NGINX Web Server Setup on Linux

We are going to show you how to install and setup an NGINX web server. We cover setting up virtual server blocks so that you can server multiple different sites from the same server. We also cover setting up SSL Encryption with Lets Encrypt. We will be adding PHP support as well. We're assuming you're running on Ubuntu Linux.
We're covering:

- NGINX Web Server Setup
- SSL
- PHP
- Security / Hardening

Install NGINX

First, update your information about available packages and versions. Then, install the nginx package.

```
sudo apt-get update
sudo apt-get install nginx
```

You can start, stop, and restart the NGINX webserver like this:

```
sudo service nginx start
sudo service nginx stop
sudo service nginx restart
```

Make sure that the service is enabled so that it will start up when your system boots.

```
sudo systemctl enable nginx
```

Any time you make a config change, you should test the change with 'nginx -t' before actually restarting. This will let you know if there are any errors before actually restarting and breaking things.

```
sudo nginx -t
sudo systemctl restart nginx
```

You can verify that the server is working using curl.

```
curl -I http://localhost
```

One of the first things that you will notice is that it tells you what version of NGINX is running. This information could be used by hackers looking for a vulnerable system. You want to reveal as little information as possilbe about your server. To fix this, turn server_tokens off in your nginx.conf file and retest it, like this:

```
sudo vi /etc/nginx/nginx.conf
server_tokens off;
sudo service nginx restart
curl -I http://localhost
```

That's it! You have a basic NGINX webserver up and running. You are probably not going to want to stop here though. You are probably going to want to setup server blocks, PHP, SSL, and more.

Nginx Server Blocks / Virtual hosts

Using server blocks, you can run mutliple sites on a single instance of NGINX. These work similar to virtual hosts in Apache. For this example, our site will be example.com. For more details on how to set this up, check out our **full guide here: NGINX Server Blocks / Virtual Hosts**.

Start out by creating a directory for our domain, changing ownership, and setting permissions.

Users:

- root - The master proces runs as this.
- www-data - The worker processes run as this.
- user1 - This users gets write access to web content. Change this if you want.

```
sudo mkdir -p /var/www/example.com/html
sudo chown -R user1:user1 /var/www/
sudo chmod -R 755 /var/www
```

Next, copy the default server block configuration to creaste a new configuration for our new site.

```
sudo cp /etc/nginx/sites-available/default /etc/nginx/sites-
available/example.com
```

Edit this configuration. It should look something like this.

```
sudo vi /etc/nginx/sites-available/example.com

server {
listen 80;
listen [::]:80;
root /var/www/example.com/html;
index index.html index.htm index.nginx-debian.html;
server_name example.com;
location / {
try_files $uri $uri/ =404;
}
}
```

To enable this server block, create a link from the sites-enabled directory to the sites-available directory.

```
sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-
enabled/
```

Set the server_hash_bucket_size to 64 if it isn't already.

```
sudo vi /etc/nginx/nginx.conf
http {
server_names_hash_bucket_size 64;
```

```
}
```

You should be able to edit the HTML for your site like this. Note that you shouldn't need sudo.

```
vi /var/www/example.com/html/index.html
```

Test the config changes and restart the service.

```
sudo nginx -t
sudo systemctl restart nginx
```

SSL With Let's Encrypt

You can add the repo like this. You don't need to do this anymore since the packages have been included in the main Ubuntu repos in newer versions of Ubuntu. We've tested this on Ubuntu 18.04.3 and this was the case. If you are running an older version of Ubuntu, you still might need to add the repo.
SKIPadding the repo if you are running a newer version of Ubuntu.

```
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
```

Install the Certbot package for NGINX and restart the server.

```
sudo apt-get install python-certbot-nginx
sudo nginx -t
sudo systemctl reload nginx
```

You can add a cert for a domain like this.

```
sudo certbot --nginx -d example.com
```

You will eventual need to renew your certs. You can test out a renewal dry run like this.

```
sudo certbot renew --dry-run
```

Make sure that you enable the rule to open both ports 443 and 80 on your firewall. Make sure you also disable the rule that only opens port 80.

```
sudo ufw allow 'Nginx Full'
sudo ufw delete allow 'Nginx HTTP'
```

Alternate Certbot Install Method

At one point, I ran into some trouble with the version in the repo because it wasn't new enough. This was a while back on a slightly older version of Ubuntu and shouldn't be an issue anymore but it is good to know about in case anything similar comes up.

```
wget https://dl.eff.org/certbot-auto
chmod a+x ./certbot-auto
sudo ./certbot-auto # will give you a menu
sudo ./certbot-auto --nginx -d example2.com # one domain at a time
```

Another interesting thing that came up was an issue with an unneeded IPv6 line in the config which was causing errors. This line is automatically added by Certbot and ultimately the solution was to just comment it out.

```
sudo vi /etc/nginx/sites-available/example2.com
# comment out the IPv6 line so the next domain will work without
conflict ...
#listen [::]:443 ssl ipv6only=on; # managed by Certbot
```

Restart NGINX and test out certificate renewal.

```
sudo nginx -t
sudo systemctl reload nginx # don't need, do it anyway
sudo ./certbot-auto renew # check that it works
```

This can be configured to run automatically as a cronjob like this.

```
crontab -e
```

```
0 5 * * * /home/user1/certbot-auto renew &>
/home/user1/log/certbot_output_`date +%Y-%m-%d_%H:%M:%S`.log
```

PHP Setup

PHP-FPM (FastCGI Process Manager) is needed for PHP support. You will first need to install php-fpm.

```
sudo apt-get install php-fpm
```

Next, make a couple quick edits to your php.ini file. These settings will improve security by limiting information leakage. Restart PHP FPM when you are done.

```
sudo vi /etc/php/7.0/fpm/php.ini
cgi.fix_pathinfo=0
expose_php = Off # already default in my case
sudo systemctl restart php7.0-fpm
```

Make a few changes to the server block as shown below:

```
index index.php index.html index.htm index.nginx-debian.html;
...
location ~ \.php$ {
include snippets/fastcgi-php.conf;
fastcgi_pass unix:/run/php/php7.0-fpm.sock;
}

location ~ /\.ht {
deny all;
}
```

Test the configuration changes and restart NGINX.

```
sudo nginx -t
sudo systemctl reload nginx
```

Extensionless PHP on NGINX

If you want PHP without the '.php' extension you can set this up with the configuration shown here. This way, if you try to access https://example.org/mypage it will automatically load https://example.org/mypage.php.

```
location / {
#try_files $uri $uri/ =404;
try_files $uri $uri/ @extensionless-php;
}
...
location @extensionless-php {
rewrite ^(.*)$ $1.php last;
}
```

For Extra Security you can redirect all error pages to the 404 error page. This reveals less information about why a page is unreachable. Just add this line in your server block.

```
error_page 401 403 404 /404.html;
```

If you want to go even further, you can audit your site with WAPITI.

```
sudo apt-get install wapiti
wapiti http://example.org -n 10 -b folder
```