Nginx Cheatsheet

By Dejan Panovski • Updated on Sep 16, 2026 •

Nginx directives at a glance: server blocks, location matching, reverse proxy headers, TLS, redirects, rate limits, caching, and log formats.

Nginx configuration is mostly a small set of directives used in the right context. This cheatsheet covers service commands, configuration layout, server blocks, location matching, root versus alias, reverse proxy and upstream settings, TLS, redirects, limits, caching, logging, and the variables you use in log formats and proxy headers.

Service and CLI Commands

Control the service and check the configuration before it goes live.

`sudo systemctl reload nginx`	Apply a new configuration without dropping connections
`sudo systemctl start nginx`	Start the service
`sudo systemctl stop nginx`	Stop the service immediately
`sudo systemctl restart nginx`	Stop and start the service
`sudo systemctl status nginx`	Show the service state and recent log lines
`sudo systemctl enable --now nginx`	Start now and at every boot
`sudo nginx -t`	Test the configuration for syntax errors
`sudo nginx -T`	Test and print the full merged configuration
`sudo nginx -s reload`	Reload through the master process signal
`sudo nginx -s quit`	Shut down gracefully after current requests finish
`sudo nginx -s reopen`	Reopen the log files after rotation
`nginx -V`	Print the version and the configure arguments

Always run `nginx -t` before a reload. A failed reload leaves the old configuration running, but a restart with a broken file leaves the service down.

Configuration Layout

Where the files live and how a site is switched on.

`/etc/nginx/nginx.conf`	Main configuration file
`/etc/nginx/conf.d/*.conf`	Common include pattern; verify it in `nginx.conf`
`/etc/nginx/sites-available/`	Site definitions on Ubuntu and Debian
`/etc/nginx/sites-enabled/`	Symlinks to the active sites
`/etc/nginx/snippets/`	Reusable fragments on Ubuntu and Debian
`/var/www/html`	Default document root on Ubuntu and Debian
`/usr/share/nginx/html`	Default document root on Fedora and RHEL
`/var/log/nginx/`	Access and error logs
`sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/`	Enable a site on Ubuntu and Debian
`sudo unlink /etc/nginx/sites-enabled/default`	Disable the default site

Fedora and RHEL have no `sites-available` directory. Put each site in its own file under `/etc/nginx/conf.d/` instead.

Server Blocks

Directives that decide which server block answers a request.

`listen 80;`	Listen on IPv4 port 80
`listen [::]:80;`	Listen on IPv6 port 80
`listen 80 default_server;`	Serve requests matching no other server name
`server_name example.com www.example.com;`	Match these host names
`server_name *.example.com;`	Match any subdomain
`server_name _;`	Invalid name used as a catch-all placeholder
`root /var/www/example.com;`	Set the document root
`index index.html index.htm;`	File served when a directory is requested
`include /etc/nginx/snippets/ssl.conf;`	Pull in a shared fragment

Nginx matches the exact name first, then the longest wildcard starting with an asterisk, then the longest wildcard ending with one, and finally the first matching regular expression.

Location Matching

Modifiers that set both the match rule and its priority.

`location = /health { ... }`	Exact match, checked first and wins immediately
`location ^~ /static/ { ... }`	Prefix match that suppresses the regex pass
`location ~ \.php$ { ... }`	Case-sensitive regular expression
`location ~* \.css$ { ... }`	Case-insensitive regular expression
`location /images/ { ... }`	Plain prefix match
`location / { ... }`	Fallback for every request
`location @fallback { ... }`	Named location, reachable only from `error_page` or `try_files`

Nginx checks the exact match first, then stores the longest matching prefix. If that prefix uses `^~`, it is used right away. Otherwise the regular expressions are tried in file order and the first match wins. The stored prefix is used only when no regular expression matches.

Serving Files

Map a request to a file on disk and decide what happens when it is missing.

`root /var/www/example.com;`	Append the current normalized URI path, without the query string
`alias /srv/media/;`	Replace the matched location prefix with this path
`try_files $uri $uri/ =404;`	Try the file, then the directory, then return 404
`try_files $uri $uri/ /index.html;`	Single-page application fallback
`try_files $uri $uri/ /index.php?$query_string;`	WordPress and PHP framework fallback
`autoindex on;`	Generate a directory listing
`error_page 404 /404.html;`	Serve a custom error page
`error_page 502 503 504 /5xx.html;`	One page for several statuses
`sendfile on;`	Copy files to the socket in the kernel

With `root`, nginx appends the current normalized URI path without the query string, so `location /images/` with `root /data` serves `/data/images/cat.png`. With `alias`, the matched prefix is replaced instead, so the same location with `alias /data/pictures/` serves `/data/pictures/cat.png`. Keep the trailing slash on both the location and the alias.

Reverse Proxy

Forward requests to an application and pass on the client details.

`proxy_pass http://127.0.0.1:3000;`	Forward to a local application
`proxy_pass http://backend;`	Forward to a named upstream group
`proxy_set_header Host $host;`	Pass the original host name
`proxy_set_header X-Real-IP $remote_addr;`	Pass the client address
`proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;`	Append the client to the forwarding chain
`proxy_set_header X-Forwarded-Proto $scheme;`	Tell the application whether TLS was used
`proxy_http_version 1.1;`	Use HTTP/1.1 for upstream keepalive and WebSocket upgrades
`proxy_set_header Upgrade $http_upgrade;`	Pass the WebSocket upgrade request
`proxy_set_header Connection "upgrade";`	Complete the WebSocket handshake
`proxy_read_timeout 300s;`	Wait longer for a slow response
`proxy_buffering off;`	Stream the response as it arrives

The trailing slash changes the result. `proxy_pass http://127.0.0.1:3000;` forwards the full request URI, while `proxy_pass http://127.0.0.1:3000/;` replaces the matched location prefix with `/`.

Load Balancing

Spread traffic across a pool of backends. Define the pool in `http` and put the other directives inside `upstream`.

`upstream backend { server 10.0.0.1:8080; server 10.0.0.2:8080; }`	Define a pool, round-robin by default
`least_conn;`	Send each request to the least busy server
`ip_hash;`	Pin a client address to one server
`hash $request_uri consistent;`	Distribute by key with minimal reshuffling
`server 10.0.0.1:8080 weight=3;`	Take three times the usual share
`server 10.0.0.2:8080 max_fails=3 fail_timeout=30s;`	After three qualifying failures within 30 seconds, mark unavailable for 30 seconds
`server 10.0.0.3:8080 backup;`	Use only when the others are down
`server 10.0.0.4:8080 down;`	Take a server out of rotation
`server unix:/run/app.sock;`	Proxy to a Unix socket
`keepalive 32;`	Cache up to 32 idle upstream connections per worker

Choose only one of `least_conn`, `ip_hash`, or `hash`; round-robin is the default. The `backup` parameter cannot be combined with `hash` or `ip_hash`. On nginx versions older than 1.29.7, set `proxy_http_version 1.1;` and `proxy_set_header Connection "";` in the proxy location to use HTTP/1.1 upstream keepalive. Since 1.29.7, HTTP/1.1 and upstream keepalive are enabled by default, and the default proxy configuration no longer sends `Connection: close`.

HTTPS and TLS

Terminate TLS and keep the protocol settings current.

`listen 443 ssl;`	Accept TLS connections
`http2 on;`	Enable HTTP/2 on nginx 1.25.1 and later
`listen 443 ssl http2;`	Enable HTTP/2 on older releases
`ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;`	Certificate and intermediate chain
`ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;`	Private key
`ssl_protocols TLSv1.2 TLSv1.3;`	Allow only modern protocol versions
`ssl_prefer_server_ciphers off;`	Let the client pick from the allowed ciphers
`ssl_session_cache shared:SSL:10m;`	Share the session cache between workers
`ssl_session_timeout 1d;`	Keep sessions resumable for a day
`add_header Strict-Transport-Security "max-age=63072000" always;`	Send HSTS on every response
`sudo certbot --nginx -d example.com -d www.example.com`	Issue and install a certificate

Keep the private key readable by root only. Never copy a key into a repository or a document root, and add `*.pem` to `.gitignore` when the configuration lives in version control.

Redirects and Rewrites

Move URLs without losing the original path.

`return 301 https://$host$request_uri;`	Redirect every request to HTTPS
`return 301 https://www.example.com $request_uri;`	Redirect to the www host
`return 302 /maintenance.html;`	Temporary redirect
`return 444;`	Close the connection without a response
`rewrite ^/old/(.*)$ /new/$1 permanent;`	301 rewrite that keeps the path tail
`rewrite ^/old/(.*)$ /new/$1 redirect;`	The same rewrite as a 302
`rewrite ^/blog/(.*)$ /$1 last;`	Rewrite internally and restart location matching
`rewrite ^/api/(.*)$ /$1 break;`	Rewrite internally and stop processing rewrites

Prefer `return` over `rewrite` for plain redirects. It is faster, easier to read, and it skips the regular expression evaluation that `rewrite` performs on every request.

Access Control and Limits

Guard the upstream against oversized uploads and traffic spikes. Use `htpasswd -c` only for a new password file, because it overwrites an existing one.

`client_max_body_size 64m;`	Raise the upload size limit
`allow 10.0.0.0/8;`	Permit a network
`deny all;`	Block everything else
`auth_basic "Restricted";`	Turn on HTTP basic authentication
`auth_basic_user_file /etc/nginx/.htpasswd;`	Point to the password file
`sudo htpasswd -c /etc/nginx/.htpasswd admin`	Create a new password file, first creation only
`sudo htpasswd /etc/nginx/.htpasswd editor`	Add or update a user in the existing file
`limit_req_zone $binary_remote_addr zone=req:10m rate=10r/s;`	Define a rate limit zone in `http`
`limit_req zone=req burst=20 nodelay;`	Apply the zone with a burst allowance
`limit_conn_zone $binary_remote_addr zone=conn:10m;`	Define a connection limit zone in `http`
`limit_conn conn 10;`	Limit active connections to ten per address
`server_tokens off;`	Hide the version number in responses and error pages

The `.htpasswd` file holds hashed credentials. Keep it outside the document root and out of version control. In HTTP/2 and HTTP/3, `limit_conn` counts each concurrent request as a separate connection.

Compression and Caching

Cut response size and avoid repeat trips to the backend.

`gzip on;`	Compress responses
`gzip_types text/css application/javascript application/json;`	Compress these types beyond `text/html`
`gzip_min_length 256;`	Skip responses too small to benefit
`gzip_comp_level 5;`	Balance CPU time against size
`gzip_vary on;`	Add `Vary: Accept-Encoding`
`expires 30d;`	Set a far-future expiry inside a static location
`add_header Cache-Control "public, immutable";`	Mark fingerprinted assets as cacheable
`proxy_cache_path /var/cache/nginx keys_zone=cache:10m max_size=1g;`	Define a proxy cache in `http`
`proxy_cache cache;`	Turn the cache on for a location
`proxy_cache_valid 200 10m;`	Cache successful responses for ten minutes
`add_header X-Cache-Status $upstream_cache_status;`	Expose cache hits and misses while debugging

By default, a block with its own `add_header` directives does not inherit its parent's `add_header` directives. Repeat the ones you still need, or use `add_header_inherit merge;` on nginx 1.29.3 and later.

Logging

Choose what gets recorded and where to watch it.

`access_log /var/log/nginx/access.log combined;`	Write access logs in the default format
`access_log off;`	Disable access logging for a location
`error_log /var/log/nginx/error.log warn;`	Set the error log file and level
`log_format main '$remote_addr $status "$request" $request_time';`	Define a custom format in `http`
`access_log /var/log/nginx/api.log main;`	Use that custom format
`sudo tail -f /var/log/nginx/error.log`	Follow errors live
`sudo journalctl -u nginx -f`	Follow service-level messages
`sudo nginx -s reopen`	Reopen log files after rotation

Error log levels run `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert`, and `emerg`, from most to least verbose, and each level includes everything more severe. The `debug` level needs a build configured with `--with-debug`.

Common Variables

Values available in log formats, proxy headers, redirects, and conditions.

`$host`	Host from the request line, then the Host header, then the matching server name
`$remote_addr`	Client IP address
`$request_uri`	Full original URI including the query string
`$uri`	Current URI after rewrites, without the query string
`$args`	Query string
`$scheme`	`http` or `https`
`$request_method`	GET, POST, and so on
`$status`	Response status code
`$body_bytes_sent`	Size of the response body
`$request_time`	Request duration in seconds
`$http_user_agent`	User-Agent header
`$proxy_add_x_forwarded_for`	Existing X-Forwarded-For plus the client address
`$upstream_addr`	Backend that served the request
`$upstream_response_time`	Backend response time
`$document_root`	Document root for the current request

Related Guides

Use these guides for the longer explanations behind these directives.

Nginx Commands You Should Know	Service, testing, and reload commands in detail
Nginx Location Blocks	Match rules and the full priority order
Nginx Reverse Proxy	proxy_pass, headers, and WebSocket support
Configuring the Nginx Error and Access Logs	Log formats, levels, and rotation
Redirect HTTP to HTTPS in Nginx	Redirect patterns and the pitfalls to avoid
How to Start, Stop, or Restart Nginx	Reload versus restart and what each one does
Nginx Server Blocks on Ubuntu	Hosting several sites on one server
How to Install Nginx on Ubuntu 26.04	Installation, firewall rules, and first steps