Fun Stuff

- BookStack Jail
- Website Jail
- Caddy Jail

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 is 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-pecl-imagick 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 buy 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 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.