

# More Fun Stuff

- [Bitwarden-rs Jail](#)
- [Gitea Jail](#)
- [Website Jail w/ Git Power-up](#)
- [IRC!Radio by dsc](#)

# 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 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

ln -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\\_bitwarde](https://www.reddit.com/r/Bitwarden/comments/dg78bi/building_selfhosted_bitwarden_via_bitwarde)

[n\\_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 = 1FFhAkIka01JhgJTRUrFujWYiv4ijqcTIfXJ9o4n1fWxz+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/utils.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>