nginx

nginx (pronounced "engine X"), is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server, written by Igor Sysoev in 2005. nginx is well known for its stability, rich feature set, simple configuration, and low resource consumption.

This article describes how to set up nginx and how to optionally integrate it with **PHP** via **#FastCGI**.

Contents

Installation

Install the package **nginx-mainline** (mainline branch: new features, updates, bugfixes) or **nginx** (stable branch: major bugfixes only).

Using the mainline branch is recommended. The main reason to use the stable branch is that you are concerned about possible impacts of new features, such as incompatibility with third-party modules or the inadvertent introduction of bugs in new features **[1]**.

> **Note:** All nginx modules available in the **official repositories** require the *nginx* package (as opposed to *nginx-mainline*) as a dependency. It may be wise to review the list of modules for any you might need/want before making the *nginx* vs *nginx-mainline* decision. Modules for *nginx-mainline* can be found in the **Arch User Repository**.

For a chroot-based installation for additional security, see **#Installation in a chroot**.

Running

Start/enable `nginx.service`.

The default page served at **http://127.0.0.1** is `/usr/share/nginx/html/index.html`.

Configuration

First steps with nginx are described in the **Beginner's Guide**. You can modify the configuration by editing the files in `/etc/nginx/` The main configuration file is located at `/etc/nginx/nginx.conf`.

More details and examples can be found in **http://wiki.nginx.org/Configuration** and the **official documentation**.

The examples below cover the most common use cases. It is assumed that you use the default location for documents (`/usr/share/nginx/html`). If that is not the case, substitute your path instead.

Configuration example

/etc/nginx/nginx.conf

```
user http;

# May be equal to `grep processor /proc/cpuinfo | wc -l`
worker_processes auto;
worker_cpu_affinity auto;

# PCRE JIT can speed up processing of regular expressions significantly.
pcre_jit on;

events {
    # Should be equal to `ulimit -n`
    worker_connections 1024;

    # Let each process accept multiple connections.
    multi_accept on;

    # Preferred connection method for newer linux versions.
    use epoll;
}

http {
    server_tokens off; # Disables the “Server” response header
    charset utf-8;

    # Sendfile copies data between one FD and other from within the kernel.
    # More efficient than read() + write(), since the requires transferring
    # data to and from the user space.
    sendfile on;

    # Tcp_nopush causes nginx to attempt to send its HTTP response head in one
    # packet, instead of using partial frames. This is useful for prepending
    # headers before calling sendfile, or for throughput optimization.
    tcp_nopush on;

    # Don't buffer data-sends (disable Nagle algorithm). Good for sending
    # frequent small bursts of data in real time.
    #
    tcp_nodelay on;

    # On Linux, AIO can be used starting from kernel version 2.6.22.
    # It is necessary to enable directio, or otherwise reading will be blocking.
    # aio threads;
    # aio_write on;
    # directio 8m;

    # Caches information about open FDs, freqently accessed files.
    # open_file_cache max=200000 inactive=20s;
    # open_file_cache_valid 60s;
    # open_file_cache_min_uses 2;
    # open_file_cache_errors on;

    # http://nginx.org/en/docs/hash.html
    types_hash_max_size 4096;
    include mime.types;
    default_type application/octet-stream;

    # Logging Settings
    access_log off;

    # Gzip Settings
    gzip on;
    gzip_comp_level 6;
    gzip_min_length 500;
    gzip_proxied expired no-cache no-store private auth;
    gzip_vary on;
    gzip_disable "MSIE [1-6]\.";
    gzip_types
        application/atom+xml
        application/javascript
        application/json
        application/ld+json
        application/manifest+json
        application/rss+xml
        application/vnd.geo+json
        application/vnd.ms-fontobject
        application/x-font-ttf
        application/x-web-app-manifest+json
        application/xhtml+xml
        application/xml
        font/opentype
        image/bmp
        image/svg+xml
        image/x-icon
        text/cache-manifest
        text/css
        text/plain
        text/vcard
        text/vnd.rim.location.xloc
        text/vtt
        text/x-component
        text/x-cross-domain-policy;

    # index index.php index.html index.htm;
    include sites-enabled/*; # See Server blocks
}
```

General configuration

Processes and connections

You should choose a fitting value for `worker_processes`. This setting ultimately defines how many connections nginx will accept and how many processors it will be able to make use of. Generally, making it the number of hardware threads in your system is a good start. Alternatively, `worker_processes` accepts the `auto` value since versions 1.3.8 and 1.2.5, which will try to autodetect the optimal value (**source**).

The maximum connections nginx will accept is given by `max_clients = worker_processes * worker_connections`.

Running under different user

By default, **nginx** runs the master process as `root` and worker processes as user `http`. To run worker processes as another user, change the `user` directive in `nginx.conf`:

/etc/nginx/nginx.conf

```
user user [group];
```

If the group is omitted, a group whose name equals that of *user* is used.

Tip: It is also possible to run nginx without anything running as `root` using **systemd**. See **#Running unprivileged using systemd**.

Server blocks

It is possible to serve multiple domains using `server` blocks. These are comparable to "VirtualHosts" in **Apache**. Also see the **upstream examples**.

In the example below the server listens for incoming connections on IPv4 and IPv6 ports 80 for two domains, `domainname1.dom` and `domainname2.dom`:

/etc/nginx/nginx.conf

```
...
server {
        listen 80;
        listen [::]:80;
        server_name domainname1.dom;
        root /usr/share/nginx/domainname1.dom/html;
        location / {
            index index.php index.html index.htm;
        }
}

server {
        listen 80;
        listen [::]:80;
        server_name domainname2.dom;
        root /usr/share/nginx/domainname2.dom/html;
        ...
}
...
```

Restart `nginx.service` to apply any changes.

Make sure the hostnames are resolvable by setting up a DNS-server like **BIND** or **dnsmasq**, or have a look at **Network configuration#Local network hostname resolution**.

Managing server entries

It is possible to put different `server` blocks in different files. This allows you to easily enable or disable certain sites.

Create the following directories:

```
# mkdir /etc/nginx/sites-available
# mkdir /etc/nginx/sites-enabled
```

Create a file inside the `sites-available` directory that contains one or more server blocks:

/etc/nginx/sites-available/example

```
server {
    ..
}
```

Append the following line at the end of the `http` block in `/etc/nginx/nginx.conf`:

```
include sites-enabled/*;
```

To enable a `server` block, simple create a symlink:

```
# ln -s /etc/nginx/sites-available/example /etc/nginx/sites-enabled/example
```

To remove a `server`:

```
# unlink /etc/nginx/sites-enabled/example
```

Reload/**restart** `nginx.service` to enable the new configuration.

TLS

This article or section needs language, wiki syntax or style improvements. See Help:Style for reference.

Reason: Do not duplicate OpenSSL#Certificates. (Discuss in Talk:Nginx#)

OpenSSL provides TLS support and is installed by default on Arch installations.

Tip:

- You may want to read the **ngx_http_ssl_module** docs first before configuring SSL.
- **Let's Encrypt** is a free, automated, and open certificate authority. A plugin is available to request valid SSL certificates straight from the command line and automatic configuration.
- Mozilla has a useful **TLS article** as well as an **automated tool** to help create a more secure configuration.
- **Cipherli.st** provides strong TLS implementation examples and tutorial for most modern webservers.

Warning: If you plan on implementing TLS, know that some variations and implementations are **still vulnerable to attack[2]**. For details on these current vulnerabilities within TLS and how to apply appropriate changes to nginx, visit **https://weakdh.org/sysadmin.html**

Create a private key and self-signed certificate. This is adequate for most installations that do not require a **CSR**:

```
# mkdir /etc/nginx/ssl
# cd /etc/nginx/ssl
# openssl req -new -x509 -nodes -newkey rsa:4096 -keyout server.key -out server.crt -days 1095
# chmod 400 server.key
# chmod 444 server.crt
```

Note: The `-days` switch is optional and RSA keysize can be as low as 2048 (default).

If you need to create a CSR, follow these instructions instead of the above:

```
# mkdir /etc/nginx/ssl
# cd /etc/nginx/ssl
# openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -out server.key
# chmod 400 server.key
# openssl req -new -sha256 -key server.key -out server.csr
# openssl x509 -req -days 1095 -in server.csr -signkey server.key -out server.crt
```

Note: For more *openssl* options, read its **man page** or peruse its **extensive documentation**.

Basic example of `/etc/nginx/nginx.conf` using TLS:

```
/etc/nginx/nginx.conf
```

```
http {
    ...
    ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ..

    # Redirect to HTTPS
    server {
        listen 80;
        server_name localhost;
        return 301 https://$host$request_uri;
    }

    server {
        #listen 80; # Uncomment to also listen for HTTP requests
        listen 443 ssl http2; # HTTP/2 is only possible when using SSL
        server_name localhost;

        ssl_certificate ssl/server.crt;
        ssl_certificate_key ssl/server.key;

        root /usr/share/nginx/html;
        location / {
            index index.html index.htm;
        }
    }
}
```

Restart `nginx.service` to apply any changes.

Per-User Directories

To replicate Apache-style `~user` URLs to users' `~/public_html` directories, try the following. (Note: if both rules are used, below, the more-specific PHP rule must come first.)

/etc/nginx/nginx.conf

```
...
server {
    ...
    # PHP in user directories, e.g. http://example.com/~user/test.php
    location ~ ^/~(.+?)(/.+\.php)$ {
        alias         /home/$1/public_html$2;
        fastcgi_pass   unix:/run/php-fpm/php-fpm.sock;
        fastcgi_index  index.php;
        include        fastcgi.conf;
    }

    # User directories, e.g. http://example.com/~user/
    location ~ ^/~(.+?)(/.*)?$ {
        alias     /home/$1/public_html$2;
        index     index.html index.htm;
        autoindex on;
    }
    ...
}
...
```

See **#PHP implementation** for more information on PHP configuration with `nginx`.

Restart `nginx.service` to enable the new configuration.

FastCGI

FastCGI, also FCGI, is a protocol for interfacing interactive programs with a web server. FastCGI is a variation on the earlier **Common Gateway Interface** (CGI); FastCGI's main aim is to reduce the overhead associated with interfacing the web server and CGI programs, allowing servers to handle more web page requests at once.

FastCGI technology is introduced into nginx to work with many external tools, e.g. **Perl**, **PHP** and **Python**.

PHP implementation

PHP-FPM is the recommended solution to run as FastCGI server for **PHP**.

Install `php-fpm` and make sure **PHP** has been installed and configured correctly. The main configuration file of PHP-FPM is `/etc/php/php-fpm.conf`. For basic usage the default configuration should be sufficient.

Finally, **enable** and **start** `php-fpm.service`.

Note:

- If you **run nginx under a different user**, make sure that the PHP-FPM socket file is accessible by this user, or use a TCP socket.
- If you run nginx in chrooted environment (chroot is `/srv/nginx-jail`, web pages are served at `/srv/nginx-jail/www`), you must modify the file `/etc/php/php-fpm.conf` to include the `chroot /srv/nginx-jail` and `listen = /srv/nginx-jail/run/php-fpm/php-fpm.sock` directives within the pool section (a default one is `[www]`). Create the directory for the socket file, if missing. Moreover, for modules that are dynamically linked to dependencies, you will

need to copy those dependencies to the chroot (e.g. for php-imagick, you will need to copy the ImageMagick libraries to the chroot, but not imagick.so itself).

nginx configuration

Adding to main configuration

When serving a PHP web-application, a `location` for PHP-FPM should to be included in each **server block [3]**, e.g.:

/etc/nginx/sites-available/example

```
server {
    ...

    root /usr/share/nginx/html;
    location / {
        index index.html index.htm;
    }

    location ~ [^/]\.php(/|$) {
        # Correctly handle request like /test.php/foo/blah.php or /test.php/
        fastcgi_split_path_info ^(.+?\.php)(/.*)$;

        try_files $uri $document_root$fastcgi_script_name =404;

        # Mitigate https://httpoxy.org/ vulnerabilities
        fastcgi_param HTTP_PROXY "";

        fastcgi_pass unix:/run/php-fpm/php-fpm.sock;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

If it is needed to process other extensions with PHP (e.g. *.html* and *.htm*):

```
location ~ [^/]\.php|html|htm(/|$) {
    ...
}
```

Non *.php* extension processing in PHP-FPM should also be explicitly added in `/etc/php/php-fpm.d/www.conf`:

```
security.limit_extensions = .php .html .htm
```

Note: Pay attention to the `fastcgi_pass` argument, as it must be the TCP or Unix socket defined by the chosen FastCGI server in its config file. The **default** (Unix) socket for `php-fpm` is:

```
fastcgi_pass unix:/run/php-fpm/php-fpm.sock;
```

You might use the common TCP socket, **not default**,

```
fastcgi_pass 127.0.0.1:9000;
```

Unix domain sockets should however be faster.

PHP configuration file

If using multiple `server` blocks with enabled PHP support, it might be easier to create a PHP config file instead:

/etc/nginx/php.conf

```
location ~ \.php$ {
    ...
}
```

To enable PHP support for a particular server, simple include `php.conf`:

/etc/nginx/nginx.conf

```
server {
    server_name example.com;
    ...
    include php.conf;
}
```

Test configuration

You need to **restart** the `php-fpm.service` and `nginx.service` units if the configuration has been changed in order to apply changes.

To test the FastCGI implementation, create a new PHP file inside the `root` folder containing:

```
<?php phpinfo(); ?>
```

Navigate this file inside a browser and you should see the informational page with the current PHP configuration.

CGI implementation

This implementation is needed for CGI applications.

fcgiwrap

Install **fcgiwrap**. The configuration file is `/usr/lib/systemd/system/fcgiwrap.socket`. **Enable** and **start** `fcgiwrap.socket`.

Multiple worker threads

If you want to spawn multiple worker threads, it is recommended that you use **multiwatch**[AUR], which will take care of restarting crashed children. You will need to use `spawn-fcgi` to create the unix socket, as multiwatch seems unable to handle the systemd-created socket, even though fcgiwrap itself does not have any trouble if invoked directly in the unit file.

Copy the unit file from `/usr/lib/systemd/system/fcgiwrap.service` to `/etc/systemd/system/fcgiwrap.service` (and the `fcgiwrap.socket` unit, if present), and modify the `ExecStart` line to suit your needs. Here is a unit file that uses **multiwatch**[AUR]. Make sure `fcgiwrap.socket` is not started or enabled, because it will conflict with this unit:

/etc/systemd/system/fcgiwrap.service

```
[Unit]
Description=Simple CGI Server
After=nss-user-lookup.target

[Service]
ExecStartPre=/bin/rm -f /run/fcgiwrap.socket
ExecStart=/usr/bin/spawn-fcgi -u http -g http -s /run/fcgiwrap.sock -n -- /usr/bin/multiwatch -f 10 -- /usr/sbin/fcgiwrap
ExecStartPost=/usr/bin/chmod 660 /run/fcgiwrap.sock
PrivateTmp=true
Restart=on-failure

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

Tweak `-f 10` to change the number of children that are spawned.

Warning: The `ExecStartPost` line is required because of strange behaviour I'm seeing when I use the `-M 660` option for `spawn-fcgi`. The wrong mode is set. This may be a bug?

nginx configuration

Inside each `server` block serving a CGI web application should appear a `location` block similar to:

```
location ~ \.cgi$ {
    root          /path/to/server/cgi-bin;
    fastcgi_pass   unix:/run/fcgiwrap.sock;
    include        fastcgi.conf;
}
```

The default socket file for `fcgiwrap` is `/run/fcgiwrap.sock`.

If you keep getting a `502 - bad Gateway` error, you should check if your CGI-application first announces the mime-type of the following content. For html this needs to be `Content-type: text/html`.

Installation in a chroot

Installing nginx in a **chroot** adds an additional layer of security. For maximum security the chroot should include only the files needed to run the nginx server and all files should have the most restrictive permissions possible, e.g., as much as possible should be owned by root, directories such as `/usr/bin` should be unreadable and unwriteable, etc.

Arch comes with an `http` user and group by default which will run the server. The chroot will be in `/srv/http`.

A perl script to create this jail is available at **jail.pl gist**. You can either use that or follow the instructions in this article. It expects to be run as root. You will need to uncomment a line before it makes any changes.

Create necessary devices

nginx needs `/dev/null`, `/dev/random`, and `/dev/urandom`. To install these in the chroot create the `/dev/` directory and add the devices with *mknod*. Avoid mounting all of `/dev/` to ensure that, even if the chroot is compromised, an attacker must break out of the chroot to access important devices like `/dev/sda1`.

Tip: Be sure that `/srv/http` is mounted without no-dev option

Tip: See **mknod(1)** and `ls -l /dev/{null,random,urandom}` to better understand the *mknod* options.

```
# export JAIL=/srv/http
# mkdir $JAIL/dev
# mknod -m 0666 $JAIL/dev/null c 1 3
# mknod -m 0666 $JAIL/dev/random c 1 8
# mknod -m 0444 $JAIL/dev/urandom c 1 9
```

Create necessary directories

nginx requires a bunch of files to run properly. Before copying them over, create the folders to store them. This assumes your nginx document root will be `/srv/http/www`.

```
# mkdir -p $JAIL/etc/nginx/logs
# mkdir -p $JAIL/usr/{lib,bin}
# mkdir -p $JAIL/usr/share/nginx
# mkdir -p $JAIL/var/{log,lib}/nginx
# mkdir -p $JAIL/www/cgi-bin
# mkdir -p $JAIL/{run,tmp}
# cd $JAIL; ln -s usr/lib lib
# cd $JAIL; ln -s usr/lib lib64
# cd $JAIL/usr; ln -s lib lib64
```

Then mount `$JAIL/tmp` and `$JAIL/run` as tmpfs's. The size should be limited to ensure an attacker cannot eat all the RAM.

```
# mount -t tmpfs none $JAIL/run -o 'noexec,size=1M'
# mount -t tmpfs none $JAIL/tmp -o 'noexec,size=100M'
```

In order to preserve the mounts across reboots, the following entries should be added to `/etc/fstab`:

```
/etc/fstab

tmpfs   /srv/http/run   tmpfs   rw,noexec,relatime,size=1024k   0       0
tmpfs   /srv/http/tmp   tmpfs   rw,noexec,relatime,size=102400k 0       0
```

Populate the chroot

First copy over the easy files.

```
# cp -r /usr/share/nginx/* $JAIL/usr/share/nginx
# cp -r /usr/share/nginx/html/* $JAIL/www
# cp /usr/bin/nginx $JAIL/usr/bin/
# cp -r /var/lib/nginx $JAIL/var/lib/nginx
```

Now copy over required libraries. Use *ldd* to list them and then copy them all to the correct location. Copying is preferred over hardlinks to ensure that even if an attacker gains write access to the files they cannot destroy or alter the true system files.

```
$ ldd /usr/bin/nginx

linux-vdso.so.1 (0x00007fffc41fe000)
libpthread.so.0 => /usr/lib/libpthread.so.0 (0x00007f57ec3e8000)
libcrypt.so.1 => /usr/lib/libcrypt.so.1 (0x00007f57ec1b1000)
libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x00007f57ebead000)
libm.so.6 => /usr/lib/libm.so.6 (0x00007f57ebbaf000)
libpcre.so.1 => /usr/lib/libpcre.so.1 (0x00007f57eb94c000)
libssl.so.1.0.0 => /usr/lib/libssl.so.1.0.0 (0x00007f57eb6e0000)
libcrypto.so.1.0.0 => /usr/lib/libcrypto.so.1.0.0 (0x00007f57eb2d6000)
libdl.so.2 => /usr/lib/libdl.so.2 (0x00007f57eb0d2000)
libz.so.1 => /usr/lib/libz.so.1 (0x00007f57eaebc000)
libGeoIP.so.1 => /usr/lib/libGeoIP.so.1 (0x00007f57eac8d000)
libgcc_s.so.1 => /usr/lib/libgcc_s.so.1 (0x00007f57eaa77000)
libc.so.6 => /usr/lib/libc.so.6 (0x00007f57ea6ca000)
/lib64/ld-linux-x86-64.so.2 (0x00007f57ec604000)
```

For files residing in `/usr/lib` you may try the following one-liner:

```
# cp $(ldd /usr/bin/nginx | grep /usr/lib | sed -sre 's/(.+)(\/usr\/lib\/\S+).+/\2/g') $JAIL/usr/lib
```

And the following for `ld-linux-x86-64.so`:

```
# cp /lib64/ld-linux-x86-64.so.2 $JAIL/lib
```

Note: Do not try to copy `linux-vdso.so`: it is not a real library and does not exist in `/usr/lib`.

Copy over some miscellaneous but necessary libraries and system files.

```
# cp /usr/lib/libnss_* $JAIL/usr/lib
# cp -rfvL /etc/{services,localtime,nsswitch.conf,nscd.conf,protocols,hosts,ld.so.cache,ld.so.conf,resolv.conf,host.conf,nginx} $JAIL/etc
```

Create restricted user/group files for the chroot. This way only the users needed for the chroot to function exist as far as the chroot knows, and none of the system users/groups are leaked to attackers should they gain access to the chroot.

```
$JAIL/etc/group

http:x:33:
nobody:x:99:
```

```
$JAIL/etc/passwd

http:x:33:33:http://:/bin/false
nobody:x:99:99:nobody:/:/bin/false
```

```
$JAIL/etc/shadow

http:x:14871::::::
nobody:x:14871::::::
```

```
$JAIL/etc/gshadow

http:::
nobody:::
```

```
# touch $JAIL/etc/shells
# touch $JAIL/run/nginx.pid
```

Finally make set very restrictive permissions. As much as possible should be owned by root and set unwritable.

```
# chown -R root:root $JAIL/

# chown -R http:http $JAIL/www
# chown -R http:http $JAIL/etc/nginx
# chown -R http:http $JAIL/var/{log,lib}/nginx
# chown http:http $JAIL/run/nginx.pid

# find $JAIL/ -gid 0 -uid 0 -type d -print | xargs chmod -rw
# find $JAIL/ -gid 0 -uid 0 -type d -print | xargs chmod +x
# find $JAIL/etc -gid 0 -uid 0 -type f -print | xargs chmod -x
# find $JAIL/usr/bin -type f -print | xargs chmod ug+rx
# find $JAIL/ -group http -user http -print | xargs chmod o-rwx
# chmod +rw $JAIL/tmp
# chmod +rw $JAIL/run
```

If your server will bind port 80 (or any other port in range [1-1023]), give the chrooted executable permission to bind these ports without root.

```
# setcap 'cap_net_bind_service=+ep' $JAIL/usr/bin/nginx
```

Modify nginx.service to start chroot

Before modifying the `nginx.service` unit file, it may be a good idea to copy it to `/etc/systemd/system/` since the unit files there take priority over those in `/usr/lib/systemd/system/`. This means upgrading nginx would not modify your custom *.service* file.

```
# cp /usr/lib/systemd/system/nginx.service /etc/systemd/system/nginx.service
```

The systemd unit must be changed to start up nginx in the chroot, as the http user, and store the pid file in the chroot.

Note: I'm not sure if the pid file needs to be stored in the chroot jail.

```
/etc/systemd/system/nginx.service
```

```
[Unit]
Description=A high performance web server and a reverse proxy server
After=syslog.target network.target

[Service]
Type=forking
PIDFile=/srv/http/run/nginx.pid
ExecStartPre=/usr/bin/chroot --userspec=http:http /srv/http /usr/bin/nginx -t -q -g 'pid /run/nginx.pid; daemon on; master_process on;'
ExecStart=/usr/bin/chroot --userspec=http:http /srv/http /usr/bin/nginx -g 'pid /run/nginx.pid; daemon on; master_process on;'
ExecReload=/usr/bin/chroot --userspec=http:http /srv/http /usr/bin/nginx -g 'pid /run/nginx.pid; daemon on; master_process on;' -s reload
ExecStop=/usr/bin/chroot --userspec=http:http /srv/http /usr/bin/nginx -g 'pid /run/nginx.pid;' -s quit

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

Note: Upgrading nginx with pacman will not upgrade the chrooted nginx installation. You have to take care of the updates manually by repeating some of the steps above. Do not forget to also update the libraries it links against.

You can now safely get rid of the non-chrooted nginx installation.

```
# pacman -Rsc nginx
```

If you do not remove the non-chrooted nginx installation, you may want to make sure that the running nginx process is in fact the chrooted one. You can do so by checking where `/proc/PID/root` symmlinks to. If should link to `/srv/http` instead of `/`.

```
# ps -C nginx | awk '{print $1}' | sed 1d | while read -r PID; do ls -l /proc/$PID/root; done
```

Tips and tricks

Running unprivileged using systemd

Edit nginx.service and set the `User` and optionally `Group` options under `[Service]`:

/etc/systemd/system/nginx.service.d/user.conf

```
[Service]
User=user
Group=group
```

We can harden the service against ever elevating privileges:

/etc/systemd/system/nginx.service.d/user.conf

```
[Service]
...
NoNewPrivileges=yes
```

Tip: See **systemd.exec(5)** for more options of confinement.

Then we need to ensure that `user` has access to everything it needs:

Port

Linux does not permit non-`root` processes to bind to ports below 1024 by default. A port above 1024 can be used:

/etc/nginx/nginx.conf

```
server {
	listen 8080;
}
```

Tip: If you want nginx accessible on port 80 or 443, configure your **firewall** to redirect requests from 80 or 443 to the ports nginx listens to.

Or you may grant the nginx process the CAP_NET_BIND_SERVICE capability which will allow it to bind to ports below 1024:

/etc/systemd/system/nginx.service.d/user.conf

```
[Service]
...
CapabilityBoundingSet=
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
AmbientCapabilities=
AmbientCapabilities=CAP_NET_BIND_SERVICE
```

PID file

nginx uses `/run/nginx.pid` by default. We can create a directory that *user* has write access to and place our PID file in there. An example using

systemd-tmpfiles:

```
/etc/tmpfiles.d/nginx.conf

d /run/nginx 0775 root group - -
```

Run the configuration:

```
# systemd-tmpfiles --create
```

Edit the PID values based on the original `nginx.service`:

```
/etc/systemd/system/nginx.service.d/user.conf

[Service]
...
PIDFile=/run/nginx/nginx.pid
ExecStart=
ExecStart=/usr/bin/nginx -g 'pid /run/nginx/nginx.pid; error_log stderr;'
ExecReload=
ExecReload=/usr/bin/nginx -s reload -g 'pid /run/nginx/nginx.pid;'
```

`/var/lib/nginx/*`

Some directories under `/var/lib/nginx` need to be bootstrapped by nginx running as `root`. It is not necessary to start the whole server to do that, nginx will do it on a simple **configuration test**. So just run one of those and you're good to go.

Log file & Directory Permissions

The step of running a configuration test will create a dangling `root`-owned log. Remove logs in `/var/log/nginx` to start fresh.

The nginx service user needs write permission to `/var/log/nginx`. This may require **changing permission** and/or ownership of this directory on your system.

Now we should be good to go. Go ahead and **start** nginx, and enjoy your completely rootless nginx.

Tip: The same setup may be desirable for your **FastCGI server** as well.

Alternative script for systemd

On pure systemd you can get advantages of chroot + systemd. **[4]** Based on set **user group** an pid on:

```
/etc/nginx/nginx.conf

user http;
pid /run/nginx.pid;
```

the absolute path of file is `/srv/http/etc/nginx/nginx.conf`.

```
/etc/systemd/system/nginx.service

[Unit]
Description=nginx (Chroot)
After=syslog.target network.target

[Service]
Type=forking
PIDFile=/srv/http/run/nginx.pid
RootDirectory=/srv/http
ExecStartPre=/usr/sbin/nginx -t -c /etc/nginx/nginx.conf
ExecStart=/usr/sbin/nginx -c /etc/nginx/nginx.conf
ExecReload=/usr/sbin/nginx -c /etc/nginx/nginx.conf -s reload
ExecStop=/usr/sbin/nginx -c /etc/nginx/nginx.conf -s stop

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

It is not necesary to set the default location, nginx loads at default `-c /etc/nginx/nginx.conf`, but it is a good idea though.

Alternatively you can run **only** `ExecStart` as chroot with parameter `RootDirectoryStartOnly` set as `yes` **man systemd service** or start it before mount point as effective or a **systemd path** is available.

```
/etc/systemd/system/nginx.path

[Unit]
Description=nginx (Chroot) path
[Path]
PathExists=/srv/http/site/Public_html
[Install]
WantedBy=default.target
```

Enable the created `nginx.path` and change the `WantedBy=default.target` to `WantedBy=nginx.path` in `/etc/systemd/system/nginx.service`.

The `PIDFile` in unit file allows systemd to monitor process (absolute path required). If it is undesired, you can change to default one-shot type, and delete the reference from the unit file.

Nginx Beautifier

nginxbeautifier[AUR] is a commandline tool used to beautify and format nginx configuration files.

Troubleshooting

Configuration validation

```
# nginx -t
```

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Accessing local IP redirects to localhost

Solution from the Arch Linux **forum**.

In `/etc/nginx/nginx.conf` locate the `server_name localhost` line without a `#` in front of it, and add below:

```
server_name_in_redirect off;
```

Default behavior is that nginx redirects any requests to the value given as `server_name` in the config.

Error: The page you are looking for is temporarily unavailable. Please try again later. (502 Bad Gateway)

This is because the FastCGI server has not been started, or the socket used has wrong permissions.

Try **out this answer** to fix the 502 error.

In Archlinux, the configuration file mentioned in above link is `/etc/php/php-fpm.conf`.

Error: No input file specified

1. Verify that variable `open_basedir` in `/etc/php/php.ini` contains the correct path specified as `root` argument in `nginx.conf` (usually `/usr/share/nginx/`). When using **PHP-FPM** as FastCGI server for PHP, you may add `fastcgi_param PHP_ADMIN_VALUE "open_basedir=$document_root/:/tmp/:/proc/";` in the `location` block which aims for processing php file in `nginx.conf`.

2. Another occasion is that, wrong `root` argument in the `location ~ \.php$` section in `nginx.conf`. Make sure the `root` points to the same directory as it in `location /` in the same server. Or you may just set root as global, do not define it in any location section.

3. Check permissions: e.g. `http` for user/group, `755` for directories and `644` for files. Remember the entire path to the `html` directory should have the correct permissions. See **File permissions and attributes#Bulk chmod** to bulk modify a directory tree.

4. You do not have the `SCRIPT_FILENAME` containing the full path to your scripts. If the configuration of nginx (`fastcgi_param SCRIPT_FILENAME`) is correct, this kind of error means php failed to load the requested script. Usually it is simply a permissions issue, you can just run php-cgi as root:

```
# spawn-fcgi -a 127.0.0.1 -p 9000 -f /usr/bin/php-cgi
```

or you should create a group and user to start the php-cgi:

```
# groupadd www
# useradd -g www www
# chmod +w /srv/www/nginx/html
# chown -R www:www /srv/www/nginx/html
# spawn-fcgi -a 127.0.0.1 -p 9000 -u www -g www -f /usr/bin/php-cgi
```

5. If you are running php-fpm with chrooted nginx ensure `chroot` is set correctly within `/etc/php-fpm/php-fpm.d/www.conf` (or `/etc/php-fpm/php-fpm.conf` if working on older version)

Warning: Could not build optimal types_hash

When starting the `nginx.service`, the process might log the message:

```
[warn] 18872#18872: could not build optimal types_hash, you should increase either types_hash_max_size: 1024 or types_hash_bucket_size: 64; ignoring types_hash_bucket_size
```

To fix this warning, increase the values for these keys inside the `http` block **[5]** **[6]**:

/etc/nginx/nginx.conf

```
http {
    types_hash_max_size 4096;
    server_names_hash_bucket_size 128;
    ...
}
```

Cannot assign requested address

The full error from `systemctl status nginx.service` is

```
[emerg] 460#460: bind() to A.B.C.D:443 failed (99: Cannot assign requested address)
```

Even, if your nginx unit-file is configured to run after `network.target` with systemd, nginx may attempt to listen at an address that is configured but not added to any interface yet. Verify that this the case by manually running **start** for nginx (thereby showing the IP address is configured properly). Configuring nginx to listen to any address will resolve this issue. Now if your use case requires listening to a specific address, one possible solution is to reconfigure systemd.

To start nginx after all configured network devices are up and assigned an IP address, append `network-online.target` to `After=` within `nginx.service` and **start/enable** `systemd-networkd-wait-online.service`.

See also

- **nginx configuration pitfalls**
- **Very good in-depth 2014 look at nginx security and Reverse Proxying**
- **Installing LEMP (nginx, PHP, MySQL with MariaDB engine and PhpMyAdmin) in Arch Linux**
- **Using SSL certificates generated with Let's Encrypt**

Category: Web server

Deutsch

Русский