

# NGINX Configuration Guide

By [Elvis Plesky](#) | [February 24, 2021](#) | [Various](#)

 Minutes

NGINX is a web server designed for use cases involving high volumes of traffic. It's a popular, lightweight, high-performance solution.

One of its many impressive features is that it can serve static content (media files, HTML) efficiently. [NGINX](#) utilizes an asynchronous event-driven model, delivering reliable performance under significant loads.

This web server hands dynamic content off to FastCGI, [CGI](#), or alternative servers (including Apache), before it's sent back to NGINX for delivery to clients.

In this guide on how to configure NGINX, we'll explore the essentials of NGINX to help you understand how it works and what benefits it offers.

- [NGINX Configuration: Understanding Directives](#)
- [What are Location Blocks?](#)





## Efficient Management and Powerful Monitoring, Combined.

Easily manage your servers, sites and licenses while monitoring your performance. All in one location.

Start Now

# NGINX Configuration: Understanding Directives

Every NGINX configuration file will be found in the `/etc/nginx/` directory, with the main configuration file located in `/etc/nginx/nginx.conf`.

NGINX configuration options are known as “directives”: these are arranged into groups, known interchangeably as **blocks** or **contexts**.

When a `#` appears before a line, these are comments and NGINX won’t interpret them. Lines that contain directives should end with a semicolon (`;`). If not, NGINX will be unable to load the configuration properly and report an error.

Below, we’ve included a shortened copy of the `/etc/nginx/nginx.conf` file included with installations from NGINX repositories. This file begins with four directives:

- **user**
- **worker\_processes**
- **error\_log**
- **pid**

These exist outside any particular context or block, and are said to be within the **main** context.

Additional directives are found within the **events** and **http** blocks, and these also exist within the **main** context.

File: `/etc/nginx/nginx.conf`



```
user nginx;  
  
worker_processes 1;
```

```
events {
```

```
    . . .
```

```
}
```

```
http {
```

```
    . . .
```

```
}
```

## What is the Http Block?

The **http** block includes directives for web traffic handling, which are generally known as *universal*. That's because they get passed on to each website configuration served by NGINX.

File: /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;
```



```
}
```

## What are Server Blocks?

The **http** block shown above features an **include** directive. This informs NGINX where website configuration files can be found.

- When installing from NGINX's official repository, the line will read `include /etc/nginx/conf.d/*.conf;` just as you can see in the **http** block placed above. Every website hosted with NGINX should feature a unique configuration file in `/etc/nginx/conf.d/`, and the name will be formatted as `example.com.conf`. Those sites that have been disabled — not served by NGINX — should be titled `example.com.conf.disabled`.
- When installing NGINX from the Ubuntu or Debian repositories, the line will read: `include /etc/nginx/sites-enabled/*;`. The `../sites-enabled/` folder will include symlinks to the site configuration files located within `/etc/nginx/sites-available/`. You can disable sites within `sites-available` if you take out the symlink to `sites-enabled`.
- According to the installation source, an illustrative configuration file can be found at `/etc/nginx/conf.d/default.conf` or `etc/nginx/sites-enabled/default`.

No matter the installation source, though, server configuration files feature one or more server blocks for a site. As an example, let's look at the below:

File: `/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;  
  
}
```





must listen for HTTP connections.

The argument `default_server` means that this virtual host will be answering requests on port 80 which don't match the listen statement of a separate virtual host. When it comes to the second statement, this will listen over [IPv6](#) and demonstrate similar behavior.

## What is Name-based Virtual Hosting?

The `server_name` directive enables a number of domains to be served from just one IP address, and the server will determine which domain it will serve according to the request header received.

Generally, you should create one file for each site or domain you wish to host on your server. Let's delve into some examples:

1. Process requests for `example.com` and `www.example.com`:
2. The `server_name` directive can utilize wildcards. `*.example.com` and `.example.com` tell the server to process requests for all `example.com` subdomains:

File: `/etc/nginx/conf.d/example.com.conf`

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

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

File: `/etc/nginx/conf.d/example.com.conf`

```
server_name example.*;
```

With NGINX, you can define server names that are invalid domain names: it utilizes the name from the HTTP header to answer requests regardless of whether the domain name is valid or invalid.

You may find non-domain hostnames helpful if your server is on a LAN or you know all the clients likely to make requests on the server. This encompasses front-end proxy servers with `/etc/hosts` entries set up for the IP address NGINX is listening on.

## What are Location Blocks?

`http://example.com/blog/`).

Let's consider a few examples:

File: `/etc/nginx/sites-available/example.com`

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

These locations are literal string matches and match any part of an HTTP request following the host segment:

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

**Returns:** Let's assume there's a `server_name` entry for `example.com`. In this case, the `location /` directive determines what occurs with this request.

With NGINX, requests are always fulfilled with the most specific match possible:

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

**Returns:** This will be fulfilled by the `location /planet/blog` directive as it's more specific, despite `location /planet` being a match too.

File: `/etc/nginx/sites-available/example.com`

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

When location directives are followed by a `~` (tilde), NGINX will perform a regular expression match, which is always case-sensitive.

For example, `IndexPage.php` would be a match with the first of the above examples, while `indexpage.php` wouldn't.



What if you prefer matches to be case-insensitive? Well, you should use a tilde followed closely by an asterisk: `~*`. You can see the above examples define that NGINX should process requests ending in a certain file extension: the first example determines that files ending in `.pl`, `PL`, `.cgi`, `.perl`, `.Perl`, `.prl`, and `.PrL` (as well as others) will all be a match for the request.

File: `/etc/nginx/sites-available/example.com`

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

When you add a caret and a tilde (`^~`) to location directives, you're informing NGINX that, should it match a particular string, it should stop searching for more specific matches and utilize these directives here instead.

Beyond this, these directives function as the literal string matches do in the first group. Even if a more specific match comes along at a later point, the settings will be utilized if a request is a match for one of these directives.

Now, let's look at additional details on location directive processing.


File: `/etc/nginx/sites-available/example.com`

```
location = / { }
```

Finally, adding an equals symbol to the location setting forces an exact match with the requested path and ends searching for matches that are more specific.

So, for example, the last example will be a match for `http://example.com` only, as opposed to `http://example.com/index.html`. If you use exact matches, you can enhance the speed of request times moderately. This can prove beneficial if certain requests are especially popular.

The processing of directives will follow this sequence flow:

1. Exact string matches will be processed first: NGINX stops searching if a match is located and will fulfill the request.
2. Any remaining literal string directives will be processed next. NGINX will stop and fulfill a request if it finds a match using the `^~` argument. If not, NGINX will continue processing of location directives.
-  3. Each location directive with a regular expression (`~` and `~*`) will get processed next. If a regular expression is a match for the request, NGINX will end its search and fulfill the

Be sure that every file and folder found under a domain is a match for one or more location directives.

Nested location blocks are not recommended and not supported.

## How to Use Location Root and Index

The location setting is another variable with its own arguments block.

When NGINX has identified the location directive that is the best match for a specific request, its response will be based on the associated location directive block's contents. So, for instance:

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

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

We can see, in this example, that the document root is based in the html/ directory. Under the NGINX default installation prefix, the location's full path is /etc/nginx/html/.

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

**Returns:** NGINX will try to serve the file found at /etc/nginx/html/blog/includes/style.css

### Please note:

Absolute paths for the root directive can be used if you wish. The index variable informs NGINX which file it should serve when or if none are specified.

So, for instance:

**Request:** http://example.com

**Returns:** NGINX will try to serve the file found at /etc/nginx/html/index.html.

When a number of files are specified for the index directive, the list will be processed in order and NGINX will fulfill the request with the first file found to exist. If index.html can't be located







Let's consider a more complicated example that showcases a number of location directives for a server responding to the example domain:

File: /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;  
  
}
```

Here, we can see that the second location block handles all requests for resources ending in a .pl extension, and it specifies a fastcgi handler for them. NGINX will use the first location directive otherwise.

Resources are found on the file system at /srv/www/example.com/public\_html/. When no exact file names are defined in the request, NGINX will search for the index.html or index.htm file and provide it. A 404 error message will be returned if zero index files are located.

Let's consider what takes place during a number of example requests:

**Request:** http://example.com/

**Returns:** /srv/www/example.com/public\_html/index.html if this exists. Otherwise, it will serve /srv/www/example.com/public\_html/index.htm. And if both of these don't exist, a 404 error will be provided.





found because it doesn't exist, a `/srv/www/example.com/public_html/blog/index.html` will be served. If neither exists, NGINX will return a 404 error.

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

**Returns:** NGINX will take advantage of the FastCGI handler to execute the file found at `/srv/www/example.com/public_html/tasks.pl` and return the relevant result.

**Request:** `http://example.com/username/roster.pl`

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

This ends our guide on how to configure NGINX. We hope it helps you get started and set up in next to no time!



### Elvis Plesky

Our fun and curious team mascot's always plugged into the latest trends. He's here to share his knowledge and help you solve your tech problems.

No comment yet, add your voice below!

## Add a Comment

Your email address will not be published. Required fields are marked \*

Comment \*

Name \*





WEBSITE

[Submit](#)

### GET LATEST NEWS AND TIPS

Yes, please, I agree to receiving my personal Plesk Newsletter! Plesk International GmbH and other WebPros group companies may store and process the data I provide for the purpose of delivering the newsletter according to the [Plesk Privacy Policy](#). In order to tailor its offerings to me, Plesk may further use additional information like usage and behavior data (Profiling). I can unsubscribe from the newsletter at any time by sending an email to [privacy@plesk.com](mailto:privacy@plesk.com) or use the unsubscribe link in any of the newsletters.

[Submit](#)

### RELATED POSTS

#### NGINX vs Apache – Which Is the Best Web Server in 2023?

[Read More »](#)

#### NGINX Performance Tuning Tips

[Read More »](#)

#### Litespeed vs NGINX

[Read More »](#)

### KNOWLEDGE BASE



#### Apache and Nginx Configuration Files

[Read More »](#)



## How to add custom Apache/nginx configuration for a domain in Plesk

[Read More »](#)

## Webmail shows 504 Gateway Time-out

[Read More »](#)

## Invalid nginx configuration: nginx: [emerg] invalid number of arguments in “add\_header”

[Read More »](#)

## nginx configuration, listen to 0.0.0.0

[Read More »](#)

### Industry Partners



AUTOMA

COMPANY

PRODUCT

KNOWLEDGE BASE

PROGRAMS

COMMUNITY

Follow us:



© 2023 Plesk International GmbH. All rights reserved. Plesk and the Plesk logo are trademarks of Plesk International GmbH.

Managed with ❤️ with [Plesk WP Toolkit](#)

