How To Configure Single and Multiple WordPress Site Settings with Nginx

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=291*

WordPress is the most popular CMS (content management system) used on the internet today. WordPress sites can be served using a HTTP server such as Apache or NGINX, while Apache is a great option to serve websites, many sites have moved to NGINX because of it's scalable event-driven architecture, low resources and better delivery of statics files. In this tutorial you will learn how to configure NGINX for various types of WordPress installations, including multisite configurations, rewrite rules and the use of .conf files to apply repeated configurations.

Requirements

In this guide, you will need**sudo**to install and edit files. I assume that you have gone through the initial server setup.

You will need to install MySQL, PHP & NGINX. You can follow these guides to install LEMP on Ubuntu or Debian.

Note that our server blocks will be different & that in this tutorial we are making PHP-FPM use a UNIX Socket.

Basic NGINX Optimization

Adjust NGINX Worker Processes & Connections

It is often recommended to set the number of NGINX workers equal the number of processors, you can determine the number of processors using:

```
cat /proc/cpuinfo | grep processor
```

Open up the main NGINX configuration file:

```
sudo nano /etc/nginx/nginx.conf
```

Increase or decrease the number of workers depending on your system's specs:

```
worker_processes 1;
```

NGINX limits the number of connections that a worker can maintain at one time, if your websites have many visitors you might want to increase the limit of connections. In theory the maximum number of connections = workers * limit.

```
worker_connections 768;
```

Enabling Gzip

Files can be compressed using Gzip to accelerate WordPress, the smaller the data size requested by the user, the faster the response. Think about CSS files & HTML files, they have many similar strings, repeated text and white spaces. Gzip uses an algorithm called DEFLATE that removes duplicate strings by linking to the previous location of that identical string and creates a much smaller file. Find the Gzip section and enable it:

```
gzip on;
gzip_types text/css text/x-component application/x-javascript
application/javascript text/javascript text/x-js text/richtext image/svg+xml
text/plain text/xsd text/xsl text/xml image/x-icon;
```

Save & exit.

Creating NGINX .conf files

Since you might be hosting more than one WordPress website, we are going to create a few .conf files that can be loaded from the server blocks instead of writing the same configuration many times on each server block.

In the next steps we will create 3 files that will hold our configurations:

- common.conf: Configurations applicable to all sites.
- wordpress.conf: Configurations applicable to all WordPress sites.
- multisite.conf: Special configurations for WordPress multisite with sub-directories.

We are going to create all the files in a directory called "global" but first we will need to create the mentioned directory:

```
sudo mkdir /etc/nginx/global
```

I am going to set **/etc/nginx/global** as the current directory just to make things easier.

```
cd /etc/nginx/global
```

common.conf file

Let's create our first .conf file applicable to any kind of websites.

```
sudo nano common.conf
```

This will open an empty file, copy the following configurations:

```
# Global configuration file.
# ESSENTIAL : Configure Nginx Listening Port
listen 80;
# ESSENTIAL : Default file to serve. If the first file isn't found,
index index.php index.html index.htm;
# ESSENTIAL : no favicon logs
location = /favicon.ico {
    log_not_found off;
    access_log off;
}
# ESSENTIAL : robots.txt
location = /robots.txt {
    allow all;
    log_not_found off;
    access_log off;
}
# ESSENTIAL : Configure 404 Pages
error_page 404 /404.html;
# ESSENTIAL : Configure 50x Pages
error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/www;
    }
# SECURITY : Deny all attempts to access hidden files .abcde
location ~ /\. {
    deny all;
}
# PERFORMANCE : Set expires headers for static files and turn off logging.
location ~*
```

```
^.+\.(js|css|swf|xml|txt|ogg|ogv|svg|svgz|eot|otf|woff|mp4|ttf|rss|atom|jpg|jpeg|gif|p
{
    access_log off; log_not_found off; expires 30d;
```

```
}
```

`listen 80;` specifies the listening port of the server.

`index index.php...` specifies the default file to serve (WordPress index.php). If the first file isn't found, the second will be used and so on. You might have HTML sites that's why we are including index.html & index.htm;.

`location = /robots.txt {allow all;}` allows the access to robots.txt, if you want to specify another directory for the robots.txt you can add an alias:

```
location /robots.txt {
    alias /var/www/example.com/public/sample_robots.txt;
}
```

`location ~ /\. {deny all;}` in the Linux operating system a hidden file begins with a ".", access to some hidden files, such as .htaccess, should be blocked for security reasons.

`location ~* ^.+\.(js|css|swf...` expires headers tell the browser whether they should request a specific file from the server or whether they should grab it from the browser's cache. With expires 30d we are telling the browser to store static files such as pictures for 30 days.

Save and exit.

wordpress.conf file

Let's create a .conf file applicable to all WordPress sites:

```
sudo nano wordpress.conf
```

This will open an empty file, copy the following configurations:

```
# WORDPRESS : Rewrite rules, sends everything through index.php and keeps the
appended query string intact
location / {
    try_files $uri $uri/ /index.php?q=$uri&$args;
}
# SECURITY : Deny all attempts to access PHP Files in the uploads directory
location ~* /(?:uploads|files)/.*\.php$ {
    deny all;
}
# REQUIREMENTS : Enable PHP Support
location ~ \.php$ {
    # SECURITY : Zero day Exploit Protection
    try_files $uri =404;
    # ENABLE : Enable PHP, listen fpm sock
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass unix:/var/run/php5-fpm.sock;
    fastcgi_index index.php;
    include fastcgi_params;
}
```

```
# PLUGINS : Enable Rewrite Rules for Yoast SEO SiteMap
rewrite ^/sitemap_index\.xml$ /index.php?sitemap=1 last;
rewrite ^/([^/]+?)-sitemap([0-9]+)?\.xml$ /index.php?sitemap=&sitemap_n=#
WORDPRESS : Rewrite rules, sends everything through index.php and keeps the
appended query string intact location / { try_files $uri $uri/
/index.php?q=$uri&$args; } # SECURITY : Deny all attempts to access PHP Files in
the uploads directory location ~* /(?:uploads|files)/.*\.php$ { deny all; } #
REQUIREMENTS : Enable PHP Support location ~ \.php$ { # SECURITY : Zero day
Exploit Protection try_files $uri =404; # ENABLE : Enable PHP, listen fpm sock
fastcgi_split_path_info ^(.+\.php)(/.+)$; fastcgi_pass
unix:/var/run/php5-fpm.sock; fastcgi_index index.php; include fastcgi_params; }
# PLUGINS : Enable Rewrite Rules for Yoast SEO SiteMap rewrite
^/sitemap_index\.xml$ /index.php?sitemap=1 last; rewrite
^/([^/]+?)-sitemap([0-9]+)?\.xml$ /index.php?sitemap=$1&sitemap_n=$2 last;
#Yeah! you did it. last;
```
#Yeah! you did it.

`try_files $uri $uri/ /index.php?q=$uri&$args`rewrite rule required to allow you to choose your custom permalink structure on WordPress.

`location ~* /(?:uploads|files)/.*\.php$ {deny all;}`this will prevent malicious code from being uploaded and executed from the WordPress media directory.

`location ~ \.php$ {...}`since WordPress is a php site, we need to tell NGINX how to a pass our php scripts to PHP5.

`try_files $uri =404;`this is a security rule, you only want to either serve a determined php file or go to a 404 error.

More Rules: You might want to add more NGINX rules, for example, if you use the same WP Plugins that require custom rules on all your installations as I do, you can add more rules in this .conf file, e.g. I use Yoast SEO on all my websites therefore I am adding the rewrite rules required here, in this way I do not have to copy the same rewrite rules for each server block.

multisite.conf file

Unlike single site WordPress, which can work with "ugly" permalinks and thus does not need any URL rewrite, a MultiSite installation requires custom rewrite rules to format URLs for your subsites. Let's create a .conf file applicable to multisite WordPress installations:

```
sudo nano multisite.conf
```

This will open an empty file, copy the required rewrite rules:

```
# Rewrite rules for WordPress Multi-site.
if (!-e $request_filename) {
rewrite /wp-admin$ $scheme://$host$uri/ permanent;
rewrite ^/[_0-9a-zA-Z-]+(/wp-.*)  last;
rewrite ^/[_0-9a-zA-Z-]+(/.*\.php)$  last;
}
```

Save & exit.

Small Note

Our current working directory is **/etc/nginx/global**, if you want to change it you can type:

```
cd /desired_directory
```

Creating Server Blocks

It is time to create our first server block. Since we already have everything configured in our .conf files there is no need to duplicate the default server block file. Let's disable the default server block:

```
sudo rm /etc/nginx/sites-enabled/default
```

And create a server block file:

```
sudo nano /etc/nginx/sites-available/demo
```

This will open an empty file, copy the following configurations depending on what you want to achieve:

Simple WordPress Installation

Imagine that you want to configure a WordPress site with this domain www.demo.com. First we will have to create a server block`server {...}`where we will put our rules. We have to specify which server block is used for a given URL, include **common.conf** & **wordpress.conf** and finally we will tell NGINX the location of the WordPress installation in our server.

```
server {
    # URL: Correct way to redirect URL's
    server_name demo.com;
    rewrite ^/(.*)$ http://www.demo.com/ permanent;
}
server {
    server_name www.demo.com;
    root /home/demouser/sitedir;
    access_log /var/log/nginx/www.demo.com.access.log;
    error_log /var/log/nginx/www.demo.com.error.log;
    include global/common.conf;
    include global/wordpress.conf;
}
```

Remember to change the following data to fit your needs:

- **server_name**: Determine which server block is used for a given URL.
- **root**: The path where your site is stored.
- **access log & error log**: Set the paths for your logs

You can see that there are two server blocks, that's because www.demo.com & demo.com are different URLs. You probably want to make sure that Google, Bing, users...etc pick the URL that you want, in this case I want my website to be www.demo.com so I have configured a **permanent redirect** from demo.com to www.demo.com. It is also possible to specify multiple domains:

```
server {
    # URL: Correct way to redirect URL's
    server_name demo.com sub.demo.com example.com;
```

Multisite with Subdirectories

If you want a multisite installation with subdirectories you will need to include the rewrite rules stored in multisite.conf:

```
# URL: add a permanent redirect if required.
server {
    server_name www.demo1.com;
    root /home/demouser/sitedir1;
    access_log /var/log/nginx/www.demo1.com.access.log;
    error_log /var/log/nginx/www.demo1.com.error.log;
    include global/common.conf;
    include global/wordpress.conf;
    include global/multisite.conf;
}
```

Multisite with Subdomains

If you want a multisite installation with subdomains you will need to configure this server block to listen to a domain with a wildcard:

```
server {
    server_name *.demo2.com;
    root /home/demouser/sitedir2;
    access_log /var/log/nginx/demo2.com.access.log;
    error_log /var/log/nginx/demo2.com.error.log;
    include global/common.conf;
    include global/wordpress.conf;
}
```

HTML & Other websites

If you want to host simple html websites or other webapps you might need to specify custom rules or create more .conf files and include them in the server block:

```
# URL: add a permanent redirect if required.
server {
    server_name www.demo3.com;
    root /home/demouser/sitedir3;
```

```
    access_log /var/log/nginx/demo3.com.access.log;
    error_log /var/log/nginx/demo3.com.error.log;
    # custom rules
}
```

Remember to save & exit.

Enabling Server Block Files

The last step is to activate the host by creating a symbolic link between the sites-available directory and the sites-enabled directory:

```
sudo ln -s /etc/nginx/sites-available/demo /etc/nginx/sites-enabled/demo
```

WeÃ¢€â„¢ve made a lot of the changes to the configuration. Reload NGINX and make the changes visible.

```
sudo service nginx reload;
```

Final Notes

To create additional virtual hosts, you can just repeat the process above, being careful to set up a new document root with the appropriate new domain name each time. It is also possible to combine multiple server blocks in just one file:

```
server {
    server_name demo.com;
    rewrite ^/(.*)$ http://www.demo.com/ permanent;
}
server {
    server_name www.demo.com;
    root /home/demouser/sitedir;
    access_log /var/log/nginx/www.demo.com.access.log;
    error_log /var/log/nginx/www.demo.com.error.log;
    include global/common.conf;
    include global/wordpress.conf;
}
server {
    server_name www.demo1.com;
    root /home/demouser/sitedir1;
    access_log /var/log/nginx/www.demo1.com.access.log;
    error_log /var/log/nginx/www.demo1.com.error.log;
    include global/common.conf;
    include global/wordpress.conf;
    include global/multisite.conf;
}
# More server blocks....
```