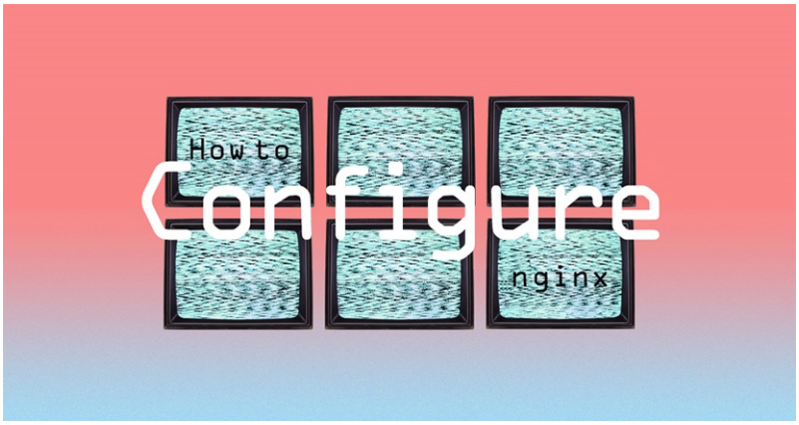
How to Configure NGINX

Updated Friday, April 5, 2019 by Linode

Written by Linode

Contribute on GitHub Report an Issue | View File | Edit File



NGINX is a lightweight, high-performance web server designed for high-traffic use cases.

One of NGINX's strongest features is the ability to efficiently serve static content such as HTML and media files. NGINX uses an asynchronous event-driven model which provides predictable performance under load.

NGINX hands off dynamic content to CGI, FastCGI, or other web servers such as Apache. This content is then passed back to NGINX for delivery to the client. This document will familiarize you with basic NGINX parameters and conventions.

Directives, Blocks, and Contexts

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

Configuration options in NGINX are called directives. Directives are organized into groups known as **blocks** or **contexts**. The two terms are synonymous.

Lines preceded by a `#` character are comments and not interpreted by NGINX. Lines containing directives must end with a `;` or NGINX will fail to load the configuration and report an error.

Below is a condensed copy of the `/etc/nginx/nginx.conf` file that is included with installations from the NGINX repositories. The file starts with 4 directives: `user`, `worker_processes`, `error_log`, and `pid`. These are outside any specific block or context, so they're said to exist in the `main` context. The `events` and `http` blocks are areas for additional directives, and they also exist in the `main` context.

See the NGINX docs for explanations of these directives and others available in the `main` context.

/etc/nginx/nginx.conf

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    . . .
}

http {
    . . .
}
```

The http Block

The `http` block contains directives for handling web traffic. These directives are often referred to as *universal* because they are passed on to all website configurations NGINX serves. See the NGINX docs for a list of available directives for the `http` block.

/etc/nginx/nginx.conf

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

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

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

Server Blocks

The `http` block above contains an `include` directive which tells NGINX where website configuration files are located.

- If you installed from the official NGINX repository, this line will say `include /etc/nginx/conf.d/*.conf;` as it does in the `http` block above. Each website you host with NGINX should have its own configuration file in `/etc/nginx/conf.d/`, with the name formatted as `example.com.conf`. Sites which are disabled (not being served by NGINX) should be named `example.com.conf.disabled`.
- If you installed NGINX from the Debian or Ubuntu repositories, this line will say `include /etc/nginx/sites-enabled/*;`. The `../sites-enabled/` folder contains symlinks to the site configuration files stored in `/etc/nginx/sites-available/`. Sites in `sites-available` can be disabled by removing the symlink to `sites-enabled`.
- Depending on your installation source, you'll find an example configuration file at `/etc/nginx/conf.d/default.conf` or `etc/nginx/sites-enabled/default`.

Regardless of the installation source, server configuration files will contain a `server` block (or blocks) for a website. For example:

/etc/nginx/conf.d/example.com.conf

```
server {
    listen       80 default_server;
    listen       [::]:80 default_server;
    server_name  example.com www.example.com;
    root         /var/www/example.com;
    index        index.html;
    try_files $uri /index.html;
}
```

Listening Ports

The `listen` directive tells NGINX the hostname/IP and the TCP port where it should listen for HTTP connections. 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 similarly.

Name-Based Virtual Hosting

The `server_name` directive 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.

You typically should create one file per domain or site you want to host on your server. Here are some examples:

1. Process requests for both `example.com` and `www.example.com`:

   /etc/nginx/conf.d/example.com.conf

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

2. The `server_name` directive can also use wildcards. `*.example.com` and `.example.com` both instruct the server to process requests for all subdomains of `example.com`:

   /etc/nginx/conf.d/example.com.conf

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

3. Process requests for all domain names beginning with `example.`:

   /etc/nginx/conf.d/example.com.conf

   ```
   server_name  example.*;
   ```

NGINX allows you to specify server names that are not valid domain names. NGINX uses the name from the HTTP header to answer requests, regardless of whether the domain name is valid or not.

Using non-domain hostnames is useful if your server is 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 with `/etc/hosts` entries configured for the IP address on which NGINX is listening.

Location Blocks

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, `location` directives cover requests for specific files and folders, such as `http://example.com/blog/`. Here are some examples:

/etc/nginx/sites-available/example.com

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

The locations above are *literal string* matches, which match any part of an HTTP request that comes after the host segment:

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 requests using the most specific match:

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

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

/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).

/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.

/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 matches 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. 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.

/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**
> Nested location blocks are not recommended or 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:

/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. Under 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 exists, a 404 message will be sent.

Here's a more complex example, showcasing a set of `location` directives for a server responding to the domain `example.com`:

/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/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/username/roster.pl`

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

> **Still have a few questions?**
> Join our Community and post your questions for other Linode and Linux enthusiasts to help you out.

Related Questions:

- Nginx troubleshooting
- Nginx font rules
- How to set up SSL with NGINX

Join our Community

Find answers, ask questions, and help others.

Write for Linode.

We're always expanding our docs. If you like to help people, can write, and have expertise in a Linux or cloud infrastructure topic, learn how you can contribute to our library.

Get started in the Linode Cloud today.

Create an Account