Nginx Cheatsheet

Nginx Cheatsheet

Installation

```
# Debian/Ubuntu
sudo apt update
sudo apt install nginx

# RHEL/CentOS
sudo yum install epel-release
sudo yum install nginx

# Alpine
sudo apk add nginx

# Verify installation
nginx -v
systemctl status nginx

# Start/enable service
sudo systemctl start nginx
sudo systemctl enable nginx
```

Basic Commands

```
# Check configuration
sudo nginx -t
sudo nginx -T

# Reload configuration (no downtime)
sudo systemctl reload nginx

# Restart service
sudo systemctl restart nginx

# Stop service
sudo systemctl stop nginx

# View logs
sudo tail -f /var/log/nginx/access.log
```

```
sudo tail -f /var/log/nginx/error.log

# Check syntax before reload
sudo nginx -t && sudo systemctl reload nginx
```

Configuration Structure

```
# /etc/nginx/nginx.conf (main configuration)
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
    use epoll;
    multi_accept on;
}

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

    # Logging
    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;

    # Performance
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    # Gzip
    gzip on;
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_types text/plain text/css text/xml application/json application/javascript;

    # Include site configs
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

Server Blocks

```
# Basic static site
server {
    listen 80;
    server_name example.com www.example.com;

    root /var/www/example.com;
```

```
    index index.html index.htm;

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

    location / {
        try_files $uri $uri/ =404;
    }

    # Deny access to hidden files
    location ~ /\. {
        deny all;
        access_log off;
        log_not_found off;
    }
}

# HTTP to HTTPS redirect
server {
    listen 80;
    server_name example.com www.example.com;
    return 301 https://$server_name$request_uri;
}

# HTTPS with SSL
server {
    listen 443 ssl http2;
    server_name example.com www.example.com;

    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    # HSTS
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

    root /var/www/example.com;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Reverse Proxy

```
# Basic reverse proxy
server {
    listen 80;
    server_name app.example.com;

    location / {
        proxy_pass http://127.0.0.1:3000;
        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;

        proxy_redirect http:// https://;
    }
}

# Reverse proxy with WebSocket support
server {
    listen 80;
    server_name websocket.example.com;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        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;
    }
}

# Load balancer (upstream)
upstream backend_servers {
    server 192.168.1.10:3000;
    server 192.168.1.11:3000;
    server 192.168.1.12:3000;
}

server {
    listen 80;
    server_name app.example.com;

    location / {
        proxy_pass http://backend_servers;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}

# Load balancer with health checks and backup
upstream backend_servers {
    server 192.168.1.10:3000 weight=3;
    server 192.168.1.11:3000 max_fails=3 fail_timeout=30s;
    server 192.168.1.12:3000 backup;
}
```

Location Blocks

```
# Exact match
location = /health {
    access_log off;
    return 200 "OK";
}
```

```
# Regex match
location ~ \.php$ {
    fastcgi_pass unix:/var/run/php/php8.1-fpm.sock;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include fastcgi_params;
}

# Case-insensitive regex
location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
    expires 1y;
    add_header Cache-Control "public, immutable";
}

# Prefixed match (longest wins)
location /api/ {
    proxy_pass http://backend_servers;
}

# Nested locations
location /app {
    location ~ \.php$ {
        fastcgi_pass unix:/var/run/php/php8.1-fpm.sock;
    }
}
```

Access Control

```
# IP whitelist
location /admin {
    allow 192.168.1.0/24;
    allow 10.0.0.0/8;
    deny all;
}

# Basic authentication
location /private {
    auth_basic "Restricted Area";
    auth_basic_user_file /etc/nginx/.htpasswd;

    # Create password file: htpasswd -c /etc/nginx/.htpasswd user
}

# Deny user agents
if ($http_user_agent ~* bot|crawl|spider) {
    return 403;
}

# Rate limiting
limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;

server {
    location /api/ {
        limit_req zone=api burst=20 nodelay;
        proxy_pass http://backend_server;
    }
}
```

Caching

```
# Proxy cache
proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=my_cache:10m max_size=1g inactive=60m;

server {
    location / {
        proxy_cache my_cache;
        proxy_cache_key "$scheme$request_method$host$request_uri";
        proxy_cache_valid 200 60m;
        proxy_cache_valid 404 1m;
        proxy_cache_bypass $http_pragma $http_authorization;
        proxy_pass http://backend_server;
    }
}

# Static file caching
location ~* \.(jpg|jpeg|png|gif|ico|css|js|svg|woff|woff2)$ {
    expires 1y;
    add_header Cache-Control "public, immutable";
    access_log off;
}

# Browser cache
location ~* \.(html|htm)$ {
    expires 1h;
    add_header Cache-Control "public";
}
```

Security Headers

```
# Add security headers
add_header X-Frame-Options "SAMEORIGIN" always;
add_header X-Content-Type-Options "nosniff" always;
add_header X-XSS-Protection "1; mode=block" always;
add_header Referrer-Policy "no-referrer-when-downgrade" always;
add_header Content-Security-Policy "default-src 'self' http: https: data: blob: 'unsafe-inlin
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

# Hide Nginx version
server_tokens off;

# Limit request size
client_max_body_size 10M;

# Limit request rate
limit_req_zone $binary_remote_addr zone=one:10m rate=5r/s;

# Limit connections
limit_conn_zone $binary_remote_addr zone=addr:10m;
limit_conn addr 5;
```

SSL/TLS Configuration

```
# Strong SSL configuration
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256;
ssl_prefer_server_ciphers on;

# SSL session caching
ssl_session_cache shared:SSL:10m;
ssl_session_timeout 1d;
ssl_session_tickets off;

# OCSP stapling
ssl_stapling on;
ssl_stapling_verify on;
ssl_trusted_certificate /etc/letsencrypt/live/example.com/chain.pem;
resolver 8.8.8.8 8.8.4.4 valid=300s;
resolver_timeout 5s;
```

Logging

```
# Custom log format
log_format detailed '$remote_addr - $remote_user [$time_local] '
                    '"$request" $status $body_bytes_sent '
                    '"$http_referer" "$http_user_agent" '
                    '$request_time $upstream_response_time';

# Conditional logging
map $status $loggable {
    ~^[23]  0;
    default 1;
}

access_log /var/log/nginx/access.log detailed if=$loggable;

# Disable logging for health checks
location /health {
    access_log off;
    return 200 "OK";
}
```

Performance Tuning

```
# Worker connections
events {
    worker_connections 2048;
    use epoll;
    multi_accept on;
}

# Buffer sizes
client_body_buffer_size 128k;
client_max_body_size 10m;
client_header_buffer_size 1k;
large_client_header_buffers 4 4k;
output_buffers 1 32k;
postpone_output 1460;
```

```
# Timeouts
client_body_timeout 10;
client_header_timeout 10;
keepalive_timeout 65;
send_timeout 10;

# TCP optimization
tcp_nopush on;
tcp_nodelay on;
sendfile on;

# File descriptors
worker_rlimit_nofile 65535;
```

Best Practices

1. **Always test configuration** with `nginx -t` before reloading
2. **Use SSL/TLS** for all production services
3. **Enable HTTP/2** for better performance
4. **Cache aggressively** - Use proxy_cache for backend responses
5. **Rate limit** - Protect against DDoS attacks
6. **Security headers** - Add CSP, HSTS, X-Frame-Options
7. **Log rotation** - Keep logs manageable
8. **Monitor** - Use tools to track uptime and response times
9. **Backup configs** - Keep your Nginx configs in version control
10. **Use upstream blocks** - For load balancing and failover

Rate Limiting

```
# Define rate limit zone (http context)
# Syntax: limit_req_zone key zone=name:size rate=rate;
limit_req_zone $binary_remote_addr zone=api_limit:10m rate=10r/s;
limit_req_zone $binary_remote_addr zone=login_limit:10m rate=5r/m;
limit_req_zone $binary_remote_addr zone=general:10m rate=30r/s;

server {
    # Apply rate limit to a location
    location /api/ {
        limit_req zone=api_limit burst=20 nodelay;
        limit_req_status 429;
        proxy_pass http://backend;
    }

    # Strict rate limit for login endpoint
    location /api/auth/login {
        limit_req zone=login_limit burst=3 nodelay;
        limit_req_status 429;
        proxy_pass http://backend;
    }

    # Custom error page for rate-limited requests
    error_page 429 = @rate_limited;
```

```
    location @rate_limited {
        default_type application/json;
        return 429 '{"error":"Too many requests","retry_after":1}';
    }
}

# Per-URL rate limiting with map
map $uri $rate_limit_key {
    /api/search  $binary_remote_addr;
    /api/upload  $binary_remote_addr;
    default      "";
}

limit_req_zone $rate_limit_key zone=per_url:10m rate=5r/s;

server {
    location / {
        limit_req zone=per_url burst=10 nodelay;
        proxy_pass http://backend;
    }
}

# Whitelist trusted IPs from rate limiting
geo $limited {
    default         1;
    10.0.0.0/8      0;
    192.168.1.0/24  0;
}

map $limited $limit_key {
    0     "";
    1     $binary_remote_addr;
}

limit_req_zone $limit_key zone=whitelisted:10m rate=10r/s;
```

Upstream Configuration

```
# Weight-based distribution
upstream weighted_backend {
    server 10.0.0.1:3000 weight=5;   # 50% of traffic
    server 10.0.0.2:3000 weight=3;   # 30% of traffic
    server 10.0.0.3:3000 weight=2;   # 20% of traffic
}

# Backup servers — only used when all primaries are down
upstream with_backup {
    server 10.0.0.1:3000 weight=5 max_fails=3 fail_timeout=30s;
    server 10.0.0.2:3000 weight=3 max_fails=3 fail_timeout=30s;
    server 10.0.0.3:3000 backup;     # standby server
    server 10.0.0.4:3000 backup;
}

# max_fails and fail_timeout
# max_fails=3     — mark server as down after 3 consecutive failures
# fail_timeout=30s — consider server down for 30 seconds
# When fail_timeout expires, Nginx sends one probe request to check recovery
```

```
upstream resilient_backend {
    server 10.0.0.1:3000 max_fails=5 fail_timeout=60s;
    server 10.0.0.2:3000 max_fails=5 fail_timeout=60s;
    server 10.0.0.3:3000 max_fails=5 fail_timeout=60s;
}

# Keepalive connections to upstream (reduces TCP handshake overhead)
upstream keepalive_backend {
    server 10.0.0.1:3000;
    server 10.0.0.2:3000;

    keepalive 32;  # number of idle keepalive connections per worker
}

server {
    location / {
        proxy_pass http://keepalive_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";  # required for keepalive to work
        proxy_set_header Host $host;
    }
}

# slow_start — gradually ramp up traffic to a recovering server (commercial only)
upstream gradual_recovery {
    server 10.0.0.1:3000 slow_start=30s;
    server 10.0.0.2:3000 slow_start=30s;
}

# Health check directive (requires nginx-plus or nginx-plus-module-healthcheck)
# Passive health checks are built-in via max_fails/fail_timeout
# Active health checks require the health_check module:
server {
    location / {
        proxy_pass http://backend;
        health_check interval=5s fails=3 passes=2;
        # health_check match=status_ok;
    }
}
```

```
# Check upstream status via stub_status or separate status endpoint
# Also useful: nginx -T | grep upstream -A 10
```

WebSocket Proxy

```
# WebSocket reverse proxy — complete configuration
map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}

upstream websocket_backend {
    server 10.0.0.1:8080;
    server 10.0.0.2:8080;
}

server {
```

```
    listen 80;
    server_name ws.example.com;

    location /ws/ {
        proxy_pass http://websocket_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
        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;

        # WebSocket timeouts (critical — WS connections are long-lived)
        proxy_read_timeout 3600s;   # 1 hour
        proxy_send_timeout 3600s;

        # Buffering should be off for real-time data
        proxy_buffering off;
    }

    # Socket.io specific example
    location /socket.io/ {
        proxy_pass http://websocket_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_read_timeout 86400s;  # 24 hours
        proxy_send_timeout 86400s;
        proxy_buffering off;
    }
}
```

Streaming & Media

```
# MP4 pseudo-streaming (seek in video without downloading entire file)
# Requires --with-http_mp4_module
location /videos/ {
    mp4;
    mp4_buffer_size 1m;
    mp4_max_buffer_size 5m;
    root /var/www/media;
}

# FLV pseudo-streaming (Flash video)
# Requires --with-http_flv_module
location /flv/ {
    flv;
    root /var/www/media;
}

# Range requests — enabled by default in Nginx
# Supports Content-Range header for partial content (206)
server {
    listen 80;
    server_name media.example.com;
```

```
    # Serve large files with efficient range request handling
    location /downloads/ {
        root /var/www/media;
        max_ranges 1;        # allow one range per request
        sendfile on;
        tcp_nopush on;
        aio on;              # async I/O for large file transfers

        # Limit download speed per connection
        limit_rate 10m;
        limit_rate_after 50m;  # full speed for first 50MB, then throttle
    }
}

# Progressive JPEG and media caching
location ~* \.(mp4|webm|ogg|mp3|flv|avi|mov)$ {
    expires 30d;
    add_header Cache-Control "public, immutable";
    add_header Accept-Ranges bytes;
    access_log off;
}
```

Lua / OpenResty

```
# OpenResty extends Nginx with Lua — install via openresty package
# https://openresty.org

# Shared dictionary for cross-request data (defined in http block)
lua_shared_dict rate_limit_store 10m;
lua_shared_dict cache_store 32m;

# access_by_lua_block — runs during access phase
location /api/ {
    access_by_lua_block {
        local limit_store = ngx.shared.rate_limit_store
        local key = ngx.var.binary_remote_addr .. ":" .. ngx.var.uri
        local count, err = limit_store:incr(key, 1, 0, 60)

        if count and count > 100 then
            ngx.exit(429)
        end
    }
    proxy_pass http://backend;
}

# content_by_lua_block — generate response directly from Lua
location /lua/hello {
    content_by_lua_block {
        ngx.header["Content-Type"] = "application/json"
        ngx.say('{"message":"Hello from Lua","timestamp":' .. ngx.now() .. '}')
    }
}

# rewrite_by_lua_block — run Lua during rewrite phase
location /redirect-by-country {
    rewrite_by_lua_block {
        local country = ngx.var.geoip_country_code
        if country == "DE" then
```

```
            return ngx.redirect("https://de.example.com" .. ngx.var.request_uri)
        elseif country == "US" then
            return ngx.redirect("https://us.example.com" .. ngx.var.request_uri)
        end
    }
}

# Simple in-memory cache with Lua
location /cached-api/ {
    content_by_lua_block {
        local cache = ngx.shared.cache_store
        local key = ngx.var.request_uri
        local data = cache:get(key)

        if data then
            ngx.header["X-Cache"] = "HIT"
            ngx.say(data)
            return
        end

        -- Fetch from upstream (simplified example)
        local http = require "resty.http"
        local httpc = http.new()
        local res, err = httpc:request_uri("http://backend:3000" .. key)

        if res then
            cache:set(key, res.body, 30)  -- cache for 30 seconds
            ngx.header["X-Cache"] = "MISS"
            ngx.say(res.body)
        else
            ngx.status = 502
            ngx.say("Bad Gateway")
        end
    }
}

# Lua init_worker — runs once per worker on startup
init_worker_by_lua_block {
    ngx.log(ngx.NOTICE, "Worker " .. ngx.worker.pid() .. " started")
}
```

Microcaching

```
# Microcaching — cache dynamic content for very short periods (1-5 seconds)
# Dramatically reduces backend load for high-traffic sites

# Define a microcache zone (http context)
proxy_cache_path /var/cache/nginx/micro
    levels=1:2
    keys_zone=microcache:100m
    max_size=500m
    inactive=10s;     # remove unused items after 10s

server {
    # Cache everything for 1 second — enough for most high-traffic APIs
    location / {
        proxy_cache microcache;
        proxy_cache_key "$scheme$request_method$host$request_uri";
```

```
        proxy_cache_valid 200 1s;
        proxy_cache_valid 301 302 1m;
        proxy_cache_valid 404 1s;
        proxy_cache_use_stale updating;
        proxy_cache_background_update on;
        proxy_cache_lock on;         # only one request populates cache

        add_header X-Micro-Cache $upstream_cache_status;

        proxy_pass http://backend;
    }

    # Longer microcache for heavy endpoints (e.g., product pages)
    location /products/ {
        proxy_cache microcache;
        proxy_cache_key "$scheme$request_method$host$request_uri";
        proxy_cache_valid 200 5s;
        proxy_cache_use_stale error timeout updating http_500 http_502 http_503 http_504;
        proxy_cache_background_update on;
        proxy_cache_lock on;

        add_header X-Micro-Cache $upstream_cache_status;

        proxy_pass http://backend;
    }
}

# stale-while-revalidate and stale-if-error with proxy_cache
proxy_cache_path /var/cache/nginx/stale levels=1:2
    keys_zone=stale_cache:50m max_size=1g;

server {
    location / {
        proxy_cache stale_cache;
        proxy_cache_valid 200 60s;

        # Serve stale while revalidating in the background
        proxy_cache_use_stale updating;
        proxy_cache_background_update on;

        # Serve stale on backend errors
        proxy_cache_use_stale error timeout http_500 http_502 http_503 http_504;

        add_header X-Cache-Status $upstream_cache_status;
        proxy_pass http://backend;
    }
}
```

GeoIP & Geo Module

```
# GeoIP2 module (requires libmaxminddb and nginx module)
# Install: apt install libmaxminddb-dev && build with --with-http_geoip2_module

# Load GeoIP database
geoip2 /usr/share/GeoIP/GeoLite2-Country.mmdb {
    $geoip_country_code source=$remote_addr country iso_code;
    $geoip_country_name source=$remote_addr country names en;
}
```

```
geoip2 /usr/share/GeoIP/GeoLite2-City.mmdb {
    $geoip_city_name source=$remote_addr city names en;
    $geoip_latitude  source=$remote_addr location latitude;
    $geoip_longitude source=$remote_addr location longitude;
}

# Country-based blocking
map $geoip_country_code $blocked_country {
    default       0;
    XX            1;   # block unknown
    "KP"          1;   # block North Korea
}

server {
    if ($blocked_country) {
        return 403;
    }

    # Geo-based routing
    location / {
        proxy_pass http://backend;
    }
}

# Geo directive for simple IP-to-variable mapping
geo $trusted_proxy {
    default         0;
    10.0.0.0/8      1;
    172.16.0.0/12   1;
    192.168.0.0/16  1;
    203.0.113.0/24  2;  # CDN/proxy tier
}

# Set real IP based on geo
set_real_ip_from 10.0.0.0/8;
set_real_ip_from 172.16.0.0/12;
real_ip_header X-Forwarded-For;
real_ip_recursive on;

# Geo with map — redirect users by country
map $geoip_country_code $redirect_host {
    default       "www.example.com";
    "DE"          "de.example.com";
    "FR"          "fr.example.com";
    "JP"          "jp.example.com";
}

server {
    if ($redirect_host != "www.example.com") {
        return 302 https://$redirect_host$request_uri;
    }
}
```

Advanced Rewrites

```
# Regex capture groups in location and rewrite
location ~ ^/user/(\d+)/(posts|comments)$ {
```

```
    # $1 = user id, $2 = "posts" or "comments"
    rewrite ^ /api/v2/users/$1/$2 last;
}

# rewrite vs return — know the difference
# return: stops processing, sends response immediately (faster)
# rewrite: re-evaluates location matching (can loop with [last])

# return — use for simple redirects
location /old-page {
    return 301 /new-page;
}

location /legacy/api {
    return 308 /api/v2;  # 308 preserves request method (POST stays POST)
}

# rewrite with flags
# permanent (301) — permanent redirect, browser caches
# redirect  (302) — temporary redirect
# last      — stop rewriting, re-search location blocks
# break     — stop rewriting, stay in current location

server {
    # Permanent redirect with regex
    rewrite ^/category/(.*)$ /products/$1 permanent;

    # Remove trailing slash (SEO)
    rewrite ^(.*)/$ $1 permanent;

    # Remove www prefix
    if ($host ~* ^www\.(.+)$) {
        return 301 $scheme://$1$request_uri;
    }

    # Add trailing slash for directories
    rewrite ^([^.]*[^/])$ $1/ permanent;

    # Lowercase all URLs (Lua is cleaner for this, but here's pure nginx)
    # Not easily done without Lua; use if with regex as workaround:
    if ($request_uri ~ [A-Z]) {
        # Requires Lua in practice — see OpenResty section
    }

    # Rewrite with conditions using set + if
    set $maintenance 0;

    if (-f /var/www/maintenance.flag) {
        set $maintenance 1;
    }

    if ($maintenance) {
        return 503;
    }
}

# map-based rewrites — cleaner than if blocks
map $request_uri $new_uri {
    /about         /about-us;
```

```
    /contact       /get-in-touch;
    /pricing       /plans;
    /blog          /articles;
    default        "";
}

server {
    if ($new_uri != "") {
        return 301 $new_uri;
    }
}
```

Connection Tuning

```
# Core connection settings — tune based on expected traffic

# Worker processes — typically match CPU cores
worker_processes auto;              # or set to number of CPU cores
worker_cpu_affinity auto;           # pin workers to CPU cores

# Worker connections — total concurrent connections = workers × connections
events {
    worker_connections 4096;         # increase for high traffic (default 512)
    use epoll;                       # Linux optimized event model
    multi_accept on;                 # accept all new connections at once
    accept_mutex off;                # set to 'off' on multi-core for lower latency
                                    # set to 'on' if workers unevenly loaded
}

# Increase file descriptor limit (must match OS ulimit)
worker_rlimit_nofile 65535;

# HTTP-level tuning
http {
    # Keepalive settings
    keepalive_timeout 65;            # default 75s; lower for high-traffic APIs
    keepalive_requests 1000;         # max requests per keepalive connection
    keepalive_disable msie6;         # disable for old browsers

    # Client connection timeouts
    client_header_timeout 10;        # time to receive client request header
    client_body_timeout 10;          # time to receive client request body
    send_timeout 10;                 # time between two successive write operations
    reset_timedout_connection on;    # reset connection on timeout (free resources)

    # Connection limits
    limit_conn_zone $binary_remote_addr zone=conn_limit:10m;

    server {
        limit_conn conn_limit 50;    # max 50 concurrent connections per IP
    }

    # Reduce TIME_WAIT sockets
    # (also tune OS: sysctl -w net.ipv4.tcp_tw_reuse=1)
}
```

```
# Check current connection stats
curl -s http://localhost/nginx_status
# Active connections: 15
# server accepts handled requests
#  8456 8456 32891
# Reading: 0 Writing: 3 Waiting: 12

# Enable stub_status in Nginx config:
# location /nginx_status {
#     stub_status;
#     allow 127.0.0.1;
#     deny all;
# }
```

Gzip & Brotli

```
# Gzip compression — built into Nginx
gzip on;
gzip_vary on;                          # add Vary: Accept-Encoding header
gzip_proxied any;                      # compress proxied responses too
gzip_comp_level 4;                     # 1-9, 4-6 is best balance (default 1)
gzip_min_length 256;                   # don't compress tiny responses
gzip_buffers 16 8k;

# Compress these MIME types
gzip_types
    text/plain
    text/css
    text/xml
    text/javascript
    application/json
    application/javascript
    application/xml
    application/rss+xml
    application/atom+xml
    image/svg+xml
    font/opentype
    font/ttf
    font/woff
    font/woff2;

# Disable compression for already-compressed formats
gzip_types ~* ^image/(?!svg);
# Alternative: exclude specific types
location ~* \.(gif|png|jpg|jpeg|mp4|webp|zip|gz)$ {
    gzip off;
}

# Brotli compression — requires ngx_brotli module
# Build: --add-module=/path/ngx_brotli
brotli on;
brotli_comp_level 4;                   # 1-11, 4-6 recommended
brotli_types text/plain text/css application/json application/javascript
             application/xml text/xml image/svg+xml;
brotli_min_length 256;

# Serve pre-compressed Brotli files if available
```

```
# Place .br files alongside originals on disk
location /assets/ {
    brotli_static on;                  # serve .br file if it exists and client accepts br
    gzip_static on;                    # serve .gz file if it exists and client accepts gzip
    expires 1y;
    add_header Cache-Control "public, immutable";
}
```

```
# Pre-compress static assets at build time
for file in $(find /var/www/assets -name '*.js' -o -name '*.css' -o -name '*.svg'); do
    gzip -k -9 "$file"    # creates file.gz
    brotli "$file"        # creates file.br
done
```

IPv6 Configuration

```
# Dual-stack listening (IPv4 + IPv6)
server {
    listen 80;
    listen [::]:80;                    # IPv6 (dual-stack)

    server_name example.com;

    # Also listen on IPv6 with SSL
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
}

# IPv6-only server block
server {
    listen [::]:80 ipv6only=on;
    listen [::]:443 ssl http2 ipv6only=on;

    server_name ipv6.example.com;

    ssl_certificate /etc/ssl/ipv6.example.com/fullchain.pem;
    ssl_certificate_key /etc/ssl/ipv6.example.com/privkey.pem;

    root /var/www/ipv6.example.com;
}

# Proxy to IPv6 upstream
upstream ipv6_backend {
    server [2001:db8::1]:3000;
    server [2001:db8::2]:3000;
}

server {
    listen [::]:80;
    location / {
        proxy_pass http://ipv6_backend;
    }
}

# IPv6 CIDR in allow/deny rules
location /internal {
    allow 2001:db8:abcd::/48;
```

```
    deny all;
}

# Listen on specific IPv6 address
server {
    listen [2001:db8::10]:80;
    server_name ipv6-specific.example.com;
}

# Resolve upstream names to IPv6 (requires resolver)
resolver 2001:4860:4860::8888 valid=300s;  # Google DNS IPv6
```

Nginx as Load Balancer Deep Dive

```
# Round-robin (default) — distributes requests sequentially
upstream rr_backend {
    server 10.0.0.1:3000;
    server 10.0.0.2:3000;
    server 10.0.0.3:3000;
}

# Least connections — sends to server with fewest active connections
# Best when requests vary significantly in processing time
upstream lc_backend {
    least_conn;
    server 10.0.0.1:3000;
    server 10.0.0.2:3000;
    server 10.0.0.3:3000 weight=2;    # weight matters for least_conn too
}

# IP hash — sticky sessions based on client IP
# Same IP always goes to same server
upstream iphash_backend {
    ip_hash;
    server 10.0.0.1:3000;
    server 10.0.0.2:3000;
    server 10.0.0.3:3000 down;        # mark as permanently unavailable
}

# Generic hash — sticky sessions based on any variable
upstream hash_backend {
    hash $request_uri consistent;       # consistent hashing minimizes re-mapping
    server 10.0.0.1:3000;
    server 10.0.0.2:3000;
    server 10.0.0.3:3000;
}

# Hash by cookie for session affinity without IP dependency
upstream cookie_backend {
    hash $cookie_session_id consistent;
    server 10.0.0.1:3000;
    server 10.0.0.2:3000;
    server 10.0.0.3:3000;
}

# Random load balancing (Nginx 1.15.1+)
upstream random_backend {
    random;                           # pure random
```

```
    random two least_conn;            # pick 2 random, then least_conn among them
    server 10.0.0.1:3000;
    server 10.0.0.2:3000;
    server 10.0.0.3:3000;
}

# Health check module (Nginx Plus / nginx-plus-module-healthcheck)
server {
    location / {
        proxy_pass http://backend;
        health_check interval=5s rises=2 falls=3 timeout=2s;
        health_check match=match_backend;

        # Optional: send health check to a specific URI
        health_check uri=/health port=3000;
    }
}

# Shared status of upstreams (Nginx Plus)
server {
    location /upstream_status {
        upstream_conf;
        allow 127.0.0.1;
        deny all;
    }
}
```

Debugging

```
# Error log levels: debug, info, notice, warn, error, crit, alert, emerg
# Default is error — increase verbosity for troubleshooting

# Set debug level per location (performance impact — use sparingly)
error_log /var/log/nginx/error.log warn;

server {
    # Debug only for specific IP
    # events { debug_connection 10.0.0.5; }  ← goes in events block

    location /api/ {
        error_log /var/log/nginx/api_debug.log debug;
        proxy_pass http://backend;
    }
}

# Enable debug_connection for a specific client IP
events {
    debug_connection 10.0.0.50;
    debug_connection 192.168.1.0/24;
}
```

```
# Dump full configuration (expanded includes)
sudo nginx -T

# Test configuration without reloading
sudo nginx -t
```

```
# Check which config files are loaded
sudo nginx -T 2>/dev/null | grep "configuration file"

# View error log in real-time
sudo tail -f /var/log/nginx/error.log

# View only errors and above
sudo tail -f /var/log/nginx/error.log | grep -E "\[error\]|\[crit\]|\[alert\]|\[emerg\]"

# Check open connections
ss -s

# Check Nginx worker processes
ps aux | grep nginx

# Verify Nginx is listening on expected ports
sudo nginx -T | grep "listen"

# Reload if config is valid
sudo nginx -t && sudo systemctl reload nginx

# Stub status endpoint
curl -s http://localhost/nginx_status
# Active connections: 291
# server accepts handled requests
#  16630948 16630948 31070465
# Reading: 6 Writing: 179 Waiting: 106

# Test upstream connectivity
curl -v -H "Host: app.example.com" http://localhost/
```

Next Steps

- Systemd Cheat Sheet — Service management
- Docker CLI Cheat Sheet — Containerize Nginx
- Terraform Cheat Sheet — Infrastructure as code
- SSL/TLS Security Cheat Sheet — Certificate management

High Performance Web Server & Reverse Proxy