

# Secure NGINX: Deploy Your Scalable Proxy or Load Balancer

Check GitHub for helpful DevOps tools:

## Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

 <https://github.com/MichaelRobotics>



Ask Personal AI Document assistant to learn interactively (FASTER)!

1


Download PDF

1

<https://github.com/MichaelRobotics/DevOpsTools/blob/main/SecureNgnix.pdf>

2

Go to website

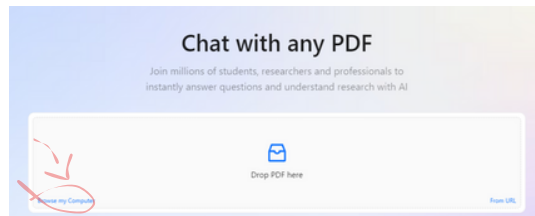
 | Click there to go to ChatPdf website

2

3

Browse file

3



4

Chat with Document

Ask questions about document!

4

# Completly new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

 <https://www.hackthebox.com/>



## What is NGNIX?

Nginx is a high-performance web server and reverse proxy server that efficiently handles large amounts of concurrent connections. It is also used for load balancing, caching, and serving static content.

## How to implement NGNIX?

To implement Nginx, install it on your server using your package manager (e.g., apt install nginx on Ubuntu) and configure the nginx.conf file for your specific needs. Start the Nginx service to begin handling web traffic.

# NGNIX: Why and When

Nginx is ideal for improving website performance, handling high traffic, and efficiently managing server resources.

For example, it can be used as a load balancer to distribute traffic across multiple servers, ensuring that no single server is overwhelmed during peak times.

## System Requirements

- 1 GB RAM (minimum, 2 GB recommended for better performance)
- 10 GB free storage (more may be required depending on the size of the DNS zone files and logs)
- Ubuntu 22.04

## NGNIX: Main components & packages

- `nginx-core` – The core package providing the main web server and reverse proxy functionalities.
- `nginx-extras` – Includes additional modules like HTTP headers, geo-location, and more advanced features.
- `nginx-full` – Comprehensive package with core features plus extra modules for enhanced functionality.
- `nginx-module-ssl` – Adds SSL/TLS encryption support for secure communication over HTTPS.

# NGINX: installation, static content, location context

## 1) Installation

Really easy to setup basic install:

```
$ sudo apt update
$ sudo apt install nginx
```

## 2) Basics

Nginx serves static content, such as HTML, CSS, JavaScript, images, videos, fonts, and other assets. To serve static files with Nginx, you need to configure it to point to the directory where these files are stored.

create directory for static files:

```
/var/www/html/
```

And fill it with html files and images like that:

```
/var/www/html/
├── index.html
└── images/
    └── logo.png
```

## Serve static content:

Open the Nginx configuration file (nginx.conf) or create a new configuration file inside the **/etc/nginx/conf.d/** directory (for example, static-site.conf).

Here's a basic example of the configuration:

```
server {  
    listen 80; # Listen on port 80  
(HTTP)  
    server_name example.com; #  
Replace with your domain name or IP  
address  
  
    # Define the root directory where  
the static files are located  
    root /var/www/html;  
  
    # The default file to serve when the  
client doesn't specify a file  
    index index.html;  
  
    # Set up location block to serve  
static files  
    location / {  
        try_files $uri $uri/ =404;  
    }  
}
```

## Explanation of the Configuration

- **listen 80;** Tells Nginx to listen on port 80, which is the default port for HTTP traffic.

- **server\_name example.com;** Defines the domain name or IP address that this server block will respond to. Replace example.com with your domain or leave it as \_ (underscore) to catch all requests.
- **root /var/www/html;** Sets the root directory where Nginx will look for files.
- **index index.html;** Specifies the default file to serve when a directory is requested (e.g., when someone visits http://example.com, Nginx will serve /var/www/html/index.html).
- **location / { try\_files \$uri \$uri/ =404; }:**
  - The location block defines how to handle requests for the root path /.
  - **try\_files \$uri \$uri/ =404;** tells Nginx to check if the file exists (\$uri), then check if a directory exists (\$uri/). If neither is found, Nginx will return a 404 Not Found error.

### 3) Location context

The location context in Nginx is used to define how the server should respond to different types of requests based on the URL path. Each location block specifies rules and configurations that apply when a request matches a particular pattern.

Nginx uses different modifiers to determine how it matches the pattern against the request's URL.

Take for example this server block:

```

server {
    listen 8080;
    root /Users/user/Desktop/mysite;

    location ~* /count/[0-9] {
        root /Users/user/Desktop/mysite;
        try_files /index.html =404;
    }
    location /fruits {
        root /Users/user/Desktop/mysite;
    }
    location /carbs {
        alias /Users/user/Desktop/mysite/fruits;
    }
}

```

### 1. **location ~\* /count/[0-9]:**

- Matches URLs like /count/1, /count/9.
- Uses a case-insensitive regex.
- Tries to serve /index.html or returns 404 if not found.

### 2. **location /fruits:**

- Matches any URL starting with /fruits.
- Uses root to serve files, e.g., /fruits/apple.html -> /Users/user/Desktop/mysite/fruits/apple.html.
- Note: Duplicate /fruits blocks are not allowed; remove extras.

### 3. **location /carbs:**

- Uses alias to directly map /carbs to /Users/user/Desktop/mysite/fruits.
- Example: /carbs/orange.html serves from /Users/user/Desktop/mysite/fruits/orange.html.

## 4) Rewrite & Redirect

Redirect (return): Sends the client to a different URL, changing the browser address. Commonly used for 301 (permanent) or 302 (temporary) redirects.

Rewrite (rewrite): Changes the URL internally on the server without altering the browser address, or can also be used to redirect.

### Permanent Redirect (301)

```
location /old-page {  
    return 301 /new-page;  
}
```

- Explanation: Any request to /old-page will be permanently redirected to /new-page.
- 301 is a permanent redirect, meaning search engines will update their links.

### Redirect to an External URL

```
location /old-site {  
    return 301 https://newsite.com;  
}
```

- Redirects /old-site to https://newsite.com.

### Internal Rewrite

```
location /blog {  
    rewrite ^/blog/(.*)$ /new-blog/$1 permanent;  
}
```

- Rewrites /product/shirt/123 to /item/shirt?id=123 internally.



# NGNIX: Reverse proxy

Here's how to set up a basic reverse proxy in Nginx:

```
server {  
    listen 80;  
  
    server_name example.com;  
  
    location / {  
        proxy_pass http://backend-server;  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
        proxy_set_header X-Forwarded-Proto $scheme;  
    }  
}
```

## Explanation

1. **listen 80;**: The server listens on port 80 (HTTP).
2. **server\_name example.com;**: The domain name that this server will respond to.
3. **location / { ... }**: All requests are routed through this block.

## Key Directives

- **proxy\_pass http://backend-server;**: Forwards requests to backend-server (can be an IP or domain, e.g., http://127.0.0.1:8080).
- **proxy\_set\_header**: Ensures the correct headers are sent to the backend:
  - **Host \$host;**: Passes the original Host header.
  - **X-Real-IP \$remote\_addr;**: Sends the client's actual IP address.
  - **X-Forwarded-For \$proxy\_add\_x\_forwarded\_for;**: Keeps track of client IPs across proxies.
  - **X-Forwarded-Proto \$scheme;**: Passes the request scheme (http or https).

# NGNIX: LoadBalancer

An Nginx load balancer distributes incoming client requests across multiple backend servers. This ensures that no single server is overwhelmed, leading to better performance, reliability, and availability. Nginx can also monitor the health of backend servers and route traffic only to those that are functioning correctly.

```
http {  
    upstream backend {  
        server server1.example.com;  
        server server2.example.com;  
        server server3.example.com;  
    }  
  
    server {  
        listen 80;  
        server_name example.com;  
  
        location / {  
            proxy_pass http://backend;  
            proxy_set_header Host $host;  
            proxy_set_header X-Real-IP $remote_addr;  
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
            proxy_set_header X-Forwarded-Proto $scheme;  
        }  
    }  
}
```

## Explanation

### 1. **upstream backend { ... }:**

- Defines a group of backend servers (server1.example.com, server2.example.com, server3.example.com) that will receive requests.
- Nginx will distribute traffic among these servers.

### 2. **Load Balancing Algorithms:**

- By default, Nginx uses a Round Robin algorithm, distributing requests evenly.
- Other algorithms:
  - `least_conn`: Sends requests to the server with the fewest active connections.
  - `ip_hash`: Ensures that the same client IP always connects to the same server.

### 3. **server Block:**

- `proxy_pass http://backend;`: Sends requests to the defined upstream group.
- `proxy_set_header`: Passes headers to maintain client information.

# NGINX: Secure configuration

## 1) Disable Server Tokens

To prevent Nginx from revealing its version number (which can be useful to attackers), disable server tokens:

```
http {  
    server_tokens off;  
}
```

## 2) Hide Nginx Header

Remove or modify the Server header to further obscure details about the server software:

```
proxy_hide_header X-Powered-By;  
more_clear_headers Server;
```

## 3) Configure IP Address Access Control

Restrict access to certain parts of your server based on IP addresses:

```
location /admin {  
    allow 192.168.1.0/24; # Allow access only from this subnet  
    deny all;           # Deny access to everyone else  
}
```

# common troubleshooting

## 1) Nginx Not Starting or Restarting

Problem: Nginx fails to start, stop, or restart.

Solution: Check configuration syntax with `nginx -t` to find errors. Make sure no other service is using the same port (`netstat -tuln | grep 80` for port 80)

## 2) 404 Not Found Error

Problem: Requests are not finding the expected content, leading to "404 Not Found" errors.

Solution: Verify that the root or alias paths in your Nginx configuration are correct and point to existing files. Use `nginx -t` to check the configuration for errors.

## 3) Nginx is Running but Website is Down

Problem: Nginx appears to be running, but the website is not accessible.

Solution: Ensure backend servers (e.g., PHP, Node.js) are running. Verify proxy settings in Nginx (`proxy_pass`). Check DNS records to confirm they point to the correct IP.

## 4) Just learn more about nginx, go to their webswite

## 5) If everything is a complete mess (in case of wrong user configuration)

Remove the `ngni` and revert the configuration to its previous state.

## Learn more about Linux Security

**Check TecMint and Chris Linux Tech, they have great docs!**

The 3 Biggest Security Mistakes Linux Users Make


Security is a journey, not a destination

 <https://christitus.com/linux-security-mistakes/>



26 Security Hardening Tips for Modern Linux Servers

Everybody says that Linux is secure by default, and to some extent

 <https://www.tecmint.com/linux-server-hardening-security-tips/>



## Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

**Check my GitHub**

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

<https://github.com/MichaelRobotics>



*PS.*

*If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!*