

Server Provisioning

- [Citadel](#)
- [Bookstack](#)
- [Java JAR Command](#)
- [systemd Timer](#)
- [Bookstack Backup](#)
- [Citadel Backup](#)
- [Full Server Backup](#)
- [Tox Bootstrap Node on Debian](#)
- [Nginx Reverse UDP & TCP Proxy](#)
- [Install Git via PowerShell](#)
- [Create a flash drive image](#)
- [Resize a flash drive image](#)
- [BibleSD Duplication](#)
- [Partial microSD Image](#)
- [Installing BibleSuperSearch on Debian](#)

Citadel

Install Citadel:

```
curl https://easyinstall.citadel.org/install | bash
```

Answer questions during automated install:

- admin: ##### (admin for webcit)
- admin password: ##### (admin webcit password)
- user: ##### (internal use citadel user)
- port: 504 (internal use citadel port)
- http: 4480 (to keep behind VPN)
- https: 4434

Open ports to VPN only (for now):

```
sudo ufw allow in on <vpn_interface> to any port 4480  
sudo ufw allow in on <vpn_interface> to any port 4434
```

WebCit configuration:

- Login via HTTP over VPN (browser, 4480)
- Reduce privileges of default registered user
- Set up accounts and mail forwarding
- Configure `site configuration` > `fully qualified domain name` and `node name` to `<domain_name>`
- Configure `domain names` > `local host aliases` to receive email to `<domain_name>` and `mail.<domain_name>`

Symbolic links to Let's Encrypt certificate:

```
ln -sfv /etc/letsencrypt/live/wilsons.life/privkey.pem /usr/local/citadel/keys/citadel.key
ln -sfv /etc/letsencrypt/live/wilsons.life/fullchain.pem /usr/local/citadel/keys/citadel.cer
```

Configure nginx reverse proxy:

```
location /citadel/ {
    proxy_set_header Host $host;
    proxy_pass http://127.0.0.1:4480/;
    proxy_redirect off;
}

location /static/ {
    proxy_set_header Host $host;
    proxy_pass http://127.0.0.1:4480/static/;
    proxy_redirect off;
}
```

Open external ports:

```
sudo ufw allow <port_to_open>/<protocol>
```

Double check with `sudo netstat -tunlp` to make sure citadel is serving on all ports before opening them. Sometimes it takes a `sudo systemctl restart citadel` to get it going.

SMTP:

25/tcp
465/tcp
587/tcp

IMAP:

143/tcp
993/tcp

XMPP:

5222

Bookstack

Installing Bookstack:

<https://jardin.icamole.fr/books/bookstack/page/installation>

```
note: corrected install command at php8.2: apt-get install php8.2 php8.2-xml libapache2-  
mod-php8.2 php8.2-fpm php8.2-curl php8.2-mbstring php8.2-ldap php8.2-tidy php8.2-zip php8.2-  
gd php8.2-mysql git
```

Change apache2 to listen on port 5080 (<IfModule ssl_module> 5443) in the following files:

```
nano /etc/apache2/ports.conf
```

```
nano /etc/apache2/sites-enabled/000-default.conf
```

Disable 000-default.conf with `/usr/sbin/a2dissite 000-default.conf`

Configure `nano /var/www/bookstack/.env`:

```
APP_URL=https://wilsons.life/bookstack/

Mail settings:
# Mail system to use
# Can be 'smtp' or 'sendmail'
MAIL_DRIVER=smtp

# Mail sender details
MAIL_FROM_NAME="BookStack"
MAIL_FROM=bookstack@wilsons.life

# SMTP mail options
# These settings can be checked using the "Send a Test Email"
# feature found in the "Settings > Maintenance" area of the system.
MAIL_HOST=mail.wilsons.life
MAIL_PORT=587
MAIL_USERNAME=bookstack
MAIL_PASSWORD=<mail_account_password>
MAIL_ENCRYPTION=tls
```

Update nginx reverse proxy:

```
location /bookstack/ {  
    proxy_set_header Host $host;  
    proxy_pass http://127.0.0.1:5080/;  
    proxy_redirect off;  
}
```

Also added the same reverse proxy settings for location /, so the bookstack becomes the default site

Java JAR Command

Creating a JAR

The following **c**reates a jar **f**ile and sets the **e**ntry point in `Manifest.txt`:

```
jar cfe my-app.jar package1.MainClass package1/*.class package2/*.class
```

Reference: <https://docs.oracle.com/javase/tutorial/deployment/jar/appman.html>

Be sure you are in the correct classpath directory (above your package directory). This is the same directory level where you can compile `.java` files and execute `.class` files:

```
javac package1/MyClass.java  
java package1.MyClass <arg0> <arg1> .. <argN>
```

Executing a JAR

The following command executes a compiled JAR file:

```
java -jar my-app.jar <arg0> <arg1> .. <argN>
```

systemd Timer

As a modern `systemd` replacement for `chron`, the following is an example of `oneshot` **Unit** triggered periodically by a **Timer**.

Reference: <https://www.buggycoder.com/network-backups-using-rsync/>

Some people dislike `systemd`, but on a modern linux distro, it's the best way to keep everything as "stock" and "plain vanilla" as possible.

Example `oneshot` **Unit** file:

```
/etc/systemd/system/<example-unit.service>
```

```
[Unit]
Description= <Example Unit Name>

#target requirements (may vary)
Requires=network.target
After=network.target


[Service]
Type=oneshot

#optional settings
Nice=19
StandardOutput=journal
IOSchedulingClass=best-effort
IOSchedulingPriority=5

ExecStart= <command> or </absolute/path/to/exec>


[Install]
WantedBy=multi-user.target
```

Example **Timer** file to execute the **Unit**:


```
/etc/systemd/system/<example-unit.timer>
```

```
[Unit]
Description= <Example Timer Name>
Requires=<example-unit>.service

[Timer]
OnCalendar=daily
Unit=<example-unit>.service

[Install]
WantedBy=timers.target
```

Enable the **Unit** and **Timer**:

```
systemctl daemon-reload

systemctl enable <example-unit>.service
systemctl enable <example-unit>.timer

systemctl start <example-unit>.service
systemctl start <example-unit>.timer

# Ensure all is well
journalctl -f -u <example-unit>.service
```

Bookstack Backup

Example shell script to backup Bookstack content:

```
# must be run as root
cd /var/www/bookstack
mysqldump -u root bookstack > bookstack-$(date +%Y-%m-%d).sql
git add -A && git commit -m "backup" && git push
```

Assumes:

- The MariaDB (or MySQL) database is named "bookstack"
- `/var/www/bookstack` is the bookstack root directory
- The bookstack root directory has been initialized as a `git` repository
- A `git` `remote` has been set

Citadel Backup

Example backup script:

```
# must be run as root
cd /usr/local/citadel
git add -A && git commit -m "backup" && git push
```

Assumes:

- `/usr/local/citadel` has been initialized as a `git` repo and a `remote` has been set

Full Server Backup

Using **dd**:

Reference links:

https://wiki.archlinux.org/title/Rsync#Full_system_backup

<https://unix.stackexchange.com/questions/132797/how-to-dd-a-remote-disk-using-ssh-on-local-machine-and-save-to-a-local-disk>

<https://askubuntu.com/questions/890497/clone-a-remote-server-to-local-machine-via-ssh>

What worked:

Run on my remote VPS, connecting back to local machine via my VPN:

```
sudo <something> # type in password, so you won't have to in next command  
sudo dd bs=4M status=progress if=/dev/sda | gzip -1 - | ssh <local>@<vpn_ip> dd  
of=backup.img.gz
```

The default local save location is into the home directory.

Tox Bootstrap Node on Debian

*These steps have been tested on **Debian 11***

References:

- <https://wiki.tox.chat/users/runningnodes>
- https://github.com/TokTok/c-toxcore/tree/master/other/bootstrap_daemon

Install Dependencies:

- `git` (to clone the repo)
- `libc6` (includes `libm`, `libthread`, `librt`)
- `libconfig`
- `cmake`
- `libnacl` (might not be necessary if `libsodium` is installed?)
- `libsodium` (I still had to install this, even with `libnacl` installed...)

```
sudo apt update
sudo apt install git libc6-dev cmake libconfig-dev libnacl-dev libsodium-dev
```

Clone latest Git repository & submodules:

- GitHub: [TokTok/c-toxcore](https://github.com/TokTok/c-toxcore)

In your home directory or wherever you prefer to do compilation work:

```
git clone https://github.com/TokTok/c-toxcore.git
cd c-toxcore
git submodule update --init
```

Compile `libtoxcore` and `tox-bootstrapd`:

In the `c-toxcore` directory (you might be in it from the previous step):

```
mkdir _build
cd _build
cmake ..
make
make install
```

Did it Compile OK?

Verify the output text from `make` is all **green** and `tox-bootstrapd` was built:

```
[100%] Linking C executable tox-bootstrapd
[100%] Built target tox-bootstrapd
```

Create a `tox-bootstrapd` User & Restricted Home Directory:

```
sudo useradd --home-dir /var/lib/tox-bootstrapd --create-home --system --shell /sbin/nologin
--comment "Account to run Tox's DHT bootstrap daemon" --user-group tox-bootstrapd
sudo chmod 700 /var/lib/tox-bootstrapd
```

Service & Configuration Files:

In the `c-toxcore` directory (`cd ..` if you're still in `_build`), copy the service and conf files to the system locations. Keep in mind there is a directory structure under `c-toxcore` that is similar to the structure under `_build`, but the **service** and **conf** files are found under `c-toxcore/other/bootstrap_daemon/` and the **executable** is found under `c-toxcore/_build/other/bootstrap_daemon/`.

Copy the **service file** to `/etc/systemd/system/`:

```
sudo cp other/bootstrap_daemon/tox-bootstrapd.service /etc/systemd/system/
```

If you will be using the default port number or a number greater than `1023`, you can ***simply copy the service file and leave it as it is***. Otherwise, you will need to un-comment the line `#CapabilityBoundingSet=CAP_NET_BIND_SERVICE`.

Copy the **configuration file** to `/etc/`:

```
sudo cp other/bootstrap_daemon/tox-bootstrapd.conf /etc/tox-bootstrapd.conf
```

Customize Configuration Settings:

Use `nano` or your favorite editor to edit `/etc/tox-bootstrapd.conf`:

```
sudo nano /etc/tox-bootstrapd.conf
```

At a minimum, edit your Message of the Day (**MOTD**), other **bootstrap nodes**, and probably also your **ports**. A current list of public bootstrap nodes is displayed at <https://nodes.tox.chat/>.

```
port = 33445

keys_file_path = "/var/lib/tox-bootstrapd/keys"

pid_file_path = "/var/run/tox-bootstrapd/tox-bootstrapd.pid"

enable_ipv6 = true

enable_ipv4_fallback = true

enable_lan_discovery = true

enable_tcp_relay = true

tcp_relay_ports = [ 443, 3389, 33445, 43334]

enable_motd = true

motd = "Write Your Custom MOTD Here! (up to 255 chars)"

bootstrap_nodes = (
    { // Tony (he's awesome)
        address = "tox.abilinski.com"
        port = 33445
        public_key = "10C00EB250C3233E343E2AEBA07115A5C28920E9C8D29492F6D00B29049EDC7E"
    },
    { // Cody (he's awesome too)
        address = "198.199.98.108"
        port = 33445
        public_key = "BEF0CFB37AF874BD17B9A8F9FE64C75521DB95A37D33C5BDB00E9CF58659C04F"
    },
    { // Gabe (he's just a geek)
        address = "104.225.141.59"
        port = 43334
        public_key = "933BA20B2E258B4C0D475B6DECE90C7E827FE83EFA9655414E7841251B19A72C"
```

```
}  
)
```

Install the `tox-bootstrapd` Executable:

Copy the tox-bootstrapd executable from `_build/other/bootstrap_daemon/` to `/usr/local/bin/`:

```
sudo cp _build/other/bootstrap_daemon/tox-bootstrapd /usr/local/bin/tox-bootstrapd
```

Enable & Start the `systemd` Service:

```
sudo systemctl daemon-reload  
sudo systemctl enable tox-bootstrapd.service  
sudo systemctl start tox-bootstrapd.service  
sudo systemctl status tox-bootstrapd.service
```

Did the Service Start OK?

If `systemctl start` didn't produce any text and `systemctl status` shows **active (running)**, then you should be up and running!

Otherwise, scroll down/right (arrow keys) through the output text under `systemctl status` or try some of [these helpful troubleshooting tips](#).

You can see a list of services that are listening on ports with:

```
sudo netstat -tunlp
```

Bootstrap Node Public Key:

Your public key should be listed near the bottom of the log entries:

```
sudo grep "tox-bootstrapd" /var/log/syslog
```

Enable External Ports:

You may need to open ports listed in `/etc/tox-bootstrapd.conf`:

```
sudo ufw allow 33445  
sudo ufw allow 443/tcp  
sudo ufw allow 3389/tcp
```


If you're not using `ufw`, you probably should be:

```
sudo apt update
sudo apt install ufw
sudo ufw allow ssh
sudo ufw enable
```

You can limit a port to a specific network interface (e.g. your VPN tunnel) using:

```
sudo ufw allow in on <interface> to any port <number>
```

Test your Bootstrap Node:

You can test outside connectivity to your node here:

<https://nodes.tox.chat/test>

Nginx Reverse Proxy:

In case this is useful, here's some information on configuring nginx as a reverse UDP & TCP proxy:

<https://wilsons.life/bookstack/books/server-provisioning/page/nginx-reverse-udp-tcp-proxy>

That's all! Feel free to send me a message via Tox at:

```
CD9E37503A5B2DFB41947B9A0E4B921381340B49FC318FEB07250789C715DA3470885905869F
```

Nginx Reverse UDP & TCP Proxy

Reference:

- <https://docs.nginx.com/nginx/admin-guide/load-balancer/tcp-udp-load-balancer/>

Nginx Reverse Proxy:

Maybe you have Nginx as a front-end server on your cloud VPS, and you prefer to run back-end services on your own hardware, tunneled over a VPN. The following can be added to the top-level nginx configuration (probably `/etc/nginx/nginx.conf`) to enable UDP and TCP reverse proxying to your back-end service:

```
stream {  
  
    server {  
        listen      12345;  
        #TCP traffic will be forwarded  
        proxy_pass  <IP_or_domain>:12345;  
    }  
  
    server {  
        listen      4444;  
        #TCP traffic will be forwarded  
        proxy_pass  <IP_or_domain>: 4444;  
    }  
  
    server {  
        listen      4444 udp;  
        #UDP traffic will be forwarded  
        proxy_pass  <IP_or_domain>: 4444;  
    }  
}
```

Be sure the `stream { ... }` block is outside of the top-level `http { ... }` block.

Install Git via PowerShell

It's easy!

Run:

```
winget install --id Git.Git -e --source winget
```

Then you'll see something like:

Even though it says `The installer will request to run as administrator, expect a prompt.`, I was not required to enter an admin password, which is nice on an IT administered work laptop.

After installation, **close the PowerShell window!**

Open another PowerShell window and you should be able to check your **git** version:

```
git --version
```

You should see:

```
git version 2.42.0.windows.1
```

You're good to go!

Create a flash drive image

Reference: <https://superuser.com/questions/668485/creating-a-fat-file-system-and-save-it-into-a-file-in-gnu-linux>

Create a blank file:

```
dd if=/dev/zero of=test-disk.img bs=1024 count=SIZE status=progress  
# size = SIZE*bs
```

Format as FAT32:

```
mkfs.vfat test-disk.img
```

Mount:

```
sudo mkdir /mnt/test-dir  
sudo mount -o loop test-disk.img /mnt/test-dir/
```

Copy files:

```
sudo rsync -rv --ignore-existing some_directory/* /mnt/test-disk/
```

Unmount:

```
sudo umount /mnt/test-disk  
sudo rmdir /mnt/test-disk
```

Clone to flash drive:

```
dd if=test-disk.img of=/dev/mmcblk0 bs=4M status=progress
```

Resize a flash drive image

Reference: <https://askubuntu.com/questions/1174487/re-size-the-img-for-smaller-sd-card-how-to-shrink-a-bootable-sd-card-image>

Get a loop device:

Question: can I use the loop device that `mount -o loop` assigned when I mounted the image?
Answer: you will need to unmount first, so you might need a new loop device. GParted can open a loop device before unmounting, but cannot resize a mounted partition. After unmounting, use `losetup` with an available loop device.

```
# get a loop device number
sudo losetup -f
# in this case: /dev/loop15
# attach the disk image to the loop device
sudo losetup /dev/loop15 test-disk.img
```

Use GParted with the device:

```
sudo gparted /dev/loop15
```

Right-click on the partition and select **Resize/Move**, and enter a new size.

Right-click and select **Information**. According to GParted (for this example), there is **980.47 MiB Unallocated** currently.

After completing resize, unload the loop device:

```
sudo losetup -d /dev/loop15
```

Use fdisk to get number of **sectors** and **unit** (or **block**) size:

```
fdisk -l test-disk.img
```

Disk /dev/loop15: 29.28 GiB, 31436800000 bytes, 61400000 sectors

Units: sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disklabel type: dos

Disk identifier: 0x00000000

61400000*512 bytes to GiB = 29.2778015 gibibytes

61400000*512 bytes to MiB = 29 980.4688 mebibytes

So we need to truncate up to 980 MiB (in this case, from GParted info) from the end of the file.

60000000*512 bytes to MiB = 29 296.875 mebibytes, which should work well in this case.

Now we use truncate to shorten the file:

```
truncate --size=$(( 60000000+1 ) * 512) test-disk.img
```

Now, might want to check/repair with GParted:

```
sudo losetup /dev/loop15 test-disk.img  
sudo gparted /dev/loop15
```

BibleSD Duplication

The simplest (but longest) method to duplicate this BibleSD is to simply select the files through the graphical interface, copy them onto your computer, and then onto new media. This approach works on all operating systems.

A faster method to duplicate this BibleSD is to copy it as one image file to your computer, and then copy that one image file onto a new microSD card or other flash media. On Mac or Linux this can be accomplished without additional software, using only system commands. On Windows or other operation systems, this requires software like [Etcher](#).

Find the device name and mount point

Open a new **Terminal** window (CTRL+ALT+T on Ubuntu-based Linux distributions).

Type one of the following commands and press Enter:

Mac	Linux
<code>diskutil list</code>	<code>df -h</code>

This command displays a table of mounted disks and media.

For Mac, we will need the **IDENTIFIER** name (example: `disk1`). For Linux, we will need the **Filesystem** name (example: `/dev/mmcblk0`) and the location **Mounted On** (example: `/media/giw/DCA2-3AA0`).

To unmount the

Wait to connect the microSD card or other flash media that will receive the image file (or disconnect if already connected).

Type this command into the **Terminal** and press the **Enter** key:

```
ls /dev
```

This command will display a list of devices connected to your computer.

Connect the microSD card or flash media to your computer and then repeat the same command. You can type it again or press the UP arrow key to cycle through previous commands and then press the **Enter** key.

Compare the difference between the two lists of devices to determine which device was connected. On Linux, the name of the microSD card or flash media will probably be `mmcblk0`, but may be different in some cases.

Unmount

Write the image file to the flash media

Partial microSD Image

Instructions to partially clone the image from a microSD (or anything else), where the filesystem does not cover the entire disk. For example, a 128GB microSD with a fresh RasPi image that only encompasses ~5.4 GB of space.

Run `sudo fdisk -l` to display partitioning details:

```
Disk /dev/mmcblk0: 119.38 GiB, 128177930240 bytes, 250347520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x1aa4489a
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/mmcblk0p1		8192	1056767	1048576	512M	c	W95 FAT32 (LBA)
/dev/mmcblk0p2		1056768	10600447	9543680	4.6G	83	Linux

In this case, sector size of `512 bytes` multiplied by the **End** sector is the total space used by these partitions.

`10600447 * 512 bytes = 5.4GB`

So divide that total size by whatever block size is preferred to use with **dd**, and round up to an integer:

`(10600447 * 512 bytes)/4MB < 1400`

Finally, run **dd** with **count**:

```
sudo dd if=/dev/mmcblk0 of=partial_image.img bs=4M status=progress count=1400
```

Check the image with **gparted**:

```
sudo gparted partial_image.img
```

Or, mount the partitions in the image using [this technique](#):

```
$ sudo mount -o loop,offset=4194304,sizelimit=536870912 ./partial_image.img  
/mnt/mount_point/mount_A  
# offset: 512 bytes * 8192 sectors = 4194304 bytes  
# sizelimit: 512 bytes * 1048576 sectors = 536870912 bytes  
  
$ sudo mount -o loop,offset=541065216,sizelimit=4886364160 ./partial_image.img  
/mnt/mount_point/mount_B  
# offset: 512 bytes * 1056768 sectors = 541065216 bytes  
# sizelimit: 512 bytes * 9543680 sectors = 4886364160 bytes
```

If it's just a single partition image, then you can get away with simply this:

```
sudo mount -o loop partial_image.img /mnt/mount_point/
```

Apparently you can also use **kpartx**:

```
$ sudo kpartx -a partial_image.img  
$ sudo mount -o loop /dev/mapper/loop0p2 /mnt/mount_point  
  
# after you're done:  
$ sudo kpartx -d partial_image.img
```

Installing BibleSuperSearch on Debian

Basic software

- **Apache2**
- **MariaDB**

```
sudo apt update && sudo apt install apache2 mariadb-server
```

Laravel dependencies

Reference: <https://shape.host/resources/laravel-setup-debian-12-tutorial>

Install Laravel dependencies:

```
sudo apt install php php-curl php-bcmath php-json php-mysql php-mbstring php-xml php-tokenizer  
php-zip php-gd php-sqlite3
```

Bible SuperSearch source

Install Bible SuperSearch source files:

- API: <https://www.biblesupersearch.com/downloads/>
- Client: <https://sourceforge.net/projects/biblesuper/files/>

Extract and copy to:

```
/var/www/html/biblesupersearch_api/
```

```
/var/www/html/biblesupersearch_client/
```

This assumes you're using the default site, otherwise copy into the directory for whichever Apache 'site' is enabled.

Configure Apache

Edit **apache.conf**:

```
sudo nano /etc/apache2/apache2.conf
```

AllowOverride **All**:

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride All # change from None to All
    Require all granted
</Directory>
```

Configure **PHP**:

```
sudo a2enmod php8.2 # change 8.2 to your version
sudo a2enmod rewrite
sudo systemctl restart apache2
```

Restart Apache:

```
sudo systemctl restart apache2
```

Configure MariaDB

References:

- <https://www.digitalocean.com/community/tutorials/how-to-install-mariadb-on-debian-11>
- <https://raspberrypi.com/install-mariadb-raspberry-pi/>

Start the MariaDB installation:

```
sudo mysql_secure_installation
```

Answer the following prompts:

- **Enter current password for root (enter for none):** Leave blank, press enter
- **Switch to unix_socket authentication [Y/n] n** Not necessary, enter **n**
- **Change the root password? [Y/n] n** Should not change, enter **n**
- Enter **Y** (default) for all remaining questions

Start the DB prompt:

```
sudo mariadb
```

Create an admin account (set my_admin_password):

```
GRANT ALL ON *.* TO 'admin'@'localhost' IDENTIFIED BY '<my_admin_password>' WITH GRANT OPTION;
```

Create a **biblesupersearch** user (set my_password):

```
CREATE USER 'biblesupersearch'@'localhost' IDENTIFIED BY '<my_password>';
```

Create a **biblesupersearch** database:

```
CREATE DATABASE biblesupersearch;
```

Grant privileges:

```
GRANT ALL PRIVILEGES ON biblesupersearch.* TO 'biblesupersearch'@'localhost';
```

Flush privileges:

```
FLUSH PRIVILEGES;
```

Exit the DB prompt:

```
exit
```

Configure Bible SuperSearch

Change directory to the root API directory:

```
cd /var/www/html
```

Copy .env.example to .env and edit:

```
sudo cp .env.example .env  
sudo nano .env
```

Update the following lines:

```
APP_URL=http://bible.local/biblesupersearch_api/ # or whatever your root path is
DB_DATABASE=biblesupersearch
DB_USERNAME=biblesupersearch
DB_PASSWORD=<my_password> # whatever database password was configured
```

Change ownership recursively to the httpd user (www-data) for biblesupersearch_api and biblesupersearch_client directories:

```
sudo chown -R www-data biblesupersearch_*
```

Bible SuperSearch admin console

Open the `/public` directory in a browser (relative to whatever the root path is):

http://bible.local/biblesupersearch_api/public

If you get a 404 Not Found error, try appending `/index.php`. If that appears to work, then Apache is not rewriting Laravel routes correctly. This will need to be troubleshot and fixed before the admin console will work correctly.

- Under the **Bibles** tab, select 100 rows to be displayed (drop-down at bottom), select all (box at upper left corner), and click **Install**.
- Under the **Options** tab, set the Application URL to http://bible.local/biblesupersearch_api (whatever the base URL should be).

Bible SuperSearch client

In the `biblesupersearch_client` directory, copy `config-example.js` to `config.js` and edit the following in **config.js**:

```
"apiUrl": "http://bible.local/biblesupersearch_api/public",
```

As a bug-workaround for Linux web clients, change `client.os.toLowerCase()` to `client.os` in **biblesupersearch.js**

At this point you should be able to open http://bible.local/biblesupersearch_client/ and see a functioning example!