Basic Nginx Configuration

Updated Tuesday, May 20th, 2014 by Alex Fornuto

Nginx is a lightweight, high performance web server designed to deliver large amounts of static content quickly with efficient use of system resources. Nginx's strong point is its ability to efficiently serve static content, like plain HTML and media files. Some consider it a less than ideal server for dynamic content.

Unlike Apache, which uses a threaded or process-oriented approach to handle requests, Nginx uses an asynchronous event-driven model which provides more predictable performance under load. Rather than using the embedded interpreter approach, Nginx hands off dynamic content to CGI, FastCGI, or even other web severs like Apache, which is then passed back to Nginx for delivery to the client.

This leads to a more complex setup for certain deployments. For these and other reasons, the configuration of Nginx can feel complicated and unintuitive at first. This document should familiarize you with basic Nginx parameters and conventions. We'll be going through Nginx's primary configuration file.

All Nginx configuration files are located in the `/etc/nginx/` directory. The primary configuration file is `/etc/nginx/nginx.conf`.

> Note
>
> This is where the files will be located if you install Nginx from the package manager. Other possible locations include `/opt/nginx/conf/`.

Prerequisites

Before we begin, make sure you have completed the following:

- Follow the Getting Started guide.
- Install the Nginx server.
- The steps required in this guide require root privileges. Be sure to run the steps below as `root` or with the **sudo** prefix. For more information on privileges see our Users and Groups guide.

If you're new to Linux server administration, you may also be interested in our Beginner's Guide and Administration Basics Guide.

Before You Start

The following sections cover a few concepts and safeguards that should be reviewed before making any changes to your Nginx configuration.

Preserve a Working Configuration

Sometimes, server configuration files can get so corrupted or convoluted that they become unusable, so it's always a good idea to have a working copy of the essential files on hand. A good first step is to copy any configuration file before you begin making changes to it, like so:

```
cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.backup
```

For even better restoration options, we recommend making regular backups of your Nginx configuration. You might want to store your entire `/etc/nginx/` directory in a Git repository so you can save the original settings and all the versions from all your different changes. Another option is to periodically create dated copies of your files. You can accomplish this by issuing the following command:

```
cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.$(date "+%b_%d_%Y_%H.%M.%S")
```

Start, Stop, Reload

Now you're ready to make changes to your Nginx configuration. Whenever you make a change to the `nginx.conf` file, you need to reload your configuration before the change will take effect. You can do this by issuing the following command:

```
service nginx reload
```

To completely stop or start the service, replace `reload` with `start` or `stop`.

Introduction to Syntax

This section showcases an excerpt from the beginning of the default configuration file, `/etc/nginx/nginx.conf`:

File excerpt: **/etc/nginx/nginx.conf**

```
user www-data;
worker_processes 4;
pid /run/nginx.pid;

events {
        worker_connections 768;
        # multi_accept on;
}
```

Normally, you will not need to change anything in this first section. Instead, we'll use this as an opportunity to explain the syntax of the configuration file.

- All lines preceded by a **pound sign** or **hash** (**#**) are comments. This means that they are not interpreted by Nginx. Programmers use comments to explain what certain blocks of code do, or leave suggestions on how to edit values. The default file has several commented sections for explanatory purposes, and you can add your own if desired. Programmers will also leave optional statements as comments. They can be "turned on" if needed by removing the pound sign.

  An example of this is shown above. The directive `multi_accept on;` has a `#` in front of it. This isn't a comment for the user, but rather a directive that can be "activated" by removing the `#`.
- Settings begin with the *variable* name and then state an *argument* or series of arguments separated by spaces. Examples include `worker_processes 1;` and `error_log logs/error.log notice;`. All statements end with a semi-colon (**;**).
- Some settings, like the `events` variable above, have arguments that are themselves settings with arguments. All sub-directives are enclosed in one set of curly brackets (**{ }**).
- Brackets are sometimes nested inside each other for multiple sets of sub-directives. If you add or edit a section with nested brackets, make sure they all come in opening and closing pairs.
- White space characters (tabs, spaces, and new line characters) are not interpreted by Nginx. This doesn't mean that they aren't important, though! Using indentation in a standardized way will greatly improve the readability of your file and make it easier for you to keep up with maintenance in the long run.

Understanding How Nginx Works

Now that we understand the syntax, let's get into the nuts and bolts of Nginx. First we'll go over the core directives in the `nginx.conf` file, which define the basic behavior of the web server. Then we'll explain the `HTTP` block in greater detail and some of the more commonly adjusted variables. From there we'll move to the `server` block, and virtual host configuration files. This is the section you will edit the most when defining the websites you'll host with Nginx.

Defining the Directives

We'll begin by explaining the core directives in `/etc/nginx/nginx.conf` Let's go back to that first section:

```
user www-data;
worker_processes 4;
pid /run/nginx.pid;

events {
        worker_connections 768;
        # multi_accept on;
}
```

user
Defines which Linux system user will own and run the Nginx server. Most Debian-based distributions use `www-data` but this may be different in other distros. There are certain use cases that benefit from changing the user; for instance if you run two simultaneous web servers, or need another program's user to have control over Nginx.
worker_process
Defines how many threads, or simultaneous instances, of Nginx to run. You can learn more about this directive and the values of adjusting it here.
pid
Defines where Nginx will write its master process ID, or PID. The PID is used by the operating system to keep track of and send signals to the Nginx process.

HTTP (Universal Configuration)

The next section of the `nginx.conf` file covers the universal directives for Nginx as it handles HTTP web traffic. The first part of the HTTP block is shown below:

File excerpt: **/etc/nginx/nginx.conf**

```
http {

    ##
    # Basic Settings
    ##

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    # server_tokens off;

    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ##
    # Logging Settings
    ##

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    ##
    # Gzip Settings
    ##

    gzip on;
    gzip_disable "msie6";
```

Most of the `http { }` block should work as-is for most Nginx configurations. We do, however, want to draw your attention to the following configuration options:

include

The `include` statement at the beginning of this section *includes* the file `mime.types` located at `/opt/nginx/conf/mime.types`. What this means is that anything written in the file `mime.types` is interpreted as if it was written inside the `http { }` block. This lets you include a lengthy amount of information in the `http { }` block without having it clutter up the main configuration file. Try to avoid too many chained inclusions (i.e., including a file that itself includes a file, etc.) Keep it to one or two levels of inclusion if possible, for readability purposes. You can always include all files in a certain directory with the directive:

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

Or to be more specific, you can include all `.conf` files in a directory:

```
include /etc/nginx/conf.d/*.conf;
```

gzip

The `gzip` directive tells the server to use on-the-fly gzip compression to limit the amount of bandwidth used and speed up some transfers. This is equivalent to Apache's `mod_deflate`. Additional settings can be uncommented in this section of the http block to modify the gzip behavior:

File excerpt: **/etc/nginx/nginx.conf**

```
# gzip_vary on;
# gzip_proxied any;
# gzip_comp_level 6;
# gzip_buffers 16 8k;
# gzip_http_version 1.1;
# gzip_types text/plain text/css application/json application/x-javascript text/xml application/xml application/xml+rss text/javascript;
```

For full definitions of the gzip options, check out this page from the Nginx docs.

> Note
>
> If you keep gzip compression enabled here, note that you are trading increased CPU costs in exchange for your lower bandwidth use. Set the `gzip_comp_level` to a value between 1 and 9, where 9 requires the greatest amount of CPU resources and 1 requires the least. The default value is 1.

Note that the code snippet shown above does not include the closing bracket (**}**), because the HTTP section isn't finished. For detailed explanations of every directive in the `HTTP` block, check out this page from the Nginx documentation.

Server (Virtual Domains Configuration)

The HTTP block of the `nginx.conf` file contains the statement `include /etc/nginx/sites-enabled/*;`. This allows for server block configurations to be loaded in from separate files found in the `sites-enabled` sub-directory. Usually these are symlinks to files stored in `/etc/nginx/sites-available/`. By using symlinks you can quickly enable or disable a virtual server while preserving its configuration file. Nginx provides a single default virtual host file, which can be used as a template to create virtual host files for other domains:

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

Now let's go over the directives and settings that make up the `server` block:

File excerpt: **/etc/nginx/sites-available/default**

```
server {
        listen 80 default_server;
        listen [::]:80 default_server ipv6only=on;

        root /usr/share/nginx/html;
        index index.html index.htm;

        # Make site accessible from http://localhost/
        server_name localhost;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                try_files $uri $uri/ /index.html;
                # Uncomment to enable naxsi on this location
                # include /etc/nginx/naxsi.rules
        }
}
```

The `server` block is where the typical Nginx user will make most of his or her changes to the default configuration. Generally, you'll want to make a separate file with its own `server` block for each virtual domain on your server. More configuration options for the server block are shown in the following sections.

Listening Ports

The `listen` directive, which is located in the `server` block, tells Nginx the hostname/IP and the TCP port where it should listen for HTTP connections. By default, Nginx will listen for HTTP connections on port `80`.

Next we'll present a few common examples for the `listen` directive.

Note

You can use more than one `listen` directive, if needed.

File excerpt: **/etc/nginx/sites-available/default**

```
listen 80 default_server;
listen [::]:80 default_server ipv6only=on;
```

These are the default listen statements in the default virtual host file. The argument `default_server` means this virtual host will answer requests on port 80 that don't specifically match another virtual host's listen statement. The second statement listens over IPv6 and behaves in the same way.

File excerpt: **/etc/nginx/sites-available/example.com**

```
listen     127.0.0.1:80;
listen     localhost:80;
```

These two examples direct Nginx to listen on `127.0.0.1`; that is, the local loopback interface. `localhost` is conventionally set as the hostname for `127.0.0.1` in `/etc/hosts`.

File excerpt: **/etc/nginx/sites-available/example.com**

```
listen     127.0.0.1:8080;
listen     localhost:8080;
```

The third pair of examples also listen on localhost, but they listen for responses on port `8080` instead of port `80`.

File excerpt: **/etc/nginx/sites-available/example.com**

```
listen     192.168.3.105:80;
listen     192.168.3.105:8080;
```

The fourth pair of examples specify a server listening for requests on the IP address `192.168.3.105`. The first listens on port `80` and the second on port `8080`.

File excerpt: **/etc/nginx/sites-available/example.com**

```
listen     80;
listen     *:80;
listen     8080;
listen     *:8080;
```

The fifth set of examples tell Nginx to listen on *all* domains and IP addresses on a specific port. `listen 80;` is equivalent to `listen *:80;`, and `listen 8080;` is equivalent to `listen *:8080;`.

File excerpt: **/etc/nginx/sites-available/example.com**

```
listen     12.34.56.77:80;
listen     12.34.56.78:80;
listen     12.34.56.79:80;
```

Finally, the last set of examples instruct the server to listen for requests on port `80` for the IP addresses `12.34.56.77`, `12.34.56.78`, and `12.34.56.79`.

Name-Based Virtual Hosting

The `server_name` directive, which is located in the `server` block, lets the administrator provide *name-based virtual hosting*. This allows multiple domains to be served from a single IP address. The server decides which domain to serve based on the request header it receives (for example, when someone requests a particular URL).

Typically, you will want to create one file per domain you want to host on your server. Each file should have its own `server` block, and the `server_name` directive is where you specify which domain this file affects.

Next we'll present a few common examples for the `server_name` directive:

File excerpt: **/etc/nginx/sites-available/example.com**

```
server_name   example.com;
```

The example above directs Nginx to process requests for `example.com`. This is the most basic configuration.

File excerpt: **/etc/nginx/sites-available/example.com**

```
server_name   example.com www.example.com;
```

The second example instructs the server to process requests for both `example.com` and `www.example.com`.

File excerpt: **/etc/nginx/sites-available/example.com**

```
server_name   *.example.com;
server_name   .example.com;
```

These two examples are equivalent. `*.example.com` and `.example.com` both instruct the server to process requests for all subdomains of `example.com`, including `www.example.com`, `foo.example.com`, etc.

File excerpt: **/etc/nginx/sites-available/example**

```
server_name   example.*;
```

The fourth example instructs the server to process requests for all domain names beginning with `example.`, including `example.com`, `example.org`, `example.net`, `example.foo.com`, etc.

File excerpt: **/etc/nginx/sites-available/multi-list**

```
server_name   example.com linode.com icann.org;
```

The fifth example instructs the server to process requests for three different domain names. Note that any combination of domain names can be listed in a single `server_name` directive.

File excerpt: **/etc/nginx/sites-available/local**

```
server_name   localhost linode galloway;
```

Nginx allows you to specify names for virtual hosts that are not valid domain names. Nginx uses the name from the HTTP header to answer requests; it doesn't matter to Nginx whether the domain name is valid or not. In this case, the hostames can be specified in the /etc/hosts file.

Using non-domain hostnames may be useful if your Nginx server is deployed on a LAN, or if you already know all of the clients that will be making requests of the server. This includes front-end proxy servers that preconfigured `/etc/hosts` entries for the IP address on which Nginx is listening.

File excerpt: **/etc/nginx/sites-available/catchall**

```
server_name   "";
```

Finally, if you set `server_name` to the empty quote set ("“), Nginx will process all requests that either do not have a hostname, or that have an unspecified hostname, such as requests for the IP address itself.

> **Note**
>
> Individual names are separated with a space. You can use regular expressions if desired.

Access Logs

The `access_log` directive can be set in the `http` block in `nginx.conf` or in the `server` block for a specific virtual domain. It sets the location of the Nginx access log. By defining the access log to a different path in each `server` block, you can sort the output specific to each virtual domain into its own file. An `access_log` directive defined in the `http` block can be used to log all access to a single file, or as a catch-all for access to virtual hosts that don't define their own log files.

You can use a path relative to the current directory:

File excerpt: **/etc/nginx/nginx.conf**

```
access_log logs/example.access.log;
```

Or, you can use a full path:

File excerpt: **/etc/nginx/sites-available/example.com**

```
access_log /srv/www/example.com/logs/access.log;
```

You can also disable the access log, although this is not recommended:

File excerpt: **/etc/nginx/nginx.conf**

```
access_log off;
```

Location (File and Folder Configuration)

The final component of the `server` settings block is the `location` directive. We'll cover the specifics of what goes *inside* of a location block in the next section; right now we're going to focus on the basic path setting.

The `location` setting lets you configure how Nginx will respond to requests for resources within the server. Just like the `server_name` directive tells Nginx how to process requests for the domain, such as **http://example.com**, the `location` directive covers requests for specific files and folders, such as **http://example.com/blog/**.

> Note
>
> You can have more than one location directive.

Here are a few examples:

File excerpt: **/etc/nginx/sites-available/example.com**

```
location / { }
location /images/ { }
location /blog/ { }
location /planet/ { }
location /planet/blog/ { }
```

These first five examples are *literal string* matches, which match any part of an HTTP request that comes after the host segment. So, for example, if someone requests:

Request: `http://example.com/`

Returns: Assuming that there is a `server_name` entry for `example.com`, the `location /` directive will determine what happens with this request.

Nginx always fulfills request using the most specific match. So, for example:

Request: `http://example.com/planet/blog/` or `http://example.com/planet/blog/about/`

Returns: This is fulfilled by the `location /planet/blog/` setting because it is more specific, even though `location /planet/` also matches this request.

File excerpt: **/etc/nginx/sites-available/example.com**

```
location ~ IndexPage\.php$ { }
location ~ ^/BlogPlanet(/|/index\.php)$ { }
```

When a `location` directive is followed by a tilde (**\~**), Nginx performs a *regular expression* match. These matches are always case-sensitive. So, `IndexPage.php` would match the first example above, but `indexpage.php` would not. In the second example, the regular expression `^/BlogPlanet(/|/index\.php)$` will match requests for `/BlogPlanet/` and `/BlogPlanet/index.php`, but **not** `/BlogPlanet`, `/blogplanet/`, or `/blogplanet/index.php`. Nginx uses Perl Compatible Regular Expressions (PCRE).

File excerpt: **/etc/nginx/sites-available/example.com**

```
location ~* \.(pl|cgi|perl|prl)$ { }
location ~* \.(md|mdwn|txt|mkdn)$ { }
```

If you want matches to be case-*insensitive*, use a tilde with an asterisk (**\~***). The examples above all specify how Nginx should process requests that end in a particular file extension. In the first example, any file ending in: `.pl`, `.PL`, `.cgi`, `.CGI`, `.perl`, `.Perl`, `.prl`, and `.PrL` (among others) will match the request.

File excerpt: **/etc/nginx/sites-available/example.com**

```
location ^~ /images/IndexPage/ { }
location ^~ /blog/BlogPlanet/ { }
```

Adding a caret and tilde (**\^\~**) to your `location` directives tells Nginx, if it makes a match for a particular string, to stop searching for more specific matches and use the directives here instead. Other than that, these directives work like the literal string matches in the first group. So, even if there's a more specific match later, if a request matches one of these directives, the settings here will be used. See below for more information about the order and priority of `location` directive processing.

File excerpt: **/etc/nginx/sites-available/example.com**

```
location = / { }
```

Finally, if you add an equals sign (**=**) to the `location` setting, this forces an exact match with the path requested and then stops searching for more specific matches. For instance, the final example will match only `http://example.com/`, not `http://example.com/index.html`. Using exact matches can speed up request times slightly, which can be useful if you have some requests that are particularly popular.

Directives are processed in the following order:

1. Exact string matches are processed first. If a match is found, Nginx stops searching and fulfills the request.
2. Remaining literal string directives are processed next. If Nginx encounters a match where the **\^\~** argument is used, it stops here and fulfills the request. Otherwise, Nginx continues to process location directives.
3. All location directives with regular expressions (**\~** and **\~***) are processed. If a regular expression matches the request, Nginx stops searching and fulfills the request.
4. If no regular expressions match, the most specific literal string match is used.

Make sure each file and folder under a domain will match at least one `location` directive.

> **Note**
>
> While Nginx's configuration parser is technically capable of reading nested location blocks, this is neither recommended nor supported.

Location Root and Index

The `location` setting is another variable that has its own block of arguments.

Once Nginx has determined which `location` directive best matches a given request, the response to this request is determined by the contents of the associated `location` directive block. Here's an example:

File excerpt: **/etc/nginx/sites-available/example.com**

```
location / {
    root html;
    index index.html index.htm;
}
```

In this example, the document root is located in the `html/` directory. Given the default installation prefix for Nginx, the full path to this location is `/etc/nginx/html/`.

Request: `http://example.com/blog/includes/style.css`

Returns: Nginx will attempt to serve the file located at `/etc/nginx/html/blog/includes/style.css`

Note

You can use absolute paths for the `root` directive if desired.

The `index` variable tells Nginx which file to serve if none is specified. For example:

Request: `http://example.com`

Returns: Nginx will attempt to serve the file located at `/etc/nginx/html/index.html`.

If multiple files are specified for the `index` directive, Nginx will process the list in order and fulfill the request with the first file that exists. If `index.html` doesn't exist in the relevant directory, then `index.htm` will be used. If neither exist, a 404 message will be sent.

Here's a more complex example, showcasing a set of `location` directives taken approximately from the Nginx and Perl-FastCGI Guide for a server responding for the domain `example.com`:

File excerpt: **/etc/nginx/sites-available/example.com location directive**

```
location / {
    root   /srv/www/example.com/public_html;
    index  index.html index.htm;
}

location ~ \.pl$ {
    gzip off;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/fcgiwrap.socket;
    fastcgi_index index.pl;
    fastcgi_param SCRIPT_FILENAME /srv/www/www.example.com/public_html$fastcgi_script_name;
}
```

In this example, all requests for resources that end in a `.pl` extension are handled by the second location block, which specifies a `fastcgi` handler for these requests. Otherwise, Nginx uses the first location directive. Resources are located on the file system at `/srv/www/example.com/public_html/`. If no file name is specified in the request, Nginx will look for and provide the `index.html` or `index.htm` file. If no `index` files are found, the server will return a 404 error.

Let's analyze what happens during a few requests:

Request: `http://example.com/`

Returns: `/srv/www/example.com/public_html/index.html` if it exists. If that file doesn't exist, it will serve

/srv/www/example.com/public_html/index.htm. If neither exists, Nginx returns a 404 error.

Request: http://example.com/blog/

Returns: /srv/www/example.com/public_html/blog/index.html if it exists. If that file doesn't exist, it will serve /srv/www/example.com/public_html/blog/index.htm. If neither exists, Nginx returns a 404 error.

Request: http://example.com/tasks.pl

Returns: Nginx will use the FastCGI handler to execute the file located at /srv/www/example.com/public_html/tasks.pl and return the result.

Request: http://example.com/squire/roster.pl

Returns: Nginx will use the FastCGI handler to execute the file located at /srv/www/example.com/public_html/squire/roster.pl and return the result.

Best Practices

The examples and explanations in the previous sections should help you learn to configure your Nginx server with elegance and confidence. In this section, we'll look at a few best practices for keeping your Nginx configuration organized:

- Where possible, put your configurations in separate files, not all in the main configuration file.
- Make a separate configuration file for each of your domains. For example, an Nginx configuration file for **example.com** might be called /etc/nignx/sites-available/example.com as shown throughout this guide. It will contain a server block for that domain. The domain name is specified in the server_name variable.
- Use include statements in the main nginx.conf configuration to include each server configuration file. These include statements go in the http { } block of the main configuration file:

File excerpt: **/etc/nginx/nginx.conf**

```
http {
    # [...]

    include /srv/www/example.com/nginx.conf;

    # [...]
}
```

Note

You can also use a relative path, if desired.

- Name your files with a consistent pattern. Keeping your files well-organized and cleanly formatted will greatly reduce the burden of maintaining an Nginx server.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

- Nginx guides in the Linode Library
- Nginx community documentation
- LEMP Application Stack guides in the Linode Library

- Nginx Website
- Nginx log module documentation

Company

About Us

Blog

Logos

Careers

Contact

Contact Us

855-4-LINODE

(855-454-6633)

Email us

Facebook

Twitter

Google+

Linkedin

Github

Terms of Service

Privacy Policy

Security