d/bitwardenrs # changed to actual
```

```
#
```

```
cat <<EOF > /usr/local/etc/rc.d/bitwardenrs
```

```
#!/bin/sh
```

```
# PROVIDE: bitwardenrs
```

```
# REQUIRE: LOGIN DAEMON NETWORKING
```

```
# KEYWORD: jail rust
```

```
# Enable this script by adding:
```

```
# bitwardenrs_enable="YES"
```

```
# ... to /etc/rc.conf
```

```
. /etc/rc.subr
```

```
name="bitwardenrs"
```

```
rcvar="bitwardenrs_enable"
```

```
bitwardenrs_chdir="/home/bitwardenrs/bitwarden_rs_dist"
```

```
# This is the tool init launches
```

```
command="/usr/sbin/daemon"
```

```
pidfile="/var/run/${name}.pid"

# This is the tool daemon launches
task="./bitwarden_rs"
procname="/bin/bash"

command_args="-u bitwardenrs -p ${pidfile} ${task}"

load_rc_config $name
run_rc_command "$1"
EOF

sudo sysrc bitwardenrs_enable="YES"

sudo chmod +x /usr/local/etc/rc.d/bitwardenrs

sudo service bitwardenrs start
```

Before going lower, be sure to create a CNAME record to catch `vault.mydomain.tld`. Done?  
Let's proceed.

## Adjust `pf.conf` to allow connections.

Just kidding! We're not touching PF. We've got caddy. Modify the `Caddyfile` in the `caddy_jail`. Add the following:

```
vault.mydomain.tld {

  gzip

  # The negotiation endpoint is also proxied to Rocket
  proxy /notifications/hub/negotiate 10.101.10.120:80 {
    transparent
  }

  # Notifications redirected to the websockets server
  proxy /notifications/hub 10.101.10.120:3012 {
```



```
websocket
}

# Proxy the root directory to Rocket
proxy / 10.101.10.120:80 {
    transparent
}
}
```

Or is it `encode gzip`? No, that's v2. Your welcome, future self.

Then reload caddy.

```
bastille service caddy-jail caddy restart
```

## After Setup! Clean Up!

First, log onto your beautiful self-hosted, powered-by-rust password manager site, and set up an account with an uncrackable password. Then...

This server is open to others to sign up and use. Go into the `.env` file and shut off new user signups!

```
## Controls if new users can register
SIGNUPS_ALLOWED=false
```

And then restart the service or restart the jail. (If you just restart the service, you may be stuck in the terminal, so I just restart the jail.) When you visit and try to sign up again with a new account, it'll pretend to allow you, and then give you a failure warning.

## Also, 2FA

Bitwarden-rs allows you to various methods of 2FA. The simplest and most common is an Authenticator app. Do it right away.

## References

Adapted from: [https://www.ixsystems.com/community/threads/how-to-build-your-own-bitwarden\\_rs-jail.81389/](https://www.ixsystems.com/community/threads/how-to-build-your-own-bitwarden_rs-jail.81389/)

More here:

[https://www.reddit.com/r/Bitwarden/comments/dg78bi/building\\_selfhosted\\_bitwarden\\_via\\_bitwarden\\_rs/](https://www.reddit.com/r/Bitwarden/comments/dg78bi/building_selfhosted_bitwarden_via_bitwarden_rs/)

Also: <https://dennisnotes.com/note/20181112-bitwarden-server/> (Ubuntu, Docker, nginx, script install, backup procedure)

# Gitea Jail

This will be our very own, lightweight personal Github/Gitlab. And we'll do something pretty cool with it later.

This should be easy by now, right? Now that it works, it sure looks short and easy...

## Set up location of repos/db

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

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

## Create the jail

```
bastille create git_jail 12.1-RELEASE 10.101.10.150
```

## Setup - pre-login

Run `/usr/local/scripts/custom_cshrc git_jail` to copy the `.cshrc`.

```
bastille start git_jail
```

## Setup - post-login

Log into console.

```
bastille console git_jail
```

Download packages possibly needed. (Possibly with `sqlite3` as well)

```
pkg install -y git gitea vim-console
```

Create the folder where the nullfs mount will occur (for one of the two; the other was created by installing gitea).

```
mkdir -p /usr/local/data/git
```

```
chown git:git /usr/local/data/git
```

The `chown` command is probably premature. After the jail is restarted with the updated `fstab`, you probably need to do it again (from within the jail), and it may need to be done for the other directory (nullfs-mounted) in the `fstab` as well.

`Exit` the console.

## Finishing setup touches

Stop the jail.

```
bastille stop git_jail
```

Edit the `fstab` of this thin jail to mount the git dataset.

#	Device	Mountpoint	FStype	Options	Dump	Pass#
	/usr/local/data/git	/usr/local/bastille/jails/git_jail/root/usr/local/data/git	nullfs	rw,late	0	0
	/usr/local/data/dbs/gitea	/usr/local/bastille/jails/git_jail/root/var/db/gitea	nullfs	rw,late	0	0

For the db, we'll need to allow raw sockets. (Actually, probably not needed if using `sqlite3`. Needed for `Mariadb` though.)

```
echo 'allow.raw_sockets = "1";' >> /usr/local/bastille/jails/git_jail/jail.conf
```

And we'll start up the jail again.

```
bastille start git_jail
```

May want to pop into the console now to change ownership (`chown`) of the "Device" entries from the `fstab`.

## Jail is ready for package setup

### Sqlite3

I tried to `pkg install` it, but it said it was already there. No further setup should be necessary. I was having issues at first, and I couldn't figure out the problem, so I ended up creating the db ahead of time in case that was it. I don't think it was, and so creating the db ahead of time should not be needed.

## Gitea

Enable it.

```
bastille sysrc git_jail gitea_enable=YES
```

Make a backup of the config file. First, log into the console.

```
bastille console git_jail
```

```
cp /usr/local/etc/gitea/conf/app.ini /usr/local/etc/gitea/conf/app.ini.bak
```

Configure as necessary the `/usr/local/etc/gitea/app.ini`. (View the changes, but you can't make them all yet. See below.)

```
#APP_NAME can be fun to change
```

```
[database]
```

```
< USER = root
```

```
> USER = git
```

```
[oauth2]
```

```
< JWT_SECRET = D56bmu6xCtEKs9vKKgMKnsa4X9FDwo64HVyaS4fQ...
```

```
> JWT_SECRET = HO8YPNfNkhB_-ESE5e637TQcbja0WylpplsiFdgm...
```

```
[picture]
```

```
DISABLE_GRAVATAR = true
```

```
[repository]
```

```
# I copied (cp -a) the .gitconfig and .ssh file and dir from /usr/local/git (the default git home dir)
```

```
< ROOT = /var/db/gitea/gitea-repositories
```

```
> ROOT = /usr/local/data/git
```

```
# I have this for later. I think I'll enable it, since I'm the only user.
```

```
> # Default is false. If true, user can create a repo by pushing local to remote (gitea)
```

```
> #ENABLE_PUSH_CREATE_USER = true
```

```
# See below for how to use gitea's built-in secret tool to replace the existing ones.

[security]
< INTERNAL_TOKEN = 1FFhAkIka01JhgJTRUrFujWYiv4ijqcTlfXJ9o4n1fWxz+XVQdXhrqDTIsnD7fvz7g
< SECRET_KEY = ChangeMeBeforeRunning
> INTERNAL_TOKEN = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmYiOiJlOTU2NDA4NjB9.oZEw2...
> SECRET_KEY = qVvCzqg4mqe2tQHmZfE99EvzADFvOMY9fO3BdTFw4vwcBVvfAdyxJyBL9Hg...

[server]
< DOMAIN = localhost
< HTTP_ADDR = 127.0.0.1
< ROOT_URL = http://localhost:3000/
> DOMAIN = gitea.mydomain.tld
> HTTP_ADDR = 10.101.10.150
> ROOT_URL = https://gitea.mydomain.tld:443/ # this is the "https clone address/port"
# Note that internally, it's still listening on port 3000. ^^ that's for the clone button
< SSH_PORT = 22
> SSH_PORT = 40202 # this is the clone port for ssh

> START_SSH_SERVER = true # to make gitea manage ssh connections, instead of the host
> SSH_LISTEN_HOST = 10.101.10.150
> SSH_LISTEN_PORT = 22002 # non-root user can't listen on 22
> LANDING_PAGE = explore # this shows the repos, instead of a gitea advert

# to prevent web registrations

[service]
< DISABLE_REGISTRATION = false
> DISABLE_REGISTRATION = true
```

What is shown above is that the secrets have already been updated. Here's how to do it.

```
sed -i .tmp 's/^JWT_SECRET.*=.*$/JWT_SECRET = `gitea generate secret JWT_SECRET`'/g' \
/usr/local/etc/gitea/conf/app.ini
```

```
sed -i .tmp 's/^INTERNAL_TOKEN.*=.*$/INTERNAL_TOKEN = `gitea generate secret INTERNAL_TOKEN`'/g' \
/usr/local/etc/gitea/conf/app.ini
```

```
sed -i .tmp 's/^SECRET_KEY.*=.*$/SECRET_KEY = `gitea generate secret SECRET_KEY`'/g' \
/usr/local/etc/gitea/conf/app.ini
```

Diff the new with the backup to make sure it looks right.

```
diff /usr/local/etc/gitea/conf/app.ini.bak /usr/local/etc/gitea/conf/app.ini
```

Check file permissions for `/var/log/gitea` and `/var/db/gitea`. You may need to `chown -R git:git`. If it doesn't work, also check `/usr/local/data/git` and ...

And get it running.

```
service gitea start
```

And check the `status`, just to make sure.

## Wrapping up

You're about to update the reverse proxy, so you better have the CNAME record by now.

### Update Caddyfile. (v1)

```
gitea.mydomain.tld {  
    proxy / 10.101.10.150:3000  
}
```

## DigitalOcean firewall

Since we're using a jail, we defined a different SSH port that PF will forward to the jail. We need to allow that port through the DigitalOcean firewall, in the Networking tab.

## PF

```
git_ssh = "40202"  
  
gitea_jail = "10.101.10.150"  
  
rdr pass inet proto tcp from any to any port $git_ssh -> $gitea_jail port 22002
```

As usual, test with `pfctl -vnf /etc/pf.conf`, and then remove `vn` if it's all good.

## Create gitea user

```
su git
```

```
gitea admin create-user --username c00ldude --password 1234superpass \  
--email username@gmailorwhatever.com --admin -c /usr/local/etc/gitea/conf/app.ini
```

Repeat that command if you want to create additional users (because you turned off web registrations).

## Log in to the web interface

You're ready to use the username and password to log in and start creating repos.

## References

Used <https://www.cammack.com/posts/jail-gitea-in-freebsd/> for some help... but it was incomplete...

Helpful stuff here too: <https://docs.gitea.io/en-us/config-cheat-sheet/>



# Website Jail w/ Git Power-up

## Website via git

You created a jail for gitea. Of course you now want to use it to track website changes via git version control. Every `git push` is a push into production, and that's cool! Let's roll with it. The future is now.

Let's pretend you've done some basics. You've got gitea running, and you created a project in gitea called 'website'. You `git` cloned it, and you have `scp`'d the files from your website jail folder to your local computer. You copied them into the repo, and you committed your changes, and you're ready to push your changes, right? Perfect.

Let's get busy on the server...

As root in the host:

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

Now we need to create these directories in their `/usr/local/data`, stop the gitea and website jails, update their `fstab` files, and restart the jails. Then make sure to set permissions (owned by git, by readable by anyone).

## First, with the gitea jail

```
bastille console git_jail
```

```
mkdir -p /usr/local/data/prod-website
```

```
exit
```

```
bastille git_jail stop
```

Edit `fstab`

```
/usr/local/data/prod-website /usr/local/bastille/jails/git_jail/root/usr/local/data/prod-website nullfs rw,late 0 0
```

```
bastille start git_jail
```

```
bastille console git_jail
```

```
cd /usr/local/data/prod-website && mkdir -p mydomain.tld && chown git:git mydomain.tld
```

Before moving along, let's add the git hook.

(This is the magic)

```
cd /usr/local/data/git/git_username/website.git/hooks
```

Edit post-receive to include

```
WEBSITE_FOLDER="/usr/local/data/prod-website/mydomain.tld"  
git --work-tree=$WEBSITE_FOLDER --git-dir=$GIT_DIR checkout -f master
```

Now double check you added those files locally and push to remote.

And it worked.

## Next, with the website jail

```
bastille console website_jail
```

Double check the location of the website. It's at `/usr/local/www/mydomain.tld` ... now...

```
bastille stop website_jail
```

Edit `fstab`

```
/usr/local/data/prod-website/mydomain.tld /usr/local/bastille/jails/website_jail/root/usr/local/www/mydomain.tld  
nullfs rw,late 0 0
```

```
bastille start website_jail
```

If I pop into the jail and run `ll` in `/usr/local/www`, I see that the git user owns the directory now, so it appears it's complete...

But it's not. Nginx is looking too high. Gotta adjust the nginx conf. It needs to dig in another dir (`.../www/mydomain.tld/mydomain.tld`). Maybe I'll decide on a more elegant (less nested) approach

later. For now, it works and is nice.

Then a final `service nginx reload` (preceded by `nginx -t`, if you wanna be extra careful), and we're good.

## Mission Accomplished

That's right. As stated at the top, you can now do development at home, testing on your localhost webserver, and then commit and push your changes whenever you're happy with them.

# IRC!Radio by dsc\_

## IRC!Radio

IRC!Radio is a radio station for IRC channels. You hang around on IRC, adding YouTube songs to the bot, listening to it with all your friends. Great fun!

## Stack

IRC!Radio aims to be minimalistic/small using:

- Python >= 3.7
- SQLite
- LiquidSoap >= 1.4.3
- Icecast2
- Quart web framework

And all in a FreeBSD jail (in this case).

## Command list

```
“ - !np - current song
  - !tune - upvote song
  - !boo - downvote song
  - !request - search and queue a song by title or YouTube id
  - !dj+ - add a YouTube ID to the radiostream
  - !dj- - remove a YouTube ID
  - !ban+ - ban a YouTube ID and/or nickname
  - !ban- - unban a YouTube ID and/or nickname
  - !skip - skips current song
  - !listeners - show current amount of listeners
  - !queue - show queued up music
  - !queue_user - queue a random song by user
  - !search - search for a title
  - !stats - stats
```

# Installation

The following assumes you have a VPS somewhere with root access (duh). It assumes you're using `bastille` for the jail manager, and it assumes you have a `caddy` jail already set up for reverse proxy and certs.

Before doing anything else, since we're on FreeBSD, create a jail. Notice that this is a thin jail with no network interface specified, therefore it'll use the `bastille0` cloned loopback device for its network.

```
bastille create radio_jail 13.0-RELEASE 10.101.10.180
```

In my case, I have a custom `.cshrc` file to make the terminal nicer looking (and a script to copy it into place).

```
/usr/local/scripts/custom_cshrc.sh radio_jail
```

The `radio` user will be doing a bunch of the heavy lifting. It needs its own `/home` directory.

```
bastille cmd radio_jail pw adduser -n radio -m -d /home/radio -s /usr/local/bin/bash -c "radio user"
```

An `icecast` user will be needed to run `icecast`, but it doesn't need its own `/home` directory.

```
bastille cmd radio_jail pw adduser -n icecast -G wheel -d /nonexistent -s /usr/sbin/nologin -c "icecast"
```

## Requirements

### Part I - Everything except for `liquidsoap`, basically

Into the jail, as root:

```
bastille console radio_jail
```

First, might as well get onto the latest package repo.

```
mkdir -p /usr/local/etc/pkg/repos
```

```
echo 'FreeBSD: { url: 'pkg+http://pkg.FreeBSD.org/${ABI}/latest', enabled: yes }' >  
/usr/local/etc/pkg/repos/FreeBSD.conf
```

Do a couple rounds of package installing. First, basics. Then specifics.

```
pkg install -y bat htop git vim-console tmux
```

```
pkg install -y icecast py38-virtualenv libogg nginx ffmpeg sqlite3 py38-sqlite3 gmake bash
```

And because there is no `liquidsoap` in ports:

```
pkg install -y ocaml-opam libmad taglib libsamplerate pkgconf gavl fdk-aac
```

## Part II - Use `opam` to install `liquidsoap`

```
su radio
```

Inside the jail, as the `radio` user, the majority of the rest will happen. Might as well get to the `/home` directory.

```
cd
```

```
opam init
```

Follow the instructions to make sure `.profile` is properly sourced.

```
vim .bashrc
```

And paste:

```
source /usr/home/radio/.profile
```

The compiler in the package repo is too old for what we need.

```
opam switch create 4.12.0
```

```
opam install fdkaac gavl
```

```
opam depext taglib mad lame vorbis cry samplerate liquidsoap
```

And the `install` command that won't work without the `env` vars (whether included in advance or part of the command):

```
C_INCLUDE_PATH=$C_INCLUDE_PATH:/usr/local/include  
CPLUS_INCLUDE_PATH=$CPLUS_INCLUDE_PATH:/usr/local/include LIBRARY_PATH=$LIBRARY_PATH:/usr/local/lib  
opam install taglib mad lame vorbis cry samplerate ffmpeg liquidsoap
```

# Clone and Setup

Still as `radio` user, still from from `~`:

```
git clone https://git.wownero.com/dsc/ircradio.git
```

```
cd ircradio/
```

The magic commands that will need to be run more than once (here, and then farther down, at the end):

```
virtualenv -p /usr/local/bin/python3.8 venv
```

```
source venv/bin/activate
```

```
pip install -r requirements.txt
```

## Adjust settings

Now that all the building blocks are in place:

```
cp settings.py_example settings.py
```

```
vim settings.py
```

Look at `settings.py` and configure it to your liking:

- Change `host` listening address at the top to internal IP given to the jail, `10.101.10.180`
- Change timezone to `America/New_York` or whatever
- Change `irc_host` from `localhost` to something like `irc.oftc.net` or `irc.libera.chat` or whatever
- If you change `irc_ssl` to `True`, change the `irc_port` accordingly.
- Change `irc_nick`, `irc_channels`, `irc_realname`, maybe `irc_command_prefix`
- Change `icecast2_hostname` to your `hostname`, i.e, `radio.example.com`
- Change the passwords under `icecast2_`
- Change the `liquidsoap_description` to whatever

Lastly, edit `ircradio/utls.py`, and comment out all of `liquidsoap_check_symlink()`, and just make it `pass`.

Alternatively, you can run the `generate` command that follows, and then run `find / -type f -name lastfm.liq`, and then as `root` put in a symlink so it will be able to find that file. But you'll need a thick jail to be able to do this. And you'll need to repeat the three magic `virtualenv` commands again.

When you are done, this will generate various initial configs (which we'll have to further edit):

```
python3.8 run.py generate
```

The generate function writes `icecast` / `liquidsoap` / `nginx` configuration files into `data/`.

## Update configs

First, while still in the `radio` user's shell:

```
which liquidsoap
```

Then, `exit` out of `radio` and back to `root`. We'll need `root` shell for this section and the next. And if needed:

```
cd /home/radio/ircradio
```

### liquidsoap

Where is `liquidsoap`? We got that above. That needs to be the path at the top of the `data/soap.liq` file. Paste it.

```
vim data/soap.liq
```

And while in there, comment out the row starting with `full`. In the final line, change `full` to `radio`. This change will remove the crossfade function unfortunately. Maybe 1.4.4 changed that function.

**TODO: figure out crossfading, cuz I want it**

Then `liquidsoap` also needs an `rc` file, rather than a `system.d` file.

```
vim /usr/local/etc/rc.d/liquidsoap
```

```
#!/bin/sh

# PROVIDE: liquidsoap
# REQUIRE: DAEMON
# BEFORE: LOGIN
# KEYWORD: shutdown

# Add the following line to /etc/rc.conf to enable `liquidsoap`.
#
```



```
#liquidsoap_enable="YES"

#
# To specify a non-default script file, set liquidsoap_script
# in /etc/rc.conf:
#
#liquidsoap_script="/home/radio/ircradio/data/soap.liq"
#

. /etc/rc.subr

name="liquidsoap"
rcvar=liquidsoap_enable

#update as necessary, the command path
command="/usr/home/radio/.opam/4.12.0/bin/liquidsoap"
command_args="--daemon 1>/dev/null"
#command_args="--daemon --quiet"
extra_commands="reload"

# read configuration and set defaults
load_rc_config "$name"
: ${liquidsoap_enable="NO"}
: ${liquidsoap_script="/home/radio/ircradio/data/soap.liq"}
: ${liquidsoap_flags="${liquidsoap_script}"}
: ${liquidsoap_user:=radio}
: ${liquidsoap_group:=radio}

required_files="${liquidsoap_script}"

run_rc_command "$1"
```

And it needs to be made executable.

```
pushd /usr/local/etc/rc.d/
```

```
chmod +x liquidsoap
```

```
popd
```

Also, `liquidsoap` will want to create a `pid` near the build `dir`, and the user needs permissions... (adjust as necessary).

```
mkdir -p /usr/home/radio/.opam/4.12.0/lib/liquidsoap/var/run/liquidsoap
```

```
pushd /usr/home/radio/.opam/4.12.0/lib/liquidsoap/var/run
```

```
chown radio:radio liquidsoap/
```

```
popd
```

## nginx

```
vim data/radio_nginx.conf
```

For `data/radio_nginx.conf`, there needs to be the following at the very top:

```
events {}
```

And underneath that, the whole server block needs to be wrapped in an `html {}` block.

And change the listen port to whatever you'll forward to from `caddy`, like `8040`, though `80` should be fine too.

## icecast

Get into `data/icecast.xml`.

```
vim data/icecast.xml
```

First, might as well adjust the location to a fun name and admin to any old email address.

I adjusted `<burst-on-connect>` to `1` and `<hostname>` to `radio.example.come` (the actual address).

The bottom of the file needs to have the user info in the `security` section, right under `changeowner` subsection:

```
<<changeowner>
  <user>icecast</user>
  <group>icecast</group>
</changeowner>
```

Change the paths to these, since the provided ones are for Linux.

```
<paths>
❏<basedir>/usr/local/share/icecast</basedir>
    <logdir>/var/log/icecast/</logdir>
❏<webroot>/usr/local/share/icecast/web</webroot>
❏<adminroot>/usr/local/share/icecast/admin</adminroot>
</paths>
```

When starting the service, there will be an annoying "error" if we don't have this file \*rolls eyes\*...

```
touch /etc/mime.types
```

One more thing for this. For `icecast` to work, it needs to be able to do what you tell it, like logging...

```
pushd /var/log
```

```
mkdir /var/log/icecast
```

```
chown -R icecast:icecast /var/log/icecast
```

```
chmod -R 760 /var/log/icecast
```

```
popd
```

## Final Tidying Up

Still as `root` ...

```
cp /home/radio/ircradio/data/icecast.xml /usr/local/etc/
```

```
cp /home/radio/ircradio/data/radio_nginx.conf /usr/local/etc/nginx/nginx.conf
```

And we can enable the services..

```
sysrc liquidsoap_enable="YES"
```

```
sysrc nginx_enable="YES"
```

```
sysrc icecast_enable="YES"
```

And start them (and ultimately this will hopefully illuminate if any errors were made above).

```
service icecast start
```

```
service liquidsoap start
```

```
service nginx start
```

## Set Up Host & Caddyfile (and cname record)

Hopefully the host doesn't need anything, actually.

Before getting to `caddy`, hop into your domain registrar and add a `cname` for the `hostname` desired, in this case `radio`. It may take a little while for the new record to propagate.

Then hop into the `Caddyfile` and add a section for `radio.domain.tld`, and reverse proxy to the jail and the listen port from the top of `nginx.conf`.

And then we are ready to finish up.

## Start It!

From the jail console, start a new `tmux` session.

```
tmux
```

Change user and get to the repo directory.

```
su radio
```

```
cd
```

```
cd ircradio/
```

Run the three magical `virtualenv` steps.

Run the thing:

```
python3.8 run.py webdev
```

Then hop onto IRC and download a few songs! Then either the music will start playing, or you can restart `liquidsoap` with `root`.

## Other

There are `html` files for the webpage inside `ircradio/templates`. Perhaps you'd like to adjust the files to customize it a bit and maybe indicate that you stole this setup from from someone else and it's really their hard work that made it possible.

## Resources

<https://git.wownero.com/dsc/ircradio>

# More Legwork - Backups

# Backups Overall

We haven't gone into backups yet. Maybe we should. In general, writing a script to dump a database, tar files, etc, is not rocket science (though it's a bit tedious to set up). The trickiest part in any of these is getting the backed-up files to a different machine.

## Host

We actually haven't done a whole heck of a lot to the host. Certainly, we've got an `rc.conf` and a `pf.conf`. We've also got a script and a `cron` entry. We'll be adding more scripts (backup scripts) and more `cron` entries. And we've potentially got data directories on the host as well, considering we'll probably `cp/tar` files into some backup directory first (which we can then ZFS snapshot!) before then `scp`'ing to remote machine.

## Bookstack

This one is semi-tricky, but actually not bad. You must use the correct credentials to dump the database to a file, and you must grab a few additional directories that are resources used by the database but not actually saved inside it.

## Website

If you're using Gitea to populate the website's document root, then perhaps backing up Gitea will suit your needs. And in that case, you also most likely have a local copy of the `git` repository working tree, so you can already repopulate the document root in a pinch.

Whether or not you're using Gitea/`git`, it's generally a simple task to `tar` up the document root directory and `scp` it to a remote machine.

## Gitea

There are several ways to look at backing this up depending on how you're using it.

# Standalone

If you're reliant solely on Gitea, then you should back things up. There are two directories and a file.

## git home

This stores the `*.git` directories containing the commit history and whatnot. It feels weird to `tar` these up when they can easily be cloned by conventional means. Choose your poison.

## Database

This category consists of the db itself as well as some accompanying files and directories. The db contains the website configuration, including users. Backing this up will depend on the type of database. I used `sqlite`, which is not authenticated. The process for `mariadb` and others would be different, though you can follow the procedure used for BookStack, since that uses `mariadb`.

## `{..}/app.ini`

These are the runtime parameters.

# Mirrors

If you're using Gitea to mirror a website from Github or Gitlab (or wherever), then there isn't much of a need for a backup because you can just recreate the Gitea repo from the repo you're mirroring.

If you're using Github (or wherever) to mirror Gitea, then... you still don't have much to worry about. The reason for this (though I don't know if it applies outside of Github) is that Github does not allow you to mirror external repos. So if you're using Github to mirror Gitea, you have accomplished this by adding a post-receive hook to your Gitea repo that pushes the changes automatically to Github. So in this case too, you can restart the Gitea repo by importing the Github repo. However, it may make sense to back up your keys, considering you had to provide authorization for Gitea to push to Github, so there may be an SSH key that you'd want to just put back into Gitea rather than creating a new key pair and loading the newly created pubkey into Github.

# Bitwarden

There's actually not much to this. You can look at their recommendations here:

[https://github.com/dani-garcia/bitwarden\\_rs/wiki/Backing-up-your-vault](https://github.com/dani-garcia/bitwarden_rs/wiki/Backing-up-your-vault) which basically come down



to running `sqlite3 db > backup` and saving your icons.

# Caddy

No backup needed, really. I mean, in its current state, it'd take two seconds to replace. The more you add to it, the more you maybe want to copy a backup of the Caddyfile somewhere, but that's about it.

# BookStack Backup

Here's a script for backing up everything you need in the event you want to rebuild the jail and bring the existing BookStack data to the new jail.

It takes no arguments. You simply set the correct variables inside, and it just works™. Here's how:

1. It checks whether the root backup directory exists. If it doesn't, it creates it.
2. It checks whether the subdirectory (based on the date) exists. If it doesn't, it creates it.
3. It `cd`'s to where you'll temporarily store the database dump (after making sure it exists).
4. It dumps the database into a file of the name you specify (which isn't important).
5. It `cd`'s to the subdirectory where the backup files will be saved.
6. It checks whether the dump file already exists (in case you already backed up today).
  1. If you didn't already back up today, it moves the db dump there.
  2. If you did already back up, it prepends a count to the file name first, as to not overwrite the previous.
7. Next, it `cd`'s into the BookStack directory and `tar`'s the remaining files and directories.
8. Lastly, it copies the `tar`'ed file to the backup subdirectory.

Before implementing this script, we need to set up the backup directory.

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

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

Now we can create the script to run from the host that will dump the db, tar the add'l resources, and save them in the directory we just created. From the host:

```
cd /usr/local/scripts
```

```
vim backup_bookstack.sh
```

```
#!/bin/sh

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

# Variables to be used
```

```
jail=bs_jail
jail_dir="/usr/local/bastille/jails/$jail/root"
db=db_bs
DUMP="$db.dmp"
NOW=$(date +"%Y-%m-%d")
bk_root="/usr/local/data/backups/bookstack"
bk_dir="$bk_root/$NOW"
scripts_dir="/usr/local/scripts"
tmp_dmp_dir="${scripts_dir}/tmp"
bs_dir="${jail_dir}/usr/local/www/bookstack"
bs_files="bookstack-files-backup.tar.gz"

mv_it() {
    FILE=$1
    SOURCE_DIR=$2
    COUNT=$3

    if [ ! -f "$COUNT.${FILE}" ]; then
        cd $SOURCE_DIR
        mv ${FILE} ${bk_dir}/${COUNT}.${FILE}
    else
        COUNT=`expr $COUNT + 1`
        mv_it $FILE $SOURCE_DIR $COUNT
    fi
}

# Create destination root dir and sub dir

if [ ! -d "${bk_root}" ]; then
    mkdir ${bk_root}
fi

if [ ! -d "${bk_dir}" ]; then
    mkdir ${bk_dir}
fi

# Prepare to export; dump to tmp dir

cd "${tmp_dmp_dir}"
```

```
# Within the jail (via bastille), dump MariaDB db to file
# (substitutue 'secret' w/ the password of the backup user) (w/ no space after the -p)

bastille cmd $jail mysqldump --single-transaction -u backup -psecret $db > $DUMP

# Move (and rename) the file from the current dir to the backup dir

COUNT_D=0

cd $bk_dir

if [ ! -f "${DUMP}" ]; then
    cd $tmp_dmp_dir
    mv $DUMP $bk_dir
else
    mv_it $DUMP $tmp_dmp_dir $COUNT_D
fi

# Tar the unrecoverable, install-specific files

cd $bs_dir
tar -czf $bs_files .env public/uploads storage/uploads

# Move to backup dir

COUNT_F=0

cd $bk_dir

if [ ! -f "${bs_files}" ]; then
    cd $bs_dir
    mv $bs_files $bk_dir
else
    mv_it $bs_files $bs_dir $COUNT_F
fi

# Write to log briefly what happened
```

```
echo "$NOW - Dumped $db, tar'ed files, saved to $bk_dir" >> ${scripts_dir}/Scripts.log
```

```
exit 0
```

And it needs to be executable.

```
chmod 755 backup_bookstack.sh
```

Notes:

1. If you named the jail something other than 'bs\_jail' then adjust the variable value.
2. Adjust the db name from 'db\_bs' to whatever your db's name is.
3. I set `bk_root` to what we just created ZFS datasets for.
4. This will be saved in `/usr/local/scripts`. Feel free to save it elsewhere.
5. Create a `/tmp` inside `/usr/local/scripts` (or your chosen dir), to be used temporarily by the script.
6. If your BookStack installation isn't in `/usr/local/www/bookstack`, then adjust accordingly.
7. This is just the name of the resources that get `tar`'ed. Name it whatever you want.

Also:

We're using decent hygiene here. The user that is performing the db dump only has access to perform that one function (pretty much). You must create it and give it that privilege. Like so...

From the host, we log into `mysql` (`mariadb`).

```
bastille cmd bs_jail mysql -u root -p
```

Then we create the user.

```
CREATE USER 'backup'@'localhost' IDENTIFIED BY 'secret';  
GRANT SELECT, SHOW VIEW, RELOAD, REPLICATION CLIENT, EVENT, TRIGGER ON *.* TO 'backup'@'localhost';  
FLUSH PRIVILEGES;  
exit;
```

Now you can run this and see the results. You may want to run a cron job (though bear in mind that a single cron job to back up BookStack will just grow over time, so you should also create a script to only keep the newest so many and purge the rest).



# More Legwork - Upgrading Versions

# Upgrade From 12.1- to 12.2- RELEASE

Did we get a choice to include `/usr/src` when setting up this droplet? I don't remember, but I don't think so. Well, let's just ignore that for the moment and proceed with the upgrade process.

## Pro Tips

1. Before proceeding - even though nothing will go wrong, right? - stop the droplet and take a snapshot.
2. You did the snapshot? Maybe you want to "resize" while you're at it. With 1 CPU and 2gb of RAM, this'll be slow!
3. If you're working with a \$5/mo droplet, plan on this taking 1-2 hours... or more.
4. Consider `tmux` or similar.

## Such Quick! Very Upgrade!

```
freebsd-update fetch
```

```
freebsd-update install
```

```
freebsd-update upgrade -r 12.2-RELEASE
```

## Not so fast!

This is a DO droplet. Maybe your droplet has FreeBSD source on it, but mine doesn't. Let's get it.

```
fetch ftp://ftp.freebsd.org/pub/FreeBSD/releases/amd64/12.1-RELEASE/src.txz
```

And then we must extract it (feel free to pipe `stdout` to `/dev/null` or something).

```
tar -xvzf src.txz -C /
```



# Start over!

```
freebsd-update fetch
```

```
freebsd-update install
```

```
freebsd-update upgrade -r 12.2-RELEASE
```

```
freebsd-update install
```

## Success! Reboot!

```
shutdown -r now
```

## Not done! Upgrade userland!

```
freebsd-update install
```

```
shutdown -r now
```

# Done!

## But are you, really?

### A tale of hurried-ness.

I started the process at 11pm. I went to bed a bit after midnight, and I let it just do its thing. When I woke up, I was presented with a few questions, and I skipped through them too quickly. One of them was to the effect of "there are changes to `/etc/ssh/sshd_config` that cannot be automatically merged. Please edit it vi."

I stared at the message for a few seconds, thought about how I needed to walk and feed the dog, and decided I would deal with it later. I just saved it with the diffs (you know, `>>>>>>>>>>>>` and `<<<<<<<<<<<<`) still in it. I answered 'yes' to the remaining questions of "does this look right?" and it proceeded to finish the system upgrade.

I then handled my dog duties and came back to finish up. I rebooted, and then upgraded userland. I had forgotten all about the `sshd_config` deal. Well, after the second reboot (first one

after kernel/system, and second one after userland), I was rejected at port 22. Ah - no big deal - I'll just access the console using the DO web interface. Just need to enter username and password. Wait. I didn't set up a user, and I didn't set up a root password. So that won't work!

I couldn't figure out how to proceed, so I popped into `#freebsd` on Freenode, and they were like "can't you boot into single user mode?" Hm. Let's see.

You can boot into single user mode one of two ways. First, you can just tell the system to do that on next boot with `nextboot -o "-s" -k kernel`, but I had no shell. The other way is to power cycle the system from the DO web interface, and then open up the console and choose option '2' to boot into single user mode.

Perfect, I have no idea what I need to fix, but at least I'm in. Let's check `.ssh/authorized_keys`. Yeah, looks fine. Hmm. Wait a minute. Yes, I remember now. I didn't bother to do anything about the `sshd_config` diffs that I was bugged about first thing in the morning. Wow, I'm lucky I remembered. Okay, so I'll just open up `vi` and at least comment out the offending rows.

Uh, no, I won't. It's mounted as read-only. But wait, yes I can. I just need to mount it as read-write. I mounted the root file system like so: `mount -u -o rw /`, but others later said `mount -u /` is enough (see reference). Anyhoo, I was then able to fix up `sshd_config` and `service sshd restart` which worked great. Of course, I was in single user mode, so I had to reboot. A minute later and I was able to `ssh` into the server again. Sweet. Finally done.

## Still Not Done!

Well, maybe it is done. But now is a good time to review the changes. Also, do you have any jails? I do, and they have not been upgraded. If you use a jail manager to create and manage your jails, you should also use it to upgrade them. In the Bastille docs (below, in References), they mention how to upgrade individual jails or simply every jail at once by upgrading the release.

## References

Release notes: <https://www.freebsd.org/releases/12.2R/relnotes.html#upgrade>

Updates per the handbook: [https://www.freebsd.org/doc/en\\_US.ISO8859-1/books/handbook/updating-upgrading-freebsdupdate.html](https://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/updating-upgrading-freebsdupdate.html)

Info reworded from a... Linux... website: <https://www.fosslinux.com/44135/freebsd-12-2-review-what-you-need-to-know-and-how-to-upgrade.htm>

Downloading source: <https://www.jan0sch.de/post/install-freebsd-sources/>

Change mount in place: <https://unix.stackexchange.com/questions/65523/unable-to-write-to-file-on-freebsd-read-only-filesystem>

BastilleBSD jail upgrades: <https://github.com/BastilleBSD/bastille>