## **NGINX**

## **🔹 What is NGINX?**

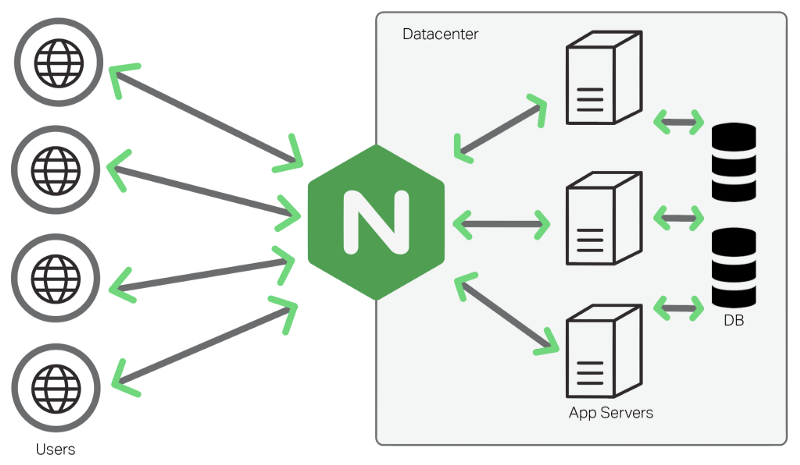
NGINX is an open-source, high-performance web server, reverse proxy, load balancer, and HTTP cache.

### **✅ Why Use NGINX?**

✔ High Performance – Handles 10,000+ concurrent connections with low memory usage.  
✔ Scalability – Efficiently distributes traffic (load balancing).  
✔ Security – Supports TLS/SSL, rate limiting, and DDoS protection.  
✔ Flexibility – Works as a static file server, API gateway, or reverse proxy (e.g., for Node.js, Python apps).

### **📌 Key Use Cases**

1. Web Server – Serve static/dynamic content.
2. Reverse Proxy – Route requests to backend apps (Node.js, Django, Flask).
3. Load Balancer – Distribute traffic across multiple servers.
4. API Gateway – Manage microservices routing.
5. Caching Layer – Accelerate content delivery.
6. Compression, Security, One Entry Point



## **🔹 Installing NGINX (Production Best Practices)**

### **Step 1: Installation**

#### **On Ubuntu/Debian:**

bash

sudo apt update && sudo apt install nginx -y

sudo systemctl start nginx

sudo systemctl enable nginx

📌 Best Practice:  
✔ Always use the latest stable version (nginx -v).  
✔ Configure automatic updates (unattended-upgrades on Ubuntu).

## **🔹 NGINX as a Web Server (Static & Dynamic Content)**

### **Basic Configuration (**/etc/nginx/nginx.conf**)**

user www-data;

worker\_processes auto; # Optimizes CPU usage

pid /run/nginx.pid;

events {

worker\_connections 1024; # Adjust based on server capacity

multi\_accept on;

}

http {

include /etc/nginx/mime.types;

default\_type application/octet-stream;

# Performance Tuning

sendfile on;

tcp\_nopush on;

tcp\_nodelay on;

keepalive\_timeout 65;

types\_hash\_max\_size 2048;

# Logging

access\_log /var/log/nginx/access.log;

error\_log /var/log/nginx/error.log;

# Gzip Compression

gzip on;

gzip\_types text/css application/javascript;

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

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

}

📌 Best Practice:  
✔ Set worker\_processes = CPU cores (auto detects it).  
✔ Enable gzip for faster content delivery.

## **🔹 NGINX as a Reverse Proxy (For Node.js, Python, etc.)**

### **Example: Proxying a Node.js App**

server {

listen 80;

server\_name example.com;

location / {

proxy\_pass http://localhost:3000; # Your app's port

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

}

📌 Best Practice:  
✔ Use HTTP/2 (listen 443 ssl http2) for better performance.  
✔ Always set proxy\_set\_header for proper request forwarding.

## **🔹 NGINX as a Load Balancer**

### **Example: Round-Robin Load Balancing**

upstream backend {

server 10.0.0.1:3000;

server 10.0.0.2:3000;

server 10.0.0.3:3000;

}

server {

listen 80;

server\_name example.com;

location / {

proxy\_pass http://backend;

}

}

### **Load Balancing Methods**

| Method | Description | Use Case |
| --- | --- | --- |
| Round Robin (Default) | Distributes requests evenly | General purpose |
| Least Connections | Sends traffic to the least busy server | Long-lived connections |
| IP Hash | Sticky sessions based on client IP | Session persistence |

📌 Best Practice:  
✔ Use health checks (max\_fails=2 fail\_timeout=30s).  
✔ Consider session persistence (if needed).

## **🔹 Securing NGINX (Production-Grade Security)**

### **1. Enable HTTPS (Let’s Encrypt with Certbot)**

sudo apt install certbot python3-certbot-nginx -y

sudo certbot --nginx -d example.com

* Automatically configures TLS/SSL and HTTP → HTTPS redirect.

### **2. Rate Limiting (Prevent DDoS & Brute Force)**

limit\_req\_zone $binary\_remote\_addr zone=req\_limit:10m rate=10r/s;

server {

location /login {

limit\_req zone=req\_limit burst=20 nodelay;

proxy\_pass http://backend;

}

}

### **3. Security Headers**

add\_header X-Frame-Options "SAMEORIGIN";

add\_header X-Content-Type-Options "nosniff";

add\_header X-XSS-Protection "1; mode=block";

add\_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

📌 Best Practice:  
✔ Renew SSL certs automatically (certbot renew --dry-run).  
✔ Use fail2ban to block malicious Ips.

## **🔹 Performance Optimization**

### **1. Enable Caching (For Static & Dynamic Content)**

proxy\_cache\_path /var/cache/nginx levels=1:2 keys\_zone=my\_cache:10m inactive=60m;

server {

location / {

proxy\_cache my\_cache;

proxy\_cache\_valid 200 60m;

}

}

### **2. Tune Kernel & NGINX for High Traffic**

# Increase file descriptor limits

echo "fs.file-max = 100000" | sudo tee -a /etc/sysctl.conf

sudo sysctl -p

# Increase NGINX worker connections

worker\_connections 4096;

📌 Best Practice:  
✔ Monitor nginx\_status for real-time metrics.  
✔ Use HTTP/2 and Brotli compression for faster loads.

## **🚀 Production Checklist**

✅ Install & Configure NGINX (Latest stable version)  
✅ Enable HTTPS (Certbot + Auto-renewal)  
✅ Set Up Reverse Proxy / Load Balancer  
✅ Implement Security Headers & Rate Limiting  
✅ Optimize Performance (Caching, HTTP/2, Compression)  
✅ Monitor Logs (/var/log/nginx/error.log)

## **📚 Official Docs & Further Reading**

* [NGINX Official Documentation](https://nginx.org/en/docs/)
* [NGINX Load Balancing](https://docs.nginx.com/nginx/admin-guide/load-balancer/)
* [Certbot (Let’s Encrypt)](https://certbot.eff.org/)