



[Nginx, Inc.](#)

# NGINX Plus Reference Guide

NGINX Plus - release 28, based on 1.23.2 core

November 24, 2022

## Copyright Notice

© 2012-2022 Nginx, Inc. All rights reserved. NGINX, NGINX Plus and any Nginx, Inc. product or service name or logo used herein are trademarks of Nginx, Inc. All other trademarks used herein belong to their respective owners. The trademarks and logos displayed herein may not be used without the prior written consent of Nginx, Inc. or their respective owners.

This documentation is provided “AS IS” and is subject to change without notice and should not be interpreted as a commitment by Nginx, Inc. This documentation may not be copied, modified or distributed without authorization of Nginx, Inc. and may be used only in connection with Nginx, Inc. products and services. Nginx, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this documentation.

# Preface

## About NGINX

NGINX<sup>®</sup> (“engine x”) is a high performance, high concurrency web server excelling at large scale content delivery, web acceleration and protecting application containers. Its precise integration with modern operating systems allows unprecedented levels of efficiency even when running on commodity hardware.

Nginx, Inc. develops and maintains NGINX open source distribution, and offers commercial support and professional services for NGINX.

## About NGINX Plus

- Offers [additional features](#) on top of the free open source NGINX version.
- Prepared, tested and supported by NGINX core engineering team led by the original author Igor Sysoev.

## For more information

- Find more details about NGINX products and support at <https://www.nginx.com/>.
- For online NGINX documentation visit <https://nginx.org/en/docs>.
- NGINX and NGINX Plus Tutorial and Admin Guide is available here: <https://www.nginx.com/resources/admin-guide/>.
- For general inquiries, please use: [nginx-inquiries@nginx.com](mailto:nginx-inquiries@nginx.com)

# Contents

<b>Title</b>	<b>1</b>
<b>Preface</b>	<b>2</b>
<b>Table of Contents</b>	<b>3</b>
<b>1 Core modules</b>	<b>6</b>
1.1 Core functionality . . . . .	6
1.2 Setting up hashes . . . . .	16
1.3 Connection processing methods . . . . .	17
1.4 Logging to syslog . . . . .	18
<b>2 HTTP server modules</b>	<b>19</b>
2.1 Module ngx_http_core_module . . . . .	19
2.2 Module ngx_http_access_module . . . . .	59
2.3 Module ngx_http_addition_module . . . . .	61
2.4 Module ngx_http_api_module . . . . .	63
2.5 Module ngx_http_auth_basic_module . . . . .	114
2.6 Module ngx_http_auth_jwt_module . . . . .	116
2.7 Module ngx_http_auth_request_module . . . . .	121
2.8 Module ngx_http_autoindex_module . . . . .	123
2.9 Module ngx_http_browser_module . . . . .	125
2.10 Module ngx_http_charset_module . . . . .	127
2.11 Module ngx_http_dav_module . . . . .	130
2.12 Module ngx_http_empty_gif_module . . . . .	133
2.13 Module ngx_http_f4f_module . . . . .	134
2.14 Module ngx_http_fastcgi_module . . . . .	135
2.15 Module ngx_http_flv_module . . . . .	156
2.16 Module ngx_http_geo_module . . . . .	157
2.17 Module ngx_http_geoip_module . . . . .	160
2.18 Module ngx_http_grpc_module . . . . .	163
2.19 Module ngx_http_gunzip_module . . . . .	172
2.20 Module ngx_http_gzip_module . . . . .	173
2.21 Module ngx_http_gzip_static_module . . . . .	177
2.22 Module ngx_http_headers_module . . . . .	178
2.23 Module ngx_http_hls_module . . . . .	181
2.24 Module ngx_http_image_filter_module . . . . .	185

2.25	Module ngx_http_index_module . . . . .	189
2.26	Module ngx_http_js_module . . . . .	190
2.27	Module ngx_http_keyval_module . . . . .	198
2.28	Module ngx_http_limit_conn_module . . . . .	200
2.29	Module ngx_http_limit_req_module . . . . .	204
2.30	Module ngx_http_log_module . . . . .	208
2.31	Module ngx_http_map_module . . . . .	212
2.32	Module ngx_http_memcached_module . . . . .	215
2.33	Module ngx_http_mirror_module . . . . .	220
2.34	Module ngx_http_mp4_module . . . . .	222
2.35	Module ngx_http_perl_module . . . . .	225
2.36	Module ngx_http_proxy_module . . . . .	231
2.37	Module ngx_http_proxy_module . . . . .	260
2.38	Module ngx_http_random_index_module . . . . .	261
2.39	Module ngx_http_realip_module . . . . .	262
2.40	Module ngx_http_referer_module . . . . .	264
2.41	Module ngx_http_rewrite_module . . . . .	266
2.42	Module ngx_http_scgi_module . . . . .	272
2.43	Module ngx_http_secure_link_module . . . . .	290
2.44	Module ngx_http_session_log_module . . . . .	293
2.45	Module ngx_http_slice_module . . . . .	295
2.46	Module ngx_http_split_clients_module . . . . .	297
2.47	Module ngx_http_ssi_module . . . . .	298
2.48	Module ngx_http_ssl_module . . . . .	304
2.49	Module ngx_http_status_module . . . . .	318
2.50	Module ngx_http_stub_status_module . . . . .	328
2.51	Module ngx_http_sub_module . . . . .	330
2.52	Module ngx_http_upstream_module . . . . .	332
2.53	Module ngx_http_upstream_conf_module . . . . .	349
2.54	Module ngx_http_upstream_hc_module . . . . .	353
2.55	Module ngx_http_userid_module . . . . .	358
2.56	Module ngx_http_uwsgi_module . . . . .	362
2.57	Module ngx_http_v2_module . . . . .	384
2.58	Module ngx_http_xslt_module . . . . .	389
<b>3</b>	<b>Stream server modules</b>	<b>392</b>
3.1	Module ngx_stream_core_module . . . . .	392
3.2	Module ngx_stream_access_module . . . . .	401
3.3	Module ngx_stream_geo_module . . . . .	402
3.4	Module ngx_stream_geoip_module . . . . .	404
3.5	Module ngx_stream_js_module . . . . .	407
3.6	Module ngx_stream_keyval_module . . . . .	415
3.7	Module ngx_stream_limit_conn_module . . . . .	417
3.8	Module ngx_stream_log_module . . . . .	420
3.9	Module ngx_stream_map_module . . . . .	423
3.10	Module ngx_stream_proxy_module . . . . .	426

---

3.11	Module ngx_stream_p ... _module . . . . .	435
3.12	Module ngx_stream_realip_module . . . . .	436
3.13	Module ngx_stream_return_module . . . . .	437
3.14	Module ngx_stream_set_module . . . . .	438
3.15	Module ngx_stream_split_clients_module . . . . .	439
3.16	Module ngx_stream_ssl_module . . . . .	440
3.17	Module ngx_stream_ssl_preread_module . . . . .	450
3.18	Module ngx_stream_upstream_module . . . . .	452
3.19	Module ngx_stream_upstream_hc_module . . . . .	460
3.20	Module ngx_stream_zone_sync_module . . . . .	464
<b>4</b>	<b>Mail server modules</b>	<b>472</b>
4.1	Module ngx_mail_core_module . . . . .	472
4.2	Module ngx_mail_auth_http_module . . . . .	478
4.3	Module ngx_mail_proxy_module . . . . .	482
4.4	Module ngx_mail_realip_module . . . . .	484
4.5	Module ngx_mail_ssl_module . . . . .	485
4.6	Module ngx_mail_imap_module . . . . .	493
4.7	Module ngx_mail_pop3_module . . . . .	495
4.8	Module ngx_mail_smtp_module . . . . .	496
<b>5</b>	<b>Miscellaneous</b>	<b>498</b>
5.1	Command-line parameters . . . . .	498
<b>A</b>	<b>Changelog for NGINX Plus</b>	<b>500</b>
<b>B</b>	<b>Legal Notices</b>	<b>511</b>
	<b>Index</b>	<b>516</b>

# Chapter 1

## Core modules

### 1.1 Core functionality

1.1.1	Example Configuration . . . . .	7
1.1.2	Directives . . . . .	7
	accept_mutex . . . . .	7
	accept_mutex_delay . . . . .	7
	daemon . . . . .	7
	debug_connection . . . . .	8
	debug_points . . . . .	8
	env . . . . .	8
	error_log . . . . .	9
	events . . . . .	10
	include . . . . .	10
	load_module . . . . .	10
	lock_file . . . . .	10
	master_process . . . . .	11
	multi_accept . . . . .	11
	pcre_jit . . . . .	11
	pid . . . . .	11
	ssl_engine . . . . .	11
	thread_pool . . . . .	12
	timer_resolution . . . . .	12
	use . . . . .	12
	user . . . . .	13
	worker_aio_requests . . . . .	13
	worker_connections . . . . .	13
	worker_cpu_affinity . . . . .	13
	worker_priority . . . . .	14
	worker_processes . . . . .	14
	worker_rlimit_core . . . . .	15
	worker_rlimit_nofile . . . . .	15
	worker_shutdown_timeout . . . . .	15
	working_directory . . . . .	15

### 1.1.1 Example Configuration

```
user www www;
worker_processes 2;

error_log /var/log/nginx-error.log info;

events {
    use kqueue;
    worker_connections 2048;
}

...
```

### 1.1.2 Directives

#### **accept\_mutex**

SYNTAX: **accept\_mutex** on | off;  
DEFAULT off  
CONTEXT: events

If `accept_mutex` is enabled, worker processes will accept new connections by turn. Otherwise, all worker processes will be notified about new connections, and if volume of new connections is low, some of the worker processes may just waste system resources.

There is no need to enable `accept_mutex` on systems that support the [EPOLLEXCLUSIVE](#) flag (1.11.3) or when using [reuseport](#).

Prior to version 1.11.3, the default value was on.

#### **accept\_mutex\_delay**

SYNTAX: **accept\_mutex\_delay** *time*;  
DEFAULT 500ms  
CONTEXT: events

If [accept\\_mutex](#) is enabled, specifies the maximum time during which a worker process will try to restart accepting new connections if another worker process is currently accepting new connections.

#### **daemon**

SYNTAX: **daemon** on | off;  
DEFAULT on  
CONTEXT: main

Determines whether nginx should become a daemon. Mainly used during development.



## debug\_connection

SYNTAX: **debug\_connection** *address* | *CIDR* | *unix*::;

DEFAULT —

CONTEXT: events

Enables debugging log for selected client connections. Other connections will use logging level set by the [error\\_log](#) directive. Debugged connections are specified by IPv4 or IPv6 (1.3.0, 1.2.1) address or network. A connection may also be specified using a hostname. For connections using UNIX-domain sockets (1.3.0, 1.2.1), debugging log is enabled by the “unix:” parameter.

```
events {
    debug_connection 127.0.0.1;
    debug_connection localhost;
    debug_connection 192.0.2.0/24;
    debug_connection ::1;
    debug_connection 2001:0db8::/32;
    debug_connection unix::;
    ...
}
```

For this directive to work, nginx needs to be built with `--with-debug`, see “[A debugging log](#)”.

## debug\_points

SYNTAX: **debug\_points** *abort* | *stop*;

DEFAULT —

CONTEXT: main

This directive is used for debugging.

When internal error is detected, e.g. the leak of sockets on restart of working processes, enabling `debug_points` leads to a core file creation (*abort*) or to stopping of a process (*stop*) for further analysis using a system debugger.

## env

SYNTAX: **env** *variable*[=*value*];

DEFAULT TZ

CONTEXT: main

By default, nginx removes all environment variables inherited from its parent process except the TZ variable. This directive allows preserving some of the inherited variables, changing their values, or creating new environment variables. These variables are then:

- inherited during a [live upgrade](#) of an executable file;
- used by the [ngx\\_http\\_perl\\_module](#) module;

- used by worker processes. One should bear in mind that controlling system libraries in this way is not always possible as it is common for libraries to check variables only during initialization, well before they can be set using this directive. An exception from this is an above mentioned [live upgrade](#) of an executable file.

The TZ variable is always inherited and available to the [ngx\\_http\\_perl-module](#) module, unless it is configured explicitly.

Usage example:

```
env MALLOC_OPTIONS;  
env PERL5LIB=/data/site/modules;  
env OPENSSL_ALLOW_PROXY_CERTS=1;
```

The NGINX environment variable is used internally by nginx and should not be set directly by the user.

## error\_log

SYNTAX: **error\_log** *file* [*level*];

DEFAULT logs/error.log error

CONTEXT: main, http, mail, stream, server, location

Configures logging. Several logs can be specified on the same configuration level (1.5.2). If on the main configuration level writing a log to a file is not explicitly defined, the default file will be used.

The first parameter defines a *file* that will store the log.

The special value `stderr` selects the standard error file. Logging to [syslog](#) can be configured by specifying the “syslog:” prefix. Logging to a [cyclic memory buffer](#) can be configured by specifying the “memory:” prefix and buffer *size*, and is generally used for debugging (1.7.11).

The second parameter determines the *level* of logging, and can be one of the following: debug, info, notice, warn, error, crit, alert, or emerg. Log levels above are listed in the order of increasing severity. Setting a certain log level will cause all messages of the specified and more severe log levels to be logged. For example, the default level error will cause error, crit, alert, and emerg messages to be logged. If this parameter is omitted then error is used.

For debug logging to work, nginx needs to be built with `--with-debug`, see “[A debugging log](#)”.

The directive can be specified on the `stream` level starting from version 1.7.11, and on the `mail` level starting from version 1.9.0.

## events

SYNTAX: **events** { ... }

DEFAULT —

CONTEXT: main

Provides the configuration file context in which the directives that affect connection processing are specified.

## include

SYNTAX: **include** *file* | *mask*;

DEFAULT —

CONTEXT: any

Includes another *file*, or files matching the specified *mask*, into configuration. Included files should consist of syntactically correct directives and blocks.

Usage example:

```
include mime.types;  
include vhosts/*.conf;
```

## load\_module

SYNTAX: **load\_module** *file*;

DEFAULT —

CONTEXT: main

THIS DIRECTIVE APPEARED IN VERSION 1.9.11.

Loads a dynamic module.

Example:

```
load_module modules/nginx_mail_module.so;
```

## lock\_file

SYNTAX: **lock\_file** *file*;

DEFAULT logs/nginx.lock

CONTEXT: main

nginx uses the locking mechanism to implement [accept\\_mutex](#) and serialize access to shared memory. On most systems the locks are implemented using atomic operations, and this directive is ignored. On other systems the “lock file” mechanism is used. This directive specifies a prefix for the names of lock files.

## master\_process

SYNTAX: **master\_process** on | off;  
DEFAULT on  
CONTEXT: main

Determines whether worker processes are started. This directive is intended for nginx developers.

## multi\_accept

SYNTAX: **multi\_accept** on | off;  
DEFAULT off  
CONTEXT: events

If `multi_accept` is disabled, a worker process will accept one new connection at a time. Otherwise, a worker process will accept all new connections at a time.

The directive is ignored if `kqueue` connection processing method is used, because it reports the number of new connections waiting to be accepted.

## pcre\_jit

SYNTAX: **pcre\_jit** on | off;  
DEFAULT off  
CONTEXT: main  
THIS DIRECTIVE APPEARED IN VERSION 1.1.12.

Enables or disables the use of “just-in-time compilation” (PCRE JIT) for the regular expressions known by the time of configuration parsing.

PCRE JIT can speed up processing of regular expressions significantly.

The JIT is available in PCRE libraries starting from version 8.20 built with the `--enable-jit` configuration parameter. When the PCRE library is built with nginx (`--with-pcre=`), the JIT support is enabled via the `--with-pcre-jit` configuration parameter.

## pid

SYNTAX: **pid** *file*;  
DEFAULT `logs/nginx.pid`  
CONTEXT: main

Defines a *file* that will store the process ID of the main process.

## ssl\_engine

SYNTAX: **ssl\_engine** *device*;  
DEFAULT —  
CONTEXT: main

Defines the name of the hardware SSL accelerator.

## thread\_pool

SYNTAX: **thread\_pool** *name* threads=*number* [max\_queue=*number*];

DEFAULT default threads=32 max\_queue=65536

CONTEXT: main

THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Defines the *name* and parameters of a thread pool used for multi-threaded reading and sending of files [without blocking](#) worker processes.

The `threads` parameter defines the number of threads in the pool.

In the event that all threads in the pool are busy, a new task will wait in the queue. The `max_queue` parameter limits the number of tasks allowed to be waiting in the queue. By default, up to 65536 tasks can wait in the queue. When the queue overflows, the task is completed with an error.

## timer\_resolution

SYNTAX: **timer\_resolution** *interval*;

DEFAULT —

CONTEXT: main

Reduces timer resolution in worker processes, thus reducing the number of `gettimeofday` system calls made. By default, `gettimeofday` is called each time a kernel event is received. With reduced resolution, `gettimeofday` is only called once per specified *interval*.

Example:

```
timer_resolution 100ms;
```

Internal implementation of the interval depends on the method used:

- the `EVFILT_TIMER` filter if `kqueue` is used;
- `timer_create` if `eventport` is used;
- `setitimer` otherwise.

## use

SYNTAX: **use** *method*;

DEFAULT —

CONTEXT: events

Specifies the [connection processing](#) *method* to use. There is normally no need to specify it explicitly, because nginx will by default use the most efficient method.

**user**

SYNTAX: **user** *user* [*group*];

DEFAULT nobody nobody

CONTEXT: main

Defines *user* and *group* credentials used by worker processes. If *group* is omitted, a group whose name equals that of *user* is used.

**worker\_aio\_requests**

SYNTAX: **worker\_aio\_requests** *number*;

DEFAULT 32

CONTEXT: events

THIS DIRECTIVE APPEARED IN VERSIONS 1.1.4 AND 1.0.7.

When using [aio](#) with the [epoll](#) connection processing method, sets the maximum *number* of outstanding asynchronous I/O operations for a single worker process.

**worker\_connections**

SYNTAX: **worker\_connections** *number*;

DEFAULT 512

CONTEXT: events

Sets the maximum number of simultaneous connections that can be opened by a worker process.

It should be kept in mind that this number includes all connections (e.g. connections with proxied servers, among others), not only connections with clients. Another consideration is that the actual number of simultaneous connections cannot exceed the current limit on the maximum number of open files, which can be changed by [worker\\_rlimit\\_nofile](#).

**worker\_cpu\_affinity**

SYNTAX: **worker\_cpu\_affinity** *cpumask* ...;

SYNTAX: **worker\_cpu\_affinity** auto [*cpumask*];

DEFAULT —

CONTEXT: main

Binds worker processes to the sets of CPUs. Each CPU set is represented by a bitmask of allowed CPUs. There should be a separate set defined for each of the worker processes. By default, worker processes are not bound to any specific CPUs.

For example,

```
worker_processes      4;
worker_cpu_affinity 0001 0010 0100 1000;
```

binds each worker process to a separate CPU, while

```
worker_processes    2;  
worker_cpu_affinity 0101 1010;
```

binds the first worker process to CPU0/CPU2, and the second worker process to CPU1/CPU3. The second example is suitable for hyper-threading.

The special value `auto` (1.9.10) allows binding worker processes automatically to available CPUs:

```
worker_processes auto;  
worker_cpu_affinity auto;
```

The optional mask parameter can be used to limit the CPUs available for automatic binding:

```
worker_cpu_affinity auto 01010101;
```

The directive is only available on FreeBSD and Linux.

### **worker\_priority**

SYNTAX: **worker\_priority** *number*;

DEFAULT 0

CONTEXT: main

Defines the scheduling priority for worker processes like it is done by the `nice` command: a negative *number* means higher priority. Allowed range normally varies from -20 to 20.

Example:

```
worker_priority -10;
```

### **worker\_processes**

SYNTAX: **worker\_processes** *number* | `auto`;

DEFAULT 1

CONTEXT: main

Defines the number of worker processes.

The optimal value depends on many factors including (but not limited to) the number of CPU cores, the number of hard disk drives that store data, and load pattern. When one is in doubt, setting it to the number of available CPU cores would be a good start (the value “`auto`” will try to autodetect it).

The `auto` parameter is supported starting from versions 1.3.8 and 1.2.5.

### **worker\_rlimit\_core**

SYNTAX: **worker\_rlimit\_core** *size*;

DEFAULT —

CONTEXT: main

Changes the limit on the largest size of a core file (RLIMIT\_CORE) for worker processes. Used to increase the limit without restarting the main process.

### **worker\_rlimit\_nofile**

SYNTAX: **worker\_rlimit\_nofile** *number*;

DEFAULT —

CONTEXT: main

Changes the limit on the maximum number of open files (RLIMIT\_NOFILE) for worker processes. Used to increase the limit without restarting the main process.

### **worker\_shutdown\_timeout**

SYNTAX: **worker\_shutdown\_timeout** *time*;

DEFAULT —

CONTEXT: main

THIS DIRECTIVE APPEARED IN VERSION 1.11.11.

Configures a timeout for a graceful shutdown of worker processes. When the *time* expires, nginx will try to close all the connections currently open to facilitate shutdown.

### **working\_directory**

SYNTAX: **working\_directory** *directory*;

DEFAULT —

CONTEXT: main

Defines the current working directory for a worker process. It is primarily used when writing a core-file, in which case a worker process should have write permission for the specified directory.



## 1.2 Setting up hashes

### 1.2.1 Overview . . . . . 16

#### 1.2.1 Overview

To quickly process static sets of data such as server names, [map](#) directive's values, MIME types, names of request header strings, nginx uses hash tables. During the start and each re-configuration nginx selects the minimum possible sizes of hash tables such that the bucket size that stores keys with identical hash values does not exceed the configured parameter (hash bucket size). The size of a table is expressed in buckets. The adjustment is continued until the table size exceeds the hash max size parameter. Most hashes have the corresponding directives that allow changing these parameters, for example, for the server names hash they are [server\\_names\\_hash\\_max\\_size](#) and [server\\_names\\_hash\\_bucket\\_size](#).

The hash bucket size parameter is aligned to the size that is a multiple of the processor's cache line size. This speeds up key search in a hash on modern processors by reducing the number of memory accesses. If hash bucket size is equal to one processor's cache line size then the number of memory accesses during the key search will be two in the worst case — first to compute the bucket address, and second during the key search inside the bucket. Therefore, if nginx emits the message requesting to increase either hash max size or hash bucket size then the first parameter should first be increased.

## 1.3 Connection processing methods

### 1.3.1 Overview . . . . . 17

#### 1.3.1 Overview

nginx supports a variety of connection processing methods. The availability of a particular method depends on the platform used. On platforms that support several methods nginx will normally select the most efficient method automatically. However, if needed, a connection processing method can be selected explicitly with the [use](#) directive.

The following connection processing methods are supported:

- `select` — standard method. The supporting module is built automatically on platforms that lack more efficient methods. The `--with-select_module` and `--without-select_module` configuration parameters can be used to forcibly enable or disable the build of this module.
- `poll` — standard method. The supporting module is built automatically on platforms that lack more efficient methods. The `--with-poll_module` and `--without-poll_module` configuration parameters can be used to forcibly enable or disable the build of this module.
- `kqueue` — efficient method used on FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0, and macOS.
- `epoll` — efficient method used on Linux 2.6+.

The `EPOLLRDHUP` (Linux 2.6.17, glibc 2.8) and `EPOLLEXCLUSIVE` (Linux 4.5, glibc 2.24) flags are supported since 1.11.3.

Some older distributions like SuSE 8.2 provide patches that add `epoll` support to 2.4 kernels.

- `/dev/poll` — efficient method used on Solaris 7 11/99+, HP/UX 11.22+ (eventport), IRIX 6.5.15+, and Tru64 UNIX 5.1A+.
- `eventport` — event ports, method used on Solaris 10+ (due to known issues, it is recommended using the `/dev/poll` method instead).

## 1.4 Logging to syslog

### 1.4.1 Overview . . . . . 18

#### 1.4.1 Overview

The [error\\_log](#) and [access\\_log](#) directives support logging to syslog. The following parameters configure logging to syslog:

`server=address`

Defines the address of a syslog server. The address can be specified as a domain name or IP address, with an optional port, or as a UNIX-domain socket path specified after the “unix:” prefix. If port is not specified, the UDP port 514 is used. If a domain name resolves to several IP addresses, the first resolved address is used.

`facility=string`

Sets facility of syslog messages, as defined in [RFC 3164](#). Facility can be one of “kern”, “user”, “mail”, “daemon”, “auth”, “intern”, “lpr”, “news”, “uucp”, “clock”, “authpriv”, “ftp”, “ntp”, “audit”, “alert”, “cron”, “local0”..“local7”. Default is “local7”.

`severity=string`

Sets severity of syslog messages for [access\\_log](#), as defined in [RFC 3164](#). Possible values are the same as for the second parameter (level) of the [error\\_log](#) directive. Default is “info”.

Severity of error messages is determined by nginx, thus the parameter is ignored in the `error_log` directive.

`tag=string`

Sets the tag of syslog messages. Default is “nginx”.

`nohostname`

Disables adding the “hostname” field into the syslog message header (1.9.7).

Example syslog configuration:

```
error_log syslog:server=192.168.1.1 debug;

access_log syslog:server=unix:/var/log/nginx.sock,nohostname;
access_log syslog:server=[2001:db8::1]:12345,facility=local7,tag=nginx,severity
=info combined;
```

Logging to syslog is available since version 1.7.1. As part of our [commercial subscription](#) logging to syslog is available since version 1.5.3.

# Chapter 2

## HTTP server modules

### 2.1 Module ngx\_http\_core\_module

2.1.1 Directives	21
absolute_redirect	21
aio	21
aio_write	22
alias	22
auth_delay	23
chunked_transfer_encoding	23
client_body_buffer_size	24
client_body_in_file_only	24
client_body_in_single_buffer	24
client_body_temp_path	24
client_body_timeout	25
client_header_buffer_size	25
client_header_timeout	25
client_max_body_size	25
connection_pool_size	26
default_type	26
directio	26
directio_alignment	26
disable_symlinks	27
error_page	28
etag	29
http	29
if_modified_since	29
ignore_invalid_headers	29
internal	30
keepalive_disable	30
keepalive_requests	31
keepalive_time	31
keepalive_timeout	31
large_client_header_buffers	32

<a href="#">limit_except</a>	32
<a href="#">limit_rate</a>	32
<a href="#">limit_rate_after</a>	33
<a href="#">lingering_close</a>	33
<a href="#">lingering_time</a>	34
<a href="#">lingering_timeout</a>	34
<a href="#">listen</a>	35
<a href="#">location</a>	38
<a href="#">log_not_found</a>	39
<a href="#">log_subrequest</a>	40
<a href="#">max_ranges</a>	40
<a href="#">merge_slashes</a>	40
<a href="#">msie_padding</a>	41
<a href="#">msie_refresh</a>	41
<a href="#">open_file_cache</a>	41
<a href="#">open_file_cache_errors</a>	42
<a href="#">open_file_cache_min_uses</a>	42
<a href="#">open_file_cache_valid</a>	42
<a href="#">output_buffers</a>	42
<a href="#">port_in_redirect</a>	42
<a href="#">postpone_output</a>	43
<a href="#">read_ahead</a>	43
<a href="#">recursive_error_pages</a>	43
<a href="#">request_pool_size</a>	43
<a href="#">reset_timedout_connection</a>	43
<a href="#">resolver</a>	44
<a href="#">resolver_timeout</a>	45
<a href="#">root</a>	45
<a href="#">satisfy</a>	45
<a href="#">send_lowat</a>	46
<a href="#">send_timeout</a>	46
<a href="#">sendfile</a>	46
<a href="#">sendfile_max_chunk</a>	47
<a href="#">server</a>	47
<a href="#">server_name</a>	47
<a href="#">server_name_in_redirect</a>	49
<a href="#">server_names_hash_bucket_size</a>	49
<a href="#">server_names_hash_max_size</a>	49
<a href="#">server_tokens</a>	50
<a href="#">subrequest_output_buffer_size</a>	50
<a href="#">tcp_nodelay</a>	50
<a href="#">tcp_nopush</a>	50
<a href="#">try_files</a>	51
<a href="#">types</a>	53
<a href="#">types_hash_bucket_size</a>	53
<a href="#">types_hash_max_size</a>	53

<a href="#">underscores_in_headers</a>	54
<a href="#">variables_hash_bucket_size</a>	54
<a href="#">variables_hash_max_size</a>	54
2.1.2 Embedded Variables	54

## 2.1.1 Directives

### absolute\_redirect

SYNTAX: **absolute\_redirect** on | off;

DEFAULT on

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.11.8.

If disabled, redirects issued by nginx will be relative.

See also [server\\_name\\_in\\_redirect](#) and [port\\_in\\_redirect](#) directives.

### aio

SYNTAX: **aio** on | off | threads[=*pool*];

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.8.11.

Enables or disables the use of asynchronous file I/O (AIO) on FreeBSD and Linux:

```
location /video/ {
    aio          on;
    output_buffers 1 64k;
}
```

On FreeBSD, AIO can be used starting from FreeBSD 4.3. Prior to FreeBSD 11.0, AIO can either be linked statically into a kernel:

```
options VFS_AIO
```

or loaded dynamically as a kernel loadable module:

```
kldload aio
```

On Linux, AIO can be used starting from kernel version 2.6.22. Also, it is necessary to enable [directio](#), or otherwise reading will be blocking:

```
location /video/ {
    aio          on;
    directio     512;
    output_buffers 1 128k;
}
```

On Linux, [directio](#) can only be used for reading blocks that are aligned on 512-byte boundaries (or 4K for XFS). File's unaligned end is read in blocking

mode. The same holds true for byte range requests and for FLV requests not from the beginning of a file: reading of unaligned data at the beginning and end of a file will be blocking.

When both AIO and [sendfile](#) are enabled on Linux, AIO is used for files that are larger than or equal to the size specified in the [directio](#) directive, while [sendfile](#) is used for files of smaller sizes or when [directio](#) is disabled.

```
location /video/ {
    sendfile      on;
    aio           on;
    directio      8m;
}
```

Finally, files can be read and [sent](#) using multi-threading (1.7.11), without blocking a worker process:

```
location /video/ {
    sendfile      on;
    aio           threads;
}
```

Read and send file operations are offloaded to threads of the specified [pool](#). If the pool name is omitted, the pool with the name “default” is used. The pool name can also be set with variables:

```
aio threads=pool$disk;
```

By default, multi-threading is disabled, it should be enabled with the `--with-threads` configuration parameter. Currently, multi-threading is compatible only with the [epoll](#), [kqueue](#), and [eventport](#) methods. Multi-threaded sending of files is only supported on Linux.

See also the [sendfile](#) directive.

## aio\_write

SYNTAX: **aio\_write** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.9.13.

If [aio](#) is enabled, specifies whether it is used for writing files. Currently, this only works when using `aio threads` and is limited to writing temporary files with data received from proxied servers.

## alias

SYNTAX: **alias** *path*;

DEFAULT —

CONTEXT: location

Defines a replacement for the specified location. For example, with the following configuration

```
location /i/ {
    alias /data/w3/images/;
}
```

on request of “/i/top.gif”, the file /data/w3/images/top.gif will be sent.

The *path* value can contain variables, except *\$document\_root* and *\$realpath\_root*.

If *alias* is used inside a location defined with a regular expression then such regular expression should contain captures and *alias* should refer to these captures (0.7.40), for example:

```
location ~ ^/users/(.+\.(:gif|jpe?g|png))$ {
    alias /data/w3/images/$1;
}
```

When location matches the last part of the directive’s value:

```
location /images/ {
    alias /data/w3/images/;
}
```

it is better to use the [root](#) directive instead:

```
location /images/ {
    root /data/w3;
}
```

## auth\_delay

SYNTAX: **auth\_delay** *time*;  
DEFAULT 0s  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.17.10.

Delays processing of unauthorized requests with 401 response code to prevent timing attacks when access is limited by [password](#), by the [result of subrequest](#), or by [JWT](#).

## chunked\_transfer\_encoding

SYNTAX: **chunked\_transfer\_encoding** on | off;  
DEFAULT on  
CONTEXT: http, server, location

Allows disabling chunked transfer encoding in HTTP/1.1. It may come in handy when using a software failing to support chunked encoding despite the standard’s requirement.



## client\_body\_buffer\_size

SYNTAX: **client\_body\_buffer\_size** *size*;

DEFAULT 8k|16k

CONTEXT: http, server, location

Sets buffer size for reading client request body. In case the request body is larger than the buffer, the whole body or only its part is written to a [temporary file](#). By default, buffer size is equal to two memory pages. This is 8K on x86, other 32-bit platforms, and x86-64. It is usually 16K on other 64-bit platforms.

## client\_body\_in\_file\_only

SYNTAX: **client\_body\_in\_file\_only** on | clean | off;

DEFAULT off

CONTEXT: http, server, location

Determines whether nginx should save the entire client request body into a file. This directive can be used during debugging, or when using the `$request_body_file` variable, or the `$r->request_body_file` method of the module [ngx\\_http\\_perl\\_module](#).

When set to the value `on`, temporary files are not removed after request processing.

The value `clean` will cause the temporary files left after request processing to be removed.

## client\_body\_in\_single\_buffer

SYNTAX: **client\_body\_in\_single\_buffer** on | off;

DEFAULT off

CONTEXT: http, server, location

Determines whether nginx should save the entire client request body in a single buffer. The directive is recommended when using the `$request_body` variable, to save the number of copy operations involved.

## client\_body\_temp\_path

SYNTAX: **client\_body\_temp\_path** *path* [*level1* [*level2* [*level3*]]];

DEFAULT client\_body\_temp

CONTEXT: http, server, location

Defines a directory for storing temporary files holding client request bodies. Up to three-level subdirectory hierarchy can be used under the specified directory. For example, in the following configuration

```
client_body_temp_path /spool/nginx/client_temp 1 2;
```

a path to a temporary file might look like this:

```
/spool/nginx/client_temp/7/45/00000123457
```

### **client\_body\_timeout**

SYNTAX: **client\_body\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for reading client request body. The timeout is set only for a period between two successive read operations, not for the transmission of the whole request body. If a client does not transmit anything within this time, the request is terminated with the 408 Request Time-out error.

### **client\_header\_buffer\_size**

SYNTAX: **client\_header\_buffer\_size** *size*;

DEFAULT 1k

CONTEXT: http, server

Sets buffer size for reading client request header. For most requests, a buffer of 1K bytes is enough. However, if a request includes long cookies, or comes from a WAP client, it may not fit into 1K. If a request line or a request header field does not fit into this buffer then larger buffers, configured by the [large\\_client\\_header\\_buffers](#) directive, are allocated.

If the directive is specified on the [server](#) level, the value from the default server can be used. Details are provided in the “[Virtual server selection](#)” section.

### **client\_header\_timeout**

SYNTAX: **client\_header\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server

Defines a timeout for reading client request header. If a client does not transmit the entire header within this time, the request is terminated with the 408 Request Time-out error.

### **client\_max\_body\_size**

SYNTAX: **client\_max\_body\_size** *size*;

DEFAULT 1m

CONTEXT: http, server, location

Sets the maximum allowed size of the client request body. If the size in a request exceeds the configured value, the 413 Request Entity Too Large error is returned to the client. Please be aware that browsers cannot

correctly display this error. Setting *size* to 0 disables checking of client request body size.

### connection\_pool\_size

SYNTAX: **connection\_pool\_size** *size*;  
DEFAULT 256|512  
CONTEXT: http, server

Allows accurate tuning of per-connection memory allocations. This directive has minimal impact on performance and should not generally be used. By default, the size is equal to 256 bytes on 32-bit platforms and 512 bytes on 64-bit platforms.

Prior to version 1.9.8, the default value was 256 on all platforms.

### default\_type

SYNTAX: **default\_type** *mime-type*;  
DEFAULT text/plain  
CONTEXT: http, server, location

Defines the default MIME type of a response. Mapping of file name extensions to MIME types can be set with the [types](#) directive.

### directio

SYNTAX: **directio** *size* | off;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.7.7.

Enables the use of the O\_DIRECT flag (FreeBSD, Linux), the F\_NOCACHE flag (macOS), or the `directio` function (Solaris), when reading files that are larger than or equal to the specified *size*. The directive automatically disables (0.7.15) the use of [sendfile](#) for a given request. It can be useful for serving large files:

```
directio 4m;
```

or when using [aio](#) on Linux.

### directio\_alignment

SYNTAX: **directio\_alignment** *size*;  
DEFAULT 512  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.8.11.

Sets the alignment for [directio](#). In most cases, a 512-byte alignment is enough. However, when using XFS under Linux, it needs to be increased to 4K.

### **disable\_symlinks**

SYNTAX: **disable\_symlinks** *off*;  
SYNTAX: **disable\_symlinks** *on* | *if\_not\_owner* [*from=part*];  
DEFAULT *off*  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.1.15.

Determines how symbolic links should be treated when opening files:

*off*

Symbolic links in the pathname are allowed and not checked. This is the default behavior.

*on*

If any component of the pathname is a symbolic link, access to a file is denied.

*if\_not\_owner*

Access to a file is denied if any component of the pathname is a symbolic link, and the link and object that the link points to have different owners.

*from=part*

When checking symbolic links (parameters *on* and *if\_not\_owner*), all components of the pathname are normally checked. Checking of symbolic links in the initial part of the pathname may be avoided by specifying additionally the *from=part* parameter. In this case, symbolic links are checked only from the pathname component that follows the specified initial part. If the value is not an initial part of the pathname checked, the whole pathname is checked as if this parameter was not specified at all. If the value matches the whole file name, symbolic links are not checked. The parameter value can contain variables.

Example:

```
disable_symlinks on from=$document_root;
```

This directive is only available on systems that have the `openat` and `fstatat` interfaces. Such systems include modern versions of FreeBSD, Linux, and Solaris.

Parameters *on* and *if\_not\_owner* add a processing overhead.

On systems that do not support opening of directories only for search, to use these parameters it is required that worker processes have read permissions for all directories being checked.

The [ngx\\_http\\_autoindex\\_module](#), [ngx\\_http\\_random\\_index\\_module](#), and [ngx\\_http\\_dav\\_module](#) modules currently ignore this directive.

## error\_page

SYNTAX: **error\_page** *code* ... [= *response*] *uri*;

DEFAULT —

CONTEXT: http, server, location, if in location

Defines the URI that will be shown for the specified errors. A *uri* value can contain variables.

Example:

```
error_page 404                /404.html;  
error_page 500 502 503 504 /50x.html;
```

This causes an internal redirect to the specified *uri* with the client request method changed to “GET” (for all methods other than “GET” and “HEAD”).

Furthermore, it is possible to change the response code to another using the “=*response*” syntax, for example:

```
error_page 404 =200 /empty.gif;
```

If an error response is processed by a proxied server or a FastCGI/uwsgi/SCGI/gRPC server, and the server may return different response codes (e.g., 200, 302, 401 or 404), it is possible to respond with the code it returns:

```
error_page 404 = /404.php;
```

If there is no need to change URI and method during internal redirection it is possible to pass error processing into a named location:

```
location / {  
    error_page 404 = @fallback;  
}  
  
location @fallback {  
    proxy_pass http://backend;  
}
```

If *uri* processing leads to an error, the status code of the last occurred error is returned to the client.

It is also possible to use URL redirects for error processing:

```
error_page 403                http://example.com/forbidden.html;  
error_page 404 =301 http://example.com/notfound.html;
```

In this case, by default, the response code 302 is returned to the client. It can only be changed to one of the redirect status codes (301, 302, 303, 307, and 308).

The code 307 was not treated as a redirect until versions 1.1.16 and 1.0.13.

The code 308 was not treated as a redirect until version 1.13.0.

These directives are inherited from the previous configuration level if and only if there are no `error_page` directives defined on the current level.

## etag

SYNTAX: **etag** on | off;  
DEFAULT on  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.3.3.

Enables or disables automatic generation of the ETag response header field for static resources.

## http

SYNTAX: **http** { ... }  
DEFAULT —  
CONTEXT: main

Provides the configuration file context in which the HTTP server directives are specified.

## if\_modified\_since

SYNTAX: **if\_modified\_since** off | exact | before;  
DEFAULT exact  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.7.24.

Specifies how to compare modification time of a response with the time in the If-Modified-Since request header field:

**off**  
the If-Modified-Since request header field is ignored (0.7.34);  
**exact**  
exact match;  
**before**  
modification time of a response is less than or equal to the time in the If-Modified-Since request header field.

## ignore\_invalid\_headers

SYNTAX: **ignore\_invalid\_headers** on | off;  
DEFAULT on  
CONTEXT: http, server

Controls whether header fields with invalid names should be ignored. Valid names are composed of English letters, digits, hyphens, and possibly underscores (as controlled by the [underscores\\_in\\_headers](#) directive).

If the directive is specified on the [server](#) level, the value from the default server can be used. Details are provided in the “[Virtual server selection](#)” section.

## internal

SYNTAX: **internal**;

DEFAULT —

CONTEXT: location

Specifies that a given location can only be used for internal requests. For external requests, the client error 404 Not Found is returned. Internal requests are the following:

- requests redirected by the [error\\_page](#), [index](#), [random\\_index](#), and [try\\_files](#) directives;
- requests redirected by the X-Accel-Redirect response header field from an upstream server;
- subrequests formed by the “include virtual” command of the [ngx\\_http\\_ssi\\_module](#) module, by the [ngx\\_http\\_addition\\_module](#) module directives, and by [auth\\_request](#) and [mirror](#) directives;
- requests changed by the [rewrite](#) directive.

Example:

```
error_page 404 /404.html;

location = /404.html {
    internal;
}
```

There is a limit of 10 internal redirects per request to prevent request processing cycles that can occur in incorrect configurations. If this limit is reached, the error 500 Internal Server Error is returned. In such cases, the “rewrite or internal redirection cycle” message can be seen in the error log.

## keepalive\_disable

SYNTAX: **keepalive\_disable** none | *browser* ...;

DEFAULT msie6

CONTEXT: http, server, location

Disables keep-alive connections with misbehaving browsers. The *browser* parameters specify which browsers will be affected. The value `msie6` disables keep-alive connections with old versions of MSIE, once a POST request is received. The value `safari` disables keep-alive connections with Safari and Safari-like browsers on macOS and macOS-like operating systems. The value `none` enables keep-alive connections with all browsers.

Prior to version 1.1.18, the value `safari` matched all Safari and Safari-like browsers on all operating systems, and keep-alive connections with them were disabled by default.

### **keepalive\_requests**

SYNTAX: **keepalive\_requests** *number*;  
DEFAULT 1000  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.8.0.

Sets the maximum number of requests that can be served through one keep-alive connection. After the maximum number of requests are made, the connection is closed.

Closing connections periodically is necessary to free per-connection memory allocations. Therefore, using too high maximum number of requests could result in excessive memory usage and not recommended.

Prior to version 1.19.10, the default value was 100.

### **keepalive\_time**

SYNTAX: **keepalive\_time** *time*;  
DEFAULT 1h  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.19.10.

Limits the maximum time during which requests can be processed through one keep-alive connection. After this time is reached, the connection is closed following the subsequent request processing.

### **keepalive\_timeout**

SYNTAX: **keepalive\_timeout** *timeout* [*header\_timeout*];  
DEFAULT 75s  
CONTEXT: http, server, location

The first parameter sets a timeout during which a keep-alive client connection will stay open on the server side. The zero value disables keep-alive client connections. The optional second parameter sets a value in the `Keep-Alive: timeout=time` response header field. Two parameters may differ.



The `Keep-Alive: timeout=time` header field is recognized by Mozilla and Konqueror. MSIE closes keep-alive connections by itself in about 60 seconds.

### `large_client_header_buffers`

SYNTAX: **large\_client\_header\_buffers** *number size*;

DEFAULT 4 8k

CONTEXT: http, server

Sets the maximum *number* and *size* of buffers used for reading large client request header. A request line cannot exceed the size of one buffer, or the 414 Request-URI Too Large error is returned to the client. A request header field cannot exceed the size of one buffer as well, or the 400 Bad Request error is returned to the client. Buffers are allocated only on demand. By default, the buffer size is equal to 8K bytes. If after the end of request processing a connection is transitioned into the keep-alive state, these buffers are released.

If the directive is specified on the [server](#) level, the value from the default server can be used. Details are provided in the “[Virtual server selection](#)” section.

### `limit_except`

SYNTAX: **limit\_except** *method* ... { ... }

DEFAULT —

CONTEXT: location

Limits allowed HTTP methods inside a location. The *method* parameter can be one of the following: GET, HEAD, POST, PUT, DELETE, MKCOL, COPY, MOVE, OPTIONS, PROPFIND, PROPPATCH, LOCK, UNLOCK, or PATCH. Allowing the GET method makes the HEAD method also allowed. Access to other methods can be limited using the [ngx\\_http\\_access\\_module](#), [ngx\\_http\\_auth\\_basic\\_module](#), and [ngx\\_http\\_auth\\_jwt\\_module](#) (1.13.10) modules directives:

```
limit_except GET {  
    allow 192.168.1.0/32;  
    deny all;  
}
```

Please note that this will limit access to all methods **except** GET and HEAD.

### `limit_rate`

SYNTAX: **limit\_rate** *rate*;

DEFAULT 0

CONTEXT: http, server, location, if in location

Limits the rate of response transmission to a client. The *rate* is specified in bytes per second. The zero value disables rate limiting.

The limit is set per a request, and so if a client simultaneously opens two connections, the overall rate will be twice as much as the specified limit.

Parameter value can contain variables (1.17.0). It may be useful in cases where rate should be limited depending on a certain condition:

```
map $slow $rate {
    1      4k;
    2      8k;
}

limit_rate $rate;
```

Rate limit can also be set in the [\\$limit\\_rate](#) variable, however, since version 1.17.0, this method is not recommended:

```
server {

    if ($slow) {
        set $limit_rate 4k;
    }

    ...

}
```

Rate limit can also be set in the X-Accel-Limit-Rate header field of a proxied server response. This capability can be disabled using the [proxy\\_ignore\\_headers](#), [fastcgi\\_ignore\\_headers](#), [uwsgi\\_ignore\\_headers](#), and [scgi\\_ignore\\_headers](#) directives.

### limit\_rate\_after

SYNTAX: **limit\_rate\_after** *size*;  
DEFAULT 0  
CONTEXT: http, server, location, if in location  
THIS DIRECTIVE APPEARED IN VERSION 0.8.0.

Sets the initial amount after which the further transmission of a response to a client will be rate limited. Parameter value can contain variables (1.17.0).

Example:

```
location /flv/ {
    flv;
    limit_rate_after 500k;
    limit_rate      50k;
}
```

### lingering\_close

SYNTAX: **lingering\_close** off | on | always;  
DEFAULT on  
CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSIONS 1.1.0 AND 1.0.6.

Controls how nginx closes client connections.

The default value “on” instructs nginx to [wait for](#) and [process](#) additional data from a client before fully closing a connection, but only if heuristics suggests that a client may be sending more data.

The value “always” will cause nginx to unconditionally wait for and process additional client data.

The value “off” tells nginx to never wait for more data and close the connection immediately. This behavior breaks the protocol and should not be used under normal circumstances.

To control closing [HTTP/2](#) connections, the directive must be specified on the [server](#) level (1.19.1).

### **lingering\_time**

SYNTAX: **lingering\_time** *time*;

DEFAULT 30s

CONTEXT: http, server, location

When [lingering\\_close](#) is in effect, this directive specifies the maximum time during which nginx will process (read and ignore) additional data coming from a client. After that, the connection will be closed, even if there will be more data.

### **lingering\_timeout**

SYNTAX: **lingering\_timeout** *time*;

DEFAULT 5s

CONTEXT: http, server, location

When [lingering\\_close](#) is in effect, this directive specifies the maximum waiting time for more client data to arrive. If data are not received during this time, the connection is closed. Otherwise, the data are read and ignored, and nginx starts waiting for more data again. The “wait-read-ignore” cycle is repeated, but no longer than specified by the [lingering\\_time](#) directive.

## listen

SYNTAX: **listen** *address[:port]* [default\_server] [ssl] [http2 | spdy]  
 [proxy\_protocol] [setfib=*number*] [fastopen=*number*]  
 [backlog=*number*] [rcvbuf=*size*] [sndbuf=*size*]  
 [accept\_filter=*filter*] [deferred] [bind] [ipv6only=on|off]  
 [reuseport] [so\_keepalive=on|off][*keepidle*]:[*keepintvl*]:[*keepcnt*];

SYNTAX: **listen** *port* [default\_server] [ssl] [http2 | spdy]  
 [proxy\_protocol] [setfib=*number*] [fastopen=*number*]  
 [backlog=*number*] [rcvbuf=*size*] [sndbuf=*size*]  
 [accept\_filter=*filter*] [deferred] [bind] [ipv6only=on|off]  
 [reuseport] [so\_keepalive=on|off][*keepidle*]:[*keepintvl*]:[*keepcnt*];

SYNTAX: **listen** *unix:path* [default\_server] [ssl] [http2 | spdy]  
 [proxy\_protocol] [backlog=*number*] [rcvbuf=*size*] [sndbuf=*size*]  
 [accept\_filter=*filter*] [deferred] [bind]  
 [so\_keepalive=on|off][*keepidle*]:[*keepintvl*]:[*keepcnt*];

DEFAULT \*:80 | \*:8000

CONTEXT: server

Sets the *address* and *port* for IP, or the *path* for a UNIX-domain socket on which the server will accept requests. Both *address* and *port*, or only *address* or only *port* can be specified. An *address* may also be a hostname, for example:

```
listen 127.0.0.1:8000;
listen 127.0.0.1;
listen 8000;
listen *:8000;
listen localhost:8000;
```

IPv6 addresses (0.7.36) are specified in square brackets:

```
listen [::]:8000;
listen [::1];
```

UNIX-domain sockets (0.8.21) are specified with the “unix:” prefix:

```
listen unix:/var/run/nginx.sock;
```

If only *address* is given, the port 80 is used.

If the directive is not present then either \*:80 is used if nginx runs with the superuser privileges, or \*:8000 otherwise.

The default\_server parameter, if present, will cause the server to become the default server for the specified *address:port* pair. If none of the directives have the default\_server parameter then the first server with the *address:port* pair will be the default server for this pair.

In versions prior to 0.8.21 this parameter is named simply default.

The ssl parameter (0.7.14) allows specifying that all connections accepted on this port should work in SSL mode. This allows for a more compact [configuration](#) for the server that handles both HTTP and HTTPS requests.

The `http2` parameter (1.9.5) configures the port to accept [HTTP/2](#) connections. Normally, for this to work the `ssl` parameter should be specified as well, but nginx can also be configured to accept HTTP/2 connections without SSL.

The `spdy` parameter (1.3.15-1.9.4) allows accepting SPDY connections on this port. Normally, for this to work the `ssl` parameter should be specified as well, but nginx can also be configured to accept SPDY connections without SSL.

The `proxy_protocol` parameter (1.5.12) allows specifying that all connections accepted on this port should use the [PROXY protocol](#).

The PROXY protocol version 2 is supported since version 1.13.11.

The `listen` directive can have several additional parameters specific to socket-related system calls. These parameters can be specified in any `listen` directive, but only once for a given *address:port* pair.

In versions prior to 0.8.21, they could only be specified in the `listen` directive together with the `default` parameter.

`setfib=number`

this parameter (0.8.44) sets the associated routing table, FIB (the `SO_SETFIB` option) for the listening socket. This currently works only on FreeBSD.

`fastopen=number`

enables “[TCP Fast Open](#)” for the listening socket (1.5.8) and [limits](#) the maximum length for the queue of connections that have not yet completed the three-way handshake.

Do not enable this feature unless the server can handle receiving the [same SYN packet with data](#) more than once.

`backlog=number`

sets the `backlog` parameter in the `listen` call that limits the maximum length for the queue of pending connections. By default, `backlog` is set to -1 on FreeBSD, DragonFly BSD, and macOS, and to 511 on other platforms.

`rcvbuf=size`

sets the receive buffer size (the `SO_RCVBUF` option) for the listening socket.

`sndbuf=size`

sets the send buffer size (the `SO_SNDBUF` option) for the listening socket.

`accept_filter=filter`

sets the name of accept filter (the `SO_ACCEPTFILTER` option) for the listening socket that filters incoming connections before passing them to accept. This works only on FreeBSD and NetBSD 5.0+. Possible values are [dataready](#) and [httpready](#).

**deferred**

instructs to use a deferred accept (the `TCP_DEFER_ACCEPT` socket option) on Linux.

**bind**

instructs to make a separate `bind` call for a given *address:port* pair. This is useful because if there are several `listen` directives with the same port but different addresses, and one of the `listen` directives listens on all addresses for the given port (*\*:port*), nginx will `bind` only to *\*:port*. It should be noted that the `getsockname` system call will be made in this case to determine the address that accepted the connection. If the `setfib`, `fastopen`, `backlog`, `rcvbuf`, `sndbuf`, `accept_filter`, `deferred`, `ipv6only`, `reuseport`, or `so_keepalive` parameters are used then for a given *address:port* pair a separate `bind` call will always be made.

**ipv6only=on|off**

this parameter (0.7.42) determines (via the `IPV6_V6ONLY` socket option) whether an IPv6 socket listening on a wildcard address `:::` will accept only IPv6 connections or both IPv6 and IPv4 connections. This parameter is turned on by default. It can only be set once on start.

Prior to version 1.3.4, if this parameter was omitted then the operating system's settings were in effect for the socket.

**reuseport**

this parameter (1.9.1) instructs to create an individual listening socket for each worker process (using the `SO_REUSEPORT` socket option on Linux 3.9+ and DragonFly BSD, or `SO_REUSEPORT_LB` on FreeBSD 12+), allowing a kernel to distribute incoming connections between worker processes. This currently works only on Linux 3.9+, DragonFly BSD, and FreeBSD 12+ (1.15.1).

Inappropriate use of this option may have its security [implications](#).

**so\_keepalive=on|off[[*keepidle*]:[*keepintvl*]:[*keepcnt*]]**

this parameter (1.1.11) configures the “TCP keepalive” behavior for the listening socket. If this parameter is omitted then the operating system's settings will be in effect for the socket. If it is set to the value “on”, the `SO_KEEPALIVE` option is turned on for the socket. If it is set to the value “off”, the `SO_KEEPALIVE` option is turned off for the socket. Some operating systems support setting of TCP keepalive parameters on a per-socket basis using the `TCP_KEEPIDLE`, `TCP_KEEPINTVL`, and `TCP_KEEPCNT` socket options. On such systems (currently, Linux 2.4+, NetBSD 5+, and FreeBSD 9.0-STABLE), they can be configured using the *keepidle*, *keepintvl*, and *keepcnt* parameters. One or two parameters may be omitted, in which case the system default setting for the corresponding socket option will be in effect. For example,

```
so_keepalive=30m::10
```

will set the idle timeout (TCP\_KEEPIDLE) to 30 minutes, leave the probe interval (TCP\_KEEPINTVL) at its system default, and set the probes count (TCP\_KEEPCNT) to 10 probes.

Example:

```
listen 127.0.0.1 default_server accept_filter=dataready backlog=1024;
```

## location

SYNTAX: **location** [ = | ~ | ~\* | ^~ ] *uri* { ... }

SYNTAX: **location** @*name* { ... }

DEFAULT —

CONTEXT: server, location

Sets configuration depending on a request URI.

The matching is performed against a normalized URI, after decoding the text encoded in the “%XX” form, resolving references to relative path components “.” and “..”, and possible [compression](#) of two or more adjacent slashes into a single slash.

A location can either be defined by a prefix string, or by a regular expression. Regular expressions are specified with the preceding “~\*” modifier (for case-insensitive matching), or the “~” modifier (for case-sensitive matching). To find location matching a given request, nginx first checks locations defined using the prefix strings (prefix locations). Among them, the location with the longest matching prefix is selected and remembered. Then regular expressions are checked, in the order of their appearance in the configuration file. The search of regular expressions terminates on the first match, and the corresponding configuration is used. If no match with a regular expression is found then the configuration of the prefix location remembered earlier is used.

location blocks can be nested, with some exceptions mentioned below.

For case-insensitive operating systems such as macOS and Cygwin, matching with prefix strings ignores a case (0.7.7). However, comparison is limited to one-byte locales.

Regular expressions can contain captures (0.7.40) that can later be used in other directives.

If the longest matching prefix location has the “^~” modifier then regular expressions are not checked.

Also, using the “=” modifier it is possible to define an exact match of URI and location. If an exact match is found, the search terminates. For example, if a “/” request happens frequently, defining “location = /” will speed up the processing of these requests, as search terminates right after the first comparison. Such a location cannot obviously contain nested locations.

In versions from 0.7.1 to 0.8.41, if a request matched the prefix location without the “=” and “^~” modifiers, the search also terminated and regular expressions were not checked.

Let’s illustrate the above by an example:

```
location = / {  
    [ configuration A ]  
}  
  
location / {  
    [ configuration B ]  
}  
  
location /documents/ {  
    [ configuration C ]  
}  
  
location ^~ /images/ {  
    [ configuration D ]  
}  
  
location ~* \.(gif|jpg|jpeg)$ {  
    [ configuration E ]  
}
```

The “/” request will match configuration A, the “/index.html” request will match configuration B, the “/documents/document.html” request will match configuration C, the “/images/1.gif” request will match configuration D, and the “/documents/1.jpg” request will match configuration E.

The “@” prefix defines a named location. Such a location is not used for a regular request processing, but instead used for request redirection. They cannot be nested, and cannot contain nested locations.

If a location is defined by a prefix string that ends with the slash character, and requests are processed by one of [proxy\\_pass](#), [fastcgi\\_pass](#), [uwsgi\\_pass](#), [scgi\\_pass](#), [memcached\\_pass](#), or [grpc\\_pass](#), then the special processing is performed. In response to a request with URI equal to this string, but without the trailing slash, a permanent redirect with the code 301 will be returned to the requested URI with the slash appended. If this is not desired, an exact match of the URI and location could be defined like this:

```
location /user/ {  
    proxy_pass http://user.example.com;  
}  
  
location = /user {  
    proxy_pass http://login.example.com;  
}
```

## log\_not\_found

SYNTAX: **log\_not\_found** on | off;

DEFAULT on

CONTEXT: http, server, location



Enables or disables logging of errors about not found files into [error\\_log](#).

### log\_subrequest

SYNTAX: **log\_subrequest** on | off;

DEFAULT off

CONTEXT: http, server, location

Enables or disables logging of subrequests into [access\\_log](#).

### max\_ranges

SYNTAX: **max\_ranges** *number*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.2.

Limits the maximum allowed number of ranges in byte-range requests. Requests that exceed the limit are processed as if there were no byte ranges specified. By default, the number of ranges is not limited. The zero value disables the byte-range support completely.

### merge\_slashes

SYNTAX: **merge\_slashes** on | off;

DEFAULT on

CONTEXT: http, server

Enables or disables compression of two or more adjacent slashes in a URI into a single slash.

Note that compression is essential for the correct matching of prefix string and regular expression locations. Without it, the “`//scripts/one.php`” request would not match

```
location /scripts/ {  
    ...  
}
```

and might be processed as a static file. So it gets converted to “`/scripts/one.php`”.

Turning the compression `off` can become necessary if a URI contains base64-encoded names, since base64 uses the “`/`” character internally. However, for security considerations, it is better to avoid turning the compression off.

If the directive is specified on the [server](#) level, the value from the default server can be used. Details are provided in the “[Virtual server selection](#)” section.

## msie\_padding

SYNTAX: **msie\_padding** on | off;

DEFAULT on

CONTEXT: http, server, location

Enables or disables adding comments to responses for MSIE clients with status greater than 400 to increase the response size to 512 bytes.

## msie\_refresh

SYNTAX: **msie\_refresh** on | off;

DEFAULT off

CONTEXT: http, server, location

Enables or disables issuing refreshes instead of redirects for MSIE clients.

## open\_file\_cache

SYNTAX: **open\_file\_cache** off;

SYNTAX: **open\_file\_cache** max=*N* [*inactive=time*];

DEFAULT off

CONTEXT: http, server, location

Configures a cache that can store:

- open file descriptors, their sizes and modification times;
- information on existence of directories;
- file lookup errors, such as “file not found”, “no read permission”, and so on.

Caching of errors should be enabled separately by the [open\\_file\\_cache\\_errors](#) directive.

The directive has the following parameters:

**max**

sets the maximum number of elements in the cache; on cache overflow the least recently used (LRU) elements are removed;

**inactive**

defines a time after which an element is removed from the cache if it has not been accessed during this time; by default, it is 60 seconds;

**off**

disables the cache.

Example:

```
open_file_cache          max=1000 inactive=20s;
open_file_cache_valid    30s;
open_file_cache_min_uses 2;
open_file_cache_errors   on;
```

### **open\_file\_cache\_errors**

SYNTAX: **open\_file\_cache\_errors** on | off;

DEFAULT off

CONTEXT: http, server, location

Enables or disables caching of file lookup errors by [open\\_file\\_cache](#).

### **open\_file\_cache\_min\_uses**

SYNTAX: **open\_file\_cache\_min\_uses** *number*;

DEFAULT 1

CONTEXT: http, server, location

Sets the minimum *number* of file accesses during the period configured by the *inactive* parameter of the [open\\_file\\_cache](#) directive, required for a file descriptor to remain open in the cache.

### **open\_file\_cache\_valid**

SYNTAX: **open\_file\_cache\_valid** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Sets a time after which [open\\_file\\_cache](#) elements should be validated.

### **output\_buffers**

SYNTAX: **output\_buffers** *number size*;

DEFAULT 2 32k

CONTEXT: http, server, location

Sets the *number* and *size* of the buffers used for reading a response from a disk.

Prior to version 1.9.5, the default value was 1 32k.

### **port\_in\_redirect**

SYNTAX: **port\_in\_redirect** on | off;

DEFAULT on

CONTEXT: http, server, location

Enables or disables specifying the port in [absolute](#) redirects issued by nginx.

The use of the primary server name in redirects is controlled by the [server\\_name\\_in\\_redirect](#) directive.

## postpone\_output

SYNTAX: **postpone\_output** *size*;  
DEFAULT 1460  
CONTEXT: http, server, location

If possible, the transmission of client data will be postponed until nginx has at least *size* bytes of data to send. The zero value disables postponing data transmission.

## read\_ahead

SYNTAX: **read\_ahead** *size*;  
DEFAULT 0  
CONTEXT: http, server, location

Sets the amount of pre-reading for the kernel when working with file.

On Linux, the `posix_fadvise(0, 0, 0, POSIX_FADV_SEQUENTIAL)` system call is used, and so the *size* parameter is ignored.

On FreeBSD, the `fctl(O_READAHEAD, size)` system call, supported since FreeBSD 9.0-CURRENT, is used. FreeBSD 7 has to be [patched](#).

## recursive\_error\_pages

SYNTAX: **recursive\_error\_pages** on | off;  
DEFAULT off  
CONTEXT: http, server, location

Enables or disables doing several redirects using the [error\\_page](#) directive. The number of such redirects is [limited](#).

## request\_pool\_size

SYNTAX: **request\_pool\_size** *size*;  
DEFAULT 4k  
CONTEXT: http, server

Allows accurate tuning of per-request memory allocations. This directive has minimal impact on performance and should not generally be used.

## reset\_timedout\_connection

SYNTAX: **reset\_timedout\_connection** on | off;  
DEFAULT off  
CONTEXT: http, server, location

Enables or disables resetting timed out connections and connections [closed](#) with the non-standard code 444 (1.15.2). The reset is performed as follows. Before closing a socket, the `SO_LINGER` option is set on it with a timeout value of 0. When the socket is closed, TCP RST is sent to the client, and

all memory occupied by this socket is released. This helps avoid keeping an already closed socket with filled buffers in a `FIN_WAIT1` state for a long time.

It should be noted that timed out keep-alive connections are closed normally.

## resolver

SYNTAX: **resolver** *address* ... [*valid=time*] [*ipv4=on|off*] [*ipv6=on|off*]  
[*status\_zone=zone*];

DEFAULT —

CONTEXT: http, server, location

Configures name servers used to resolve names of upstream servers into addresses, for example:

```
resolver 127.0.0.1 [::1]:5353;
```

The address can be specified as a domain name or IP address, with an optional port (1.3.1, 1.2.2). If port is not specified, the port 53 is used. Name servers are queried in a round-robin fashion.

Before version 1.1.7, only a single name server could be configured. Specifying name servers using IPv6 addresses is supported starting from versions 1.3.1 and 1.2.2.

By default, nginx will look up both IPv4 and IPv6 addresses while resolving. If looking up of IPv4 or IPv6 addresses is not desired, the `ipv4=off` (1.23.1) or the `ipv6=off` parameter can be specified.

Resolving of names into IPv6 addresses is supported starting from version 1.5.8.

By default, nginx caches answers using the TTL value of a response. An optional `valid` parameter allows overriding it:

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

Before version 1.1.9, tuning of caching time was not possible, and nginx always cached answers for the duration of 5 minutes.

To prevent DNS spoofing, it is recommended configuring DNS servers in a properly secured trusted local network.

The optional `status_zone` parameter (1.17.1) enables [collection](#) of DNS server statistics of requests and responses in the specified *zone*. The parameter is available as part of our [commercial subscription](#).

## resolver\_timeout

SYNTAX: **resolver\_timeout** *time*;

DEFAULT 30s

CONTEXT: http, server, location

Sets a timeout for name resolution, for example:

```
resolver_timeout 5s;
```

## root

SYNTAX: **root** *path*;

DEFAULT html

CONTEXT: http, server, location, if in location

Sets the root directory for requests. For example, with the following configuration

```
location /i/ {  
    root /data/w3;  
}
```

The `/data/w3/i/top.gif` file will be sent in response to the “`/i/top.gif`” request.

The *path* value can contain variables, except *\$document\_root* and *\$realpath\_root*.

A path to the file is constructed by merely adding a URI to the value of the root directive. If a URI has to be modified, the [alias](#) directive should be used.

## satisfy

SYNTAX: **satisfy** all | any;

DEFAULT all

CONTEXT: http, server, location

Allows access if all (all) or at least one (any) of the [ngx\\_http\\_access\\_module](#), [ngx\\_http\\_auth\\_basic\\_module](#), [ngx\\_http\\_auth\\_request\\_module](#), or [ngx\\_http\\_auth\\_jwt\\_module](#) modules allow access.

Example:

```
location / {  
    satisfy any;  
  
    allow 192.168.1.0/32;  
    deny all;  
  
    auth_basic "closed site";  
    auth_basic_user_file conf/htpasswd;  
}
```

## send\_lowat

SYNTAX: **send\_lowat** *size*;  
DEFAULT 0  
CONTEXT: http, server, location

If the directive is set to a non-zero value, nginx will try to minimize the number of send operations on client sockets by using either `NOTE_LOWAT` flag of the [kqueue](#) method or the `SO_SNDLOWAT` socket option. In both cases the specified *size* is used.

This directive is ignored on Linux, Solaris, and Windows.

## send\_timeout

SYNTAX: **send\_timeout** *time*;  
DEFAULT 60s  
CONTEXT: http, server, location

Sets a timeout for transmitting a response to the client. The timeout is set only between two successive write operations, not for the transmission of the whole response. If the client does not receive anything within this time, the connection is closed.

## sendfile

SYNTAX: **sendfile** on | off;  
DEFAULT off  
CONTEXT: http, server, location, if in location

Enables or disables the use of `sendfile`.

Starting from nginx 0.8.12 and FreeBSD 5.2.1, [aio](#) can be used to pre-load data for `sendfile`:

```
location /video/ {  
    sendfile      on;  
    tcp_nopush    on;  
    aio           on;  
}
```

In this configuration, `sendfile` is called with the `SF_NODISKIO` flag which causes it not to block on disk I/O, but, instead, report back that the data are not in memory. nginx then initiates an asynchronous data load by reading one byte. On the first read, the FreeBSD kernel loads the first 128K bytes of a file into memory, although next reads will only load data in 16K chunks. This can be changed using the [read\\_ahead](#) directive.

Before version 1.7.11, pre-loading could be enabled with `aio sendfile;`.

## sendfile\_max\_chunk

SYNTAX: **sendfile\_max\_chunk** *size*;

DEFAULT 2m

CONTEXT: http, server, location

Limits the amount of data that can be transferred in a single `sendfile` call. Without the limit, one fast connection may seize the worker process entirely.

Prior to version 1.21.4, by default there was no limit.

## server

SYNTAX: **server** { ... }

DEFAULT —

CONTEXT: http

Sets configuration for a virtual server. There is no clear separation between IP-based (based on the IP address) and name-based (based on the `Host` request header field) virtual servers. Instead, the [listen](#) directives describe all addresses and ports that should accept connections for the server, and the [server\\_name](#) directive lists all server names. Example configurations are provided in the “[How nginx processes a request](#)” document.

## server\_name

SYNTAX: **server\_name** *name* ...;

DEFAULT ""

CONTEXT: server

Sets names of a virtual server, for example:

```
server {
    server_name example.com www.example.com;
}
```

The first name becomes the primary server name.

Server names can include an asterisk (“`*`”) replacing the first or last part of a name:

```
server {
    server_name example.com *.example.com www.example.*;
}
```

Such names are called wildcard names.

The first two of the names mentioned above can be combined in one:

```
server {
    server_name .example.com;
}
```



It is also possible to use regular expressions in server names, preceding the name with a tilde (“~”):

```
server {
    server_name www.example.com ~^www\d+\.example\.com$;
}
```

Regular expressions can contain captures (0.7.40) that can later be used in other directives:

```
server {
    server_name ~^(www\.)?(.+)$;

    location / {
        root /sites/$2;
    }
}

server {
    server_name _;

    location / {
        root /sites/default;
    }
}
```

Named captures in regular expressions create variables (0.8.25) that can later be used in other directives:

```
server {
    server_name ~^(www\.)?(?<domain>.+)$;

    location / {
        root /sites/$domain;
    }
}

server {
    server_name _;

    location / {
        root /sites/default;
    }
}
```

If the directive’s parameter is set to “*\$hostname*” (0.9.4), the machine’s hostname is inserted.

It is also possible to specify an empty server name (0.7.11):

```
server {
    server_name www.example.com "";
}
```

It allows this server to process requests without the `Host` header field — instead of the default server — for the given address:port pair. This is the default setting.

Before 0.8.48, the machine's hostname was used by default.

During searching for a virtual server by name, if the name matches more than one of the specified variants, (e.g. both a wildcard name and regular expression match), the first matching variant will be chosen, in the following order of priority:

1. the exact name
2. the longest wildcard name starting with an asterisk, e.g. “`*.example.com`”
3. the longest wildcard name ending with an asterisk, e.g. “`mail.*`”
4. the first matching regular expression (in order of appearance in the configuration file)

Detailed description of server names is provided in a separate [Server names](#) document.

#### **server\_name\_in\_redirect**

SYNTAX: **server\_name\_in\_redirect** on | off;

DEFAULT off

CONTEXT: http, server, location

Enables or disables the use of the primary server name, specified by the [server\\_name](#) directive, in [absolute](#) redirects issued by nginx. When the use of the primary server name is disabled, the name from the `Host` request header field is used. If this field is not present, the IP address of the server is used.

The use of a port in redirects is controlled by the [port\\_in\\_redirect](#) directive.

#### **server\_names\_hash\_bucket\_size**

SYNTAX: **server\_names\_hash\_bucket\_size** *size*;

DEFAULT 32 | 64 | 128

CONTEXT: http

Sets the bucket size for the server names hash tables. The default value depends on the size of the processor's cache line. The details of setting up hash tables are provided in a separate [document](#).

#### **server\_names\_hash\_max\_size**

SYNTAX: **server\_names\_hash\_max\_size** *size*;

DEFAULT 512

CONTEXT: http

Sets the maximum *size* of the server names hash tables. The details of setting up hash tables are provided in a separate [document](#).

## server\_tokens

SYNTAX: **server\_tokens** on | off | build | *string*;

DEFAULT on

CONTEXT: http, server, location

Enables or disables emitting nginx version on error pages and in the Server response header field.

The build parameter (1.11.10) enables emitting a [build name](#) along with nginx version.

Additionally, as part of our [commercial subscription](#), starting from version 1.9.13 the signature on error pages and the Server response header field value can be set explicitly using the *string* with variables. An empty string disables the emission of the Server field.

## subrequest\_output\_buffer\_size

SYNTAX: **subrequest\_output\_buffer\_size** *size*;

DEFAULT 4k|8k

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.13.10.

Sets the *size* of the buffer used for storing the response body of a subrequest. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform. It can be made smaller, however.

The directive is applicable only for subrequests with response bodies saved into memory. For example, such subrequests are created by [SSI](#).

## tcp\_nodelay

SYNTAX: **tcp\_nodelay** on | off;

DEFAULT on

CONTEXT: http, server, location

Enables or disables the use of the TCP\_NODELAY option. The option is enabled when a connection is transitioned into the keep-alive state. Additionally, it is enabled on SSL connections, for unbuffered proxying, and for [WebSocket](#) proxying.

## tcp\_nopush

SYNTAX: **tcp\_nopush** on | off;

DEFAULT off

CONTEXT: http, server, location

Enables or disables the use of the TCP\_NOPUSH socket option on FreeBSD or the TCP\_CORK socket option on Linux. The options are enabled only when [sendfile](#) is used. Enabling the option allows

- sending the response header and the beginning of a file in one packet, on Linux and FreeBSD 4.\*;

- sending a file in full packets.

## try\_files

SYNTAX: **try\_files** *file* ... *uri*;  
SYNTAX: **try\_files** *file* ... =*code*;  
DEFAULT —  
CONTEXT: server, location

Checks the existence of files in the specified order and uses the first found file for request processing; the processing is performed in the current context. The path to a file is constructed from the *file* parameter according to the [root](#) and [alias](#) directives. It is possible to check directory's existence by specifying a slash at the end of a name, e.g. “\$uri/”. If none of the files were found, an internal redirect to the *uri* specified in the last parameter is made. For example:

```
location /images/ {
    try_files $uri /images/default.gif;
}

location = /images/default.gif {
    expires 30s;
}
```

The last parameter can also point to a named location, as shown in examples below. Starting from version 0.7.51, the last parameter can also be a *code*:

```
location / {
    try_files $uri $uri/index.html $uri.html =404;
}
```

### Example in proxying Mongrel:

```
location / {
    try_files /system/maintenance.html
             $uri $uri/index.html $uri.html
             @mongrel;
}

location @mongrel {
    proxy_pass http://mongrel;
}
```

### Example for Drupal/FastCGI:

```
location / {
    try_files $uri $uri/ @drupal;
}

location ~ /\.php$ {
    try_files $uri @drupal;

    fastcgi_pass ...;

    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;
```

```

        fastcgi_param SCRIPT_NAME      $fastcgi_script_name;
        fastcgi_param QUERY_STRING     $args;

        ... other fastcgi_param's
    }

    location @drupal {
        fastcgi_pass ...;

        fastcgi_param SCRIPT_FILENAME /path/to/index.php;
        fastcgi_param SCRIPT_NAME     /index.php;
        fastcgi_param QUERY_STRING     q=$uri&$args;

        ... other fastcgi_param's
    }

```

In the following example,

```

location / {
    try_files $uri $uri/ @drupal;
}

```

the `try_files` directive is equivalent to

```

location / {
    error_page 404 = @drupal;
    log_not_found off;
}

```

And here,

```

location ~ /\.php$ {
    try_files $uri @drupal;

    fastcgi_pass ...;

    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;

    ...
}

```

`try_files` checks the existence of the PHP file before passing the request to the FastCGI server.

Example for Wordpress and Joomla:

```

location / {
    try_files $uri $uri/ @wordpress;
}

location ~ /\.php$ {
    try_files $uri @wordpress;

    fastcgi_pass ...;

    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;
    ... other fastcgi_param's
}

location @wordpress {
    fastcgi_pass ...;

    fastcgi_param SCRIPT_FILENAME /path/to/index.php;
    ... other fastcgi_param's
}

```

```
}

```

## types

SYNTAX: **types** { ... }

DEFAULT `text/html html; image/gif gif; image/jpeg jpeg;`

CONTEXT: http, server, location

Maps file name extensions to MIME types of responses. Extensions are case-insensitive. Several extensions can be mapped to one type, for example:

```
types {
    application/octet-stream bin exe dll;
    application/octet-stream deb;
    application/octet-stream dmg;
}
```

A sufficiently full mapping table is distributed with nginx in the `conf/mime.types` file.

To make a particular location emit the “application/octet-stream” MIME type for all requests, the following configuration can be used:

```
location /download/ {
    types { }
    default_type application/octet-stream;
}
```

## types\_hash\_bucket\_size

SYNTAX: **types\_hash\_bucket\_size** *size*;

DEFAULT `64`

CONTEXT: http, server, location

Sets the bucket size for the types hash tables. The details of setting up hash tables are provided in a separate [document](#).

Prior to version 1.5.13, the default value depended on the size of the processor’s cache line.

## types\_hash\_max\_size

SYNTAX: **types\_hash\_max\_size** *size*;

DEFAULT `1024`

CONTEXT: http, server, location

Sets the maximum *size* of the types hash tables. The details of setting up hash tables are provided in a separate [document](#).

### **underscores\_in\_headers**

SYNTAX: **underscores\_in\_headers** on | off;

DEFAULT off

CONTEXT: http, server

Enables or disables the use of underscores in client request header fields. When the use of underscores is disabled, request header fields whose names contain underscores are marked as invalid and become subject to the [ignore\\_invalid\\_headers](#) directive.

If the directive is specified on the [server](#) level, the value from the default server can be used. Details are provided in the “[Virtual server selection](#)” section.

### **variables\_hash\_bucket\_size**

SYNTAX: **variables\_hash\_bucket\_size** *size*;

DEFAULT 64

CONTEXT: http

Sets the bucket size for the variables hash table. The details of setting up hash tables are provided in a separate [document](#).

### **variables\_hash\_max\_size**

SYNTAX: **variables\_hash\_max\_size** *size*;

DEFAULT 1024

CONTEXT: http

Sets the maximum *size* of the variables hash table. The details of setting up hash tables are provided in a separate [document](#).

Prior to version 1.5.13, the default value was 512.

## **2.1.2 Embedded Variables**

The `ngx_http_core_module` module supports embedded variables with names matching the Apache Server variables. First of all, these are variables representing client request header fields, such as `$http_user_agent`, `$http_cookie`, and so on. Also there are other variables:

*\$arg\_name*

argument *name* in the request line

*\$args*

arguments in the request line

*\$binary\_remote\_addr*

client address in a binary form, value's length is always 4 bytes for IPv4 addresses or 16 bytes for IPv6 addresses

*\$body\_bytes\_sent*

number of bytes sent to a client, not counting the response header; this variable is compatible with the “%B” parameter of the `mod_log_config` Apache module

*\$bytes\_sent*

number of bytes sent to a client (1.3.8, 1.2.5)

*\$connection*

connection serial number (1.3.8, 1.2.5)

*\$connection\_requests*

current number of requests made through a connection (1.3.8, 1.2.5)

*\$connection\_time*

connection time in seconds with a milliseconds resolution (1.19.10)

*\$content\_length*

Content-Length request header field

*\$content\_type*

Content-Type request header field

*\$cookie\_name*

the *name* cookie

*\$document\_root*

[root](#) or [alias](#) directive’s value for the current request

*\$document\_uri*

same as *\$uri*

*\$host*

in this order of precedence: host name from the request line, or host name from the `Host` request header field, or the server name matching a request

*\$hostname*

host name

*\$http\_name*

arbitrary request header field; the last part of a variable name is the field name converted to lower case with dashes replaced by underscores

*\$https*

“on” if connection operates in SSL mode, or an empty string otherwise

*\$is\_args*

“?” if a request line has arguments, or an empty string otherwise

*\$limit\_rate*

setting this variable enables response rate limiting; see [limit\\_rate](#)

*\$msec*

current time in seconds with the milliseconds resolution (1.3.9, 1.2.6)

*\$nginx\_version*

nginx version

*\$pid*

PID of the worker process

*\$pipe*

“p” if request was pipelined, “.” otherwise (1.3.12, 1.2.7)



*\$proxy\_protocol\_addr*

client address from the PROXY protocol header (1.5.12)

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the `listen` directive.

*\$proxy\_protocol\_port*

client port from the PROXY protocol header (1.11.0)

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the `listen` directive.

*\$proxy\_protocol\_server\_addr*

server address from the PROXY protocol header (1.17.6)

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the `listen` directive.

*\$proxy\_protocol\_server\_port*

server port from the PROXY protocol header (1.17.6)

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the `listen` directive.

*\$proxy\_protocol\_tlv\_name*

TLV from the PROXY Protocol header (1.23.2). The name can be a TLV type name or its numeric value. In the latter case, the value is hexadecimal and should be prefixed with `0x`:

```
$proxy_protocol_tlv_alpn
$proxy_protocol_tlv_0x01
```

SSL TLVs can also be accessed by TLV type name or its numeric value, both prefixed by `ssl_`:

```
$proxy_protocol_tlv_ssl_version
$proxy_protocol_tlv_ssl_0x21
```

The following TLV type names are supported:

- `alpn` (0x01) - upper layer protocol used over the connection
- `authority` (0x02) - host name value passed by the client
- `unique_id` (0x05) - unique connection id
- `netns` (0x30) - name of the namespace
- `ssl` (0x20) - binary SSL TLV structure

The following SSL TLV type names are supported:

- `ssl_version` (0x21) - SSL version used in client connection
- `ssl_cn` (0x22) - SSL certificate Common Name
- `ssl_cipher` (0x23) - name of the used cipher
- `ssl_sig_alg` (0x24) - algorithm used to sign the certificate
- `ssl_key_alg` (0x25) - public-key algorithm

Also, the following special SSL TLV type name is supported:

- `ssl_verify` - client SSL certificate verification result, 0 if the client presented a certificate and it was successfully verified, non-zero otherwise.

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the [listen](#) directive.

*\$query\_string*

same as *\$args*

*\$realpath\_root*

an absolute pathname corresponding to the [root](#) or [alias](#) directive's value for the current request, with all symbolic links resolved to real paths

*\$remote\_addr*

client address

*\$remote\_port*

client port

*\$remote\_user*

user name supplied with the Basic authentication

*\$request*

full original request line

*\$request\_body*

request body

The variable's value is made available in locations processed by the [proxy\\_pass](#), [fastcgi\\_pass](#), [uwsgi\\_pass](#), and [scgi\\_pass](#) directives when the request body was read to a [memory buffer](#).

*\$request\_body\_file*

name of a temporary file with the request body

At the end of processing, the file needs to be removed. To always write the request body to a file, [client\\_body\\_in\\_file\\_only](#) needs to be enabled. When the name of a temporary file is passed in a proxied request or in a request to a FastCGI/uwsgi/SCGI server, passing the request body should be disabled by the [proxy\\_pass\\_request\\_body off](#), [fastcgi\\_pass\\_request\\_body off](#), [uwsgi\\_pass\\_request\\_body off](#), or [scgi\\_pass\\_request\\_body off](#) directives, respectively.

*\$request\_completion*

“OK” if a request has completed, or an empty string otherwise

*\$request\_filename*

file path for the current request, based on the [root](#) or [alias](#) directives, and the request URI

*\$request\_id*

unique request identifier generated from 16 random bytes, in hexadecimal (1.11.0)

*\$request\_length*

request length (including request line, header, and request body) (1.3.12, 1.2.7)

*\$request\_method*

request method, usually “GET” or “POST”

*\$request\_time*

request processing time in seconds with a milliseconds resolution (1.3.9, 1.2.6); time elapsed since the first bytes were read from the client

*\$request\_uri*

full original request URI (with arguments)

*\$scheme*

request scheme, “http” or “https”

*\$sent\_http\_name*

arbitrary response header field; the last part of a variable name is the field name converted to lower case with dashes replaced by underscores

*\$sent\_trailer\_name*

arbitrary field sent at the end of the response (1.13.2); the last part of a variable name is the field name converted to lower case with dashes replaced by underscores

*\$server\_addr*

an address of the server which accepted a request

Computing a value of this variable usually requires one system call. To avoid a system call, the [listen](#) directives must specify addresses and use the `bind` parameter.

*\$server\_name*

name of the server which accepted a request

*\$server\_port*

port of the server which accepted a request

*\$server\_protocol*

request protocol, usually “HTTP/1.0”, “HTTP/1.1”, or “[HTTP/2.0](#)”

*\$status*

response status (1.3.2, 1.2.2)

*\$tcpinfo\_rtt*, *\$tcpinfo\_rttvar*, *\$tcpinfo\_snd\_cwnd*, *\$tcpinfo\_rcv\_space*

information about the client TCP connection; available on systems that support the `TCP_INFO` socket option

*\$time\_iso8601*

local time in the ISO 8601 standard format (1.3.12, 1.2.7)

*\$time\_local*

local time in the Common Log Format (1.3.12, 1.2.7)

*\$uri*

current URI in request, [normalized](#)

The value of *\$uri* may change during request processing, e.g. when doing internal redirects, or when using index files.

## 2.2 Module ngx\_http\_access\_module

2.2.1	Summary	59
2.2.2	Example Configuration	59
2.2.3	Directives	59
	allow	59
	deny	59

### 2.2.1 Summary

The `ngx_http_access_module` module allows limiting access to certain client addresses.

Access can also be limited by [password](#), by the [result of subrequest](#), or by [JWT](#). Simultaneous limitation of access by address and by password is controlled by the [satisfy](#) directive.

### 2.2.2 Example Configuration

```
location / {
    deny 192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0db8::/32;
    deny all;
}
```

The rules are checked in sequence until the first match is found. In this example, access is allowed only for IPv4 networks `10.1.1.0/16` and `192.168.1.0/24` excluding the address `192.168.1.1`, and for IPv6 network `2001:0db8::/32`. In case of a lot of rules, the use of the [ngx-http-geo-module](#) module variables is preferable.

### 2.2.3 Directives

#### allow

SYNTAX: **allow** *address* | *CIDR* | `unix:` | `all`;

DEFAULT —

CONTEXT: http, server, location, limit\_except

Allows access for the specified network or address. If the special value `unix:` is specified (1.5.1), allows access for all UNIX-domain sockets.

#### deny

SYNTAX: **deny** *address* | *CIDR* | `unix:` | `all`;

DEFAULT —

CONTEXT: http, server, location, limit\_except

Denies access for the specified network or address. If the special value `unix:` is specified (1.5.1), denies access for all UNIX-domain sockets.

## 2.3 Module ngx\_http\_addition\_module

2.3.1	Summary	61
2.3.2	Example Configuration	61
2.3.3	Directives	61
	add_before_body	61
	add_after_body	61
	addition_types	62

### 2.3.1 Summary

The ngx\_http\_addition\_module module is a filter that adds text before and after a response. This module is not built by default, it should be enabled with the `--with-http_addition_module` configuration parameter.

### 2.3.2 Example Configuration

```
location / {
    add_before_body /before_action;
    add_after_body  /after_action;
}
```

### 2.3.3 Directives

#### add\_before\_body

SYNTAX: **add\_before\_body** *uri*;

DEFAULT —

CONTEXT: http, server, location

Adds the text returned as a result of processing a given subrequest before the response body. An empty string ("") as a parameter cancels addition inherited from the previous configuration level.

#### add\_after\_body

SYNTAX: **add\_after\_body** *uri*;

DEFAULT —

CONTEXT: http, server, location

Adds the text returned as a result of processing a given subrequest after the response body. An empty string ("") as a parameter cancels addition inherited from the previous configuration level.

## **addition\_types**

SYNTAX: **addition\_types** *mime-type* ...;

DEFAULT `text/html`

CONTEXT: `http`, `server`, `location`

THIS DIRECTIVE APPEARED IN VERSION 0.7.9.

Allows adding text in responses with the specified MIME types, in addition to “`text/html`”. The special value “`*`” matches any MIME type (0.8.29).

## 2.4 Module ngx\_http\_api\_module

2.4.1	Summary	63
2.4.2	Example Configuration	63
2.4.3	Directives	64
	api	64
	status_zone	65
2.4.4	Compatibility	65
2.4.5	Endpoints	66
2.4.6	Response Objects	88

### 2.4.1 Summary

The `ngx_http_api_module` module (1.13.3) provides REST API for accessing various status information, configuring upstream server groups on-the-fly, and managing [key-value pairs](#) without the need of reconfiguring nginx.

The module supersedes the [ngx\\_http\\_status\\_module](#) and [ngx\\_http\\_upstream\\_conf\\_module](#) modules.

When using the PATCH or POST methods, make sure that the payload does not exceed the [buffer size](#) for reading the client request body, otherwise, the 413 Request Entity Too Large error may be returned.

This module is available as part of our [commercial subscription](#).

### 2.4.2 Example Configuration

```
http {
    upstream backend {
        zone http_backend 64k;

        server backend1.example.com weight=5;
        server backend2.example.com;
    }

    proxy_cache_path /data/nginx/cache_backend keys_zone=cache_backend:10m;

    server {
        server_name backend.example.com;

        location / {
            proxy_pass http://backend;
            proxy_cache cache_backend;

            health_check;
        }

        status_zone server_backend;
    }

    keyval_zone zone=one:32k state=one.keyval;
    keyval $arg_text $text zone=one;
```



```
server {
    listen 127.0.0.1;

    location /api {
        api write=on;
        allow 127.0.0.1;
        deny all;
    }
}

stream {
    upstream backend {
        zone stream_backend 64k;

        server backend1.example.com:12345 weight=5;
        server backend2.example.com:12345;
    }

    server {
        listen      127.0.0.1:12345;
        proxy_pass  backend;
        status_zone server_backend;
        health_check;
    }
}
```

All API requests include a supported API version in the URI. Examples of API requests with this configuration:

```
http://127.0.0.1/api/8/
http://127.0.0.1/api/8/nginx
http://127.0.0.1/api/8/connections
http://127.0.0.1/api/8/http/requests
http://127.0.0.1/api/8/http/server_zones/server_backend
http://127.0.0.1/api/8/http/caches/cache_backend
http://127.0.0.1/api/8/http/upstreams/backend
http://127.0.0.1/api/8/http/upstreams/backend/servers/
http://127.0.0.1/api/8/http/upstreams/backend/servers/1
http://127.0.0.1/api/8/http/keyvals/one?key=arg1
http://127.0.0.1/api/8/stream/
http://127.0.0.1/api/8/stream/server_zones/server_backend
http://127.0.0.1/api/8/stream/upstreams/
http://127.0.0.1/api/8/stream/upstreams/backend
http://127.0.0.1/api/8/stream/upstreams/backend/servers/1
```

## 2.4.3 Directives

### api

SYNTAX: **api** [write=on|off];

DEFAULT —

CONTEXT: location

Turns on the REST API interface in the surrounding location. Access to this location should be [limited](#).

The `write` parameter determines whether the API is read-only or read-write. By default, the API is read-only.

All API requests should contain a supported API version in the URI. If the request URI equals the location prefix, the list of supported API versions

is returned. The current API version is “8”.

The optional “fields” argument in the request line specifies which fields of the requested objects will be output:

```
http://127.0.0.1/api/8/nginx?fields=version,build
```

### status\_zone

SYNTAX: **status\_zone** *zone*;

DEFAULT —

CONTEXT: server, location, if in location

THIS DIRECTIVE APPEARED IN VERSION 1.13.12.

Enables collection of virtual [http](#) or [stream](#) server status information in the specified *zone*. Several servers may share the same zone.

Starting from 1.17.0, status information can be collected per [location](#). The special value `off` disables statistics collection in nested location blocks. Note that the statistics is collected in the context of a location where processing ends. It may be different from the original location, if an [internal redirect](#) happens during request processing.

## 2.4.4 Compatibility

- The `ssl` data for each HTTP [upstream](#), [server zone](#), and stream [upstream](#), [server zone](#), were added in version 8.
- The `codes` data in responses for each HTTP [upstream](#), [server zone](#), and [location zone](#) were added in version 7.
- The `/stream/limit_conns/` data were added in version 6.
- The `/http/limit_conns/` data were added in version 6.
- The `/http/limit_reqs/` data were added in version 6.
- The “expire” parameter of a [key-value](#) pair can be [set](#) or [changed](#) since version 5.
- The `/resolvers/` data were added in version 5.
- The `/http/location_zones/` data were added in version 5.
- The `path` and `method` fields of [nginx error object](#) were removed in version 4. These fields continue to exist in earlier api versions, but show an empty value.
- The `/stream/zone_sync/` data were added in version 3.
- The [drain](#) parameter was added in version 2.
- The `/stream/keyvals/` data were added in version 2.

## 2.4.5 Endpoints

/

Supported methods:

- GET - Return list of root endpoints

Returns a list of root endpoints.

Possible responses:

- 200 - Success, returns an array of strings
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

/nginx

Supported methods:

- GET - Return status of nginx running instance

Returns nginx version, build name, address, number of configuration reloads, IDs of master and worker processes.

Request parameters:

`fields` (string, optional)

Limits which fields of nginx running instance will be output.

Possible responses:

- 200 - Success, returns [nginx](#)
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

/processes

Supported methods:

- GET - Return nginx processes status

Returns the number of abnormally terminated and respawned child processes.

Possible responses:

- 200 - Success, returns [Processes](#)
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

DELETE - Reset nginx processes statistics

Resets counters of abnormally terminated and respawned child processes.

Possible responses:

- – 204 - Success
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

/connections

Supported methods:

- GET - Return client connections statistics

Returns statistics of client connections.

Request parameters:

`fields` (string, optional)

Limits which fields of the connections statistics will be output.

Possible responses:

- 200 - Success, returns [Connections](#)
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

DELETE - Reset client connections statistics

Resets statistics of accepted and dropped client connections.

Possible responses:

- – 204 - Success
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

`/slabs/`

Supported methods:

- GET - Return status of all slabs

Returns status of slabs for each shared memory zone with slab allocator.

Request parameters:

`fields` (string, optional)

Limits which fields of slab zones will be output. If the “`fields`” value is empty, then only zone names will be output.

Possible responses:

- 200 - Success, returns a collection of “[Shared memory zone with slab allocator](#)” objects for all slabs
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/slabs/{slabZoneName}`

Parameters common for all methods:

`slabZoneName` (string, required)

The name of the shared memory zone with slab allocator.

Supported methods:

- GET - Return status of a slab

Returns status of slabs for a particular shared memory zone with slab allocator.

Request parameters:

`fields` (string, optional)

Limits which fields of the slab zone will be output.

Possible responses:

- 200 - Success, returns [Shared memory zone with slab allocator](#)
- 404 - Slab not found (SlabNotFound), unknown version (UnknownVersion), returns [Error](#)

DELETE - Reset slab statistics

Resets the “reqs” and “fails” metrics for each memory slot.

Possible responses:

- – 204 - Success
- 404 - Slab not found (SlabNotFound), unknown version (UnknownVersion), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)

/http/

Supported methods:

- GET - Return list of HTTP-related endpoints
- Returns a list of first level HTTP endpoints.

Possible responses:

- 200 - Success, returns an array of strings
- 404 - Unknown version (UnknownVersion), returns [Error](#)

/http/requests

Supported methods:

- GET - Return HTTP requests statistics
- Returns status of client HTTP requests.

Request parameters:

`fields` (string, optional)

Limits which fields of client HTTP requests statistics will be output.

Possible responses:

- 200 - Success, returns [HTTP Requests](#)
- 404 - Unknown version (UnknownVersion), returns [Error](#)

DELETE - Reset HTTP requests statistics

Resets the number of total client HTTP requests.

Possible responses:

- – 204 - Success
- 404 - Unknown version (UnknownVersion), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)

/http/server\_zones/

Supported methods:

- GET - Return status of all HTTP server zones

Returns status information for each HTTP [server zone](#).

Request parameters:

`fields` (string, optional)

Limits which fields of server zones will be output. If the “fields” value is empty, then only server zone names will be output.

Possible responses:

- 200 - Success, returns a collection of “[HTTP Server Zone](#)” objects for all HTTP server zones
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/http/server_zones/{httpServerZoneName}`

Parameters common for all methods:

`httpServerZoneName` (string, required)

The name of an HTTP server zone.

Supported methods:

- GET - Return status of an HTTP server zone

Returns status of a particular HTTP server zone.

Request parameters:

`fields` (string, optional)

Limits which fields of the server zone will be output.

Possible responses:

- 200 - Success, returns [HTTP Server Zone](#)
- 404 - Server zone not found (`ServerZoneNotFound`), unknown version (`UnknownVersion`), returns [Error](#)

DELETE - Reset statistics for an HTTP server zone

Resets statistics of accepted and discarded requests, responses, received and sent bytes, counters of SSL handshakes and session reuses in a particular HTTP server zone.

Possible responses:

- – 204 - Success
- 404 - Server zone not found (`ServerZoneNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

`/http/location_zones/`

Supported methods:

- GET - Return status of all HTTP location zones

Returns status information for each HTTP location zone.

Request parameters:

`fields` (string, optional)

Limits which fields of location zones will be output. If the “fields” value is empty, then only zone names will be output.

Possible responses:

- 200 - Success, returns a collection of “[HTTP Location Zone](#)” objects for all HTTP location zones
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/http/location_zones/{httpLocationZoneName}`

Parameters common for all methods:

`httpLocationZoneName` (string, required)

The name of an HTTP location zone.

Supported methods:

- GET - Return status of an HTTP location zone  
Returns status of a particular HTTP location zone.

Request parameters:

`fields` (string, optional)

Limits which fields of the location zone will be output.

Possible responses:

- 200 - Success, returns [HTTP Location Zone](#)
- 404 - Location zone not found (`LocationZoneNotFound`), unknown version (`UnknownVersion`), returns [Error](#)

DELETE - Reset statistics for a location zone.

Resets statistics of accepted and discarded requests, responses, received and sent bytes in a particular location zone.

Possible responses:

- – 204 - Success
- 404 - Location zone not found (`LocationZoneNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

`/http/caches/`

Supported methods:

- GET - Return status of all caches  
Returns status of each cache configured by [proxy\\_cache\\_path](#) and other “\*\_cache\_path” directives.

Request parameters:

`fields` (string, optional)

Limits which fields of cache zones will be output. If the “fields” value is empty, then only names of cache zones will be output.

Possible responses:

- 200 - Success, returns a collection of "[HTTP Cache](#)" objects for all HTTP caches
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/http/caches/{httpCacheZoneName}`

Parameters common for all methods:

`httpCacheZoneName` (string, required)

The name of the cache zone.

Supported methods:

- GET - Return status of a cache  
Returns status of a particular cache.

Request parameters:

`fields` (string, optional)

Limits which fields of the cache zone will be output.

Possible responses:

- 200 - Success, returns [HTTP Cache](#)
- 404 - Cache not found (`CacheNotFound`), unknown version (`UnknownVersion`), returns [Error](#)

DELETE - Reset cache statistics

Resets statistics of cache hits/misses in a particular cache zone.

Possible responses:

- – 204 - Success
- 404 - Cache not found (`CacheNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

`/http/limit_conns/`

Supported methods:

- GET - Return status of all HTTP `limit_conn` zones  
Returns status information for each HTTP [limit\\_conn zone](#).

Request parameters:

`fields` (string, optional)

Limits which fields of `limit_conn` zones will be output. If the "fields" value is empty, then only zone names will be output.

Possible responses:

- 200 - Success, returns a collection of "[HTTP Connections Limiting](#)" objects for all HTTP `limit_conns`
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)



/http/limit\_conns/{httpLimitConnZoneName}

Parameters common for all methods:

httpLimitConnZoneName (string, required)

The name of a [limit\\_conn zone](#).

Supported methods:

- GET - Return status of an HTTP limit\_conn zone

Returns status of a particular HTTP [limit\\_conn zone](#).

Request parameters:

fields (string, optional)

Limits which fields of the [limit\\_conn zone](#) will be output.

Possible responses:

- 200 - Success, returns [HTTP Connections Limiting](#)
- 404 - limit\_conn not found (LimitConnNotFound), unknown version (UnknownVersion), returns [Error](#)

DELETE - Reset statistics for an HTTP limit\_conn zone

Resets the connection limiting statistics.

Possible responses:

- – 204 - Success
- 404 - limit\_conn not found (LimitConnNotFound), unknown version (UnknownVersion), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)

/http/limit\_reqs/

Supported methods:

- GET - Return status of all HTTP limit\_req zones

Returns status information for each HTTP [limit\\_req zone](#).

Request parameters:

fields (string, optional)

Limits which fields of limit\_req zones will be output. If the “fields” value is empty, then only zone names will be output.

Possible responses:

- 200 - Success, returns a collection of “[HTTP Requests Rate Limiting](#)” objects for all HTTP limit reqs
- 404 - Unknown version (UnknownVersion), returns [Error](#)

/http/limit\_reqs/{httpLimitReqZoneName}

Parameters common for all methods:

httpLimitReqZoneName (string, required)

The name of a [limit\\_req zone](#).

Supported methods:

- GET - Return status of an HTTP limit\_req zone  
Returns status of a particular HTTP [limit\\_req zone](#).  
Request parameters:  
`fields` (string, optional)  
Limits which fields of the [limit\\_req zone](#) will be output.  
Possible responses:
  - 200 - Success, returns [HTTP Requests Rate Limiting](#)
  - 404 - limit\_req not found (`LimitReqNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- DELETE - Reset statistics for an HTTP limit\_req zone  
Resets the requests limiting statistics.  
Possible responses:
  - 204 - Success
  - 404 - limit\_req not found (`LimitReqNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
  - 405 - Method disabled (`MethodDisabled`), returns [Error](#)

/http/upstreams/

Supported methods:

- GET - Return status of all HTTP upstream server groups  
Returns status of each HTTP upstream server group and its servers.  
Request parameters:  
`fields` (string, optional)  
Limits which fields of upstream server groups will be output. If the “fields” value is empty, only names of upstreams will be output.  
Possible responses:
  - 200 - Success, returns a collection of “[HTTP Upstream](#)” objects for all HTTP upstreams
  - 404 - Unknown version (`UnknownVersion`), returns [Error](#)

/http/upstreams/{httpUpstreamName}/

Parameters common for all methods:

`httpUpstreamName` (string, required)  
The name of an HTTP upstream server group.

Supported methods:

- GET - Return status of an HTTP upstream server group  
Returns status of a particular HTTP upstream server group and its servers.  
Request parameters:

fields (string, optional)

Limits which fields of the upstream server group will be output.

Possible responses:

- 200 - Success, returns [HTTP Upstream](#)
- 400 - Upstream is static (UpstreamStatic), returns [Error](#)
- 404 - Unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)

DELETE - Reset statistics of an HTTP upstream server group

Resets the statistics for each upstream server in an upstream server group and queue statistics.

Possible responses:

- – 204 - Success
- 400 - Upstream is static (UpstreamStatic), returns [Error](#)
- 404 - Unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)

/http/upstreams/{httpUpstreamName}/servers/

Parameters common for all methods:

httpUpstreamName (string, required)

The name of an upstream server group.

Supported methods:

- GET - Return configuration of all servers in an HTTP upstream server group

Returns configuration of each server in a particular HTTP upstream server group.

Possible responses:

- 200 - Success, returns an array of [HTTP Upstream Servers](#)
- 400 - Upstream is static (UpstreamStatic), returns [Error](#)
- 404 - Unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)

POST - Add a server to an HTTP upstream server group

Adds a new server to an HTTP upstream server group. Server parameters are specified in the JSON format.

Request parameters:

- postHttpUpstreamServer ([HTTP Upstream Server](#), required)  
Address of a new server and other optional parameters in the JSON format. The “ID”, “backup”, and “service” parameters cannot be changed.

Possible responses:

- 201 - Created, returns [HTTP Upstream Server](#)
- 400 - Upstream is static (`UpstreamStatic`), invalid “*parameter*” value (`UpstreamConfFormatError`), missing “*server*” argument (`UpstreamConfFormatError`), unknown parameter “*name*” (`UpstreamConfFormatError`), nested object or list (`UpstreamConfFormatError`), “*error*” while parsing (`UpstreamBadAddress`), service upstream “*host*” may not have port (`UpstreamBadAddress`), service upstream “*host*” requires domain name (`UpstreamBadAddress`), invalid “*weight*” (`UpstreamBadWeight`), invalid “*max\_conns*” (`UpstreamBadMaxConns`), invalid “*max\_fails*” (`UpstreamBadMaxFails`), invalid “*fail\_timeout*” (`UpstreamBadFailTimeout`), invalid “*slow\_start*” (`UpstreamBadSlowStart`), reading request body failed (`BodyReadError`), route is too long (`UpstreamBadRoute`), “*service*” is empty (`UpstreamBadService`), no resolver defined to resolve (`UpstreamConfNoResolver`), upstream “*name*” has no backup (`UpstreamNoBackup`), upstream “*name*” memory exhausted (`UpstreamOutOfMemory`), returns [Error](#)
- 404 - Unknown version (`UnknownVersion`), upstream not found (`UpstreamNotFound`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)
- 409 - Entry exists (`EntryExists`), returns [Error](#)
- 415 - JSON error (`JsonError`), returns [Error](#)

```
/http/upstreams/{httpUpstreamName}/servers/
{httpUpstreamServerId}
```

Parameters common for all methods:

`httpUpstreamName` (string, required)

The name of the upstream server group.

`httpUpstreamServerId` (string, required)

The ID of the server.

Supported methods:

- GET - Return configuration of a server in an HTTP upstream server group

Returns configuration of a particular server in the HTTP upstream server group.

Possible responses:

- 200 - Success, returns [HTTP Upstream Server](#)
- 400 - Upstream is static (`UpstreamStatic`), invalid server ID (`UpstreamBadServerId`), returns [Error](#)

- 404 - Server with ID “*id*” does not exist (UpstreamServerNotFound), unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)

PATCH - Modify a server in an HTTP upstream server group

Modifies settings of a particular server in an HTTP upstream server group. Server parameters are specified in the JSON format.

Request parameters:

- patchHttpUpstreamServer ([HTTP Upstream Server](#), required)  
Server parameters, specified in the JSON format. The “ID”, “backup”, and “service” parameters cannot be changed.

Possible responses:

- 200 - Success, returns [HTTP Upstream Server](#)
- 400 - Upstream is static (UpstreamStatic), invalid “parameter” value (UpstreamConfFormatError), unknown parameter “name” (UpstreamConfFormatError), nested object or list (UpstreamConfFormatError), “error” while parsing (UpstreamBadAddress), invalid “server” argument (UpstreamBadAddress), invalid server ID (UpstreamBadServerId), invalid “weight” (UpstreamBadWeight), invalid “max\_conns” (UpstreamBadMaxConns), invalid “max\_fails” (UpstreamBadMaxFails), invalid “fail\_timeout” (UpstreamBadFailTimeout), invalid “slow\_start” (UpstreamBadSlowStart), reading request body failed BodyReadError), route is too long (UpstreamBadRoute), “service” is empty (UpstreamBadService), server “ID” address is immutable (UpstreamServerImmutable), server “ID” weight is immutable (UpstreamServerWeightImmutable), upstream “name” memory exhausted (UpstreamOutOfMemory), returns [Error](#)
- 404 - Server with ID “*id*” does not exist (UpstreamServerNotFound), unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)
- 415 - JSON error (JsonError), returns [Error](#)

DELETE - Remove a server from an HTTP upstream server group

Removes a server from an HTTP upstream server group.

Possible responses:

- – 200 - Success, returns an array of [HTTP Upstream Servers](#)

- 400 - Upstream is static (UpstreamStatic), invalid server ID (UpstreamBadServerId), server “*id*” not removable (UpstreamServerImmutable), returns [Error](#)
- 404 - Server with ID “*id*” does not exist (UpstreamServerNotFound), unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)

/http/keyvals/

Supported methods:

- GET - Return key-value pairs from all HTTP keyval zones  
Returns key-value pairs for each HTTP keyval shared memory [zone](#).  
Request parameters:

fields (string, optional)

If the “fields” value is empty, then only HTTP keyval zone names will be output.

Possible responses:

- 200 - Success, returns a collection of “[HTTP Keyval Shared Memory Zone](#)” objects for all HTTP keyvals
- 404 - Unknown version (UnknownVersion), returns [Error](#)

/http/keyvals/{httpKeyvalZoneName}

Parameters common for all methods:

httpKeyvalZoneName (string, required)

The name of an HTTP keyval shared memory zone.

Supported methods:

- GET - Return key-value pairs from an HTTP keyval zone  
Returns key-value pairs stored in a particular HTTP keyval shared memory [zone](#).

Request parameters:

key (string, optional)

Get a particular key-value pair from the HTTP keyval zone.

Possible responses:

- 200 - Success, returns [HTTP Keyval Shared Memory Zone](#)
- 404 - Keyval not found (KeyvalNotFound), keyval key not found (KeyvalKeyNotFound), unknown version (UnknownVersion), returns [Error](#)

POST - Add a key-value pair to the HTTP keyval zone

Adds a new key-value pair to the HTTP keyval shared memory [zone](#). Several key-value pairs can be entered if the HTTP keyval shared memory zone is empty.

Request parameters:

- Key-value ([HTTP Keyval Shared Memory Zone](#), required)  
A key-value pair is specified in the JSON format. Several key-value pairs can be entered if the HTTP keyval shared memory zone is empty. Expiration time in milliseconds can be specified for a key-value pair with the `expire` parameter which overrides the `timeout` parameter of the `keyval_zone` directive.

Possible responses:

- 201 - Created
- 400 - Invalid JSON (`KeyvalFormatError`), invalid key format (`KeyvalFormatError`), key required (`KeyvalFormatError`), keyval timeout is not enabled (`KeyvalFormatError`), only one key can be added (`KeyvalFormatError`), reading request body failed (`BodyReadError`), returns [Error](#)
- 404 - Keyval not found (`KeyvalNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)
- 409 - Entry exists (`EntryExists`), key already exists (`KeyvalKeyExists`), returns [Error](#)
- 413 - Request Entity Too Large, returns [Error](#)
- 415 - JSON error (`JsonError`), returns [Error](#)

PATCH - Modify a key-value or delete a key

Changes the value of the selected key in the key-value pair, deletes a key by setting the key value to null, changes expiration time of a key-value pair. If [synchronization](#) of keyval zones in a cluster is enabled, deletes a key only on a target cluster node. Expiration time in milliseconds can be specified for a key-value pair with the `expire` parameter which overrides the `timeout` parameter of the `keyval_zone` directive.

Request parameters:

- `httpKeyvalZoneKeyValue` ([HTTP Keyval Shared Memory Zone](#), required)  
A new value for the key is specified in the JSON format.

Possible responses:

- 204 - Success
- 400 - Invalid JSON (`KeyvalFormatError`), key required (`KeyvalFormatError`), keyval timeout is not enabled (`KeyvalFormatError`), only one key can be updated (`KeyvalFormatError`), reading request body failed (`BodyReadError`), returns [Error](#)
- 404 - Keyval not found (`KeyvalNotFound`), keyval key not found (`KeyvalKeyNotFound`), unknown version (`UnknownVersion`), returns [Error](#)

- 405 - Method disabled (`MethodDisabled`), returns [Error](#)
- 413 - Request Entity Too Large, returns [Error](#)
- 415 - JSON error (`JsonError`), returns [Error](#)

DELETE - Empty the HTTP keyval zone

Deletes all key-value pairs from the HTTP keyval shared memory [zone](#). If [synchronization](#) of keyval zones in a cluster is enabled, empties the keyval zone only on a target cluster node.

Possible responses:

- – 204 - Success
- 404 - Keyval not found (`KeyvalNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

`/stream/`

Supported methods:

- GET - Return list of stream-related endpoints
- Returns a list of first level stream endpoints.

Possible responses:

- 200 - Success, returns an array of strings
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/stream/server_zones/`

Supported methods:

- GET - Return status of all stream server zones
- Returns status information for each stream [server zone](#).

Request parameters:

`fields (string, optional)`

Limits which fields of server zones will be output. If the “fields” value is empty, then only server zone names will be output.

Possible responses:

- 200 - Success, returns a collection of “[Stream Server Zone](#)” objects for all stream server zones
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/stream/server_zones/{streamServerZoneName}`

Parameters common for all methods:

`streamServerZoneName (string, required)`

The name of a stream server zone.

Supported methods:



- GET - Return status of a stream server zone

Returns status of a particular stream server zone.

Request parameters:

`fields` (string, optional)

Limits which fields of the server zone will be output.

Possible responses:

- 200 - Success, returns [Stream Server Zone](#)
- 404 - Server zone not found (`ServerZoneNotFound`), unknown version (`UnknownVersion`), returns [Error](#)

DELETE - Reset statistics for a stream server zone

Resets statistics of accepted and discarded connections, sessions, received and sent bytes, counters of SSL handshakes and session reuses in a particular stream server zone.

Possible responses:

- – 204 - Success
- 404 - Server zone not found (`ServerZoneNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

`/stream/limit_conns/`

Supported methods:

- GET - Return status of all stream limit\_conn zones

Returns status information for each stream [limit\\_conn zone](#).

Request parameters:

`fields` (string, optional)

Limits which fields of limit\_conn zones will be output. If the “fields” value is empty, then only zone names will be output.

Possible responses:

- 200 - Success, returns a collection of “[Stream Connections Limiting](#)” objects for all stream limit conns
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/stream/limit_conns/{streamLimitConnZoneName}`

Parameters common for all methods:

`streamLimitConnZoneName` (string, required)

The name of a [limit\\_conn zone](#).

Supported methods:

- GET - Return status of an stream limit\_conn zone

Returns status of a particular stream [limit\\_conn zone](#).

Request parameters:

`fields` (string, optional)

Limits which fields of the [limit\\_conn zone](#) will be output.

Possible responses:

- 200 - Success, returns [Stream Connections Limiting](#)
- 404 - `limit_conn` not found (`LimitConnNotFound`), unknown version (`UnknownVersion`), returns [Error](#)

DELETE - Reset statistics for a stream `limit_conn` zone

Resets the connection limiting statistics.

Possible responses:

- – 204 - Success
- 404 - `limit_conn` not found (`LimitConnNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

`/stream/upstreams/`

Supported methods:

- GET - Return status of all stream upstream server groups
- Returns status of each stream upstream server group and its servers.

Request parameters:

`fields` (string, optional)

Limits which fields of upstream server groups will be output. If the “`fields`” value is empty, only names of upstreams will be output.

Possible responses:

- 200 - Success, returns a collection of “[Stream Upstream](#)” objects for all stream upstreams
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/stream/upstreams/{streamUpstreamName}/`

Parameters common for all methods:

`streamUpstreamName` (string, required)

The name of a stream upstream server group.

Supported methods:

- GET - Return status of a stream upstream server group
- Returns status of a particular stream upstream server group and its servers.

Request parameters:

`fields` (string, optional)

Limits which fields of the upstream server group will be output.

Possible responses:

- 200 - Success, returns [Stream Upstream](#)

- 400 - Upstream is static (UpstreamStatic), returns [Error](#)
- 404 - Unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)

DELETE - Reset statistics of a stream upstream server group

Resets the statistics for each upstream server in an upstream server group.

Possible responses:

- – 204 - Success
- 400 - Upstream is static (UpstreamStatic), returns [Error](#)
- 404 - Unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)

/stream/upstreams/{streamUpstreamName}/servers/

Parameters common for all methods:

streamUpstreamName (string, required)

The name of an upstream server group.

Supported methods:

- GET - Return configuration of all servers in a stream upstream server group

Returns configuration of each server in a particular stream upstream server group.

Possible responses:

- 200 - Success, returns an array of [Stream Upstream Servers](#)
- 400 - Upstream is static (UpstreamStatic), returns [Error](#)
- 404 - Unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)

POST - Add a server to a stream upstream server group

Adds a new server to a stream upstream server group. Server parameters are specified in the JSON format.

Request parameters:

- postStreamUpstreamServer ([Stream Upstream Server](#), required)

Address of a new server and other optional parameters in the JSON format. The “ID”, “backup”, and “service” parameters cannot be changed.

Possible responses:

- 201 - Created, returns [Stream Upstream Server](#)
- 400 - Upstream is static (UpstreamStatic), invalid “parameter” value (UpstreamConfFormatError), missing “server” argument (UpstreamConfFormatError), unknown parameter “name” (UpstreamConfFormatError),

nested object or list (UpstreamConfFormatError), “error” while parsing (UpstreamBadAddress), no port in server “host” (UpstreamBadAddress), service upstream “host” may not have port (UpstreamBadAddress), service upstream “host” requires domain name (UpstreamBadAddress), invalid “weight” (UpstreamBadWeight), invalid “max\_conns” (UpstreamBadMaxConns), invalid “max\_fails” (UpstreamBadMaxFails), invalid “fail\_timeout” (UpstreamBadFailTimeout), invalid “slow\_start” (UpstreamBadSlowStart), “service” is empty (UpstreamBadService), no resolver defined to resolve (UpstreamConfNoResolver), upstream “name” has no backup (UpstreamNoBackup), upstream “name” memory exhausted (UpstreamOutOfMemory), reading request body failed BodyReadError), returns [Error](#)

- 404 - Unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)
- 409 - Entry exists (EntryExists), returns [Error](#)
- 415 - JSON error (JsonError), returns [Error](#)

/stream/upstreams/{streamUpstreamName}/servers/  
{streamUpstreamServerId}

Parameters common for all methods:

streamUpstreamName (string, required)

The name of the upstream server group.

streamUpstreamServerId (string, required)

The ID of the server.

Supported methods:

- GET - Return configuration of a server in a stream upstream server group

Returns configuration of a particular server in the stream upstream server group.

Possible responses:

- 200 - Success, returns [Stream Upstream Server](#)
- 400 - Upstream is static (UpstreamStatic), invalid server ID (UpstreamBadServerId), returns [Error](#)
- 404 - Unknown version (UnknownVersion), upstream not found (UpstreamNotFound), server with ID “id” does not exist (UpstreamServerNotFound), returns [Error](#)

PATCH - Modify a server in a stream upstream server group

Modifies settings of a particular server in a stream upstream server group. Server parameters are specified in the JSON format.

Request parameters:

- patchStreamUpstreamServer ([Stream Upstream Server](#), required)  
Server parameters, specified in the JSON format. The “ID”, “backup”, and “service” parameters cannot be changed.

Possible responses:

- 200 - Success, returns [Stream Upstream Server](#)
- 400 - Upstream is static (UpstreamStatic), invalid “parameter” value (UpstreamConfFormatError), unknown parameter “name” (UpstreamConfFormatError), nested object or list (UpstreamConfFormatError), “error” while parsing (UpstreamBadAddress), invalid “server” argument (UpstreamBadAddress), no port in server “host” (UpstreamBadAddress), invalid server ID (UpstreamBadServerId), invalid “weight” (UpstreamBadWeight), invalid “max\_conns” (UpstreamBadMaxConns), invalid “max\_fails” (UpstreamBadMaxFails), invalid “fail\_timeout” (UpstreamBadFailTimeout), invalid “slow\_start” (UpstreamBadSlowStart), reading request body failed (BodyReadError), “service” is empty (UpstreamBadService), server “ID” address is immutable (UpstreamServerImmutable), server “ID” weight is immutable (UpstreamServerWeightImmutable), upstream “name” memory exhausted (UpstreamOutOfMemory), returns [Error](#)
- 404 - Server with ID “id” does not exist (UpstreamServerNotFound), unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)
- 415 - JSON error (JsonError), returns [Error](#)

DELETE - Remove a server from a stream upstream server group

Removes a server from a stream server group.

Possible responses:

- – 200 - Success, returns an array of [Stream Upstream Servers](#)
- 400 - Upstream is static (UpstreamStatic), invalid server ID (UpstreamBadServerId), server “id” not removable (UpstreamServerImmutable), returns [Error](#)
- 404 - Server with ID “id” does not exist (UpstreamServerNotFound), unknown version (UnknownVersion), upstream not found (UpstreamNotFound), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)

/stream/keyvals/

Supported methods:

- GET - Return key-value pairs from all stream keyval zones  
Returns key-value pairs for each stream keyval shared memory [zone](#).

Request parameters:

`fields` (string, optional)

If the “fields” value is empty, then only stream keyval zone names will be output.

Possible responses:

- 200 - Success, returns a collection of “[Stream Keyval Shared Memory Zone](#)” objects for all stream keyvals
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

/stream/keyvals/{streamKeyvalZoneName}

Parameters common for all methods:

`streamKeyvalZoneName` (string, required)

The name of a stream keyval shared memory zone.

Supported methods:

- GET - Return key-value pairs from a stream keyval zone  
Returns key-value pairs stored in a particular stream keyval shared memory [zone](#).

Request parameters:

`key` (string, optional)

Get a particular key-value pair from the stream keyval zone.

Possible responses:

- 200 - Success, returns [Stream Keyval Shared Memory Zone](#)
- 404 - Keyval not found (`KeyvalNotFound`), keyval key not found (`KeyvalKeyNotFound`), unknown version (`UnknownVersion`), returns [Error](#)

POST - Add a key-value pair to the stream keyval zone

Adds a new key-value pair to the stream keyval shared memory [zone](#). Several key-value pairs can be entered if the stream keyval shared memory zone is empty.

Request parameters:

- Key-value ([Stream Keyval Shared Memory Zone](#), required)  
A key-value pair is specified in the JSON format. Several key-value pairs can be entered if the stream keyval shared memory zone is empty. Expiration time in milliseconds can be specified for a key-value pair with the `expire` parameter which overrides the `timeout` parameter of the [keyval\\_zone](#) directive.

Possible responses:

- 201 - Created
- 400 - Invalid JSON (KeyvalFormatError), invalid key format (KeyvalFormatError), key required (KeyvalFormatError), keyval timeout is not enabled (KeyvalFormatError), only one key can be added (KeyvalFormatError), reading request body failed (BodyReadError), returns [Error](#)
- 404 - Keyval not found (KeyvalNotFound), unknown version (UnknownVersion), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)
- 409 - Entry exists (EntryExists), key already exists (KeyvalKeyExists), returns [Error](#)
- 413 - Request Entity Too Large, returns [Error](#)
- 415 - JSON error (JsonError), returns [Error](#)

PATCH - Modify a key-value or delete a key

Changes the value of the selected key in the key-value pair, deletes a key by setting the key value to null, changes expiration time of a key-value pair. If [synchronization](#) of keyval zones in a cluster is enabled, deletes a key only on a target cluster node. Expiration time is specified in milliseconds with the `expire` parameter which overrides the `timeout` parameter of the [keyval\\_zone](#) directive.

Request parameters:

- `streamKeyvalZoneKeyValue` ([Stream Keyval Shared Memory Zone](#), required)  
A new value for the key is specified in the JSON format.

Possible responses:

- 204 - Success
- 400 - Invalid JSON (KeyvalFormatError), key required (KeyvalFormatError), keyval timeout is not enabled (KeyvalFormatError), only one key can be updated (KeyvalFormatError), reading request body failed (BodyReadError), returns [Error](#)
- 404 - Keyval not found (KeyvalNotFound), keyval key not found (KeyvalKeyNotFound), unknown version (UnknownVersion), returns [Error](#)
- 405 - Method disabled (MethodDisabled), returns [Error](#)
- 413 - Request Entity Too Large, returns [Error](#)
- 415 - JSON error (JsonError), returns [Error](#)

DELETE - Empty the stream keyval zone

Deletes all key-value pairs from the stream keyval shared memory [zone](#). If [synchronization](#) of keyval zones in a cluster is enabled, empties the keyval zone only on a target cluster node.

Possible responses:

- – 204 - Success
- 404 - Keyval not found (`KeyvalNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

`/stream/zone_sync/`

Supported methods:

- GET - Return sync status of a node
- Returns synchronization status of a cluster node.

Possible responses:

- 200 - Success, returns [Stream Zone Sync Node](#)
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/resolvers/`

Supported methods:

- GET - Return status for all resolver zones
- Returns status information for each resolver zone.

Request parameters:

`fields` (string, optional)

Limits which fields of resolvers statistics will be output.

Possible responses:

- 200 - Success, returns a collection of "[Resolver Zone](#)" objects for all resolvers
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

`/resolvers/{resolverZoneName}`

Parameters common for all methods:

`resolverZoneName` (string, required)

The name of a resolver zone.

Supported methods:

- GET - Return statistics of a resolver zone
- Returns statistics stored in a particular resolver zone.

Request parameters:

`fields` (string, optional)

Limits which fields of the resolver zone will be output (requests, responses, or both).

Possible responses:

- 200 - Success, returns [Resolver Zone](#)
- 404 - Resolver zone not found (`ResolverZoneNotFound`), unknown version (`UnknownVersion`), returns [Error](#)



DELETE - Reset statistics for a resolver zone.

Resets statistics in a particular resolver zone.

Possible responses:

- – 204 - Success
- 404 - Resolver zone not found (`ResolverZoneNotFound`), unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

/ssl

Supported methods:

- GET - Return SSL statistics

Returns SSL statistics.

Request parameters:

`fields (string, optional)`

Limits which fields of SSL statistics will be output.

Possible responses:

- 200 - Success, returns [SSL](#)
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)

DELETE - Reset SSL statistics

Resets counters of SSL handshakes and session reuses.

Possible responses:

- – 204 - Success
- 404 - Unknown version (`UnknownVersion`), returns [Error](#)
- 405 - Method disabled (`MethodDisabled`), returns [Error](#)

## 2.4.6 Response Objects

- nginx:

General information about nginx:

`version (string)`

Version of nginx.

`build (string)`

Name of nginx build.

`address (string)`

The address of the server that accepted status request.

`generation (integer)`

The total number of configuration [reloads](#).

`load_timestamp (string)`

Time of the last reload of configuration, in the ISO 8601 format with millisecond resolution.

timestamp (string)

Current time in the ISO 8601 format with millisecond resolution.

pid (integer)

The ID of the worker process that handled status request.

ppid (integer)

The ID of the master process that started the [worker process](#).

Example:

```
{
  "nginx" : {
    "version" : "1.21.6",
    "build" : "nginx-plus-r27",
    "address" : "206.251.255.64",
    "generation" : 6,
    "load_timestamp" : "2022-06-28T11:15:44.467Z",
    "timestamp" : "2022-06-28T09:26:07.305Z",
    "pid" : 32212,
    "ppid" : 32210
  }
}
```

- Processes:

respawned (integer)

The total number of abnormally terminated and respawned child processes.

Example:

```
{
  "respawned" : 0
}
```

- Connections:

The number of accepted, dropped, active, and idle connections.

accepted (integer)

The total number of accepted client connections.

dropped (integer)

The total number of dropped client connections.

active (integer)

The current number of active client connections.

idle (integer)

The current number of idle client connections.

Example:

```
{
  "accepted" : 4968119,
  "dropped" : 0,
  "active" : 5,
```

```

    "idle" : 117
}

```

- SSL:

handshakes (integer)

The total number of successful SSL handshakes.

handshakes\_failed (integer)

The total number of failed SSL handshakes.

session\_reuses (integer)

The total number of session reuses during SSL handshake.

no\_common\_protocol (integer)

The number of SSL handshakes failed because of no common protocol.

no\_common\_cipher (integer)

The number of SSL handshakes failed because of no shared cipher.

handshake\_timeout (integer)

The number of SSL handshakes failed because of a timeout.

peer\_rejected\_cert (integer)

The number of failed SSL handshakes when nginx presented the certificate to the client but it was rejected with a corresponding alert message.

verify\_failures

SSL certificate verification errors

no\_cert (integer)

A client did not provide the required certificate.

expired\_cert (integer)

An expired or not yet valid certificate was presented by a client.

revoked\_cert (integer)

A revoked certificate was presented by a client.

hostname\_mismatch (integer)

Server's certificate doesn't match the hostname.

other (integer)

Other SSL certificate verification errors.

Example:

```

{
  "handshakes" : 79572,
  "handshakes_failed" : 21025,
  "session_reuses" : 15762,
  "no_common_protocol" : 4,
  "no_common_cipher" : 2,
  "handshake_timeout" : 0,
  "peer_rejected_cert" : 0,
  "verify_failures" : {
    "no_cert" : 0,
    "expired_cert" : 2,
    "revoked_cert" : 1,
    "hostname_mismatch" : 2,

```

```

    "other" : 1
  }
}

```

- Shared memory zone with slab allocator:

Status zone with slab allocator

pages

The number of free and used memory pages.

used (integer)

The current number of used memory pages.

free (integer)

The current number of free memory pages.

slots

Status data for memory slots (8, 16, 32, 64, 128, etc.)

A collection of "Memory Slot" objects

Example:

```

{
  "pages" : {
    "used" : 1143,
    "free" : 2928
  },
  "slots" : {
    "8" : {
      "used" : 0,
      "free" : 0,
      "reqs" : 0,
      "fails" : 0
    },
    "16" : {
      "used" : 0,
      "free" : 0,
      "reqs" : 0,
      "fails" : 0
    },
    "32" : {
      "used" : 0,
      "free" : 0,
      "reqs" : 0,
      "fails" : 0
    },
    "64" : {
      "used" : 1,
      "free" : 63,
      "reqs" : 1,
      "fails" : 0
    },
    "128" : {
      "used" : 0,
      "free" : 0,
      "reqs" : 0,
      "fails" : 0
    },
    "256" : {
      "used" : 18078,
      "free" : 178,
      "reqs" : 1635736,
      "fails" : 0
    }
  }
}

```

```

    }
}

```

- Memory Slot:

`used (integer)`

The current number of used memory slots.

`free (integer)`

The current number of free memory slots.

`reqs (integer)`

The total number of attempts to allocate memory of specified size.

`fails (integer)`

The number of unsuccessful attempts to allocate memory of specified size.

#### HTTP Requests:

- `total (integer)`

The total number of client requests.

`current (integer)`

The current number of client requests.

#### Example:

```

{
  "total" : 10624511,
  "current" : 4
}

```

- HTTP Server Zone:

`processing (integer)`

The number of client requests that are currently being processed.

`requests (integer)`

The total number of client requests received from clients.

`responses`

The total number of responses sent to clients, the number of responses with status codes “1xx”, “2xx”, “3xx”, “4xx”, and “5xx”, and the number of responses per each status code.

`1xx (integer)`

The number of responses with “1xx” status codes.

`2xx (integer)`

The number of responses with “2xx” status codes.

`3xx (integer)`

The number of responses with “3xx” status codes.

`4xx (integer)`

The number of responses with “4xx” status codes.

`5xx (integer)`  
The number of responses with “5xx” status codes.

`codes`  
The number of responses per each status code.

`codeNumber (integer)`  
The number of responses with this particular status code.

`total (integer)`  
The total number of responses sent to clients.

`discarded (integer)`  
The total number of requests completed without sending a response.

`received (integer)`  
The total number of bytes received from clients.

`sent (integer)`  
The total number of bytes sent to clients.

`ssl`

`handshakes (integer)`  
The total number of successful SSL handshakes.

`handshakes_failed (integer)`  
The total number of failed SSL handshakes.

`session_reuses (integer)`  
The total number of session reuses during SSL handshake.

`no_common_protocol (integer)`  
The number of SSL handshakes failed because of no common protocol.

`no_common_cipher (integer)`  
The number of SSL handshakes failed because of no shared cipher.

`handshake_timeout (integer)`  
The number of SSL handshakes failed because of a timeout.

`peer_rejected_cert (integer)`  
The number of failed SSL handshakes when nginx presented the certificate to the client but it was rejected with a corresponding alert message.

`verify_failures`  
SSL certificate verification errors

`no_cert (integer)`  
A client did not provide the required certificate.

`expired_cert (integer)`  
An expired or not yet valid certificate was presented by a client.

`revoked_cert (integer)`  
A revoked certificate was presented by a client.

`other (integer)`  
Other SSL certificate verification errors.

Example:

```
{
  "processing" : 1,
  "requests" : 706690,
  "responses" : {
    "1xx" : 0,
    "2xx" : 699482,
    "3xx" : 4522,
    "4xx" : 907,
    "5xx" : 266,
    "codes" : {
      "200" : 699482,
      "301" : 4522,
      "404" : 907,
      "503" : 266
    },
    "total" : 705177
  },
  "discarded" : 1513,
  "received" : 172711587,
  "sent" : 19415530115,
  "ssl" : {
    "handshakes" : 104303,
    "handshakes_failed" : 1421,
    "session_reuses" : 54645,
    "no_common_protocol" : 4,
    "no_common_cipher" : 2,
    "handshake_timeout" : 0,
    "peer_rejected_cert" : 0,
    "verify_failures" : {
      "no_cert" : 0,
      "expired_cert" : 2,
      "revoked_cert" : 1,
      "other" : 1
    }
  }
}
```

- HTTP Location Zone:

`requests (integer)`

The total number of client requests received from clients.

`responses`

The total number of responses sent to clients, the number of responses with status codes “1xx”, “2xx”, “3xx”, “4xx”, and “5xx”, and the number of responses per each status code.

`1xx (integer)`

The number of responses with “1xx” status codes.

`2xx (integer)`

The number of responses with “2xx” status codes.

`3xx (integer)`

The number of responses with “3xx” status codes.

`4xx (integer)`

The number of responses with “4xx” status codes.

`5xx (integer)`

The number of responses with “5xx” status codes.

codes

The number of responses per each status code.

codeNumber (integer)

The number of responses with this particular status code.

total (integer)

The total number of responses sent to clients.

discarded (integer)

The total number of requests completed without sending a response.

received (integer)

The total number of bytes received from clients.

sent (integer)

The total number of bytes sent to clients.

Example:

```
{
  "requests" : 706690,
  "responses" : {
    "1xx" : 0,
    "2xx" : 699482,
    "3xx" : 4522,
    "4xx" : 907,
    "5xx" : 266,
    "codes" : {
      "200" : 112674,
      "301" : 4522,
      "404" : 2504,
      "503" : 266
    },
    "total" : 705177
  },
  "discarded" : 1513,
  "received" : 172711587,
  "sent" : 19415530115
}
```

- HTTP Cache:

size (integer)

The current size of the cache.

max\_size (integer)

The limit on the maximum size of the cache specified in the configuration.

cold (boolean)

A boolean value indicating whether the “cache loader” process is still loading data from disk into the cache.

hit

responses (integer)

The total number of [valid](#) responses read from the cache.

bytes (integer)

The total number of bytes read from the cache.

stale



responses (integer)  
The total number of expired responses read from the cache (see [proxy\\_cache\\_use\\_stale](#) and other “\*\_cache\_use\_stale” directives).

bytes (integer)  
The total number of bytes read from the cache.

updating

responses (integer)  
The total number of expired responses read from the cache while responses were being updated (see [proxy\\_cache\\_use\\_stale](#) and other “\*\_cache\_use\_stale” directives).

bytes (integer)  
The total number of bytes read from the cache.

revalidated

responses (integer)  
The total number of expired and revalidated responses read from the cache (see [proxy\\_cache\\_revalidate](#) and other “\*\_cache\_revalidate” directives).

bytes (integer)  
The total number of bytes read from the cache.

miss

responses (integer)  
The total number of responses not found in the cache.

bytes (integer)  
The total number of bytes read from the proxied server.

responses\_written (integer)  
The total number of responses written to the cache.

bytes\_written (integer)  
The total number of bytes written to the cache.

expired

responses (integer)  
The total number of expired responses not taken from the cache.

bytes (integer)  
The total number of bytes read from the proxied server.

responses\_written (integer)  
The total number of responses written to the cache.

bytes\_written (integer)  
The total number of bytes written to the cache.

bypass

responses (integer)  
The total number of responses not looked up in the cache due to the [proxy\\_cache\\_bypass](#) and other “\*\_cache\_bypass” directives.

bytes (integer)

The total number of bytes read from the proxied server.  
 responses\_written (integer)  
 The total number of responses written to the cache.  
 bytes\_written (integer)  
 The total number of bytes written to the cache.

Example:

```
{
  "size" : 530915328,
  "max_size" : 536870912,
  "cold" : false,
  "hit" : {
    "responses" : 254032,
    "bytes" : 6685627875
  },
  "stale" : {
    "responses" : 0,
    "bytes" : 0
  },
  "updating" : {
    "responses" : 0,
    "bytes" : 0
  },
  "revalidated" : {
    "responses" : 0,
    "bytes" : 0
  },
  "miss" : {
    "responses" : 1619201,
    "bytes" : 53841943822
  },
  "expired" : {
    "responses" : 45859,
    "bytes" : 1656847080,
    "responses_written" : 44992,
    "bytes_written" : 1641825173
  },
  "bypass" : {
    "responses" : 200187,
    "bytes" : 5510647548,
    "responses_written" : 200173,
    "bytes_written" : 44992
  }
}
```

- HTTP Connections Limiting:

passed (integer)  
 The total number of connections that were neither limited nor accounted as limited.  
 rejected (integer)  
 The total number of connections that were rejected.  
 rejected\_dry\_run (integer)  
 The total number of connections accounted as rejected in the [dry run](#) mode.

Example:

```
{
  "passed" : 15,
  "rejected" : 0,
  "rejected_dry_run" : 2
}
```

- HTTP Requests Rate Limiting:

`passed (integer)`

The total number of requests that were neither limited nor accounted as limited.

`delayed (integer)`

The total number of requests that were delayed.

`rejected (integer)`

The total number of requests that were rejected.

`delayed_dry_run (integer)`

The total number of requests accounted as delayed in the [dry run](#) mode.

`rejected_dry_run (integer)`

The total number of requests accounted as rejected in the [dry run](#) mode.

Example:

```
{
  "passed" : 15,
  "delayed" : 4,
  "rejected" : 0,
  "delayed_dry_run" : 1,
  "rejected_dry_run" : 2
}
```

- HTTP Upstream:

`peers`

An array of:

`id (integer)`

The ID of the server.

`server (string)`

An [address](#) of the server.

`service (string)`

The [service](#) parameter value of the [server](#) directive.

`name (string)`

The name of the server specified in the [server](#) directive.

`backup (boolean)`

A boolean value indicating whether the server is a [backup](#) server.

`weight (integer)`  
Weight of the server.

`state (string)`  
Current state, which may be one of “up”, “draining”, “down”, “unavail”, “checking”, and “unhealthy”.

`active (integer)`  
The current number of active connections.

`ssl`

`handshakes (integer)`  
    The total number of successful SSL handshakes.

`handshakes_failed (integer)`  
    The total number of failed SSL handshakes.

`session_reuses (integer)`  
    The total number of session reuses during SSL handshake.

`no_common_protocol (integer)`  
    The number of SSL handshakes failed because of no common protocol.

`handshake_timeout (integer)`  
    The number of SSL handshakes failed because of a timeout.

`peer_rejected_cert (integer)`  
    The number of failed SSL handshakes when nginx presented the certificate to the upstream server but it was rejected with a corresponding alert message.

`verify_failures`  
    SSL certificate verification errors

`expired_cert (integer)`  
        An expired or not yet valid certificate was presented by an upstream server.

`revoked_cert (integer)`  
        A revoked certificate was presented by an upstream server.

`hostname_mismatch (integer)`  
        Server’s certificate doesn’t match the hostname.

`other (integer)`  
        Other SSL certificate verification errors.

`max_conns (integer)`  
The `max_conns` limit for the server.

`requests (integer)`  
The total number of client requests forwarded to this server.

`responses`

`1xx (integer)`  
    The number of responses with “1xx” status codes.

`2xx (integer)`  
    The number of responses with “2xx” status codes.

`3xx (integer)`

The number of responses with “3xx” status codes.  
 4xx (integer)  
 The number of responses with “4xx” status codes.  
 5xx (integer)  
 The number of responses with “5xx” status codes.  
 codes  
 The number of responses per each status code.  
 codeNumber (integer)  
 The number of responses with this particular status code.  
 total (integer)  
 The total number of responses obtained from this server.  
 sent (integer)  
 The total number of bytes sent to this server.  
 received (integer)  
 The total number of bytes received from this server.  
 fails (integer)  
 The total number of unsuccessful attempts to communicate with the server.  
 unavail (integer)  
 How many times the server became unavailable for client requests (state “unavail”) due to the number of unsuccessful attempts reaching the [max\\_fails](#) threshold.  
 health\_checks  
 checks (integer)  
 The total number of [health check](#) requests made.  
 fails (integer)  
 The number of failed health checks.  
 unhealthy (integer)  
 How many times the server became unhealthy (state “unhealthy”).  
 last\_passed (boolean)  
 Boolean indicating if the last health check request was successful and passed [tests](#).  
 downtime (integer)  
 Total time the server was in the “unavail”, “checking”, and “unhealthy” states.  
 downstart (string)  
 The time when the server became “unavail”, “checking”, or “unhealthy”, in the ISO 8601 format with millisecond resolution.  
 selected (string)  
 The time when the server was last selected to process a request, in the ISO 8601 format with millisecond resolution.  
 header\_time (integer)  
 The average time to get the [response header](#) from the server.

`response_time (integer)`

The average time to get the [full response](#) from the server.

`keepalive (integer)`

The current number of idle [keepalive](#) connections.

`zombies (integer)`

The current number of servers removed from the group but still processing active client requests.

`zone (string)`

The name of the shared memory [zone](#) that keeps the group's configuration and run-time state.

`queue`

For the requests [queue](#), the following data are provided:

`size (integer)`

The current number of requests in the queue.

`max_size (integer)`

The maximum number of requests that can be in the queue at the same time.

`overflows (integer)`

The total number of requests rejected due to the queue overflow.

Example:

```
{
  "upstream_backend" : {
    "peers" : [
      {
        "id" : 0,
        "server" : "10.0.0.1:8088",
        "name" : "10.0.0.1:8088",
        "backup" : false,
        "weight" : 5,
        "state" : "up",
        "active" : 0,
        "ssl" : {
          "handshakes" : 620311,
          "handshakes_failed" : 3432,
          "session_reuses" : 36442,
          "no_common_protocol" : 4,
          "handshake_timeout" : 0,
          "peer_rejected_cert" : 0,
          "verify_failures" : {
            "expired_cert" : 2,
            "revoked_cert" : 1,
            "hostname_mismatch" : 2,
            "other" : 1
          }
        }
      },
    ],
    "max_conns" : 20,
    "requests" : 667231,
    "header_time" : 20,
    "response_time" : 36,
    "responses" : {
      "1xx" : 0,
      "2xx" : 666310,
      "3xx" : 0,
      "4xx" : 915,
      "5xx" : 6,
      "codes" : {
```

```

        "200" : 666310,
        "404" : 915,
        "503" : 6
    },
    "total" : 667231
},
"sent" : 251946292,
"received" : 19222475454,
"fails" : 0,
"unavail" : 0,
"health_checks" : {
    "checks" : 26214,
    "fails" : 0,
    "unhealthy" : 0,
    "last_passed" : true
},
"downtime" : 0,
"downstart" : "2022-06-28T11:09:21.602Z",
"selected" : "2022-06-28T15:01:25.000Z"
},
{
    "id" : 1,
    "server" : "10.0.0.1:8089",
    "name" : "10.0.0.1:8089",
    "backup" : true,
    "weight" : 1,
    "state" : "unhealthy",
    "active" : 0,
    "max_conns" : 20,
    "requests" : 0,
    "responses" : {
        "1xx" : 0,
        "2xx" : 0,
        "3xx" : 0,
        "4xx" : 0,
        "5xx" : 0,
        "codes" : {
        },
        "total" : 0
    },
    "sent" : 0,
    "received" : 0,
    "fails" : 0,
    "unavail" : 0,
    "health_checks" : {
        "checks" : 26284,
        "fails" : 26284,
        "unhealthy" : 1,
        "last_passed" : false
    },
    "downtime" : 262925617,
    "downstart" : "2022-06-28T11:09:21.602Z",
    "selected" : "2022-06-28T15:01:25.000Z"
}
],
"keepalive" : 0,
"zombies" : 0,
"zone" : "upstream_backend"
}
}

```

- HTTP Upstream Server:

Dynamically configurable parameters of an HTTP upstream [server](#):

`id (integer)`

The ID of the HTTP upstream server. The ID is assigned automatically and cannot be changed.

`server (string)`

Same as the `address` parameter of the HTTP upstream server. When adding a server, it is possible to specify it as a domain name. In this case, changes of the IP addresses that correspond to a domain name will be monitored and automatically applied to the upstream configuration without the need of restarting nginx. This requires the `resolver` directive in the “http” block. See also the `resolve` parameter of the HTTP upstream server.

`service (string)`

Same as the `service` parameter of the HTTP upstream server. This parameter cannot be changed.

`weight (integer)`

Same as the `weight` parameter of the HTTP upstream server.

`max_conns (integer)`

Same as the `max_conns` parameter of the HTTP upstream server.

`max_fails (integer)`

Same as the `max_fails` parameter of the HTTP upstream server.

`fail_timeout (string)`

Same as the `fail_timeout` parameter of the HTTP upstream server.

`slow_start (string)`

Same as the `slow_start` parameter of the HTTP upstream server.

`route (string)`

Same as the `route` parameter of the HTTP upstream server.

`backup (boolean)`

When true, adds a `backup` server. This parameter cannot be changed.

`down (boolean)`

Same as the `down` parameter of the HTTP upstream server.

`drain (boolean)`

Same as the `drain` parameter of the HTTP upstream server.

`parent (string)`

Parent server ID of the resolved server. The ID is assigned automatically and cannot be changed.

`host (string)`

Hostname of the resolved server. The hostname is assigned automatically and cannot be changed.

Example:

```
{
  "id" : 1,
  "server" : "10.0.0.1:8089",
  "weight" : 4,
  "max_conns" : 0,
  "max_fails" : 0,
  "fail_timeout" : "10s",
```



```

    "slow_start" : "10s",
    "route" : "",
    "backup" : true,
    "down" : true
}

```

- HTTP Keyval Shared Memory Zone:

Contents of an HTTP keyval shared memory zone when using the GET method.

Example:

```

{
  "key1" : "value1",
  "key2" : "value2",
  "key3" : "value3"
}

```

- HTTP Keyval Shared Memory Zone:

Contents of an HTTP keyval shared memory zone when using the POST or PATCH methods.

Example:

```

{
  "key1" : "value1",
  "key2" : "value2",
  "key3" : {
    "value" : "value3",
    "expire" : 30000
  }
}

```

- Stream Server Zone:

processing (integer)

The number of client connections that are currently being processed.

connections (integer)

The total number of connections accepted from clients.

sessions

The total number of completed sessions, and the number of sessions completed with status codes “2xx”, “4xx”, or “5xx”.

2xx (integer)

The total number of sessions completed with [status codes](#) “2xx”.

4xx (integer)

The total number of sessions completed with [status codes](#) “4xx”.

5xx (integer)

The total number of sessions completed with [status codes](#) “5xx”.

`total (integer)`  
 The total number of completed client sessions.

`discarded (integer)`  
 The total number of connections completed without creating a session.

`received (integer)`  
 The total number of bytes received from clients.

`sent (integer)`  
 The total number of bytes sent to clients.

`ssl`

`handshakes (integer)`  
 The total number of successful SSL handshakes.

`handshakes_failed (integer)`  
 The total number of failed SSL handshakes.

`session_reuses (integer)`  
 The total number of session reuses during SSL handshake.

`no_common_protocol (integer)`  
 The number of SSL handshakes failed because of no common protocol.

`no_common_cipher (integer)`  
 The number of SSL handshakes failed because of no shared cipher.

`handshake_timeout (integer)`  
 The number of SSL handshakes failed because of a timeout.

`peer_rejected_cert (integer)`  
 The number of failed SSL handshakes when nginx presented the certificate to the client but it was rejected with a corresponding alert message.

`verify_failures`  
 SSL certificate verification errors

`no_cert (integer)`  
 A client did not provide the required certificate.

`expired_cert (integer)`  
 An expired or not yet valid certificate was presented by a client.

`revoked_cert (integer)`  
 A revoked certificate was presented by a client.

`other (integer)`  
 Other SSL certificate verification errors.

Example:

```
{
  "dns" : {
    "processing" : 1,
    "connections" : 155569,
    "sessions" : {
```

```

        "2xx" : 155564,
        "4xx" : 0,
        "5xx" : 0,
        "total" : 155569
    },
    "discarded" : 0,
    "received" : 4200363,
    "sent" : 20489184,
    "ssl" : {
        "handshakes" : 76455,
        "handshakes_failed" : 432,
        "session_reuses" : 28770,
        "no_common_protocol" : 4,
        "no_common_cipher" : 2,
        "handshake_timeout" : 0,
        "peer_rejected_cert" : 0,
        "verify_failures" : {
            "no_cert" : 0,
            "expired_cert" : 2,
            "revoked_cert" : 1,
            "other" : 1
        }
    }
}
}
}

```

- Stream Connections Limiting:

passed (integer)

The total number of connections that were neither limited nor accounted as limited.

rejected (integer)

The total number of connections that were rejected.

rejected\_dry\_run (integer)

The total number of connections accounted as rejected in the [dry run](#) mode.

Example:

```

{
    "passed" : 15,
    "rejected" : 0,
    "rejected_dry_run" : 2
}

```

- Stream Upstream:

peers

An array of:

id (integer)

The ID of the server.

server (string)

An [address](#) of the server.

service (string)

The [service](#) parameter value of the [server](#) directive.

name (string)

The name of the server specified in the [server](#) directive.

backup (boolean)

A boolean value indicating whether the server is a [backup](#) server.

weight (integer)

[Weight](#) of the server.

state (string)

Current state, which may be one of “up”, “down”, “unavail”, “checking”, or “unhealthy”.

active (integer)

The current number of connections.

ssl

handshakes (integer)

The total number of successful SSL handshakes.

handshakes\_failed (integer)

The total number of failed SSL handshakes.

session\_reuses (integer)

The total number of session reuses during SSL handshake.

no\_common\_protocol (integer)

The number of SSL handshakes failed because of no common protocol.

handshake\_timeout (integer)

The number of SSL handshakes failed because of a timeout.

peer\_rejected\_cert (integer)

The number of failed SSL handshakes when nginx presented the certificate to the upstream server but it was rejected with a corresponding alert message.

verify\_failures

SSL certificate verification errors

expired\_cert (integer)

An expired or not yet valid certificate was presented by an upstream server.

revoked\_cert (integer)

A revoked certificate was presented by an upstream server.

hostname\_mismatch (integer)

Server's certificate doesn't match the hostname.

other (integer)

Other SSL certificate verification errors.

max\_conns (integer)

The [max\\_conns](#) limit for the server.

connections (integer)

The total number of client connections forwarded to this server.

connect\_time (integer)

The average time to connect to the upstream server.  
`first_byte_time` (integer)  
The average time to receive the first byte of data.  
`response_time` (integer)  
The average time to receive the last byte of data.  
`sent` (integer)  
The total number of bytes sent to this server.  
`received` (integer)  
The total number of bytes received from this server.  
`fails` (integer)  
The total number of unsuccessful attempts to communicate with the server.  
`unavail` (integer)  
How many times the server became unavailable for client connections (state “unavail”) due to the number of unsuccessful attempts reaching the `max_fails` threshold.  
`health_checks`  
`checks` (integer)  
The total number of `health check` requests made.  
`fails` (integer)  
The number of failed health checks.  
`unhealthy` (integer)  
How many times the server became unhealthy (state “unhealthy”).  
`last_passed` (boolean)  
Boolean indicating whether the last health check request was successful and passed `tests`.  
`downtime` (integer)  
Total time the server was in the “unavail”, “checking”, and “unhealthy” states.  
`downstart` (string)  
The time when the server became “unavail”, “checking”, or “unhealthy”, in the ISO 8601 format with millisecond resolution.  
`selected` (string)  
The time when the server was last selected to process a connection, in the ISO 8601 format with millisecond resolution.  
`zombies` (integer)  
The current number of servers removed from the group but still processing active client connections.  
`zone` (string)  
The name of the shared memory `zone` that keeps the group’s configuration and run-time state.

Example:

```

{
  "dns" : {
    "peers" : [
      {
        "id" : 0,
        "server" : "10.0.0.1:12347",
        "name" : "10.0.0.1:12347",
        "backup" : false,
        "weight" : 5,
        "state" : "up",
        "active" : 0,
        "ssl" : {
          "handshakes" : 200,
          "handshakes_failed" : 4,
          "session_reuses" : 189,
          "no_common_protocol" : 4,
          "handshake_timeout" : 0,
          "peer_rejected_cert" : 0,
          "verify_failures" : {
            "expired_cert" : 2,
            "revoked_cert" : 1,
            "hostname_mismatch" : 2,
            "other" : 1
          }
        },
        "max_conns" : 50,
        "connections" : 667231,
        "sent" : 251946292,
        "received" : 19222475454,
        "fails" : 0,
        "unavail" : 0,
        "health_checks" : {
          "checks" : 26214,
          "fails" : 0,
          "unhealthy" : 0,
          "last_passed" : true
        },
        "downtime" : 0,
        "downstart" : "2022-06-28T11:09:21.602Z",
        "selected" : "2022-06-28T15:01:25.000Z"
      },
      {
        "id" : 1,
        "server" : "10.0.0.1:12348",
        "name" : "10.0.0.1:12348",
        "backup" : true,
        "weight" : 1,
        "state" : "unhealthy",
        "active" : 0,
        "max_conns" : 50,
        "connections" : 0,
        "sent" : 0,
        "received" : 0,
        "fails" : 0,
        "unavail" : 0,
        "health_checks" : {
          "checks" : 26284,
          "fails" : 26284,
          "unhealthy" : 1,
          "last_passed" : false
        },
        "downtime" : 262925617,
        "downstart" : "2022-06-28T11:09:21.602Z",
        "selected" : "2022-06-28T15:01:25.000Z"
      }
    ],
    "zombies" : 0,
    "zone" : "dns"
  }
}

```

```
}

```

- Stream Upstream Server:

Dynamically configurable parameters of a stream upstream [server](#):

`id (integer)`

The ID of the stream upstream server. The ID is assigned automatically and cannot be changed.

`server (string)`

Same as the [address](#) parameter of the stream upstream server. When adding a server, it is possible to specify it as a domain name. In this case, changes of the IP addresses that correspond to a domain name will be monitored and automatically applied to the upstream configuration without the need of restarting nginx. This requires the [resolver](#) directive in the “stream” block. See also the [resolve](#) parameter of the stream upstream server.

`service (string)`

Same as the [service](#) parameter of the stream upstream server. This parameter cannot be changed.

`weight (integer)`

Same as the [weight](#) parameter of the stream upstream server.

`max_conns (integer)`

Same as the [max\\_conns](#) parameter of the stream upstream server.

`max_fails (integer)`

Same as the [max\\_fails](#) parameter of the stream upstream server.

`fail_timeout (string)`

Same as the [fail\\_timeout](#) parameter of the stream upstream server.

`slow_start (string)`

Same as the [slow\\_start](#) parameter of the stream upstream server.

`backup (boolean)`

When true, adds a [backup](#) server. This parameter cannot be changed.

`down (boolean)`

Same as the [down](#) parameter of the stream upstream server.

`parent (string)`

Parent server ID of the resolved server. The ID is assigned automatically and cannot be changed.

`host (string)`

Hostname of the resolved server. The hostname is assigned automatically and cannot be changed.

Example:

```
{
  "id" : 0,
  "server" : "10.0.0.1:12348",
  "weight" : 1,

```

```

    "max_conns" : 0,
    "max_fails" : 1,
    "fail_timeout" : "10s",
    "slow_start" : 0,
    "backup" : false,
    "down" : false
}

```

- Stream Keyval Shared Memory Zone:

Contents of a stream keyval shared memory zone when using the GET method.

Example:

```

{
  "key1" : "value1",
  "key2" : "value2",
  "key3" : "value3"
}

```

- Stream Keyval Shared Memory Zone:

Contents of a stream keyval shared memory zone when using the POST or PATCH methods.

Example:

```

{
  "key1" : "value1",
  "key2" : "value2",
  "key3" : {
    "value" : "value3",
    "expire" : 30000
  }
}

```

- Stream Zone Sync Node:

zones

Synchronization information per each shared memory zone.

A collection of "[Sync Zone](#)" objects

status

Synchronization information per node in a cluster.

bytes\_in (integer)

The number of bytes received by this node.

msgs\_in (integer)

The number of messages received by this node.

msgs\_out (integer)

The number of messages sent by this node.

bytes\_out (integer)

The number of bytes sent by this node.

nodes\_online (integer)

The number of peers this node is connected to.



Example:

```
{
  "zones" : {
    "zone1" : {
      "records_pending" : 2061,
      "records_total" : 260575
    },
    "zone2" : {
      "records_pending" : 0,
      "records_total" : 14749
    }
  },
  "status" : {
    "bytes_in" : 1364923761,
    "msgs_in" : 337236,
    "msgs_out" : 346717,
    "bytes_out" : 1402765472,
    "nodes_online" : 15
  }
}
```

- Sync Zone:

Synchronization status of a shared memory zone.

`records_pending (integer)`

The number of records that need to be sent to the cluster.

`records_total (integer)`

The total number of records stored in the shared memory zone.

Resolver Zone:

Statistics of DNS requests and responses per particular resolver zone.

- `requests`

`name (integer)`

The total number of requests to resolve names to addresses.

`srv (integer)`

The total number of requests to resolve SRV records.

`addr (integer)`

The total number of requests to resolve addresses to names.

`responses`

`noerror (integer)`

The total number of successful responses.

`formerr (integer)`

The total number of FORMERR (Format error) responses.

`servfail (integer)`

The total number of SERVFAIL (Server failure) responses.

`nxdomain (integer)`

The total number of NXDOMAIN (Host not found) responses.

notimp (integer)

The total number of NOTIMP (Unimplemented) responses.

refused (integer)

The total number of REFUSED (Operation refused) responses.

timedout (integer)

The total number of timed out requests.

unknown (integer)

The total number of requests completed with an unknown error.

Example:

```
{
  "resolver_zone1" : {
    "requests" : {
      "name" : 25460,
      "srv" : 130,
      "addr" : 2580
    },
    "responses" : {
      "noerror" : 26499,
      "formerr" : 0,
      "servfail" : 3,
      "nxdomain" : 0,
      "notimp" : 0,
      "refused" : 0,
      "timedout" : 243,
      "unknown" : 478
    }
  }
}
```

- Error:

nginx error object.

error

status (integer)

HTTP error code.

text (string)

Error description.

code (string)

Internal nginx error code.

request\_id (string)

The ID of the request, equals the value of the [\\$request\\_id](#) variable.

href (string)

Link to reference documentation.

## 2.5 Module ngx\_http\_auth\_basic\_module

2.5.1	Summary	114
2.5.2	Example Configuration	114
2.5.3	Directives	114
	auth_basic	114
	auth_basic_user_file	114

### 2.5.1 Summary

The `ngx_http_auth_basic_module` module allows limiting access to resources by validating the user name and password using the “HTTP Basic Authentication” protocol.

Access can also be limited by [address](#), by the [result of subrequest](#), or by [JWT](#). Simultaneous limitation of access by address and by password is controlled by the [satisfy](#) directive.

### 2.5.2 Example Configuration

```
location / {
    auth_basic          "closed site";
    auth_basic_user_file conf/htpasswd;
}
```

### 2.5.3 Directives

#### auth\_basic

SYNTAX: **auth\_basic** *string* | `off`;  
 DEFAULT `off`  
 CONTEXT: http, server, location, limit\_except

Enables validation of user name and password using the “HTTP Basic Authentication” protocol. The specified parameter is used as a *realm*. Parameter value can contain variables (1.3.10, 1.2.7). The special value `off` cancels the effect of the `auth_basic` directive inherited from the previous configuration level.

#### auth\_basic\_user\_file

SYNTAX: **auth\_basic\_user\_file** *file*;  
 DEFAULT `—`  
 CONTEXT: http, server, location, limit\_except

Specifies a file that keeps user names and passwords, in the following format:

```
# comment
name1:password1
name2:password2:comment
```

```
name3:password3
```

The *file* name can contain variables.

The following password types are supported:

- encrypted with the `crypt` function; can be generated using the “`htpasswd`” utility from the Apache HTTP Server distribution or the “`openssl passwd`” command;
- hashed with the Apache variant of the MD5-based password algorithm (`apr1`); can be generated with the same tools;
- specified by the “`{scheme}data`” syntax (1.0.3+) as described in [RFC 2307](#); currently implemented schemes include `PLAIN` (an example one, should not be used), `SHA` (1.3.13) (plain SHA-1 hashing, should not be used) and `SSHA` (salted SHA-1 hashing, used by some software packages, notably OpenLDAP and Dovecot).

Support for SHA scheme was added only to aid in migration from other web servers. It should not be used for new passwords, since unsalted SHA-1 hashing that it employs is vulnerable to [rainbow table](#) attacks.

## 2.6 Module ngx\_http\_auth\_jwt\_module

2.6.1	Summary	116
2.6.2	Supported Algorithms	116
2.6.3	Example Configuration	117
2.6.4	Directives	117
	auth_jwt	117
	auth_jwt_claim_set	117
	auth_jwt_header_set	118
	auth_jwt_key_cache	118
	auth_jwt_key_file	118
	auth_jwt_key_request	119
	auth_jwt_leeway	119
	auth_jwt_type	120
	auth_jwt_require	120
2.6.5	Embedded Variables	120

### 2.6.1 Summary

The `ngx_http_auth_jwt_module` module (1.11.3) implements client authorization by validating the provided [JSON Web Token](#) (JWT) using the specified keys. The module supports [JSON Web Signature](#) (JWS), [JSON Web Encryption](#) (JWE) (1.19.7), and Nested JWT (1.21.0). The module can be used for [OpenID Connect](#) authentication.

The module may be combined with other access modules, such as `ngx_http_access_module`, `ngx_http_auth_basic_module`, and `ngx_http_auth_request_module`, via the `satisfy` directive.

This module is available as part of our [commercial subscription](#).

### 2.6.2 Supported Algorithms

The module supports the following JSON Web [Algorithms](#).

JWS algorithms:

- HS256, HS384, HS512
- RS256, RS384, RS512
- ES256, ES384, ES512
- EdDSA (Ed25519 and Ed448 signatures) (1.15.7)

Prior to version 1.13.7, only HS256, RS256, ES256 algorithms were supported.

JWE content encryption algorithms (1.19.7):

- A128CBC-HS256, A192CBC-HS384, A256CBC-HS512
- A128GCM, A192GCM, A256GCM

JWE key management algorithms (1.19.9):

- A128KW, A192KW, A256KW
- A128GCMKW, A192GCMKW, A256GCMKW
- dir - direct use of a shared symmetric key as the content encryption key
- RSA-OAEP, RSA-OAEP-256, RSA-OAEP-384, RSA-OAEP-512 (1.21.0)

### 2.6.3 Example Configuration

```
location / {
    auth_jwt          "closed site";
    auth_jwt_key_file conf/keys.json;
}
```

### 2.6.4 Directives

#### auth\_jwt

SYNTAX: **auth\_jwt** *string* [token=*\$variable*] | off;

DEFAULT off

CONTEXT: http, server, location, limit\_except

Enables validation of JSON Web Token. The specified *string* is used as a realm. Parameter value can contain variables.

The optional token parameter specifies a variable that contains JSON Web Token. By default, JWT is passed in the Authorization header as a [Bearer Token](#). JWT may be also passed as a cookie or a part of a query string:

```
auth_jwt "closed site" token=$cookie_auth_token;
```

The special value off cancels the effect of the auth\_jwt directive inherited from the previous configuration level.

#### auth\_jwt\_claim\_set

SYNTAX: **auth\_jwt\_claim\_set** *\$variable name* ...;

DEFAULT —

CONTEXT: http

THIS DIRECTIVE APPEARED IN VERSION 1.11.10.

Sets the *variable* to a JWT claim parameter identified by key names. Name matching starts from the top level of the JSON tree. For arrays, the variable keeps a list of array elements separated by commas.

```
auth_jwt_claim_set $email info e-mail;  
auth_jwt_claim_set $job info "job title";
```

Prior to version 1.13.7, only one key name could be specified, and the result was undefined for arrays.

Variable values for tokens encrypted with JWE are available only after decryption which occurs during the Access phase.

### auth\_jwt\_header\_set

SYNTAX: **auth\_jwt\_header\_set** *\$variable name ...*;

DEFAULT —

CONTEXT: http

THIS DIRECTIVE APPEARED IN VERSION 1.11.10.

Sets the *variable* to a JOSE header parameter identified by key names. Name matching starts from the top level of the JSON tree. For arrays, the variable keeps a list of array elements separated by commas.

Prior to version 1.13.7, only one key name could be specified, and the result was undefined for arrays.

### auth\_jwt\_key\_cache

SYNTAX: **auth\_jwt\_key\_cache** *time*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.21.4.

Enables or disables caching of keys obtained from a [file](#) or from a [subrequest](#), and sets caching time for them. Caching of keys obtained from variables is not supported. By default, caching of keys is disabled.

### auth\_jwt\_key\_file

SYNTAX: **auth\_jwt\_key\_file** *file*;

DEFAULT —

CONTEXT: http, server, location, limit\_except

Specifies a *file* in [JSON Web Key Set](#) format for validating JWT signature. Parameter value can contain variables.

Several `auth_jwt_key_file` directives can be specified on the same level (1.21.1):

```
auth_jwt_key_file conf/keys.json;  
auth_jwt_key_file conf/key.jwk;
```

If at least one of the specified keys cannot be loaded or processed, nginx will return the 500 Internal Server Error error.

### auth\_jwt\_key\_request

SYNTAX: **auth\_jwt\_key\_request** *uri*;

DEFAULT —

CONTEXT: http, server, location, limit\_except

THIS DIRECTIVE APPEARED IN VERSION 1.15.6.

Allows retrieving a [JSON Web Key Set](#) file from a subrequest for validating JWT signature and sets the URI where the subrequest will be sent to. Parameter value can contain variables. To avoid validation overhead, it is recommended to cache the key file:

```
proxy_cache_path /data/nginx/cache levels=1 keys_zone=foo:10m;

server {
    ...

    location / {
        auth_jwt          "closed site";
        auth_jwt_key_request /jwks_uri;
    }

    location = /jwks_uri {
        internal;
        proxy_cache foo;
        proxy_pass http://idp.example.com/keys;
    }
}
```

Several `auth_jwt_key_request` directives can be specified on the same level (1.21.1):

```
auth_jwt_key_request /jwks_uri;
auth_jwt_key_request /jwks2_uri;
```

If at least one of the specified keys cannot be loaded or processed, nginx will return the 500 Internal Server Error error.

### auth\_jwt\_leeway

SYNTAX: **auth\_jwt\_leeway** *time*;

DEFAULT 0s

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.13.10.

Sets the maximum allowable leeway to compensate clock skew when verifying the [exp](#) and [nbf](#) JWT claims.



## auth\_jwt\_type

SYNTAX: **auth\_jwt\_type** *signed* | *encrypted* | *nested*;

DEFAULT *signed*

CONTEXT: http, server, location, limit\_except

THIS DIRECTIVE APPEARED IN VERSION 1.19.7.

Specifies which type of JSON Web Token to expect: JWS (*signed*), JWE (*encrypted*), or signed and then encrypted Nested JWT (*nested*) (1.21.0).

## auth\_jwt\_require

SYNTAX: **auth\_jwt\_require** *\$value* ...[*error=401* | *403*];

DEFAULT —

CONTEXT: http, server, location, limit\_except

THIS DIRECTIVE APPEARED IN VERSION 1.21.2.

Specifies additional checks for JWT validation. The value can contain text, variables, and their combination, and must start with a variable (1.21.7). The authentication will succeed only if all the values are not empty and are not equal to “0”.

```
map $jwt_claim_iss $valid_jwt_iss {
    "good" 1;
}
...

auth_jwt_require $valid_jwt_iss;
```

If any of the checks fails, the 401 error code is returned. The optional error parameter (1.21.7) allows redefining the error code to 403.

### 2.6.5 Embedded Variables

The `ngx_http_auth_jwt_module` module supports embedded variables:

*\$jwt\_header\_name*

returns the value of a specified [JOSE header](#)

*\$jwt\_claim\_name*

returns the value of a specified [JWT claim](#)

For nested claims and claims including a dot (“.”), the value of the variable cannot be evaluated; the [auth\\_jwt\\_claim\\_set](#) directive should be used instead.

Variable values for tokens encrypted with JWE are available only after decryption which occurs during the Access phase.

*\$jwt\_payload*

returns the decrypted top-level payload of *nested* or *encrypted* tokens (1.21.2). For nested tokens returns the enclosed JWS token. For encrypted tokens returns JSON with claims.

## 2.7 Module ngx\_http\_auth\_request\_module

2.7.1	Summary	121
2.7.2	Example Configuration	121
2.7.3	Directives	121
	<a href="#">auth_request</a>	121
	<a href="#">auth_request_set</a>	122

### 2.7.1 Summary

The `ngx_http_auth_request_module` (1.5.4+) implements client authorization based on the result of a subrequest. If the subrequest returns a 2xx response code, the access is allowed. If it returns 401 or 403, the access is denied with the corresponding error code. Any other response code returned by the subrequest is considered an error.

For the 401 error, the client also receives the `WWW-Authenticate` header from the subrequest response.

This module is not built by default, it should be enabled with the `--with-http_auth_request_module` configuration parameter.

The module may be combined with other access modules, such as [ngx-http-access-module](#), [ngx-http-auth-basic-module](#), and [ngx-http-auth-jwt-module](#), via the [satisfy](#) directive.

Before version 1.7.3, responses to authorization subrequests could not be cached (using [proxy-cache](#), [proxy-store](#), etc.).

### 2.7.2 Example Configuration

```
location /private/ {
    auth_request /auth;
    ...
}

location = /auth {
    proxy_pass ...
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";
    proxy_set_header X-Original-URI $request_uri;
}
```

### 2.7.3 Directives

#### `auth_request`

SYNTAX: **auth\_request** *uri* | `off`;

DEFAULT `off`

CONTEXT: http, server, location

Enables authorization based on the result of a subrequest and sets the URI to which the subrequest will be sent.

### **auth\_request\_set**

SYNTAX: **auth\_request\_set** *\$variable value*;

DEFAULT —

CONTEXT: http, server, location

Sets the request *variable* to the given *value* after the authorization request completes. The value may contain variables from the authorization request, such as *\$upstream\_http\_\**.

## 2.8 Module ngx\_http\_autoindex\_module

2.8.1	Summary	123
2.8.2	Example Configuration	123
2.8.3	Directives	123
	autoindex	123
	autoindex_exact_size	123
	autoindex_format	123
	autoindex_localtime	124

### 2.8.1 Summary

The `ngx_http_autoindex_module` module processes requests ending with the slash character (`/`) and produces a directory listing. Usually a request is passed to the `ngx_http_autoindex_module` module when the `ngx_http_index_module` module cannot find an index file.

### 2.8.2 Example Configuration

```
location / {
    autoindex on;
}
```

### 2.8.3 Directives

#### autoindex

SYNTAX: **autoindex** on | off;

DEFAULT off

CONTEXT: http, server, location

Enables or disables the directory listing output.

#### autoindex\_exact\_size

SYNTAX: **autoindex\_exact\_size** on | off;

DEFAULT on

CONTEXT: http, server, location

For the HTML `format`, specifies whether exact file sizes should be output in the directory listing, or rather rounded to kilobytes, megabytes, and gigabytes.

#### autoindex\_format

SYNTAX: **autoindex\_format** html | xml | json | jsonp;

DEFAULT html

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.9.

Sets the format of a directory listing.

When the JSONP format is used, the name of a callback function is set with the `callback` request argument. If the argument is missing or has an empty value, then the JSON format is used.

The XML output can be transformed using the [ngx\\_http\\_xslt\\_module](#) module.

### **autoindex\_localtime**

SYNTAX: **autoindex\_localtime** on | off;

DEFAULT off

CONTEXT: http, server, location

For the HTML [format](#), specifies whether times in the directory listing should be output in the local time zone or UTC.

## 2.9 Module ngx\_http\_browser\_module

2.9.1	Summary	125
2.9.2	Example Configuration	125
2.9.3	Directives	126
	ancient_browser	126
	ancient_browser_value	126
	modern_browser	126
	modern_browser_value	126

### 2.9.1 Summary

The `ngx_http_browser_module` module creates variables whose values depend on the value of the User-Agent request header field:

*\$modern\_browser*

equals the value set by the `modern_browser_value` directive, if a browser was identified as modern;

*\$ancient\_browser*

equals the value set by the `ancient_browser_value` directive, if a browser was identified as ancient;

*\$msie*

equals “1” if a browser was identified as MSIE of any version.

### 2.9.2 Example Configuration

Choosing an index file:

```
modern_browser_value "modern.";

modern_browser msie      5.5;
modern_browser gecko     1.0.0;
modern_browser opera     9.0;
modern_browser safari    413;
modern_browser konqueror 3.0;

index index.${modern_browser}html index.html;
```

Redirection for old browsers:

```
modern_browser msie      5.0;
modern_browser gecko     0.9.1;
modern_browser opera     8.0;
modern_browser safari    413;
modern_browser konqueror 3.0;

modern_browser unlisted;

ancient_browser Links Lynx netscape4;

if ($ancient_browser) {
    rewrite ^ /ancient.html;
}
```

## 2.9.3 Directives

### ancient\_browser

SYNTAX: **ancient\_browser** *string* ...;

DEFAULT —

CONTEXT: http, server, location

If any of the specified substrings is found in the User-Agent request header field, the browser will be considered ancient. The special string “netscape4” corresponds to the regular expression “^Mozilla/[1-4]”.

### ancient\_browser\_value

SYNTAX: **ancient\_browser\_value** *string*;

DEFAULT 1

CONTEXT: http, server, location

Sets a value for the *\$ancient\_browser* variables.

### modern\_browser

SYNTAX: **modern\_browser** *browser version*;

SYNTAX: **modern\_browser** *unlisted*;

DEFAULT —

CONTEXT: http, server, location

Specifies a version starting from which a browser is considered modern. A browser can be any one of the following: msie, gecko (browsers based on Mozilla), opera, safari, or konqueror.

Versions can be specified in the following formats: X, X.X, X.X.X, or X.X.X.X. The maximum values for each of the format are 4000, 4000.99, 4000.99.99, and 4000.99.99.99, respectively.

The special value *unlisted* specifies to consider a browser as modern if it was not listed by the *modern\_browser* and [ancient\\_browser](#) directives. Otherwise such a browser is considered ancient. If a request does not provide the User-Agent field in the header, the browser is treated as not being listed.

### modern\_browser\_value

SYNTAX: **modern\_browser\_value** *string*;

DEFAULT 1

CONTEXT: http, server, location

Sets a value for the *\$modern\_browser* variables.

## 2.10 Module ngx\_http\_charset\_module

2.10.1 Summary	127
2.10.2 Example Configuration	127
2.10.3 Directives	127
charset	127
charset_map	128
charset_types	129
override_charset	129
source_charset	129

### 2.10.1 Summary

The `ngx_http_charset_module` module adds the specified charset to the Content-Type response header field. In addition, the module can convert data from one charset to another, with some limitations:

- conversion is performed one way — from server to client,
- only single-byte charsets can be converted
- or single-byte charsets to/from UTF-8.

### 2.10.2 Example Configuration

```
include      conf/koi-win;

charset      windows-1251;
source_charset koi8-r;
```

### 2.10.3 Directives

#### charset

SYNTAX: **charset** *charset* | `off`;

DEFAULT `off`

CONTEXT: http, server, location, if in location

Adds the specified charset to the Content-Type response header field. If this charset is different from the charset specified in the [source\\_charset](#) directive, a conversion is performed.

The parameter `off` cancels the addition of charset to the Content-Type response header field.

A charset can be defined with a variable:

```
charset $charset;
```



In such a case, all possible values of a variable need to be present in the configuration at least once in the form of the [charset\\_map](#), [charset](#), or [source\\_charset](#) directives. For `utf-8`, `windows-1251`, and `koi8-r` charsets, it is sufficient to include the files `conf/koi-win`, `conf/koi-utf`, and `conf/win-utf` into configuration. For other charsets, simply making a fictitious conversion table works, for example:

```
charset_map iso-8859-5 _ { }
```

In addition, a charset can be set in the `X-Accel-Charset` response header field. This capability can be disabled using the [proxy\\_ignore\\_headers](#), [fastcgi\\_ignore\\_headers](#), [uwsgi\\_ignore\\_headers](#), [scgi\\_ignore\\_headers](#), and [grpc\\_ignore\\_headers](#) directives.

### charset\_map

SYNTAX: **charset\_map** *charset1 charset2* { ... }

DEFAULT —

CONTEXT: http

Describes the conversion table from one charset to another. A reverse conversion table is built using the same data. Character codes are given in hexadecimal. Missing characters in the range 80-FF are replaced with “?”. When converting from UTF-8, characters missing in a one-byte charset are replaced with “&#XXXX;”.

Example:

```
charset_map koi8-r windows-1251 {  
    C0 FE ; # small yu  
    C1 E0 ; # small a  
    C2 E1 ; # small b  
    C3 F6 ; # small ts  
    ...  
}
```

When describing a conversion table to UTF-8, codes for the UTF-8 charset should be given in the second column, for example:

```
charset_map koi8-r utf-8 {  
    C0 D18E ; # small yu  
    C1 D0B0 ; # small a  
    C2 D0B1 ; # small b  
    C3 D186 ; # small ts  
    ...  
}
```

Full conversion tables from `koi8-r` to `windows-1251`, and from `koi8-r` and `windows-1251` to `utf-8` are provided in the distribution files `conf/koi-win`, `conf/koi-utf`, and `conf/win-utf`.

## charset\_types

SYNTAX: **charset\_types** *mime-type* ...;  
DEFAULT text/html text/xml text/plain text/vnd.wap.wml  
application/javascript application/rss+xml  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.7.9.

Enables module processing in responses with the specified MIME types in addition to “text/html”. The special value “\*” matches any MIME type (0.8.29).

Until version 1.5.4, “application/x-javascript” was used as the default MIME type instead of “application/javascript”.

## override\_charset

SYNTAX: **override\_charset** on | off;  
DEFAULT off  
CONTEXT: http, server, location, if in location

Determines whether a conversion should be performed for answers received from a proxied or a FastCGI/uwsgi/SCGI/gRPC server when the answers already carry a charset in the Content-Type response header field. If conversion is enabled, a charset specified in the received response is used as a source charset.

It should be noted that if a response is received in a subrequest then the conversion from the response charset to the main request charset is always performed, regardless of the `override_charset` directive setting.

## source\_charset

SYNTAX: **source\_charset** *charset*;  
DEFAULT —  
CONTEXT: http, server, location, if in location

Defines the source charset of a response. If this charset is different from the charset specified in the [charset](#) directive, a conversion is performed.

## 2.11 Module ngx\_http\_dav\_module

2.11.1 Summary	130
2.11.2 Example Configuration	130
2.11.3 Directives	130
create_full_put_path	130
dav_access	131
dav_methods	131
min_delete_depth	131

### 2.11.1 Summary

The `ngx_http_dav_module` module is intended for file management automation via the WebDAV protocol. The module processes HTTP and WebDAV methods PUT, DELETE, MKCOL, COPY, and MOVE.

This module is not built by default, it should be enabled with the `--with-http_dav_module` configuration parameter.

WebDAV clients that require additional WebDAV methods to operate will not work with this module.

### 2.11.2 Example Configuration

```
location / {
    root                /data/www;

    client_body_temp_path /data/client_temp;

    dav_methods PUT DELETE MKCOL COPY MOVE;

    create_full_put_path on;
    dav_access          group:rw all:r;

    limit_except GET {
        allow 192.168.1.0/32;
        deny  all;
    }
}
```

### 2.11.3 Directives

#### create\_full\_put\_path

SYNTAX: **create\_full\_put\_path** on | off;

DEFAULT off

CONTEXT: http, server, location

The WebDAV specification only allows creating files in already existing directories. This directive allows creating all needed intermediate directories.

## dav\_access

SYNTAX: **dav\_access** *users:permissions ...*;

DEFAULT `user:rw`

CONTEXT: http, server, location

Sets access permissions for newly created files and directories, e.g.:

```
dav_access user:rw group:rw all:r;
```

If any group or all access permissions are specified then user permissions may be omitted:

```
dav_access group:rw all:r;
```

## dav\_methods

SYNTAX: **dav\_methods** `off` | *method ...*;

DEFAULT `off`

CONTEXT: http, server, location

Allows the specified HTTP and WebDAV methods. The parameter `off` denies all methods processed by this module. The following methods are supported: PUT, DELETE, MKCOL, COPY, and MOVE.

A file uploaded with the PUT method is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the persistent store can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both saved files and a directory holding temporary files, set by the [client\\_body\\_temp\\_path](#) directive, are put on the same file system.

When creating a file with the PUT method, it is possible to specify the modification date by passing it in the Date header field.

## min\_delete\_depth

SYNTAX: **min\_delete\_depth** *number*;

DEFAULT `0`

CONTEXT: http, server, location

Allows the DELETE method to remove files provided that the number of elements in a request path is not less than the specified number. For example, the directive

```
min_delete_depth 4;
```

allows removing files on requests

```
/users/00/00/name  
/users/00/00/name/pic.jpg
```

```
/users/00/00/page.html
```

and denies the removal of

```
/users/00/00
```

## 2.12 Module ngx\_http\_empty\_gif\_module

2.12.1 Summary	133
2.12.2 Example Configuration	133
2.12.3 Directives	133
empty_gif	133

### 2.12.1 Summary

The ngx\_http\_empty\_gif\_module module emits single-pixel transparent GIF.

### 2.12.2 Example Configuration

```
location = /_.gif {  
    empty_gif;  
}
```

### 2.12.3 Directives

#### empty\_gif

SYNTAX: **empty\_gif**;

DEFAULT —

CONTEXT: location

Turns on module processing in a surrounding location.

## 2.13 Module ngx\_http\_f4f\_module

2.13.1 Summary	134
2.13.2 Example Configuration	134
2.13.3 Directives	134
f4f	134
f4f_buffer_size	134

### 2.13.1 Summary

The `ngx_http_f4f_module` module provides server-side support for Adobe HTTP Dynamic Streaming (HDS).

This module implements handling of HTTP Dynamic Streaming requests in the “/videoSeg1-Frag1” form — extracting the needed fragment from the `videoSeg1.f4f` file using the `videoSeg1.f4x` index file. This module is an alternative to the Adobe’s `f4f` module (HTTP Origin Module) for Apache.

Usual pre-processing with Adobe’s `f4fpackager` is required, see relevant documentation for details.

This module is available as part of our [commercial subscription](#).

### 2.13.2 Example Configuration

```
location /video/ {
    f4f;
    ...
}
```

### 2.13.3 Directives

#### f4f

SYNTAX: **f4f**;

DEFAULT —

CONTEXT: location

Turns on module processing in the surrounding location.

#### f4f\_buffer\_size

SYNTAX: **f4f\_buffer\_size** *size*;

DEFAULT 512k

CONTEXT: http, server, location

Sets the *size* of the buffer used for reading the `.f4x` index file.

## 2.14 Module ngx\_http\_fastcgi\_module

2.14.1	Summary	136
2.14.2	Example Configuration	136
2.14.3	Directives	136
	fastcgi_bind	136
	fastcgi_buffer_size	137
	fastcgi_buffering	137
	fastcgi_buffers	137
	fastcgi_busy_buffers_size	138
	fastcgi_cache	138
	fastcgi_cache_background_update	138
	fastcgi_cache_bypass	138
	fastcgi_cache_key	139
	fastcgi_cache_lock	139
	fastcgi_cache_lock_age	139
	fastcgi_cache_lock_timeout	139
	fastcgi_cache_max_range_offset	140
	fastcgi_cache_methods	140
	fastcgi_cache_min_uses	140
	fastcgi_cache_path	140
	fastcgi_cache_purge	142
	fastcgi_cache_revalidate	143
	fastcgi_cache_use_stale	143
	fastcgi_cache_valid	144
	fastcgi_catch_stderr	144
	fastcgi_connect_timeout	145
	fastcgi_force_ranges	145
	fastcgi_hide_header	145
	fastcgi_ignore_client_abort	145
	fastcgi_ignore_headers	146
	fastcgi_index	146
	fastcgi_intercept_errors	146
	fastcgi_keep_conn	147
	fastcgi_limit_rate	147
	fastcgi_max_temp_file_size	147
	fastcgi_next_upstream	147
	fastcgi_next_upstream_timeout	148
	fastcgi_next_upstream_tries	149
	fastcgi_no_cache	149
	fastcgi_param	149
	fastcgi_pass	150
	fastcgi_pass_header	150
	fastcgi_pass_request_body	150
	fastcgi_pass_request_headers	151
	fastcgi_read_timeout	151



<a href="#">fastcgi_request_buffering</a>	151
<a href="#">fastcgi_send_lowat</a>	151
<a href="#">fastcgi_send_timeout</a>	152
<a href="#">fastcgi_socket_keepalive</a>	152
<a href="#">fastcgi_split_path_info</a>	152
<a href="#">fastcgi_store</a>	152
<a href="#">fastcgi_store_access</a>	153
<a href="#">fastcgi_temp_file_write_size</a>	154
<a href="#">fastcgi_temp_path</a>	154
<a href="#">2.14.4 Parameters Passed to a FastCGI Server</a>	154
<a href="#">2.14.5 Embedded Variables</a>	154

### 2.14.1 Summary

The `ngx_http_fastcgi_module` module allows passing requests to a FastCGI server.

### 2.14.2 Example Configuration

```
location / {
    fastcgi_pass    localhost:9000;
    fastcgi_index   index.php;

    fastcgi_param   SCRIPT_FILENAME  /home/www/scripts/php$fastcgi_script_name;
    fastcgi_param   QUERY_STRING     $query_string;
    fastcgi_param   REQUEST_METHOD   $request_method;
    fastcgi_param   CONTENT_TYPE     $content_type;
    fastcgi_param   CONTENT_LENGTH   $content_length;
}
```

### 2.14.3 Directives

#### **fastcgi\_bind**

SYNTAX: **fastcgi\_bind** *address* [transparent] | off;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.8.22.

Makes outgoing connections to a FastCGI server originate from the specified local IP address with an optional port (1.11.2). Parameter value can contain variables (1.3.12). The special value `off` (1.3.12) cancels the effect of the `fastcgi_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address and port.

The `transparent` parameter (1.11.0) allows outgoing connections to a FastCGI server originate from a non-local IP address, for example, from a real IP address of a client:

```
fastcgi_bind $remote_addr transparent;
```

In order for this parameter to work, it is usually necessary to run nginx worker processes with the [superuser](#) privileges. On Linux it is not required (1.13.8) as if the [transparent](#) parameter is specified, worker processes inherit the `CAP_NET_RAW` capability from the master process. It is also necessary to configure kernel routing table to intercept network traffic from the FastCGI server.

### **fastcgi\_buffer\_size**

SYNTAX: **fastcgi\_buffer\_size** *size*;

DEFAULT 4k | 8k

CONTEXT: http, server, location

Sets the *size* of the buffer used for reading the first part of the response received from the FastCGI server. This part usually contains a small response header. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform. It can be made smaller, however.

### **fastcgi\_buffering**

SYNTAX: **fastcgi\_buffering** on | off;

DEFAULT on

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.6.

Enables or disables buffering of responses from the FastCGI server.

When buffering is enabled, nginx receives a response from the FastCGI server as soon as possible, saving it into the buffers set by the [fastcgi\\_buffer\\_size](#) and [fastcgi\\_buffers](#) directives. If the whole response does not fit into memory, a part of it can be saved to a [temporary file](#) on the disk. Writing to temporary files is controlled by the [fastcgi\\_max\\_temp\\_file\\_size](#) and [fastcgi\\_temp\\_file\\_write\\_size](#) directives.

When buffering is disabled, the response is passed to a client synchronously, immediately as it is received. nginx will not try to read the whole response from the FastCGI server. The maximum size of the data that nginx can receive from the server at a time is set by the [fastcgi\\_buffer\\_size](#) directive.

Buffering can also be enabled or disabled by passing “yes” or “no” in the `X-Accel-Buffering` response header field. This capability can be disabled using the [fastcgi\\_ignore\\_headers](#) directive.

### **fastcgi\_buffers**

SYNTAX: **fastcgi\_buffers** *number size*;

DEFAULT 8 4k | 8k

CONTEXT: http, server, location

Sets the *number* and *size* of the buffers used for reading a response from the FastCGI server, for a single connection. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

## fastcgi\_busy\_buffers\_size

SYNTAX: **fastcgi\_busy\_buffers\_size** *size*;

DEFAULT 8k | 16k

CONTEXT: http, server, location

When [buffering](#) of responses from the FastCGI server is enabled, limits the total *size* of buffers that can be busy sending a response to the client while the response is not yet fully read. In the meantime, the rest of the buffers can be used for reading the response and, if needed, buffering part of the response to a temporary file. By default, *size* is limited by the size of two buffers set by the [fastcgi\\_buffer\\_size](#) and [fastcgi\\_buffers](#) directives.

## fastcgi\_cache

SYNTAX: **fastcgi\_cache** *zone* | off;

DEFAULT off

CONTEXT: http, server, location

Defines a shared memory zone used for caching. The same zone can be used in several places. Parameter value can contain variables (1.7.9). The `off` parameter disables caching inherited from the previous configuration level.

## fastcgi\_cache\_background\_update

SYNTAX: **fastcgi\_cache\_background\_update** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.11.10.

Allows starting a background subrequest to update an expired cache item, while a stale cached response is returned to the client. Note that it is necessary to allow the usage of a stale cached response when it is being updated.

## fastcgi\_cache\_bypass

SYNTAX: **fastcgi\_cache\_bypass** *string* ...;

DEFAULT —

CONTEXT: http, server, location

Defines conditions under which the response will not be taken from a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be taken from the cache:

```
fastcgi_cache_bypass $cookie_nocache $arg_nocache$arg_comment;  
fastcgi_cache_bypass $http_pragma $http_authorization;
```

Can be used along with the [fastcgi\\_no\\_cache](#) directive.

## fastcgi\_cache\_key

SYNTAX: **fastcgi\_cache\_key** *string*;

DEFAULT —

CONTEXT: http, server, location

Defines a key for caching, for example

```
fastcgi_cache_key localhost:9000$request_uri;
```

## fastcgi\_cache\_lock

SYNTAX: **fastcgi\_cache\_lock** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.12.

When enabled, only one request at a time will be allowed to populate a new cache element identified according to the [fastcgi\\_cache\\_key](#) directive by passing a request to a FastCGI server. Other requests of the same cache element will either wait for a response to appear in the cache or the cache lock for this element to be released, up to the time set by the [fastcgi\\_cache\\_lock\\_timeout](#) directive.

## fastcgi\_cache\_lock\_age

SYNTAX: **fastcgi\_cache\_lock\_age** *time*;

DEFAULT 5s

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.8.

If the last request passed to the FastCGI server for populating a new cache element has not completed for the specified *time*, one more request may be passed to the FastCGI server.

## fastcgi\_cache\_lock\_timeout

SYNTAX: **fastcgi\_cache\_lock\_timeout** *time*;

DEFAULT 5s

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.12.

Sets a timeout for [fastcgi\\_cache\\_lock](#). When the *time* expires, the request will be passed to the FastCGI server, however, the response will not be cached.

Before 1.7.8, the response could be cached.

### **fastcgi\_cache\_max\_range\_offset**

SYNTAX: **fastcgi\_cache\_max\_range\_offset** *number*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.11.6.

Sets an offset in bytes for byte-range requests. If the range is beyond the offset, the range request will be passed to the FastCGI server and the response will not be cached.

### **fastcgi\_cache\_methods**

SYNTAX: **fastcgi\_cache\_methods** GET | HEAD | POST ...;

DEFAULT GET HEAD

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.7.59.

If the client request method is listed in this directive then the response will be cached. “GET” and “HEAD” methods are always added to the list, though it is recommended to specify them explicitly. See also the [fastcgi\\_no\\_cache](#) directive.

### **fastcgi\_cache\_min\_uses**

SYNTAX: **fastcgi\_cache\_min\_uses** *number*;

DEFAULT 1

CONTEXT: http, server, location

Sets the *number* of requests after which the response will be cached.

### **fastcgi\_cache\_path**

SYNTAX: **fastcgi\_cache\_path** *path* [levels=*levels*]  
[use\_temp\_path=on|off] keys\_zone=*name:size* [inactive=*time*]  
[max\_size=*size*] [min\_free=*size*] [manager\_files=*number*]  
[manager\_sleep=*time*] [manager\_threshold=*time*]  
[loader\_files=*number*] [loader\_sleep=*time*]  
[loader\_threshold=*time*] [purger=on|off]  
[purger\_files=*number*] [purger\_sleep=*time*]  
[purger\_threshold=*time*];

DEFAULT —

CONTEXT: http

Sets the path and other parameters of a cache. Cache data are stored in files. Both the key and file name in a cache are a result of applying the MD5 function to the proxied URL.

The *levels* parameter defines hierarchy levels of a cache: from 1 to 3, each level accepts values 1 or 2. For example, in the following configuration

```
fastcgi_cache_path /data/nginx/cache levels=1:2 keys_zone=one:10m;
```

file names in a cache will look like this:

```
/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

A cached response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the cache can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both cache and a directory holding temporary files are put on the same file system. A directory for temporary files is set based on the `use_temp_path` parameter (1.7.10). If this parameter is omitted or set to the value `on`, the directory set by the `fastcgi_temp_path` directive for the given location will be used. If the value is set to `off`, temporary files will be put directly in the cache directory.

In addition, all active keys and information about data are stored in a shared memory zone, whose *name* and *size* are configured by the `keys_zone` parameter. One megabyte zone can store about 8 thousand keys.

As part of [commercial subscription](#), the shared memory zone also stores extended cache [information](#), thus, it is required to specify a larger zone size for the same number of keys. For example, one megabyte zone can store about 4 thousand keys.

Cached data that are not accessed during the time specified by the `inactive` parameter get removed from the cache regardless of their freshness. By default, `inactive` is set to 10 minutes.

The special “cache manager” process monitors the maximum cache size set by the `max_size` parameter, and the minimum amount of free space set by the `min_free` (1.19.1) parameter on the file system with cache. When the size is exceeded or there is not enough free space, it removes the least recently used data. The data is removed in iterations configured by `manager_files`, `manager_threshold`, and `manager_sleep` parameters (1.11.5). During one iteration no more than `manager_files` items are deleted (by default, 100). The duration of one iteration is limited by the `manager_threshold` parameter (by default, 200 milliseconds). Between iterations, a pause configured by the `manager_sleep` parameter (by default, 50 milliseconds) is made.

A minute after the start the special “cache loader” process is activated. It loads information about previously cached data stored on file system into a cache zone. The loading is also done in iterations. During one iteration no more than `loader_files` items are loaded (by default, 100). Besides, the duration of one iteration is limited by the `loader_threshold` parameter (by default, 200 milliseconds). Between iterations, a pause configured by the `loader_sleep` parameter (by default, 50 milliseconds) is made.

Additionally, the following parameters are available as part of our [commercial subscription](#):

`purger=on|off`

Instructs whether cache entries that match a [wildcard key](#) will be removed from the disk by the cache purger (1.7.12). Setting the parameter to `on` (default is `off`) will activate the “cache purger” process that permanently iterates through all cache entries and deletes the entries that match the wildcard key.

`purger_files=number`

Sets the number of items that will be scanned during one iteration (1.7.12). By default, `purger_files` is set to 10.

`purger_threshold=number`

Sets the duration of one iteration (1.7.12). By default, `purger_threshold` is set to 50 milliseconds.

`purger_sleep=number`

Sets a pause between iterations (1.7.12). By default, `purger_sleep` is set to 50 milliseconds.

In versions 1.7.3, 1.7.7, and 1.11.10 cache header format has been changed. Previously cached responses will be considered invalid after upgrading to a newer nginx version.

## fastcgi\_cache\_purge

SYNTAX: **fastcgi\_cache\_purge**string ...;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Defines conditions under which the request will be considered a cache purge request. If at least one value of the string parameters is not empty and is not equal to “0” then the cache entry with a corresponding [cache key](#) is removed. The result of successful operation is indicated by returning the 204 No Content response.

If the [cache key](#) of a purge request ends with an asterisk (“\*”), all cache entries matching the wildcard key will be removed from the cache. However, these entries will remain on the disk until they are deleted for either [inactivity](#), or processed by the [cache purger](#) (1.7.12), or a client attempts to access them.

Example configuration:

```
fastcgi_cache_path /data/nginx/cache keys_zone=cache_zone:10m;

map $request_method $purge_method {
    PURGE    1;
    default  0;
}

server {
    ...
}
```

```
location / {
    fastcgi_pass            backend;
    fastcgi_cache           cache_zone;
    fastcgi_cache_key       $uri;
    fastcgi_cache_purge     $purge_method;
}
}
```

This functionality is available as part of our [commercial subscription](#).

### **fastcgi\_cache\_revalidate**

SYNTAX: **fastcgi\_cache\_revalidate** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Enables revalidation of expired cache items using conditional requests with the If-Modified-Since and If-None-Match header fields.

### **fastcgi\_cache\_use\_stale**

SYNTAX: **fastcgi\_cache\_use\_stale** error | timeout | invalid\_header  
| updating | http\_500 | http\_503 | http\_403 | http\_404 |  
http\_429 | off ...;

DEFAULT off

CONTEXT: http, server, location

Determines in which cases a stale cached response can be used when an error occurs during communication with the FastCGI server. The directive's parameters match the parameters of the [fastcgi\\_next\\_upstream](#) directive.

The `error` parameter also permits using a stale cached response if a FastCGI server to process a request cannot be selected.

Additionally, the `updating` parameter permits using a stale cached response if it is currently being updated. This allows minimizing the number of accesses to FastCGI servers when updating cached data.

Using a stale cached response can also be enabled directly in the response header for a specified number of seconds after the response became stale (1.11.10). This has lower priority than using the directive parameters.

- The “[stale-while-revalidate](#)” extension of the Cache-Control header field permits using a stale cached response if it is currently being updated.
- The “[stale-if-error](#)” extension of the Cache-Control header field permits using a stale cached response in case of an error.

To minimize the number of accesses to FastCGI servers when populating a new cache element, the [fastcgi\\_cache\\_lock](#) directive can be used.



## fastcgi\_cache\_valid

SYNTAX: **fastcgi\_cache\_valid** [*code ...*] *time*;

DEFAULT —

CONTEXT: http, server, location

Sets caching time for different response codes. For example, the following directives

```
fastcgi_cache_valid 200 302 10m;  
fastcgi_cache_valid 404 1m;
```

set 10 minutes of caching for responses with codes 200 and 302 and 1 minute for responses with code 404.

If only caching *time* is specified

```
fastcgi_cache_valid 5m;
```

then only 200, 301, and 302 responses are cached.

In addition, the any parameter can be specified to cache any responses:

```
fastcgi_cache_valid 200 302 10m;  
fastcgi_cache_valid 301 1h;  
fastcgi_cache_valid any 1m;
```

Parameters of caching can also be set directly in the response header. This has higher priority than setting of caching time using the directive.

- The X-Accel-Expires header field sets caching time of a response in seconds. The zero value disables caching for a response. If the value starts with the @ prefix, it sets an absolute time in seconds since Epoch, up to which the response may be cached.
- If the header does not include the X-Accel-Expires field, parameters of caching may be set in the header fields Expires or Cache-Control.
- If the header includes the Set-Cookie field, such a response will not be cached.
- If the header includes the Vary field with the special value “\*”, such a response will not be cached (1.7.7). If the header includes the Vary field with another value, such a response will be cached taking into account the corresponding request header fields (1.7.7).

Processing of one or more of these response header fields can be disabled using the [fastcgi\\_ignore\\_headers](#) directive.

## fastcgi\_catch\_stderr

SYNTAX: **fastcgi\_catch\_stderr** *string*;

DEFAULT —

CONTEXT: http, server, location

Sets a string to search for in the error stream of a response received from a FastCGI server. If the *string* is found then it is considered that the FastCGI server has returned an [invalid response](#). This allows handling application errors in nginx, for example:

```
location /php/ {
    fastcgi_pass backend:9000;
    ...
    fastcgi_catch_stderr "PHP Fatal error";
    fastcgi_next_upstream error timeout invalid_header;
}
```

### **fastcgi\_connect\_timeout**

SYNTAX: **fastcgi\_connect\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for establishing a connection with a FastCGI server. It should be noted that this timeout cannot usually exceed 75 seconds.

### **fastcgi\_force\_ranges**

SYNTAX: **fastcgi\_force\_ranges** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.7.

Enables byte-range support for both cached and uncached responses from the FastCGI server regardless of the `Accept-Ranges` field in these responses.

### **fastcgi\_hide\_header**

SYNTAX: **fastcgi\_hide\_header** *field*;

DEFAULT —

CONTEXT: http, server, location

By default, nginx does not pass the header fields `Status` and `X-Accel-...` from the response of a FastCGI server to a client. The `fastcgi_hide_header` directive sets additional fields that will not be passed. If, on the contrary, the passing of fields needs to be permitted, the [fastcgi\\_pass\\_header](#) directive can be used.

### **fastcgi\_ignore\_client\_abort**

SYNTAX: **fastcgi\_ignore\_client\_abort** on | off;

DEFAULT off

CONTEXT: http, server, location

Determines whether the connection with a FastCGI server should be closed when a client closes the connection without waiting for a response.

## fastcgi\_ignore\_headers

SYNTAX: **fastcgi\_ignore\_headers** *field* ...;

DEFAULT —

CONTEXT: http, server, location

Disables processing of certain response header fields from the FastCGI server. The following fields can be ignored: X-Accel-Redirect, X-Accel-Expires, X-Accel-Limit-Rate (1.1.6), X-Accel-Buffering (1.1.6), X-Accel-Charset (1.1.6), Expires, Cache-Control, Set-Cookie (0.8.44), and Vary (1.7.7).

If not disabled, processing of these header fields has the following effect:

- X-Accel-Expires, Expires, Cache-Control, Set-Cookie, and Vary set the parameters of response [caching](#);
- X-Accel-Redirect performs an [internal redirect](#) to the specified URI;
- X-Accel-Limit-Rate sets the [rate limit](#) for transmission of a response to a client;
- X-Accel-Buffering enables or disables [buffering](#) of a response;
- X-Accel-Charset sets the desired [charset](#) of a response.

## fastcgi\_index

SYNTAX: **fastcgi\_index** *name*;

DEFAULT —

CONTEXT: http, server, location

Sets a file name that will be appended after a URI that ends with a slash, in the value of the `$fastcgi_script_name` variable. For example, with these settings

```
fastcgi_index index.php;  
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
```

and the `/page.php` request, the `SCRIPT_FILENAME` parameter will be equal to `/home/www/scripts/php/page.php`, and with the `/` request it will be equal to `/home/www/scripts/php/index.php`.

## fastcgi\_intercept\_errors

SYNTAX: **fastcgi\_intercept\_errors** on | off;

DEFAULT off

CONTEXT: http, server, location

Determines whether FastCGI server responses with codes greater than or equal to 300 should be passed to a client or be intercepted and redirected to nginx for processing with the [error\\_page](#) directive.

## fastcgi\_keep\_conn

SYNTAX: **fastcgi\_keep\_conn** on | off;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.1.4.

By default, a FastCGI server will close a connection right after sending the response. However, when this directive is set to the value on, nginx will instruct a FastCGI server to keep connections open. This is necessary, in particular, for [keepalive](#) connections to FastCGI servers to function.

## fastcgi\_limit\_rate

SYNTAX: **fastcgi\_limit\_rate** *rate*;  
DEFAULT 0  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.7.

Limits the speed of reading the response from the FastCGI server. The *rate* is specified in bytes per second. The zero value disables rate limiting. The limit is set per a request, and so if nginx simultaneously opens two connections to the FastCFI server, the overall rate will be twice as much as the specified limit. The limitation works only if [buffering](#) of responses from the FastCGI server is enabled.

## fastcgi\_max\_temp\_file\_size

SYNTAX: **fastcgi\_max\_temp\_file\_size** *size*;  
DEFAULT 1024m  
CONTEXT: http, server, location

When [buffering](#) of responses from the FastCGI server is enabled, and the whole response does not fit into the buffers set by the [fastcgi\\_buffer\\_size](#) and [fastcgi\\_buffers](#) directives, a part of the response can be saved to a temporary file. This directive sets the maximum *size* of the temporary file. The size of data written to the temporary file at a time is set by the [fastcgi\\_temp\\_file\\_write\\_size](#) directive.

The zero value disables buffering of responses to temporary files.

This restriction does not apply to responses that will be [cached](#) or [stored](#) on disk.

## fastcgi\_next\_upstream

SYNTAX: **fastcgi\_next\_upstream** error | timeout | invalid\_header |  
http\_500 | http\_503 | http\_403 | http\_404 | http\_429 |  
non\_idempotent | off ...;  
DEFAULT error timeout  
CONTEXT: http, server, location

Specifies in which cases a request should be passed to the next server:

`error`  
 an error occurred while establishing a connection with the server, passing a request to it, or reading the response header;

`timeout`  
 a timeout has occurred while establishing a connection with the server, passing a request to it, or reading the response header;

`invalid_header`  
 a server returned an empty or invalid response;

`http_500`  
 a server returned a response with the code 500;

`http_503`  
 a server returned a response with the code 503;

`http_403`  
 a server returned a response with the code 403;

`http_404`  
 a server returned a response with the code 404;

`http_429`  
 a server returned a response with the code 429 (1.11.13);

`non_idempotent`  
 normally, requests with a [non-idempotent](#) method (POST, LOCK, PATCH) are not passed to the next server if a request has been sent to an upstream server (1.9.13); enabling this option explicitly allows retrying such requests;

`off`  
 disables passing a request to the next server.

One should bear in mind that passing a request to the next server is only possible if nothing has been sent to a client yet. That is, if an error or timeout occurs in the middle of the transferring of a response, fixing this is impossible.

The directive also defines what is considered an [unsuccessful attempt](#) of communication with a server. The cases of `error`, `timeout` and `invalid_header` are always considered unsuccessful attempts, even if they are not specified in the directive. The cases of `http_500`, `http_503`, and `http_429` are considered unsuccessful attempts only if they are specified in the directive. The cases of `http_403` and `http_404` are never considered unsuccessful attempts.

Passing a request to the next server can be limited by [the number of tries](#) and by [time](#).

### **fastcgi\_next\_upstream\_timeout**

SYNTAX: **fastcgi\_next\_upstream\_timeout** *time*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.5.

Limits the time during which a request can be passed to the [next server](#). The 0 value turns off this limitation.

### **fastcgi\_next\_upstream\_tries**

SYNTAX: **fastcgi\_next\_upstream\_tries** *number*;  
 DEFAULT 0  
 CONTEXT: http, server, location  
 THIS DIRECTIVE APPEARED IN VERSION 1.7.5.

Limits the number of possible tries for passing a request to the [next server](#). The 0 value turns off this limitation.

### **fastcgi\_no\_cache**

SYNTAX: **fastcgi\_no\_cache** *string* ...;  
 DEFAULT —  
 CONTEXT: http, server, location

Defines conditions under which the response will not be saved to a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be saved:

```
fastcgi_no_cache $cookie_nocache $arg_nocache$arg_comment;  
fastcgi_no_cache $http_pragma $http_authorization;
```

Can be used along with the [fastcgi\\_cache\\_bypass](#) directive.

### **fastcgi\_param**

SYNTAX: **fastcgi\_param** *parameter value* [if\_not\_empty];  
 DEFAULT —  
 CONTEXT: http, server, location

Sets a *parameter* that should be passed to the FastCGI server. The *value* can contain text, variables, and their combination. These directives are inherited from the previous configuration level if and only if there are no **fastcgi\_param** directives defined on the current level.

The following example shows the minimum required settings for PHP:

```
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;  
fastcgi_param QUERY_STRING $query_string;
```

The `SCRIPT_FILENAME` parameter is used in PHP for determining the script name, and the `QUERY_STRING` parameter is used to pass request parameters.

For scripts that process POST requests, the following three parameters are also required:

```
fastcgi_param REQUEST_METHOD $request_method;  
fastcgi_param CONTENT_TYPE $content_type;
```

```
fastcgi_param CONTENT_LENGTH $content_length;
```

If PHP was built with the `--enable-force-cgi-redirect` configuration parameter, the `REDIRECT_STATUS` parameter should also be passed with the value “200”:

```
fastcgi_param REDIRECT_STATUS 200;
```

If the directive is specified with `if_not_empty` (1.1.11) then such a parameter will be passed to the server only if its value is not empty:

```
fastcgi_param HTTPS          $https if_not_empty;
```

## fastcgi\_pass

SYNTAX: **fastcgi\_pass** *address*;

DEFAULT —

CONTEXT: location, if in location

Sets the address of a FastCGI server. The address can be specified as a domain name or IP address, and a port:

```
fastcgi_pass localhost:9000;
```

or as a UNIX-domain socket path:

```
fastcgi_pass unix:/tmp/fastcgi.socket;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a [server group](#).

Parameter value can contain variables. In this case, if an address is specified as a domain name, the name is searched among the described [server groups](#), and, if not found, is determined using a [resolver](#).

## fastcgi\_pass\_header

SYNTAX: **fastcgi\_pass\_header** *field*;

DEFAULT —

CONTEXT: http, server, location

Permits passing [otherwise disabled](#) header fields from a FastCGI server to a client.

## fastcgi\_pass\_request\_body

SYNTAX: **fastcgi\_pass\_request\_body** on | off;

DEFAULT on

CONTEXT: http, server, location

Indicates whether the original request body is passed to the FastCGI server. See also the [fastcgi\\_pass\\_request\\_headers](#) directive.

### **fastcgi\_pass\_request\_headers**

SYNTAX: **fastcgi\_pass\_request\_headers** on | off;  
DEFAULT on  
CONTEXT: http, server, location

Indicates whether the header fields of the original request are passed to the FastCGI server. See also the [fastcgi\\_pass\\_request\\_body](#) directive.

### **fastcgi\_read\_timeout**

SYNTAX: **fastcgi\_read\_timeout** *time*;  
DEFAULT 60s  
CONTEXT: http, server, location

Defines a timeout for reading a response from the FastCGI server. The timeout is set only between two successive read operations, not for the transmission of the whole response. If the FastCGI server does not transmit anything within this time, the connection is closed.

### **fastcgi\_request\_buffering**

SYNTAX: **fastcgi\_request\_buffering** on | off;  
DEFAULT on  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Enables or disables buffering of a client request body.

When buffering is enabled, the entire request body is [read](#) from the client before sending the request to a FastCGI server.

When buffering is disabled, the request body is sent to the FastCGI server immediately as it is received. In this case, the request cannot be passed to the [next server](#) if nginx already started sending the request body.

### **fastcgi\_send\_lowat**

SYNTAX: **fastcgi\_send\_lowat** *size*;  
DEFAULT 0  
CONTEXT: http, server, location

If the directive is set to a non-zero value, nginx will try to minimize the number of send operations on outgoing connections to a FastCGI server by using either NOTE\_LOWAT flag of the [kqueue](#) method, or the SO\_SNDLOWAT socket option, with the specified *size*.

This directive is ignored on Linux, Solaris, and Windows.



## fastcgi\_send\_timeout

SYNTAX: **fastcgi\_send\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Sets a timeout for transmitting a request to the FastCGI server. The timeout is set only between two successive write operations, not for the transmission of the whole request. If the FastCGI server does not receive anything within this time, the connection is closed.

## fastcgi\_socket\_keepalive

SYNTAX: **fastcgi\_socket\_keepalive** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.15.6.

Configures the “TCP keepalive” behavior for outgoing connections to a FastCGI server. By default, the operating system’s settings are in effect for the socket. If the directive is set to the value “on”, the SO\_KEEPALIVE socket option is turned on for the socket.

## fastcgi\_split\_path\_info

SYNTAX: **fastcgi\_split\_path\_info** *regex*;

DEFAULT —

CONTEXT: location

Defines a regular expression that captures a value for the *\$fastcgi\_path\_info* variable. The regular expression should have two captures: the first becomes a value of the *\$fastcgi\_script\_name* variable, the second becomes a value of the *\$fastcgi\_path\_info* variable. For example, with these settings

```
location ~ ^(.+\.php)(.*)$ {
    fastcgi_split_path_info      ^(.+\.php)(.*)$;
    fastcgi_param SCRIPT_FILENAME /path/to/php$fastcgi_script_name;
    fastcgi_param PATH_INFO      $fastcgi_path_info;
```

and the “/show.php/article/0001” request, the SCRIPT\_FILENAME parameter will be equal to “/path/to/php/show.php”, and the PATH\_INFO parameter will be equal to “/article/0001”.

## fastcgi\_store

SYNTAX: **fastcgi\_store** on | off | *string*;

DEFAULT off

CONTEXT: http, server, location

Enables saving of files to a disk. The on parameter saves files with paths corresponding to the directives [alias](#) or [root](#). The off parameter disables

saving of files. In addition, the file name can be set explicitly using the *string* with variables:

```
fastcgi_store /data/www$original_uri;
```

The modification time of files is set according to the received Last-Modified response header field. The response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the persistent store can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both saved files and a directory holding temporary files, set by the [fastcgi\\_temp\\_path](#) directive, are put on the same file system.

This directive can be used to create local copies of static unchangeable files, e.g.:

```
location /images/ {
    root                /data/www;
    error_page          404 = /fetch$uri;
}

location /fetch/ {
    internal;

    fastcgi_pass        backend:9000;
    ...

    fastcgi_store        on;
    fastcgi_store_access user:rw group:rw all:r;
    fastcgi_temp_path    /data/temp;

    alias                /data/www/;
}
```

### fastcgi\_store\_access

SYNTAX: **fastcgi\_store\_access** *users:permissions* ...;

DEFAULT user:rw

CONTEXT: http, server, location

Sets access permissions for newly created files and directories, e.g.:

```
fastcgi_store_access user:rw group:rw all:r;
```

If any group or all access permissions are specified then user permissions may be omitted:

```
fastcgi_store_access group:rw all:r;
```

### fastcgi\_temp\_file\_write\_size

SYNTAX: **fastcgi\_temp\_file\_write\_size** *size*;

DEFAULT 8k|16k

CONTEXT: http, server, location

Limits the *size* of data written to a temporary file at a time, when buffering of responses from the FastCGI server to temporary files is enabled. By default, *size* is limited by two buffers set by the [fastcgi\\_buffer\\_size](#) and [fastcgi\\_buffers](#) directives. The maximum size of a temporary file is set by the [fastcgi\\_max\\_temp\\_file\\_size](#) directive.

### fastcgi\_temp\_path

SYNTAX: **fastcgi\_temp\_path** *path* [*level1* [*level2* [*level3*]]];

DEFAULT fastcgi\_temp

CONTEXT: http, server, location

Defines a directory for storing temporary files with data received from FastCGI servers. Up to three-level subdirectory hierarchy can be used underneath the specified directory. For example, in the following configuration

```
fastcgi_temp_path /spool/nginx/fastcgi_temp 1 2;
```

a temporary file might look like this:

```
/spool/nginx/fastcgi_temp/7/45/00000123457
```

See also the `use_temp_path` parameter of the [fastcgi\\_cache\\_path](#) directive.

## 2.14.4 Parameters Passed to a FastCGI Server

HTTP request header fields are passed to a FastCGI server as parameters. In applications and scripts running as FastCGI servers, these parameters are usually made available as environment variables. For example, the `User-Agent` header field is passed as the `HTTP_USER_AGENT` parameter. In addition to HTTP request header fields, it is possible to pass arbitrary parameters using the [fastcgi\\_param](#) directive.

## 2.14.5 Embedded Variables

The `ngx_http_fastcgi_module` module supports embedded variables that can be used to set parameters using the [fastcgi\\_param](#) directive:

*\$fastcgi\_script\_name*

request URI or, if a URI ends with a slash, request URI with an index file name configured by the [fastcgi\\_index](#) directive appended to it.

This variable can be used to set the `SCRIPT_FILENAME` and `PATH_TRANSLATED` parameters that determine the script name in PHP. For example, for the `/info/` request with the following directives

```
fastcgi_index index.php;  
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
```

the `SCRIPT_FILENAME` parameter will be equal to `/home/www/scripts/php/info/index.php`.

When using the [fastcgi\\_split\\_path\\_info](#) directive, the `$fastcgi_script_name` variable equals the value of the first capture set by the directive.

*\$fastcgi\_path\_info*

the value of the second capture set by the [fastcgi\\_split\\_path\\_info](#) directive. This variable can be used to set the `PATH_INFO` parameter.

## 2.15 Module ngx\_http\_flv\_module

2.15.1 Summary	156
2.15.2 Example Configuration	156
2.15.3 Directives	156
flv	156

### 2.15.1 Summary

The `ngx_http_flv_module` module provides pseudo-streaming server-side support for Flash Video (FLV) files.

It handles requests with the `start` argument in the request URI's query string specially, by sending back the contents of a file starting from the requested byte offset and with the prepended FLV header.

This module is not built by default, it should be enabled with the `--with-http_flv_module` configuration parameter.

### 2.15.2 Example Configuration

```
location ~ /\.flv$ {
    flv;
}
```

### 2.15.3 Directives

#### flv

SYNTAX: **flv**;

DEFAULT —

CONTEXT: location

Turns on module processing in a surrounding location.

## 2.16 Module ngx\_http\_geo\_module

2.16.1 Summary	157
2.16.2 Example Configuration	157
2.16.3 Directives	157
geo	157

### 2.16.1 Summary

The `ngx_http_geo_module` module creates variables with values depending on the client IP address.

### 2.16.2 Example Configuration

```
geo $geo {
    default          0;

    127.0.0.1        2;
    192.168.1.0/24    1;
    10.1.0.0/16       1;

    ::1              2;
    2001:0db8::/32    1;
}
```

### 2.16.3 Directives

#### geo

SYNTAX: **geo** [*\$address*] *\$variable* { ... }

DEFAULT —

CONTEXT: http

Describes the dependency of values of the specified variable on the client IP address. By default, the address is taken from the `$remote_addr` variable, but it can also be taken from another variable (0.7.27), for example:

```
geo $arg_remote_addr $geo {
    ...;
}
```

Since variables are evaluated only when used, the mere existence of even a large number of declared “geo” variables does not cause any extra costs for request processing.

If the value of a variable does not represent a valid IP address then the “255.255.255.255” address is used.

Addresses are specified either as prefixes in CIDR notation (including individual addresses) or as ranges (0.7.23).

IPv6 prefixes are supported starting from versions 1.3.10 and 1.2.7.

The following special parameters are also supported:

`delete`

deletes the specified network (0.7.23).

`default`

a value set to the variable if the client address does not match any of the specified addresses. When addresses are specified in CIDR notation, “0.0.0.0/0” and “::/0” can be used instead of `default`. When `default` is not specified, the default value will be an empty string.

`include`

includes a file with addresses and values. There can be several inclusions.

`proxy`

defines trusted addresses (0.8.7, 0.7.63). When a request comes from a trusted address, an address from the `X-Forwarded-For` request header field will be used instead. In contrast to the regular addresses, trusted addresses are checked sequentially.

Trusted IPv6 addresses are supported starting from versions 1.3.0 and 1.2.1.

`proxy_recursive`

enables recursive address search (1.3.0, 1.2.1). If recursive search is disabled then instead of the original client address that matches one of the trusted addresses, the last address sent in `X-Forwarded-For` will be used. If recursive search is enabled then instead of the original client address that matches one of the trusted addresses, the last non-trusted address sent in `X-Forwarded-For` will be used.

`ranges`

indicates that addresses are specified as ranges (0.7.23). This parameter should be the first. To speed up loading of a geo base, addresses should be put in ascending order.

Example:

```
geo $country {
    default      ZZ;
    include      conf/geo.conf;
    delete      127.0.0.0/16;
    proxy        192.168.100.0/24;
    proxy        2001:0db8::/32;

    127.0.0.0/24  US;
    127.0.0.1/32  RU;
    10.1.0.0/16   RU;
    192.168.1.0/24 UK;
}
```

The `conf/geo.conf` file could contain the following lines:

```
10.2.0.0/16    RU;  
192.168.2.0/24 RU;
```

A value of the most specific match is used. For example, for the 127.0.0.1 address the value “RU” will be chosen, not “US”.

Example with ranges:

```
geo $country {  
    ranges;  
    default                ZZ;  
    127.0.0.0-127.0.0.0    US;  
    127.0.0.1-127.0.0.1    RU;  
    127.0.0.1-127.0.0.255  US;  
    10.1.0.0-10.1.255.255  RU;  
    192.168.1.0-192.168.1.255 UK;  
}
```



## 2.17 Module ngx\_http\_geoip\_module

2.17.1 Summary	160
2.17.2 Example Configuration	160
2.17.3 Directives	160
geoip_country	160
geoip_city	161
geoip_org	162
geoip_proxy	162
geoip_proxy_recursive	162

### 2.17.1 Summary

The ngx\_http\_geoip\_module module (0.8.6+) creates variables with values depending on the client IP address, using the precompiled [MaxMind](#) databases.

When using the databases with IPv6 support (1.3.12, 1.2.7), IPv4 addresses are looked up as IPv4-mapped IPv6 addresses.

This module is not built by default, it should be enabled with the `--with-http_geoip_module` configuration parameter.

This module requires the [MaxMind GeoIP](#) library.

### 2.17.2 Example Configuration

```
http {
    geoip_country      GeoIP.dat;
    geoip_city         GeoLiteCity.dat;
    geoip_proxy        192.168.100.0/24;
    geoip_proxy        2001:0db8::/32;
    geoip_proxy_recursive on;
    ...
}
```

### 2.17.3 Directives

#### geoip\_country

SYNTAX: **geoip\_country** *file*;

DEFAULT —

CONTEXT: http

Specifies a database used to determine the country depending on the client IP address. The following variables are available when using this database:

*\$geoip\_country\_code*

two-letter country code, for example, “RU”, “US”.

*\$geoip\_country\_code3*

three-letter country code, for example, “RUS”, “USA”.

*\$geoip\_country\_name*

country name, for example, “Russian Federation”, “United States”.

## geoip\_city

SYNTAX: **geoip\_city** *file*;

DEFAULT —

CONTEXT: http

Specifies a database used to determine the country, region, and city depending on the client IP address. The following variables are available when using this database:

*\$geoip\_area\_code*

telephone area code (US only).

This variable may contain outdated information since the corresponding database field is deprecated.

*\$geoip\_city\_continent\_code*

two-letter continent code, for example, “EU”, “NA”.

*\$geoip\_city\_country\_code*

two-letter country code, for example, “RU”, “US”.

*\$geoip\_city\_country\_code3*

three-letter country code, for example, “RUS”, “USA”.

*\$geoip\_city\_country\_name*

country name, for example, “Russian Federation”, “United States”.

*\$geoip\_dma\_code*

DMA region code in US (also known as “metro code”), according to the [geotargeting](#) in Google AdWords API.

*\$geoip\_latitude*

latitude.

*\$geoip\_longitude*

longitude.

*\$geoip\_region*

two-symbol country region code (region, territory, state, province, federal land and the like), for example, “48”, “DC”.

*\$geoip\_region\_name*

country region name (region, territory, state, province, federal land and the like), for example, “Moscow City”, “District of Columbia”.

*\$geoip\_city*

city name, for example, “Moscow”, “Washington”.

*\$geoip\_postal\_code*

postal code.

### **geoip\_org**

SYNTAX: **geoip\_org** *file*;

DEFAULT —

CONTEXT: http

THIS DIRECTIVE APPEARED IN VERSION 1.0.3.

Specifies a database used to determine the organization depending on the client IP address. The following variable is available when using this database:

*\$geoip\_org*

organization name, for example, “The University of Melbourne”.

### **geoip\_proxy**

SYNTAX: **geoip\_proxy** *address* | *CIDR*;

DEFAULT —

CONTEXT: http

THIS DIRECTIVE APPEARED IN VERSIONS 1.3.0 AND 1.2.1.

Defines trusted addresses. When a request comes from a trusted address, an address from the X-Forwarded-For request header field will be used instead.

### **geoip\_proxy\_recursive**

SYNTAX: **geoip\_proxy\_recursive** on | off;

DEFAULT off

CONTEXT: http

THIS DIRECTIVE APPEARED IN VERSIONS 1.3.0 AND 1.2.1.

If recursive search is disabled then instead of the original client address that matches one of the trusted addresses, the last address sent in X-Forwarded-For will be used. If recursive search is enabled then instead of the original client address that matches one of the trusted addresses, the last non-trusted address sent in X-Forwarded-For will be used.

## 2.18 Module ngx\_http\_grpc\_module

2.18.1	Summary	163
2.18.2	Example Configuration	163
2.18.3	Directives	164
	grpc_bind	164
	grpc_buffer_size	164
	grpc_connect_timeout	164
	grpc_hide_header	165
	grpc_ignore_headers	165
	grpc_intercept_errors	165
	grpc_next_upstream	165
	grpc_next_upstream_timeout	166
	grpc_next_upstream_tries	167
	grpc_pass	167
	grpc_pass_header	167
	grpc_read_timeout	168
	grpc_send_timeout	168
	grpc_set_header	168
	grpc_socket_keepalive	168
	grpc_ssl_certificate	169
	grpc_ssl_certificate_key	169
	grpc_ssl_ciphers	169
	grpc_ssl_conf_command	169
	grpc_ssl_crl	170
	grpc_ssl_name	170
	grpc_ssl_password_file	170
	grpc_ssl_protocols	170
	grpc_ssl_server_name	170
	grpc_ssl_session_reuse	171
	grpc_ssl_trusted_certificate	171
	grpc_ssl_verify	171
	grpc_ssl_verify_depth	171

### 2.18.1 Summary

The `ngx_http_grpc_module` module allows passing requests to a gRPC server (1.13.10). The module requires the [ngx\\_http\\_v2\\_module](#) module.

### 2.18.2 Example Configuration

```
server {
    listen 9000 http2;

    location / {
        grpc_pass 127.0.0.1:9000;
    }
}
```

### 2.18.3 Directives

#### **grpc\_bind**

SYNTAX: **grpc\_bind** *address* [transparent ] | off;

DEFAULT —

CONTEXT: http, server, location

Makes outgoing connections to a gRPC server originate from the specified local IP address with an optional port. Parameter value can contain variables. The special value `off` cancels the effect of the `grpc_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address and port.

The `transparent` parameter allows outgoing connections to a gRPC server originate from a non-local IP address, for example, from a real IP address of a client:

```
grpc_bind $remote_addr transparent;
```

In order for this parameter to work, it is usually necessary to run nginx worker processes with the [superuser](#) privileges. On Linux it is not required as if the `transparent` parameter is specified, worker processes inherit the `CAP_NET_RAW` capability from the master process. It is also necessary to configure kernel routing table to intercept network traffic from the gRPC server.

#### **grpc\_buffer\_size**

SYNTAX: **grpc\_buffer\_size** *size*;

DEFAULT 4k | 8k

CONTEXT: http, server, location

Sets the *size* of the buffer used for reading the response received from the gRPC server. The response is passed to the client synchronously, as soon as it is received.

#### **grpc\_connect\_timeout**

SYNTAX: **grpc\_connect\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for establishing a connection with a gRPC server. It should be noted that this timeout cannot usually exceed 75 seconds.

## grpc\_hide\_header

SYNTAX: **grpc\_hide\_header** *field*;

DEFAULT —

CONTEXT: http, server, location

By default, nginx does not pass the header fields `Date`, `Server`, and `X-Accel-...` from the response of a gRPC server to a client. The `grpc_hide_header` directive sets additional fields that will not be passed. If, on the contrary, the passing of fields needs to be permitted, the [grpc\\_pass\\_header](#) directive can be used.

## grpc\_ignore\_headers

SYNTAX: **grpc\_ignore\_headers** *field* ...;

DEFAULT —

CONTEXT: http, server, location

Disables processing of certain response header fields from the gRPC server. The following fields can be ignored: `X-Accel-Redirect` and `X-Accel-Charset`.

If not disabled, processing of these header fields has the following effect:

- `X-Accel-Redirect` performs an [internal redirect](#) to the specified URI;
- `X-Accel-Charset` sets the desired [charset](#) of a response.

## grpc\_intercept\_errors

SYNTAX: **grpc\_intercept\_errors** on | off;

DEFAULT off

CONTEXT: http, server, location

Determines whether gRPC server responses with codes greater than or equal to 300 should be passed to a client or be intercepted and redirected to nginx for processing with the [error\\_page](#) directive.

## grpc\_next\_upstream

SYNTAX: **grpc\_next\_upstream** error | timeout | invalid\_header |  
http\_500 | http\_502 | http\_503 | http\_504 | http\_403 |  
http\_404 | http\_429 | non\_idempotent | off ...;

DEFAULT error timeout

CONTEXT: http, server, location

Specifies in which cases a request should be passed to the next server:

### error

an error occurred while establishing a connection with the server, passing a request to it, or reading the response header;

**timeout**  
 a timeout has occurred while establishing a connection with the server,  
 passing a request to it, or reading the response header;

**invalid\_header**  
 a server returned an empty or invalid response;

**http\_500**  
 a server returned a response with the code 500;

**http\_502**  
 a server returned a response with the code 502;

**http\_503**  
 a server returned a response with the code 503;

**http\_504**  
 a server returned a response with the code 504;

**http\_403**  
 a server returned a response with the code 403;

**http\_404**  
 a server returned a response with the code 404;

**http\_429**  
 a server returned a response with the code 429;

**non\_idempotent**  
 normally, requests with a [non-idempotent](#) method (POST, LOCK, PATCH)  
 are not passed to the next server if a request has been sent to an upstream  
 server; enabling this option explicitly allows retrying such requests;

**off**  
 disables passing a request to the next server.

One should bear in mind that passing a request to the next server is only possible if nothing has been sent to a client yet. That is, if an error or timeout occurs in the middle of the transferring of a response, fixing this is impossible.

The directive also defines what is considered an [unsuccessful attempt](#) of communication with a server. The cases of `error`, `timeout` and `invalid_header` are always considered unsuccessful attempts, even if they are not specified in the directive. The cases of `http_500`, `http_502`, `http_503`, `http_504`, and `http_429` are considered unsuccessful attempts only if they are specified in the directive. The cases of `http_403` and `http_404` are never considered unsuccessful attempts.

Passing a request to the next server can be limited by [the number of tries](#) and by [time](#).

### **grpc\_next\_upstream\_timeout**

SYNTAX: **grpc\_next\_upstream\_timeout** *time*;

DEFAULT 0

CONTEXT: http, server, location

Limits the time during which a request can be passed to the [next server](#). The 0 value turns off this limitation.

### grpc\_next\_upstream\_tries

SYNTAX: **grpc\_next\_upstream\_tries** *number*;  
DEFAULT 0  
CONTEXT: http, server, location

Limits the number of possible tries for passing a request to the [next server](#). The 0 value turns off this limitation.

### grpc\_pass

SYNTAX: **grpc\_pass** *address*;  
DEFAULT —  
CONTEXT: location, if in location

Sets the gRPC server address. The address can be specified as a domain name or IP address, and a port:

```
grpc_pass localhost:9000;
```

or as a UNIX-domain socket path:

```
grpc_pass unix:/tmp/grpc.socket;
```

Alternatively, the “`grpc://`” scheme can be used:

```
grpc_pass grpc://127.0.0.1:9000;
```

To use gRPC over SSL, the “`grpcs://`” scheme should be used:

```
grpc_pass grpcs://127.0.0.1:443;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a [server group](#).

Parameter value can contain variables (1.17.8). In this case, if an address is specified as a domain name, the name is searched among the described [server groups](#), and, if not found, is determined using a [resolver](#).

### grpc\_pass\_header

SYNTAX: **grpc\_pass\_header** *field*;  
DEFAULT —  
CONTEXT: http, server, location

Permits passing [otherwise disabled](#) header fields from a gRPC server to a client.



### grpc\_read\_timeout

SYNTAX: **grpc\_read\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for reading a response from the gRPC server. The timeout is set only between two successive read operations, not for the transmission of the whole response. If the gRPC server does not transmit anything within this time, the connection is closed.

### grpc\_send\_timeout

SYNTAX: **grpc\_send\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Sets a timeout for transmitting a request to the gRPC server. The timeout is set only between two successive write operations, not for the transmission of the whole request. If the gRPC server does not receive anything within this time, the connection is closed.

### grpc\_set\_header

SYNTAX: **grpc\_set\_header** *field value*;

DEFAULT Content-Length \$content\_length

CONTEXT: http, server, location

Allows redefining or appending fields to the request header passed to the gRPC server. The *value* can contain text, variables, and their combinations. These directives are inherited from the previous configuration level if and only if there are no `grpc_set_header` directives defined on the current level.

If the value of a header field is an empty string then this field will not be passed to a gRPC server:

```
grpc_set_header Accept-Encoding "";
```

### grpc\_socket\_keepalive

SYNTAX: **grpc\_socket\_keepalive** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.15.6.

Configures the “TCP keepalive” behavior for outgoing connections to a gRPC server. By default, the operating system’s settings are in effect for the socket. If the directive is set to the value “on”, the `SO_KEEPALIVE` socket option is turned on for the socket.

### grpc\_ssl\_certificate

SYNTAX: **grpc\_ssl\_certificate** *file*;

DEFAULT —

CONTEXT: http, server, location

Specifies a *file* with the certificate in the PEM format used for authentication to a gRPC SSL server.

Since version 1.21.0, variables can be used in the *file* name.

### grpc\_ssl\_certificate\_key

SYNTAX: **grpc\_ssl\_certificate\_key** *file*;

DEFAULT —

CONTEXT: http, server, location

Specifies a *file* with the secret key in the PEM format used for authentication to a gRPC SSL server.

The value *engine:name:id* can be specified instead of the *file*, which loads a secret key with a specified *id* from the OpenSSL engine *name*.

Since version 1.21.0, variables can be used in the *file* name.

### grpc\_ssl\_ciphers

SYNTAX: **grpc\_ssl\_ciphers** *ciphers*;

DEFAULT DEFAULT

CONTEXT: http, server, location

Specifies the enabled ciphers for requests to a gRPC SSL server. The ciphers are specified in the format understood by the OpenSSL library.

The full list can be viewed using the “`openssl ciphers`” command.

### grpc\_ssl\_conf\_command

SYNTAX: **grpc\_ssl\_conf\_command** *name value*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

Sets arbitrary OpenSSL configuration [commands](#) when establishing a connection with the gRPC SSL server.

The directive is supported when using OpenSSL 1.0.2 or higher.

Several `grpc_ssl_conf_command` directives can be specified on the same level. These directives are inherited from the previous configuration level if and only if there are no `grpc_ssl_conf_command` directives defined on the current level.

Note that configuring OpenSSL directly might result in unexpected behavior.

### grpc\_ssl\_crl

SYNTAX: **grpc\_ssl\_crl** *file*;  
DEFAULT —  
CONTEXT: http, server, location

Specifies a *file* with revoked certificates (CRL) in the PEM format used to [verify](#) the certificate of the gRPC SSL server.

### grpc\_ssl\_name

SYNTAX: **grpc\_ssl\_name** *name*;  
DEFAULT host from `grpc_pass`  
CONTEXT: http, server, location

Allows overriding the server name used to [verify](#) the certificate of the gRPC SSL server and to be [passed through SNI](#) when establishing a connection with the gRPC SSL server.

By default, the host part from [grpc\\_pass](#) is used.

### grpc\_ssl\_password\_file

SYNTAX: **grpc\_ssl\_password\_file** *file*;  
DEFAULT —  
CONTEXT: http, server, location

Specifies a *file* with passphrases for [secret keys](#) where each passphrase is specified on a separate line. Passphrases are tried in turn when loading the key.

### grpc\_ssl\_protocols

SYNTAX: **grpc\_ssl\_protocols** [SSLv2] [SSLv3] [TLSv1] [TLSv1.1]  
[TLSv1.2] [TLSv1.3];  
DEFAULT TLSv1 TLSv1.1 TLSv1.2  
CONTEXT: http, server, location

Enables the specified protocols for requests to a gRPC SSL server.

### grpc\_ssl\_server\_name

SYNTAX: **grpc\_ssl\_server\_name** on | off;  
DEFAULT off  
CONTEXT: http, server, location

Enables or disables passing of the server name through [TLS Server Name Indication extension](#) (SNI, RFC 6066) when establishing a connection with the gRPC SSL server.

### **grpc\_ssl\_session\_reuse**

SYNTAX: **grpc\_ssl\_session\_reuse** on | off;  
DEFAULT on  
CONTEXT: http, server, location

Determines whether SSL sessions can be reused when working with the gRPC server. If the errors “SSL3\_GET\_FINISHED:digest check failed” appear in the logs, try disabling session reuse.

### **grpc\_ssl\_trusted\_certificate**

SYNTAX: **grpc\_ssl\_trusted\_certificate** *file*;  
DEFAULT —  
CONTEXT: http, server, location

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) the certificate of the gRPC SSL server.

### **grpc\_ssl\_verify**

SYNTAX: **grpc\_ssl\_verify** on | off;  
DEFAULT off  
CONTEXT: http, server, location

Enables or disables verification of the gRPC SSL server certificate.

### **grpc\_ssl\_verify\_depth**

SYNTAX: **grpc\_ssl\_verify\_depth** *number*;  
DEFAULT 1  
CONTEXT: http, server, location

Sets the verification depth in the gRPC SSL server certificates chain.

## 2.19 Module ngx\_http\_gunzip\_module

2.19.1 Summary	172
2.19.2 Example Configuration	172
2.19.3 Directives	172
gunzip	172
gunzip_buffers	172

### 2.19.1 Summary

The `ngx_http_gunzip_module` module is a filter that decompresses responses with “Content-Encoding: gzip” for clients that do not support “gzip” encoding method. The module will be useful when it is desirable to store data compressed to save space and reduce I/O costs.

This module is not built by default, it should be enabled with the `--with-http_gunzip_module` configuration parameter.

### 2.19.2 Example Configuration

```
location /storage/ {
    gunzip on;
    ...
}
```

### 2.19.3 Directives

#### gunzip

SYNTAX: **gunzip** on | off;  
 DEFAULT off  
 CONTEXT: http, server, location

Enables or disables decompression of gzipped responses for clients that lack gzip support. If enabled, the following directives are also taken into account when determining if clients support gzip: [gzip\\_http\\_version](#), [gzip\\_proxied](#), and [gzip\\_disable](#). See also the [gzip\\_vary](#) directive.

#### gunzip\_buffers

SYNTAX: **gunzip\_buffers** *number size*;  
 DEFAULT 32 4k | 16 8k  
 CONTEXT: http, server, location

Sets the *number* and *size* of buffers used to decompress a response. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

## 2.20 Module ngx\_http\_gzip\_module

2.20.1	Summary	173
2.20.2	Example Configuration	173
2.20.3	Directives	173
	gzip	173
	gzip_buffers	174
	gzip_comp_level	174
	gzip_disable	174
	gzip_http_version	174
	gzip_min_length	174
	gzip_proxied	175
	gzip_types	175
	gzip_vary	176
2.20.4	Embedded Variables	176

### 2.20.1 Summary

The `ngx_http_gzip_module` module is a filter that compresses responses using the “gzip” method. This often helps to reduce the size of transmitted data by half or even more.

When using the SSL/TLS protocol, compressed responses may be subject to [BREACH](#) attacks.

### 2.20.2 Example Configuration

```
gzip            on;
gzip_min_length 1000;
gzip_proxied    expired no-cache no-store private auth;
gzip_types      text/plain application/xml;
```

The `$gzip_ratio` variable can be used to log the achieved compression ratio.

### 2.20.3 Directives

#### gzip

SYNTAX: **gzip** on | off;

DEFAULT off

CONTEXT: http, server, location, if in location

Enables or disables gzipping of responses.

## gzip\_buffers

SYNTAX: **gzip\_buffers** *number size*;  
DEFAULT 32 4k | 16 8k  
CONTEXT: http, server, location

Sets the *number* and *size* of buffers used to compress a response. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

Until version 0.7.28, four 4K or 8K buffers were used by default.

## gzip\_comp\_level

SYNTAX: **gzip\_comp\_level** *level*;  
DEFAULT 1  
CONTEXT: http, server, location

Sets a gzip compression *level* of a response. Acceptable values are in the range from 1 to 9.

## gzip\_disable

SYNTAX: **gzip\_disable** *regex ...*;  
DEFAULT —  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.6.23.

Disables gzipping of responses for requests with User-Agent header fields matching any of the specified regular expressions.

The special mask “msie6” (0.7.12) corresponds to the regular expression “MSIE [4–6] \.”, but works faster. Starting from version 0.8.11, “MSIE 6.0; ...SV1” is excluded from this mask.

## gzip\_http\_version

SYNTAX: **gzip\_http\_version** 1.0 | 1.1;  
DEFAULT 1.1  
CONTEXT: http, server, location

Sets the minimum HTTP version of a request required to compress a response.

## gzip\_min\_length

SYNTAX: **gzip\_min\_length** *length*;  
DEFAULT 20  
CONTEXT: http, server, location

Sets the minimum length of a response that will be gzipped. The length is determined only from the Content-Length response header field.

## gzip\_proxied

SYNTAX: **gzip\_proxied** off | expired | no-cache | no-store | private | no\_last\_modified | no\_etag | auth | any ...;

DEFAULT off

CONTEXT: http, server, location

Enables or disables gzipping of responses for proxied requests depending on the request and response. The fact that the request is proxied is determined by the presence of the `Via` request header field. The directive accepts multiple parameters:

off

disables compression for all proxied requests, ignoring other parameters;

expired

enables compression if a response header includes the `Expires` field with a value that disables caching;

no-cache

enables compression if a response header includes the `Cache-Control` field with the “no-cache” parameter;

no-store

enables compression if a response header includes the `Cache-Control` field with the “no-store” parameter;

private

enables compression if a response header includes the `Cache-Control` field with the “private” parameter;

no\_last\_modified

enables compression if a response header does not include the `Last-Modified` field;

no\_etag

enables compression if a response header does not include the `ETag` field;

auth

enables compression if a request header includes the `Authorization` field;

any

enables compression for all proxied requests.

## gzip\_types

SYNTAX: **gzip\_types** *mime-type* ...;

DEFAULT text/html

CONTEXT: http, server, location

Enables gzipping of responses for the specified MIME types in addition to “text/html”. The special value “\*” matches any MIME type (0.8.29). Responses with the “text/html” type are always compressed.



**gzip\_vary**

SYNTAX: **gzip\_vary** on | off;

DEFAULT off

CONTEXT: http, server, location

Enables or disables inserting the `Vary: Accept-Encoding` response header field if the directives `gzip`, `gzip_static`, or `gunzip` are active.

**2.20.4 Embedded Variables**

*\$gzip\_ratio*

achieved compression ratio, computed as the ratio between the original and compressed response sizes.

## 2.21 Module ngx\_http\_gzip\_static\_module

2.21.1 Summary	177
2.21.2 Example Configuration	177
2.21.3 Directives	177
gzip_static	177

### 2.21.1 Summary

The `ngx_http_gzip_static_module` module allows sending precompressed files with the “.gz” filename extension instead of regular files.

This module is not built by default, it should be enabled with the `--with-http_gzip_static_module` configuration parameter.

### 2.21.2 Example Configuration

```
gzip_static on;
gzip_proxied expired no-cache no-store private auth;
```

### 2.21.3 Directives

#### gzip\_static

SYNTAX: **gzip\_static** on | off | always;

DEFAULT off

CONTEXT: http, server, location

Enables (“on”) or disables (“off”) checking the existence of precompressed files. The following directives are also taken into account: [gzip\\_http\\_version](#), [gzip\\_proxied](#), [gzip\\_disable](#), and [gzip\\_vary](#).

With the “always” value (1.3.6), gzipped file is used in all cases, without checking if the client supports it. It is useful if there are no uncompressed files on the disk anyway or the [ngx\\_http\\_gunzip\\_module](#) is used.

The files can be compressed using the `gzip` command, or any other compatible one. It is recommended that the modification date and time of original and compressed files be the same.

## 2.22 Module ngx\_http\_headers\_module

2.22.1 Summary	178
2.22.2 Example Configuration	178
2.22.3 Directives	178
add_header	178
add_trailer	178
expires	179

### 2.22.1 Summary

The ngx\_http\_headers\_module module allows adding the Expires and Cache-Control header fields, and arbitrary fields, to a response header.

### 2.22.2 Example Configuration

```
expires      24h;
expires      modified +24h;
expires      @24h;
expires      0;
expires      -1;
expires      epoch;
expires      $expires;
add_header   Cache-Control private;
```

### 2.22.3 Directives

#### add\_header

SYNTAX: **add\_header** *name value* [always];

DEFAULT —

CONTEXT: http, server, location, if in location

Adds the specified field to a response header provided that the response code equals 200, 201 (1.3.10), 204, 206, 301, 302, 303, 304, 307 (1.1.16, 1.0.13), or 308 (1.13.0). Parameter value can contain variables.

There could be several add\_header directives. These directives are inherited from the previous configuration level if and only if there are no add\_header directives defined on the current level.

If the always parameter is specified (1.7.5), the header field will be added regardless of the response code.

#### add\_trailer

SYNTAX: **add\_trailer** *name value* [always];

DEFAULT —

CONTEXT: http, server, location, if in location

THIS DIRECTIVE APPEARED IN VERSION 1.13.2.

Adds the specified field to the end of a response provided that the response code equals 200, 201, 206, 301, 302, 303, 307, or 308. Parameter value can contain variables.

There could be several `add_trailer` directives. These directives are inherited from the previous configuration level if and only if there are no `add_trailer` directives defined on the current level.

If the `always` parameter is specified the specified field will be added regardless of the response code.

## expires

SYNTAX: **expires** [modified] *time*;  
 SYNTAX: **expires** epoch | max | off;  
 DEFAULT off  
 CONTEXT: http, server, location, if in location

Enables or disables adding or modifying the Expires and Cache-Control response header fields provided that the response code equals 200, 201 (1.3.10), 204, 206, 301, 302, 303, 304, 307 (1.1.16, 1.0.13), or 308 (1.13.0). The parameter can be a positive or negative [time](#).

The time in the Expires field is computed as a sum of the current time and *time* specified in the directive. If the `modified` parameter is used (0.7.0, 0.6.32) then the time is computed as a sum of the file's modification time and the *time* specified in the directive.

In addition, it is possible to specify a time of day using the “@” prefix (0.7.9, 0.6.34):

```
expires @15h30m;
```

The contents of the Cache-Control field depends on the sign of the specified time:

- time is negative — Cache-Control: no-cache.
- time is positive or zero — Cache-Control: max-age=*t*, where *t* is a time specified in the directive, in seconds.

The `epoch` parameter sets Expires to the value “Thu, 01 Jan 1970 00:00:01 GMT”, and Cache-Control to “no-cache”.

The `max` parameter sets Expires to the value “Thu, 31 Dec 2037 23:55:55 GMT”, and Cache-Control to 10 years.

The `off` parameter disables adding or modifying the Expires and Cache-Control response header fields.

The last parameter value can contain variables (1.7.9):

```
map $sent_http_content_type $expires {
    default      off;
    application/pdf 42d;
    ~image/      max;
}
```

```
expires $expires;
```

## 2.23 Module ngx\_http\_hls\_module

2.23.1 Summary	181
2.23.2 Example Configuration	181
2.23.3 Directives	182
hls	182
hls_buffers	182
hls_forward_args	182
hls_fragment	183
hls_mp4_buffer_size	183
hls_mp4_max_buffer_size	184

### 2.23.1 Summary

The `ngx_http_hls_module` module provides HTTP Live Streaming (HLS) server-side support for MP4 and MOV media files. Such files typically have the `.mp4`, `.m4v`, `.m4a`, `.mov`, or `.qt` filename extensions. The module supports H.264 video codec, AAC and MP3 audio codecs.

For each media file, two URIs are supported:

- A playlist URI with the “`.m3u8`” filename extension. The URI can accept optional arguments:
  - “`start`” and “`end`” define playlist boundaries in seconds (1.9.0).
  - “`offset`” shifts an initial playback position to the time offset in seconds (1.9.0). A positive value sets a time offset from the beginning of the playlist. A negative value sets a time offset from the end of the last fragment in the playlist.
  - “`len`” defines the fragment length in seconds.
- A fragment URI with the “`.ts`” filename extension. The URI can accept optional arguments:
  - “`start`” and “`end`” define fragment boundaries in seconds.

This module is available as part of our [commercial subscription](#).

### 2.23.2 Example Configuration

```
location / {
    hls;
    hls_fragment          5s;
    hls_buffers           10 10m;
    hls_mp4_buffer_size   1m;
    hls_mp4_max_buffer_size 5m;
    root /var/video/;
}
```

With this configuration, the following URIs are supported for the “/var/www/video/test.mp4” file:

```
http://hls.example.com/test.mp4.m3u8?offset=1.000&start=1.000&end=2.200
http://hls.example.com/test.mp4.m3u8?len=8.000
http://hls.example.com/test.mp4.ts?start=1.000&end=2.200
```

### 2.23.3 Directives

#### hls

SYNTAX: **hls**;  
 DEFAULT —  
 CONTEXT: location

Turns on HLS streaming in the surrounding location.

#### hls\_buffers

SYNTAX: **hls\_buffers** *number size*;  
 DEFAULT 8 2m  
 CONTEXT: http, server, location

Sets the maximum *number* and *size* of buffers that are used for reading and writing data frames.

#### hls\_forward\_args

SYNTAX: **hls\_forward\_args** on | off;  
 DEFAULT off  
 CONTEXT: http, server, location  
 THIS DIRECTIVE APPEARED IN VERSION 1.5.12.

Adds arguments from a playlist request to URIs of fragments. This may be useful for performing client authorization at the moment of requesting a fragment, or when protecting an HLS stream with the [ngx\\_http\\_secure\\_link\\_module](#) module.

For example, if a client requests a playlist `http://example.com/hls/test.mp4.m3u8?a=1&b=2`, the arguments `a=1` and `b=2` will be added to URIs of fragments after the arguments `start` and `end`:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:15
#EXT-X-PLAYLIST-TYPE:VOD

#EXTINF:9.333,
test.mp4.ts?start=0.000&end=9.333&a=1&b=2
#EXTINF:7.167,
test.mp4.ts?start=9.333&end=16.500&a=1&b=2
#EXTINF:5.416,
test.mp4.ts?start=16.500&end=21.916&a=1&b=2
#EXTINF:5.500,
test.mp4.ts?start=21.916&end=27.416&a=1&b=2
```

```
#EXTINF:15.167,
test.mp4.ts?start=27.416&end=42.583&a=1&b=2
#EXTINF:9.626,
test.mp4.ts?start=42.583&end=52.209&a=1&b=2

#EXT-X-ENDLIST
```

If an HLS stream is protected with the [ngx\\_http\\_secure\\_link\\_module](#) module, *\$uri* should not be used in the [secure\\_link\\_md5](#) expression because this will cause errors when requesting the fragments. [Base URI](#) should be used instead of *\$uri* (*\$hls\_uri* in the example):

```
http {
    ...

    map $uri $hls_uri {
        ~^(?<base_uri>.*).m3u8$ $base_uri;
        ~^(?<base_uri>.*).ts$   $base_uri;
        default                 $uri;
    }

    server {
        ...

        location /hls/ {
            hls;
            hls_forward_args on;

            alias /var/videos/;

            secure_link $arg_md5,$arg_expires;
            secure_link_md5 "$secure_link_expires$hls_uri$remote_addr secret";

            if ($secure_link = "") {
                return 403;
            }

            if ($secure_link = "0") {
                return 410;
            }
        }
    }
}
```

## hls\_fragment

SYNTAX: **hls\_fragment** *time*;

DEFAULT 5s

CONTEXT: http, server, location

Defines the default fragment length for playlist URIs requested without the “len” argument.

## hls\_mp4\_buffer\_size

SYNTAX: **hls\_mp4\_buffer\_size** *size*;

DEFAULT 512k

CONTEXT: http, server, location

Sets the initial *size* of the buffer used for processing MP4 and MOV files.



## hls\_mp4\_max\_buffer\_size

SYNTAX: **hls\_mp4\_max\_buffer\_size** *size*;

DEFAULT 10m

CONTEXT: http, server, location

During metadata processing, a larger buffer may become necessary. Its size cannot exceed the specified *size*, or else nginx will return the server error 500 Internal Server Error, and log the following message:

```
"/some/movie/file.mp4" mp4 moov atom is too large:
12583268, you may want to increase hls_mp4_max_buffer_size
```

## 2.24 Module ngx\_http\_image\_filter\_module

2.24.1 Summary	185
2.24.2 Example Configuration	185
2.24.3 Directives	186
image_filter	186
image_filter_buffer	187
image_filter_interlace	187
image_filter_jpeg_quality	187
image_filter_sharpen	187
image_filter_transparency	187
image_filter_webp_quality	188

### 2.24.1 Summary

The `ngx_http_image_filter_module` module (0.7.54+) is a filter that transforms images in JPEG, GIF, PNG, and WebP formats.

This module is not built by default, it should be enabled with the `--with-http_image_filter_module` configuration parameter.

This module utilizes the [libgd](#) library. It is recommended to use the latest available version of the library.

The WebP format support appeared in version 1.11.6. To transform images in this format, the `libgd` library must be compiled with the WebP support.

### 2.24.2 Example Configuration

```
location /img/ {
    proxy_pass    http://backend;
    image_filter  resize 150 100;
    image_filter  rotate 90;
    error_page    415 = /empty;
}

location = /empty {
    empty_gif;
}
```

## 2.24.3 Directives

### image\_filter

SYNTAX: **image\_filter** off;  
 SYNTAX: **image\_filter** test;  
 SYNTAX: **image\_filter** size;  
 SYNTAX: **image\_filter** rotate 90 | 180 | 270;  
 SYNTAX: **image\_filter** resize *width height*;  
 SYNTAX: **image\_filter** crop *width height*;  
 DEFAULT off  
 CONTEXT: location

Sets the type of transformation to perform on images:

**off**

turns off module processing in a surrounding location.

**test**

ensures that responses are images in either JPEG, GIF, PNG, or WebP format. Otherwise, the 415 Unsupported Media Type error is returned.

**size**

outputs information about images in a JSON format, e.g.:

```
{ "img" : { "width": 100, "height": 100, "type": "gif" } }
```

In case of an error, the output is as follows:

```
{ }
```

**rotate 90|180|270**

rotates images counter-clockwise by the specified number of degrees. Parameter value can contain variables. This mode can be used either alone or along with the **resize** and **crop** transformations.

**resize *width height***

proportionally reduces an image to the specified sizes. To reduce by only one dimension, another dimension can be specified as “-”. In case of an error, the server will return code 415 Unsupported Media Type. Parameter values can contain variables. When used along with the **rotate** parameter, the rotation happens **after** reduction.

**crop *width height***

proportionally reduces an image to the larger side size and crops extraneous edges by another side. To reduce by only one dimension, another dimension can be specified as “-”. In case of an error, the server will return code 415 Unsupported Media Type. Parameter values can contain variables. When used along with the **rotate** parameter, the rotation happens **before** reduction.

### image\_filter\_buffer

SYNTAX: **image\_filter\_buffer** *size*;

DEFAULT 1M

CONTEXT: http, server, location

Sets the maximum size of the buffer used for reading images. When the size is exceeded the server returns error 415 Unsupported Media Type.

### image\_filter\_interlace

SYNTAX: **image\_filter\_interlace** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.3.15.

If enabled, final images will be interlaced. For JPEG, final images will be in “progressive JPEG” format.

### image\_filter\_jpeg\_quality

SYNTAX: **image\_filter\_jpeg\_quality** *quality*;

DEFAULT 75

CONTEXT: http, server, location

Sets the desired *quality* of the transformed JPEG images. Acceptable values are in the range from 1 to 100. Lesser values usually imply both lower image quality and less data to transfer. The maximum recommended value is 95. Parameter value can contain variables.

### image\_filter\_sharpen

SYNTAX: **image\_filter\_sharpen** *percent*;

DEFAULT 0

CONTEXT: http, server, location

Increases sharpness of the final image. The sharpness percentage can exceed 100. The zero value disables sharpening. Parameter value can contain variables.

### image\_filter\_transparency

SYNTAX: **image\_filter\_transparency** on|off;

DEFAULT on

CONTEXT: http, server, location

Defines whether transparency should be preserved when transforming GIF images or PNG images with colors specified by a palette. The loss of transparency results in images of a better quality. The alpha channel transparency in PNG is always preserved.

## **image\_filter\_webp\_quality**

SYNTAX: **image\_filter\_webp\_quality** *quality*;

DEFAULT 80

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.11.6.

Sets the desired *quality* of the transformed WebP images. Acceptable values are in the range from 1 to 100. Lesser values usually imply both lower image quality and less data to transfer. Parameter value can contain variables.

## 2.25 Module ngx\_http\_index\_module

2.25.1 Summary	189
2.25.2 Example Configuration	189
2.25.3 Directives	189
index	189

### 2.25.1 Summary

The `ngx_http_index_module` module processes requests ending with the slash character (`'/'`). Such requests can also be processed by the [ngx-http-autoindex-module](#) and [ngx-http-random-index-module](#) modules.

### 2.25.2 Example Configuration

```
location / {
    index index.$geo.html index.html;
}
```

### 2.25.3 Directives

#### index

SYNTAX: **index** *file* ...;  
 DEFAULT index.html  
 CONTEXT: http, server, location

Defines files that will be used as an index. The *file* name can contain variables. Files are checked in the specified order. The last element of the list can be a file with an absolute path. Example:

```
index index.$geo.html index.0.html /index.html;
```

It should be noted that using an index file causes an internal redirect, and the request can be processed in a different location. For example, with the following configuration:

```
location = / {
    index index.html;
}

location / {
    ...
}
```

a `"/` request will actually be processed in the second location as `"/index.html"`.

## 2.26 Module ngx\_http\_js\_module

2.26.1	Summary	190
2.26.2	Example Configuration	190
2.26.3	Directives	192
	js_body_filter	192
	js_content	193
	js_fetch_buffer_size	193
	js_fetch_ciphers	193
	js_fetch_max_response_buffer_size	193
	js_fetch_protocols	194
	js_fetch_timeout	194
	js_fetch_trusted_certificate	194
	js_fetch_verify	194
	js_fetch_verify_depth	194
	js_header_filter	195
	js_import	195
	js_include	195
	js_path	196
	js_preload_object	196
	js_set	196
	js_var	197
2.26.4	Request Argument	197

### 2.26.1 Summary

The `ngx_http_js_module` module is used to implement location and variable handlers in `njs` — a subset of the JavaScript language.

Download and install instructions are available [here](#).

### 2.26.2 Example Configuration

The example works since 0.4.0.

```
http {
    js_import http.js;

    js_set $foo      http.foo;
    js_set $summary  http.summary;
    js_set $hash     http.hash;

    resolver 10.0.0.1;

    server {
        listen 8000;

        location / {
            add_header X-Foo $foo;
            js_content http.baz;
        }

        location = /summary {
            return 200 $summary;
        }
    }
}
```

```

    }

    location = /hello {
        js_content http.hello;
    }

    # since 0.7.0
    location = /fetch {
        js_content          http.fetch;
        js_fetch_trusted_certificate /path/to/ISRG_Root_X1.pem;
    }

    # since 0.7.0
    location = /crypto {
        add_header Hash $hash;
        return      200;
    }
}

```

The `http.js` file:

```

function foo(r) {
    r.log("hello from foo() handler");
    return "foo";
}

function summary(r) {
    var a, s, h;

    s = "JS summary\n\n";

    s += "Method: " + r.method + "\n";
    s += "HTTP version: " + r.httpVersion + "\n";
    s += "Host: " + r.headersIn.host + "\n";
    s += "Remote Address: " + r.remoteAddress + "\n";
    s += "URI: " + r.uri + "\n";

    s += "Headers:\n";
    for (h in r.headersIn) {
        s += "  header '" + h + "' is '" + r.headersIn[h] + "'\n";
    }

    s += "Args:\n";
    for (a in r.args) {
        s += "  arg '" + a + "' is '" + r.args[a] + "'\n";
    }

    return s;
}

function baz(r) {
    r.status = 200;
    r.headersOut.foo = 1234;
    r.headersOut['Content-Type'] = "text/plain; charset=utf-8";
    r.headersOut['Content-Length'] = 15;
    r.sendHeader();
    r.send("nginx");
    r.send("java");
    r.send("script");

    r.finish();
}

function hello(r) {
    r.return(200, "Hello world!");
}

// since 0.7.0

```



```

async function fetch(r) {
    let results = await Promise.all([ngx.fetch('https://nginx.org/'),
                                     ngx.fetch('https://nginx.org/en/')]);

    r.return(200, JSON.stringify(results, undefined, 4));
}

// since 0.7.0
async function hash(r) {
    let hash = await crypto.subtle.digest('SHA-512', r.headersIn.host);
    r.setReturnValue(Buffer.from(hash).toString('hex'));
}

export default {foo, summary, baz, hello, fetch, hash};

```

## 2.26.3 Directives

### js\_body\_filter

SYNTAX: **js\_body\_filter** *function* | *module.function* [*buffer\_type=string* | *buffer*];

DEFAULT —

CONTEXT: location, if in location, limit\_except

THIS DIRECTIVE APPEARED IN VERSION 0.5.2.

Sets an `njs` function as a response body filter. The filter function is called for each data chunk of a response body with the following arguments:

- `r`  
the HTTP request object
- `data`  
the incoming data chunk, may be a string or Buffer depending on the `buffer_type` value, by default is a string.
- `flags`  
an object with the following properties:
  - `last`  
a boolean value, true if data is a last buffer.

The filter function can pass its own modified version of the input data chunk to the next body filter by calling `r.sendBuffer()`. For example, to transform all the lowercase letters in the response body:

```

function filter(r, data, flags) {
    r.sendBuffer(data.toLowerCase(), flags);
}

```

To stop filtering (following data chunks will be passed to client without calling `js_body_filter`), `r.done()` can be used.

If the filter function changes the length of the response body, then it is required to clear out the Content-Length response header (if any) in `js_header_filter` to enforce chunked transfer encoding.

As the `js_body_filter` handler returns its result immediately, it supports only synchronous operations. Thus, asynchronous operations such as `r.subrequest()` or `setTimeout()` are not supported.

The directive can be specified inside the `if` block since 0.7.7.

### **js\_content**

SYNTAX: **js\_content** *function* | *module.function*;

DEFAULT —

CONTEXT: location, if in location, limit\_except

Sets an `njs` function as a location content handler. Since 0.4.0, a module function can be referenced.

The directive can be specified inside the `if` block since 0.7.7.

### **js\_fetch\_buffer\_size**

SYNTAX: **js\_fetch\_buffer\_size** *size*;

DEFAULT 16k

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.7.4.

Sets the *size* of the buffer used for reading and writing with Fetch API.

### **js\_fetch\_ciphers**

SYNTAX: **js\_fetch\_ciphers** *ciphers*;

DEFAULT HIGH:!aNULL:!MD5

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.7.0.

Specifies the enabled ciphers for HTTPS requests with Fetch API. The ciphers are specified in the format understood by the OpenSSL library.

The full list can be viewed using the “`openssl ciphers`” command.

### **js\_fetch\_max\_response\_buffer\_size**

SYNTAX: **js\_fetch\_max\_response\_buffer\_size** *size*;

DEFAULT 1m

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.7.4.

Sets the maximum *size* of the response received with Fetch API.

### **js\_fetch\_protocols**

SYNTAX: **js\_fetch\_protocols** [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];  
DEFAULT TLSv1 TLSv1.1 TLSv1.2  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.7.0.

Enables the specified protocols for HTTPS requests with Fetch API.

### **js\_fetch\_timeout**

SYNTAX: **js\_fetch\_timeout** *time*;  
DEFAULT 60s  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.7.4.

Defines a timeout for reading and writing for Fetch API. The timeout is set only between two successive read/write operations, not for the whole response. If no data is transmitted within this time, the connection is closed.

### **js\_fetch\_trusted\_certificate**

SYNTAX: **js\_fetch\_trusted\_certificate** *file*;  
DEFAULT —  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.7.0.

Specifies a *file* with trusted CA certificates in the PEM format used to verify the HTTPS certificate with Fetch API.

### **js\_fetch\_verify**

SYNTAX: **js\_fetch\_verify** on | off;  
DEFAULT on  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.7.4.

Enables or disables verification of the HTTPS server certificate with Fetch API.

### **js\_fetch\_verify\_depth**

SYNTAX: **js\_fetch\_verify\_depth** *number*;  
DEFAULT 100  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.7.0.

Sets the verification depth in the HTTPS server certificates chain with Fetch API.

## js\_header\_filter

SYNTAX: **js\_header\_filter** *function* | *module.function*;

DEFAULT —

CONTEXT: location, if in location, limit\_except

THIS DIRECTIVE APPEARED IN VERSION 0.5.1.

Sets an njs function as a response header filter. The directive allows changing arbitrary header fields of a response header.

As the `js_header_filter` handler returns its result immediately, it supports only synchronous operations. Thus, asynchronous operations such as `r.subrequest()` or `setTimeout()` are not supported.

The directive can be specified inside the `if` block since 0.7.7.

## js\_import

SYNTAX: **js\_import** *module.js* | *export\_name from module.js*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.4.0.

Imports a module that implements location and variable handlers in njs. The `export_name` is used as a namespace to access module functions. If the `export_name` is not specified, the module name will be used as a namespace.

```
js_import http.js;
```

Here, the module name `http` is used as a namespace while accessing exports. If the imported module exports `foo()`, `http.foo` is used to refer to it.

Several `js_import` directives can be specified.

The directive can be specified on the `server` and `location` level since 0.7.7.

## js\_include

SYNTAX: **js\_include** *file*;

DEFAULT —

CONTEXT: http

Specifies a file that implements location and variable handlers in njs:

```
nginx.conf:
js_include http.js;
location /version {
    js_content version;
}
```

```
http.js:
function version(r) {
    r.return(200, njs.version);
}
```

The directive was made obsolete in version 0.4.0 and was removed in version 0.7.1. The [js\\_import](#) directive should be used instead.

## js\_path

SYNTAX: **js\_path** *path*;  
DEFAULT —  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.3.0.

Sets an additional path for njs modules.

The directive can be specified on the `server` and `location` level since 0.7.7.

## js\_preload\_object

SYNTAX: **js\_preload\_object** *name.json* | *name* from *file.json*;  
DEFAULT —  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 0.7.8.

Preloads an immutable object at configure time. The name is used a name of the global variable through which the object is available in njs code. If the name is not specified, the file name will be used instead.

```
js_preload_object map.json;
```

Here, the `map` is used as a name while accessing the preloaded object. Several `js_preload_object` directives can be specified.

## js\_set

SYNTAX: **js\_set** *\$variable function* | *module.function*;  
DEFAULT —  
CONTEXT: http, server, location

Sets an njs function for the specified variable. Since 0.4.0, a module function can be referenced.

The function is called when the variable is referenced for the first time for a given request. The exact moment depends on a phase at which the variable is referenced. This can be used to perform some logic not related to variable evaluation. For example, if the variable is referenced only in the [log\\_format](#) directive, its handler will not be executed until the log phase. This handler can be used to do some cleanup right before the request is freed.

As the `js_set` handler returns its result immediately, it supports only synchronous operations. Thus, asynchronous operations such as `r.subrequest()` or `setTimeout()` are not supported.

The directive can be specified on the `server` and `location` level since 0.7.7.

### **js\_var**

SYNTAX: **js\_var** *\$variable* [*value*];

DEFAULT —

CONTEXT: `http`, `server`, `location`

THIS DIRECTIVE APPEARED IN VERSION 0.5.3.

Declares a writable variable. The value can contain text, variables, and their combination. The variable is not overwritten after a redirect unlike variables created with the [set](#) directive.

The directive can be specified on the `server` and `location` level since 0.7.7.

## **2.26.4 Request Argument**

Each HTTP `njs` handler receives one argument, a request object.

## 2.27 Module ngx\_http\_keyval\_module

2.27.1 Summary	198
2.27.2 Example Configuration	198
2.27.3 Directives	198
keyval	198
keyval_zone	199

### 2.27.1 Summary

The `ngx_http_keyval_module` module (1.13.3) creates variables with values taken from key-value pairs managed by the [API](#) or a variable (1.15.10) that can also be set with [njs](#).

This module is available as part of our [commercial subscription](#).

### 2.27.2 Example Configuration

```
http {
    keyval_zone zone=one:32k state=/var/lib/nginx/state/one.keyval;
    keyval $arg_text $text zone=one;
    ...
    server {
        ...
        location / {
            return 200 $text;
        }

        location /api {
            api write=on;
        }
    }
}
```

### 2.27.3 Directives

#### keyval

SYNTAX: **keyval** *key* *\$variable* zone=*name*;

DEFAULT —

CONTEXT: http

Creates a new *\$variable* whose value is looked up by the *key* in the key-value database. Matching rules are defined by the *type* parameter of the [keyval\\_zone](#) directive. The database is stored in a shared memory zone specified by the *zone* parameter.

## keyval\_zone

SYNTAX: **keyval\_zone** zone=*name:size* [*state=file*] [*timeout=time*]  
 [*type=string|ip|prefix*] [*sync*];

DEFAULT —

CONTEXT: http

Sets the *name* and *size* of the shared memory zone that keeps the key-value database. Key-value pairs are managed by the [API](#).

The optional *state* parameter specifies a *file* that keeps the current state of the key-value database in the JSON format and makes it persistent across nginx restarts. Changing the file content directly should be avoided.

Examples:

```
keyval_zone zone=one:32k state=/var/lib/nginx/state/one.keyval; # path for
Linux
keyval_zone zone=one:32k state=/var/db/nginx/state/one.keyval; # path for
FreeBSD
```

The optional *timeout* parameter (1.15.0) sets the time after which key-value pairs are removed from the zone.

The optional *type* parameter (1.17.1) activates an extra index optimized for matching the key of a certain type and defines matching rules when evaluating a [keyval](#) \$variable.

The index is stored in the same shared memory zone and thus requires additional storage.

*type=string*

default, no index is enabled; variable lookup is performed using exact match of the record key and a search key

*type=ip*

the search key is the textual representation of IPv4 or IPv6 address or CIDR range; to match a record key, the search key must belong to a subnet specified by a record key or exactly match an IP address

*type=prefix*

variable lookup is performed using prefix match of a record key and a search key (1.17.5); to match a record key, the record key must be a prefix of the search key

The optional *sync* parameter (1.15.0) enables [synchronization](#) of the shared memory zone. The synchronization requires the *timeout* parameter to be set.

If the synchronization is enabled, removal of key-value pairs (no matter [one](#) or [all](#)) will be performed only on a target cluster node. The same key-value pairs on other cluster nodes will be removed upon *timeout*.



## 2.28 Module ngx\_http\_limit\_conn\_module

2.28.1	Summary	200
2.28.2	Example Configuration	200
2.28.3	Directives	200
	<a href="#">limit_conn</a>	200
	<a href="#">limit_conn_dry_run</a>	201
	<a href="#">limit_conn_log_level</a>	201
	<a href="#">limit_conn_status</a>	202
	<a href="#">limit_conn_zone</a>	202
	<a href="#">limit_zone</a>	202
2.28.4	Embedded Variables	203

### 2.28.1 Summary

The `ngx_http_limit_conn_module` module is used to limit the number of connections per the defined key, in particular, the number of connections from a single IP address.

Not all connections are counted. A connection is counted only if it has a request being processed by the server and the whole request header has already been read.

### 2.28.2 Example Configuration

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {
        ...

        location /download/ {
            limit_conn addr 1;
        }
    }
}
```

### 2.28.3 Directives

#### **limit\_conn**

SYNTAX: **limit\_conn** *zone number*;

DEFAULT —

CONTEXT: http, server, location

Sets the shared memory zone and the maximum allowed number of connections for a given key value. When this limit is exceeded, the server will return the [error](#) in reply to a request. For example, the directives

```
limit_conn_zone $binary_remote_addr zone=addr:10m;

server {
    location /download/ {
        limit_conn addr 1;
    }
}
```

allow only one connection per an IP address at a time.

In HTTP/2 and SPDY, each concurrent request is considered a separate connection.

There could be several `limit_conn` directives. For example, the following configuration will limit the number of connections to the server per a client IP and, at the same time, the total number of connections to the virtual server:

```
limit_conn_zone $binary_remote_addr zone=perip:10m;
limit_conn_zone $server_name zone=perserver:10m;

server {
    ...
    limit_conn perip 10;
    limit_conn perserver 100;
}
```

These directives are inherited from the previous configuration level if and only if there are no `limit_conn` directives defined on the current level.

### limit\_conn\_dry\_run

SYNTAX: **limit\_conn\_dry\_run** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.17.6.

Enables the dry run mode. In this mode, the number of connections is not limited, however, in the shared memory zone, the number of excessive connections is accounted as usual.

### limit\_conn\_log\_level

SYNTAX: **limit\_conn\_log\_level** info | notice | warn | error;

DEFAULT error

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.8.18.

Sets the desired logging level for cases when the server limits the number of connections.

## limit\_conn\_status

SYNTAX: **limit\_conn\_status** *code*;

DEFAULT 503

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.3.15.

Sets the status code to return in response to rejected requests.

## limit\_conn\_zone

SYNTAX: **limit\_conn\_zone** *key* zone=*name:size*;

DEFAULT —

CONTEXT: http

Sets parameters for a shared memory zone that will keep states for various keys. In particular, the state includes the current number of connections. The *key* can contain text, variables, and their combination. Requests with an empty key value are not accounted.

Prior to version 1.7.6, a *key* could contain exactly one variable.

Usage example:

```
limit_conn_zone $binary_remote_addr zone=addr:10m;
```

Here, a client IP address serves as a key. Note that instead of *\$remote\_addr*, the *\$binary\_remote\_addr* variable is used here. The *\$remote\_addr* variable's size can vary from 7 to 15 bytes. The stored state occupies either 32 or 64 bytes of memory on 32-bit platforms and always 64 bytes on 64-bit platforms. The *\$binary\_remote\_addr* variable's size is always 4 bytes for IPv4 addresses or 16 bytes for IPv6 addresses. The stored state always occupies 32 or 64 bytes on 32-bit platforms and 64 bytes on 64-bit platforms. One megabyte zone can keep about 32 thousand 32-byte states or about 16 thousand 64-byte states. If the zone storage is exhausted, the server will return the [error](#) to all further requests.

Additionally, as part of our [commercial subscription](#), the [status information](#) for each such shared memory zone can be [obtained](#) or [reset](#) with the [API](#) since 1.17.7.

## limit\_zone

SYNTAX: **limit\_zone** *name* *\$variable* *size*;

DEFAULT —

CONTEXT: http

This directive was made obsolete in version 1.1.8 and was removed in version 1.7.6. An equivalent [limit\\_conn\\_zone](#) directive with a changed syntax should be used instead:

```
limit_conn_zone $variable zone=name:size;
```

## 2.28.4 Embedded Variables

*\$limit\_conn\_status*

keeps the result of limiting the number of connections (1.17.6): PASSED, REJECTED, or REJECTED\_DRY\_RUN

## 2.29 Module ngx\_http\_limit\_req\_module

2.29.1 Summary	204
2.29.2 Example Configuration	204
2.29.3 Directives	204
limit_req	204
limit_req_dry_run	205
limit_req_log_level	205
limit_req_status	206
limit_req_zone	206
2.29.4 Embedded Variables	207

### 2.29.1 Summary

The `ngx_http_limit_req_module` module (0.7.21) is used to limit the request processing rate per a defined key, in particular, the processing rate of requests coming from a single IP address. The limitation is done using the “leaky bucket” method.

### 2.29.2 Example Configuration

```
http {
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

    ...

    server {

        ...

        location /search/ {
            limit_req zone=one burst=5;
        }
    }
}
```

### 2.29.3 Directives

#### limit\_req

SYNTAX: **limit\_req** zone=*name* [*burst=number*] [*nodelay* | *delay=number*];

DEFAULT —

CONTEXT: http, server, location

Sets the shared memory zone and the maximum burst size of requests. If the requests rate exceeds the rate configured for a zone, their processing is delayed such that requests are processed at a defined rate. Excessive requests are delayed until their number exceeds the maximum burst size in which case the request is terminated with an [error](#). By default, the maximum burst size is equal to zero. For example, the directives

```
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

server {
    location /search/ {
        limit_req zone=one burst=5;
    }
}
```

allow not more than 1 request per second at an average, with bursts not exceeding 5 requests.

If delaying of excessive requests while requests are being limited is not desired, the parameter `nodelay` should be used:

```
limit_req zone=one burst=5 nodelay;
```

The `delay` parameter (1.15.7) specifies a limit at which excessive requests become delayed. Default value is zero, i.e. all excessive requests are delayed.

There could be several `limit_req` directives. For example, the following configuration will limit the processing rate of requests coming from a single IP address and, at the same time, the request processing rate by the virtual server:

```
limit_req_zone $binary_remote_addr zone=perip:10m rate=1r/s;
limit_req_zone $server_name zone=perserver:10m rate=10r/s;

server {
    ...
    limit_req zone=perip burst=5 nodelay;
    limit_req zone=perserver burst=10;
}
```

These directives are inherited from the previous configuration level if and only if there are no `limit_req` directives defined on the current level.

### limit\_req\_dry\_run

SYNTAX: **limit\_req\_dry\_run** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.17.1.

Enables the dry run mode. In this mode, requests processing rate is not limited, however, in the shared memory zone, the number of excessive requests is accounted as usual.

### limit\_req\_log\_level

SYNTAX: **limit\_req\_log\_level** info | notice | warn | error;

DEFAULT error

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.8.18.

Sets the desired logging level for cases when the server refuses to process requests due to rate exceeding, or delays request processing. Logging level for delays is one point less than for refusals; for example, if “`limit_req_log_level notice`” is specified, delays are logged with the `info` level.

### **limit\_req\_status**

SYNTAX: **limit\_req\_status** *code*;  
DEFAULT 503  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.3.15.

Sets the status code to return in response to rejected requests.

### **limit\_req\_zone**

SYNTAX: **limit\_req\_zone** *key* zone=*name:size* rate=*rate* [*sync*];  
DEFAULT —  
CONTEXT: http

Sets parameters for a shared memory zone that will keep states for various keys. In particular, the state stores the current number of excessive requests. The *key* can contain text, variables, and their combination. Requests with an empty key value are not accounted.

Prior to version 1.7.6, a *key* could contain exactly one variable.

Usage example:

```
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
```

Here, the states are kept in a 10 megabyte zone “one”, and an average request processing rate for this zone cannot exceed 1 request per second.

A client IP address serves as a key. Note that instead of *\$remote\_addr*, the *\$binary\_remote\_addr* variable is used here. The *\$binary\_remote\_addr* variable’s size is always 4 bytes for IPv4 addresses or 16 bytes for IPv6 addresses. The stored state always occupies 64 bytes on 32-bit platforms and 128 bytes on 64-bit platforms. One megabyte zone can keep about 16 thousand 64-byte states or about 8 thousand 128-byte states.

If the zone storage is exhausted, the least recently used state is removed. If even after that a new state cannot be created, the request is terminated with an [error](#).

The rate is specified in requests per second (r/s). If a rate of less than one request per second is desired, it is specified in request per minute (r/m). For example, half-request per second is 30r/m.

The `sync` parameter (1.15.3) enables [synchronization](#) of the shared memory zone.

The `sync` parameter is available as part of our [commercial subscription](#).

Additionally, as part of our [commercial subscription](#), the [status information](#) for each such shared memory zone can be [obtained](#) or [reset](#) with the [API](#) since 1.17.7.

#### 2.29.4 Embedded Variables

*\$limit\_req\_status*

keeps the result of limiting the request processing rate (1.17.6): PASSED, DELAYED, REJECTED, DELAYED\_DRY\_RUN, or REJECTED\_DRY\_RUN



## 2.30 Module ngx\_http\_log\_module

2.30.1 Summary	208
2.30.2 Example Configuration	208
2.30.3 Directives	208
access_log	208
log_format	210
open_log_file_cache	211

### 2.30.1 Summary

The ngx\_http\_log\_module module writes request logs in the specified format.

Requests are logged in the context of a location where processing ends. It may be different from the original location, if an [internal redirect](#) happens during request processing.

### 2.30.2 Example Configuration

```
log_format compression '$remote_addr - $remote_user [$time_local] '
                        '$request' $status $bytes_sent '
                        '$http_referer' '$http_user_agent' '$gzip_ratio';

access_log /spool/logs/nginx-access.log compression buffer=32k;
```

### 2.30.3 Directives

#### access\_log

SYNTAX: **access\_log** *path* [*format* [*buffer=size*] [*gzip[=level]*]  
[*flush=time*] [*if=condition*]];

SYNTAX: **access\_log** off;

DEFAULT logs/access.log combined

CONTEXT: http, server, location, if in location, limit\_except

Sets the path, format, and configuration for a buffered log write. Several logs can be specified on the same configuration level. Logging to [syslog](#) can be configured by specifying the “syslog:” prefix in the first parameter. The special value `off` cancels all `access_log` directives on the current level. If the format is not specified then the predefined “combined” format is used.

If either the `buffer` or `gzip` (1.3.10, 1.2.7) parameter is used, writes to log will be buffered.

The buffer size must not exceed the size of an atomic write to a disk file. For FreeBSD this size is unlimited.

When buffering is enabled, the data will be written to the file:

- if the next log line does not fit into the buffer;
- if the buffered data is older than specified by the `flush` parameter (1.3.10, 1.2.7);
- when a worker process is [re-opening](#) log files or is shutting down.

If the `gzip` parameter is used, then the buffered data will be compressed before writing to the file. The compression level can be set between 1 (fastest, less compression) and 9 (slowest, best compression). By default, the buffer size is equal to 64K bytes, and the compression level is set to 1. Since the data is compressed in atomic blocks, the log file can be decompressed or read by “`zcat`” at any time.

Example:

```
access_log /path/to/log.gz combined gzip flush=5m;
```

For `gzip` compression to work, `nginx` must be built with the `zlib` library.

The file path can contain variables (0.7.6+), but such logs have some constraints:

- the [user](#) whose credentials are used by worker processes should have permissions to create files in a directory with such logs;
- buffered writes do not work;
- the file is opened and closed for each log write. However, since the descriptors of frequently used files can be stored in a [cache](#), writing to the old file can continue during the time specified by the `open_log_file_-cache` directive’s `valid` parameter
- during each log write the existence of the request’s [root directory](#) is checked, and if it does not exist the log is not created. It is thus a good idea to specify both [root](#) and `access_log` on the same configuration level:

```
server {  
    root      /spool/vhost/data/$host;  
    access_log /spool/vhost/logs/$host;  
    ...  
}
```

The `if` parameter (1.7.0) enables conditional logging. A request will not be logged if the *condition* evaluates to “0” or an empty string. In the following example, the requests with response codes 2xx and 3xx will not be logged:

```
map $status $loggable {  
    ~^[23] 0;  
    default 1;  
}  
  
access_log /path/to/access.log combined if=$loggable;
```

## log\_format

SYNTAX: **log\_format** *name* [escape=default|json|none] *string* ...;

DEFAULT combined "..."

CONTEXT: http

Specifies log format.

The `escape` parameter (1.11.8) allows setting `json` or `default` characters escaping in variables, by default, `default` escaping is used. The `none` value (1.13.10) disables escaping.

For `default` escaping, characters “`"`”, “`\`”, and other characters with values less than 32 (0.7.0) or above 126 (1.1.6) are escaped as “`\xxx`”. If the variable value is not found, a hyphen (“`-`”) will be logged.

For `json` escaping, all characters not allowed in JSON [strings](#) will be escaped: characters “`"`” and “`\`” are escaped as “`\`” and “`\\`”, characters with values less than 32 are escaped as “`\n`”, “`\r`”, “`\t`”, “`\b`”, “`\f`”, or “`\u00XX`”.

The log format can contain common variables, and variables that exist only at the time of a log write:

*\$bytes\_sent*

the number of bytes sent to a client

*\$connection*

connection serial number

*\$connection\_requests*

the current number of requests made through a connection (1.1.18)

*\$msec*

time in seconds with a milliseconds resolution at the time of the log write

*\$pipe*

“`p`” if request was pipelined, “`.`” otherwise

*\$request\_length*

request length (including request line, header, and request body)

*\$request\_time*

request processing time in seconds with a milliseconds resolution; time elapsed between the first bytes were read from the client and the log write after the last bytes were sent to the client

*\$status*

response status

*\$time\_iso8601*

local time in the ISO 8601 standard format

*\$time\_local*

local time in the Common Log Format

In the modern nginx versions variables [\\$status](#) (1.3.2, 1.2.2), [\\$bytes\\_sent](#) (1.3.8, 1.2.5), [\\$connection](#) (1.3.8, 1.2.5), [\\$connection\\_requests](#) (1.3.8, 1.2.5), [\\$msec](#) (1.3.9, 1.2.6), [\\$request\\_time](#) (1.3.9, 1.2.6), [\\$pipe](#) (1.3.12, 1.2.7), [\\$request\\_length](#) (1.3.12, 1.2.7), [\\$time\\_iso8601](#) (1.3.12, 1.2.7), and [\\$time\\_local](#) (1.3.12, 1.2.7) are also available as common variables.

Header lines sent to a client have the prefix “sent\_http\_”, for example, *\$sent\_http\_content\_range*.

The configuration always includes the predefined “combined” format:

```
log_format combined '$remote_addr - $remote_user [$time_local] '
                    '$request" $status $body_bytes_sent '
                    '$http_referer" "$http_user_agent"';
```

### open\_log\_file\_cache

SYNTAX: **open\_log\_file\_cache** max=*N* [inactive=*time*] [min\_uses=*N*]  
[valid=*time*];

SYNTAX: **open\_log\_file\_cache** off;

DEFAULT off

CONTEXT: http, server, location

Defines a cache that stores the file descriptors of frequently used logs whose names contain variables. The directive has the following parameters:

max

sets the maximum number of descriptors in a cache; if the cache becomes full the least recently used (LRU) descriptors are closed

inactive

sets the time after which the cached descriptor is closed if there were no access during this time; by default, 10 seconds

min\_uses

sets the minimum number of file uses during the time defined by the *inactive* parameter to let the descriptor stay open in a cache; by default, 1

valid

sets the time after which it should be checked that the file still exists with the same name; by default, 60 seconds

off

disables caching

Usage example:

```
open_log_file_cache max=1000 inactive=20s valid=1m min_uses=2;
```

## 2.31 Module ngx\_http\_map\_module

2.31.1 Summary	212
2.31.2 Example Configuration	212
2.31.3 Directives	212
map	212
map_hash_bucket_size	214
map_hash_max_size	214

### 2.31.1 Summary

The `ngx_http_map_module` module creates variables whose values depend on values of other variables.

### 2.31.2 Example Configuration

```
map $http_host $name {
    hostnames;

    default      0;

    example.com  1;
    *.example.com 1;
    example.org  2;
    *.example.org 2;
    .example.net 3;
    wap.*        4;
}

map $http_user_agent $mobile {
    default      0;
    "~Opera Mini" 1;
}
```

### 2.31.3 Directives

#### map

SYNTAX: **map** *string* *\$variable* { ... }

DEFAULT —

CONTEXT: http

Creates a new variable whose value depends on values of one or more of the source variables specified in the first parameter.

Before version 0.9.0 only a single variable could be specified in the first parameter.

Since variables are evaluated only when they are used, the mere declaration even of a large number of “map” variables does not add any extra costs to request processing.

Parameters inside the map block specify a mapping between source and resulting values.

Source values are specified as strings or regular expressions (0.9.6).

Strings are matched ignoring the case.

A regular expression should either start from the “~” symbol for a case-sensitive matching, or from the “~\*” symbols (1.0.4) for case-insensitive matching. A regular expression can contain named and positional captures that can later be used in other directives along with the resulting variable.

If a source value matches one of the names of special parameters described below, it should be prefixed with the “\” symbol.

The resulting value can contain text, variable (0.9.0), and their combination (1.11.0).

The following special parameters are also supported:

`default value`

sets the resulting value if the source value matches none of the specified variants. When `default` is not specified, the default resulting value will be an empty string.

`hostnames`

indicates that source values can be hostnames with a prefix or suffix mask:

```
*.example.com 1;  
example.*     1;
```

The following two records

```
example.com 1;  
*.example.com 1;
```

can be combined:

```
.example.com 1;
```

This parameter should be specified before the list of values.

`include file`

includes a file with values. There can be several inclusions.

`volatile`

indicates that the variable is not cacheable (1.11.7).

If the source value matches more than one of the specified variants, e.g. both a mask and a regular expression match, the first matching variant will be chosen, in the following order of priority:

1. string value without a mask
2. longest string value with a prefix mask, e.g. “\*.example.com”
3. longest string value with a suffix mask, e.g. “mail.\*”

4. first matching regular expression (in order of appearance in a configuration file)
5. default value

### **map\_hash\_bucket\_size**

SYNTAX: **map\_hash\_bucket\_size** *size*;  
DEFAULT 32|64|128  
CONTEXT: http

Sets the bucket size for the [map](#) variables hash tables. Default value depends on the processor's cache line size. The details of setting up hash tables are provided in a separate [document](#).

### **map\_hash\_max\_size**

SYNTAX: **map\_hash\_max\_size** *size*;  
DEFAULT 2048  
CONTEXT: http

Sets the maximum *size* of the [map](#) variables hash tables. The details of setting up hash tables are provided in a separate [document](#).

## 2.32 Module ngx\_http\_memcached\_module

2.32.1 Summary	215
2.32.2 Example Configuration	215
2.32.3 Directives	215
memcached_bind	215
memcached_buffer_size	216
memcached_connect_timeout	216
memcached_gzip_flag	216
memcached_next_upstream	217
memcached_next_upstream_timeout	217
memcached_next_upstream_tries	217
memcached_pass	218
memcached_read_timeout	218
memcached_send_timeout	218
memcached_socket_keepalive	218
2.32.4 Embedded Variables	219

### 2.32.1 Summary

The `ngx_http_memcached_module` module is used to obtain responses from a memcached server. The key is set in the `$memcached_key` variable. A response should be put in memcached in advance by means external to nginx.

### 2.32.2 Example Configuration

```
server {
    location / {
        set          $memcached_key "$uri?$args";
        memcached_pass host:11211;
        error_page    404 502 504 = @fallback;
    }

    location @fallback {
        proxy_pass    http://backend;
    }
}
```

### 2.32.3 Directives

#### memcached\_bind

SYNTAX: **memcached\_bind** *address* [*transparent* ] | *off*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.8.22.

Makes outgoing connections to a memcached server originate from the specified local IP address with an optional port (1.11.2). Parameter value can contain variables (1.3.12). The special value `off` (1.3.12) cancels the effect



of the `memcached_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address and port.

The `transparent` parameter (1.11.0) allows outgoing connections to a memcached server originate from a non-local IP address, for example, from a real IP address of a client:

```
memcached_bind $remote_addr transparent;
```

In order for this parameter to work, it is usually necessary to run nginx worker processes with the `superuser` privileges. On Linux it is not required (1.13.8) as if the `transparent` parameter is specified, worker processes inherit the `CAP_NET_RAW` capability from the master process. It is also necessary to configure kernel routing table to intercept network traffic from the memcached server.

### **memcached\_buffer\_size**

SYNTAX: **memcached\_buffer\_size** *size*;

DEFAULT 4k | 8k

CONTEXT: http, server, location

Sets the *size* of the buffer used for reading the response received from the memcached server. The response is passed to the client synchronously, as soon as it is received.

### **memcached\_connect\_timeout**

SYNTAX: **memcached\_connect\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for establishing a connection with a memcached server. It should be noted that this timeout cannot usually exceed 75 seconds.

### **memcached\_gzip\_flag**

SYNTAX: **memcached\_gzip\_flag** *flag*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.3.6.

Enables the test for the *flag* presence in the memcached server response and sets the “Content-Encoding” response header field to “gzip” if the flag is set.

## memcached\_next\_upstream

SYNTAX: **memcached\_next\_upstream** error | timeout |  
invalid\_response | not\_found | off ...;

DEFAULT error timeout

CONTEXT: http, server, location

Specifies in which cases a request should be passed to the next server:

error

an error occurred while establishing a connection with the server, passing a request to it, or reading the response header;

timeout

a timeout has occurred while establishing a connection with the server, passing a request to it, or reading the response header;

invalid\_response

a server returned an empty or invalid response;

not\_found

a response was not found on the server;

off

disables passing a request to the next server.

One should bear in mind that passing a request to the next server is only possible if nothing has been sent to a client yet. That is, if an error or timeout occurs in the middle of the transferring of a response, fixing this is impossible.

The directive also defines what is considered an [unsuccessful attempt](#) of communication with a server. The cases of `error`, `timeout` and `invalid_response` are always considered unsuccessful attempts, even if they are not specified in the directive. The case of `not_found` is never considered an unsuccessful attempt.

Passing a request to the next server can be limited by [the number of tries](#) and by [time](#).

## memcached\_next\_upstream\_timeout

SYNTAX: **memcached\_next\_upstream\_timeout** *time*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.5.

Limits the time during which a request can be passed to the [next server](#). The 0 value turns off this limitation.

## memcached\_next\_upstream\_tries

SYNTAX: **memcached\_next\_upstream\_tries** *number*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.5.

Limits the number of possible tries for passing a request to the [next server](#). The 0 value turns off this limitation.

### memcached\_pass

SYNTAX: **memcached\_pass** *address*;

DEFAULT —

CONTEXT: location, if in location

Sets the memcached server address. The address can be specified as a domain name or IP address, and a port:

```
memcached_pass localhost:11211;
```

or as a UNIX-domain socket path:

```
memcached_pass unix:/tmp/memcached.socket;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a [server group](#).

### memcached\_read\_timeout

SYNTAX: **memcached\_read\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for reading a response from the memcached server. The timeout is set only between two successive read operations, not for the transmission of the whole response. If the memcached server does not transmit anything within this time, the connection is closed.

### memcached\_send\_timeout

SYNTAX: **memcached\_send\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Sets a timeout for transmitting a request to the memcached server. The timeout is set only between two successive write operations, not for the transmission of the whole request. If the memcached server does not receive anything within this time, the connection is closed.

### memcached\_socket\_keepalive

SYNTAX: **memcached\_socket\_keepalive** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.15.6.

Configures the “TCP keepalive” behavior for outgoing connections to a memcached server. By default, the operating system’s settings are in effect for the socket. If the directive is set to the value “on”, the `SO_KEEPALIVE` socket option is turned on for the socket.

#### 2.32.4 Embedded Variables

*\$memcached\_key*

Defines a key for obtaining response from a memcached server.

## 2.33 Module ngx\_http\_mirror\_module

2.33.1 Summary	220
2.33.2 Example Configuration	220
2.33.3 Directives	220
mirror	220
mirror_request_body	220

### 2.33.1 Summary

The `ngx_http_mirror_module` module (1.13.4) implements mirroring of an original request by creating background mirror subrequests. Responses to mirror subrequests are ignored.

### 2.33.2 Example Configuration

```
location / {
    mirror /mirror;
    proxy_pass http://backend;
}

location = /mirror {
    internal;
    proxy_pass http://test_backend$request_uri;
}
```

### 2.33.3 Directives

#### mirror

SYNTAX: **mirror** *uri* | off;  
 DEFAULT off  
 CONTEXT: http, server, location

Sets the URI to which an original request will be mirrored. Several mirrors can be specified on the same configuration level.

#### mirror\_request\_body

SYNTAX: **mirror\_request\_body** on | off;  
 DEFAULT on  
 CONTEXT: http, server, location

Indicates whether the client request body is mirrored. When enabled, the client request body will be read prior to creating mirror subrequests. In this case, unbuffered client request body proxying set by the [proxy\\_request\\_buffering](#), [fastcgi\\_request\\_buffering](#), [scgi\\_request\\_buffering](#), and [uwsgi\\_request\\_buffering](#) directives will be disabled.

```
location / {
    mirror /mirror;
    mirror_request_body off;
    proxy_pass http://backend;
}

location = /mirror {
    internal;
    proxy_pass http://log_backend;
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";
    proxy_set_header X-Original-URI $request_uri;
}
```

## 2.34 Module ngx\_http\_mp4\_module

2.34.1	Summary	222
2.34.2	Example Configuration	223
2.34.3	Directives	223
	mp4	223
	mp4_buffer_size	223
	mp4_max_buffer_size	224
	mp4_limit_rate	224
	mp4_limit_rate_after	224
	mp4_start_key_frame	224

### 2.34.1 Summary

The `ngx_http_mp4_module` module provides pseudo-streaming server-side support for MP4 files. Such files typically have the `.mp4`, `.m4v`, or `.m4a` filename extensions.

Pseudo-streaming works in alliance with a compatible media player. The player sends an HTTP request to the server with the start time specified in the query string argument (named simply `start` and specified in seconds), and the server responds with the stream such that its start position corresponds to the requested time, for example:

```
http://example.com/elephants_dream.mp4?start=238.88
```

This allows performing a random seeking at any time, or starting playback in the middle of the timeline.

To support seeking, H.264-based formats store metadata in a so-called “moov atom”. It is a part of the file that holds the index information for the whole file.

To start playback, the player first needs to read metadata. This is done by sending a special request with the `start=0` argument. A lot of encoding software insert the metadata at the end of the file. This is suboptimal for pseudo-streaming, because the player has to download the entire file before starting playback. If the metadata are located at the beginning of the file, it is enough for nginx to simply start sending back the file contents. If the metadata are located at the end of the file, nginx must read the entire file and prepare a new stream so that the metadata come before the media data. This involves some CPU, memory, and disk I/O overhead, so it is a good idea to [prepare an original file for pseudo-streaming](#) in advance, rather than having nginx do this on every such request.

The module also supports the `end` argument of an HTTP request (1.5.13) which sets the end point of playback. The `end` argument can be specified with the `start` argument or separately:

```
http://example.com/elephants_dream.mp4?start=238.88&end=555.55
```

For a matching request with a non-zero `start` or `end` argument, nginx will read the metadata from the file, prepare the stream with the requested time range, and send it to the client. This has the same overhead as described above.

If the `start` argument points to a non-key video frame, the beginning of such video will be broken. To fix this issue, the video [can](#) be prepended with the key frame before `start` point and with all intermediate frames between them. These frames will be hidden from playback using an edit list (1.21.4).

If a matching request does not include the `start` and `end` arguments, there is no overhead, and the file is sent simply as a static resource. Some players also support byte-range requests, and thus do not require this module.

This module is not built by default, it should be enabled with the `--with-http_mp4_module` configuration parameter.

If a third-party mp4 module was previously used, it should be disabled.

A similar pseudo-streaming support for FLV files is provided by the [ngx-http\\_flv\\_module](#) module.

## 2.34.2 Example Configuration

```
location /video/ {
    mp4;
    mp4_buffer_size          1m;
    mp4_max_buffer_size      5m;
    mp4_limit_rate           on;
    mp4_limit_rate_after     30s;
}
```

## 2.34.3 Directives

### mp4

SYNTAX: **mp4**;

DEFAULT —

CONTEXT: location

Turns on module processing in a surrounding location.

### mp4\_buffer\_size

SYNTAX: **mp4\_buffer\_size** *size*;

DEFAULT 512K

CONTEXT: http, server, location

Sets the initial *size* of the buffer used for processing MP4 files.



## mp4\_max\_buffer\_size

SYNTAX: **mp4\_max\_buffer\_size** *size*;

DEFAULT 10M

CONTEXT: http, server, location

During metadata processing, a larger buffer may become necessary. Its size cannot exceed the specified *size*, or else nginx will return the 500 Internal Server Error server error, and log the following message:

```
"/some/movie/file.mp4" mp4 moov atom is too large:
12583268, you may want to increase mp4_max_buffer_size
```

## mp4\_limit\_rate

SYNTAX: **mp4\_limit\_rate** on | off | *factor*;

DEFAULT off

CONTEXT: http, server, location

Limits the rate of response transmission to a client. The rate is limited based on the average bitrate of the MP4 file served. To calculate the rate, the bitrate is multiplied by the specified *factor*. The special value “on” corresponds to the factor of 1.1. The special value “off” disables rate limiting. The limit is set per a request, and so if a client simultaneously opens two connections, the overall rate will be twice as much as the specified limit.

This directive is available as part of our [commercial subscription](#).

## mp4\_limit\_rate\_after

SYNTAX: **mp4\_limit\_rate\_after** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Sets the initial amount of media data (measured in playback time) after which the further transmission of the response to a client will be rate limited.

This directive is available as part of our [commercial subscription](#).

## mp4\_start\_key\_frame

SYNTAX: **mp4\_start\_key\_frame** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.21.4.

Forces output video to always start with a key video frame. If the `start` argument does not point to a key frame, initial frames are hidden using an mp4 edit list. Edit lists are supported by major players and browsers such as Chrome, Safari, QuickTime and ffmpeg, partially supported by Firefox.

## 2.35 Module ngx\_http\_perl\_module

2.35.1	Summary	225
2.35.2	Known Issues	225
2.35.3	Example Configuration	226
2.35.4	Directives	226
	perl	226
	perl_modules	227
	perl_require	227
	perl_set	227
2.35.5	Calling Perl from SSI	227
2.35.6	The \$r Request Object Methods	227

### 2.35.1 Summary

The `ngx_http_perl_module` module is used to implement location and variable handlers in Perl and insert Perl calls into SSI.

This module is not built by default, it should be enabled with the `--with-http_perl_module` configuration parameter.

This module requires [Perl](#) version 5.6.1 or higher. The C compiler should be compatible with the one used to build Perl.

### 2.35.2 Known Issues

The module is experimental, caveat emptor applies.

In order for Perl to recompile the modified modules during reconfiguration, it should be built with the `-Dusemultiplicity=yes` or `-Dusethreads=yes` parameters. Also, to make Perl leak less memory at run time, it should be built with the `-Dusymalloc=no` parameter. To check the values of these parameters in an already built Perl (preferred values are specified in the example), run:

```
$ perl -V:usemultiplicity -V:usymalloc
usemultiplicity='define';
usymalloc='n';
```

Note that after rebuilding Perl with the new `-Dusemultiplicity=yes` or `-Dusethreads=yes` parameters, all binary Perl modules will have to be rebuilt as well — they will just stop working with the new Perl.

There is a possibility that the main process and then worker processes will grow in size after every reconfiguration. If the main process grows to an unacceptable size, the [live upgrade](#) procedure can be applied without changing the executable file.

While the Perl module is performing a long-running operation, such as resolving a domain name, connecting to another server, or querying a database, other requests assigned to the current worker process will not be processed. It

is thus recommended to perform only such operations that have predictable and short execution time, such as accessing the local file system.

### 2.35.3 Example Configuration

```
http {

    perl_modules perl/lib;
    perl_require hello.pm;

    perl_set $msie6 '

        sub {
            my $r = shift;
            my $ua = $r->header_in("User-Agent");

            return "" if $ua =~ /Opera/;
            return "1" if $ua =~ / MSIE [6-9]\.\d+\/;
            return "";
        }

    ';

    server {
        location / {
            perl hello::handler;
        }
    }
}
```

The `perl/lib/hello.pm` module:

```
package hello;

use nginx;

sub handler {
    my $r = shift;

    $r->send_http_header("text/html");
    return OK if $r->header_only;

    $r->print("hello!\n<br/>");

    if (-f $r->filename or -d _) {
        $r->print($r->uri, " exists!\n");
    }

    return OK;
}

1;
__END__
```

### 2.35.4 Directives

#### perl

SYNTAX: **perl** *module::function*['sub { ... }'];

DEFAULT —

CONTEXT: location, limit\_except

Sets a Perl handler for the given location.

### **perl\_modules**

SYNTAX: **perl\_modules** *path*;

DEFAULT —

CONTEXT: http

Sets an additional path for Perl modules.

### **perl\_require**

SYNTAX: **perl\_require** *module*;

DEFAULT —

CONTEXT: http

Defines the name of a module that will be loaded during each reconfiguration. Several `perl_require` directives can be present.

### **perl\_set**

SYNTAX: **perl\_set** *\$variable* *module::function*['sub { ... }'];

DEFAULT —

CONTEXT: http

Installs a Perl handler for the specified variable.

## **2.35.5 Calling Perl from SSI**

An SSI command calling Perl has the following format:

```
<!--# perl sub="module::function" arg="parameter1" arg="parameter2" ...  
-->
```

## **2.35.6 The \$r Request Object Methods**

`$r->args`

returns request arguments.

`$r->filename`

returns a filename corresponding to the request URI.

`$r->has_request_body(handler)`

returns 0 if there is no body in a request. If there is a body, the specified handler is set for the request and 1 is returned. After reading the request body, nginx will call the specified handler. Note that the handler function should be passed by reference. Example:

```
package hello;  
  
use nginx;
```

```

sub handler {
    my $r = shift;

    if ($r->request_method ne "POST") {
        return DECLINED;
    }

    if ($r->has_request_body(\&post)) {
        return OK;
    }

    return HTTP_BAD_REQUEST;
}

sub post {
    my $r = shift;

    $r->send_http_header;

    $r->print("request_body: \"", $r->request_body, "\"<br/>");
    $r->print("request_body_file: \"", $r->request_body_file, "\"<br/>\n");

    return OK;
}

1;

__END__

```

`$r->allow_ranges`

enables the use of byte ranges when sending responses.

`$r->discard_request_body`

instructs nginx to discard the request body.

`$r->header_in(field)`

returns the value of the specified client request header field.

`$r->header_only`

determines whether the whole response or only its header should be sent to the client.

`$r->header_out(field, value)`

sets a value for the specified response header field.

`$r->internal_redirect(uri)`

does an internal redirect to the specified *uri*. An actual redirect happens after the Perl handler execution is completed.

Since version 1.17.2, the method accepts escaped URIs and supports redirections to named locations.

`$r->log_error(errno, message)`

writes the specified *message* into the [error\\_log](#). If *errno* is non-zero, an error code and its description will be appended to the message.

`$r->print(text, ...)`

passes data to a client.

`$r->request_body`

returns the client request body if it has not been written to a temporary file. To ensure that the client request body is in memory, its size should

be limited by `client_max_body_size`, and a sufficient buffer size should be set using `client_body_buffer_size`.

`$r->request_body_file`

returns the name of the file with the client request body. After the processing, the file should be removed. To always write a request body to a file, `client_body_in_file_only` should be enabled.

`$r->request_method`

returns the client request HTTP method.

`$r->remote_addr`

returns the client IP address.

`$r->flush`

immediately sends data to the client.

`$r->sendfile(name[, offset[, length]])`

sends the specified file content to the client. Optional parameters specify the initial offset and length of the data to be transmitted. The actual data transmission happens after the Perl handler has completed.

`$r->send_http_header([type])`

sends the response header to the client. The optional *type* parameter sets the value of the Content-Type response header field. If the value is an empty string, the Content-Type header field will not be sent.

`$r->status(code)`

sets a response code.

`$r->sleep(milliseconds, handler)`

sets the specified handler and stops request processing for the specified time. In the meantime, nginx continues to process other requests. After the specified time has elapsed, nginx will call the installed handler. Note that the handler function should be passed by reference. In order to pass data between handlers, `$r->variable()` should be used. Example:

```
package hello;

use nginx;

sub handler {
    my $r = shift;

    $r->discard_request_body;
    $r->variable("var", "OK");
    $r->sleep(1000, \&next);

    return OK;
}

sub next {
    my $r = shift;

    $r->send_http_header;
    $r->print($r->variable("var"));

    return OK;
}

1;

__END__
```

`$r->unescape(text)`  
decodes a text encoded in the “%XX” form.

`$r->uri`  
returns a request URI.

`$r->variable(name[, value])`  
returns or sets the value of the specified variable. Variables are local to each request.

## 2.36 Module ngx\_http\_proxy\_module

2.36.1	Summary	232
2.36.2	Example Configuration	232
2.36.3	Directives	233
	proxy_bind	233
	proxy_buffer_size	233
	proxy_buffering	233
	proxy_buffers	234
	proxy_busy_buffers_size	234
	proxy_cache	234
	proxy_cache_background_update	234
	proxy_cache_bypass	235
	proxy_cache_convert_head	235
	proxy_cache_key	235
	proxy_cache_lock	235
	proxy_cache_lock_age	236
	proxy_cache_lock_timeout	236
	proxy_cache_max_range_offset	236
	proxy_cache_methods	236
	proxy_cache_min_uses	237
	proxy_cache_path	237
	proxy_cache_purge	239
	proxy_cache_revalidate	239
	proxy_cache_use_stale	240
	proxy_cache_valid	240
	proxy_connect_timeout	241
	proxy_cookie_domain	241
	proxy_cookie_flags	242
	proxy_cookie_path	243
	proxy_force_ranges	243
	proxy_headers_hash_bucket_size	244
	proxy_headers_hash_max_size	244
	proxy_hide_header	244
	proxy_http_version	244
	proxy_ignore_client_abort	244
	proxy_ignore_headers	245
	proxy_intercept_errors	245
	proxy_limit_rate	245
	proxy_max_temp_file_size	246
	proxy_method	246
	proxy_next_upstream	246
	proxy_next_upstream_timeout	247
	proxy_next_upstream_tries	247
	proxy_no_cache	248
	proxy_pass	248



proxy_pass_header	249
proxy_pass_request_body	250
proxy_pass_request_headers	250
proxy_read_timeout	250
proxy_redirect	250
proxy_request_buffering	252
proxy_send_lowat	252
proxy_send_timeout	252
proxy_set_body	253
proxy_set_header	253
proxy_socket_keepalive	254
proxy_ssl_certificate	254
proxy_ssl_certificate_key	254
proxy_ssl_ciphers	254
proxy_ssl_conf_command	255
proxy_ssl_crl	255
proxy_ssl_name	255
proxy_ssl_password_file	255
proxy_ssl_protocols	256
proxy_ssl_server_name	256
proxy_ssl_session_reuse	256
proxy_ssl_trusted_certificate	256
proxy_ssl_verify	256
proxy_ssl_verify_depth	257
proxy_store	257
proxy_store_access	258
proxy_temp_file_write_size	258
proxy_temp_path	258
2.36.4 Embedded Variables	259

### 2.36.1 Summary

The `ngx_http_proxy_module` module allows passing requests to another server.

### 2.36.2 Example Configuration

```
location / {
    proxy_pass      http://localhost:8000;
    proxy_set_header Host      $host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

## 2.36.3 Directives

### proxy\_bind

SYNTAX: **proxy\_bind** *address* [transparent] | off;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.8.22.

Makes outgoing connections to a proxied server originate from the specified local IP address with an optional port (1.11.2). Parameter value can contain variables (1.3.12). The special value `off` (1.3.12) cancels the effect of the `proxy_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address and port.

The `transparent` parameter (1.11.0) allows outgoing connections to a proxied server originate from a non-local IP address, for example, from a real IP address of a client:

```
proxy_bind $remote_addr transparent;
```

In order for this parameter to work, it is usually necessary to run nginx worker processes with the [superuser](#) privileges. On Linux it is not required (1.13.8) as if the `transparent` parameter is specified, worker processes inherit the `CAP_NET_RAW` capability from the master process. It is also necessary to configure kernel routing table to intercept network traffic from the proxied server.

### proxy\_buffer\_size

SYNTAX: **proxy\_buffer\_size** *size*;

DEFAULT 4k | 8k

CONTEXT: http, server, location

Sets the *size* of the buffer used for reading the first part of the response received from the proxied server. This part usually contains a small response header. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform. It can be made smaller, however.

### proxy\_buffering

SYNTAX: **proxy\_buffering** on | off;

DEFAULT on

CONTEXT: http, server, location

Enables or disables buffering of responses from the proxied server.

When buffering is enabled, nginx receives a response from the proxied server as soon as possible, saving it into the buffers set by the [proxy\\_buffer\\_size](#) and [proxy\\_buffers](#) directives. If the whole response does not fit into memory, a part of it can be saved to a [temporary file](#) on the disk. Writing to temporary files

is controlled by the [proxy\\_max\\_temp\\_file\\_size](#) and [proxy\\_temp\\_file\\_write\\_size](#) directives.

When buffering is disabled, the response is passed to a client synchronously, immediately as it is received. nginx will not try to read the whole response from the proxied server. The maximum size of the data that nginx can receive from the server at a time is set by the [proxy\\_buffer\\_size](#) directive.

Buffering can also be enabled or disabled by passing “yes” or “no” in the X-Accel-Buffering response header field. This capability can be disabled using the [proxy\\_ignore\\_headers](#) directive.

### proxy\_buffers

SYNTAX: **proxy\_buffers** *number size*;  
DEFAULT 8 4k|8k  
CONTEXT: http, server, location

Sets the *number* and *size* of the buffers used for reading a response from the proxied server, for a single connection. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

### proxy\_busy\_buffers\_size

SYNTAX: **proxy\_busy\_buffers\_size** *size*;  
DEFAULT 8k|16k  
CONTEXT: http, server, location

When [buffering](#) of responses from the proxied server is enabled, limits the total *size* of buffers that can be busy sending a response to the client while the response is not yet fully read. In the meantime, the rest of the buffers can be used for reading the response and, if needed, buffering part of the response to a temporary file. By default, *size* is limited by the size of two buffers set by the [proxy\\_buffer\\_size](#) and [proxy\\_buffers](#) directives.

### proxy\_cache

SYNTAX: **proxy\_cache** *zone* | off;  
DEFAULT off  
CONTEXT: http, server, location

Defines a shared memory zone used for caching. The same zone can be used in several places. Parameter value can contain variables (1.7.9). The `off` parameter disables caching inherited from the previous configuration level.

### proxy\_cache\_background\_update

SYNTAX: **proxy\_cache\_background\_update** on | off;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.11.10.

Allows starting a background subrequest to update an expired cache item, while a stale cached response is returned to the client. Note that it is necessary to allow the usage of a stale cached response when it is being updated.

### proxy\_cache\_bypass

SYNTAX: **proxy\_cache\_bypass** *string* ...;

DEFAULT —

CONTEXT: http, server, location

Defines conditions under which the response will not be taken from a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be taken from the cache:

```
proxy_cache_bypass $cookie_nocache $arg_nocache$arg_comment;
proxy_cache_bypass $http_pragma $http_authorization;
```

Can be used along with the [proxy\\_no\\_cache](#) directive.

### proxy\_cache\_convert\_head

SYNTAX: **proxy\_cache\_convert\_head** on | off;

DEFAULT on

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.9.7.

Enables or disables the conversion of the “HEAD” method to “GET” for caching. When the conversion is disabled, the [cache key](#) should be configured to include the *\$request\_method*.

### proxy\_cache\_key

SYNTAX: **proxy\_cache\_key** *string*;

DEFAULT *\$scheme\$proxy\_host\$request\_uri*

CONTEXT: http, server, location

Defines a key for caching, for example

```
proxy_cache_key "$host$request_uri $cookie_user";
```

By default, the directive’s value is close to the string

```
proxy_cache_key $scheme$proxy_host$uri$is_args$args;
```

### proxy\_cache\_lock

SYNTAX: **proxy\_cache\_lock** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.12.

When enabled, only one request at a time will be allowed to populate a new cache element identified according to the [proxy\\_cache\\_key](#) directive by passing a request to a proxied server. Other requests of the same cache element will either wait for a response to appear in the cache or the cache lock for this element to be released, up to the time set by the [proxy\\_cache\\_lock\\_timeout](#) directive.

### proxy\_cache\_lock\_age

SYNTAX: **proxy\_cache\_lock\_age** *time*;

DEFAULT 5s

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.8.

If the last request passed to the proxied server for populating a new cache element has not completed for the specified *time*, one more request may be passed to the proxied server.

### proxy\_cache\_lock\_timeout

SYNTAX: **proxy\_cache\_lock\_timeout** *time*;

DEFAULT 5s

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.12.

Sets a timeout for [proxy\\_cache\\_lock](#). When the *time* expires, the request will be passed to the proxied server, however, the response will not be cached.

Before 1.7.8, the response could be cached.

### proxy\_cache\_max\_range\_offset

SYNTAX: **proxy\_cache\_max\_range\_offset** *number*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.11.6.

Sets an offset in bytes for byte-range requests. If the range is beyond the offset, the range request will be passed to the proxied server and the response will not be cached.

### proxy\_cache\_methods

SYNTAX: **proxy\_cache\_methods** GET | HEAD | POST ...;

DEFAULT GET HEAD

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 0.7.59.

If the client request method is listed in this directive then the response will be cached. “GET” and “HEAD” methods are always added to the list, though

it is recommended to specify them explicitly. See also the [proxy\\_no\\_cache](#) directive.

### proxy\_cache\_min\_uses

SYNTAX: **proxy\_cache\_min\_uses** *number*;

DEFAULT 1

CONTEXT: http, server, location

Sets the *number* of requests after which the response will be cached.

### proxy\_cache\_path

SYNTAX: **proxy\_cache\_path** *path* [*levels=levels*]  
 [*use\_temp\_path=on|off*] *keys\_zone=name:size* [*inactive=time*]  
 [*max\_size=size*] [*min\_free=size*] [*manager\_files=number*]  
 [*manager\_sleep=time*] [*manager\_threshold=time*]  
 [*loader\_files=number*] [*loader\_sleep=time*]  
 [*loader\_threshold=time*] [*purger=on|off*]  
 [*purger\_files=number*] [*purger\_sleep=time*]  
 [*purger\_threshold=time*];

DEFAULT —

CONTEXT: http

Sets the path and other parameters of a cache. Cache data are stored in files. The file name in a cache is a result of applying the MD5 function to the [cache key](#). The *levels* parameter defines hierarchy levels of a cache: from 1 to 3, each level accepts values 1 or 2. For example, in the following configuration

```
proxy_cache_path /data/nginx/cache levels=1:2 keys_zone=one:10m;
```

file names in a cache will look like this:

```
/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

A cached response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the cache can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both cache and a directory holding temporary files are put on the same file system. The directory for temporary files is set based on the *use\_temp\_path* parameter (1.7.10). If this parameter is omitted or set to the value *on*, the directory set by the [proxy\\_temp\\_path](#) directive for the given location will be used. If the value is set to *off*, temporary files will be put directly in the cache directory.

In addition, all active keys and information about data are stored in a shared memory zone, whose *name* and *size* are configured by the *keys\_zone* parameter. One megabyte zone can store about 8 thousand keys.

As part of [commercial subscription](#), the shared memory zone also stores extended cache [information](#), thus, it is required to specify a larger zone size for the same number of keys. For example, one megabyte zone can store about 4 thousand keys.

Cached data that are not accessed during the time specified by the `inactive` parameter get removed from the cache regardless of their freshness. By default, `inactive` is set to 10 minutes.

The special “cache manager” process monitors the maximum cache size set by the `max_size` parameter, and the minimum amount of free space set by the `min_free` (1.19.1) parameter on the file system with cache. When the size is exceeded or there is not enough free space, it removes the least recently used data. The data is removed in iterations configured by `manager_files`, `manager_threshold`, and `manager_sleep` parameters (1.11.5). During one iteration no more than `manager_files` items are deleted (by default, 100). The duration of one iteration is limited by the `manager_threshold` parameter (by default, 200 milliseconds). Between iterations, a pause configured by the `manager_sleep` parameter (by default, 50 milliseconds) is made.

A minute after the start the special “cache loader” process is activated. It loads information about previously cached data stored on file system into a cache zone. The loading is also done in iterations. During one iteration no more than `loader_files` items are loaded (by default, 100). Besides, the duration of one iteration is limited by the `loader_threshold` parameter (by default, 200 milliseconds). Between iterations, a pause configured by the `loader_sleep` parameter (by default, 50 milliseconds) is made.

Additionally, the following parameters are available as part of our [commercial subscription](#):

`purger=on|off`

Instructs whether cache entries that match a [wildcard key](#) will be removed from the disk by the cache purger (1.7.12). Setting the parameter to `on` (default is `off`) will activate the “cache purger” process that permanently iterates through all cache entries and deletes the entries that match the wildcard key.

`purger_files=number`

Sets the number of items that will be scanned during one iteration (1.7.12). By default, `purger_files` is set to 10.

`purger_threshold=number`

Sets the duration of one iteration (1.7.12). By default, `purger_threshold` is set to 50 milliseconds.

`purger_sleep=number`

Sets a pause between iterations (1.7.12). By default, `purger_sleep` is set to 50 milliseconds.

In versions 1.7.3, 1.7.7, and 1.11.10 cache header format has been changed. Previously cached responses will be considered invalid after upgrading to a newer nginx version.

## proxy\_cache\_purge

SYNTAX: **proxy\_cache\_purge**string ...;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Defines conditions under which the request will be considered a cache purge request. If at least one value of the string parameters is not empty and is not equal to “0” then the cache entry with a corresponding [cache key](#) is removed. The result of successful operation is indicated by returning the 204 No Content response.

If the [cache key](#) of a purge request ends with an asterisk (“\*”), all cache entries matching the wildcard key will be removed from the cache. However, these entries will remain on the disk until they are deleted for either [inactivity](#), or processed by the [cache purger](#) (1.7.12), or a client attempts to access them.

Example configuration:

```
proxy_cache_path /data/nginx/cache keys_zone=cache_zone:10m;

map $request_method $purge_method {
    PURGE    1;
    default  0;
}

server {
    ...
    location / {
        proxy_pass http://backend;
        proxy_cache cache_zone;
        proxy_cache_key $uri;
        proxy_cache_purge $purge_method;
    }
}
```

This functionality is available as part of our [commercial subscription](#).

## proxy\_cache\_revalidate

SYNTAX: **proxy\_cache\_revalidate** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Enables revalidation of expired cache items using conditional requests with the If-Modified-Since and If-None-Match header fields.



## proxy\_cache\_use\_stale

SYNTAX: **proxy\_cache\_use\_stale** error | timeout | invalid\_header |  
 updating | http\_500 | http\_502 | http\_503 | http\_504 |  
 http\_403 | http\_404 | http\_429 | off ...;

DEFAULT off

CONTEXT: http, server, location

Determines in which cases a stale cached response can be used during communication with the proxied server. The directive's parameters match the parameters of the [proxy\\_next\\_upstream](#) directive.

The error parameter also permits using a stale cached response if a proxied server to process a request cannot be selected.

Additionally, the updating parameter permits using a stale cached response if it is currently being updated. This allows minimizing the number of accesses to proxied servers when updating cached data.

Using a stale cached response can also be enabled directly in the response header for a specified number of seconds after the response became stale (1.11.10). This has lower priority than using the directive parameters.

- The “[stale-while-revalidate](#)” extension of the Cache-Control header field permits using a stale cached response if it is currently being updated.
- The “[stale-if-error](#)” extension of the Cache-Control header field permits using a stale cached response in case of an error.

To minimize the number of accesses to proxied servers when populating a new cache element, the [proxy\\_cache\\_lock](#) directive can be used.

## proxy\_cache\_valid

SYNTAX: **proxy\_cache\_valid** [*code ...*] *time*;

DEFAULT —

CONTEXT: http, server, location

Sets caching time for different response codes. For example, the following directives

```
proxy_cache_valid 200 302 10m;
proxy_cache_valid 404 1m;
```

set 10 minutes of caching for responses with codes 200 and 302 and 1 minute for responses with code 404.

If only caching *time* is specified

```
proxy_cache_valid 5m;
```

then only 200, 301, and 302 responses are cached.

In addition, the any parameter can be specified to cache any responses:

```
proxy_cache_valid 200 302 10m;  
proxy_cache_valid 301      1h;  
proxy_cache_valid any      1m;
```

Parameters of caching can also be set directly in the response header. This has higher priority than setting of caching time using the directive.

- The `X-Accel-Expires` header field sets caching time of a response in seconds. The zero value disables caching for a response. If the value starts with the `@` prefix, it sets an absolute time in seconds since Epoch, up to which the response may be cached.
- If the header does not include the `X-Accel-Expires` field, parameters of caching may be set in the header fields `Expires` or `Cache-Control`.
- If the header includes the `Set-Cookie` field, such a response will not be cached.
- If the header includes the `Vary` field with the special value “\*”, such a response will not be cached (1.7.7). If the header includes the `Vary` field with another value, such a response will be cached taking into account the corresponding request header fields (1.7.7).

Processing of one or more of these response header fields can be disabled using the [proxy\\_ignore\\_headers](#) directive.

### proxy\_connect\_timeout

SYNTAX: **proxy\_connect\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for establishing a connection with a proxied server. It should be noted that this timeout cannot usually exceed 75 seconds.

### proxy\_cookie\_domain

SYNTAX: **proxy\_cookie\_domain** *off*;

SYNTAX: **proxy\_cookie\_domain** *domain replacement*;

DEFAULT *off*

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.15.

Sets a text that should be changed in the `domain` attribute of the `Set-Cookie` header fields of a proxied server response. Suppose a proxied server returned the `Set-Cookie` header field with the attribute “`domain=localhost`”. The directive

```
proxy_cookie_domain localhost example.org;
```

will rewrite this attribute to “domain=example.org”.

A dot at the beginning of the *domain* and *replacement* strings and the domain attribute is ignored. Matching is case-insensitive.

The *domain* and *replacement* strings can contain variables:

```
proxy_cookie_domain www.$host $host;
```

The directive can also be specified using regular expressions. In this case, *domain* should start from the “~” symbol. A regular expression can contain named and positional captures, and *replacement* can reference them:

```
proxy_cookie_domain ~\.(?P<sl_domain>[-0-9a-z]+\.[a-z]+)$ $sl_domain;
```

Several `proxy_cookie_domain` directives can be specified on the same level:

```
proxy_cookie_domain localhost example.org;  
proxy_cookie_domain ~\.[a-z]+\.[a-z]+$ $1;
```

If several directives can be applied to the cookie, the first matching directive will be chosen.

The `off` parameter cancels the effect of the `proxy_cookie_domain` directives inherited from the previous configuration level.

### proxy\_cookie\_flags

SYNTAX: **proxy\_cookie\_flags** `off` | *cookie* [*flag* ...];

DEFAULT `off`

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.19.3.

Sets one or more flags for the cookie. The *cookie* can contain text, variables, and their combinations. The *flag* can contain text, variables, and their combinations (1.19.8). The `secure`, `httponly`, `samesite=strict`, `samesite=lax`, `samesite=none` parameters add the corresponding flags. The `nosecure`, `nohttponly`, `nosamesite` parameters remove the corresponding flags.

The cookie can also be specified using regular expressions. In this case, *cookie* should start from the “~” symbol.

Several `proxy_cookie_flags` directives can be specified on the same configuration level:

```
proxy_cookie_flags one httponly;  
proxy_cookie_flags ~ nosecure samesite=strict;
```

If several directives can be applied to the cookie, the first matching directive will be chosen. In the example, the `httponly` flag is added to the cookie one, for all other cookies the `samesite=strict` flag is added and the `secure` flag is deleted.

The `off` parameter cancels the effect of the `proxy_cookie_flags` directives inherited from the previous configuration level.

### proxy\_cookie\_path

SYNTAX: **proxy\_cookie\_path** `off`;  
SYNTAX: **proxy\_cookie\_path** *path replacement*;  
DEFAULT `off`  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.1.15.

Sets a text that should be changed in the `path` attribute of the `Set-Cookie` header fields of a proxied server response. Suppose a proxied server returned the `Set-Cookie` header field with the attribute “`path=/two/some/uri/`”. The directive

```
proxy_cookie_path /two/ /;
```

will rewrite this attribute to “`path=/some/uri/`”.

The *path* and *replacement* strings can contain variables:

```
proxy_cookie_path $uri /some$uri;
```

The directive can also be specified using regular expressions. In this case, *path* should either start from the “`~`” symbol for a case-sensitive matching, or from the “`~*`” symbols for case-insensitive matching. The regular expression can contain named and positional captures, and *replacement* can reference them:

```
proxy_cookie_path ~*^/user/([^/]+) /u/$1;
```

Several `proxy_cookie_path` directives can be specified on the same level:

```
proxy_cookie_path /one/ /;  
proxy_cookie_path / /two/;
```

If several directives can be applied to the cookie, the first matching directive will be chosen.

The `off` parameter cancels the effect of the `proxy_cookie_path` directives inherited from the previous configuration level.

### proxy\_force\_ranges

SYNTAX: **proxy\_force\_ranges** `on` | `off`;  
DEFAULT `off`  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.7.

Enables byte-range support for both cached and uncached responses from the proxied server regardless of the `Accept-Ranges` field in these responses.

## proxy\_headers\_hash\_bucket\_size

SYNTAX: **proxy\_headers\_hash\_bucket\_size** *size*;

DEFAULT 64

CONTEXT: http, server, location

Sets the bucket *size* for hash tables used by the [proxy\\_hide\\_header](#) and [proxy\\_set\\_header](#) directives. The details of setting up hash tables are provided in a separate [document](#).

## proxy\_headers\_hash\_max\_size

SYNTAX: **proxy\_headers\_hash\_max\_size** *size*;

DEFAULT 512

CONTEXT: http, server, location

Sets the maximum *size* of hash tables used by the [proxy\\_hide\\_header](#) and [proxy\\_set\\_header](#) directives. The details of setting up hash tables are provided in a separate [document](#).

## proxy\_hide\_header

SYNTAX: **proxy\_hide\_header** *field*;

DEFAULT —

CONTEXT: http, server, location

By default, nginx does not pass the header fields Date, Server, X-Pad, and X-Accel-... from the response of a proxied server to a client. The `proxy_hide_header` directive sets additional fields that will not be passed. If, on the contrary, the passing of fields needs to be permitted, the [proxy\\_pass\\_header](#) directive can be used.

## proxy\_http\_version

SYNTAX: **proxy\_http\_version** 1.0 | 1.1;

DEFAULT 1.0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.4.

Sets the HTTP protocol version for proxying. By default, version 1.0 is used. Version 1.1 is recommended for use with [keepalive](#) connections and [NTLM authentication](#).

## proxy\_ignore\_client\_abort

SYNTAX: **proxy\_ignore\_client\_abort** on | off;

DEFAULT off

CONTEXT: http, server, location

Determines whether the connection with a proxied server should be closed when a client closes the connection without waiting for a response.

## proxy\_ignore\_headers

SYNTAX: **proxy\_ignore\_headers** *field* ...;

DEFAULT —

CONTEXT: http, server, location

Disables processing of certain response header fields from the proxied server. The following fields can be ignored: X-Accel-Redirect, X-Accel-Expires, X-Accel-Limit-Rate (1.1.6), X-Accel-Buffering (1.1.6), X-Accel-Charset (1.1.6), Expires, Cache-Control, Set-Cookie (0.8.44), and Vary (1.7.7).

If not disabled, processing of these header fields has the following effect:

- X-Accel-Expires, Expires, Cache-Control, Set-Cookie, and Vary set the parameters of response [caching](#);
- X-Accel-Redirect performs an [internal redirect](#) to the specified URI;
- X-Accel-Limit-Rate sets the [rate limit](#) for transmission of a response to a client;
- X-Accel-Buffering enables or disables [buffering](#) of a response;
- X-Accel-Charset sets the desired [charset](#) of a response.

## proxy\_intercept\_errors

SYNTAX: **proxy\_intercept\_errors** on | off;

DEFAULT off

CONTEXT: http, server, location

Determines whether proxied responses with codes greater than or equal to 300 should be passed to a client or be intercepted and redirected to nginx for processing with the [error\\_page](#) directive.

## proxy\_limit\_rate

SYNTAX: **proxy\_limit\_rate** *rate*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.7.

Limits the speed of reading the response from the proxied server. The *rate* is specified in bytes per second. The zero value disables rate limiting. The limit is set per a request, and so if nginx simultaneously opens two connections to the proxied server, the overall rate will be twice as much as the specified limit. The limitation works only if [buffering](#) of responses from the proxied server is enabled.

## proxy\_max\_temp\_file\_size

SYNTAX: **proxy\_max\_temp\_file\_size** *size*;

DEFAULT 1024m

CONTEXT: http, server, location

When [buffering](#) of responses from the proxied server is enabled, and the whole response does not fit into the buffers set by the [proxy\\_buffer\\_size](#) and [proxy\\_buffers](#) directives, a part of the response can be saved to a temporary file. This directive sets the maximum *size* of the temporary file. The size of data written to the temporary file at a time is set by the [proxy\\_temp\\_file\\_write\\_size](#) directive.

The zero value disables buffering of responses to temporary files.

This restriction does not apply to responses that will be [cached](#) or [stored](#) on disk.

## proxy\_method

SYNTAX: **proxy\_method** *method*;

DEFAULT —

CONTEXT: http, server, location

Specifies the HTTP *method* to use in requests forwarded to the proxied server instead of the method from the client request. Parameter value can contain variables (1.11.6).

## proxy\_next\_upstream

SYNTAX: **proxy\_next\_upstream** error | timeout | invalid\_header |  
http\_500 | http\_502 | http\_503 | http\_504 | http\_403 |  
http\_404 | http\_429 | non\_idempotent | off ...;

DEFAULT error timeout

CONTEXT: http, server, location

Specifies in which cases a request should be passed to the next server:

error

an error occurred while establishing a connection with the server, passing a request to it, or reading the response header;

timeout

a timeout has occurred while establishing a connection with the server, passing a request to it, or reading the response header;

invalid\_header

a server returned an empty or invalid response;

http\_500

a server returned a response with the code 500;

http\_502

a server returned a response with the code 502;

`http_503`  
 a server returned a response with the code 503;  
`http_504`  
 a server returned a response with the code 504;  
`http_403`  
 a server returned a response with the code 403;  
`http_404`  
 a server returned a response with the code 404;  
`http_429`  
 a server returned a response with the code 429 (1.11.13);  
`non_idempotent`  
 normally, requests with a [non-idempotent](#) method (POST, LOCK, PATCH) are not passed to the next server if a request has been sent to an upstream server (1.9.13); enabling this option explicitly allows retrying such requests;  
`off`  
 disables passing a request to the next server.

One should bear in mind that passing a request to the next server is only possible if nothing has been sent to a client yet. That is, if an error or timeout occurs in the middle of the transferring of a response, fixing this is impossible.

The directive also defines what is considered an [unsuccessful attempt](#) of communication with a server. The cases of `error`, `timeout` and `invalid_header` are always considered unsuccessful attempts, even if they are not specified in the directive. The cases of `http_500`, `http_502`, `http_503`, `http_504`, and `http_429` are considered unsuccessful attempts only if they are specified in the directive. The cases of `http_403` and `http_404` are never considered unsuccessful attempts.

Passing a request to the next server can be limited by [the number of tries](#) and by [time](#).

### **proxy\_next\_upstream\_timeout**

SYNTAX: **proxy\_next\_upstream\_timeout** *time*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.5.

Limits the time during which a request can be passed to the [next server](#). The 0 value turns off this limitation.

### **proxy\_next\_upstream\_tries**

SYNTAX: **proxy\_next\_upstream\_tries** *number*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.5.



Limits the number of possible tries for passing a request to the [next server](#). The 0 value turns off this limitation.

### proxy\_no\_cache

SYNTAX: **proxy\_no\_cache** *string* ...;

DEFAULT —

CONTEXT: http, server, location

Defines conditions under which the response will not be saved to a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be saved:

```
proxy_no_cache $cookie_nocache $arg_nocache$arg_comment;  
proxy_no_cache $http_pragma $http_authorization;
```

Can be used along with the [proxy\\_cache\\_bypass](#) directive.

### proxy\_pass

SYNTAX: **proxy\_pass** *URL*;

DEFAULT —

CONTEXT: location, if in location, limit\_except

Sets the protocol and address of a proxied server and an optional URI to which a location should be mapped. As a protocol, “http” or “https” can be specified. The address can be specified as a domain name or IP address, and an optional port:

```
proxy_pass http://localhost:8000/uri/;
```

or as a UNIX-domain socket path specified after the word “unix” and enclosed in colons:

```
proxy_pass http://unix:/tmp/backend.socket:/uri/;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a [server group](#).

Parameter value can contain variables. In this case, if an address is specified as a domain name, the name is searched among the described server groups, and, if not found, is determined using a [resolver](#).

A request URI is passed to the server as follows:

- If the `proxy_pass` directive is specified with a URI, then when a request is passed to the server, the part of a [normalized](#) request URI matching the location is replaced by a URI specified in the directive:

```
location /name/ {  
    proxy_pass http://127.0.0.1/remote/;  
}
```

- If `proxy_pass` is specified without a URI, the request URI is passed to the server in the same form as sent by a client when the original request is processed, or the full normalized request URI is passed when processing the changed URI:

```
location /some/path/ {
    proxy_pass http://127.0.0.1;
}
```

Before version 1.1.12, if `proxy_pass` is specified without a URI, the original request URI might be passed instead of the changed URI in some cases.

In some cases, the part of a request URI to be replaced cannot be determined:

- When location is specified using a regular expression, and also inside named locations.

In these cases, `proxy_pass` should be specified without a URI.

- When the URI is changed inside a proxied location using the [rewrite](#) directive, and this same configuration will be used to process a request (break):

```
location /name/ {
    rewrite /name/([^\/]*) /users?name=$1 break;
    proxy_pass http://127.0.0.1;
}
```

In this case, the URI specified in the directive is ignored and the full changed request URI is passed to the server.

- When variables are used in `proxy_pass`:

```
location /name/ {
    proxy_pass http://127.0.0.1$request_uri;
}
```

In this case, if URI is specified in the directive, it is passed to the server as is, replacing the original request URI.

[WebSocket](#) proxying requires special configuration and is supported since version 1.3.13.

### `proxy_pass_header`

SYNTAX: **`proxy_pass_header`** *field*;

DEFAULT —

CONTEXT: http, server, location

Permits passing [otherwise disabled](#) header fields from a proxied server to a client.

## proxy\_pass\_request\_body

SYNTAX: **proxy\_pass\_request\_body** on | off;

DEFAULT on

CONTEXT: http, server, location

Indicates whether the original request body is passed to the proxied server.

```
location /x-accel-redirect-here/ {
    proxy_method GET;
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";

    proxy_pass ...
}
```

See also the [proxy\\_set\\_header](#) and [proxy\\_pass\\_request\\_headers](#) directives.

## proxy\_pass\_request\_headers

SYNTAX: **proxy\_pass\_request\_headers** on | off;

DEFAULT on

CONTEXT: http, server, location

Indicates whether the header fields of the original request are passed to the proxied server.

```
location /x-accel-redirect-here/ {
    proxy_method GET;
    proxy_pass_request_headers off;
    proxy_pass_request_body off;

    proxy_pass ...
}
```

See also the [proxy\\_set\\_header](#) and [proxy\\_pass\\_request\\_body](#) directives.

## proxy\_read\_timeout

SYNTAX: **proxy\_read\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for reading a response from the proxied server. The timeout is set only between two successive read operations, not for the transmission of the whole response. If the proxied server does not transmit anything within this time, the connection is closed.

## proxy\_redirect

SYNTAX: **proxy\_redirect** default;

SYNTAX: **proxy\_redirect** off;

SYNTAX: **proxy\_redirect** *redirect replacement*;

DEFAULT default

CONTEXT: http, server, location

Sets the text that should be changed in the `Location` and `Refresh` header fields of a proxied server response. Suppose a proxied server returned the header field “`Location: http://localhost:8000/two/some/uri/`”. The directive

```
proxy_redirect http://localhost:8000/two/ http://frontend/one/;
```

will rewrite this string to “`Location: http://frontend/one/some/uri/`”.

A server name may be omitted in the *replacement* string:

```
proxy_redirect http://localhost:8000/two/ /;
```

then the primary server’s name and port, if different from 80, will be inserted.

The default replacement specified by the default parameter uses the parameters of the `location` and `proxy_pass` directives. Hence, the two configurations below are equivalent:

```
location /one/ {
    proxy_pass      http://upstream:port/two/;
    proxy_redirect  default;
```

```
location /one/ {
    proxy_pass      http://upstream:port/two/;
    proxy_redirect  http://upstream:port/two/ /one/;
```

The default parameter is not permitted if `proxy_pass` is specified using variables.

A *replacement* string can contain variables:

```
proxy_redirect http://localhost:8000/ http://$host:$server_port/;
```

A *redirect* can also contain (1.1.11) variables:

```
proxy_redirect http://$proxy_host:8000/ /;
```

The directive can be specified (1.1.11) using regular expressions. In this case, *redirect* should either start with the “`~`” symbol for a case-sensitive matching, or with the “`~*`” symbols for case-insensitive matching. The regular expression can contain named and positional captures, and *replacement* can reference them:

```
proxy_redirect ~^(http://[^\:]+\):\d+(/.+)$ $1$2;
proxy_redirect ~*/user/([^\:]+)/(.+)$      http://$1.example.com/$2;
```

Several `proxy_redirect` directives can be specified on the same level:

```
proxy_redirect default;
proxy_redirect http://localhost:8000/ /;
proxy_redirect http://www.example.com/ /;
```

If several directives can be applied to the header fields of a proxied server response, the first matching directive will be chosen.

The `off` parameter cancels the effect of the `proxy_redirect` directives inherited from the previous configuration level.

Using this directive, it is also possible to add host names to relative redirects issued by a proxied server:

```
proxy_redirect / /;
```

### **proxy\_request\_buffering**

SYNTAX: **proxy\_request\_buffering** on | off;

DEFAULT on

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Enables or disables buffering of a client request body.

When buffering is enabled, the entire request body is [read](#) from the client before sending the request to a proxied server.

When buffering is disabled, the request body is sent to the proxied server immediately as it is received. In this case, the request cannot be passed to the [next server](#) if nginx already started sending the request body.

When HTTP/1.1 chunked transfer encoding is used to send the original request body, the request body will be buffered regardless of the directive value unless HTTP/1.1 is [enabled](#) for proxying.

### **proxy\_send\_lowat**

SYNTAX: **proxy\_send\_lowat** *size*;

DEFAULT 0

CONTEXT: http, server, location

If the directive is set to a non-zero value, nginx will try to minimize the number of send operations on outgoing connections to a proxied server by using either `NOTE_LOWAT` flag of the [kqueue](#) method, or the `SO_SNDLOWAT` socket option, with the specified *size*.

This directive is ignored on Linux, Solaris, and Windows.

### **proxy\_send\_timeout**

SYNTAX: **proxy\_send\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Sets a timeout for transmitting a request to the proxied server. The timeout is set only between two successive write operations, not for the transmission of the whole request. If the proxied server does not receive anything within this time, the connection is closed.

## proxy\_set\_body

SYNTAX: **proxy\_set\_body** *value*;

DEFAULT —

CONTEXT: http, server, location

Allows redefining the request body passed to the proxied server. The *value* can contain text, variables, and their combination.

## proxy\_set\_header

SYNTAX: **proxy\_set\_header** *field value*;

DEFAULT Host \$proxy\_host

DEFAULT Connection close

CONTEXT: http, server, location

Allows redefining or appending fields to the request header [passed](#) to the proxied server. The *value* can contain text, variables, and their combinations. These directives are inherited from the previous configuration level if and only if there are no `proxy_set_header` directives defined on the current level. By default, only two fields are redefined:

```
proxy_set_header Host      $proxy_host;
proxy_set_header Connection close;
```

If caching is enabled, the header fields If-Modified-Since, If-Unmodified-Since, If-None-Match, If-Match, Range, and If-Range from the original request are not passed to the proxied server.

An unchanged Host request header field can be passed like this:

```
proxy_set_header Host      $http_host;
```

However, if this field is not present in a client request header then nothing will be passed. In such a case it is better to use the *\$host* variable - its value equals the server name in the Host request header field or the primary server name if this field is not present:

```
proxy_set_header Host      $host;
```

In addition, the server name can be passed together with the port of the proxied server:

```
proxy_set_header Host      $host:$proxy_port;
```

If the value of a header field is an empty string then this field will not be passed to a proxied server:

```
proxy_set_header Accept-Encoding "";
```

## proxy\_socket\_keepalive

SYNTAX: **proxy\_socket\_keepalive** on | off;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.15.6.

Configures the “TCP keepalive” behavior for outgoing connections to a proxied server. By default, the operating system’s settings are in effect for the socket. If the directive is set to the value “on”, the `SO_KEEPALIVE` socket option is turned on for the socket.

## proxy\_ssl\_certificate

SYNTAX: **proxy\_ssl\_certificate** *file*;  
DEFAULT —  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.8.

Specifies a *file* with the certificate in the PEM format used for authentication to a proxied HTTPS server.

Since version 1.21.0, variables can be used in the *file* name.

## proxy\_ssl\_certificate\_key

SYNTAX: **proxy\_ssl\_certificate\_key** *file*;  
DEFAULT —  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.8.

Specifies a *file* with the secret key in the PEM format used for authentication to a proxied HTTPS server.

The value `engine:name:id` can be specified instead of the *file* (1.7.9), which loads a secret key with a specified *id* from the OpenSSL engine *name*.

Since version 1.21.0, variables can be used in the *file* name.

## proxy\_ssl\_ciphers

SYNTAX: **proxy\_ssl\_ciphers** *ciphers*;  
DEFAULT DEFAULT  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.5.6.

Specifies the enabled ciphers for requests to a proxied HTTPS server. The ciphers are specified in the format understood by the OpenSSL library.

The full list can be viewed using the “`openssl ciphers`” command.

## proxy\_ssl\_conf\_command

SYNTAX: **proxy\_ssl\_conf\_command** *name value*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

Sets arbitrary OpenSSL configuration [commands](#) when establishing a connection with the proxied HTTPS server.

The directive is supported when using OpenSSL 1.0.2 or higher.

Several `proxy_ssl_conf_command` directives can be specified on the same level. These directives are inherited from the previous configuration level if and only if there are no `proxy_ssl_conf_command` directives defined on the current level.

Note that configuring OpenSSL directly might result in unexpected behavior.

## proxy\_ssl\_crl

SYNTAX: **proxy\_ssl\_crl** *file*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Specifies a *file* with revoked certificates (CRL) in the PEM format used to [verify](#) the certificate of the proxied HTTPS server.

## proxy\_ssl\_name

SYNTAX: **proxy\_ssl\_name** *name*;

DEFAULT `$proxy_host`

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Allows overriding the server name used to [verify](#) the certificate of the proxied HTTPS server and to be [passed through SNI](#) when establishing a connection with the proxied HTTPS server.

By default, the host part of the [proxy\\_pass](#) URL is used.

## proxy\_ssl\_password\_file

SYNTAX: **proxy\_ssl\_password\_file** *file*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.8.



Specifies a *file* with passphrases for [secret keys](#) where each passphrase is specified on a separate line. Passphrases are tried in turn when loading the key.

### proxy\_ssl\_protocols

SYNTAX: **proxy\_ssl\_protocols** [SSLv2] [SSLv3] [TLSv1] [TLSv1.1]  
[TLSv1.2] [TLSv1.3];  
DEFAULT TLSv1 TLSv1.1 TLSv1.2  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.5.6.

Enables the specified protocols for requests to a proxied HTTPS server.

### proxy\_ssl\_server\_name

SYNTAX: **proxy\_ssl\_server\_name** on | off;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Enables or disables passing of the server name through [TLS Server Name Indication extension](#) (SNI, RFC 6066) when establishing a connection with the proxied HTTPS server.

### proxy\_ssl\_session\_reuse

SYNTAX: **proxy\_ssl\_session\_reuse** on | off;  
DEFAULT on  
CONTEXT: http, server, location

Determines whether SSL sessions can be reused when working with the proxied server. If the errors “SSL3\_GET\_FINISHED:digest check failed” appear in the logs, try disabling session reuse.

### proxy\_ssl\_trusted\_certificate

SYNTAX: **proxy\_ssl\_trusted\_certificate** *file*;  
DEFAULT —  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) the certificate of the proxied HTTPS server.

### proxy\_ssl\_verify

SYNTAX: **proxy\_ssl\_verify** on | off;  
DEFAULT off  
CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Enables or disables verification of the proxied HTTPS server certificate.

### proxy\_ssl\_verify\_depth

SYNTAX: **proxy\_ssl\_verify\_depth** *number*;

DEFAULT 1

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Sets the verification depth in the proxied HTTPS server certificates chain.

### proxy\_store

SYNTAX: **proxy\_store** on | off | *string*;

DEFAULT off

CONTEXT: http, server, location

Enables saving of files to a disk. The `on` parameter saves files with paths corresponding to the directives [alias](#) or [root](#). The `off` parameter disables saving of files. In addition, the file name can be set explicitly using the *string* with variables:

```
proxy_store /data/www$original_uri;
```

The modification time of files is set according to the received Last-Modified response header field. The response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the persistent store can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both saved files and a directory holding temporary files, set by the [proxy\\_temp\\_path](#) directive, are put on the same file system.

This directive can be used to create local copies of static unchangeable files, e.g.:

```
location /images/ {
    root                /data/www;
    error_page          404 = /fetch$uri;
}

location /fetch/ {
    internal;

    proxy_pass          http://backend/;
    proxy_store          on;
    proxy_store_access  user:rw group:rw all:r;
    proxy_temp_path     /data/temp;

    alias               /data/www/;
}
```

or like this:

```
location /images/ {
    root          /data/www;
    error_page    404 = @fetch;
}

location @fetch {
    internal;

    proxy_pass     http://backend;
    proxy_store    on;
    proxy_store_access user:rw group:rw all:r;
    proxy_temp_path /data/temp;

    root          /data/www;
}
```

### proxy\_store\_access

SYNTAX: **proxy\_store\_access** *users:permissions ...*;

DEFAULT `user:rw`

CONTEXT: http, server, location

Sets access permissions for newly created files and directories, e.g.:

```
proxy_store_access user:rw group:rw all:r;
```

If any group or all access permissions are specified then user permissions may be omitted:

```
proxy_store_access group:rw all:r;
```

### proxy\_temp\_file\_write\_size

SYNTAX: **proxy\_temp\_file\_write\_size** *size*;

DEFAULT `8k|16k`

CONTEXT: http, server, location

Limits the *size* of data written to a temporary file at a time, when buffering of responses from the proxied server to temporary files is enabled. By default, *size* is limited by two buffers set by the [proxy\\_buffer\\_size](#) and [proxy\\_buffers](#) directives. The maximum size of a temporary file is set by the [proxy\\_max\\_temp\\_file\\_size](#) directive.

### proxy\_temp\_path

SYNTAX: **proxy\_temp\_path** *path* [*level1* [*level2* [*level3*]]];

DEFAULT `proxy_temp`

CONTEXT: http, server, location

Defines a directory for storing temporary files with data received from proxied servers. Up to three-level subdirectory hierarchy can be used underneath the specified directory. For example, in the following configuration

```
proxy_temp_path /spool/nginx/proxy_temp 1 2;
```

a temporary file might look like this:

```
/spool/nginx/proxy_temp/7/45/00000123457
```

See also the `use_temp_path` parameter of the [proxy\\_cache\\_path](#) directive.

### 2.36.4 Embedded Variables

The `ngx_http_proxy_module` module supports embedded variables that can be used to compose headers using the [proxy\\_set\\_header](#) directive:

*\$proxy\_host*

name and port of a proxied server as specified in the [proxy\\_pass](#) directive;

*\$proxy\_port*

port of a proxied server as specified in the [proxy\\_pass](#) directive, or the protocol's default port;

*\$proxy\_add\_x\_forwarded\_for*

the X-Forwarded-For client request header field with the *\$remote\_addr* variable appended to it, separated by a comma. If the X-Forwarded-For field is not present in the client request header, the *\$proxy\_add\_x\_forwarded\_for* variable is equal to the *\$remote\_addr* variable.

## 2.37 Module ngx\_http\_proxy\_protocol\_vendor\_module

2.37.1 Summary	260
2.37.2 Example Configuration	260
2.37.3 Embedded Variables	260

### 2.37.1 Summary

The `ngx_http_proxy_protocol_vendor_module` module (1.23.3) allows obtaining additional information about a connection in cloud platforms from application-specific TLVs of the [PROXY protocol](#) header.

Supported cloud platforms:

- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the [listen](#) directive.

This module is available as part of our [commercial subscription](#).

### 2.37.2 Example Configuration

```
proxy_set_header X-Conn-ID $proxy_protocol_tlv_gcp_conn_id;

server {
    listen 80    proxy_protocol;
    listen 443  ssl proxy_protocol;
    ...
}
```

### 2.37.3 Embedded Variables

*\$proxy\_protocol\_tlv\_aws\_vpce\_id*

TLV value from the PROXY Protocol header representing the [ID of AWS VPC endpoint](#)

*\$proxy\_protocol\_tlv\_azure\_pe\_id*

TLV value from the PROXY Protocol header representing the [LinkID of Azure private endpoint](#)

*\$proxy\_protocol\_tlv\_gcp\_conn\_id*

TLV value from the PROXY Protocol header representing [Google Cloud PSC connection ID](#)

## 2.38 Module ngx\_http\_random\_index\_module

2.38.1 Summary	261
2.38.2 Example Configuration	261
2.38.3 Directives	261
random_index	261

### 2.38.1 Summary

The `ngx_http_random_index_module` module processes requests ending with the slash character (`/`) and picks a random file in a directory to serve as an index file. The module is processed before the [ngx\\_http\\_index\\_module](#) module.

This module is not built by default, it should be enabled with the `--with-http_random_index_module` configuration parameter.

### 2.38.2 Example Configuration

```
location / {  
    random_index on;  
}
```

### 2.38.3 Directives

#### random\_index

SYNTAX: **random\_index** on | off;

DEFAULT off

CONTEXT: location

Enables or disables module processing in a surrounding location.

## 2.39 Module ngx\_http\_realip\_module

2.39.1 Summary	262
2.39.2 Example Configuration	262
2.39.3 Directives	262
set_real_ip_from	262
real_ip_header	262
real_ip_recursive	263
2.39.4 Embedded Variables	263

### 2.39.1 Summary

The `ngx_http_realip_module` module is used to change the client address and optional port to those sent in the specified header field.

This module is not built by default, it should be enabled with the `--with-http_realip_module` configuration parameter.

### 2.39.2 Example Configuration

```
set_real_ip_from 192.168.1.0/24;
set_real_ip_from 192.168.2.1;
set_real_ip_from 2001:0db8::/32;
real_ip_header X-Forwarded-For;
real_ip_recursive on;
```

### 2.39.3 Directives

#### set\_real\_ip\_from

SYNTAX: **set\_real\_ip\_from** *address* | *CIDR* | `unix::`;

DEFAULT —

CONTEXT: http, server, location

Defines trusted addresses that are known to send correct replacement addresses. If the special value `unix:` is specified, all UNIX-domain sockets will be trusted. Trusted addresses may also be specified using a hostname (1.13.1).

IPv6 addresses are supported starting from versions 1.3.0 and 1.2.1.

#### real\_ip\_header

SYNTAX: **real\_ip\_header** *field* | `X-Real-IP` | `X-Forwarded-For` | `proxy_protocol`;

DEFAULT `X-Real-IP`

CONTEXT: http, server, location

Defines the request header field whose value will be used to replace the client address.

The request header field value that contains an optional port is also used to replace the client port (1.11.0). The address and port should be specified according to [RFC 3986](#).

The `proxy_protocol` parameter (1.5.12) changes the client address to the one from the PROXY protocol header. The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the [listen](#) directive.

### **realip\_recursive**

SYNTAX: **realip\_recursive** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSIONS 1.3.0 AND 1.2.1.

If recursive search is disabled, the original client address that matches one of the trusted addresses is replaced by the last address sent in the request header field defined by the [realip\\_header](#) directive. If recursive search is enabled, the original client address that matches one of the trusted addresses is replaced by the last non-trusted address sent in the request header field.

## **2.39.4 Embedded Variables**

*\$realip\_remote\_addr*

keeps the original client address (1.9.7)

*\$realip\_remote\_port*

keeps the original client port (1.11.0)



## 2.40 Module ngx\_http\_referer\_module

2.40.1 Summary	264
2.40.2 Example Configuration	264
2.40.3 Directives	264
<a href="#">referer_hash_bucket_size</a>	264
<a href="#">referer_hash_max_size</a>	264
<a href="#">valid_referers</a>	265
2.40.4 Embedded Variables	265

### 2.40.1 Summary

The `ngx_http_referer_module` module is used to block access to a site for requests with invalid values in the `Referer` header field. It should be kept in mind that fabricating a request with an appropriate `Referer` field value is quite easy, and so the intended purpose of this module is not to block such requests thoroughly but to block the mass flow of requests sent by regular browsers. It should also be taken into consideration that regular browsers may not send the `Referer` field even for valid requests.

### 2.40.2 Example Configuration

```
valid_referers none blocked server_names
               *.example.com example.* www.example.org/galleries/
               ~\.google\.;

if ($invalid_referer) {
    return 403;
}
```

### 2.40.3 Directives

#### `referer_hash_bucket_size`

SYNTAX: **referer\_hash\_bucket\_size** *size*;

DEFAULT 64

CONTEXT: server, location

THIS DIRECTIVE APPEARED IN VERSION 1.0.5.

Sets the bucket size for the valid referers hash tables. The details of setting up hash tables are provided in a separate [document](#).

#### `referer_hash_max_size`

SYNTAX: **referer\_hash\_max\_size** *size*;

DEFAULT 2048

CONTEXT: server, location

THIS DIRECTIVE APPEARED IN VERSION 1.0.5.

Sets the maximum *size* of the valid referers hash tables. The details of setting up hash tables are provided in a separate [document](#).

### valid\_referers

SYNTAX: **valid\_referers** none | blocked | server\_names | *string* ...;

DEFAULT —

CONTEXT: server, location

Specifies the `Referer` request header field values that will cause the embedded `$invalid_referer` variable to be set to an empty string. Otherwise, the variable will be set to “1”. Search for a match is case-insensitive.

Parameters can be as follows:

none

the `Referer` field is missing in the request header;

blocked

the `Referer` field is present in the request header, but its value has been deleted by a firewall or proxy server; such values are strings that do not start with “`http://`” or “`https://`”;

server\_names

the `Referer` request header field contains one of the server names;

arbitrary string

defines a server name and an optional URI prefix. A server name can have an “`*`” at the beginning or end. During the checking, the server’s port in the `Referer` field is ignored;

regular expression

the first symbol should be a “`~`”. It should be noted that an expression will be matched against the text starting after the “`http://`” or “`https://`”.

Example:

```
valid_referers none blocked server_names
                *.example.com example.* www.example.org/galleries/
                ~\.google\.;
```

## 2.40.4 Embedded Variables

`$invalid_referer`

Empty string, if the `Referer` request header field value is considered [valid](#), otherwise “1”.

## 2.41 Module ngx\_http\_rewrite\_module

2.41.1 Summary	266
2.41.2 Directives	266
break	266
if	267
return	268
rewrite	268
rewrite_log	270
set	270
uninitialized_variable_warn	270
2.41.3 Internal Implementation	270

### 2.41.1 Summary

The `ngx_http_rewrite_module` module is used to change request URI using PCRE regular expressions, return redirects, and conditionally select configurations.

The `break`, `if`, `return`, `rewrite`, and `set` directives are processed in the following order:

- the directives of this module specified on the `server` level are executed sequentially;
- repeatedly:
  - a `location` is searched based on a request URI;
  - the directives of this module specified inside the found location are executed sequentially;
  - the loop is repeated if a request URI was `rewritten`, but not more than 10 times.

### 2.41.2 Directives

#### break

SYNTAX: **break**;

DEFAULT —

CONTEXT: server, location, if

Stops processing the current set of `ngx_http_rewrite_module` directives.

If a directive is specified inside the `location`, further processing of the request continues in this location.

Example:

```
if ($slow) {  
    limit_rate 10k;  
    break;  
}
```

## if

SYNTAX: **if** (*condition*) { ... }

DEFAULT —

CONTEXT: server, location

The specified *condition* is evaluated. If true, this module directives specified inside the braces are executed, and the request is assigned the configuration inside the `if` directive. Configurations inside the `if` directives are inherited from the previous configuration level.

A condition may be any of the following:

- a variable name; false if the value of a variable is an empty string or “0”;

Before version 1.0.1, any string starting with “0” was considered a false value.

- comparison of a variable with a string using the “=” and “!=” operators;
- matching of a variable against a regular expression using the “~” (for case-sensitive matching) and “~\*” (for case-insensitive matching) operators. Regular expressions can contain captures that are made available for later reuse in the *\$1..\$9* variables. Negative operators “!~” and “!~\*” are also available. If a regular expression includes the “}” or “;” characters, the whole expressions should be enclosed in single or double quotes.
- checking of a file existence with the “-f” and “!-f” operators;
- checking of a directory existence with the “-d” and “!-d” operators;
- checking of a file, directory, or symbolic link existence with the “-e” and “!-e” operators;
- checking for an executable file with the “-x” and “!-x” operators.

Examples:

```
if ($http_user_agent ~ MSIE) {  
    rewrite ^(.*)$ /msie/$1 break;  
}  
  
if ($http_cookie ~* "id=([^;]+)(?:;|$)") {  
    set $id $1;  
}  
  
if ($request_method = POST) {  
    return 405;  
}
```

```

}

if ($slow) {
    limit_rate 10k;
}

if ($invalid_referer) {
    return 403;
}

```

A value of the *\$invalid\_referer* embedded variable is set by the [valid\\_referers](#) directive.

## return

SYNTAX: **return** *code* [*text*];

SYNTAX: **return** *code* *URL*;

SYNTAX: **return** *URL*;

DEFAULT —

CONTEXT: server, location, if

Stops processing and returns the specified *code* to a client. The non-standard code 444 closes a connection without sending a response header.

Starting from version 0.8.42, it is possible to specify either a redirect URL (for codes 301, 302, 303, 307, and 308) or the response body *text* (for other codes). A response body text and redirect URL can contain variables. As a special case, a redirect URL can be specified as a URI local to this server, in which case the full redirect URL is formed according to the request scheme (*\$scheme*) and the [server\\_name\\_in\\_redirect](#) and [port\\_in\\_redirect](#) directives.

In addition, a *URL* for temporary redirect with the code 302 can be specified as the sole parameter. Such a parameter should start with the “http://”, “https://”, or “\$scheme” string. A *URL* can contain variables.

Only the following codes could be returned before version 0.7.51: 204, 400, 402 — 406, 408, 410, 411, 413, 416, and 500 — 504.

The code 307 was not treated as a redirect until versions 1.1.16 and 1.0.13.

The code 308 was not treated as a redirect until version 1.13.0.

See also the [error\\_page](#) directive.

## rewrite

SYNTAX: **rewrite** *regex replacement* [*flag*];

DEFAULT —

CONTEXT: server, location, if

If the specified regular expression matches a request URI, URI is changed as specified in the *replacement* string. The *rewrite* directives are executed

sequentially in order of their appearance in the configuration file. It is possible to terminate further processing of the directives using flags. If a replacement string starts with “http://”, “https://”, or “\$scheme”, the processing stops and the redirect is returned to a client.

An optional *flag* parameter can be one of:

**last**

stops processing the current set of `ngx_http_rewrite_module` directives and starts a search for a new location matching the changed URI;

**break**

stops processing the current set of `ngx_http_rewrite_module` directives as with the `break` directive;

**redirect**

returns a temporary redirect with the 302 code; used if a replacement string does not start with “http://”, “https://”, or “\$scheme”;

**permanent**

returns a permanent redirect with the 301 code.

The full redirect URL is formed according to the request scheme (*\$scheme*) and the `server_name_in_redirect` and `port_in_redirect` directives.

Example:

```
server {
    ...
    rewrite ^(/download/.*)/media/(.*)\..*$ $1/mp3/$2.mp3 last;
    rewrite ^(/download/.*)/audio/(.*)\..*$ $1/mp3/$2.ra last;
    return 403;
    ...
}
```

But if these directives are put inside the “/download/” location, the `last` flag should be replaced by `break`, or otherwise nginx will make 10 cycles and return the 500 error:

```
location /download/ {
    rewrite ^(/download/.*)/media/(.*)\..*$ $1/mp3/$2.mp3 break;
    rewrite ^(/download/.*)/audio/(.*)\..*$ $1/mp3/$2.ra break;
    return 403;
}
```

If a *replacement* string includes the new request arguments, the previous request arguments are appended after them. If this is undesired, putting a question mark at the end of a replacement string avoids having them appended, for example:

```
rewrite ^/users/(.*)$ /show?user=$1? last;
```

If a regular expression includes the “}” or “;” characters, the whole expressions should be enclosed in single or double quotes.

## rewrite\_log

SYNTAX: **rewrite\_log** on | off;

DEFAULT off

CONTEXT: http, server, location, if

Enables or disables logging of `ngx_http_rewrite_module` module directives processing results into the [error\\_log](#) at the notice level.

## set

SYNTAX: **set** *\$variable value*;

DEFAULT —

CONTEXT: server, location, if

Sets a *value* for the specified *variable*. The *value* can contain text, variables, and their combination.

## uninitialized\_variable\_warn

SYNTAX: **uninitialized\_variable\_warn** on | off;

DEFAULT on

CONTEXT: http, server, location, if

Controls whether warnings about uninitialized variables are logged.

### 2.41.3 Internal Implementation

The `ngx_http_rewrite_module` module directives are compiled at the configuration stage into internal instructions that are interpreted during request processing. An interpreter is a simple virtual stack machine.

For example, the directives

```
location /download/ {
    if ($forbidden) {
        return 403;
    }

    if ($slow) {
        limit_rate 10k;
    }

    rewrite ^/(download/.*)/media/(.*)\..*$ /$1/mp3/$2.mp3 break;
}
```

will be translated into these instructions:

```
variable $forbidden
check against zero
    return 403
end of code
variable $slow
check against zero
match of regular expression
copy "/"
copy $1
```

```
copy "/mp3/"
copy $2
copy ".mp3"
end of regular expression
end of code
```

Note that there are no instructions for the `limit_rate` directive above as it is unrelated to the `ngx_http_rewrite_module` module. A separate configuration is created for the `if` block. If the condition holds true, a request is assigned this configuration where `limit_rate` equals to 10k.

The directive

```
rewrite ^(download/.*)/media/(.*)\.*$ /$1/mp3/$2.mp3 break;
```

can be made smaller by one instruction if the first slash in the regular expression is put inside the parentheses:

```
rewrite ^(/download/.*)/media/(.*)\.*$ $1/mp3/$2.mp3 break;
```

The corresponding instructions will then look like this:

```
match of regular expression
copy $1
copy "/mp3/"
copy $2
copy ".mp3"
end of regular expression
end of code
```



## 2.42 Module ngx\_http\_scgi\_module

2.42.1	Summary	273
2.42.2	Example Configuration	273
2.42.3	Directives	273
	scgi_bind	273
	scgi_buffer_size	273
	scgi_buffering	274
	scgi_buffers	274
	scgi_busy_buffers_size	274
	scgi_cache	275
	scgi_cache_background_update	275
	scgi_cache_bypass	275
	scgi_cache_key	275
	scgi_cache_lock	275
	scgi_cache_lock_age	276
	scgi_cache_lock_timeout	276
	scgi_cache_max_range_offset	276
	scgi_cache_methods	276
	scgi_cache_min_uses	277
	scgi_cache_path	277
	scgi_cache_purge	279
	scgi_cache_revalidate	279
	scgi_cache_use_stale	279
	scgi_cache_valid	280
	scgi_connect_timeout	281
	scgi_force_ranges	281
	scgi_hide_header	281
	scgi_ignore_client_abort	282
	scgi_ignore_headers	282
	scgi_intercept_errors	282
	scgi_limit_rate	282
	scgi_max_temp_file_size	283
	scgi_next_upstream	283
	scgi_next_upstream_timeout	284
	scgi_next_upstream_tries	284
	scgi_no_cache	285
	scgi_param	285
	scgi_pass	285
	scgi_pass_header	286
	scgi_pass_request_body	286
	scgi_pass_request_headers	286
	scgi_read_timeout	286
	scgi_request_buffering	287
	scgi_send_timeout	287
	scgi_socket_keepalive	287

<a href="#">scgi_store</a>	287
<a href="#">scgi_store_access</a>	288
<a href="#">scgi_temp_file_write_size</a>	288
<a href="#">scgi_temp_path</a>	289

### 2.42.1 Summary

The `ngx_http_scgi_module` module allows passing requests to an SCGI server.

### 2.42.2 Example Configuration

```
location / {
    include    scgi_params;
    scgi_pass  localhost:9000;
}
```

### 2.42.3 Directives

#### `scgi_bind`

SYNTAX: **scgi\_bind** *address* [transparent] | off;

DEFAULT —

CONTEXT: http, server, location

Makes outgoing connections to an SCGI server originate from the specified local IP address with an optional port (1.11.2). Parameter value can contain variables (1.3.12). The special value `off` (1.3.12) cancels the effect of the `scgi_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address and port.

The `transparent` parameter (1.11.0) allows outgoing connections to an SCGI server originate from a non-local IP address, for example, from a real IP address of a client:

```
scgi_bind $remote_addr transparent;
```

In order for this parameter to work, it is usually necessary to run `nginx` worker processes with the [superuser](#) privileges. On Linux it is not required (1.13.8) as if the `transparent` parameter is specified, worker processes inherit the `CAP_NET_RAW` capability from the master process. It is also necessary to configure kernel routing table to intercept network traffic from the SCGI server.

#### `scgi_buffer_size`

SYNTAX: **scgi\_buffer\_size** *size*;

DEFAULT 4k | 8k

CONTEXT: http, server, location

Sets the *size* of the buffer used for reading the first part of the response received from the SCGI server. This part usually contains a small response header. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform. It can be made smaller, however.

### scgi\_buffering

SYNTAX: **scgi\_buffering** on | off;

DEFAULT on

CONTEXT: http, server, location

Enables or disables buffering of responses from the SCGI server.

When buffering is enabled, nginx receives a response from the SCGI server as soon as possible, saving it into the buffers set by the [scgi\\_buffer\\_size](#) and [scgi\\_buffers](#) directives. If the whole response does not fit into memory, a part of it can be saved to a [temporary file](#) on the disk. Writing to temporary files is controlled by the [scgi\\_max\\_temp\\_file\\_size](#) and [scgi\\_temp\\_file\\_write\\_size](#) directives.

When buffering is disabled, the response is passed to a client synchronously, immediately as it is received. nginx will not try to read the whole response from the SCGI server. The maximum size of the data that nginx can receive from the server at a time is set by the [scgi\\_buffer\\_size](#) directive.

Buffering can also be enabled or disabled by passing “yes” or “no” in the X-Accel-Buffering response header field. This capability can be disabled using the [scgi\\_ignore\\_headers](#) directive.

### scgi\_buffers

SYNTAX: **scgi\_buffers** *number size*;

DEFAULT 8 4k | 8k

CONTEXT: http, server, location

Sets the *number* and *size* of the buffers used for reading a response from the SCGI server, for a single connection. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

### scgi\_busy\_buffers\_size

SYNTAX: **scgi\_busy\_buffers\_size** *size*;

DEFAULT 8k | 16k

CONTEXT: http, server, location

When [buffering](#) of responses from the SCGI server is enabled, limits the total *size* of buffers that can be busy sending a response to the client while the response is not yet fully read. In the meantime, the rest of the buffers can be used for reading the response and, if needed, buffering part of the response to a temporary file. By default, *size* is limited by the size of two buffers set by the [scgi\\_buffer\\_size](#) and [scgi\\_buffers](#) directives.

## scgi\_cache

SYNTAX: **scgi\_cache** *zone* | *off*;

DEFAULT *off*

CONTEXT: http, server, location

Defines a shared memory zone used for caching. The same zone can be used in several places. Parameter value can contain variables (1.7.9). The *off* parameter disables caching inherited from the previous configuration level.

## scgi\_cache\_background\_update

SYNTAX: **scgi\_cache\_background\_update** *on* | *off*;

DEFAULT *off*

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.11.10.

Allows starting a background subrequest to update an expired cache item, while a stale cached response is returned to the client. Note that it is necessary to allow the usage of a stale cached response when it is being updated.

## scgi\_cache\_bypass

SYNTAX: **scgi\_cache\_bypass** *string* ...;

DEFAULT —

CONTEXT: http, server, location

Defines conditions under which the response will not be taken from a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be taken from the cache:

```
scgi_cache_bypass $cookie_nocache $arg_nocache$arg_comment;  
scgi_cache_bypass $http_pragma $http_authorization;
```

Can be used along with the [scgi\\_no\\_cache](#) directive.

## scgi\_cache\_key

SYNTAX: **scgi\_cache\_key** *string*;

DEFAULT —

CONTEXT: http, server, location

Defines a key for caching, for example

```
scgi_cache_key localhost:9000$request_uri;
```

## scgi\_cache\_lock

SYNTAX: **scgi\_cache\_lock** *on* | *off*;

DEFAULT *off*

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.12.

When enabled, only one request at a time will be allowed to populate a new cache element identified according to the [scgi\\_cache\\_key](#) directive by passing a request to an SCGI server. Other requests of the same cache element will either wait for a response to appear in the cache or the cache lock for this element to be released, up to the time set by the [scgi\\_cache\\_lock\\_timeout](#) directive.

### **scgi\_cache\_lock\_age**

SYNTAX: **scgi\_cache\_lock\_age** *time*;

DEFAULT 5s

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.8.

If the last request passed to the SCGI server for populating a new cache element has not completed for the specified *time*, one more request may be passed to the SCGI server.

### **scgi\_cache\_lock\_timeout**

SYNTAX: **scgi\_cache\_lock\_timeout** *time*;

DEFAULT 5s

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.12.

Sets a timeout for [scgi\\_cache\\_lock](#). When the *time* expires, the request will be passed to the SCGI server, however, the response will not be cached.

Before 1.7.8, the response could be cached.

### **scgi\_cache\_max\_range\_offset**

SYNTAX: **scgi\_cache\_max\_range\_offset** *number*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.11.6.

Sets an offset in bytes for byte-range requests. If the range is beyond the offset, the range request will be passed to the SCGI server and the response will not be cached.

### **scgi\_cache\_methods**

SYNTAX: **scgi\_cache\_methods** GET | HEAD | POST ...;

DEFAULT GET HEAD

CONTEXT: http, server, location

If the client request method is listed in this directive then the response will be cached. “GET” and “HEAD” methods are always added to the list, though it is recommended to specify them explicitly. See also the [scgi\\_no\\_cache](#) directive.

### scgi\_cache\_min\_uses

SYNTAX: **scgi\_cache\_min\_uses** *number*;

DEFAULT 1

CONTEXT: http, server, location

Sets the *number* of requests after which the response will be cached.

### scgi\_cache\_path

SYNTAX: **scgi\_cache\_path** *path* [*levels=levels*] [*use\_temp\_path=on|off*]  
[*keys\_zone=name:size*] [*inactive=time*] [*max\_size=size*]  
[*min\_free=size*] [*manager\_files=number*] [*manager\_sleep=time*]  
[*manager\_threshold=time*] [*loader\_files=number*]  
[*loader\_sleep=time*] [*loader\_threshold=time*]  
[*purger=on|off*] [*purger\_files=number*] [*purger\_sleep=time*]  
[*purger\_threshold=time*];

DEFAULT —

CONTEXT: http

Sets the path and other parameters of a cache. Cache data are stored in files. The file name in a cache is a result of applying the MD5 function to the [cache key](#). The *levels* parameter defines hierarchy levels of a cache: from 1 to 3, each level accepts values 1 or 2. For example, in the following configuration

```
scgi_cache_path /data/nginx/cache levels=1:2 keys_zone=one:10m;
```

file names in a cache will look like this:

```
/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

A cached response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the cache can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both cache and a directory holding temporary files are put on the same file system. A directory for temporary files is set based on the *use\_temp\_path* parameter (1.7.10). If this parameter is omitted or set to the value *on*, the directory set by the [scgi\\_temp\\_path](#) directive for the given location will be used. If the value is set to *off*, temporary files will be put directly in the cache directory.

In addition, all active keys and information about data are stored in a shared memory zone, whose *name* and *size* are configured by the *keys\_zone* parameter. One megabyte zone can store about 8 thousand keys.

As part of [commercial subscription](#), the shared memory zone also stores extended cache [information](#), thus, it is required to specify a larger zone size

for the same number of keys. For example, one megabyte zone can store about 4 thousand keys.

Cached data that are not accessed during the time specified by the `inactive` parameter get removed from the cache regardless of their freshness. By default, `inactive` is set to 10 minutes.

The special “cache manager” process monitors the maximum cache size set by the `max_size` parameter, and the minimum amount of free space set by the `min_free` (1.19.1) parameter on the file system with cache. When the size is exceeded or there is not enough free space, it removes the least recently used data. The data is removed in iterations configured by `manager_files`, `manager_threshold`, and `manager_sleep` parameters (1.11.5). During one iteration no more than `manager_files` items are deleted (by default, 100). The duration of one iteration is limited by the `manager_threshold` parameter (by default, 200 milliseconds). Between iterations, a pause configured by the `manager_sleep` parameter (by default, 50 milliseconds) is made.

A minute after the start the special “cache loader” process is activated. It loads information about previously cached data stored on file system into a cache zone. The loading is also done in iterations. During one iteration no more than `loader_files` items are loaded (by default, 100). Besides, the duration of one iteration is limited by the `loader_threshold` parameter (by default, 200 milliseconds). Between iterations, a pause configured by the `loader_sleep` parameter (by default, 50 milliseconds) is made.

Additionally, the following parameters are available as part of our [commercial subscription](#):

`purger=on|off`

Instructs whether cache entries that match a [wildcard key](#) will be removed from the disk by the cache purger (1.7.12). Setting the parameter to `on` (default is `off`) will activate the “cache purger” process that permanently iterates through all cache entries and deletes the entries that match the wildcard key.

`purger_files=number`

Sets the number of items that will be scanned during one iteration (1.7.12). By default, `purger_files` is set to 10.

`purger_threshold=number`

Sets the duration of one iteration (1.7.12). By default, `purger_threshold` is set to 50 milliseconds.

`purger_sleep=number`

Sets a pause between iterations (1.7.12). By default, `purger_sleep` is set to 50 milliseconds.

In versions 1.7.3, 1.7.7, and 1.11.10 cache header format has been changed. Previously cached responses will be considered invalid after upgrading to a newer nginx version.

## scgi\_cache\_purge

SYNTAX: **scgi\_cache\_purge**string ...;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Defines conditions under which the request will be considered a cache purge request. If at least one value of the string parameters is not empty and is not equal to “0” then the cache entry with a corresponding [cache key](#) is removed. The result of successful operation is indicated by returning the 204 No Content response.

If the [cache key](#) of a purge request ends with an asterisk (“\*”), all cache entries matching the wildcard key will be removed from the cache. However, these entries will remain on the disk until they are deleted for either [inactivity](#), or processed by the [cache purger](#) (1.7.12), or a client attempts to access them.

Example configuration:

```

scgi_cache_path /data/nginx/cache keys_zone=cache_zone:10m;

map $request_method $purge_method {
    PURGE    1;
    default 0;
}

server {
    ...
    location / {
        scgi_pass          backend;
        scgi_cache          cache_zone;
        scgi_cache_key      $uri;
        scgi_cache_purge    $purge_method;
    }
}

```

This functionality is available as part of our [commercial subscription](#).

## scgi\_cache\_revalidate

SYNTAX: **scgi\_cache\_revalidate** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Enables revalidation of expired cache items using conditional requests with the If-Modified-Since and If-None-Match header fields.

## scgi\_cache\_use\_stale

SYNTAX: **scgi\_cache\_use\_stale** error | timeout | invalid\_header |  
updating | http\_500 | http\_503 | http\_403 | http\_404 |  
http\_429 | off ...;

DEFAULT off

CONTEXT: http, server, location



Determines in which cases a stale cached response can be used when an error occurs during communication with the SCGI server. The directive's parameters match the parameters of the [scgi\\_next\\_upstream](#) directive.

The `error` parameter also permits using a stale cached response if an SCGI server to process a request cannot be selected.

Additionally, the `updating` parameter permits using a stale cached response if it is currently being updated. This allows minimizing the number of accesses to SCGI servers when updating cached data.

Using a stale cached response can also be enabled directly in the response header for a specified number of seconds after the response became stale (1.11.10). This has lower priority than using the directive parameters.

- The “[stale-while-revalidate](#)” extension of the `Cache-Control` header field permits using a stale cached response if it is currently being updated.
- The “[stale-if-error](#)” extension of the `Cache-Control` header field permits using a stale cached response in case of an error.

To minimize the number of accesses to SCGI servers when populating a new cache element, the [scgi\\_cache\\_lock](#) directive can be used.

### **scgi\_cache\_valid**

SYNTAX: **scgi\_cache\_valid** [*code ...*] *time*;

DEFAULT —

CONTEXT: http, server, location

Sets caching time for different response codes. For example, the following directives

```
scgi_cache_valid 200 302 10m;  
scgi_cache_valid 404 1m;
```

set 10 minutes of caching for responses with codes 200 and 302 and 1 minute for responses with code 404.

If only caching *time* is specified

```
scgi_cache_valid 5m;
```

then only 200, 301, and 302 responses are cached.

In addition, the `any` parameter can be specified to cache any responses:

```
scgi_cache_valid 200 302 10m;  
scgi_cache_valid 301 1h;  
scgi_cache_valid any 1m;
```

Parameters of caching can also be set directly in the response header. This has higher priority than setting of caching time using the directive.

- The `X-Accel-Expires` header field sets caching time of a response in seconds. The zero value disables caching for a response. If the value starts with the `@` prefix, it sets an absolute time in seconds since Epoch, up to which the response may be cached.
- If the header does not include the `X-Accel-Expires` field, parameters of caching may be set in the header fields `Expires` or `Cache-Control`.
- If the header includes the `Set-Cookie` field, such a response will not be cached.
- If the header includes the `Vary` field with the special value “\*”, such a response will not be cached (1.7.7). If the header includes the `Vary` field with another value, such a response will be cached taking into account the corresponding request header fields (1.7.7).

Processing of one or more of these response header fields can be disabled using the [scgi\\_ignore\\_headers](#) directive.

### **scgi\_connect\_timeout**

SYNTAX: **scgi\_connect\_timeout** *time*;  
DEFAULT 60s  
CONTEXT: http, server, location

Defines a timeout for establishing a connection with an SCGI server. It should be noted that this timeout cannot usually exceed 75 seconds.

### **scgi\_force\_ranges**

SYNTAX: **scgi\_force\_ranges** on | off;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.7.

Enables byte-range support for both cached and uncached responses from the SCGI server regardless of the `Accept-Ranges` field in these responses.

### **scgi\_hide\_header**

SYNTAX: **scgi\_hide\_header** *field*;  
DEFAULT —  
CONTEXT: http, server, location

By default, nginx does not pass the header fields `Status` and `X-Accel-...` from the response of an SCGI server to a client. The `scgi_hide_header` directive sets additional fields that will not be passed. If, on the contrary, the passing of fields needs to be permitted, the [scgi\\_pass\\_header](#) directive can be used.

### scgi\_ignore\_client\_abort

SYNTAX: **scgi\_ignore\_client\_abort** on | off;

DEFAULT off

CONTEXT: http, server, location

Determines whether the connection with an SCGI server should be closed when a client closes the connection without waiting for a response.

### scgi\_ignore\_headers

SYNTAX: **scgi\_ignore\_headers** *field* ...;

DEFAULT —

CONTEXT: http, server, location

Disables processing of certain response header fields from the SCGI server. The following fields can be ignored: X-Accel-Redirect, X-Accel-Expires, X-Accel-Limit-Rate (1.1.6), X-Accel-Buffering (1.1.6), X-Accel-Charset (1.1.6), Expires, Cache-Control, Set-Cookie (0.8.44), and Vary (1.7.7).

If not disabled, processing of these header fields has the following effect:

- X-Accel-Expires, Expires, Cache-Control, Set-Cookie, and Vary set the parameters of response [caching](#);
- X-Accel-Redirect performs an [internal redirect](#) to the specified URI;
- X-Accel-Limit-Rate sets the [rate limit](#) for transmission of a response to a client;
- X-Accel-Buffering enables or disables [buffering](#) of a response;
- X-Accel-Charset sets the desired [charset](#) of a response.

### scgi\_intercept\_errors

SYNTAX: **scgi\_intercept\_errors** on | off;

DEFAULT off

CONTEXT: http, server, location

Determines whether an SCGI server responses with codes greater than or equal to 300 should be passed to a client or be intercepted and redirected to nginx for processing with the [error\\_page](#) directive.

### scgi\_limit\_rate

SYNTAX: **scgi\_limit\_rate** *rate*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.7.

Limits the speed of reading the response from the SCGI server. The *rate* is specified in bytes per second. The zero value disables rate limiting. The limit is set per a request, and so if nginx simultaneously opens two connections to the SCGI server, the overall rate will be twice as much as the specified limit. The limitation works only if [buffering](#) of responses from the SCGI server is enabled.

### **scgi\_max\_temp\_file\_size**

SYNTAX: **scgi\_max\_temp\_file\_size** *size*;

DEFAULT 1024m

CONTEXT: http, server, location

When [buffering](#) of responses from the SCGI server is enabled, and the whole response does not fit into the buffers set by the [scgi\\_buffer\\_size](#) and [scgi\\_buffers](#) directives, a part of the response can be saved to a temporary file. This directive sets the maximum *size* of the temporary file. The size of data written to the temporary file at a time is set by the [scgi\\_temp\\_file\\_write\\_size](#) directive.

The zero value disables buffering of responses to temporary files.

This restriction does not apply to responses that will be [cached](#) or [stored](#) on disk.

### **scgi\_next\_upstream**

SYNTAX: **scgi\_next\_upstream** error | timeout | invalid\_header |  
http\_500 | http\_503 | http\_403 | http\_404 | http\_429 |  
non\_idempotent | off ...;

DEFAULT error timeout

CONTEXT: http, server, location

Specifies in which cases a request should be passed to the next server:

error

an error occurred while establishing a connection with the server, passing a request to it, or reading the response header;

timeout

a timeout has occurred while establishing a connection with the server, passing a request to it, or reading the response header;

invalid\_header

a server returned an empty or invalid response;

http\_500

a server returned a response with the code 500;

http\_503

a server returned a response with the code 503;

http\_403

a server returned a response with the code 403;

`http_404`  
a server returned a response with the code 404;

`http_429`  
a server returned a response with the code 429 (1.11.13);

`non_idempotent`  
normally, requests with a [non-idempotent](#) method (POST, LOCK, PATCH) are not passed to the next server if a request has been sent to an upstream server (1.9.13); enabling this option explicitly allows retrying such requests;

`off`  
disables passing a request to the next server.

One should bear in mind that passing a request to the next server is only possible if nothing has been sent to a client yet. That is, if an error or timeout occurs in the middle of the transferring of a response, fixing this is impossible.

The directive also defines what is considered an [unsuccessful attempt](#) of communication with a server. The cases of `error`, `timeout` and `invalid_header` are always considered unsuccessful attempts, even if they are not specified in the directive. The cases of `http_500`, `http_503`, and `http_429` are considered unsuccessful attempts only if they are specified in the directive. The cases of `http_403` and `http_404` are never considered unsuccessful attempts.

Passing a request to the next server can be limited by [the number of tries](#) and by [time](#).

### **scgi\_next\_upstream\_timeout**

SYNTAX: **scgi\_next\_upstream\_timeout** *time*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.5.

Limits the time during which a request can be passed to the [next server](#). The 0 value turns off this limitation.

### **scgi\_next\_upstream\_tries**

SYNTAX: **scgi\_next\_upstream\_tries** *number*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.5.

Limits the number of possible tries for passing a request to the [next server](#). The 0 value turns off this limitation.

## scgi\_no\_cache

SYNTAX: **scgi\_no\_cache** *string* ...;

DEFAULT —

CONTEXT: http, server, location

Defines conditions under which the response will not be saved to a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be saved:

```
scgi_no_cache $cookie_nocache $arg_nocache$arg_comment;  
scgi_no_cache $http_pragma $http_authorization;
```

Can be used along with the [scgi\\_cache\\_bypass](#) directive.

## scgi\_param

SYNTAX: **scgi\_param** *parameter value* [if\_not\_empty];

DEFAULT —

CONTEXT: http, server, location

Sets a *parameter* that should be passed to the SCGI server. The *value* can contain text, variables, and their combination. These directives are inherited from the previous configuration level if and only if there are no `scgi_param` directives defined on the current level.

Standard [CGI environment variables](#) should be provided as SCGI headers, see the `scgi_params` file provided in the distribution:

```
location / {  
    include scgi_params;  
    ...  
}
```

If the directive is specified with `if_not_empty` (1.1.11) then such a parameter will be passed to the server only if its value is not empty:

```
scgi_param HTTPS $https if_not_empty;
```

## scgi\_pass

SYNTAX: **scgi\_pass** *address*;

DEFAULT —

CONTEXT: location, if in location

Sets the address of an SCGI server. The address can be specified as a domain name or IP address, and a port:

```
scgi_pass localhost:9000;
```

or as a UNIX-domain socket path:

```
scgi_pass unix:/tmp/scgi.socket;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a [server group](#).

Parameter value can contain variables. In this case, if an address is specified as a domain name, the name is searched among the described [server groups](#), and, if not found, is determined using a [resolver](#).

### scgi\_pass\_header

SYNTAX: **scgi\_pass\_header** *field*;

DEFAULT —

CONTEXT: http, server, location

Permits passing [otherwise disabled](#) header fields from an SCGI server to a client.

### scgi\_pass\_request\_body

SYNTAX: **scgi\_pass\_request\_body** on | off;

DEFAULT on

CONTEXT: http, server, location

Indicates whether the original request body is passed to the SCGI server. See also the [scgi\\_pass\\_request\\_headers](#) directive.

### scgi\_pass\_request\_headers

SYNTAX: **scgi\_pass\_request\_headers** on | off;

DEFAULT on

CONTEXT: http, server, location

Indicates whether the header fields of the original request are passed to the SCGI server. See also the [scgi\\_pass\\_request\\_body](#) directive.

### scgi\_read\_timeout

SYNTAX: **scgi\_read\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for reading a response from the SCGI server. The timeout is set only between two successive read operations, not for the transmission of the whole response. If the SCGI server does not transmit anything within this time, the connection is closed.

### scgi\_request\_buffering

SYNTAX: **scgi\_request\_buffering** on | off;  
DEFAULT on  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Enables or disables buffering of a client request body.

When buffering is enabled, the entire request body is [read](#) from the client before sending the request to an SCGI server.

When buffering is disabled, the request body is sent to the SCGI server immediately as it is received. In this case, the request cannot be passed to the [next server](#) if nginx already started sending the request body.

When HTTP/1.1 chunked transfer encoding is used to send the original request body, the request body will be buffered regardless of the directive value.

### scgi\_send\_timeout

SYNTAX: **scgi\_send\_timeout** *time*;  
DEFAULT 60s  
CONTEXT: http, server, location

Sets a timeout for transmitting a request to the SCGI server. The timeout is set only between two successive write operations, not for the transmission of the whole request. If the SCGI server does not receive anything within this time, the connection is closed.

### scgi\_socket\_keepalive

SYNTAX: **scgi\_socket\_keepalive** on | off;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.15.6.

Configures the “TCP keepalive” behavior for outgoing connections to an SCGI server. By default, the operating system’s settings are in effect for the socket. If the directive is set to the value “on”, the `SO_KEEPALIVE` socket option is turned on for the socket.

### scgi\_store

SYNTAX: **scgi\_store** on | off | *string*;  
DEFAULT off  
CONTEXT: http, server, location

Enables saving of files to a disk. The `on` parameter saves files with paths corresponding to the directives [alias](#) or [root](#). The `off` parameter disables saving of files. In addition, the file name can be set explicitly using the *string* with variables:



```
scgi_store /data/www$original_uri;
```

The modification time of files is set according to the received Last-Modified response header field. The response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the persistent store can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both saved files and a directory holding temporary files, set by the [scgi\\_temp\\_path](#) directive, are put on the same file system.

This directive can be used to create local copies of static unchangeable files, e.g.:

```
location /images/ {
    root          /data/www;
    error_page    404 = /fetch$uri;
}

location /fetch/ {
    internal;

    scgi_pass     backend:9000;
    ...

    scgi_store    on;
    scgi_store_access user:rw group:rw all:r;
    scgi_temp_path /data/temp;

    alias         /data/www/;
}
```

### scgi\_store\_access

SYNTAX: **scgi\_store\_access** *users:permissions* ...;

DEFAULT user:rw

CONTEXT: http, server, location

Sets access permissions for newly created files and directories, e.g.:

```
scgi_store_access user:rw group:rw all:r;
```

If any group or all access permissions are specified then user permissions may be omitted:

```
scgi_store_access group:rw all:r;
```

### scgi\_temp\_file\_write\_size

SYNTAX: **scgi\_temp\_file\_write\_size** *size*;

DEFAULT 8k|16k

CONTEXT: http, server, location

Limits the *size* of data written to a temporary file at a time, when buffering of responses from the SCGI server to temporary files is enabled. By default, *size* is limited by two buffers set by the [scgi\\_buffer\\_size](#) and [scgi\\_buffers](#) directives. The maximum size of a temporary file is set by the [scgi\\_max\\_temp\\_file\\_size](#) directive.

### scgi\_temp\_path

SYNTAX: **scgi\_temp\_path** *path* [*level1* [*level2* [*level3*]]];

DEFAULT `scgi_temp`

CONTEXT: http, server, location

Defines a directory for storing temporary files with data received from SCGI servers. Up to three-level subdirectory hierarchy can be used underneath the specified directory. For example, in the following configuration

```
scgi_temp_path /spool/nginx/scgi_temp 1 2;
```

a temporary file might look like this:

```
/spool/nginx/scgi_temp/7/45/00000123457
```

See also the `use_temp_path` parameter of the [scgi\\_cache\\_path](#) directive.

## 2.43 Module ngx\_http\_secure\_link\_module

2.43.1 Summary	290
2.43.2 Directives	290
<a href="#">secure_link</a>	290
<a href="#">secure_link_md5</a>	291
<a href="#">secure_link_secret</a>	291
2.43.3 Embedded Variables	292

### 2.43.1 Summary

The `ngx_http_secure_link_module` (0.7.18) is used to check authenticity of requested links, protect resources from unauthorized access, and limit link lifetime.

The authenticity of a requested link is verified by comparing the checksum value passed in a request with the value computed for the request. If a link has a limited lifetime and the time has expired, the link is considered outdated. The status of these checks is made available in the `$secure_link` variable.

The module provides two alternative operation modes. The first mode is enabled by the [secure\\_link\\_secret](#) directive and is used to check authenticity of requested links as well as protect resources from unauthorized access. The second mode (0.8.50) is enabled by the [secure\\_link](#) and [secure\\_link\\_md5](#) directives and is also used to limit lifetime of links.

This module is not built by default, it should be enabled with the `--with-http_secure_link_module` configuration parameter.

### 2.43.2 Directives

#### `secure_link`

SYNTAX: **secure\_link** *expression*;

DEFAULT —

CONTEXT: http, server, location

Defines a string with variables from which the checksum value and lifetime of a link will be extracted.

Variables used in an *expression* are usually associated with a request; see [example](#) below.

The checksum value extracted from the string is compared with the MD5 hash value of the expression defined by the [secure\\_link\\_md5](#) directive. If the checksums are different, the `$secure_link` variable is set to an empty string. If the checksums are the same, the link lifetime is checked. If the link has a limited lifetime and the time has expired, the `$secure_link` variable is set to “0”. Otherwise, it is set to “1”. The MD5 hash value passed in a request is encoded in [base64url](#).

If a link has a limited lifetime, the expiration time is set in seconds since Epoch (Thu, 01 Jan 1970 00:00:00 GMT). The value is specified in the expression after the MD5 hash, and is separated by a comma. The expiration

time passed in a request is available through the `$secure_link_expires` variable for a use in the `secure_link_md5` directive. If the expiration time is not specified, a link has the unlimited lifetime.

### `secure_link_md5`

SYNTAX: **`secure_link_md5`** *expression*;

DEFAULT —

CONTEXT: http, server, location

Defines an expression for which the MD5 hash value will be computed and compared with the value passed in a request.

The expression should contain the secured part of a link (resource) and a secret ingredient. If the link has a limited lifetime, the expression should also contain `$secure_link_expires`.

To prevent unauthorized access, the expression may contain some information about the client, such as its address and browser version.

Example:

```
location /s/ {
    secure_link $arg_md5,$arg_expires;
    secure_link_md5 "$secure_link_expires$uri$remote_addr secret";

    if ($secure_link = "") {
        return 403;
    }

    if ($secure_link = "0") {
        return 410;
    }

    ...
}
```

The `"/s/link?md5=_e4Nc3iduzkWRm01TBBNYw&expires=2147483647"` link restricts access to `"/s/link"` for the client with the IP address 127.0.0.1. The link also has the limited lifetime until January 19, 2038 (GMT).

On UNIX, the `md5` request argument value can be obtained as:

```
echo -n '2147483647/s/link127.0.0.1 secret' | \
    openssl md5 -binary | openssl base64 | tr +/ -_ | tr -d =
```

### `secure_link_secret`

SYNTAX: **`secure_link_secret`** *word*;

DEFAULT —

CONTEXT: location

Defines a secret *word* used to check authenticity of requested links.

The full URI of a requested link looks as follows:

```
/prefix/hash/link
```

where *hash* is a hexadecimal representation of the MD5 hash computed for the concatenation of the link and secret word, and *prefix* is an arbitrary string without slashes.

If the requested link passes the authenticity check, the *\$secure\_link* variable is set to the link extracted from the request URI. Otherwise, the *\$secure\_link* variable is set to an empty string.

Example:

```
location /p/ {
    secure_link_secret secret;

    if ($secure_link = "") {
        return 403;
    }

    rewrite ^ /secure/$secure_link;
}

location /secure/ {
    internal;
}
```

A request of “/p/5e814704a28d9bc1914ff19fa0c4a00a/link” will be internally redirected to “/secure/link”.

On UNIX, the hash value for this example can be obtained as:

```
echo -n 'linksecret' | openssl md5 -hex
```

### 2.43.3 Embedded Variables

#### *\$secure\_link*

The status of a link check. The specific value depends on the selected operation mode.

#### *\$secure\_link\_expires*

The lifetime of a link passed in a request; intended to be used only in the [secure\\_link\\_md5](#) directive.

## 2.44 Module ngx\_http\_session\_log\_module

2.44.1 Summary	293
2.44.2 Example Configuration	293
2.44.3 Directives	293
session_log	293
session_log_format	293
session_log_zone	294
2.44.4 Embedded Variables	294

### 2.44.1 Summary

The `ngx_http_session_log_module` module enables logging sessions (that is, aggregates of multiple HTTP requests) instead of individual HTTP requests.

This module is available as part of our [commercial subscription](#).

### 2.44.2 Example Configuration

The following configuration sets up a session log and maps requests to sessions according to the request client address and User-Agent request header field:

```
session_log_zone /path/to/log format=combined
                 zone=one:1m timeout=30s
                 md5=$binary_remote_addr$http_user_agent;

location /media/ {
    session_log one;
}
```

### 2.44.3 Directives

#### session\_log

SYNTAX: **session\_log** *name* | `off`;

DEFAULT `off`

CONTEXT: `http`, `server`, `location`

Enables the use of the specified session log. The special value `off` cancels the effect of the `session_log` directives inherited from the previous configuration level.

#### session\_log\_format

SYNTAX: **session\_log\_format** *name string* ...;

DEFAULT `combined "..."`

CONTEXT: `http`

Specifies the output format of a log. The value of the *\$body\_bytes\_sent* variable is aggregated across all requests in a session. The values of all other variables available for logging correspond to the first request in a session.

### **session\_log\_zone**

SYNTAX: **session\_log\_zone** *path* zone=*name:size* [*format=format*]  
[*timeout=time*] [*id=id*] [*md5=md5*];

DEFAULT —

CONTEXT: http

Sets the path to a log file and configures the shared memory zone that is used to store currently active sessions.

A session is considered active for as long as the time elapsed since the last request in the session does not exceed the specified *timeout* (by default, 30 seconds). Once a session is no longer active, it is written to the log.

The *id* parameter identifies the session to which a request is mapped. The *id* parameter is set to the hexadecimal representation of an MD5 hash (for example, obtained from a cookie using variables). If this parameter is not specified or does not represent the valid MD5 hash, nginx computes the MD5 hash from the value of the *md5* parameter and creates a new session using this hash. Both the *id* and *md5* parameters can contain variables.

The *format* parameter sets the custom session log format configured by the [session\\_log\\_format](#) directive. If *format* is not specified, the predefined “combined” format is used.

## **2.44.4 Embedded Variables**

The `ngx_http_session_log_module` module supports two embedded variables:

*\$session\_log\_id*

current session ID;

*\$session\_log\_binary\_id*

current session ID in binary form (16 bytes).

## 2.45 Module ngx\_http\_slice\_module

2.45.1 Summary	295
2.45.2 Example Configuration	295
2.45.3 Directives	295
<a href="#">slice</a>	295
2.45.4 Embedded Variables	295

### 2.45.1 Summary

The `ngx_http_slice_module` module (1.9.8) is a filter that splits a request into subrequests, each returning a certain range of response. The filter provides more effective caching of big responses.

This module is not built by default, it should be enabled with the `--with-http_slice_module` configuration parameter.

### 2.45.2 Example Configuration

```
location / {
    slice          1m;
    proxy_cache    cache;
    proxy_cache_key $uri$is_args$args$slice_range;
    proxy_set_header Range $slice_range;
    proxy_cache_valid 200 206 1h;
    proxy_pass      http://localhost:8000;
}
```

In this example, the response is split into 1-megabyte cacheable slices.

### 2.45.3 Directives

#### `slice`

SYNTAX: **slice** *size*;

DEFAULT 0

CONTEXT: http, server, location

Sets the *size* of the slice. The zero value disables splitting responses into slices. Note that a too low value may result in excessive memory usage and opening a large number of files.

In order for a subrequest to return the required range, the `$slice_range` variable should be [passed](#) to the proxied server as the Range request header field. If [caching](#) is enabled, `$slice_range` should be added to the [cache key](#) and caching of responses with 206 status code should be [enabled](#).

### 2.45.4 Embedded Variables

The `ngx_http_slice_module` module supports the following embedded variables:



*\$slice\_range*

the current slice range in [HTTP byte range](#) format, for example,  
bytes=0-1048575.

## 2.46 Module ngx\_http\_split\_clients\_module

2.46.1 Summary	297
2.46.2 Example Configuration	297
2.46.3 Directives	297
split_clients	297

### 2.46.1 Summary

The `ngx_http_split_clients_module` module creates variables suitable for A/B testing, also known as split testing.

### 2.46.2 Example Configuration

```
http {
    split_clients "${remote_addr}AAA" $variant {
        0.5% .one;
        2.0% .two;
        *    "";
    }

    server {
        location / {
            index index${variant}.html;
        }
    }
}
```

### 2.46.3 Directives

#### split\_clients

SYNTAX: **split\_clients** *string* *\$variable* { ... }

DEFAULT —

CONTEXT: http

Creates a variable for A/B testing, for example:

```
split_clients "${remote_addr}AAA" $variant {
    0.5% .one;
    2.0% .two;
    *    "";
}
```

The value of the original string is hashed using MurmurHash2. In the example given, hash values from 0 to 21474835 (0.5%) correspond to the value `".one"` of the `$variant` variable, hash values from 21474836 to 107374180 (2%) correspond to the value `".two"`, and hash values from 107374181 to 4294967295 correspond to the value `""` (an empty string).

## 2.47 Module ngx\_http\_ssi\_module

2.47.1 Summary	298
2.47.2 Example Configuration	298
2.47.3 Directives	298
ssi	298
ssi_last_modified	298
ssi_min_file_chunk	299
ssi_silent_errors	299
ssi_types	299
ssi_value_length	299
2.47.4 SSI Commands	299
2.47.5 Embedded Variables	303

### 2.47.1 Summary

The `ngx_http_ssi_module` module is a filter that processes SSI (Server Side Includes) commands in responses passing through it. Currently, the list of supported SSI commands is incomplete.

### 2.47.2 Example Configuration

```
location / {
    ssi on;
    ...
}
```

### 2.47.3 Directives

#### ssi

SYNTAX: **ssi** on | off;

DEFAULT off

CONTEXT: http, server, location, if in location

Enables or disables processing of SSI commands in responses.

#### ssi\_last\_modified

SYNTAX: **ssi\_last\_modified** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.1.

Allows preserving the `Last-Modified` header field from the original response during SSI processing to facilitate response caching.

By default, the header field is removed as contents of the response are modified during processing and may contain dynamically generated elements or parts that are changed independently of the original response.

### **ssi\_min\_file\_chunk**

SYNTAX: **ssi\_min\_file\_chunk** *size*;

DEFAULT 1k

CONTEXT: http, server, location

Sets the minimum *size* for parts of a response stored on disk, starting from which it makes sense to send them using [sendfile](#).

### **ssi\_silent\_errors**

SYNTAX: **ssi\_silent\_errors** on | off;

DEFAULT off

CONTEXT: http, server, location

If enabled, suppresses the output of the “[an error occurred while processing the directive]” string if an error occurred during SSI processing.

### **ssi\_types**

SYNTAX: **ssi\_types** *mime-type* ...;

DEFAULT text/html

CONTEXT: http, server, location

Enables processing of SSI commands in responses with the specified MIME types in addition to “text/html”. The special value “\*” matches any MIME type (0.8.29).

### **ssi\_value\_length**

SYNTAX: **ssi\_value\_length** *length*;

DEFAULT 256

CONTEXT: http, server, location

Sets the maximum length of parameter values in SSI commands.

## **2.47.4 SSI Commands**

SSI commands have the following generic format:

```
<!--# command parameter1=value1 parameter2=value2 ... -->
```

The following commands are supported:

#### **block**

Defines a block that can be used as a stub in the `include` command. The block can contain other SSI commands. The command has the following parameter:

name  
block name.

Example:

```
<!--# block name="one" -->
stub
<!--# endblock -->
```

**config**

Sets some parameters used during SSI processing, namely:

**errmsg**

a string that is output if an error occurs during SSI processing. By default, the following string is output:

```
[an error occurred while processing the directive]
```

**timefmt**

a format string passed to the `strftime` function used to output date and time. By default, the following format is used:

```
"%A, %d-%b-%Y %H:%M:%S %Z"
```

The “%s” format is suitable to output time in seconds.

**echo**

Outputs the value of a variable. The command has the following parameters:

**var**

the variable name.

**encoding**

the encoding method. Possible values include `none`, `url`, and `entity`. By default, `entity` is used.

**default**

a non-standard parameter that sets a string to be output if a variable is undefined. By default, “(none)” is output. The command

```
<!--# echo var="name" default="no" -->
```

replaces the following sequence of commands:

```
<!--# if expr="$name" --><!--# echo var="name" --><!--#
else -->no<!--# endif -->
```

**if**

Performs a conditional inclusion. The following commands are supported:

```
<!--# if expr="..." -->
...
<!--# elif expr="..." -->
...
<!--# else -->
...
<!--# endif -->
```

```
<!--# else -->
...
<!--# endif -->
```

Only one level of nesting is currently supported. The command has the following parameter:

`expr`

expression. An expression can be:

- variable existence check:

```
<!--# if expr="$name" -->
```

- comparison of a variable with a text:

```
<!--# if expr="$name = text" -->
<!--# if expr="$name != text" -->
```

- comparison of a variable with a regular expression:

```
<!--# if expr="$name = /text/" -->
<!--# if expr="$name != /text/" -->
```

If a *text* contains variables, their values are substituted. A regular expression can contain positional and named captures that can later be used through variables, for example:

```
<!--# if expr="$name = /(.)@(?P<domain>.+)/" -->
  <!--# echo var="1" -->
  <!--# echo var="domain" -->
<!--# endif -->
```

`include`

Includes the result of another request into a response. The command has the following parameters:

`file`

specifies an included file, for example:

```
<!--# include file="footer.html" -->
```

`virtual`

specifies an included request, for example:

```
<!--# include virtual="/remote/body.php?argument=value" -->
```

Several requests specified on one page and processed by proxied or FastCGI/uwsgi/SCGI/gRPC servers run in parallel. If sequential processing is desired, the `wait` parameter should be used.

**stub**

a non-standard parameter that names the block whose content will be output if the included request results in an empty body or if an error occurs during the request processing, for example:

```
<!--# block name="one" -->&nbsp;  <!--# endblock -->
<!--# include virtual="/remote/body.php?argument=value" stub="one"
-->
```

The replacement block content is processed in the included request context.

**wait**

a non-standard parameter that instructs to wait for a request to fully complete before continuing with SSI processing, for example:

```
<!--# include virtual="/remote/body.php?argument=value" wait="yes"
-->
```

**set**

a non-standard parameter that instructs to write a successful result of request processing to the specified variable, for example:

```
<!--# include virtual="/remote/body.php?argument=value" set="one"
-->
```

The maximum size of the response is set by the [subrequest\\_output\\_buffer\\_size](#) directive (1.13.10):

```
location /remote/ {
    subrequest_output_buffer_size 64k;
    ...
}
```

Prior to version 1.13.10, only the results of responses obtained using the [ngx\\_http\\_proxy\\_module](#), [ngx\\_http\\_memcached\\_module](#), [ngx\\_http\\_fastcgi\\_module](#) (1.5.6), [ngx\\_http\\_uwsgi\\_module](#) (1.5.6), and [ngx\\_http\\_scgi\\_module](#) (1.5.6) modules could be written into variables. The maximum size of the response was set with the [proxy\\_buffer\\_size](#), [memcached\\_buffer\\_size](#), [fastcgi\\_buffer\\_size](#), [uwsgi\\_buffer\\_size](#), and [scgi\\_buffer\\_size](#) directives.

**set**

Sets a value of a variable. The command has the following parameters:

**var**

the variable name.

**value**

the variable value. If an assigned value contains variables, their values are substituted.

## 2.47.5 Embedded Variables

The `ngx_http_ssi_module` module supports two embedded variables:

*\$date\_local*

current time in the local time zone. The format is set by the `config` command with the `timefmt` parameter.

*\$date\_gmt*

current time in GMT. The format is set by the `config` command with the `timefmt` parameter.



## 2.48 Module ngx\_http\_ssl\_module

2.48.1	Summary	304
2.48.2	Example Configuration	305
2.48.3	Directives	305
	ssl	305
	ssl_buffer_size	305
	ssl_certificate	306
	ssl_certificate_key	307
	ssl_ciphers	307
	ssl_client_certificate	307
	ssl_conf_command	307
	ssl_crl	308
	ssl_dhparam	308
	ssl_early_data	308
	ssl_ecdh_curve	309
	ssl_ocsp	309
	ssl_ocsp_cache	310
	ssl_ocsp_responder	310
	ssl_password_file	310
	ssl_prefer_server_ciphers	311
	ssl_protocols	311
	ssl_reject_handshake	311
	ssl_session_cache	311
	ssl_session_ticket_key	312
	ssl_session_tickets	313
	ssl_session_timeout	313
	ssl_stapling	313
	ssl_stapling_file	313
	ssl_stapling_responder	314
	ssl_stapling_verify	314
	ssl_trusted_certificate	314
	ssl_verify_client	314
	ssl_verify_depth	315
2.48.4	Error Processing	315
2.48.5	Embedded Variables	315

### 2.48.1 Summary

The `ngx_http_ssl_module` module provides the necessary support for HTTPS.

This module is not built by default, it should be enabled with the `--with-http_ssl_module` configuration parameter.

This module requires the [OpenSSL](#) library.

## 2.48.2 Example Configuration

To reduce the processor load it is recommended to

- set the number of [worker processes](#) equal to the number of processors,
- enable [keep-alive](#) connections,
- enable the [shared](#) session cache,
- disable the [built-in](#) session cache,
- and possibly increase the session [lifetime](#) (by default, 5 minutes):

```
worker_processes auto;

http {
    ...

    server {
        listen      443 ssl;
        keepalive_timeout 70;

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers  AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate /usr/local/nginx/conf/cert.pem;
        ssl_certificate_key /usr/local/nginx/conf/cert.key;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 10m;

        ...
    }
}
```

## 2.48.3 Directives

### ssl

SYNTAX: **ssl** on | off;

DEFAULT off

CONTEXT: http, server

This directive was made obsolete in version 1.15.0. The `ssl` parameter of the [listen](#) directive should be used instead.

### ssl\_buffer\_size

SYNTAX: **ssl\_buffer\_size** *size*;

DEFAULT 16k

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.5.9.

Sets the size of the buffer used for sending data.

By default, the buffer size is 16k, which corresponds to minimal overhead when sending big responses. To minimize Time To First Byte it may be beneficial to use smaller values, for example:

```
ssl_buffer_size 4k;
```

## ssl\_certificate

SYNTAX: **ssl\_certificate** *file*;

DEFAULT —

CONTEXT: http, server

Specifies a *file* with the certificate in the PEM format for the given virtual server. If intermediate certificates should be specified in addition to a primary certificate, they should be specified in the same file in the following order: the primary certificate comes first, then the intermediate certificates. A secret key in the PEM format may be placed in the same file.

Since version 1.11.0, this directive can be specified multiple times to load certificates of different types, for example, RSA and ECDSA:

```
server {
    listen          443 ssl;
    server_name     example.com;

    ssl_certificate  example.com.rsa.crt;
    ssl_certificate_key example.com.rsa.key;

    ssl_certificate  example.com.ecdsa.crt;
    ssl_certificate_key example.com.ecdsa.key;

    ...
}
```

Only OpenSSL 1.0.2 or higher supports separate [certificate chains](#) for different certificates. With older versions, only one certificate chain can be used.

Since version 1.15.9, variables can be used in the *file* name when using OpenSSL 1.0.2 or higher:

```
ssl_certificate      $ssl_server_name.crt;
ssl_certificate_key  $ssl_server_name.key;
```

Note that using variables implies that a certificate will be loaded for each SSL handshake, and this may have a negative impact on performance.

The value `data:$variable` can be specified instead of the *file* (1.15.10), which loads a certificate from a variable without using intermediate files. Note that inappropriate use of this syntax may have its security implications, such as writing secret key data to [error log](#).

It should be kept in mind that due to the HTTPS protocol limitations for maximum interoperability virtual servers should listen on [different IP addresses](#).

## ssl\_certificate\_key

SYNTAX: **ssl\_certificate\_key** *file*;

DEFAULT —

CONTEXT: http, server

Specifies a *file* with the secret key in the PEM format for the given virtual server.

The value `engine:name:id` can be specified instead of the *file* (1.7.9), which loads a secret key with a specified *id* from the OpenSSL engine *name*.

The value `data:$variable` can be specified instead of the *file* (1.15.10), which loads a secret key from a variable without using intermediate files. Note that inappropriate use of this syntax may have its security implications, such as writing secret key data to [error log](#).

Since version 1.15.9, variables can be used in the *file* name when using OpenSSL 1.0.2 or higher.

## ssl\_ciphers

SYNTAX: **ssl\_ciphers** *ciphers*;

DEFAULT HIGH:!aNULL:!MD5

CONTEXT: http, server

Specifies the enabled ciphers. The ciphers are specified in the format understood by the OpenSSL library, for example:

```
ssl_ciphers ALL:!aNULL:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
```

The full list can be viewed using the “`openssl ciphers`” command.

The previous versions of nginx used [different](#) ciphers by default.

## ssl\_client\_certificate

SYNTAX: **ssl\_client\_certificate** *file*;

DEFAULT —

CONTEXT: http, server

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) client certificates and OCSP responses if [ssl\\_stapling](#) is enabled.

The list of certificates will be sent to clients. If this is not desired, the [ssl\\_trusted\\_certificate](#) directive can be used.

## ssl\_conf\_command

SYNTAX: **ssl\_conf\_command** *name value*;

DEFAULT —

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

Sets arbitrary OpenSSL configuration [commands](#).

The directive is supported when using OpenSSL 1.0.2 or higher.

Several `ssl_conf_command` directives can be specified on the same level:

```
ssl_conf_command Options PrioritizeChaCha;  
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

These directives are inherited from the previous configuration level if and only if there are no `ssl_conf_command` directives defined on the current level.

Note that configuring OpenSSL directly might result in unexpected behavior.

### ssl\_crl

SYNTAX: **ssl\_crl** *file*;  
DEFAULT —  
CONTEXT: http, server  
THIS DIRECTIVE APPEARED IN VERSION 0.8.7.

Specifies a *file* with revoked certificates (CRL) in the PEM format used to [verify](#) client certificates.

### ssl\_dhparam

SYNTAX: **ssl\_dhparam** *file*;  
DEFAULT —  
CONTEXT: http, server  
THIS DIRECTIVE APPEARED IN VERSION 0.7.2.

Specifies a *file* with DH parameters for DHE ciphers.

By default no parameters are set, and therefore DHE ciphers will not be used.

Prior to version 1.11.0, builtin parameters were used by default.

### ssl\_early\_data

SYNTAX: **ssl\_early\_data** on | off;  
DEFAULT off  
CONTEXT: http, server  
THIS DIRECTIVE APPEARED IN VERSION 1.15.3.

Enables or disables TLS 1.3 [early data](#).

Requests sent within early data are subject to [replay attacks](#). To protect against such attacks at the application layer, the `$ssl_early_data` variable should be used.

```
proxy_set_header Early-Data $ssl_early_data;
```

The directive is supported when using OpenSSL 1.1.1 or higher (1.15.4) and [BoringSSL](#).

### ssl\_ecdh\_curve

SYNTAX: **ssl\_ecdh\_curve** *curve*;  
DEFAULT `auto`  
CONTEXT: `http, server`  
THIS DIRECTIVE APPEARED IN VERSIONS 1.1.0 AND 1.0.6.

Specifies a *curve* for ECDHE ciphers.

When using OpenSSL 1.0.2 or higher, it is possible to specify multiple curves (1.11.0), for example:

```
ssl_ecdh_curve prime256v1:secp384r1;
```

The special value `auto` (1.11.0) instructs nginx to use a list built into the OpenSSL library when using OpenSSL 1.0.2 or higher, or `prime256v1` with older versions.

Prior to version 1.11.0, the `prime256v1` curve was used by default.

When using OpenSSL 1.0.2 or higher, this directive sets the list of curves supported by the server. Thus, in order for ECDSA certificates to work, it is important to include the curves used in the certificates.

### ssl\_ocsp

SYNTAX: **ssl\_ocsp** `on` | `off` | `leaf`;  
DEFAULT `off`  
CONTEXT: `http, server`  
THIS DIRECTIVE APPEARED IN VERSION 1.19.0.

Enables OCSP validation of the client certificate chain. The `leaf` parameter enables validation of the client certificate only.

For the OCSP validation to work, the [ssl\\_verify\\_client](#) directive should be set to `on` or `optional`.

To resolve the OCSP responder hostname, the [resolver](#) directive should also be specified.

Example:

```
ssl_verify_client on;  
ssl_ocsp          on;  
resolver          192.0.2.1;
```

## ssl\_ocsp\_cache

SYNTAX: **ssl\_ocsp\_cache** *off* | [*shared:name:size*];

DEFAULT *off*

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.19.0.

Sets *name* and *size* of the cache that stores client certificates status for OCSP validation. The cache is shared between all worker processes. A cache with the same name can be used in several virtual servers.

The *off* parameter prohibits the use of the cache.

## ssl\_ocsp\_responder

SYNTAX: **ssl\_ocsp\_responder** *url*;

DEFAULT —

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.19.0.

Overrides the URL of the OCSP responder specified in the “[Authority Information Access](#)” certificate extension for [validation](#) of client certificates.

Only “[http://](#)” OCSP responders are supported:

```
ssl_ocsp_responder http://ocsp.example.com/;
```

## ssl\_password\_file

SYNTAX: **ssl\_password\_file** *file*;

DEFAULT —

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.7.3.

Specifies a *file* with passphrases for [secret keys](#) where each passphrase is specified on a separate line. Passphrases are tried in turn when loading the key.

Example:

```
http {
    ssl_password_file /etc/keys/global.pass;
    ...

    server {
        server_name www1.example.com;
        ssl_certificate_key /etc/keys/first.key;
    }

    server {
        server_name www2.example.com;

        # named pipe can also be used instead of a file
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

## ssl\_prefer\_server\_ciphers

SYNTAX: **ssl\_prefer\_server\_ciphers** on | off;  
DEFAULT off  
CONTEXT: http, server

Specifies that server ciphers should be preferred over client ciphers when using the SSLv3 and TLS protocols.

## ssl\_protocols

SYNTAX: **ssl\_protocols** [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];  
DEFAULT TLSv1 TLSv1.1 TLSv1.2  
CONTEXT: http, server

Enables the specified protocols.

The TLSv1.1 and TLSv1.2 parameters (1.1.13, 1.0.12) work only when OpenSSL 1.0.1 or higher is used.

The TLSv1.3 parameter (1.13.0) works only when OpenSSL 1.1.1 or higher is used.

## ssl\_reject\_handshake

SYNTAX: **ssl\_reject\_handshake** on | off;  
DEFAULT off  
CONTEXT: http, server  
THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

If enabled, SSL handshakes in the [server](#) block will be rejected.

For example, in the following configuration, SSL handshakes with server names other than `example.com` are rejected:

```
server {
    listen          443 ssl default_server;
    ssl_reject_handshake on;
}

server {
    listen          443 ssl;
    server_name     example.com;
    ssl_certificate example.com.crt;
    ssl_certificate_key example.com.key;
}
```

## ssl\_session\_cache

SYNTAX: **ssl\_session\_cache** off | none | [builtin[:size]] [shared:name:size];  
DEFAULT none  
CONTEXT: http, server



Sets the types and sizes of caches that store session parameters. A cache can be of any of the following types:

`off`

the use of a session cache is strictly prohibited: nginx explicitly tells a client that sessions may not be reused.

`none`

the use of a session cache is gently disallowed: nginx tells a client that sessions may be reused, but does not actually store session parameters in the cache.

`builtin`

a cache built in OpenSSL; used by one worker process only. The cache size is specified in sessions. If size is not given, it is equal to 20480 sessions. Use of the built-in cache can cause memory fragmentation.

`shared`

a cache shared between all worker processes. The cache size is specified in bytes; one megabyte can store about 4000 sessions. Each shared cache should have an arbitrary name. A cache with the same name can be used in several virtual servers. It is also used to automatically generate, store, and periodically rotate TLS session ticket keys (1.23.2) unless configured explicitly using the [ssl\\_session\\_ticket\\_key](#) directive.

Both cache types can be used simultaneously, for example:

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

but using only shared cache without the built-in cache should be more efficient.

### **ssl\_session\_ticket\_key**

SYNTAX: **ssl\_session\_ticket\_key** *file*;

DEFAULT —

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Sets a *file* with the secret key used to encrypt and decrypt TLS session tickets. The directive is necessary if the same key has to be shared between multiple servers. By default, a randomly generated key is used.

If several keys are specified, only the first key is used to encrypt TLS session tickets. This allows configuring key rotation, for example:

```
ssl_session_ticket_key current.key;  
ssl_session_ticket_key previous.key;
```

The *file* must contain 80 or 48 bytes of random data and can be created using the following command:

```
openssl rand 80 > ticket.key
```

Depending on the file size either AES256 (for 80-byte keys, 1.11.8) or AES128 (for 48-byte keys) is used for encryption.

### ssl\_session\_tickets

SYNTAX: **ssl\_session\_tickets** on | off;  
DEFAULT on  
CONTEXT: http, server  
THIS DIRECTIVE APPEARED IN VERSION 1.5.9.

Enables or disables session resumption through [TLS session tickets](#).

### ssl\_session\_timeout

SYNTAX: **ssl\_session\_timeout** *time*;  
DEFAULT 5m  
CONTEXT: http, server

Specifies a time during which a client may reuse the session parameters.

### ssl\_stapling

SYNTAX: **ssl\_stapling** on | off;  
DEFAULT off  
CONTEXT: http, server  
THIS DIRECTIVE APPEARED IN VERSION 1.3.7.

Enables or disables [stapling of OCSP responses](#) by the server. Example:

```
ssl_stapling on;  
resolver 192.0.2.1;
```

For the OCSP stapling to work, the certificate of the server certificate issuer should be known. If the [ssl\\_certificate](#) file does not contain intermediate certificates, the certificate of the server certificate issuer should be present in the [ssl\\_trusted\\_certificate](#) file.

For a resolution of the OCSP responder hostname, the [resolver](#) directive should also be specified.

### ssl\_stapling\_file

SYNTAX: **ssl\_stapling\_file** *file*;  
DEFAULT —  
CONTEXT: http, server  
THIS DIRECTIVE APPEARED IN VERSION 1.3.7.

When set, the stapled OCSP response will be taken from the specified *file* instead of querying the OCSP responder specified in the server certificate.

The file should be in the DER format as produced by the “`openssl ocsp`” command.

## ssl\_stapling\_responder

SYNTAX: **ssl\_stapling\_responder** *url*;

DEFAULT —

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.3.7.

Overrides the URL of the OCSP responder specified in the “[Authority Information Access](#)” certificate extension.

Only “http://” OCSP responders are supported:

```
ssl_stapling_responder http://ocsp.example.com/;
```

## ssl\_stapling\_verify

SYNTAX: **ssl\_stapling\_verify** on | off;

DEFAULT off

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.3.7.

Enables or disables verification of OCSP responses by the server.

For verification to work, the certificate of the server certificate issuer, the root certificate, and all intermediate certificates should be configured as trusted using the [ssl\\_trusted\\_certificate](#) directive.

## ssl\_trusted\_certificate

SYNTAX: **ssl\_trusted\_certificate** *file*;

DEFAULT —

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.3.7.

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) client certificates and OCSP responses if [ssl\\_stapling](#) is enabled.

In contrast to the certificate set by [ssl\\_client\\_certificate](#), the list of these certificates will not be sent to clients.

## ssl\_verify\_client

SYNTAX: **ssl\_verify\_client** on | off | optional | optional\_no\_ca;

DEFAULT off

CONTEXT: http, server

Enables verification of client certificates. The verification result is stored in the [\\$ssl\\_client\\_verify](#) variable.

The `optional` parameter (0.8.7+) requests the client certificate and verifies it if the certificate is present.

The `optional_no_ca` parameter (1.3.8, 1.2.5) requests the client certificate but does not require it to be signed by a trusted CA certificate. This is intended for the use in cases when a service that is external to nginx

performs the actual certificate verification. The contents of the certificate is accessible through the `$ssl_client_cert` variable.

### `ssl_verify_depth`

SYNTAX: `ssl_verify_depth` *number*;

DEFAULT 1

CONTEXT: http, server

Sets the verification depth in the client certificates chain.

## 2.48.4 Error Processing

The `ngx_http_ssl_module` module supports several non-standard error codes that can be used for redirects using the `error_page` directive:

495

an error has occurred during the client certificate verification;

496

a client has not presented the required certificate;

497

a regular request has been sent to the HTTPS port.

The redirection happens after the request is fully parsed and the variables, such as `$request_uri`, `$uri`, `$args` and others, are available.

## 2.48.5 Embedded Variables

The `ngx_http_ssl_module` module supports embedded variables:

`$ssl_alpn_protocol`

returns the protocol selected by ALPN during the SSL handshake, or an empty string otherwise (1.21.4);

`$ssl_cipher`

returns the name of the cipher used for an established SSL connection;

`$ssl_ciphers`

returns the list of ciphers supported by the client (1.11.7). Known ciphers are listed by names, unknown are shown in hexadecimal, for example:

```
AES128-SHA:AES256-SHA:0x00ff
```

The variable is fully supported only when using OpenSSL version 1.0.2 or higher. With older versions, the variable is available only for new sessions and lists only known ciphers.

`$ssl_client_escaped_cert`

returns the client certificate in the PEM format (urlencoded) for an established SSL connection (1.13.5);

*\$ssl\_client\_cert*

returns the client certificate in the PEM format for an established SSL connection, with each line except the first prepended with the tab character; this is intended for the use in the [proxy\\_set\\_header](#) directive;

The variable is deprecated, the *\$ssl\_client\_escaped\_cert* variable should be used instead.

*\$ssl\_client\_fingerprint*

returns the SHA1 fingerprint of the client certificate for an established SSL connection (1.7.1);

*\$ssl\_client\_i\_dn*

returns the “issuer DN” string of the client certificate for an established SSL connection according to [RFC 2253](#) (1.11.6);

*\$ssl\_client\_i\_dn\_legacy*

returns the “issuer DN” string of the client certificate for an established SSL connection;

Prior to version 1.11.6, the variable name was *\$ssl\_client\_i\_dn*.

*\$ssl\_client\_raw\_cert*

returns the client certificate in the PEM format for an established SSL connection;

*\$ssl\_client\_s\_dn*

returns the “subject DN” string of the client certificate for an established SSL connection according to [RFC 2253](#) (1.11.6);

*\$ssl\_client\_s\_dn\_legacy*

returns the “subject DN” string of the client certificate for an established SSL connection;

Prior to version 1.11.6, the variable name was *\$ssl\_client\_s\_dn*.

*\$ssl\_client\_serial*

returns the serial number of the client certificate for an established SSL connection;

*\$ssl\_client\_v\_end*

returns the end date of the client certificate (1.11.7);

*\$ssl\_client\_v\_remain*

returns the number of days until the client certificate expires (1.11.7);

*\$ssl\_client\_v\_start*

returns the start date of the client certificate (1.11.7);

*\$ssl\_client\_verify*

returns the result of client certificate verification: “SUCCESS”, “FAILED: *reason*”, and “NONE” if a certificate was not present;

Prior to version 1.11.7, the “FAILED” result did not contain the *reason* string.

#### *\$ssl\_curve*

returns the negotiated curve used for SSL handshake key exchange process (1.21.5). Known curves are listed by names, unknown are shown in hexadecimal, for example:

```
prime256v1
```

The variable is supported only when using OpenSSL version 3.0 or higher. With older versions, the variable value will be an empty string.

#### *\$ssl\_curves*

returns the list of curves supported by the client (1.11.7). Known curves are listed by names, unknown are shown in hexadecimal, for example:

```
0x001d:prime256v1:secp521r1:secp384r1
```

The variable is supported only when using OpenSSL version 1.0.2 or higher. With older versions, the variable value will be an empty string.

The variable is available only for new sessions.

#### *\$ssl\_early\_data*

returns “1” if TLS 1.3 [early data](#) is used and the handshake is not complete, otherwise “” (1.15.3).

#### *\$ssl\_protocol*

returns the protocol of an established SSL connection;

#### *\$ssl\_server\_name*

returns the server name requested through [SNI](#) (1.7.0);

#### *\$ssl\_session\_id*

returns the session identifier of an established SSL connection;

#### *\$ssl\_session\_reused*

returns “r” if an SSL session was reused, or “.” otherwise (1.5.11).

## 2.49 Module ngx\_http\_status\_module

2.49.1 Summary	318
2.49.2 Example Configuration	318
2.49.3 Directives	319
status	319
status_format	319
status_zone	320
2.49.4 Data	320
2.49.5 Compatibility	327

### 2.49.1 Summary

The ngx\_http\_status\_module module provides access to various status information.

This module was available as part of our [commercial subscription](#) until 1.13.10. It was superseded by the [ngx\\_http\\_api\\_module](#) module in 1.13.3.

### 2.49.2 Example Configuration

```
http {
    upstream backend {
        zone http_backend 64k;

        server backend1.example.com weight=5;
        server backend2.example.com;
    }

    proxy_cache_path /data/nginx/cache_backend keys_zone=cache_backend:10m;

    server {
        server_name backend.example.com;

        location / {
            proxy_pass http://backend;
            proxy_cache cache_backend;

            health_check;
        }

        status_zone server_backend;
    }

    server {
        listen 127.0.0.1;

        location /upstream_conf {
            upstream_conf;
        }

        location /status {
            status;
        }

        location = /status.html {
        }
    }
}
```

```
    }  
}  
  
stream {  
    upstream backend {  
        zone stream_backend 64k;  
  
        server backend1.example.com:12345 weight=5;  
        server backend2.example.com:12345;  
    }  
  
    server {  
        listen      127.0.0.1:12345;  
        proxy_pass  backend;  
        status_zone server_backend;  
        health_check;  
    }  
}
```

Examples of status requests with this configuration:

```
http://127.0.0.1/status  
http://127.0.0.1/status/nginx_version  
http://127.0.0.1/status/caches/cache_backend  
http://127.0.0.1/status/upstreams  
http://127.0.0.1/status/upstreams/backend  
http://127.0.0.1/status/upstreams/backend/peers/1  
http://127.0.0.1/status/upstreams/backend/peers/1/weight  
http://127.0.0.1/status/stream  
http://127.0.0.1/status/stream/upstreams  
http://127.0.0.1/status/stream/upstreams/backend  
http://127.0.0.1/status/stream/upstreams/backend/peers/1  
http://127.0.0.1/status/stream/upstreams/backend/peers/1/weight
```

The simple monitoring page is shipped with this distribution, accessible as “/status.html” in the default configuration. It requires the locations “/status” and “/status.html” to be configured as shown above.

### 2.49.3 Directives

#### status

SYNTAX: **status**;

DEFAULT —

CONTEXT: location

The status information will be accessible from the surrounding location. Access to this location should be [limited](#).

#### status\_format

SYNTAX: **status\_format** json;

SYNTAX: **status\_format** jsonp [*callback*];

DEFAULT json

CONTEXT: http, server, location

By default, status information is output in the JSON format.

Alternatively, data may be output as JSONP. The *callback* parameter specifies the name of a callback function. Parameter value can contain



variables. If parameter is omitted, or the computed value is an empty string, then “ngx\_status\_jsonp\_callback” is used.

### status\_zone

SYNTAX: **status\_zone** *zone*;

DEFAULT —

CONTEXT: server

Enables collection of virtual [http](#) or [stream](#) (1.7.11) server status information in the specified *zone*. Several servers may share the same zone.

## 2.49.4 Data

The following status information is provided:

version

Version of the provided data set. The current version is 8.

nginx\_version

Version of nginx.

nginx\_build

Name of nginx build.

address

The address of the server that accepted status request.

generation

The total number of configuration [reloads](#).

load\_timestamp

Time of the last reload of configuration, in milliseconds since Epoch.

timestamp

Current time in milliseconds since Epoch.

pid

The ID of the worker process that handled status request.

ppid

The ID of the master process that started the [worker process](#).

processes

respawned

The total number of abnormally terminated and respawned child processes.

connections

accepted

The total number of accepted client connections.

dropped

The total number of dropped client connections.

active

The current number of active client connections.

`idle`  
The current number of idle client connections.

`ssl`

`handshakes`  
The total number of successful SSL handshakes.

`handshakes_failed`  
The total number of failed SSL handshakes.

`session_reuses`  
The total number of session reuses during SSL handshake.

`requests`

`total`  
The total number of client requests.

`current`  
The current number of client requests.

`server_zones`  
For each [status\\_zone](#):

`processing`  
The number of client requests that are currently being processed.

`requests`  
The total number of client requests received from clients.

`responses`

`total`  
The total number of responses sent to clients.

`1xx, 2xx, 3xx, 4xx, 5xx`  
The number of responses with status codes 1xx, 2xx, 3xx, 4xx, and 5xx.

`discarded`  
The total number of requests completed without sending a response.

`received`  
The total number of bytes received from clients.

`sent`  
The total number of bytes sent to clients.

`slabs`  
For each shared memory zone that uses slab allocator:

`pages`

`used`  
The current number of used memory pages.

`free`  
The current number of free memory pages.

`slots`  
For each memory slot size (8, 16, 32, 64, 128, etc.) the following data are provided:

used

The current number of used memory slots.

free

The current number of free memory slots.

reqs

The total number of attempts to allocate memory of specified size.

fails

The number of unsuccessful attempts to allocate memory of specified size.

upstreams

For each [dynamically configurable group](#), the following data are provided:

peers

For each [server](#), the following data are provided:

id

The ID of the server.

server

An [address](#) of the server.

name

The name of the server specified in the [server](#) directive.

service

The [service](#) parameter value of the [server](#) directive.

backup

A boolean value indicating whether the server is a [backup](#) server.

weight

[Weight](#) of the server.

state

Current state, which may be one of “up”, “draining”, “down”, “unavail”, “checking”, or “unhealthy”.

active

The current number of active connections.

max\_conns

The [max\\_conns](#) limit for the server.

requests

The total number of client requests forwarded to this server.

responses

total

The total number of responses obtained from this server.

1xx, 2xx, 3xx, 4xx, 5xx

The number of responses with status codes 1xx, 2xx, 3xx, 4xx, and 5xx.

sent

The total number of bytes sent to this server.

received

The total number of bytes received from this server.

fails

The total number of unsuccessful attempts to communicate with the server.

unavail

How many times the server became unavailable for client requests (state “unavail”) due to the number of unsuccessful attempts reaching the [max\\_fails](#) threshold.

health\_checks

checks

The total number of [health check](#) requests made.

fails

The number of failed health checks.

unhealthy

How many times the server became unhealthy (state “unhealthy”).

last\_passed

Boolean indicating if the last health check request was successful and passed [tests](#).

downtime

Total time the server was in the “unavail”, “checking”, and “unhealthy” states.

downstart

The time (in milliseconds since Epoch) when the server became “unavail”, “checking”, or “unhealthy”.

selected

The time (in milliseconds since Epoch) when the server was last selected to process a request (1.7.5).

header\_time

The average time to get the [response header](#) from the server (1.7.10). Prior to version 1.11.6, the field was available only when using the [least\\_time](#) load balancing method.

response\_time

The average time to get the [full response](#) from the server (1.7.10). Prior to version 1.11.6, the field was available only when using the [least\\_time](#) load balancing method.

keepalive

The current number of idle [keepalive](#) connections.

zombies

The current number of servers removed from the group but still processing active client requests.

zone

The name of the shared memory [zone](#) that keeps the group’s configuration and run-time state.

### queue

For the requests [queue](#), the following data are provided:

#### size

The current number of requests in the queue.

#### max\_size

The maximum number of requests that can be in the queue at the same time.

#### overflows

The total number of requests rejected due to the queue overflow.

### caches

For each cache (configured by [proxy\\_cache\\_path](#) and the likes):

#### size

The current size of the cache.

#### max\_size

The limit on the maximum size of the cache specified in the configuration.

#### cold

A boolean value indicating whether the “cache loader” process is still loading data from disk into the cache.

#### hit, stale, updating, revalidated

##### responses

The total number of responses read from the cache (hits, or stale responses due to [proxy\\_cache\\_use\\_stale](#) and the likes).

##### bytes

The total number of bytes read from the cache.

#### miss, expired, bypass

##### responses

The total number of responses not taken from the cache (misses, expires, or bypasses due to [proxy\\_cache\\_bypass](#) and the likes).

##### bytes

The total number of bytes read from the proxied server.

##### responses\_written

The total number of responses written to the cache.

##### bytes\_written

The total number of bytes written to the cache.

### stream

#### server\_zones

For each [status\\_zone](#):

##### processing

The number of client connections that are currently being processed.

##### connections

The total number of connections accepted from clients.

### sessions

#### total

The total number of completed client sessions.

#### 2xx, 4xx, 5xx

The number of sessions completed with [status codes](#) 2xx, 4xx, or 5xx.

### discarded

The total number of connections completed without creating a session.

### received

The total number of bytes received from clients.

### sent

The total number of bytes sent to clients.

### upstreams

For each [dynamically configurable group](#), the following data are provided:

#### peers

For each [server](#) the following data are provided:

##### id

The ID of the server.

##### server

An [address](#) of the server.

##### name

The name of the server specified in the [server](#) directive.

##### service

The [service](#) parameter value of the [server](#) directive.

##### backup

A boolean value indicating whether the server is a [backup](#) server.

##### weight

[Weight](#) of the server.

##### state

Current state, which may be one of “up”, “down”, “unavail”, “checking”, or “unhealthy”.

##### active

The current number of connections.

##### max\_conns

The [max\\_conns](#) limit for the server.

##### connections

The total number of client connections forwarded to this server.

##### connect\_time

The average time to connect to the upstream server. Prior to version 1.11.6, the field was available only when using the [least\\_time](#) load balancing method.

**first\_byte\_time**

The average time to receive the first byte of data. Prior to version 1.11.6, the field was available only when using the [least\\_time](#) load balancing method.

**response\_time**

The average time to receive the last byte of data. Prior to version 1.11.6, the field was available only when using the [least\\_time](#) load balancing method.

**sent**

The total number of bytes sent to this server.

**received**

The total number of bytes received from this server.

**fails**

The total number of unsuccessful attempts to communicate with the server.

**unavail**

How many times the server became unavailable for client connections (state “unavail”) due to the number of unsuccessful attempts reaching the [max\\_fails](#) threshold.

**health\_checks****checks**

The total number of [health check](#) requests made.

**fails**

The number of failed health checks.

**unhealthy**

How many times the server became unhealthy (state “unhealthy”).

**last\_passed**

Boolean indicating if the last health check request was successful and passed [tests](#).

**downtime**

Total time the server was in the “unavail”, “checking”, and “unhealthy” states.

**downstart**

The time (in milliseconds since Epoch) when the server became “unavail”, “checking”, or “unhealthy”.

**selected**

The time (in milliseconds since Epoch) when the server was last selected to process a connection.

**zombies**

The current number of servers removed from the group but still processing active client connections.

**zone**

The name of the shared memory [zone](#) that keeps the group’s configuration and run-time state.

## 2.49.5 Compatibility

- The `zone` field in `http` and `stream` upstreams was added in [version 8](#).
- The `slabs` status data were added in [version 8](#).
- The `checking` state was added in [version 8](#).
- The `name` and `service` fields in `http` and `stream` upstreams were added in [version 8](#).
- The `nginx_build` and `ppid` fields were added in [version 8](#).
- The `sessions` status data and the `discarded` field in stream `server_zones` were added in [version 7](#).
- The `zombies` field was moved from nginx `debug` version in [version 6](#).
- The `ssl` status data were added in [version 6](#).
- The `discarded` field in `server_zones` was added in [version 6](#).
- The `queue` status data were added in [version 6](#).
- The `pid` field was added in [version 6](#).
- The list of servers in `upstreams` was moved into `peers` in [version 6](#).
- The `keepalive` field of an upstream server was removed in [version 5](#).
- The `stream` status data were added in [version 5](#).
- The `generation` field was added in [version 5](#).
- The `respawned` field in `processes` was added in [version 5](#).
- The `header_time` and `response_time` fields in `upstreams` were added in [version 5](#).
- The `selected` field in `upstreams` was added in [version 4](#).
- The `draining` state in `upstreams` was added in [version 4](#).
- The `id` and `max_conns` fields in `upstreams` were added in [version 3](#).
- The `revalidated` field in `caches` was added in [version 3](#).
- The `server_zones`, `caches`, and `load_timestamp` status data were added in [version 2](#).



## 2.50 Module ngx\_http\_stub\_status\_module

2.50.1 Summary	328
2.50.2 Example Configuration	328
2.50.3 Directives	328
stub_status	328
2.50.4 Data	329
2.50.5 Embedded Variables	329

### 2.50.1 Summary

The ngx\_http\_stub\_status\_module module provides access to basic status information.

This module is not built by default, it should be enabled with the `--with-http_stub_status_module` configuration parameter.

### 2.50.2 Example Configuration

```
location = /basic_status {
    stub_status;
}
```

This configuration creates a simple web page with basic status data which may look like as follows:

```
Active connections: 291
server accepts handled requests
 16630948 16630948 31070465
Reading: 6 Writing: 179 Waiting: 106
```

### 2.50.3 Directives

#### stub\_status

SYNTAX: **stub\_status**;

DEFAULT —

CONTEXT: server, location

The basic status information will be accessible from the surrounding location.

In versions prior to 1.7.5, the directive syntax required an arbitrary argument, for example, “`stub_status on`”.

## 2.50.4 Data

The following status information is provided:

Active connections

The current number of active client connections including Waiting connections.

accepts

The total number of accepted client connections.

handled

The total number of handled connections. Generally, the parameter value is the same as `accepts` unless some resource limits have been reached (for example, the [worker\\_connections](#) limit).

requests

The total number of client requests.

Reading

The current number of connections where nginx is reading the request header.

Writing

The current number of connections where nginx is writing the response back to the client.

Waiting

The current number of idle client connections waiting for a request.

## 2.50.5 Embedded Variables

The `ngx_http_stub_status_module` module supports the following embedded variables (1.3.14):

*\$connections\_active*

same as the `Active connections` value;

*\$connections\_reading*

same as the `Reading` value;

*\$connections\_writing*

same as the `Writing` value;

*\$connections\_waiting*

same as the `Waiting` value.

## 2.51 Module ngx\_http\_sub\_module

2.51.1 Summary	330
2.51.2 Example Configuration	330
2.51.3 Directives	330
sub_filter	330
sub_filter_last_modified	330
sub_filter_once	331
sub_filter_types	331

### 2.51.1 Summary

The ngx\_http\_sub\_module module is a filter that modifies a response by replacing one specified string by another.

This module is not built by default, it should be enabled with the `--with-http_sub_module` configuration parameter.

### 2.51.2 Example Configuration

```
location / {
    sub_filter ' <a href="http://127.0.0.1:8080/' ' <a href="https://$host/';
    sub_filter ' <img src="http://127.0.0.1:8080/' ' <img src="https://$host/';
    sub_filter_once on;
}
```

### 2.51.3 Directives

#### sub\_filter

SYNTAX: **sub\_filter** *string replacement*;

DEFAULT —

CONTEXT: http, server, location

Sets a string to replace and a replacement string. The string to replace is matched ignoring the case. The string to replace (1.9.4) and replacement string can contain variables. Several `sub_filter` directives can be specified on the same configuration level (1.9.4). These directives are inherited from the previous configuration level if and only if there are no `sub_filter` directives defined on the current level.

#### sub\_filter\_last\_modified

SYNTAX: **sub\_filter\_last\_modified** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.1.

Allows preserving the Last-Modified header field from the original response during replacement to facilitate response caching.

By default, the header field is removed as contents of the response are modified during processing.

### **sub\_filter\_once**

SYNTAX: **sub\_filter\_once** on | off;  
DEFAULT on  
CONTEXT: http, server, location

Indicates whether to look for each string to replace once or repeatedly.

### **sub\_filter\_types**

SYNTAX: **sub\_filter\_types** *mime-type* ...;  
DEFAULT text/html  
CONTEXT: http, server, location

Enables string replacement in responses with the specified MIME types in addition to “text/html”. The special value “\*” matches any MIME type (0.8.29).

## 2.52 Module ngx\_http\_upstream\_module

2.52.1	Summary	332
2.52.2	Example Configuration	332
2.52.3	Directives	333
	upstream	333
	server	333
	zone	336
	state	336
	hash	337
	ip_hash	337
	keepalive	338
	keepalive_requests	339
	keepalive_time	340
	keepalive_timeout	340
	ntlm	340
	least_conn	341
	least_time	341
	queue	342
	random	342
	resolver	342
	resolver_timeout	343
	sticky	343
	sticky_cookie_insert	346
2.52.4	Embedded Variables	346

### 2.52.1 Summary

The `ngx_http_upstream_module` module is used to define groups of servers that can be referenced by the [proxy\\_pass](#), [fastcgi\\_pass](#), [uwsgi\\_pass](#), [scgi\\_pass](#), [memcached\\_pass](#), and [grpc\\_pass](#) directives.

### 2.52.2 Example Configuration

```
upstream backend {
    server backend1.example.com      weight=5;
    server backend2.example.com:8080;
    server unix:/tmp/backend3;

    server backup1.example.com:8080  backup;
    server backup2.example.com:8080  backup;
}

server {
    location / {
        proxy_pass http://backend;
    }
}
```

Dynamically configurable group with periodic [health checks](#) is available as part of our [commercial subscription](#):

```
resolver 10.0.0.1;

upstream dynamic {
    zone upstream_dynamic 64k;

    server backend1.example.com      weight=5;
    server backend2.example.com:8080 fail_timeout=5s slow_start=30s;
    server 192.0.2.1                  max_fails=3;
    server backend3.example.com      resolve;
    server backend4.example.com      service=http resolve;

    server backup1.example.com:8080  backup;
    server backup2.example.com:8080  backup;
}

server {
    location / {
        proxy_pass http://dynamic;
        health_check;
    }
}
```

## 2.52.3 Directives

### upstream

SYNTAX: **upstream** *name* { ... }

DEFAULT —

CONTEXT: http

Defines a group of servers. Servers can listen on different ports. In addition, servers listening on TCP and UNIX-domain sockets can be mixed.

Example:

```
upstream backend {
    server backend1.example.com weight=5;
    server 127.0.0.1:8080        max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend3;

    server backup1.example.com  backup;
}
```

By default, requests are distributed between the servers using a weighted round-robin balancing method. In the above example, each 7 requests will be distributed as follows: 5 requests go to `backend1.example.com` and one request to each of the second and third servers. If an error occurs during communication with a server, the request will be passed to the next server, and so on until all of the functioning servers will be tried. If a successful response could not be obtained from any of the servers, the client will receive the result of the communication with the last server.

### server

SYNTAX: **server** *address* [*parameters*];

DEFAULT —

CONTEXT: upstream

Defines the *address* and other *parameters* of a server. The address can be specified as a domain name or IP address, with an optional port, or as a UNIX-domain socket path specified after the “unix:” prefix. If a port is not specified, the port 80 is used. A domain name that resolves to several IP addresses defines multiple servers at once.

The following parameters can be defined:

`weight=number`

sets the weight of the server, by default, 1.

`max_conns=number`

limits the maximum *number* of simultaneous active connections to the proxied server (1.11.5). Default value is zero, meaning there is no limit. If the server group does not reside in the [shared memory](#), the limitation works per each worker process.

If [idle keepalive](#) connections, multiple [workers](#), and the [shared memory](#) are enabled, the total number of active and idle connections to the proxied server may exceed the `max_conns` value.

Since version 1.5.9 and prior to version 1.11.5, this parameter was available as part of our [commercial subscription](#).

`max_fails=number`

sets the number of unsuccessful attempts to communicate with the server that should happen in the duration set by the `fail_timeout` parameter to consider the server unavailable for a duration also set by the `fail_timeout` parameter. By default, the number of unsuccessful attempts is set to 1. The zero value disables the accounting of attempts. What is considered an unsuccessful attempt is defined by the [proxy\\_next\\_upstream](#), [fastcgi\\_next\\_upstream](#), [uwsgi\\_next\\_upstream](#), [scgi\\_next\\_upstream](#), [memcached\\_next\\_upstream](#), and [grpc\\_next\\_upstream](#) directives.

`fail_timeout=time`

sets

- the time during which the specified number of unsuccessful attempts to communicate with the server should happen to consider the server unavailable;
- and the period of time the server will be considered unavailable.

By default, the parameter is set to 10 seconds.

`backup`

marks the server as a backup server. It will be passed requests when the primary servers are unavailable.

The parameter cannot be used along with the [hash](#), [ip\\_hash](#), and [random](#) load balancing methods.

`down`

marks the server as permanently unavailable.

Additionally, the following parameters are available as part of our [commercial subscription](#):

`resolve`

monitors changes of the IP addresses that correspond to a domain name of the server, and automatically modifies the upstream configuration without the need of restarting nginx (1.5.12). The server group must reside in the [shared memory](#).

In order for this parameter to work, the `resolver` directive must be specified in the [http](#) block or in the corresponding [upstream](#) block.

`route=string`

sets the server route name.

`service=name`

enables resolving of DNS [SRV](#) records and sets the service *name* (1.9.13). In order for this parameter to work, it is necessary to specify the [resolve](#) parameter for the server and specify a hostname without a port number. If the service name does not contain a dot (“.”), then the [RFC](#)-compliant name is constructed and the TCP protocol is added to the service prefix. For example, to look up the `_http._tcp.backend.example.com` SRV record, it is necessary to specify the directive:

```
server backend.example.com service=http resolve;
```

If the service name contains one or more dots, then the name is constructed by joining the service prefix and the server name. For example, to look up the `_http._tcp.backend.example.com` and `server1.backend.example.com` SRV records, it is necessary to specify the directives:

```
server backend.example.com service=_http._tcp resolve;  
server example.com service=server1.backend resolve;
```

Highest-priority SRV records (records with the same lowest-number priority value) are resolved as primary servers, the rest of SRV records are resolved as backup servers. If the [backup](#) parameter is specified for the server, high-priority SRV records are resolved as backup servers, the rest of SRV records are ignored.

`slow_start=time`

sets the *time* during which the server will recover its weight from zero to a nominal value, when unhealthy server becomes [healthy](#), or when the server becomes available after a period of time it was considered [unavailable](#). Default value is zero, i.e. slow start is disabled.



The parameter cannot be used along with the [hash](#), [ip\\_hash](#), and [random](#) load balancing methods.

#### **drain**

puts the server into the “draining” mode (1.13.6). In this mode, only requests [bound](#) to the server will be proxied to it.

Prior to version 1.13.6, the parameter could be changed only with the [API](#) module.

If there is only a single server in a group, `max_fails`, `fail_timeout` and `slow_start` parameters are ignored, and such a server will never be considered unavailable.

#### **zone**

SYNTAX: **zone** *name* [*size*];

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.9.0.

Defines the *name* and *size* of the shared memory zone that keeps the group’s configuration and run-time state that are shared between worker processes. Several groups may share the same zone. In this case, it is enough to specify the *size* only once.

Additionally, as part of our [commercial subscription](#), such groups allow changing the group membership or modifying the settings of a particular server without the need of restarting nginx. The configuration is accessible via the [API](#) module (1.13.3).

Prior to version 1.13.3, the configuration was accessible only via a special location handled by [upstream\\_conf](#).

#### **state**

SYNTAX: **state** *file*;

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.9.7.

Specifies a *file* that keeps the state of the dynamically configurable group. Examples:

```
state /var/lib/nginx/state/servers.conf; # path for Linux
state /var/db/nginx/state/servers.conf;  # path for FreeBSD
```

The state is currently limited to the list of servers with their parameters. The file is read when parsing the configuration and is updated each time the

upstream configuration is [changed](#). Changing the file content directly should be avoided. The directive cannot be used along with the [server](#) directive.

Changes made during [configuration reload](#) or [binary upgrade](#) can be lost.

This directive is available as part of our [commercial subscription](#).

## hash

SYNTAX: **hash** *key* [consistent];

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.7.2.

Specifies a load balancing method for a server group where the client-server mapping is based on the hashed *key* value. The *key* can contain text, variables, and their combinations. Note that adding or removing a server from the group may result in remapping most of the keys to different servers. The method is compatible with the [Cache::Memcached](#) Perl library.

If the `consistent` parameter is specified, the [ketama](#) consistent hashing method will be used instead. The method ensures that only a few keys will be remapped to different servers when a server is added to or removed from the group. This helps to achieve a higher cache hit ratio for caching servers. The method is compatible with the [Cache::Memcached::Fast](#) Perl library with the `ketama_points` parameter set to 160.

## ip\_hash

SYNTAX: **ip\_hash**;

DEFAULT —

CONTEXT: upstream

Specifies that a group should use a load balancing method where requests are distributed between servers based on client IP addresses. The first three octets of the client IPv4 address, or the entire IPv6 address, are used as a hashing key. The method ensures that requests from the same client will always be passed to the same server except when this server is unavailable. In the latter case client requests will be passed to another server. Most probably, it will always be the same server as well.

IPv6 addresses are supported starting from versions 1.3.2 and 1.2.2.

If one of the servers needs to be temporarily removed, it should be marked with the `down` parameter in order to preserve the current hashing of client IP addresses.

Example:

```
upstream backend {
    ip_hash;

    server backend1.example.com;
    server backend2.example.com;
    server backend3.example.com down;
    server backend4.example.com;
}
```

Until versions 1.3.1 and 1.2.2, it was not possible to specify a weight for servers using the `ip_hash` load balancing method.

## keepalive

SYNTAX: **keepalive** *connections*;

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.1.4.

Activates the cache for connections to upstream servers.

The *connections* parameter sets the maximum number of idle keepalive connections to upstream servers that are preserved in the cache of each worker process. When this number is exceeded, the least recently used connections are closed.

It should be particularly noted that the `keepalive` directive does not limit the total number of connections to upstream servers that an nginx worker process can open. The *connections* parameter should be set to a number small enough to let upstream servers process new incoming connections as well.

When using load balancing methods other than the default round-robin method, it is necessary to activate them before the `keepalive` directive.

Example configuration of memcached upstream with keepalive connections:

```
upstream memcached_backend {
    server 127.0.0.1:11211;
    server 10.0.0.2:11211;

    keepalive 32;
}

server {
    ...

    location /memcached/ {
        set $memcached_key $uri;
        memcached_pass memcached_backend;
    }
}
```

For HTTP, the [proxy\\_http\\_version](#) directive should be set to “1.1” and the Connection header field should be cleared:

```
upstream http_backend {
    server 127.0.0.1:8080;

    keepalive 16;
}

server {
    ...

    location /http/ {
        proxy_pass http://http_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        ...
    }
}
```

Alternatively, HTTP/1.0 persistent connections can be used by passing the Connection: Keep-Alive header field to an upstream server, though this method is not recommended.

For FastCGI servers, it is required to set [fastcgi\\_keep\\_conn](#) for keepalive connections to work:

```
upstream fastcgi_backend {
    server 127.0.0.1:9000;

    keepalive 8;
}

server {
    ...

    location /fastcgi/ {
        fastcgi_pass fastcgi_backend;
        fastcgi_keep_conn on;
        ...
    }
}
```

SCGI and uwsgi protocols do not have a notion of keepalive connections.

## keepalive\_requests

SYNTAX: **keepalive\_requests** *number*;

DEFAULT 1000

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.15.3.

Sets the maximum number of requests that can be served through one keepalive connection. After the maximum number of requests is made, the connection is closed.

Closing connections periodically is necessary to free per-connection memory allocations. Therefore, using too high maximum number of requests could result in excessive memory usage and not recommended.

Prior to version 1.19.10, the default value was 100.

### keepalive\_time

SYNTAX: **keepalive\_time** *time*;  
DEFAULT 1h  
CONTEXT: upstream  
THIS DIRECTIVE APPEARED IN VERSION 1.19.10.

Limits the maximum time during which requests can be processed through one keepalive connection. After this time is reached, the connection is closed following the subsequent request processing.

### keepalive\_timeout

SYNTAX: **keepalive\_timeout** *timeout*;  
DEFAULT 60s  
CONTEXT: upstream  
THIS DIRECTIVE APPEARED IN VERSION 1.15.3.

Sets a timeout during which an idle keepalive connection to an upstream server will stay open.

### ntlm

SYNTAX: **ntlm**;  
DEFAULT —  
CONTEXT: upstream  
THIS DIRECTIVE APPEARED IN VERSION 1.9.2.

Allows proxying requests with [NTLM Authentication](#). The upstream connection is bound to the client connection once the client sends a request with the `Authorization` header field value starting with “Negotiate” or “NTLM”. Further client requests will be proxied through the same upstream connection, keeping the authentication context.

In order for NTLM authentication to work, it is necessary to enable keepalive connections to upstream servers. The [proxy\\_http\\_version](#) directive should be set to “1.1” and the `Connection` header field should be cleared:

```
upstream http_backend {  
    server 127.0.0.1:8080;  
  
    ntlm;  
}  
  
server {  
    ...
```

```
location /http/ {  
    proxy_pass http://http_backend;  
    proxy_http_version 1.1;  
    proxy_set_header Connection "";  
    ...  
}
```

When using load balancer methods other than the default round-robin method, it is necessary to activate them before the `ntlm` directive.

This directive is available as part of our [commercial subscription](#).

### **least\_conn**

SYNTAX: **least\_conn**;

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSIONS 1.3.1 AND 1.2.2.

Specifies that a group should use a load balancing method where a request is passed to the server with the least number of active connections, taking into account weights of servers. If there are several such servers, they are tried in turn using a weighted round-robin balancing method.

### **least\_time**

SYNTAX: **least\_time** header | last\_byte [inflight];

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.7.10.

Specifies that a group should use a load balancing method where a request is passed to the server with the least average response time and least number of active connections, taking into account weights of servers. If there are several such servers, they are tried in turn using a weighted round-robin balancing method.

If the `header` parameter is specified, time to receive the [response header](#) is used. If the `last_byte` parameter is specified, time to receive the [full response](#) is used. If the `inflight` parameter is specified (1.11.6), incomplete requests are also taken into account.

Prior to version 1.11.6, incomplete requests were taken into account by default.

This directive is available as part of our [commercial subscription](#).

## queue

SYNTAX: **queue** *number* [timeout=*time*];

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.5.12.

If an upstream server cannot be selected immediately while processing a request, the request will be placed into the queue. The directive specifies the maximum *number* of requests that can be in the queue at the same time. If the queue is filled up, or the server to pass the request to cannot be selected within the time period specified in the *timeout* parameter, the 502 Bad Gateway error will be returned to the client.

The default value of the *timeout* parameter is 60 seconds.

When using load balancer methods other than the default round-robin method, it is necessary to activate them before the `queue` directive.

This directive is available as part of our [commercial subscription](#).

## random

SYNTAX: **random** [two [*method*]];

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.15.1.

Specifies that a group should use a load balancing method where a request is passed to a randomly selected server, taking into account weights of servers.

The optional *two* parameter instructs nginx to randomly select [two](#) servers and then choose a server using the specified *method*. The default method is `least_conn` which passes a request to a server with the least number of active connections.

The `least_time` method passes a request to a server with the least average response time and least number of active connections. If `least_time=header` is specified, the time to receive the [response header](#) is used. If `least_time=last_byte` is specified, the time to receive the [full response](#) is used.

The `least_time` method is available as a part of our [commercial subscription](#).

## resolver

SYNTAX: **resolver** *address* ... [valid=*time*] [ipv4=on|off] [ipv6=on|off]  
[status\_zone=*zone*];

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.17.5.

Configures name servers used to resolve names of upstream servers into addresses, for example:

```
resolver 127.0.0.1 [::1]:5353;
```

The address can be specified as a domain name or IP address, with an optional port. If port is not specified, the port 53 is used. Name servers are queried in a round-robin fashion.

By default, nginx will look up both IPv4 and IPv6 addresses while resolving. If looking up of IPv4 or IPv6 addresses is not desired, the `ipv4=off` (1.23.1) or the `ipv6=off` parameter can be specified.

By default, nginx caches answers using the TTL value of a response. An optional `valid` parameter allows overriding it:

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

To prevent DNS spoofing, it is recommended configuring DNS servers in a properly secured trusted local network.

The optional `status_zone` parameter enables [collection](#) of DNS server statistics of requests and responses in the specified *zone*.

This directive is available as part of our [commercial subscription](#).

### resolver\_timeout

SYNTAX: **resolver\_timeout** *time*;

DEFAULT 30s

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.17.5.

Sets a timeout for name resolution, for example:

```
resolver_timeout 5s;
```

This directive is available as part of our [commercial subscription](#).

### sticky

SYNTAX: **sticky** cookie *name* [expires=*time*] [domain=*domain*] [httponly] [samesite=strict|lax|none|*\$variable*] [secure] [path=*path*];

SYNTAX: **sticky** route *\$variable* ...;

SYNTAX: **sticky** learn create=*\$variable* lookup=*\$variable* zone=*name:size* [timeout=*time*] [header] [sync];

DEFAULT —

CONTEXT: upstream



THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Enables session affinity, which causes requests from the same client to be passed to the same server in a group of servers. Three methods are available:

#### `cookie`

When the `cookie` method is used, information about the designated server is passed in an HTTP cookie generated by nginx:

```
upstream backend {
    server backend1.example.com;
    server backend2.example.com;

    sticky cookie srv_id expires=1h domain=.example.com path=/;
}
```

A request that comes from a client not yet bound to a particular server is passed to the server selected by the configured balancing method. Further requests with this cookie will be passed to the designated server. If the designated server cannot process a request, the new server is selected as if the client has not been bound yet.

As a load balancing method always tries to evenly distribute the load considering already bound requests, the server with a higher number of active bound requests has less possibility of getting new unbound requests.

The first parameter sets the name of the cookie to be set or inspected. The cookie value is a hexadecimal representation of the MD5 hash of the IP address and port, or of the UNIX-domain socket path. However, if the “route” parameter of the `server` directive is specified, the cookie value will be the value of the “route” parameter:

```
upstream backend {
    server backend1.example.com route=a;
    server backend2.example.com route=b;

    sticky cookie srv_id expires=1h domain=.example.com path=/;
}
```

In this case, the value of the “`srv_id`” cookie will be either *a* or *b*. Additional parameters may be as follows:

#### `expires=`*time*

Sets the *time* for which a browser should keep the cookie. The special value `max` will cause the cookie to expire on “31 Dec 2037 23:55:55 GMT”. If the parameter is not specified, it will cause the cookie to expire at the end of a browser session.

#### `domain=`*domain*

Defines the *domain* for which the cookie is set. Parameter value can contain variables (1.11.5).

`httponly`

Adds the `HttpOnly` attribute to the cookie (1.7.11).

`samesite=strict | lax | none | $variable`

Adds the `SameSite` (1.19.4) attribute to the cookie with one of the following values: `Strict`, `Lax`, `None`, or using variables (1.23.3).

In the latter case, if the variable value is empty, the `SameSite` attribute will not be added to the cookie, if the value is resolved to `Strict`, `Lax`, or `None`, the the corresponding value will be assigned, otherwise the `Strict` value will be assigned.

`secure`

Adds the `Secure` attribute to the cookie (1.7.11).

`path=path`

Defines the *path* for which the cookie is set.

If any parameters are omitted, the corresponding cookie fields are not set.

`route`

When the `route` method is used, proxied server assigns client a route on receipt of the first request. All subsequent requests from this client will carry routing information in a cookie or URI. This information is compared with the “`route`” parameter of the [server](#) directive to identify the server to which the request should be proxied. If the “`route`” parameter is not specified, the route name will be a hexadecimal representation of the MD5 hash of the IP address and port, or of the UNIX-domain socket path. If the designated server cannot process a request, the new server is selected by the configured balancing method as if there is no routing information in the request.

The parameters of the `route` method specify variables that may contain routing information. The first non-empty variable is used to find the matching server.

Example:

```
map $cookie_jsessionid $route_cookie {
    ~.+\. (?P<route>\w+)$ $route;
}

map $request_uri $route_uri {
    ~jsessionid=.+\. (?P<route>\w+)$ $route;
}

upstream backend {
    server backend1.example.com route=a;
    server backend2.example.com route=b;

    sticky route $route_cookie $route_uri;
}
```

Here, the route is taken from the “`JSESSIONID`” cookie if present in a request. Otherwise, the route from the URI is used.

`learn`

When the `learn` method (1.7.1) is used, nginx analyzes upstream server responses and learns server-initiated sessions usually passed in an HTTP

cookie.

```
upstream backend {
    server backend1.example.com:8080;
    server backend2.example.com:8081;

    sticky learn
        create=$upstream_cookie_examplecookie
        lookup=$cookie_examplecookie
        zone=client_sessions:1m;
}
```

In the example, the upstream server creates a session by setting the cookie “EXAMPLECOOKIE” in the response. Further requests with this cookie will be passed to the same server. If the server cannot process the request, the new server is selected as if the client has not been bound yet.

The parameters `create` and `lookup` specify variables that indicate how new sessions are created and existing sessions are searched, respectively. Both parameters may be specified more than once, in which case the first non-empty variable is used.

Sessions are stored in a shared memory zone, whose *name* and *size* are configured by the `zone` parameter. One megabyte zone can store about 4000 sessions on the 64-bit platform. The sessions that are not accessed during the time specified by the `timeout` parameter get removed from the zone. By default, `timeout` is set to 10 minutes.

The `header` parameter (1.13.1) allows creating a session right after receiving response headers from the upstream server.

The `sync` parameter (1.13.8) enables [synchronization](#) of the shared memory zone.

This directive is available as part of our [commercial subscription](#).

### sticky\_cookie\_insert

SYNTAX: **sticky\_cookie\_insert** *name* [*expires=time*] [*domain=domain*]  
[*path=path*];

DEFAULT —

CONTEXT: upstream

This directive is obsolete since version 1.5.7. An equivalent [sticky](#) directive with a new syntax should be used instead:

```
sticky cookie    name    [expires=time]    [domain=domain]  
[path=path];
```

## 2.52.4 Embedded Variables

The `ngx_http_upstream_module` module supports the following embedded variables:

*\$upstream\_addr*

keeps the IP address and port, or the path to the UNIX-domain socket of the upstream server. If several servers were contacted during request processing, their addresses are separated by commas, e.g. “192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock”. If an internal redirect from one server group to another happens, initiated by X-Accel-Redirect or [error\\_page](#), then the server addresses from different groups are separated by colons, e.g. “192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock : 192.168.10.1:80, 192.168.10.2:80”. If a server cannot be selected, the variable keeps the name of the server group.

*\$upstream\_bytes\_received*

number of bytes received from an upstream server (1.11.4). Values from several connections are separated by commas and colons like addresses in the [\\$upstream\\_addr](#) variable.

*\$upstream\_bytes\_sent*

number of bytes sent to an upstream server (1.15.8). Values from several connections are separated by commas and colons like addresses in the [\\$upstream\\_addr](#) variable.

*\$upstream\_cache\_status*

keeps the status of accessing a response cache (0.8.3). The status can be either “MISS”, “BYPASS”, “EXPIRED”, “STALE”, “UPDATING”, “REVALIDATED”, or “HIT”.

*\$upstream\_connect\_time*

keeps time spent on establishing a connection with the upstream server (1.9.1); the time is kept in seconds with millisecond resolution. In case of SSL, includes time spent on handshake. Times of several connections are separated by commas and colons like addresses in the [\\$upstream\\_addr](#) variable.

*\$upstream\_cookie\_name*

cookie with the specified *name* sent by the upstream server in the Set-Cookie response header field (1.7.1). Only the cookies from the response of the last server are saved.

*\$upstream\_header\_time*

keeps time spent on receiving the response header from the upstream server (1.7.10); the time is kept in seconds with millisecond resolution. Times of several responses are separated by commas and colons like addresses in the [\\$upstream\\_addr](#) variable.

*\$upstream\_http\_name*

keep server response header fields. For example, the Server response header field is available through the *\$upstream\_http\_server* variable. The rules of converting header field names to variable names are the same as for the variables that start with the “[\\$http\\_](#)” prefix. Only the header fields from the response of the last server are saved.

*\$upstream\_queue\_time*

keeps time the request spent in the upstream [queue](#) (1.13.9); the time is

kept in seconds with millisecond resolution. Times of several responses are separated by commas and colons like addresses in the [\\$upstream\\_addr](#) variable.

*\$upstream\_response\_length*

keeps the length of the response obtained from the upstream server (0.7.27); the length is kept in bytes. Lengths of several responses are separated by commas and colons like addresses in the [\\$upstream\\_addr](#) variable.

*\$upstream\_response\_time*

keeps time spent on receiving the response from the upstream server; the time is kept in seconds with millisecond resolution. Times of several responses are separated by commas and colons like addresses in the [\\$upstream\\_addr](#) variable.

*\$upstream\_status*

keeps status code of the response obtained from the upstream server. Status codes of several responses are separated by commas and colons like addresses in the [\\$upstream\\_addr](#) variable. If a server cannot be selected, the variable keeps the 502 Bad Gateway status code.

*\$upstream\_trailer\_name*

keeps fields from the end of the response obtained from the upstream server (1.13.10).

## 2.53 Module ngx\_http\_upstream\_conf\_module

2.53.1 Summary	349
2.53.2 Example Configuration	349
2.53.3 Directives	349
upstream_conf	349

### 2.53.1 Summary

The `ngx_http_upstream_conf_module` module allows configuring upstream server groups on-the-fly via a simple HTTP interface without the need of restarting nginx. The [http](#) or [stream](#) server group must reside in the shared memory.

This module was available as part of our [commercial subscription](#) until 1.13.10. It was superseded by the [ngx\\_http\\_api\\_module](#) module in 1.13.3.

### 2.53.2 Example Configuration

```
upstream backend {
    zone upstream_backend 64k;

    ...
}

server {
    location /upstream_conf {
        upstream_conf;
        allow 127.0.0.1;
        deny all;
    }
}
```

### 2.53.3 Directives

#### upstream\_conf

SYNTAX: **upstream\_conf**;

DEFAULT —

CONTEXT: location

Turns on the HTTP interface of upstream configuration in the surrounding location. Access to this location should be [limited](#).

Configuration commands can be used to:

- view the group configuration;
- view, modify, or remove a server;
- add a new server.

Since addresses in a group are not required to be unique, specific servers in a group are referenced by their IDs. IDs are assigned automatically and shown when adding a new server or viewing the group configuration.

A configuration command consists of parameters passed as request arguments, for example:

```
http://127.0.0.1/upstream_conf?upstream=backend
```

The following parameters are supported:

`stream=`

Selects a [stream](#) upstream server group. Without this parameter, selects an [http](#) upstream server group.

`upstream=name`

Selects a group to work with. This parameter is mandatory.

`id=number`

Selects a server for viewing, modifying, or removing.

`remove=`

Removes a server from the group.

`add=`

Adds a new server to the group.

`backup=`

Required to add a backup server.

Before version 1.7.2, `backup=` was also required to view, modify, or remove existing backup servers.

`server=address`

Same as the “address” parameter of the [http](#) or [stream](#) upstream server. When adding a server, it is possible to specify it as a domain name. In this case, changes of the IP addresses that correspond to a domain name will be monitored and automatically applied to the upstream configuration without the need of restarting nginx (1.7.2). This requires the “resolver” directive in the [http](#) or [stream](#) block. See also the “resolve” parameter of the [http](#) or [stream](#) upstream server.

`service=name`

Same as the “service” parameter of the [http](#) or [stream](#) upstream server (1.9.13).

`weight=number`

Same as the “weight” parameter of the [http](#) or [stream](#) upstream server.

`max_conns=number`

Same as the “max\_conns” parameter of the [http](#) or [stream](#) upstream server.

`max_fails=number`

Same as the “max\_fails” parameter of the [http](#) or [stream](#) upstream server.

`fail_timeout=`*time*

Same as the “fail\_timeout” parameter of the [http](#) or [stream](#) upstream server.

`slow_start=`*time*

Same as the “slow\_start” parameter of the [http](#) or [stream](#) upstream server.

`down=`

Same as the “down” parameter of the [http](#) or [stream](#) upstream server.

`drain=`

Puts the [http](#) upstream server into the “draining” mode (1.7.5). In this mode, only requests [bound](#) to the server will be proxied to it.

`up=`

The opposite of the “down” parameter of the [http](#) or [stream](#) upstream server.

`route=`*string*

Same as the “route” parameter of the [http](#) upstream server.

The first three parameters select an object. This can be either the whole [http](#) or [stream](#) upstream server group, or a specific server. Without other parameters, the configuration of the selected group or server is shown.

For example, to view the configuration of the whole group, send:

```
http://127.0.0.1/upstream_conf?upstream=backend
```

To view the configuration of a specific server, also specify its ID:

```
http://127.0.0.1/upstream_conf?upstream=backend&id=42
```

To add a new server, specify its address in the “server=” parameter. Without other parameters specified, a server will be added with other parameters set to their default values (see the [http](#) or [stream](#) “server” directive).

For example, to add a new primary server, send:

```
http://127.0.0.1/upstream_conf?add=&upstream=backend&server=127.0.0.1:8080
```

To add a new backup server, send:

```
http://127.0.0.1/upstream_conf?add=&upstream=backend&backup=&server=127.0.0.1:8080
```

To add a new primary server, set its parameters to non-default values and mark it as “down”, send:

```
http://127.0.0.1/upstream_conf?add=&upstream=backend&server=127.0.0.1:8080&weight=2&down=
```

To remove a server, specify its ID:



```
http://127.0.0.1/upstream_conf?remove=&upstream=backend&id=42
```

To mark an existing server as “down”, send:

```
http://127.0.0.1/upstream_conf?upstream=backend&id=42&down=
```

To modify the address of an existing server, send:

```
http://127.0.0.1/upstream_conf?upstream=backend&id=42&server=192.0.2.3:8123
```

To modify other parameters of an existing server, send:

```
http://127.0.0.1/upstream_conf?upstream=backend&id=42&max_fails=3&weight=4
```

The above examples are for an [http](#) upstream server group. Similar examples for a [stream](#) upstream server group require the “stream=” parameter.

## 2.54 Module ngx\_http\_upstream\_hc\_module

2.54.1 Summary	353
2.54.2 Example Configuration	353
2.54.3 Directives	354
health_check	354
match	355

### 2.54.1 Summary

The `ngx_http_upstream_hc_module` module allows enabling periodic health checks of the servers in a [group](#) referenced in the surrounding location. The server group must reside in the [shared memory](#).

If a health check fails, the server will be considered unhealthy. If several health checks are defined for the same group of servers, a single failure of any check will make the corresponding server be considered unhealthy. Client requests are not passed to unhealthy servers and servers in the “checking” state.

Please note that most of the variables will have empty values when used with health checks.

This module is available as part of our [commercial subscription](#).

### 2.54.2 Example Configuration

```
upstream dynamic {
    zone upstream_dynamic 64k;

    server backend1.example.com      weight=5;
    server backend2.example.com:8080 fail_timeout=5s slow_start=30s;
    server 192.0.2.1                 max_fails=3;

    server backup1.example.com:8080  backup;
    server backup2.example.com:8080  backup;
}

server {
    location / {
        proxy_pass http://dynamic;
        health_check;
    }
}
```

With this configuration, nginx will send “/” requests to each server in the backend group every five seconds. If any communication error or timeout occurs, or a proxied server responds with the status code other than 2xx or 3xx, the health check will fail, and the server will be considered unhealthy.

Health checks can be configured to test the status code of a response, presence of certain header fields and their values, and the body contents. Tests are configured separately using the [match](#) directive and referenced in the `match` parameter of the [health\\_check](#) directive:

```
http {
    server {
        ...
        location / {
            proxy_pass http://backend;
            health_check match=welcome;
        }

        match welcome {
            status 200;
            header Content-Type = text/html;
            body ~ "Welcome to nginx!";
        }
    }
}
```

This configuration shows that in order for a health check to pass, the response to a health check request should succeed, have status 200, and contain “Welcome to nginx!” in the body.

### 2.54.3 Directives

#### health\_check

SYNTAX: **health\_check** [*parameters*];

DEFAULT —

CONTEXT: location

Enables periodic health checks of the servers in a [group](#) referenced in the surrounding location.

The following optional parameters are supported:

*interval=time*

sets the interval between two consecutive health checks, by default, 5 seconds.

*jitter=time*

sets the time within which each health check will be randomly delayed, by default, there is no delay.

*fails=number*

sets the number of consecutive failed health checks of a particular server after which this server will be considered unhealthy, by default, 1.

*passes=number*

sets the number of consecutive passed health checks of a particular server after which the server will be considered healthy, by default, 1.

*uri=uri*

defines the URI used in health check requests, by default, “/”.

*mandatory* [*persistent*]

sets the initial “checking” state for a server until the first health check is completed (1.11.7). Client requests are not passed to servers in the “checking” state. If the parameter is not specified, the server will be initially considered healthy.

The `persistent` parameter (1.19.7) sets the initial “up” state for a server after reload if the server was considered healthy before reload.

`match=name`

specifies the match block configuring the tests that a response should pass in order for a health check to pass. By default, the response should have status code 2xx or 3xx.

`port=number`

defines the port used when connecting to a server to perform a health check (1.9.7). By default, equals the `server` port.

`type=grpc [grpc_service=name] [grpc_status=code]`

enables periodic [health checks](#) of a gRPC server or a particular gRPC service specified with the optional `grpc_service` parameter (1.19.5). If the server does not support the gRPC Health Checking Protocol, the optional `grpc_status` parameter can be used to specify non-zero gRPC [status](#) (for example, status code “12” / “UNIMPLEMENTED”) that will be treated as healthy:

```
health_check mandatory type=grpc grpc_status=12;
```

The `type=grpc` parameter must be specified after all other directive parameters, `grpc_service` and `grpc_status` must follow `type=grpc`. The parameter is not compatible with `uri` or `match` parameters.

`keepalive_time=time`

enables [keepalive](#) connections for health checks and specifies the time during which requests can be processed through one keepalive connection (1.21.7). By default keepalive connections are disabled.

## match

SYNTAX: **match** *name* { ... }

DEFAULT —

CONTEXT: http

Defines the named test set used to verify responses to health check requests. The following items can be tested in a response:

```
status 200;
    status is 200
status ! 500;
    status is not 500
status 200 204;
    status is 200 or 204
status ! 301 302;
    status is neither 301 nor 302
status 200-399;
    status is in the range from 200 to 399
```

```

status ! 400-599;
    status is not in the range from 400 to 599
status 301-303 307;
    status is either 301, 302, 303, or 307

header Content-Type = text/html;
    header contains Content-Type with value text/html
header Content-Type != text/html;
    header contains Content-Type with value other than text/html
header Connection ~ close;
    header contains Connection with value matching regular expression
    close
header Connection !~ close;
    header contains Connection with value not matching regular
    expression close
header Host;
    header contains Host
header ! X-Accel-Redirect;
    header lacks X-Accel-Redirect

body ~ "Welcome to nginx!";
    body matches regular expression "Welcome to nginx!"
body !~ "Welcome to nginx!";
    body does not match regular expression "Welcome to nginx!"

require $variable ...;
    all specified variables are not empty and not equal to "0" (1.15.9).

```

If several tests are specified, the response matches only if it matches all tests.

Only the first 256k of the response body are examined.

Examples:

```

# status is 200, content type is "text/html",
# and body contains "Welcome to nginx!"
match welcome {
    status 200;
    header Content-Type = text/html;
    body ~ "Welcome to nginx!";
}

```

```

# status is not one of 301, 302, 303, or 307, and header does not have "Refresh
#"
match not_redirect {
    status ! 301-303 307;
    header ! Refresh;
}

```

```
# status ok and not in maintenance mode
match server_ok {
    status 200-399;
    body !~ "maintenance mode";
}
```

```
# status is 200 or 204
map $upstream_status $good_status {
    200 1;
    204 1;
}

match server_ok {
    require $good_status;
}
```

## 2.55 Module ngx\_http\_userid\_module

2.55.1 Summary	358
2.55.2 Example Configuration	358
2.55.3 Directives	358
userid	358
userid_domain	359
userid_expires	359
userid_flags	359
userid_mark	359
userid_name	360
userid_p3p	360
userid_path	360
userid_service	360
2.55.4 Embedded Variables	360

### 2.55.1 Summary

The `ngx_http_userid_module` module sets cookies suitable for client identification. Received and set cookies can be logged using the embedded variables `$uid_got` and `$uid_set`. This module is compatible with the `mod_uid` module for Apache.

### 2.55.2 Example Configuration

```
userid          on;
userid_name     uid;
userid_domain   example.com;
userid_path     /;
userid_expires  365d;
userid_p3p      'policyref="/w3c/p3p.xml", CP="CUR ADM OUR NOR STA NID"';
```

### 2.55.3 Directives

#### userid

SYNTAX: **userid** on | v1 | log | off;

DEFAULT off

CONTEXT: http, server, location

Enables or disables setting cookies and logging the received cookies:

on

enables the setting of version 2 cookies and logging of the received cookies;

v1

enables the setting of version 1 cookies and logging of the received cookies;

**log**  
disables the setting of cookies, but enables logging of the received cookies;  
**off**  
disables the setting of cookies and logging of the received cookies.

### userid\_domain

SYNTAX: **userid\_domain** *name* | none;  
DEFAULT none  
CONTEXT: http, server, location

Defines a domain for which the cookie is set. The none parameter disables setting of a domain for the cookie.

### userid\_expires

SYNTAX: **userid\_expires** *time* | max | off;  
DEFAULT off  
CONTEXT: http, server, location

Sets a time during which a browser should keep the cookie. The parameter max will cause the cookie to expire on “31 Dec 2037 23:55:55 GMT”. The parameter off will cause the cookie to expire at the end of a browser session.

### userid\_flags

SYNTAX: **userid\_flags** off | *flag* ...;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.19.3.

If the parameter is not off, defines one or more additional flags for the cookie: secure, httponly, samesite=strict, samesite=lax, samesite=none.

### userid\_mark

SYNTAX: **userid\_mark** *letter* | *digit* | = | off;  
DEFAULT off  
CONTEXT: http, server, location

If the parameter is not off, enables the cookie marking mechanism and sets the character used as a mark. This mechanism is used to add or change [userid\\_p3p](#) and/or a cookie expiration time while preserving the client identifier. A mark can be any letter of the English alphabet (case-sensitive), digit, or the “=” character.

If the mark is set, it is compared with the first padding symbol in the base64 representation of the client identifier passed in a cookie. If they do not



match, the cookie is resent with the specified mark, expiration time, and P3P header.

### **userid\_name**

SYNTAX: **userid\_name** *name*;

DEFAULT `uid`

CONTEXT: http, server, location

Sets the cookie name.

### **userid\_p3p**

SYNTAX: **userid\_p3p** *string* | none;

DEFAULT `none`

CONTEXT: http, server, location

Sets a value for the P3P header field that will be sent along with the cookie. If the directive is set to the special value none, the P3P header will not be sent in a response.

### **userid\_path**

SYNTAX: **userid\_path** *path*;

DEFAULT `/`

CONTEXT: http, server, location

Defines a path for which the cookie is set.

### **userid\_service**

SYNTAX: **userid\_service** *number*;

DEFAULT IP address of the server

CONTEXT: http, server, location

If identifiers are issued by multiple servers (services), each service should be assigned its own *number* to ensure that client identifiers are unique. For version 1 cookies, the default value is zero. For version 2 cookies, the default value is the number composed from the last four octets of the server's IP address.

## **2.55.4 Embedded Variables**

The `ngx_http_userid_module` module supports the following embedded variables:

*\$uid\_got*

The cookie name and received client identifier.

*\$uid\_reset*

If the variable is set to a non-empty string that is not "0", the client identifiers are reset. The special value "log" additionally leads to the output of messages about the reset identifiers to the [error\\_log](#).

*\$uid\_set*

The cookie name and sent client identifier.

## 2.56 Module ngx\_http\_uwsgi\_module

2.56.1	Summary	363
2.56.2	Example Configuration	363
2.56.3	Directives	363
	uwsgi_bind	363
	uwsgi_buffer_size	364
	uwsgi_buffering	364
	uwsgi_buffers	365
	uwsgi_busy_buffers_size	365
	uwsgi_cache	365
	uwsgi_cache_background_update	365
	uwsgi_cache_bypass	365
	uwsgi_cache_key	366
	uwsgi_cache_lock	366
	uwsgi_cache_lock_age	366
	uwsgi_cache_lock_timeout	366
	uwsgi_cache_max_range_offset	367
	uwsgi_cache_methods	367
	uwsgi_cache_min_uses	367
	uwsgi_cache_path	367
	uwsgi_cache_purge	369
	uwsgi_cache_revalidate	370
	uwsgi_cache_use_stale	370
	uwsgi_cache_valid	371
	uwsgi_connect_timeout	371
	uwsgi_force_ranges	372
	uwsgi_hide_header	372
	uwsgi_ignore_client_abort	372
	uwsgi_ignore_headers	372
	uwsgi_intercept_errors	373
	uwsgi_limit_rate	373
	uwsgi_max_temp_file_size	373
	uwsgi_modifier1	374
	uwsgi_modifier2	374
	uwsgi_next_upstream	374
	uwsgi_next_upstream_timeout	375
	uwsgi_next_upstream_tries	375
	uwsgi_no_cache	375
	uwsgi_param	376
	uwsgi_pass	376
	uwsgi_pass_header	377
	uwsgi_pass_request_body	377
	uwsgi_pass_request_headers	377
	uwsgi_read_timeout	377
	uwsgi_request_buffering	377

<a href="#">uwsgi_send_timeout</a>	378
<a href="#">uwsgi_socket_keepalive</a>	378
<a href="#">uwsgi_ssl_certificate</a>	378
<a href="#">uwsgi_ssl_certificate_key</a>	378
<a href="#">uwsgi_ssl_ciphers</a>	379
<a href="#">uwsgi_ssl_conf_command</a>	379
<a href="#">uwsgi_ssl_crl</a>	379
<a href="#">uwsgi_ssl_name</a>	379
<a href="#">uwsgi_ssl_password_file</a>	380
<a href="#">uwsgi_ssl_protocols</a>	380
<a href="#">uwsgi_ssl_server_name</a>	380
<a href="#">uwsgi_ssl_session_reuse</a>	380
<a href="#">uwsgi_ssl_trusted_certificate</a>	380
<a href="#">uwsgi_ssl_verify</a>	381
<a href="#">uwsgi_ssl_verify_depth</a>	381
<a href="#">uwsgi_store</a>	381
<a href="#">uwsgi_store_access</a>	382
<a href="#">uwsgi_temp_file_write_size</a>	382
<a href="#">uwsgi_temp_path</a>	382

### 2.56.1 Summary

The `ngx_http_uwsgi_module` module allows passing requests to a uwsgi server.

### 2.56.2 Example Configuration

```
location / {
    include    uwsgi_params;
    uwsgi_pass localhost:9000;
}
```

### 2.56.3 Directives

#### `uwsgi_bind`

SYNTAX: **uwsgi\_bind** *address* [transparent] | off;

DEFAULT —

CONTEXT: http, server, location

Makes outgoing connections to a uwsgi server originate from the specified local IP address with an optional port (1.11.2). Parameter value can contain variables (1.3.12). The special value `off` (1.3.12) cancels the effect of the `uwsgi_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address and port.

The `transparent` parameter (1.11.0) allows outgoing connections to a uwsgi server originate from a non-local IP address, for example, from a real IP address of a client:

```
uwsgi_bind $remote_addr transparent;
```

In order for this parameter to work, it is usually necessary to run nginx worker processes with the [superuser](#) privileges. On Linux it is not required (1.13.8) as if the `transparent` parameter is specified, worker processes inherit the `CAP_NET_RAW` capability from the master process. It is also necessary to configure kernel routing table to intercept network traffic from the uwsgi server.

### uwsgi\_buffer\_size

SYNTAX: **uwsgi\_buffer\_size** *size*;

DEFAULT 4k | 8k

CONTEXT: http, server, location

Sets the *size* of the buffer used for reading the first part of the response received from the uwsgi server. This part usually contains a small response header. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform. It can be made smaller, however.

### uwsgi\_buffering

SYNTAX: **uwsgi\_buffering** on | off;

DEFAULT on

CONTEXT: http, server, location

Enables or disables buffering of responses from the uwsgi server.

When buffering is enabled, nginx receives a response from the uwsgi server as soon as possible, saving it into the buffers set by the [uwsgi\\_buffer\\_size](#) and [uwsgi\\_buffers](#) directives. If the whole response does not fit into memory, a part of it can be saved to a [temporary file](#) on the disk. Writing to temporary files is controlled by the [uwsgi\\_max\\_temp\\_file\\_size](#) and [uwsgi\\_temp\\_file\\_write\\_size](#) directives.

When buffering is disabled, the response is passed to a client synchronously, immediately as it is received. nginx will not try to read the whole response from the uwsgi server. The maximum size of the data that nginx can receive from the server at a time is set by the [uwsgi\\_buffer\\_size](#) directive.

Buffering can also be enabled or disabled by passing “yes” or “no” in the `X-Accel-Buffering` response header field. This capability can be disabled using the [uwsgi\\_ignore\\_headers](#) directive.

## uwsgi\_buffers

SYNTAX: **uwsgi\_buffers** *number size*;  
DEFAULT 8 4k | 8k  
CONTEXT: http, server, location

Sets the *number* and *size* of the buffers used for reading a response from the uwsgi server, for a single connection. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

## uwsgi\_busy\_buffers\_size

SYNTAX: **uwsgi\_busy\_buffers\_size** *size*;  
DEFAULT 8k | 16k  
CONTEXT: http, server, location

When [buffering](#) of responses from the uwsgi server is enabled, limits the total *size* of buffers that can be busy sending a response to the client while the response is not yet fully read. In the meantime, the rest of the buffers can be used for reading the response and, if needed, buffering part of the response to a temporary file. By default, *size* is limited by the size of two buffers set by the [uwsgi\\_buffer\\_size](#) and [uwsgi\\_buffers](#) directives.

## uwsgi\_cache

SYNTAX: **uwsgi\_cache** *zone* | off;  
DEFAULT off  
CONTEXT: http, server, location

Defines a shared memory zone used for caching. The same zone can be used in several places. Parameter value can contain variables (1.7.9). The `off` parameter disables caching inherited from the previous configuration level.

## uwsgi\_cache\_background\_update

SYNTAX: **uwsgi\_cache\_background\_update** on | off;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.11.10.

Allows starting a background subrequest to update an expired cache item, while a stale cached response is returned to the client. Note that it is necessary to allow the usage of a stale cached response when it is being updated.

## uwsgi\_cache\_bypass

SYNTAX: **uwsgi\_cache\_bypass** *string* ...;  
DEFAULT —  
CONTEXT: http, server, location

Defines conditions under which the response will not be taken from a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be taken from the cache:

```
uwsgi_cache_bypass $cookie_nocache $arg_nocache$arg_comment;  
uwsgi_cache_bypass $http_pragma      $http_authorization;
```

Can be used along with the [uwsgi\\_no\\_cache](#) directive.

### uwsgi\_cache\_key

SYNTAX: **uwsgi\_cache\_key** *string*;

DEFAULT —

CONTEXT: http, server, location

Defines a key for caching, for example

```
uwsgi_cache_key localhost:9000$request_uri;
```

### uwsgi\_cache\_lock

SYNTAX: **uwsgi\_cache\_lock** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.12.

When enabled, only one request at a time will be allowed to populate a new cache element identified according to the [uwsgi\\_cache\\_key](#) directive by passing a request to a uwsgi server. Other requests of the same cache element will either wait for a response to appear in the cache or the cache lock for this element to be released, up to the time set by the [uwsgi\\_cache\\_lock\\_timeout](#) directive.

### uwsgi\_cache\_lock\_age

SYNTAX: **uwsgi\_cache\_lock\_age** *time*;

DEFAULT 5s

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.8.

If the last request passed to the uwsgi server for populating a new cache element has not completed for the specified *time*, one more request may be passed to the uwsgi server.

### uwsgi\_cache\_lock\_timeout

SYNTAX: **uwsgi\_cache\_lock\_timeout** *time*;

DEFAULT 5s

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.12.

Sets a timeout for [uwsgi\\_cache\\_lock](#). When the *time* expires, the request will be passed to the uwsgi server, however, the response will not be cached.

Before 1.7.8, the response could be cached.

### **uwsgi\_cache\_max\_range\_offset**

SYNTAX: **uwsgi\_cache\_max\_range\_offset** *number*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.11.6.

Sets an offset in bytes for byte-range requests. If the range is beyond the offset, the range request will be passed to the uwsgi server and the response will not be cached.

### **uwsgi\_cache\_methods**

SYNTAX: **uwsgi\_cache\_methods** GET | HEAD | POST ...;

DEFAULT GET HEAD

CONTEXT: http, server, location

If the client request method is listed in this directive then the response will be cached. “GET” and “HEAD” methods are always added to the list, though it is recommended to specify them explicitly. See also the [uwsgi\\_no\\_cache](#) directive.

### **uwsgi\_cache\_min\_uses**

SYNTAX: **uwsgi\_cache\_min\_uses** *number*;

DEFAULT 1

CONTEXT: http, server, location

Sets the *number* of requests after which the response will be cached.

### **uwsgi\_cache\_path**

SYNTAX: **uwsgi\_cache\_path** *path* [*levels=levels*]  
 [*use\_temp\_path=on|off*] *keys\_zone=name:size* [*inactive=time*]  
 [*max\_size=size*] [*min\_free=size*] [*manager\_files=number*]  
 [*manager\_sleep=time*] [*manager\_threshold=time*]  
 [*loader\_files=number*] [*loader\_sleep=time*]  
 [*loader\_threshold=time*] [*purger=on|off*]  
 [*purger\_files=number*] [*purger\_sleep=time*]  
 [*purger\_threshold=time*];

DEFAULT —

CONTEXT: http

Sets the path and other parameters of a cache. Cache data are stored in files. The file name in a cache is a result of applying the MD5 function to



the [cache key](#). The `levels` parameter defines hierarchy levels of a cache: from 1 to 3, each level accepts values 1 or 2. For example, in the following configuration

```
uwsgi_cache_path /data/nginx/cache levels=1:2 keys_zone=one:10m;
```

file names in a cache will look like this:

```
/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

A cached response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the cache can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both cache and a directory holding temporary files are put on the same file system. A directory for temporary files is set based on the `use_temp_path` parameter (1.7.10). If this parameter is omitted or set to the value `on`, the directory set by the [uwsgi\\_temp\\_path](#) directive for the given location will be used. If the value is set to `off`, temporary files will be put directly in the cache directory.

In addition, all active keys and information about data are stored in a shared memory zone, whose *name* and *size* are configured by the `keys_zone` parameter. One megabyte zone can store about 8 thousand keys.

As part of [commercial subscription](#), the shared memory zone also stores extended cache [information](#), thus, it is required to specify a larger zone size for the same number of keys. For example, one megabyte zone can store about 4 thousand keys.

Cached data that are not accessed during the time specified by the `inactive` parameter get removed from the cache regardless of their freshness. By default, `inactive` is set to 10 minutes.

The special “cache manager” process monitors the maximum cache size set by the `max_size` parameter, and the minimum amount of free space set by the `min_free` (1.19.1) parameter on the file system with cache. When the size is exceeded or there is not enough free space, it removes the least recently used data. The data is removed in iterations configured by `manager_files`, `manager_threshold`, and `manager_sleep` parameters (1.11.5). During one iteration no more than `manager_files` items are deleted (by default, 100). The duration of one iteration is limited by the `manager_threshold` parameter (by default, 200 milliseconds). Between iterations, a pause configured by the `manager_sleep` parameter (by default, 50 milliseconds) is made.

A minute after the start the special “cache loader” process is activated. It loads information about previously cached data stored on file system into a cache zone. The loading is also done in iterations. During one iteration no more than `loader_files` items are loaded (by default, 100). Besides, the

duration of one iteration is limited by the `loader_threshold` parameter (by default, 200 milliseconds). Between iterations, a pause configured by the `loader_sleep` parameter (by default, 50 milliseconds) is made.

Additionally, the following parameters are available as part of our [commercial subscription](#):

`purger=on|off`

Instructs whether cache entries that match a [wildcard key](#) will be removed from the disk by the cache purger (1.7.12). Setting the parameter to `on` (default is `off`) will activate the “cache purger” process that permanently iterates through all cache entries and deletes the entries that match the wildcard key.

`purger_files=number`

Sets the number of items that will be scanned during one iteration (1.7.12). By default, `purger_files` is set to 10.

`purger_threshold=number`

Sets the duration of one iteration (1.7.12). By default, `purger_threshold` is set to 50 milliseconds.

`purger_sleep=number`

Sets a pause between iterations (1.7.12). By default, `purger_sleep` is set to 50 milliseconds.

In versions 1.7.3, 1.7.7, and 1.11.10 cache header format has been changed. Previously cached responses will be considered invalid after upgrading to a newer nginx version.

## uwsgi\_cache\_purge

SYNTAX: **uwsgi\_cache\_purge**string ...;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Defines conditions under which the request will be considered a cache purge request. If at least one value of the string parameters is not empty and is not equal to “0” then the cache entry with a corresponding [cache key](#) is removed. The result of successful operation is indicated by returning the 204 No Content response.

If the [cache key](#) of a purge request ends with an asterisk (“\*”), all cache entries matching the wildcard key will be removed from the cache. However, these entries will remain on the disk until they are deleted for either [inactivity](#), or processed by the [cache purger](#) (1.7.12), or a client attempts to access them.

Example configuration:

```
uwsgi_cache_path /data/nginx/cache keys_zone=cache_zone:10m;

map $request_method $purge_method {
    PURGE    1;
}
```

```
    default 0;
}

server {
    ...
    location / {
        uwsgi_pass          backend;
        uwsgi_cache          cache_zone;
        uwsgi_cache_key      $uri;
        uwsgi_cache_purge    $purge_method;
    }
}
```

This functionality is available as part of our [commercial subscription](#).

### uwsgi\_cache\_revalidate

SYNTAX: **uwsgi\_cache\_revalidate** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Enables revalidation of expired cache items using conditional requests with the If-Modified-Since and If-None-Match header fields.

### uwsgi\_cache\_use\_stale

SYNTAX: **uwsgi\_cache\_use\_stale** error | timeout | invalid\_header |  
updating | http\_500 | http\_503 | http\_403 | http\_404 |  
http\_429 | off ...;

DEFAULT off

CONTEXT: http, server, location

Determines in which cases a stale cached response can be used when an error occurs during communication with the uwsgi server. The directive's parameters match the parameters of the [uwsgi\\_next\\_upstream](#) directive.

The `error` parameter also permits using a stale cached response if a uwsgi server to process a request cannot be selected.

Additionally, the `updating` parameter permits using a stale cached response if it is currently being updated. This allows minimizing the number of accesses to uwsgi servers when updating cached data.

Using a stale cached response can also be enabled directly in the response header for a specified number of seconds after the response became stale (1.11.10). This has lower priority than using the directive parameters.

- The “[stale-while-revalidate](#)” extension of the Cache-Control header field permits using a stale cached response if it is currently being updated.
- The “[stale-if-error](#)” extension of the Cache-Control header field permits using a stale cached response in case of an error.

To minimize the number of accesses to uwsgi servers when populating a new cache element, the [uwsgi\\_cache\\_lock](#) directive can be used.

## uwsgi\_cache\_valid

SYNTAX: **uwsgi\_cache\_valid** [*code ...*] *time*;

DEFAULT —

CONTEXT: http, server, location

Sets caching time for different response codes. For example, the following directives

```
uwsgi_cache_valid 200 302 10m;  
uwsgi_cache_valid 404 1m;
```

set 10 minutes of caching for responses with codes 200 and 302 and 1 minute for responses with code 404.

If only caching *time* is specified

```
uwsgi_cache_valid 5m;
```

then only 200, 301, and 302 responses are cached.

In addition, the any parameter can be specified to cache any responses:

```
uwsgi_cache_valid 200 302 10m;  
uwsgi_cache_valid 301 1h;  
uwsgi_cache_valid any 1m;
```

Parameters of caching can also be set directly in the response header. This has higher priority than setting of caching time using the directive.

- The X-Accel-Expires header field sets caching time of a response in seconds. The zero value disables caching for a response. If the value starts with the @ prefix, it sets an absolute time in seconds since Epoch, up to which the response may be cached.
- If the header does not include the X-Accel-Expires field, parameters of caching may be set in the header fields Expires or Cache-Control.
- If the header includes the Set-Cookie field, such a response will not be cached.
- If the header includes the Vary field with the special value “\*”, such a response will not be cached (1.7.7). If the header includes the Vary field with another value, such a response will be cached taking into account the corresponding request header fields (1.7.7).

Processing of one or more of these response header fields can be disabled using the [uwsgi\\_ignore\\_headers](#) directive.

## uwsgi\_connect\_timeout

SYNTAX: **uwsgi\_connect\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for establishing a connection with a uwsgi server. It should be noted that this timeout cannot usually exceed 75 seconds.

### **uwsgi\_force\_ranges**

SYNTAX: **uwsgi\_force\_ranges** on | off;  
DEFAULT off  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.7.

Enables byte-range support for both cached and uncached responses from the uwsgi server regardless of the `Accept-Ranges` field in these responses.

### **uwsgi\_hide\_header**

SYNTAX: **uwsgi\_hide\_header** *field*;  
DEFAULT —  
CONTEXT: http, server, location

By default, nginx does not pass the header fields `Status` and `X-Accel-...` from the response of a uwsgi server to a client. The `uwsgi_hide_header` directive sets additional fields that will not be passed. If, on the contrary, the passing of fields needs to be permitted, the [uwsgi\\_pass\\_header](#) directive can be used.

### **uwsgi\_ignore\_client\_abort**

SYNTAX: **uwsgi\_ignore\_client\_abort** on | off;  
DEFAULT off  
CONTEXT: http, server, location

Determines whether the connection with a uwsgi server should be closed when a client closes the connection without waiting for a response.

### **uwsgi\_ignore\_headers**

SYNTAX: **uwsgi\_ignore\_headers** *field* ...;  
DEFAULT —  
CONTEXT: http, server, location

Disables processing of certain response header fields from the uwsgi server. The following fields can be ignored: `X-Accel-Redirect`, `X-Accel-Expires`, `X-Accel-Limit-Rate` (1.1.6), `X-Accel-Buffering` (1.1.6), `X-Accel-Charset` (1.1.6), `Expires`, `Cache-Control`, `Set-Cookie` (0.8.44), and `Vary` (1.7.7).

If not disabled, processing of these header fields has the following effect:

- `X-Accel-Expires`, `Expires`, `Cache-Control`, `Set-Cookie`, and `Vary` set the parameters of response [caching](#);
- `X-Accel-Redirect` performs an [internal redirect](#) to the specified URI;

- X-Accel-Limit-Rate sets the [rate limit](#) for transmission of a response to a client;
- X-Accel-Buffering enables or disables [buffering](#) of a response;
- X-Accel-Charset sets the desired [charset](#) of a response.

### **uwsgi\_intercept\_errors**

SYNTAX: **uwsgi\_intercept\_errors** on | off;  
DEFAULT off  
CONTEXT: http, server, location

Determines whether a uwsgi server responses with codes greater than or equal to 300 should be passed to a client or be intercepted and redirected to nginx for processing with the [error\\_page](#) directive.

### **uwsgi\_limit\_rate**

SYNTAX: **uwsgi\_limit\_rate** *rate*;  
DEFAULT 0  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.7.7.

Limits the speed of reading the response from the uwsgi server. The *rate* is specified in bytes per second. The zero value disables rate limiting. The limit is set per a request, and so if nginx simultaneously opens two connections to the uwsgi server, the overall rate will be twice as much as the specified limit. The limitation works only if [buffering](#) of responses from the uwsgi server is enabled.

### **uwsgi\_max\_temp\_file\_size**

SYNTAX: **uwsgi\_max\_temp\_file\_size** *size*;  
DEFAULT 1024m  
CONTEXT: http, server, location

When [buffering](#) of responses from the uwsgi server is enabled, and the whole response does not fit into the buffers set by the [uwsgi\\_buffer\\_size](#) and [uwsgi\\_buffers](#) directives, a part of the response can be saved to a temporary file. This directive sets the maximum *size* of the temporary file. The size of data written to the temporary file at a time is set by the [uwsgi\\_temp\\_file\\_write\\_size](#) directive.

The zero value disables buffering of responses to temporary files.

This restriction does not apply to responses that will be [cached](#) or [stored](#) on disk.

## uwsgi\_modifier1

SYNTAX: **uwsgi\_modifier1** *number*;

DEFAULT 0

CONTEXT: http, server, location

Sets the value of the `modifier1` field in the [uwsgi packet header](#).

## uwsgi\_modifier2

SYNTAX: **uwsgi\_modifier2** *number*;

DEFAULT 0

CONTEXT: http, server, location

Sets the value of the `modifier2` field in the [uwsgi packet header](#).

## uwsgi\_next\_upstream

SYNTAX: **uwsgi\_next\_upstream** *error | timeout | invalid\_header |  
http\_500 | http\_503 | http\_403 | http\_404 | http\_429 |  
non\_idempotent | off ...*;

DEFAULT *error timeout*

CONTEXT: http, server, location

Specifies in which cases a request should be passed to the next server:

*error*

an error occurred while establishing a connection with the server, passing a request to it, or reading the response header;

*timeout*

a timeout has occurred while establishing a connection with the server, passing a request to it, or reading the response header;

*invalid\_header*

a server returned an empty or invalid response;

*http\_500*

a server returned a response with the code 500;

*http\_503*

a server returned a response with the code 503;

*http\_403*

a server returned a response with the code 403;

*http\_404*

a server returned a response with the code 404;

*http\_429*

a server returned a response with the code 429 (1.11.13);

*non\_idempotent*

normally, requests with a [non-idempotent](#) method (POST, LOCK, PATCH) are not passed to the next server if a request has been sent to an upstream server (1.9.13); enabling this option explicitly allows retrying such requests;

`off`

disables passing a request to the next server.

One should bear in mind that passing a request to the next server is only possible if nothing has been sent to a client yet. That is, if an error or timeout occurs in the middle of the transferring of a response, fixing this is impossible.

The directive also defines what is considered an [unsuccessful attempt](#) of communication with a server. The cases of `error`, `timeout` and `invalid_header` are always considered unsuccessful attempts, even if they are not specified in the directive. The cases of `http_500`, `http_503`, and `http_429` are considered unsuccessful attempts only if they are specified in the directive. The cases of `http_403` and `http_404` are never considered unsuccessful attempts.

Passing a request to the next server can be limited by [the number of tries](#) and by [time](#).

### **uwsgi\_next\_upstream\_timeout**

SYNTAX: **uwsgi\_next\_upstream\_timeout** *time*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.5.

Limits the time during which a request can be passed to the [next server](#). The 0 value turns off this limitation.

### **uwsgi\_next\_upstream\_tries**

SYNTAX: **uwsgi\_next\_upstream\_tries** *number*;

DEFAULT 0

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.5.

Limits the number of possible tries for passing a request to the [next server](#). The 0 value turns off this limitation.

### **uwsgi\_no\_cache**

SYNTAX: **uwsgi\_no\_cache** *string* ...;

DEFAULT —

CONTEXT: http, server, location

Defines conditions under which the response will not be saved to a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be saved:

```
uwsgi_no_cache $cookie_nocache $arg_nocache$arg_comment;  
uwsgi_no_cache $http_pragma $http_authorization;
```

Can be used along with the [uwsgi\\_cache\\_bypass](#) directive.



## uwsgi\_param

SYNTAX: **uwsgi\_param** *parameter value* [if\_not\_empty];

DEFAULT —

CONTEXT: http, server, location

Sets a *parameter* that should be passed to the uwsgi server. The *value* can contain text, variables, and their combination. These directives are inherited from the previous configuration level if and only if there are no uwsgi\_param directives defined on the current level.

Standard [CGI environment variables](#) should be provided as uwsgi headers, see the uwsgi\_params file provided in the distribution:

```
location / {
    include uwsgi_params;
    ...
}
```

If the directive is specified with if\_not\_empty (1.1.11) then such a parameter will be passed to the server only if its value is not empty:

```
uwsgi_param HTTPS $https if_not_empty;
```

## uwsgi\_pass

SYNTAX: **uwsgi\_pass** [*protocol://*] *address*;

DEFAULT —

CONTEXT: location, if in location

Sets the protocol and address of a uwsgi server. As a *protocol*, “uwsgi” or “suwsgi” (secured uwsgi, uwsgi over SSL) can be specified. The address can be specified as a domain name or IP address, and a port:

```
uwsgi_pass localhost:9000;
uwsgi_pass uwsgi://localhost:9000;
uwsgi_pass suwsgi://[2001:db8::1]:9090;
```

or as a UNIX-domain socket path:

```
uwsgi_pass unix:/tmp/uwsgi.socket;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a [server group](#).

Parameter value can contain variables. In this case, if an address is specified as a domain name, the name is searched among the described [server groups](#), and, if not found, is determined using a [resolver](#).

Secured uwsgi protocol is supported since version 1.5.8.

## uwsgi\_pass\_header

SYNTAX: **uwsgi\_pass\_header** *field*;

DEFAULT —

CONTEXT: http, server, location

Permits passing [otherwise disabled](#) header fields from a uwsgi server to a client.

## uwsgi\_pass\_request\_body

SYNTAX: **uwsgi\_pass\_request\_body** on | off;

DEFAULT on

CONTEXT: http, server, location

Indicates whether the original request body is passed to the uwsgi server. See also the [uwsgi\\_pass\\_request\\_headers](#) directive.

## uwsgi\_pass\_request\_headers

SYNTAX: **uwsgi\_pass\_request\_headers** on | off;

DEFAULT on

CONTEXT: http, server, location

Indicates whether the header fields of the original request are passed to the uwsgi server. See also the [uwsgi\\_pass\\_request\\_body](#) directive.

## uwsgi\_read\_timeout

SYNTAX: **uwsgi\_read\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Defines a timeout for reading a response from the uwsgi server. The timeout is set only between two successive read operations, not for the transmission of the whole response. If the uwsgi server does not transmit anything within this time, the connection is closed.

## uwsgi\_request\_buffering

SYNTAX: **uwsgi\_request\_buffering** on | off;

DEFAULT on

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Enables or disables buffering of a client request body.

When buffering is enabled, the entire request body is [read](#) from the client before sending the request to a uwsgi server.

When buffering is disabled, the request body is sent to the uwsgi server immediately as it is received. In this case, the request cannot be passed to the [next server](#) if nginx already started sending the request body.

When HTTP/1.1 chunked transfer encoding is used to send the original request body, the request body will be buffered regardless of the directive value.

### **uwsgi\_send\_timeout**

SYNTAX: **uwsgi\_send\_timeout** *time*;

DEFAULT 60s

CONTEXT: http, server, location

Sets a timeout for transmitting a request to the uwsgi server. The timeout is set only between two successive write operations, not for the transmission of the whole request. If the uwsgi server does not receive anything within this time, the connection is closed.

### **uwsgi\_socket\_keepalive**

SYNTAX: **uwsgi\_socket\_keepalive** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.15.6.

Configures the “TCP keepalive” behavior for outgoing connections to a uwsgi server. By default, the operating system’s settings are in effect for the socket. If the directive is set to the value “on”, the `SO_KEEPALIVE` socket option is turned on for the socket.

### **uwsgi\_ssl\_certificate**

SYNTAX: **uwsgi\_ssl\_certificate** *file*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.8.

Specifies a *file* with the certificate in the PEM format used for authentication to a secured uwsgi server.

Since version 1.21.0, variables can be used in the *file* name.

### **uwsgi\_ssl\_certificate\_key**

SYNTAX: **uwsgi\_ssl\_certificate\_key** *file*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.8.

Specifies a *file* with the secret key in the PEM format used for authentication to a secured uwsgi server.

The value `engine:name:id` can be specified instead of the *file* (1.7.9), which loads a secret key with a specified *id* from the OpenSSL engine *name*.

Since version 1.21.0, variables can be used in the *file* name.

## uwsgi\_ssl\_ciphers

SYNTAX: **uwsgi\_ssl\_ciphers** *ciphers*;

DEFAULT DEFAULT

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.8.

Specifies the enabled ciphers for requests to a secured uwsgi server. The ciphers are specified in the format understood by the OpenSSL library.

The full list can be viewed using the “`openssl ciphers`” command.

## uwsgi\_ssl\_conf\_command

SYNTAX: **uwsgi\_ssl\_conf\_command** *name value*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

Sets arbitrary OpenSSL configuration [commands](#) when establishing a connection with the secured uwsgi server.

The directive is supported when using OpenSSL 1.0.2 or higher.

Several `uwsgi_ssl_conf_command` directives can be specified on the same level. These directives are inherited from the previous configuration level if and only if there are no `uwsgi_ssl_conf_command` directives defined on the current level.

Note that configuring OpenSSL directly might result in unexpected behavior.

## uwsgi\_ssl\_crl

SYNTAX: **uwsgi\_ssl\_crl** *file*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Specifies a *file* with revoked certificates (CRL) in the PEM format used to [verify](#) the certificate of the secured uwsgi server.

## uwsgi\_ssl\_name

SYNTAX: **uwsgi\_ssl\_name** *name*;

DEFAULT host from uwsgi\_pass

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Allows overriding the server name used to [verify](#) the certificate of the secured uwsgi server and to be [passed through SNI](#) when establishing a connection with the secured uwsgi server.

By default, the host part from [uwsgi\\_pass](#) is used.

### **uwsgi\_ssl\_password\_file**

SYNTAX: **uwsgi\_ssl\_password\_file** *file*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.8.

Specifies a *file* with passphrases for [secret keys](#) where each passphrase is specified on a separate line. Passphrases are tried in turn when loading the key.

### **uwsgi\_ssl\_protocols**

SYNTAX: **uwsgi\_ssl\_protocols** [SSLv2] [SSLv3] [TLSv1] [TLSv1.1]  
[TLSv1.2] [TLSv1.3];

DEFAULT TLSv1 TLSv1.1 TLSv1.2

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.8.

Enables the specified protocols for requests to a secured uwsgi server.

### **uwsgi\_ssl\_server\_name**

SYNTAX: **uwsgi\_ssl\_server\_name** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Enables or disables passing of the server name through [TLS Server Name Indication extension](#) (SNI, RFC 6066) when establishing a connection with the secured uwsgi server.

### **uwsgi\_ssl\_session\_reuse**

SYNTAX: **uwsgi\_ssl\_session\_reuse** on | off;

DEFAULT on

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.8.

Determines whether SSL sessions can be reused when working with a secured uwsgi server. If the errors “SSL3\_GET\_FINISHED:digest check failed” appear in the logs, try disabling session reuse.

### **uwsgi\_ssl\_trusted\_certificate**

SYNTAX: **uwsgi\_ssl\_trusted\_certificate** *file*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) the certificate of the secured uwsgi server.

### **uwsgi\_ssl\_verify**

SYNTAX: **uwsgi\_ssl\_verify** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Enables or disables verification of the secured uwsgi server certificate.

### **uwsgi\_ssl\_verify\_depth**

SYNTAX: **uwsgi\_ssl\_verify\_depth** *number*;

DEFAULT 1

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.7.0.

Sets the verification depth in the secured uwsgi server certificates chain.

### **uwsgi\_store**

SYNTAX: **uwsgi\_store** on | off | *string*;

DEFAULT off

CONTEXT: http, server, location

Enables saving of files to a disk. The `on` parameter saves files with paths corresponding to the directives [alias](#) or [root](#). The `off` parameter disables saving of files. In addition, the file name can be set explicitly using the *string* with variables:

```
uwsgi_store /data/www$original_uri;
```

The modification time of files is set according to the received Last-Modified response header field. The response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the persistent store can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both saved files and a directory holding temporary files, set by the [uwsgi\\_temp\\_path](#) directive, are put on the same file system.

This directive can be used to create local copies of static unchangeable files, e.g.:

```
location /images/ {
    root          /data/www;
    error_page    404 = /fetch$uri;
}

location /fetch/ {
    internal;
```

```

uwsgi_pass          backend:9000;
...

uwsgi_store          on;
uwsgi_store_access   user:rw group:rw all:r;
uwsgi_temp_path      /data/temp;

alias                /data/www/;
}

```

### uwsgi\_store\_access

SYNTAX: **uwsgi\_store\_access** *users:permissions* ...;

DEFAULT `user:rw`

CONTEXT: http, server, location

Sets access permissions for newly created files and directories, e.g.:

```
uwsgi_store_access user:rw group:rw all:r;
```

If any group or all access permissions are specified then user permissions may be omitted:

```
uwsgi_store_access group:rw all:r;
```

### uwsgi\_temp\_file\_write\_size

SYNTAX: **uwsgi\_temp\_file\_write\_size** *size*;

DEFAULT `8k|16k`

CONTEXT: http, server, location

Limits the *size* of data written to a temporary file at a time, when buffering of responses from the uwsgi server to temporary files is enabled. By default, *size* is limited by two buffers set by the [uwsgi\\_buffer\\_size](#) and [uwsgi\\_buffers](#) directives. The maximum size of a temporary file is set by the [uwsgi\\_max-temp\\_file\\_size](#) directive.

### uwsgi\_temp\_path

SYNTAX: **uwsgi\_temp\_path** *path* [*level1* [*level2* [*level3*]]];

DEFAULT `uwsgi_temp`

CONTEXT: http, server, location

Defines a directory for storing temporary files with data received from uwsgi servers. Up to three-level subdirectory hierarchy can be used underneath the specified directory. For example, in the following configuration

```
uwsgi_temp_path /spool/nginx/uwsgi_temp 1 2;
```

a temporary file might look like this:

```
/spool/nginx/uwsgi_temp/7/45/00000123457
```

See also the `use_temp_path` parameter of the [uwsgi\\_cache\\_path](#) directive.



## 2.57 Module ngx\_http\_v2\_module

2.57.1 Summary	384
2.57.2 Known Issues	384
2.57.3 Example Configuration	384
2.57.4 Directives	385
http2_body_preread_size	385
http2_chunk_size	385
http2_idle_timeout	385
http2_max_concurrent_pushes	385
http2_max_concurrent_streams	386
http2_max_field_size	386
http2_max_header_size	386
http2_max_requests	386
http2_push	387
http2_push_preload	387
http2_recv_buffer_size	387
http2_recv_timeout	387
2.57.5 Embedded Variables	388

### 2.57.1 Summary

The `ngx_http_v2_module` module (1.9.5) provides support for [HTTP/2](#) and supersedes the `ngx_http_spdy_module` module.

This module is not built by default, it should be enabled with the `--with-http_v2_module` configuration parameter.

### 2.57.2 Known Issues

Before version 1.9.14, buffering of a client request body could not be disabled regardless of [proxy\\_request\\_buffering](#), [fastcgi\\_request\\_buffering](#), [uwsgi\\_request\\_buffering](#), and [scgi\\_request\\_buffering](#) directive values.

Before version 1.19.1, the [lingering\\_close](#) mechanism was not used to control closing HTTP/2 connections.

### 2.57.3 Example Configuration

```
server {
    listen 443 ssl http2;

    ssl_certificate server.crt;
    ssl_certificate_key server.key;
}
```

Note that accepting HTTP/2 connections over TLS requires the “Application-Layer Protocol Negotiation” (ALPN) TLS extension support, which is available since [OpenSSL](#) version 1.0.2.

Also note that if the [ssl\\_prefer\\_server\\_ciphers](#) directive is set to the value “on”, the [ciphers](#) should be configured to comply with [RFC 7540](#), [Appendix A](#) black list and supported by clients.

## 2.57.4 Directives

### **http2\_body\_preread\_size**

SYNTAX: **http2\_body\_preread\_size** *size*;  
DEFAULT 64k  
CONTEXT: http, server  
THIS DIRECTIVE APPEARED IN VERSION 1.11.0.

Sets the *size* of the buffer per each request in which the request body may be saved before it is started to be processed.

### **http2\_chunk\_size**

SYNTAX: **http2\_chunk\_size** *size*;  
DEFAULT 8k  
CONTEXT: http, server, location

Sets the maximum size of chunks into which the response body is sliced. A too low value results in higher overhead. A too high value impairs prioritization due to [HOL blocking](#).

### **http2\_idle\_timeout**

SYNTAX: **http2\_idle\_timeout** *time*;  
DEFAULT 3m  
CONTEXT: http, server

This directive is obsolete since version 1.19.7. The [keepalive\\_timeout](#) directive should be used instead.

Sets the timeout of inactivity after which the connection is closed.

### **http2\_max\_concurrent\_pushes**

SYNTAX: **http2\_max\_concurrent\_pushes** *number*;  
DEFAULT 10  
CONTEXT: http, server  
THIS DIRECTIVE APPEARED IN VERSION 1.13.9.

Limits the maximum number of concurrent [push](#) requests in a connection.

## http2\_max\_concurrent\_streams

SYNTAX: **http2\_max\_concurrent\_streams** *number*;

DEFAULT 128

CONTEXT: http, server

Sets the maximum number of concurrent HTTP/2 streams in a connection.

## http2\_max\_field\_size

SYNTAX: **http2\_max\_field\_size** *size*;

DEFAULT 4k

CONTEXT: http, server

This directive is obsolete since version 1.19.7. The [large\\_client\\_header\\_buffers](#) directive should be used instead.

Limits the maximum size of an [HPACK](#)-compressed request header field. The limit applies equally to both name and value. Note that if Huffman encoding is applied, the actual size of decompressed name and value strings may be larger. For most requests, the default limit should be enough.

## http2\_max\_header\_size

SYNTAX: **http2\_max\_header\_size** *size*;

DEFAULT 16k

CONTEXT: http, server

This directive is obsolete since version 1.19.7. The [large\\_client\\_header\\_buffers](#) directive should be used instead.

Limits the maximum size of the entire request header list after [HPACK](#) decompression. For most requests, the default limit should be enough.

## http2\_max\_requests

SYNTAX: **http2\_max\_requests** *number*;

DEFAULT 1000

CONTEXT: http, server

THIS DIRECTIVE APPEARED IN VERSION 1.11.6.

This directive is obsolete since version 1.19.7. The [keepalive\\_requests](#) directive should be used instead.

Sets the maximum number of requests (including [push](#) requests) that can be served through one HTTP/2 connection, after which the next client request will lead to connection closing and the need of establishing a new connection.

Closing connections periodically is necessary to free per-connection memory allocations. Therefore, using too high maximum number of requests could result in excessive memory usage and not recommended.

## http2\_push

SYNTAX: **http2\_push** *uri* | *off*;  
DEFAULT *off*  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.13.9.

Pre-emptively sends ([pushes](#)) a request to the specified *uri* along with the response to the original request. Only relative URIs with absolute path will be processed, for example:

```
http2_push /static/css/main.css;
```

The *uri* value can contain variables.

Several `http2_push` directives can be specified on the same configuration level. The `off` parameter cancels the effect of the `http2_push` directives inherited from the previous configuration level.

## http2\_push\_preload

SYNTAX: **http2\_push\_preload** *on* | *off*;  
DEFAULT *off*  
CONTEXT: http, server, location  
THIS DIRECTIVE APPEARED IN VERSION 1.13.9.

Enables automatic conversion of [preload links](#) specified in the `Link` response header fields into [push](#) requests.

## http2\_recv\_buffer\_size

SYNTAX: **http2\_recv\_buffer\_size** *size*;  
DEFAULT *256k*  
CONTEXT: http

Sets the size of the per [worker](#) input buffer.

## http2\_recv\_timeout

SYNTAX: **http2\_recv\_timeout** *time*;  
DEFAULT *30s*  
CONTEXT: http, server

This directive is obsolete since version 1.19.7. The [client\\_header\\_timeout](#) directive should be used instead.

Sets the timeout for expecting more data from the client, after which the connection is closed.

### 2.57.5 Embedded Variables

The `ngx_http_v2_module` module supports the following embedded variables:

*\$http2*

negotiated protocol identifier: “h2” for HTTP/2 over TLS, “h2c” for HTTP/2 over cleartext TCP, or an empty string otherwise.

## 2.58 Module ngx\_http\_xslt\_module

2.58.1 Summary	389
2.58.2 Example Configuration	389
2.58.3 Directives	389
xml_entities	389
xslt_last_modified	390
xslt_param	390
xslt_string_param	390
xslt_stylesheet	390
xslt_types	391

### 2.58.1 Summary

The `ngx_http_xslt_module` (0.7.8+) is a filter that transforms XML responses using one or more XSLT stylesheets.

This module is not built by default, it should be enabled with the `--with-http_xslt_module` configuration parameter.

This module requires the [libxml2](#) and [libxslt](#) libraries.

### 2.58.2 Example Configuration

```
location / {
    xml_entities    /site/dtd/entities.dtd;
    xslt_stylesheet /site/xslt/one.xslt param=value;
    xslt_stylesheet /site/xslt/two.xslt;
}
```

### 2.58.3 Directives

#### xml\_entities

SYNTAX: **xml\_entities** *path*;

DEFAULT —

CONTEXT: http, server, location

Specifies the DTD file that declares character entities. This file is compiled at the configuration stage. For technical reasons, the module is unable to use the external subset declared in the processed XML, so it is ignored and a specially defined file is used instead. This file should not describe the XML structure. It is enough to declare just the required character entities, for example:

```
<!ENTITY nbsp "&#xa0;">
```

## xslt\_last\_modified

SYNTAX: **xslt\_last\_modified** on | off;

DEFAULT off

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.5.1.

Allows preserving the Last-Modified header field from the original response during XSLT transformations to facilitate response caching.

By default, the header field is removed as contents of the response are modified during transformations and may contain dynamically generated elements or parts that are changed independently of the original response.

## xslt\_param

SYNTAX: **xslt\_param** *parameter value*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.18.

Defines the parameters for XSLT stylesheets. The *value* is treated as an XPath expression. The *value* can contain variables. To pass a string value to a stylesheet, the [xslt\\_string\\_param](#) directive can be used.

There could be several `xslt_param` directives. These directives are inherited from the previous configuration level if and only if there are no `xslt_param` and [xslt\\_string\\_param](#) directives defined on the current level.

## xslt\_string\_param

SYNTAX: **xslt\_string\_param** *parameter value*;

DEFAULT —

CONTEXT: http, server, location

THIS DIRECTIVE APPEARED IN VERSION 1.1.18.

Defines the string parameters for XSLT stylesheets. XPath expressions in the *value* are not interpreted. The *value* can contain variables.

There could be several `xslt_string_param` directives. These directives are inherited from the previous configuration level if and only if there are no [xslt\\_param](#) and `xslt_string_param` directives defined on the current level.

## xslt\_stylesheet

SYNTAX: **xslt\_stylesheet** *stylesheet* [*parameter=value ...*];

DEFAULT —

CONTEXT: location

Defines the XSLT stylesheet and its optional parameters. A stylesheet is compiled at the configuration stage.

Parameters can either be specified separately, or grouped in a single line using the “:” delimiter. If a parameter includes the “:” character, it should be

escaped as “%3A”. Also, `libxslt` requires to enclose parameters that contain non-alphanumeric characters into single or double quotes, for example:

```
param1='http%3A//www.example.com':param2=value2
```

The parameters description can contain variables, for example, the whole line of parameters can be taken from a single variable:

```
location / {
    xslt_stylesheet /site/xslt/one.xslt
                   $arg_xslt_params
                   param1=' $value1':param2=value2
                   param3=value3;
}
```

It is possible to specify several stylesheets. They will be applied sequentially in the specified order.

### **xslt\_types**

SYNTAX: **xslt\_types** *mime-type* ...;

DEFAULT `text/xml`

CONTEXT: `http`, `server`, `location`

Enables transformations in responses with the specified MIME types in addition to “`text/xml`”. The special value “`*`” matches any MIME type (0.8.29). If the transformation result is an HTML response, its MIME type is changed to “`text/html`”.



# Chapter 3

## Stream server modules

### 3.1 Module ngx\_stream\_core\_module

3.1.1	Summary	392
3.1.2	Example Configuration	392
3.1.3	Directives	393
	listen	393
	preread_buffer_size	395
	preread_timeout	396
	proxy_protocol_timeout	396
	resolver	396
	resolver_timeout	397
	server	397
	stream	397
	tcp_nodelay	397
	variables_hash_bucket_size	397
	variables_hash_max_size	398
3.1.4	Embedded Variables	398

#### 3.1.1 Summary

The ngx\_stream\_core\_module module is available since version 1.9.0. This module is not built by default, it should be enabled with the `--with-stream` configuration parameter.

#### 3.1.2 Example Configuration

```
worker_processes auto;

error_log /var/log/nginx/error.log info;

events {
    worker_connections 1024;
}

stream {
    upstream backend {
```

```

        hash $remote_addr consistent;

        server backend1.example.com:12345 weight=5;
        server 127.0.0.1:12345          max_fails=3 fail_timeout=30s;
        server unix:/tmp/backend3;
    }

    upstream dns {
        server 192.168.0.1:53535;
        server dns.example.com:53;
    }

    server {
        listen 12345;
        proxy_connect_timeout 1s;
        proxy_timeout 3s;
        proxy_pass backend;
    }

    server {
        listen 127.0.0.1:53 udp reuseport;
        proxy_timeout 20s;
        proxy_pass dns;
    }

    server {
        listen [::1]:12345;
        proxy_pass unix:/tmp/stream.socket;
    }
}

```

### 3.1.3 Directives

#### listen

SYNTAX: **listen** *address:port* [*ssl*] [*udp*] [*proxy\_protocol*]  
 [*fastopen=number*] [*backlog=number*] [*rcvbuf=size*]  
 [*sndbuf=size*] [*bind*] [*ipv6only=on|off*] [*reuseport*]  
 [*so\_keepalive=on|off*][*keepidle*]:[*keepintvl*]:[*keepcnt*];

DEFAULT —

CONTEXT: server

Sets the *address* and *port* for the socket on which the server will accept connections. It is possible to specify just the port. The address can also be a hostname, for example:

```

listen 127.0.0.1:12345;
listen *:12345;
listen 12345;      # same as *:12345
listen localhost:12345;

```

IPv6 addresses are specified in square brackets:

```

listen [::1]:12345;
listen [::]:12345;

```

UNIX-domain sockets are specified with the “unix:” prefix:

```

listen unix:/var/run/nginx.sock;

```

Port ranges (1.15.10) are specified with the first and last port separated by a hyphen:

```
listen 127.0.0.1:12345-12399;  
listen 12345-12399;
```

The `ssl` parameter allows specifying that all connections accepted on this port should work in SSL mode.

The `udp` parameter configures a listening socket for working with datagrams (1.9.13). In order to handle packets from the same address and port in the same session, the `reuseport` parameter should also be specified.

The `proxy_protocol` parameter (1.11.4) allows specifying that all connections accepted on this port should use the [PROXY protocol](#).

The PROXY protocol version 2 is supported since version 1.13.11.

The `listen` directive can have several additional parameters specific to socket-related system calls.

`fastopen=number`

enables “[TCP Fast Open](#)” for the listening socket (1.21.0) and [limits](#) the maximum length for the queue of connections that have not yet completed the three-way handshake.

Do not enable this feature unless the server can handle receiving the [same SYN packet with data](#) more than once.

`backlog=number`

sets the `backlog` parameter in the `listen` call that limits the maximum length for the queue of pending connections (1.9.2). By default, `backlog` is set to -1 on FreeBSD, DragonFly BSD, and macOS, and to 511 on other platforms.

`rcvbuf=size`

sets the receive buffer size (the `SO_RCVBUF` option) for the listening socket (1.11.13).

`sndbuf=size`

sets the send buffer size (the `SO_SNDBUF` option) for the listening socket (1.11.13).

`bind`

this parameter instructs to make a separate `bind` call for a given address:port pair. The fact is that if there are several `listen` directives with the same port but different addresses, and one of the `listen` directives listens on all addresses for the given port (`*:port`), nginx will bind only to `*:port`. It should be noted that the `getsockname` system call will be made in this case to determine the address that accepted the connection. If the `backlog`, `rcvbuf`, `sndbuf`, `ipv6only`, `reuseport`, or `so_keepalive` parameters are used then for a given `address:port` pair a separate `bind` call will always be made.

`ipv6only=on|off`

this parameter determines (via the `IPV6_V6ONLY` socket option) whether an IPv6 socket listening on a wildcard address `[::]` will accept only IPv6 connections or both IPv6 and IPv4 connections. This parameter is turned on by default. It can only be set once on start.

`reuseport`

this parameter (1.9.1) instructs to create an individual listening socket for each worker process (using the `SO_REUSEPORT` socket option on Linux 3.9+ and DragonFly BSD, or `SO_REUSEPORT_LB` on FreeBSD 12+), allowing a kernel to distribute incoming connections between worker processes. This currently works only on Linux 3.9+, DragonFly BSD, and FreeBSD 12+ (1.15.1).

Inappropriate use of this option may have its security [implications](#).

`so_keepalive=on|off[[keepidle]:[keepintvl]:[keepcnt]]`

this parameter configures the “TCP keepalive” behavior for the listening socket. If this parameter is omitted then the operating system’s settings will be in effect for the socket. If it is set to the value “on”, the `SO_KEEPALIVE` option is turned on for the socket. If it is set to the value “off”, the `SO_KEEPALIVE` option is turned off for the socket. Some operating systems support setting of TCP keepalive parameters on a per-socket basis using the `TCP_KEEPIDLE`, `TCP_KEEPINTVL`, and `TCP_KEEPCNT` socket options. On such systems (currently, Linux 2.4+, NetBSD 5+, and FreeBSD 9.0-STABLE), they can be configured using the *keepidle*, *keepintvl*, and *keepcnt* parameters. One or two parameters may be omitted, in which case the system default setting for the corresponding socket option will be in effect. For example,

```
so_keepalive=30m::10
```

will set the idle timeout (`TCP_KEEPIDLE`) to 30 minutes, leave the probe interval (`TCP_KEEPINTVL`) at its system default, and set the probes count (`TCP_KEEPCNT`) to 10 probes.

Different servers must listen on different *address:port* pairs.

### **preread\_buffer\_size**

SYNTAX: **preread\_buffer\_size** *size*;

DEFAULT 16k

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.11.5.

Specifies a *size* of the preread buffer.

## preread\_timeout

SYNTAX: **preread\_timeout** *timeout*;  
DEFAULT 30s  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 1.11.5.

Specifies a *timeout* of the preread phase.

## proxy\_protocol\_timeout

SYNTAX: **proxy\_protocol\_timeout** *timeout*;  
DEFAULT 30s  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 1.11.4.

Specifies a *timeout* for reading the PROXY protocol header to complete. If no entire header is transmitted within this time, the connection is closed.

## resolver

SYNTAX: **resolver** *address* ... [valid=*time*] [ipv4=on|off] [ipv6=on|off]  
[status\_zone=*zone*];  
DEFAULT —  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 1.11.3.

Configures name servers used to resolve names of upstream servers into addresses, for example:

```
resolver 127.0.0.1 [::1]:5353;
```

The address can be specified as a domain name or IP address, with an optional port. If port is not specified, the port 53 is used. Name servers are queried in a round-robin fashion.

By default, nginx will look up both IPv4 and IPv6 addresses while resolving. If looking up of IPv4 or IPv6 addresses is not desired, the `ipv4=off` (1.23.1) or the `ipv6=off` parameter can be specified.

By default, nginx caches answers using the TTL value of a response. The optional `valid` parameter allows overriding it:

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

To prevent DNS spoofing, it is recommended configuring DNS servers in a properly secured trusted local network.

The optional `status_zone` parameter (1.17.1) enables [collection](#) of DNS server statistics of requests and responses in the specified *zone*. The parameter is available as part of our [commercial subscription](#).

Before version 1.11.3, this directive was available as part of our [commercial subscription](#).

### resolver\_timeout

SYNTAX: **resolver\_timeout** *time*;

DEFAULT 30s

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.11.3.

Sets a timeout for name resolution, for example:

```
resolver_timeout 5s;
```

Before version 1.11.3, this directive was available as part of our [commercial subscription](#).

### server

SYNTAX: **server** { ... }

DEFAULT —

CONTEXT: stream

Sets the configuration for a server.

### stream

SYNTAX: **stream** { ... }

DEFAULT —

CONTEXT: main

Provides the configuration file context in which the stream server directives are specified.

### tcp\_nodelay

SYNTAX: **tcp\_nodelay** on | off;

DEFAULT on

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.9.4.

Enables or disables the use of the TCP\_NODELAY option. The option is enabled for both client and proxied server connections.

### variables\_hash\_bucket\_size

SYNTAX: **variables\_hash\_bucket\_size** *size*;

DEFAULT 64

CONTEXT: stream

THIS DIRECTIVE APPEARED IN VERSION 1.11.2.

Sets the bucket size for the variables hash table. The details of setting up hash tables are provided in a separate [document](#).

### **variables\_hash\_max\_size**

SYNTAX: **variables\_hash\_max\_size** *size*;

DEFAULT 1024

CONTEXT: stream

THIS DIRECTIVE APPEARED IN VERSION 1.11.2.

Sets the maximum *size* of the variables hash table. The details of setting up hash tables are provided in a separate [document](#).

## **3.1.4 Embedded Variables**

The `ngx_stream_core_module` module supports variables since 1.11.2.

*\$binary\_remote\_addr*

client address in a binary form, value's length is always 4 bytes for IPv4 addresses or 16 bytes for IPv6 addresses

*\$bytes\_received*

number of bytes received from a client (1.11.4)

*\$bytes\_sent*

number of bytes sent to a client

*\$connection*

connection serial number

*\$hostname*

host name

*\$msec*

current time in seconds with the milliseconds resolution

*\$nginx\_version*

nginx version

*\$pid*

PID of the worker process

*\$protocol*

protocol used to communicate with the client: TCP or UDP (1.11.4)

*\$proxy\_protocol\_addr*

client address from the PROXY protocol header (1.11.4)

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the [listen](#) directive.

*\$proxy\_protocol\_port*

client port from the PROXY protocol header (1.11.4)

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the [listen](#) directive.

*\$proxy\_protocol\_server\_addr*

server address from the PROXY protocol header (1.17.6)

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the [listen](#) directive.

*\$proxy\_protocol\_server\_port*

server port from the PROXY protocol header (1.17.6)

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the [listen](#) directive.

*\$proxy\_protocol\_tlv\_name*

TLV from the PROXY Protocol header (1.23.2). The name can be a TLV type name or its numeric value. In the latter case, the value is hexadecimal and should be prefixed with `0x`:

```
$proxy_protocol_tlv_alpn
$proxy_protocol_tlv_0x01
```

SSL TLVs can also be accessed by TLV type name or its numeric value, both prefixed by `ssl_`:

```
$proxy_protocol_tlv_ssl_version
$proxy_protocol_tlv_ssl_0x21
```

The following TLV type names are supported:

- `alpn (0x01)` - upper layer protocol used over the connection
- `authority (0x02)` - host name value passed by the client
- `unique_id (0x05)` - unique connection id
- `netns (0x30)` - name of the namespace
- `ssl (0x20)` - binary SSL TLV structure

The following SSL TLV type names are supported:

- `ssl_version (0x21)` - SSL version used in client connection
- `ssl_cn (0x22)` - SSL certificate Common Name
- `ssl_cipher (0x23)` - name of the used cipher
- `ssl_sig_alg (0x24)` - algorithm used to sign the certificate
- `ssl_key_alg (0x25)` - public-key algorithm

Also, the following special SSL TLV type name is supported:

- `ssl_verify` - client SSL certificate verification result, zero if the client presented a certificate and it was successfully verified, and non-zero otherwise

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the [listen](#) directive.

*\$remote\_addr*

client address

*\$remote\_port*

client port



*\$server\_addr*

an address of the server which accepted a connection

Computing a value of this variable usually requires one system call. To avoid a system call, the [listen](#) directives must specify addresses and use the `bind` parameter.

*\$server\_port*

port of the server which accepted a connection

*\$session\_time*

session duration in seconds with a milliseconds resolution (1.11.4);

*\$status*

session status (1.11.4), can be one of the following:

200

session completed successfully

400

client data could not be parsed, for example, the PROXY protocol header

403

access forbidden, for example, when access is limited for [certain client addresses](#)

500

internal server error

502

bad gateway, for example, if an upstream server could not be selected or reached.

503

service unavailable, for example, when access is limited by the [number of connections](#)

*\$time\_iso8601*

local time in the ISO 8601 standard format

*\$time\_local*

local time in the Common Log Format

## 3.2 Module ngx\_stream\_access\_module

3.2.1	Summary	401
3.2.2	Example Configuration	401
3.2.3	Directives	401
	allow	401
	deny	401

### 3.2.1 Summary

The `ngx_stream_access_module` module (1.9.2) allows limiting access to certain client addresses.

### 3.2.2 Example Configuration

```
server {
    ...
    deny 192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0db8::/32;
    deny all;
}
```

The rules are checked in sequence until the first match is found. In this example, access is allowed only for IPv4 networks `10.1.1.0/16` and `192.168.1.0/24` excluding the address `192.168.1.1`, and for IPv6 network `2001:0db8::/32`.

### 3.2.3 Directives

#### allow

SYNTAX: **allow** *address* | *CIDR* | `unix:` | `all`;  
 DEFAULT —  
 CONTEXT: stream, server

Allows access for the specified network or address. If the special value `unix:` is specified, allows access for all UNIX-domain sockets.

#### deny

SYNTAX: **deny** *address* | *CIDR* | `unix:` | `all`;  
 DEFAULT —  
 CONTEXT: stream, server

Denies access for the specified network or address. If the special value `unix:` is specified, denies access for all UNIX-domain sockets.

## 3.3 Module ngx\_stream\_geo\_module

3.3.1	Summary	402
3.3.2	Example Configuration	402
3.3.3	Directives	402
	geo	402

### 3.3.1 Summary

The ngx\_stream\_geo\_module module (1.11.3) creates variables with values depending on the client IP address.

### 3.3.2 Example Configuration

```
geo $geo {
    default          0;

    127.0.0.1        2;
    192.168.1.0/24    1;
    10.1.0.0/16       1;

    ::1              2;
    2001:0db8::/32    1;
}
```

### 3.3.3 Directives

#### geo

SYNTAX: **geo** [*\$address*] *\$variable* { ... }

DEFAULT —

CONTEXT: stream

Describes the dependency of values of the specified variable on the client IP address. By default, the address is taken from the *\$remote\_addr* variable, but it can also be taken from another variable, for example:

```
geo $arg_remote_addr $geo {
    ...;
}
```

Since variables are evaluated only when used, the mere existence of even a large number of declared “geo” variables does not cause any extra costs for connection processing.

If the value of a variable does not represent a valid IP address then the “255.255.255.255” address is used.

Addresses are specified either as prefixes in CIDR notation (including individual addresses) or as ranges.

The following special parameters are also supported:

`delete`

deletes the specified network.

`default`

a value set to the variable if the client address does not match any of the specified addresses. When addresses are specified in CIDR notation, “0.0.0.0/0” and “::/0” can be used instead of `default`. When `default` is not specified, the default value will be an empty string.

`include`

includes a file with addresses and values. There can be several inclusions.

`ranges`

indicates that addresses are specified as ranges. This parameter should be the first. To speed up loading of a geo base, addresses should be put in ascending order.

Example:

```
geo $country {
    default      ZZ;
    include      conf/geo.conf;
    delete      127.0.0.0/16;

    127.0.0.0/24  US;
    127.0.0.1/32  RU;
    10.1.0.0/16   RU;
    192.168.1.0/24 UK;
}
```

The `conf/geo.conf` file could contain the following lines:

```
10.2.0.0/16    RU;
192.168.2.0/24 RU;
```

A value of the most specific match is used. For example, for the 127.0.0.1 address the value “RU” will be chosen, not “US”.

Example with ranges:

```
geo $country {
    ranges;
    default      ZZ;
    127.0.0.0-127.0.0.0    US;
    127.0.0.1-127.0.0.1    RU;
    127.0.0.1-127.0.0.255  US;
    10.1.0.0-10.1.255.255  RU;
    192.168.1.0-192.168.1.255 UK;
}
```

## 3.4 Module ngx\_stream\_geoip\_module

3.4.1	Summary	404
3.4.2	Example Configuration	404
3.4.3	Directives	404
	<a href="#">geoip_country</a>	404
	<a href="#">geoip_city</a>	405
	<a href="#">geoip_org</a>	406

### 3.4.1 Summary

The `ngx_stream_geoip_module` module (1.11.3) creates variables with values depending on the client IP address, using the precompiled [MaxMind](#) databases.

When using the databases with IPv6 support, IPv4 addresses are looked up as IPv4-mapped IPv6 addresses.

This module is not built by default, it should be enabled with the `--with-stream_geoip_module` configuration parameter.

This module requires the [MaxMind GeoIP](#) library.

### 3.4.2 Example Configuration

```
stream {
    geoip_country      GeoIP.dat;
    geoip_city         GeoLiteCity.dat;

    map $geoip_city_continent_code $nearest_server {
        default        example.com;
        EU             eu.example.com;
        NA             na.example.com;
        AS             as.example.com;
    }
    ...
}
```

### 3.4.3 Directives

#### `geoip_country`

SYNTAX: **geoip\_country** *file*;

DEFAULT —

CONTEXT: stream

Specifies a database used to determine the country depending on the client IP address. The following variables are available when using this database:

*\$geoip\_country\_code*

two-letter country code, for example, “RU”, “US”.

*\$geoip\_country\_code3*

three-letter country code, for example, “RUS”, “USA”.

*\$geoip\_country\_name*

country name, for example, “Russian Federation”, “United States”.

## geoip\_city

SYNTAX: **geoip\_city** *file*;

DEFAULT —

CONTEXT: stream

Specifies a database used to determine the country, region, and city depending on the client IP address. The following variables are available when using this database:

*\$geoip\_area\_code*

telephone area code (US only).

This variable may contain outdated information since the corresponding database field is deprecated.

*\$geoip\_city\_continent\_code*

two-letter continent code, for example, “EU”, “NA”.

*\$geoip\_city\_country\_code*

two-letter country code, for example, “RU”, “US”.

*\$geoip\_city\_country\_code3*

three-letter country code, for example, “RUS”, “USA”.

*\$geoip\_city\_country\_name*

country name, for example, “Russian Federation”, “United States”.

*\$geoip\_dma\_code*

DMA region code in US (also known as “metro code”), according to the [geotargeting](#) in Google AdWords API.

*\$geoip\_latitude*

latitude.

*\$geoip\_longitude*

longitude.

*\$geoip\_region*

two-symbol country region code (region, territory, state, province, federal land and the like), for example, “48”, “DC”.

*\$geoip\_region\_name*

country region name (region, territory, state, province, federal land and the like), for example, “Moscow City”, “District of Columbia”.

*\$geoip\_city*

city name, for example, “Moscow”, “Washington”.

*\$geoip\_postal\_code*

postal code.

**geoip\_org**

SYNTAX: **geoip\_org** *file*;

DEFAULT —

CONTEXT: stream

Specifies a database used to determine the organization depending on the client IP address. The following variable is available when using this database:

*\$geoip\_org*

organization name, for example, “The University of Melbourne”.

## 3.5 Module ngx\_stream\_js\_module

3.5.1	Summary	407
3.5.2	Example Configuration	407
3.5.3	Directives	409
	js_access	409
	js_fetch_buffer_size	409
	js_fetch_ciphers	409
	js_fetch_max_response_buffer_size	409
	js_fetch_protocols	410
	js_fetch_timeout	410
	js_fetch_trusted_certificate	410
	js_fetch_verify	410
	js_fetch_verify_depth	410
	js_filter	411
	js_import	411
	js_include	411
	js_path	412
	js_preload_object	412
	js_preread	412
	js_set	413
	js_var	413
3.5.4	Session Object Properties	414

### 3.5.1 Summary

The `ngx_stream_js_module` module is used to implement handlers in `njs` — a subset of the JavaScript language.

Download and install instructions are available [here](#).

### 3.5.2 Example Configuration

The example works since 0.4.0.

```
stream {
    js_import stream.js;

    js_set $bar stream.bar;
    js_set $req_line stream.req_line;

    server {
        listen 12345;

        js_preread stream.preread;
        return $req_line;
    }

    server {
        listen 12346;

        js_access stream.access;
        proxy_pass 127.0.0.1:8000;
        js_filter stream.header_inject;
    }
}
```



```

    }
}

http {
    server {
        listen 8000;
        location / {
            return 200 $http_foo\n;
        }
    }
}

```

The stream.js file:

```

var line = '';

function bar(s) {
    var v = s.variables;
    s.log("hello from bar() handler!");
    return "bar-var" + v.remote_port + "; pid=" + v.pid;
}

function preread(s) {
    s.on('upload', function (data, flags) {
        var n = data.indexOf('\n');
        if (n != -1) {
            line = data.substr(0, n);
            s.done();
        }
    });
}

function req_line(s) {
    return line;
}

// Read HTTP request line.
// Collect bytes in 'req' until
// request line is read.
// Injects HTTP header into a client's request

var my_header = 'Foo: foo';
function header_inject(s) {
    var req = '';
    s.on('upload', function(data, flags) {
        req += data;
        var n = req.search('\n');
        if (n != -1) {
            var rest = req.substr(n + 1);
            req = req.substr(0, n + 1);
            s.send(req + my_header + '\r\n' + rest, flags);
            s.off('upload');
        }
    });
}

function access(s) {
    if (s.remoteAddress.match('^192.*')) {
        s.deny();
        return;
    }

    s.allow();
}

export default {bar, preread, req_line, header_inject, access};

```

### 3.5.3 Directives

#### js\_access

SYNTAX: **js\_access** *function* | *module.function*;

DEFAULT —

CONTEXT: stream, server

Sets an njs function which will be called at the access phase. Since 0.4.0, a module function can be referenced.

The function is called once at the moment when the stream session reaches the access phase for the first time. The function is called with the following arguments:

`s`

the Stream Session object

At this phase, it is possible to perform initialization or register a callback with the `s.on()` method for each incoming data chunk until one of the following methods are called: `s.allow()`, `s.decline()`, `s.done()`. As soon as one of these methods is called, the stream session processing switches to the next phase and all current `s.on()` callbacks are dropped.

#### js\_fetch\_buffer\_size

SYNTAX: **js\_fetch\_buffer\_size** *size*;

DEFAULT 16k

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 0.7.4.

Sets the *size* of the buffer used for reading and writing with Fetch API.

#### js\_fetch\_ciphers

SYNTAX: **js\_fetch\_ciphers** *ciphers*;

DEFAULT HIGH:!aNULL:!MD5

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 0.7.0.

Specifies the enabled ciphers for HTTPS connections with Fetch API. The ciphers are specified in the format understood by the OpenSSL library.

The full list can be viewed using the “`openssl ciphers`” command.

#### js\_fetch\_max\_response\_buffer\_size

SYNTAX: **js\_fetch\_max\_response\_buffer\_size** *size*;

DEFAULT 1m

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 0.7.4.

Sets the maximum *size* of the response received with Fetch API.

### **js\_fetch\_protocols**

SYNTAX: **js\_fetch\_protocols** [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];  
DEFAULT TLSv1 TLSv1.1 TLSv1.2  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 0.7.0.

Enables the specified protocols for HTTPS connections with Fetch API.

### **js\_fetch\_timeout**

SYNTAX: **js\_fetch\_timeout** *time*;  
DEFAULT 60s  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 0.7.4.

Defines a timeout for reading and writing for Fetch API. The timeout is set only between two successive read/write operations, not for the whole response. If no data is transmitted within this time, the connection is closed.

### **js\_fetch\_trusted\_certificate**

SYNTAX: **js\_fetch\_trusted\_certificate** *file*;  
DEFAULT —  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 0.7.0.

Specifies a *file* with trusted CA certificates in the PEM format used to verify the HTTPS certificate with Fetch API.

### **js\_fetch\_verify**

SYNTAX: **js\_fetch\_verify** on | off;  
DEFAULT on  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 0.7.4.

Enables or disables verification of the HTTPS server certificate with Fetch API.

### **js\_fetch\_verify\_depth**

SYNTAX: **js\_fetch\_verify\_depth** *number*;  
DEFAULT 100  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 0.7.0.

Sets the verification depth in the HTTPS server certificates chain with Fetch API.

## js\_filter

SYNTAX: **js\_filter** *function* | *module.function*;

DEFAULT —

CONTEXT: stream, server

Sets a data filter. Since 0.4.0, a module function can be referenced. The filter function is called once at the moment when the stream session reaches the content phase.

The filter function is called with the following arguments:

- s  
the Stream Session object

At this phase, it is possible to perform initialization or register a callback with the `s.on()` method for each incoming data chunk. The `s.off()` method may be used to unregister a callback and stop filtering.

As the `js_filter` handler returns its result immediately, it supports only synchronous operations. Thus, asynchronous operations such as `ngx.fetch()` or `setTimeout()` are not supported.

## js\_import

SYNTAX: **js\_import** *module.js* | *export\_name from module.js*;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 0.4.0.

Imports a module that implements location and variable handlers in `njs`. The `export_name` is used as a namespace to access module functions. If the `export_name` is not specified, the module name will be used as a namespace.

```
js_import stream.js;
```

Here, the module name `stream` is used as a namespace while accessing exports. If the imported module exports `foo()`, `stream.foo` is used to refer to it.

Several `js_import` directives can be specified.

The directive can be specified on the `server` level since 0.7.7.

## js\_include

SYNTAX: **js\_include** *file*;

DEFAULT —

CONTEXT: stream

Specifies a file that implements server and variable handlers in `njs`:

```
nginx.conf:
js_include stream.js;
js_set     $js_addr address;
server {
    listen 127.0.0.1:12345;
    return $js_addr;
}

stream.js:
function address(s) {
    return s.remoteAddress;
}
```

The directive was made obsolete in version 0.4.0 and was removed in version 0.7.1. The [js\\_import](#) directive should be used instead.

## js\_path

SYNTAX: **js\_path** *path*;  
DEFAULT —  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 0.3.0.

Sets an additional path for njs modules.

The directive can be specified on the server level since 0.7.7.

## js\_preload\_object

SYNTAX: **js\_preload\_object** *name.json* | *name* from *file.json*;  
DEFAULT —  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 0.7.8.

Preloads an immutable object at configure time. The name is used a name of the global variable though which the object is available in njs code. If the name is not specified, the file name will be used instead.

```
js_preload_object map.json;
```

Here, the map is used as a name while accessing the preloaded object. Several `js_preload_object` directives can be specified.

## js\_preread

SYNTAX: **js\_preread** *function* | *module.function*;  
DEFAULT —  
CONTEXT: stream, server

Sets an njs function which will be called at the preread phase. Since 0.4.0, a module function can be referenced.

The function is called once at the moment when the stream session reaches the preread phase for the first time. The function is called with the following arguments:

`s`  
the Stream Session object

At this phase, it is possible to perform initialization or register a callback with the `s.on()` method for each incoming data chunk until one of the following methods are called: `s.allow()`, `s.decline()`, `s.done()`. When one of these methods is called, the stream session switches to the next phase and all current `s.on()` callbacks are dropped.

As the `js_preread` handler returns its result immediately, it supports only synchronous callbacks. Thus, asynchronous callbacks such as `ngx.fetch()` or `setTimeout()` are not supported. Nevertheless, asynchronous operations are supported in `s.on()` callbacks in the preread phase. See [this example](#) for more information.

## js\_set

SYNTAX: **js\_set** *\$variable function | module.function*;  
DEFAULT —  
CONTEXT: stream, server

Sets an `njs` function for the specified variable. Since 0.4.0, a module function can be referenced.

The function is called when the variable is referenced for the first time for a given request. The exact moment depends on a phase at which the variable is referenced. This can be used to perform some logic not related to variable evaluation. For example, if the variable is referenced only in the [log\\_format](#) directive, its handler will not be executed until the log phase. This handler can be used to do some cleanup right before the request is freed.

As the `js_set` handler returns its result immediately, it supports only synchronous callbacks. Thus, asynchronous callbacks such as `ngx.fetch()` or `setTimeout()` are not supported.

The directive can be specified on the server level since 0.7.7.

## js\_var

SYNTAX: **js\_var** *\$variable [value]*;  
DEFAULT —  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 0.5.3.

Declares a writable variable. The value can contain text, variables, and their combination.

The directive can be specified on the `server` level since 0.7.7.

### 3.5.4 Session Object Properties

Each stream njs handler receives one argument, a stream session object.

## 3.6 Module ngx\_stream\_keyval\_module

3.6.1	Summary	415
3.6.2	Example Configuration	415
3.6.3	Directives	415
	keyval	415
	keyval_zone	416

### 3.6.1 Summary

The ngx\_stream\_keyval\_module module (1.13.7) creates variables with values taken from key-value pairs managed by the [API](#) or a variable that can also be set with [njs](#).

This module is available as part of our [commercial subscription](#).

### 3.6.2 Example Configuration

```
http {
    server {
        ...
        location /api {
            api write=on;
        }
    }
}

stream {
    keyval_zone zone=one:32k state=/var/lib/nginx/state/one.keyval;
    keyval      $ssl_server_name $name zone=one;

    server {
        listen          12345 ssl;
        proxy_pass       $name;
        ssl_certificate  /usr/local/nginx/conf/cert.pem;
        ssl_certificate_key /usr/local/nginx/conf/cert.key;
    }
}
```

### 3.6.3 Directives

#### keyval

SYNTAX: **keyval** *key* *\$variable* zone=*name*;

DEFAULT —

CONTEXT: stream

Creates a new *\$variable* whose value is looked up by the *key* in the key-value database. Matching rules are defined by the type parameter of the [keyval\\_zone](#) directive. The database is stored in a shared memory zone specified by the zone parameter.



## keyval\_zone

SYNTAX: **keyval\_zone** zone=*name:size* [state=*file*] [timeout=*time*]  
 [type=string|ip|prefix] [sync];

DEFAULT —

CONTEXT: stream

Sets the *name* and *size* of the shared memory zone that keeps the key-value database. Key-value pairs are managed by the [API](#).

The optional *state* parameter specifies a *file* that keeps the current state of the key-value database in the JSON format and makes it persistent across nginx restarts. Changing the file content directly should be avoided.

Examples:

```
keyval_zone zone=one:32k state=/var/lib/nginx/state/one.keyval; # path for
Linux
keyval_zone zone=one:32k state=/var/db/nginx/state/one.keyval; # path for
FreeBSD
```

The optional *timeout* parameter (1.15.0) sets the time after which key-value pairs are removed from the zone.

The optional *type* parameter (1.17.1) activates an extra index optimized for matching the key of a certain type and defines matching rules when evaluating a [keyval](#) \$variable.

The index is stored in the same shared memory zone and thus requires additional storage.

type=string

default, no index is enabled; variable lookup is performed using exact match of the record key and a search key

type=ip

the search key is the textual representation of IPv4 or IPv6 address or CIDR range; to match a record key, the search key must belong to a subnet specified by a record key or exactly match an IP address

type=prefix

variable lookup is performed using prefix match of a record key and a search key (1.17.5); to match a record key, the record key must be a prefix of the search key

The optional *sync* parameter (1.15.0) enables [synchronization](#) of the shared memory zone. The synchronization requires the *timeout* parameter to be set.

If the synchronization is enabled, removal of key-value pairs (no matter [one](#) or [all](#)) will be performed only on a target cluster node. The same key-value pairs on other cluster nodes will be removed upon *timeout*.

## 3.7 Module ngx\_stream\_limit\_conn\_module

3.7.1	Summary	417
3.7.2	Example Configuration	417
3.7.3	Directives	417
	<a href="#">limit_conn</a>	417
	<a href="#">limit_conn_dry_run</a>	418
	<a href="#">limit_conn_log_level</a>	418
	<a href="#">limit_conn_zone</a>	418
3.7.4	Embedded Variables	419

### 3.7.1 Summary

The `ngx_stream_limit_conn_module` module (1.9.3) is used to limit the number of connections per the defined key, in particular, the number of connections from a single IP address.

### 3.7.2 Example Configuration

```
stream {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {
        ...

        limit_conn      addr 1;
        limit_conn_log_level error;
    }
}
```

### 3.7.3 Directives

#### `limit_conn`

SYNTAX: **limit\_conn** *zone number*;

DEFAULT —

CONTEXT: stream, server

Sets the shared memory zone and the maximum allowed number of connections for a given key value. When this limit is exceeded, the server will close the connection. For example, the directives

```
limit_conn_zone $binary_remote_addr zone=addr:10m;

server {
    ...
    limit_conn addr 1;
}
```

allow only one connection per an IP address at a time.

When several `limit_conn` directives are specified, any configured limit will apply.

These directives are inherited from the previous configuration level if and only if there are no `limit_conn` directives defined on the current level.

### **limit\_conn\_dry\_run**

SYNTAX: **limit\_conn\_dry\_run** on | off;

DEFAULT off

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.17.6.

Enables the dry run mode. In this mode, the number of connections is not limited, however, in the shared memory zone, the number of excessive connections is accounted as usual.

### **limit\_conn\_log\_level**

SYNTAX: **limit\_conn\_log\_level** info | notice | warn | error;

DEFAULT error

CONTEXT: stream, server

Sets the desired logging level for cases when the server limits the number of connections.

### **limit\_conn\_zone**

SYNTAX: **limit\_conn\_zone** *key* zone=*name:size*;

DEFAULT —

CONTEXT: stream

Sets parameters for a shared memory zone that will keep states for various keys. In particular, the state includes the current number of connections. The *key* can contain text, variables, and their combinations (1.11.2). Connections with an empty key value are not accounted. Usage example:

```
limit_conn_zone $binary_remote_addr zone=addr:10m;
```

Here, the key is a client IP address set by the `$binary_remote_addr` variable. The size of `$binary_remote_addr` is 4 bytes for IPv4 addresses or 16 bytes for IPv6 addresses. The stored state always occupies 32 or 64 bytes on 32-bit platforms and 64 bytes on 64-bit platforms. One megabyte zone can keep about 32 thousand 32-byte states or about 16 thousand 64-byte states. If the zone storage is exhausted, the server will close the connection.

Additionally, as part of our [commercial subscription](#), the [status information](#) for each such shared memory zone can be [obtained](#) or [reset](#) with the [API](#) since 1.17.7.

### 3.7.4 Embedded Variables

*\$limit\_conn\_status*

keeps the result of limiting the number of connections (1.17.6): PASSED, REJECTED, or REJECTED\_DRY\_RUN

## 3.8 Module ngx\_stream\_log\_module

3.8.1	Summary	420
3.8.2	Example Configuration	420
3.8.3	Directives	420
	access_log	420
	log_format	421
	open_log_file_cache	422

### 3.8.1 Summary

The ngx\_stream\_log\_module module (1.11.4) writes session logs in the specified format.

### 3.8.2 Example Configuration

```
log_format basic '$remote_addr [$time_local] '
                '$protocol $status $bytes_sent $bytes_received '
                '$session_time';

access_log /spool/logs/nginx-access.log basic buffer=32k;
```

### 3.8.3 Directives

#### access\_log

SYNTAX: **access\_log** *path format* [buffer=*size*] [gzip[=*level*]]  
 [flush=*time*] [if=*condition*];

SYNTAX: **access\_log** off;

DEFAULT off

CONTEXT: stream, server

Sets the path, [format](#), and configuration for a buffered log write. Several logs can be specified on the same configuration level. Logging to [syslog](#) can be configured by specifying the “syslog:” prefix in the first parameter. The special value off cancels all access\_log directives on the current level.

If either the buffer or gzip parameter is used, writes to log will be buffered.

The buffer size must not exceed the size of an atomic write to a disk file. For FreeBSD this size is unlimited.

When buffering is enabled, the data will be written to the file:

- if the next log line does not fit into the buffer;
- if the buffered data is older than specified by the flush parameter;
- when a worker process is [re-opening](#) log files or is shutting down.

If the `gzip` parameter is used, then the buffered data will be compressed before writing to the file. The compression level can be set between 1 (fastest, less compression) and 9 (slowest, best compression). By default, the buffer size is equal to 64K bytes, and the compression level is set to 1. Since the data is compressed in atomic blocks, the log file can be decompressed or read by “`zcat`” at any time.

Example:

```
access_log /path/to/log.gz basic gzip flush=5m;
```

For gzip compression to work, nginx must be built with the zlib library.

The file path can contain variables, but such logs have some constraints:

- the `user` whose credentials are used by worker processes should have permissions to create files in a directory with such logs;
- buffered writes do not work;
- the file is opened and closed for each log write. However, since the descriptors of frequently used files can be stored in a `cache`, writing to the old file can continue during the time specified by the `open_log_file_-cache` directive’s `valid` parameter

The `if` parameter enables conditional logging. A session will not be logged if the *condition* evaluates to “0” or an empty string.

## log\_format

SYNTAX: **log\_format** *name* [`escape=default|json|none`] *string* ...;

DEFAULT —

CONTEXT: stream

Specifies the log format, for example:

```
log_format proxy '$remote_addr [$time_local] '
                 '$protocol $status $bytes_sent $bytes_received '
                 '$session_time "$upstream_addr" '
                 '"$upstream_bytes_sent" "$upstream_bytes_received" '
                 '$upstream_connect_time';
```

The `escape` parameter (1.11.8) allows setting `json` or `default` characters escaping in variables, by default, `default` escaping is used. The `none` parameter (1.13.10) disables escaping.

For `default` escaping, characters “`"`”, “`\`”, and other characters with values less than 32 or above 126 are escaped as “`\xxx`”. If the variable value is not found, a hyphen (“`-`”) will be logged.

For `json` escaping, all characters not allowed in JSON `strings` will be escaped: characters “`"`” and “`\`” are escaped as “`\`” and “`\\`”, characters with values less than 32 are escaped as “`\n`”, “`\r`”, “`\t`”, “`\b`”, “`\f`”, or “`\u00XX`”.

## open\_log\_file\_cache

SYNTAX: **open\_log\_file\_cache** max=*N* [*inactive=time*] [*min\_uses=N*]  
[*valid=time*];

SYNTAX: **open\_log\_file\_cache** off;

DEFAULT off

CONTEXT: stream, server

Defines a cache that stores the file descriptors of frequently used logs whose names contain variables. The directive has the following parameters:

max

sets the maximum number of descriptors in a cache; if the cache becomes full the least recently used (LRU) descriptors are closed

inactive

sets the time after which the cached descriptor is closed if there were no access during this time; by default, 10 seconds

min\_uses

sets the minimum number of file uses during the time defined by the *inactive* parameter to let the descriptor stay open in a cache; by default, 1

valid

sets the time after which it should be checked that the file still exists with the same name; by default, 60 seconds

off

disables caching

Usage example:

```
open_log_file_cache max=1000 inactive=20s valid=1m min_uses=2;
```

## 3.9 Module ngx\_stream\_map\_module

3.9.1	Summary	423
3.9.2	Example Configuration	423
3.9.3	Directives	423
	map	423
	map_hash_bucket_size	424
	map_hash_max_size	425

### 3.9.1 Summary

The ngx\_stream\_map\_module module (1.11.2) creates variables whose values depend on values of other variables.

### 3.9.2 Example Configuration

```
map $remote_addr $limit {
    127.0.0.1      "";
    default        $binary_remote_addr;
}

limit_conn_zone $limit zone=addr:10m;
limit_conn addr 1;
```

### 3.9.3 Directives

#### map

SYNTAX: **map** *string* *\$variable* { ... }

DEFAULT —

CONTEXT: stream

Creates a new variable whose value depends on values of one or more of the source variables specified in the first parameter.

Since variables are evaluated only when they are used, the mere declaration even of a large number of “map” variables does not add any extra costs to connection processing.

Parameters inside the map block specify a mapping between source and resulting values.

Source values are specified as strings or regular expressions.

Strings are matched ignoring the case.

A regular expression should either start from the “~” symbol for a case-sensitive matching, or from the “~\*” symbols for case-insensitive matching. A regular expression can contain named and positional captures that can later be used in other directives along with the resulting variable.



If a source value matches one of the names of special parameters described below, it should be prefixed with the “\” symbol.

The resulting value can contain text, variable, and their combination.

The following special parameters are also supported:

`default` *value*

sets the resulting value if the source value matches none of the specified variants. When `default` is not specified, the default resulting value will be an empty string.

`hostnames`

indicates that source values can be hostnames with a prefix or suffix mask:

```
*.example.com 1;
example.*     1;
```

The following two records

```
example.com 1;
*.example.com 1;
```

can be combined:

```
.example.com 1;
```

This parameter should be specified before the list of values.

`include` *file*

includes a file with values. There can be several inclusions.

`volatile`

indicates that the variable is not cacheable (1.11.7).

If the source value matches more than one of the specified variants, e.g. both a mask and a regular expression match, the first matching variant will be chosen, in the following order of priority:

1. string value without a mask
2. longest string value with a prefix mask, e.g. “`*.example.com`”
3. longest string value with a suffix mask, e.g. “`mail.*`”
4. first matching regular expression (in order of appearance in a configuration file)
5. default value

## **map\_hash\_bucket\_size**

SYNTAX: **map\_hash\_bucket\_size** *size*;

DEFAULT 32 | 64 | 128

CONTEXT: stream

Sets the bucket size for the [map](#) variables hash tables. Default value depends on the processor's cache line size. The details of setting up hash tables are provided in a separate [document](#).

### **map\_hash\_max\_size**

SYNTAX: **map\_hash\_max\_size** *size*;

DEFAULT 2048

CONTEXT: stream

Sets the maximum *size* of the [map](#) variables hash tables. The details of setting up hash tables are provided in a separate [document](#).

## 3.10 Module ngx\_stream\_proxy\_module

3.10.1	Summary	426
3.10.2	Example Configuration	426
3.10.3	Directives	427
	proxy_bind	427
	proxy_buffer_size	427
	proxy_connect_timeout	428
	proxy_download_rate	428
	proxy_half_close	428
	proxy_next_upstream	428
	proxy_next_upstream_timeout	429
	proxy_next_upstream_tries	429
	proxy_pass	429
	proxy_protocol	429
	proxy_requests	430
	proxy_responses	430
	proxy_session_drop	430
	proxy_socket_keepalive	431
	proxy_ssl	431
	proxy_ssl_certificate	431
	proxy_ssl_certificate_key	431
	proxy_ssl_ciphers	431
	proxy_ssl_conf_command	432
	proxy_ssl_crl	432
	proxy_ssl_name	432
	proxy_ssl_password_file	432
	proxy_ssl_protocols	433
	proxy_ssl_server_name	433
	proxy_ssl_session_reuse	433
	proxy_ssl_trusted_certificate	433
	proxy_ssl_verify	433
	proxy_ssl_verify_depth	433
	proxy_timeout	434
	proxy_upload_rate	434

### 3.10.1 Summary

The ngx\_stream\_proxy\_module module (1.9.0) allows proxying data streams over TCP, UDP (1.9.13), and UNIX-domain sockets.

### 3.10.2 Example Configuration

```
server {
    listen 127.0.0.1:12345;
    proxy_pass 127.0.0.1:8080;
}
```

```
server {
    listen 12345;
    proxy_connect_timeout 1s;
    proxy_timeout 1m;
    proxy_pass example.com:12345;
}

server {
    listen 53 udp reuseport;
    proxy_timeout 20s;
    proxy_pass dns.example.com:53;
}

server {
    listen [::1]:12345;
    proxy_pass unix:/tmp/stream.socket;
}
```

### 3.10.3 Directives

#### proxy\_bind

SYNTAX: **proxy\_bind** *address* [transparent] | off;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.9.2.

Makes outgoing connections to a proxied server originate from the specified local IP *address*. Parameter value can contain variables (1.11.2). The special value `off` cancels the effect of the `proxy_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address.

The `transparent` parameter (1.11.0) allows outgoing connections to a proxied server originate from a non-local IP address, for example, from a real IP address of a client:

```
proxy_bind $remote_addr transparent;
```

In order for this parameter to work, it is usually necessary to run nginx worker processes with the [superuser](#) privileges. On Linux it is not required (1.13.8) as if the `transparent` parameter is specified, worker processes inherit the `CAP_NET_RAW` capability from the master process. It is also necessary to configure kernel routing table to intercept network traffic from the proxied server.

#### proxy\_buffer\_size

SYNTAX: **proxy\_buffer\_size** *size*;

DEFAULT 16k

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.9.4.

Sets the *size* of the buffer used for reading data from the proxied server. Also sets the *size* of the buffer used for reading data from the client.

### proxy\_connect\_timeout

SYNTAX: **proxy\_connect\_timeout** *time*;  
DEFAULT 60s  
CONTEXT: stream, server

Defines a timeout for establishing a connection with a proxied server.

### proxy\_download\_rate

SYNTAX: **proxy\_download\_rate** *rate*;  
DEFAULT 0  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 1.9.3.

Limits the speed of reading the data from the proxied server. The *rate* is specified in bytes per second. The zero value disables rate limiting. The limit is set per a connection, so if nginx simultaneously opens two connections to the proxied server, the overall rate will be twice as much as the specified limit.

Parameter value can contain variables (1.17.0). It may be useful in cases where rate should be limited depending on a certain condition:

```
map $slow $rate {  
    1      4k;  
    2      8k;  
}  
  
proxy_download_rate $rate;
```

### proxy\_half\_close

SYNTAX: **proxy\_half\_close** on | off;  
DEFAULT off  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 1.21.4.

Enables or disables closing each direction of a TCP connection independently (“TCP half-close”). If enabled, proxying over TCP will be kept until both sides close the connection.

### proxy\_next\_upstream

SYNTAX: **proxy\_next\_upstream** on | off;  
DEFAULT on  
CONTEXT: stream, server

When a connection to the proxied server cannot be established, determines whether a client connection will be passed to the next server.

Passing a connection to the next server can be limited by [the number of tries](#) and by [time](#).

### **proxy\_next\_upstream\_timeout**

SYNTAX: **proxy\_next\_upstream\_timeout** *time*;  
DEFAULT 0  
CONTEXT: stream, server

Limits the time allowed to pass a connection to the [next server](#). The 0 value turns off this limitation.

### **proxy\_next\_upstream\_tries**

SYNTAX: **proxy\_next\_upstream\_tries** *number*;  
DEFAULT 0  
CONTEXT: stream, server

Limits the number of possible tries for passing a connection to the [next server](#). The 0 value turns off this limitation.

### **proxy\_pass**

SYNTAX: **proxy\_pass** *address*;  
DEFAULT —  
CONTEXT: server

Sets the address of a proxied server. The address can be specified as a domain name or IP address, and a port:

```
proxy_pass localhost:12345;
```

or as a UNIX-domain socket path:

```
proxy_pass unix:/tmp/stream.socket;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a [server group](#).

The address can also be specified using variables (1.11.3):

```
proxy_pass $upstream;
```

In this case, the server name is searched among the described [server groups](#), and, if not found, is determined using a [resolver](#).

### **proxy\_protocol**

SYNTAX: **proxy\_protocol** on | off;  
DEFAULT off  
CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.9.2.

Enables the [PROXY protocol](#) for connections to a proxied server.

### proxy\_requests

SYNTAX: **proxy\_requests** *number*;

DEFAULT 0

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.15.7.

Sets the number of client datagrams at which binding between a client and existing UDP stream session is dropped. After receiving the specified number of datagrams, next datagram from the same client starts a new session. The session terminates when all client datagrams are transmitted to a proxied server and the expected number of [responses](#) is received, or when it reaches a [timeout](#).

### proxy\_responses

SYNTAX: **proxy\_responses** *number*;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.9.13.

Sets the number of datagrams expected from the proxied server in response to a client datagram if the UDP protocol is used. The number serves as a hint for session termination. By default, the number of datagrams is not limited.

If zero value is specified, no response is expected. However, if a response is received and the session is still not finished, the response will be handled.

### proxy\_session\_drop

SYNTAX: **proxy\_session\_drop** on | off;

DEFAULT off

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.15.8.

Enables terminating all sessions to a proxied server after it was removed from the group or marked as permanently unavailable. This can occur because of [re-resolve](#) or with the API [DELETE](#) command. A server can be marked as permanently unavailable if it is considered [unhealthy](#) or with the API [PATCH](#) command. Each session is terminated when the next read or write event is processed for the client or proxied server.

This directive is available as part of our [commercial subscription](#).

## proxy\_socket\_keepalive

SYNTAX: **proxy\_socket\_keepalive** on | off;  
DEFAULT off  
CONTEXT: stream, server  
THIS DIRECTIVE APPEARED IN VERSION 1.15.6.

Configures the “TCP keepalive” behavior for outgoing connections to a proxied server. By default, the operating system’s settings are in effect for the socket. If the directive is set to the value “on”, the `SO_KEEPALIVE` socket option is turned on for the socket.

## proxy\_ssl

SYNTAX: **proxy\_ssl** on | off;  
DEFAULT off  
CONTEXT: stream, server

Enables the SSL/TLS protocol for connections to a proxied server.

## proxy\_ssl\_certificate

SYNTAX: **proxy\_ssl\_certificate** *file*;  
DEFAULT —  
CONTEXT: stream, server

Specifies a *file* with the certificate in the PEM format used for authentication to a proxied server.

Since version 1.21.0, variables can be used in the *file* name.

## proxy\_ssl\_certificate\_key

SYNTAX: **proxy\_ssl\_certificate\_key** *file*;  
DEFAULT —  
CONTEXT: stream, server

Specifies a *file* with the secret key in the PEM format used for authentication to a proxied server.

Since version 1.21.0, variables can be used in the *file* name.

## proxy\_ssl\_ciphers

SYNTAX: **proxy\_ssl\_ciphers** *ciphers*;  
DEFAULT DEFAULT  
CONTEXT: stream, server

Specifies the enabled ciphers for connections to a proxied server. The ciphers are specified in the format understood by the OpenSSL library.

The full list can be viewed using the “`openssl ciphers`” command.



## proxy\_ssl\_conf\_command

SYNTAX: **proxy\_ssl\_conf\_command** *name value*;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

Sets arbitrary OpenSSL configuration [commands](#) when establishing a connection with the proxied server.

The directive is supported when using OpenSSL 1.0.2 or higher.

Several `proxy_ssl_conf_command` directives can be specified on the same level. These directives are inherited from the previous configuration level if and only if there are no `proxy_ssl_conf_command` directives defined on the current level.

Note that configuring OpenSSL directly might result in unexpected behavior.

## proxy\_ssl\_crl

SYNTAX: **proxy\_ssl\_crl** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with revoked certificates (CRL) in the PEM format used to [verify](#) the certificate of the proxied server.

## proxy\_ssl\_name

SYNTAX: **proxy\_ssl\_name** *name*;

DEFAULT host from proxy\_pass

CONTEXT: stream, server

Allows overriding the server name used to [verify](#) the certificate of the proxied server and to be [passed through SNI](#) when establishing a connection with the proxied server. The server name can also be specified using variables (1.11.3).

By default, the host part of the [proxy\\_pass](#) address is used.

## proxy\_ssl\_password\_file

SYNTAX: **proxy\_ssl\_password\_file** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with passphrases for [secret keys](#) where each passphrase is specified on a separate line. Passphrases are tried in turn when loading the key.

### proxy\_ssl\_protocols

SYNTAX: **proxy\_ssl\_protocols** [SSLv2] [SSLv3] [TLSv1] [TLSv1.1]  
[TLSv1.2] [TLSv1.3];  
DEFAULT TLSv1 TLSv1.1 TLSv1.2  
CONTEXT: stream, server

Enables the specified protocols for connections to a proxied server.

### proxy\_ssl\_server\_name

SYNTAX: **proxy\_ssl\_server\_name** on | off;  
DEFAULT off  
CONTEXT: stream, server

Enables or disables passing of the server name through [TLS Server Name Indication extension](#) (SNI, RFC 6066) when establishing a connection with the proxied server.

### proxy\_ssl\_session\_reuse

SYNTAX: **proxy\_ssl\_session\_reuse** on | off;  
DEFAULT on  
CONTEXT: stream, server

Determines whether SSL sessions can be reused when working with the proxied server. If the errors “SSL3\_GET\_FINISHED:digest check failed” appear in the logs, try disabling session reuse.

### proxy\_ssl\_trusted\_certificate

SYNTAX: **proxy\_ssl\_trusted\_certificate** *file*;  
DEFAULT —  
CONTEXT: stream, server

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) the certificate of the proxied server.

### proxy\_ssl\_verify

SYNTAX: **proxy\_ssl\_verify** on | off;  
DEFAULT off  
CONTEXT: stream, server

Enables or disables verification of the proxied server certificate.

### proxy\_ssl\_verify\_depth

SYNTAX: **proxy\_ssl\_verify\_depth** *number*;  
DEFAULT 1  
CONTEXT: stream, server

Sets the verification depth in the proxied server certificates chain.

### **proxy\_timeout**

SYNTAX: **proxy\_timeout** *timeout*;

DEFAULT 10m

CONTEXT: stream, server

Sets the *timeout* between two successive read or write operations on client or proxied server connections. If no data is transmitted within this time, the connection is closed.

### **proxy\_upload\_rate**

SYNTAX: **proxy\_upload\_rate** *rate*;

DEFAULT 0

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.9.3.

Limits the speed of reading the data from the client. The *rate* is specified in bytes per second. The zero value disables rate limiting. The limit is set per a connection, so if the client simultaneously opens two connections, the overall rate will be twice as much as the specified limit.

Parameter value can contain variables (1.17.0). It may be useful in cases where rate should be limited depending on a certain condition:

```
map $slow $rate {  
    1      4k;  
    2      8k;  
}  
  
proxy_upload_rate $rate;
```

## 3.11 Module ngx\_stream\_proxy\_protocol\_vendor\_module

3.11.1 Summary	435
3.11.2 Example Configuration	435
3.11.3 Embedded Variables	435

### 3.11.1 Summary

The `ngx_stream_proxy_protocol_vendor_module` module (1.23.3) allows obtaining additional information about a connection in cloud platforms from application-specific TLVs of the [PROXY protocol](#) header.

Supported cloud platforms:

- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure

The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the [listen](#) directive.

This module is available as part of our [commercial subscription](#).

### 3.11.2 Example Configuration

```
server {
    listen 12345 proxy_protocol;
    return $proxy_protocol_tlv_gcp_conn_id;
}
```

### 3.11.3 Embedded Variables

*\$proxy\_protocol\_tlv\_aws\_vpce\_id*

TLV value from the PROXY Protocol header representing the [ID of AWS VPC endpoint](#)

*\$proxy\_protocol\_tlv\_azure\_pel\_id*

TLV value from the PROXY Protocol header representing the [LinkID of Azure private endpoint](#)

*\$proxy\_protocol\_tlv\_gcp\_conn\_id*

TLV value from the PROXY Protocol header representing [Google Cloud PSC connection ID](#)

## 3.12 Module ngx\_stream\_realip\_module

3.12.1 Summary	436
3.12.2 Example Configuration	436
3.12.3 Directives	436
set_real_ip_from	436
3.12.4 Embedded Variables	436

### 3.12.1 Summary

The `ngx_stream_realip_module` module is used to change the client address and port to the ones sent in the PROXY protocol header (1.11.4). The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the `listen` directive.

This module is not built by default, it should be enabled with the `--with-stream_realip_module` configuration parameter.

### 3.12.2 Example Configuration

```
listen 12345 proxy_protocol;

set_real_ip_from 192.168.1.0/24;
set_real_ip_from 192.168.2.1;
set_real_ip_from 2001:0db8::/32;
```

### 3.12.3 Directives

#### set\_real\_ip\_from

SYNTAX: **set\_real\_ip\_from** *address* | *CIDR* | `unix::`;

DEFAULT —

CONTEXT: stream, server

Defines trusted addresses that are known to send correct replacement addresses. If the special value `unix:` is specified, all UNIX-domain sockets will be trusted.

### 3.12.4 Embedded Variables

*\$realip\_remote\_addr*

keeps the original client address

*\$realip\_remote\_port*

keeps the original client port

## 3.13 Module ngx\_stream\_return\_module

3.13.1 Summary	437
3.13.2 Example Configuration	437
3.13.3 Directives	437
return	437

### 3.13.1 Summary

The `ngx_stream_return_module` module (1.11.2) allows sending a specified value to the client and then closing the connection.

### 3.13.2 Example Configuration

```
server {
    listen 12345;
    return $time_iso8601;
}
```

### 3.13.3 Directives

#### return

SYNTAX: **return** *value*;

DEFAULT —

CONTEXT: server

Specifies a *value* to send to the client. The value can contain text, variables, and their combination.

## 3.14 Module ngx\_stream\_set\_module

3.14.1 Summary	438
3.14.2 Example Configuration	438
3.14.3 Directives	438
set	438

### 3.14.1 Summary

The ngx\_stream\_set\_module module (1.19.3) allows setting a value for a variable.

### 3.14.2 Example Configuration

```
server {
    listen 12345;
    set    $true 1;
}
```

### 3.14.3 Directives

#### set

SYNTAX: **set** *\$variable value*;

DEFAULT —

CONTEXT: server

Sets a *value* for the specified *variable*. The *value* can contain text, variables, and their combination.

## 3.15 Module ngx\_stream\_split\_clients\_module

3.15.1 Summary	439
3.15.2 Example Configuration	439
3.15.3 Directives	439
split_clients	439

### 3.15.1 Summary

The `ngx_stream_split_clients_module` module (1.11.3) creates variables suitable for A/B testing, also known as split testing.

### 3.15.2 Example Configuration

```
stream {
    ...
    split_clients "${remote_addr}AAA" $upstream {
        0.5%      feature_test1;
        2.0%      feature_test2;
        *         production;
    }

    server {
        ...
        proxy_pass $upstream;
    }
}
```

### 3.15.3 Directives

#### split\_clients

SYNTAX: **split\_clients** *string \$variable* { ... }

DEFAULT —

CONTEXT: stream

Creates a variable for A/B testing, for example:

```
split_clients "${remote_addr}AAA" $variant {
    0.5%      .one;
    2.0%      .two;
    *         "";
}
```

The value of the original string is hashed using MurmurHash2. In the example given, hash values from 0 to 21474835 (0.5%) correspond to the value `".one"` of the `$variant` variable, hash values from 21474836 to 107374180 (2%) correspond to the value `".two"`, and hash values from 107374181 to 4294967295 correspond to the value `""` (an empty string).



## 3.16 Module ngx\_stream\_ssl\_module

3.16.1 Summary	440
3.16.2 Example Configuration	440
3.16.3 Directives	441
ssl_alpn	441
ssl_certificate	441
ssl_certificate_key	442
ssl_ciphers	442
ssl_client_certificate	443
ssl_conf_command	443
ssl_crl	443
ssl_dhparam	444
ssl_ecdh_curve	444
ssl_handshake_timeout	444
ssl_password_file	444
ssl_prefer_server_ciphers	445
ssl_protocols	445
ssl_session_cache	445
ssl_session_ticket_key	446
ssl_session_tickets	447
ssl_session_timeout	447
ssl_trusted_certificate	447
ssl_verify_client	447
ssl_verify_depth	448
3.16.4 Embedded Variables	448

### 3.16.1 Summary

The `ngx_stream_ssl_module` module (1.9.0) provides the necessary support for a stream proxy server to work with the SSL/TLS protocol. This module is not built by default, it should be enabled with the `--with-stream_ssl_module` configuration parameter.

### 3.16.2 Example Configuration

To reduce the processor load, it is recommended to

- set the number of [worker processes](#) equal to the number of processors,
- enable the [shared](#) session cache,
- disable the [built-in](#) session cache,
- and possibly increase the session [lifetime](#) (by default, 5 minutes):

```
worker_processes auto;

stream {
    ...

    server {
        listen          12345 ssl;

        ssl_protocols   TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers      AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate   /usr/local/nginx/conf/cert.pem;
        ssl_certificate_key /usr/local/nginx/conf/cert.key;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 10m;

        ...
    }
}
```

### 3.16.3 Directives

#### ssl\_alpn

SYNTAX: **ssl\_alpn** *protocol* ...;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.21.4.

Specifies the list of supported [ALPN](#) protocols. One of the protocols must be [negotiated](#) if the client uses ALPN:

```
map $ssl_alpn_protocol $proxy {
    h2          127.0.0.1:8001;
    http/1.1    127.0.0.1:8002;
}

server {
    listen          12346;
    proxy_pass      $proxy;
    ssl_alpn        h2 http/1.1;
}
```

#### ssl\_certificate

SYNTAX: **ssl\_certificate** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with the certificate in the PEM format for the given server. If intermediate certificates should be specified in addition to a primary certificate, they should be specified in the same file in the following order: the primary certificate comes first, then the intermediate certificates. A secret key in the PEM format may be placed in the same file.

Since version 1.11.0, this directive can be specified multiple times to load certificates of different types, for example, RSA and ECDSA:

```
server {
    listen          12345 ssl;

    ssl_certificate  example.com.rsa.crt;
    ssl_certificate_key example.com.rsa.key;

    ssl_certificate  example.com.ecdsa.crt;
    ssl_certificate_key example.com.ecdsa.key;

    ...
}
```

Only OpenSSL 1.0.2 or higher supports separate certificate chains for different certificates. With older versions, only one certificate chain can be used.

Since version 1.15.9, variables can be used in the *file* name when using OpenSSL 1.0.2 or higher:

```
ssl_certificate      $ssl_server_name.crt;
ssl_certificate_key  $ssl_server_name.key;
```

Note that using variables implies that a certificate will be loaded for each SSL handshake, and this may have a negative impact on performance.

The value `data:$variable` can be specified instead of the *file* (1.15.10), which loads a certificate from a variable without using intermediate files. Note that inappropriate use of this syntax may have its security implications, such as writing secret key data to [error log](#).

### ssl\_certificate\_key

SYNTAX: **ssl\_certificate\_key** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with the secret key in the PEM format for the given server.

The value `engine:name:id` can be specified instead of the *file*, which loads a secret key with a specified *id* from the OpenSSL engine *name*.

The value `data:$variable` can be specified instead of the *file* (1.15.10), which loads a secret key from a variable without using intermediate files. Note that inappropriate use of this syntax may have its security implications, such as writing secret key data to [error log](#).

Since version 1.15.9, variables can be used in the *file* name when using OpenSSL 1.0.2 or higher.

### ssl\_ciphers

SYNTAX: **ssl\_ciphers** *ciphers*;

DEFAULT HIGH:!aNULL:!MD5

CONTEXT: stream, server

Specifies the enabled ciphers. The ciphers are specified in the format understood by the OpenSSL library, for example:

```
ssl_ciphers ALL:!aNULL:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
```

The full list can be viewed using the “`openssl ciphers`” command.

### ssl\_client\_certificate

SYNTAX: **ssl\_client\_certificate** *file*;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.11.8.

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) client certificates.

The list of certificates will be sent to clients. If this is not desired, the [ssl\\_trusted\\_certificate](#) directive can be used.

### ssl\_conf\_command

SYNTAX: **ssl\_conf\_command** *name value*;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

Sets arbitrary OpenSSL configuration [commands](#).

The directive is supported when using OpenSSL 1.0.2 or higher.

Several `ssl_conf_command` directives can be specified on the same level:

```
ssl_conf_command Options PrioritizeChaCha;  
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

These directives are inherited from the previous configuration level if and only if there are no `ssl_conf_command` directives defined on the current level.

Note that configuring OpenSSL directly might result in unexpected behavior.

### ssl\_crl

SYNTAX: **ssl\_crl** *file*;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.11.8.

Specifies a *file* with revoked certificates (CRL) in the PEM format used to [verify](#) client certificates.

## ssl\_dhparam

SYNTAX: **ssl\_dhparam** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with DH parameters for DHE ciphers.

By default no parameters are set, and therefore DHE ciphers will not be used.

Prior to version 1.11.0, builtin parameters were used by default.

## ssl\_ecdh\_curve

SYNTAX: **ssl\_ecdh\_curve** *curve*;

DEFAULT `auto`

CONTEXT: stream, server

Specifies a *curve* for ECDHE ciphers.

When using OpenSSL 1.0.2 or higher, it is possible to specify multiple curves (1.11.0), for example:

```
ssl_ecdh_curve prime256v1:secp384r1;
```

The special value `auto` (1.11.0) instructs nginx to use a list built into the OpenSSL library when using OpenSSL 1.0.2 or higher, or `prime256v1` with older versions.

Prior to version 1.11.0, the `prime256v1` curve was used by default.

When using OpenSSL 1.0.2 or higher, this directive sets the list of curves supported by the server. Thus, in order for ECDSA certificates to work, it is important to include the curves used in the certificates.

## ssl\_handshake\_timeout

SYNTAX: **ssl\_handshake\_timeout** *time*;

DEFAULT `60s`

CONTEXT: stream, server

Specifies a timeout for the SSL handshake to complete.

## ssl\_password\_file

SYNTAX: **ssl\_password\_file** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with passphrases for [secret keys](#) where each passphrase is specified on a separate line. Passphrases are tried in turn when loading the key.

Example:

```
stream {
    ssl_password_file /etc/keys/global.pass;
    ...

    server {
        listen 127.0.0.1:12345;
        ssl_certificate_key /etc/keys/first.key;
    }

    server {
        listen 127.0.0.1:12346;

        # named pipe can also be used instead of a file
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

### ssl\_prefer\_server\_ciphers

SYNTAX: **ssl\_prefer\_server\_ciphers** on | off;

DEFAULT off

CONTEXT: stream, server

Specifies that server ciphers should be preferred over client ciphers when the SSLv3 and TLS protocols are used.

### ssl\_protocols

SYNTAX: **ssl\_protocols** [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

DEFAULT TLSv1 TLSv1.1 TLSv1.2

CONTEXT: stream, server

Enables the specified protocols.

The TLSv1.1 and TLSv1.2 parameters work only when OpenSSL 1.0.1 or higher is used.

The TLSv1.3 parameter (1.13.0) works only when OpenSSL 1.1.1 or higher is used.

### ssl\_session\_cache

SYNTAX: **ssl\_session\_cache** off | none | [builtin[:size]] [shared:name:size];

DEFAULT none

CONTEXT: stream, server

Sets the types and sizes of caches that store session parameters. A cache can be of any of the following types:

`off`

the use of a session cache is strictly prohibited: nginx explicitly tells a client that sessions may not be reused.

`none`

the use of a session cache is gently disallowed: nginx tells a client that sessions may be reused, but does not actually store session parameters in the cache.

`builtin`

a cache built in OpenSSL; used by one worker process only. The cache size is specified in sessions. If size is not given, it is equal to 20480 sessions. Use of the built-in cache can cause memory fragmentation.

`shared`

a cache shared between all worker processes. The cache size is specified in bytes; one megabyte can store about 4000 sessions. Each shared cache should have an arbitrary name. A cache with the same name can be used in several servers.

Both cache types can be used simultaneously, for example:

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

but using only shared cache without the built-in cache should be more efficient.

### `ssl_session_ticket_key`

SYNTAX: **`ssl_session_ticket_key`** *file*;

DEFAULT —

CONTEXT: stream, server

Sets a *file* with the secret key used to encrypt and decrypt TLS session tickets. The directive is necessary if the same key has to be shared between multiple servers. By default, a randomly generated key is used.

If several keys are specified, only the first key is used to encrypt TLS session tickets. This allows configuring key rotation, for example:

```
ssl_session_ticket_key current.key;  
ssl_session_ticket_key previous.key;
```

The *file* must contain 80 or 48 bytes of random data and can be created using the following command:

```
openssl rand 80 > ticket.key
```

Depending on the file size either AES256 (for 80-byte keys, 1.11.8) or AES128 (for 48-byte keys) is used for encryption.

### ssl\_session\_tickets

SYNTAX: **ssl\_session\_tickets** on | off;

DEFAULT on

CONTEXT: stream, server

Enables or disables session resumption through [TLS session tickets](#).

### ssl\_session\_timeout

SYNTAX: **ssl\_session\_timeout** *time*;

DEFAULT 5m

CONTEXT: stream, server

Specifies a time during which a client may reuse the session parameters.

### ssl\_trusted\_certificate

SYNTAX: **ssl\_trusted\_certificate** *file*;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.11.8.

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) client certificates.

In contrast to the certificate set by [ssl\\_client\\_certificate](#), the list of these certificates will not be sent to clients.

### ssl\_verify\_client

SYNTAX: **ssl\_verify\_client** on | off | optional | optional\_no\_ca;

DEFAULT off

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.11.8.

Enables verification of client certificates. The verification result is stored in the [\\$ssl\\_client\\_verify](#) variable. If an error has occurred during the client certificate verification or a client has not presented the required certificate, the connection is closed.

The `optional` parameter requests the client certificate and verifies it if the certificate is present.

The `optional_no_ca` parameter requests the client certificate but does not require it to be signed by a trusted CA certificate. This is intended for the use in cases when a service that is external to nginx performs the actual certificate verification. The contents of the certificate is accessible through the [\\$ssl\\_client\\_cert](#) variable.



## ssl\_verify\_depth

SYNTAX: **ssl\_verify\_depth** *number*;

DEFAULT 1

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.11.8.

Sets the verification depth in the client certificates chain.

### 3.16.4 Embedded Variables

The `ngx_stream_ssl_module` module supports variables since 1.11.2.

#### *\$ssl\_alpn\_protocol*

returns the protocol selected by ALPN during the SSL handshake, or an empty string otherwise (1.21.4);

#### *\$ssl\_cipher*

returns the name of the cipher used for an established SSL connection;

#### *\$ssl\_ciphers*

returns the list of ciphers supported by the client (1.11.7). Known ciphers are listed by names, unknown are shown in hexadecimal, for example:

```
AES128-SHA:AES256-SHA:0x00ff
```

The variable is fully supported only when using OpenSSL version 1.0.2 or higher. With older versions, the variable is available only for new sessions and lists only known ciphers.

#### *\$ssl\_client\_cert*

returns the client certificate in the PEM format for an established SSL connection, with each line except the first prepended with the tab character (1.11.8);

#### *\$ssl\_client\_fingerprint*

returns the SHA1 fingerprint of the client certificate for an established SSL connection (1.11.8);

#### *\$ssl\_client\_i\_dn*

returns the “issuer DN” string of the client certificate for an established SSL connection according to [RFC 2253](#) (1.11.8);

#### *\$ssl\_client\_raw\_cert*

returns the client certificate in the PEM format for an established SSL connection (1.11.8);

#### *\$ssl\_client\_s\_dn*

returns the “subject DN” string of the client certificate for an established SSL connection according to [RFC 2253](#) (1.11.8);

#### *\$ssl\_client\_serial*

returns the serial number of the client certificate for an established SSL connection (1.11.8);

*\$ssl\_client\_v\_end*

returns the end date of the client certificate (1.11.8);

*\$ssl\_client\_v\_remain*

returns the number of days until the client certificate expires (1.11.8);

*\$ssl\_client\_v\_start*

returns the start date of the client certificate (1.11.8);

*\$ssl\_client\_verify*

returns the result of client certificate verification (1.11.8): “SUCCESS”, “FAILED:reason”, and “NONE” if a certificate was not present;

*\$ssl\_curve*

returns the negotiated curve used for SSL handshake key exchange process (1.21.5). Known curves are listed by names, unknown are shown in hexadecimal, for example:

```
prime256v1
```

The variable is supported only when using OpenSSL version 3.0 or higher. With older versions, the variable value will be an empty string.

*\$ssl\_curves*

returns the list of curves supported by the client (1.11.7). Known curves are listed by names, unknown are shown in hexadecimal, for example:

```
0x001d:prime256v1:secp521r1:secp384r1
```

The variable is supported only when using OpenSSL version 1.0.2 or higher. With older versions, the variable value will be an empty string.

The variable is available only for new sessions.

*\$ssl\_protocol*

returns the protocol of an established SSL connection;

*\$ssl\_server\_name*

returns the server name requested through [SNI](#);

*\$ssl\_session\_id*

returns the session identifier of an established SSL connection;

*\$ssl\_session\_reused*

returns “r” if an SSL session was reused, or “.” otherwise.

## 3.17 Module ngx\_stream\_ssl\_preread\_module

3.17.1 Summary	450
3.17.2 Example Configuration	450
3.17.3 Directives	451
ssl_preread	451
3.17.4 Embedded Variables	451

### 3.17.1 Summary

The `ngx_stream_ssl_preread_module` module (1.11.5) allows extracting information from the [ClientHello](#) message without terminating SSL/TLS, for example, the server name requested through [SNI](#) or protocols advertised in [ALPN](#). This module is not built by default, it should be enabled with the `--with-stream-ssl_preread_module` configuration parameter.

### 3.17.2 Example Configuration

Selecting an upstream based on server name:

```
map $ssl_preread_server_name $name {
    backend.example.com    backend;
    default                backend2;
}

upstream backend {
    server 192.168.0.1:12345;
    server 192.168.0.2:12345;
}

upstream backend2 {
    server 192.168.0.3:12345;
    server 192.168.0.4:12345;
}

server {
    listen      12346;
    proxy_pass  $name;
    ssl_preread on;
}
```

Selecting an upstream based on protocol:

```
map $ssl_preread_alpn_protocols $proxy {
    ~\bh2\b          127.0.0.1:8001;
    ~\bhttp/1.1\b    127.0.0.1:8002;
    ~\bxmlpp-client\b 127.0.0.1:8003;
}

server {
    listen      9000;
    proxy_pass  $proxy;
    ssl_preread on;
}
```

Selecting an upstream based on SSL protocol version:

```
map $ssl_preread_protocol $upstream {
    ""      ssh.example.com:22;
    "TLSv1.2" new.example.com:443;
    default  tls.example.com:443;
}

# ssh and https on the same port
server {
    listen      192.168.0.1:443;
    proxy_pass  $upstream;
    ssl_preread on;
}
```

### 3.17.3 Directives

#### ssl\_preread

SYNTAX: **ssl\_preread** on | off;

DEFAULT off

CONTEXT: stream, server

Enables extracting information from the ClientHello message at the preread phase.

### 3.17.4 Embedded Variables

*\$ssl\_preread\_protocol*

the highest SSL protocol version supported by the client (1.15.2)

*\$ssl\_preread\_server\_name*

server name requested through SNI

*\$ssl\_preread\_alpn\_protocols*

list of protocols advertised by the client through ALPN (1.13.10). The values are separated by commas.

## 3.18 Module ngx\_stream\_upstream\_module

3.18.1 Summary	452
3.18.2 Example Configuration	452
3.18.3 Directives	453
upstream	453
server	453
zone	455
state	456
hash	456
least_conn	457
least_time	457
random	457
resolver	458
resolver_timeout	459
3.18.4 Embedded Variables	459

### 3.18.1 Summary

The ngx\_stream\_upstream\_module module (1.9.0) is used to define groups of servers that can be referenced by the [proxy\\_pass](#) directive.

### 3.18.2 Example Configuration

```
upstream backend {
    hash $remote_addr consistent;

    server backend1.example.com:12345 weight=5;
    server backend2.example.com:12345;
    server unix:/tmp/backend3;

    server backup1.example.com:12345 backup;
    server backup2.example.com:12345 backup;
}

server {
    listen 12346;
    proxy_pass backend;
}
```

Dynamically configurable group with periodic [health checks](#) is available as part of our [commercial subscription](#):

```
resolver 10.0.0.1;

upstream dynamic {
    zone upstream_dynamic 64k;

    server backend1.example.com:12345 weight=5;
    server backend2.example.com:12345 fail_timeout=5s slow_start=30s;
    server 192.0.2.1:12345 max_fails=3;
    server backend3.example.com:12345 resolve;
    server backend4.example.com service=http resolve;

    server backup1.example.com:12345 backup;
}
```

```
server backup2.example.com:12345 backup;
}

server {
    listen 12346;
    proxy_pass dynamic;
    health_check;
}
```

### 3.18.3 Directives

#### upstream

SYNTAX: **upstream** *name* { ... }

DEFAULT —

CONTEXT: stream

Defines a group of servers. Servers can listen on different ports. In addition, servers listening on TCP and UNIX-domain sockets can be mixed.

Example:

```
upstream backend {
    server backend1.example.com:12345 weight=5;
    server 127.0.0.1:12345 max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend2;
    server backend3.example.com:12345 resolve;

    server backup1.example.com:12345 backup;
}
```

By default, connections are distributed between the servers using a weighted round-robin balancing method. In the above example, each 7 connections will be distributed as follows: 5 connections go to backend1.example.com:12345 and one connection to each of the second and third servers. If an error occurs during communication with a server, the connection will be passed to the next server, and so on until all of the functioning servers will be tried. If communication with all servers fails, the connection will be closed.

#### server

SYNTAX: **server** *address* [*parameters*];

DEFAULT —

CONTEXT: upstream

Defines the *address* and other *parameters* of a server. The address can be specified as a domain name or IP address with an obligatory port, or as a UNIX-domain socket path specified after the “unix:” prefix. A domain name that resolves to several IP addresses defines multiple servers at once.

The following parameters can be defined:

*weight=number*

sets the weight of the server, by default, 1.

`max_conns=number`

limits the maximum *number* of simultaneous connections to the proxied server (1.11.5). Default value is zero, meaning there is no limit. If the server group does not reside in the [shared memory](#), the limitation works per each worker process.

Prior to version 1.11.5, this parameter was available as part of our [commercial subscription](#).

`max_fails=number`

sets the number of unsuccessful attempts to communicate with the server that should happen in the duration set by the `fail_timeout` parameter to consider the server unavailable for a duration also set by the `fail_timeout` parameter. By default, the number of unsuccessful attempts is set to 1. The zero value disables the accounting of attempts. Here, an unsuccessful attempt is an error or timeout while establishing a connection with the server.

`fail_timeout=time`

sets

- the time during which the specified number of unsuccessful attempts to communicate with the server should happen to consider the server unavailable;
- and the period of time the server will be considered unavailable.

By default, the parameter is set to 10 seconds.

`backup`

marks the server as a backup server. Connections to the backup server will be passed when the primary servers are unavailable.

The parameter cannot be used along with the [hash](#) and [random](#) load balancing methods.

`down`

marks the server as permanently unavailable.

Additionally, the following parameters are available as part of our [commercial subscription](#):

`resolve`

monitors changes of the IP addresses that correspond to a domain name of the server, and automatically modifies the upstream configuration without the need of restarting nginx. The server group must reside in the [shared memory](#).

In order for this parameter to work, the `resolver` directive must be specified in the [stream](#) block or in the corresponding [upstream](#) block.

`service=name`

enables resolving of DNS [SRV](#) records and sets the service *name* (1.9.13). In order for this parameter to work, it is necessary to specify the [resolve](#) parameter for the server and specify a hostname without a port number. If the service name does not contain a dot (“.”), then the [RFC](#)-compliant name is constructed and the TCP protocol is added to the service prefix. For example, to look up the `_http._tcp.backend.example.com` SRV record, it is necessary to specify the directive:

```
server backend.example.com service=http resolve;
```

If the service name contains one or more dots, then the name is constructed by joining the service prefix and the server name. For example, to look up the `_http._tcp.backend.example.com` and `server1.backend.example.com` SRV records, it is necessary to specify the directives:

```
server backend.example.com service=_http._tcp resolve;  
server example.com service=server1.backend resolve;
```

Highest-priority SRV records (records with the same lowest-number priority value) are resolved as primary servers, the rest of SRV records are resolved as backup servers. If the [backup](#) parameter is specified for the server, high-priority SRV records are resolved as backup servers, the rest of SRV records are ignored.

`slow_start=time`

sets the *time* during which the server will recover its weight from zero to a nominal value, when unhealthy server becomes [healthy](#), or when the server becomes available after a period of time it was considered [unavailable](#). Default value is zero, i.e. slow start is disabled.

The parameter cannot be used along with the [hash](#) and [random](#) load balancing methods.

If there is only a single server in a group, `max_fails`, `fail_timeout` and `slow_start` parameters are ignored, and such a server will never be considered unavailable.

## zone

SYNTAX: **zone** *name* [*size*];

DEFAULT —

CONTEXT: upstream

Defines the *name* and *size* of the shared memory zone that keeps the group’s configuration and run-time state that are shared between worker processes. Several groups may share the same zone. In this case, it is enough to specify the *size* only once.



Additionally, as part of our [commercial subscription](#), such groups allow changing the group membership or modifying the settings of a particular server without the need of restarting nginx. The configuration is accessible via the [API](#) module (1.13.3).

Prior to version 1.13.3, the configuration was accessible only via a special location handled by [upstream\\_conf](#).

## state

SYNTAX: **state** *file*;

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.9.7.

Specifies a *file* that keeps the state of the dynamically configurable group. Examples:

```
state /var/lib/nginx/state/servers.conf; # path for Linux
state /var/db/nginx/state/servers.conf; # path for FreeBSD
```

The state is currently limited to the list of servers with their parameters. The file is read when parsing the configuration and is updated each time the upstream configuration is [changed](#). Changing the file content directly should be avoided. The directive cannot be used along with the [server](#) directive.

Changes made during [configuration reload](#) or [binary upgrade](#) can be lost.

This directive is available as part of our [commercial subscription](#).

## hash

SYNTAX: **hash** *key* [consistent];

DEFAULT —

CONTEXT: upstream

Specifies a load balancing method for a server group where the client-server mapping is based on the hashed *key* value. The *key* can contain text, variables, and their combinations (1.11.2). Usage example:

```
hash $remote_addr;
```

Note that adding or removing a server from the group may result in remapping most of the keys to different servers. The method is compatible with the [Cache::Memcached](#) Perl library.

If the *consistent* parameter is specified, the [ketama](#) consistent hashing method will be used instead. The method ensures that only a few keys will be remapped to different servers when a server is added to or removed from the

group. This helps to achieve a higher cache hit ratio for caching servers. The method is compatible with the [Cache::Memcached::Fast](#) Perl library with the *ketama\_points* parameter set to 160.

### least\_conn

SYNTAX: **least\_conn**;

DEFAULT —

CONTEXT: upstream

Specifies that a group should use a load balancing method where a connection is passed to the server with the least number of active connections, taking into account weights of servers. If there are several such servers, they are tried in turn using a weighted round-robin balancing method.

### least\_time

SYNTAX: **least\_time** connect | first\_byte | last\_byte [inflight];

DEFAULT —

CONTEXT: upstream

Specifies that a group should use a load balancing method where a connection is passed to the server with the least average time and least number of active connections, taking into account weights of servers. If there are several such servers, they are tried in turn using a weighted round-robin balancing method.

If the *connect* parameter is specified, time to [connect](#) to the upstream server is used. If the *first\_byte* parameter is specified, time to receive the [first byte](#) of data is used. If the *last\_byte* is specified, time to receive the [last byte](#) of data is used. If the *inflight* parameter is specified (1.11.6), incomplete connections are also taken into account.

Prior to version 1.11.6, incomplete connections were taken into account by default.

This directive is available as part of our [commercial subscription](#).

### random

SYNTAX: **random** [two [*method*]];

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.15.1.

Specifies that a group should use a load balancing method where a connection is passed to a randomly selected server, taking into account weights of servers.

The optional *two* parameter instructs nginx to randomly select [two](#) servers and then choose a server using the specified *method*. The default method is

`least_conn` which passes a connection to a server with the least number of active connections.

The `least_time` method passes a connection to a server with the least average time and least number of active connections. If `least_time=connect` parameter is specified, time to [connect](#) to the upstream server is used. If `least_time=first_byte` parameter is specified, time to receive the [first byte](#) of data is used. If `least_time=last_byte` is specified, time to receive the [last byte](#) of data is used.

The `least_time` method is available as a part of our [commercial subscription](#).

## resolver

SYNTAX: **resolver** *address* ... [*valid=time*] [*ipv4=on|off*] [*ipv6=on|off*]  
[*status\_zone=zone*];

DEFAULT —

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.17.5.

Configures name servers used to resolve names of upstream servers into addresses, for example:

```
resolver 127.0.0.1 [::1]:5353;
```

The address can be specified as a domain name or IP address, with an optional port. If port is not specified, the port 53 is used. Name servers are queried in a round-robin fashion.

By default, nginx will look up both IPv4 and IPv6 addresses while resolving. If looking up of IPv4 or IPv6 addresses is not desired, the `ipv4=off` (1.23.1) or the `ipv6=off` parameter can be specified.

By default, nginx caches answers using the TTL value of a response. The optional `valid` parameter allows overriding it:

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

To prevent DNS spoofing, it is recommended configuring DNS servers in a properly secured trusted local network.

The optional `status_zone` parameter enables [collection](#) of DNS server statistics of requests and responses in the specified *zone*.

This directive is available as part of our [commercial subscription](#).

## resolver\_timeout

SYNTAX: **resolver\_timeout** *time*;

DEFAULT 30s

CONTEXT: upstream

THIS DIRECTIVE APPEARED IN VERSION 1.17.5.

Sets a timeout for name resolution, for example:

```
resolver_timeout 5s;
```

This directive is available as part of our [commercial subscription](#).

### 3.18.4 Embedded Variables

The `ngx_stream_upstream_module` module supports the following embedded variables:

#### *\$upstream\_addr*

keeps the IP address and port, or the path to the UNIX-domain socket of the upstream server (1.11.4). If several servers were contacted during proxying, their addresses are separated by commas, e.g. “192.168.1.1:12345, 192.168.1.2:12345, unix:/tmp/sock”. If a server cannot be selected, the variable keeps the name of the server group.

#### *\$upstream\_bytes\_received*

number of bytes received from an upstream server (1.11.4). Values from several connections are separated by commas like addresses in the [\\$upstream\\_addr](#) variable.

#### *\$upstream\_bytes\_sent*

number of bytes sent to an upstream server (1.11.4). Values from several connections are separated by commas like addresses in the [\\$upstream\\_addr](#) variable.

#### *\$upstream\_connect\_time*

time to connect to the upstream server (1.11.4); the time is kept in seconds with millisecond resolution. Times of several connections are separated by commas like addresses in the [\\$upstream\\_addr](#) variable.

#### *\$upstream\_first\_byte\_time*

time to receive the first byte of data (1.11.4); the time is kept in seconds with millisecond resolution. Times of several connections are separated by commas like addresses in the [\\$upstream\\_addr](#) variable.

#### *\$upstream\_session\_time*

session duration in seconds with millisecond resolution (1.11.4). Times of several connections are separated by commas like addresses in the [\\$upstream\\_addr](#) variable.

## 3.19 Module ngx\_stream\_upstream\_hc\_module

3.19.1 Summary	460
3.19.2 Example Configuration	460
3.19.3 Directives	461
health_check	461
health_check_timeout	462
match	462

### 3.19.1 Summary

The `ngx_stream_upstream_hc_module` module (1.9.0) allows enabling periodic health checks of the servers in a [group](#). The server group must reside in the [shared memory](#).

If a health check fails, the server will be considered unhealthy. If several health checks are defined for the same group of servers, a single failure of any check will make the corresponding server be considered unhealthy. Client connections are not passed to unhealthy servers and servers in the “checking” state.

This module is available as part of our [commercial subscription](#).

### 3.19.2 Example Configuration

```
upstream tcp {
    zone upstream_tcp 64k;

    server backend1.example.com:12345 weight=5;
    server backend2.example.com:12345 fail_timeout=5s slow_start=30s;
    server 192.0.2.1:12345 max_fails=3;

    server backup1.example.com:12345 backup;
    server backup2.example.com:12345 backup;
}

server {
    listen 12346;
    proxy_pass tcp;
    health_check;
}
```

With this configuration, nginx will check the ability to establish a TCP connection to each server in the `tcp` group every five seconds. When a connection to the server cannot be established, the health check will fail, and the server will be considered unhealthy.

Health checks can be configured for the UDP protocol:

```
upstream dns_upstream {
```

```
zone    dns_zone 64k;

server dns1.example.com:53;
server dns2.example.com:53;
server dns3.example.com:53;
}

server {
    listen      53 udp;
    proxy_pass  dns_upstream;
    health_check udp;
}
```

In this case, the absence of ICMP “Destination Unreachable” message is expected in reply to the sent string “nginx health check”.

Health checks can also be configured to test data obtained from the server. Tests are configured separately using the [match](#) directive and referenced in the match parameter of the [health\\_check](#) directive.

### 3.19.3 Directives

#### health\_check

SYNTAX: **health\_check** [*parameters*];

DEFAULT —

CONTEXT: server

Enables periodic health checks of the servers in a [group](#).

The following optional parameters are supported:

*interval=time*

sets the interval between two consecutive health checks, by default, 5 seconds.

*jitter=time*

sets the time within which each health check will be randomly delayed, by default, there is no delay.

*fails=number*

sets the number of consecutive failed health checks of a particular server after which this server will be considered unhealthy, by default, 1.

*passes=number*

sets the number of consecutive passed health checks of a particular server after which the server will be considered healthy, by default, 1.

*mandatory* [*persistent*]

sets the initial “checking” state for a server until the first health check is completed (1.11.7). Client connections are not passed to servers in the “checking” state. If the parameter is not specified, the server will be initially considered healthy.

The *persistent* parameter (1.21.1) sets the initial “up” state for a server after reload if the server was considered healthy before reload.

*match=name*

specifies the *match* block configuring the tests that a successful connection should pass in order for a health check to pass. By default,

for TCP, only the ability to establish a TCP connection with the server is checked. For UDP, the absence of ICMP “Destination Unreachable” message is expected in reply to the sent string “nginx health check”.

Prior to version 1.11.7, by default, UDP health check required a match block with the `send` and `expect` parameters.

`port=number`

defines the port used when connecting to a server to perform a health check (1.9.7). By default, equals the `server` port.

`udp`

specifies that the UDP protocol should be used for health checks instead of the default TCP protocol (1.9.13).

### health\_check\_timeout

SYNTAX: **health\_check\_timeout** *timeout*;

DEFAULT 5s

CONTEXT: stream, server

Overrides the `proxy_timeout` value for health checks.

### match

SYNTAX: **match** *name* { ... }

DEFAULT —

CONTEXT: stream

Defines the named test set used to verify server responses to health checks. The following parameters can be configured:

`send string`;

sends a *string* to the server;

`expect string | ~ regex`;

a literal string (1.9.12) or a regular expression that the data obtained from the server should match. The regular expression is specified with the preceding “~\*” modifier (for case-insensitive matching), or the “~” modifier (for case-sensitive matching).

Both `send` and `expect` parameters can contain hexadecimal literals with the prefix “\x” followed by two hex digits, for example, “\x80” (1.9.12).

Health check is passed if:

- the TCP connection was successfully established;
- the *string* from the `send` parameter, if specified, was sent;
- the data obtained from the server matched the string or regular expression from the `expect` parameter, if specified;

- the time elapsed does not exceed the value specified in the [health\\_check\\_timeout](#) directive.

Example:

```
upstream backend {
    zone    upstream_backend 10m;
    server  127.0.0.1:12345;
}

match http {
    send      "GET / HTTP/1.0\r\nHost: localhost\r\n\r\n";
    expect ~  "200 OK";
}

server {
    listen      12346;
    proxy_pass  backend;
    health_check match=http;
}
```

Only the first [proxy\\_buffer\\_size](#) bytes of data obtained from the server are examined.



## 3.20 Module ngx\_stream\_zone\_sync\_module

3.20.1	Summary	464
3.20.2	Example Configuration	464
3.20.3	Directives	466
	zone_sync	466
	zone_sync_buffers	466
	zone_sync_connect_retry_interval	466
	zone_sync_connect_timeout	466
	zone_sync_interval	466
	zone_sync_recv_buffer_size	467
	zone_sync_server	467
	zone_sync_ssl	467
	zone_sync_ssl_certificate	468
	zone_sync_ssl_certificate_key	468
	zone_sync_ssl_ciphers	468
	zone_sync_ssl_conf_command	468
	zone_sync_ssl_crl	469
	zone_sync_ssl_name	469
	zone_sync_ssl_password_file	469
	zone_sync_ssl_protocols	469
	zone_sync_ssl_server_name	469
	zone_sync_ssl_trusted_certificate	470
	zone_sync_ssl_verify	470
	zone_sync_ssl_verify_depth	470
	zone_sync_timeout	470
3.20.4	API endpoints	470
3.20.5	Starting, stopping, removing a cluster node	470

### 3.20.1 Summary

The ngx\_stream\_zone\_sync\_module module (1.13.8) provides the necessary support for synchronizing contents of [shared memory zones](#) between nodes of a cluster. To enable synchronization for a particular zone, a corresponding module must support this feature. Currently, it is possible to synchronize HTTP [sticky](#) sessions, information about [excessive HTTP requests](#), and key-value pairs both in [http](#) and [stream](#).

This module is available as part of our [commercial subscription](#).

### 3.20.2 Example Configuration

Minimal configuration:

```
http {
    ...

    upstream backend {
        server backend1.example.com:8080;
        server backend2.example.com:8081;

        sticky learn
            create=$upstream_cookie_examplecookie
            lookup=$cookie_examplecookie
            zone=client_sessions:1m sync;
    }

    ...
}

stream {
    ...

    server {
        zone_sync;

        listen 127.0.0.1:12345;

        # cluster of 2 nodes
        zone_sync_server a.example.com:12345;
        zone_sync_server b.example.com:12345;
    }
}
```

A more complex configuration with SSL enabled and with cluster members defined by DNS:

```
...

stream {
    ...

    resolver 127.0.0.1 valid=10s;

    server {
        zone_sync;

        # the name resolves to multiple addresses that correspond to cluster
        # nodes
        zone_sync_server cluster.example.com:12345 resolve;

        listen 127.0.0.1:4433 ssl;

        ssl_certificate      localhost.crt;
        ssl_certificate_key  localhost.key;

        zone_sync_ssl on;

        zone_sync_ssl_certificate      localhost.crt;
        zone_sync_ssl_certificate_key  localhost.key;
    }
}
```

### 3.20.3 Directives

#### zone\_sync

SYNTAX: **zone\_sync**;

DEFAULT —

CONTEXT: server

Enables the synchronization of shared memory zones between cluster nodes. Cluster nodes are defined using [zone\\_sync\\_server](#) directives.

#### zone\_sync\_buffers

SYNTAX: **zone\_sync\_buffers** *number size*;

DEFAULT 8 4k | 8k

CONTEXT: stream, server

Sets the *number* and *size* of the per-zone buffers used for pushing zone contents. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

A single buffer must be large enough to hold any entry of each zone being synchronized.

#### zone\_sync\_connect\_retry\_interval

SYNTAX: **zone\_sync\_connect\_retry\_interval** *time*;

DEFAULT 1s

CONTEXT: stream, server

Defines an interval between connection attempts to another cluster node.

#### zone\_sync\_connect\_timeout

SYNTAX: **zone\_sync\_connect\_timeout** *time*;

DEFAULT 5s

CONTEXT: stream, server

Defines a timeout for establishing a connection with another cluster node.

#### zone\_sync\_interval

SYNTAX: **zone\_sync\_interval** *time*;

DEFAULT 1s

CONTEXT: stream, server

Defines an interval for polling updates in a shared memory zone.

### zone\_sync\_recv\_buffer\_size

SYNTAX: **zone\_sync\_recv\_buffer\_size** *size*;

DEFAULT 4k | 8k

CONTEXT: stream, server

Sets *size* of a per-connection receive buffer used to parse incoming stream of synchronization messages. The buffer size must be equal or greater than one of the [zone\\_sync\\_buffers](#). By default, the buffer size is equal to [zone\\_sync\\_buffers](#) *size* multiplied by *number*.

### zone\_sync\_server

SYNTAX: **zone\_sync\_server** *address* [resolve];

DEFAULT —

CONTEXT: server

Defines the *address* of a cluster node. The address can be specified as a domain name or IP address with a mandatory port, or as a UNIX-domain socket path specified after the “unix:” prefix. A domain name that resolves to several IP addresses defines multiple nodes at once.

The `resolve` parameter instructs nginx to monitor changes of the IP addresses that correspond to a domain name of the node and automatically modify the configuration without the need of restarting nginx.

Cluster nodes are specified either dynamically as a single `zone_sync_server` directive with the `resolve` parameter, or statically as a series of several directives without the parameter.

Each cluster node should be specified only once.

All cluster nodes should use the same configuration.

In order for the `resolve` parameter to work, the [resolver](#) directive must be specified in the [stream](#) block. Example:

```
stream {
    resolver 10.0.0.1;

    server {
        zone_sync;
        zone_sync_server cluster.example.com:12345 resolve;
        ...
    }
}
```

### zone\_sync\_ssl

SYNTAX: **zone\_sync\_ssl** on | off;

DEFAULT off

CONTEXT: stream, server

Enables the SSL/TLS protocol for connections to another cluster server.

### zone\_sync\_ssl\_certificate

SYNTAX: **zone\_sync\_ssl\_certificate** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with the certificate in the PEM format used for authentication to another cluster server.

### zone\_sync\_ssl\_certificate\_key

SYNTAX: **zone\_sync\_ssl\_certificate\_key** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with the secret key in the PEM format used for authentication to another cluster server.

### zone\_sync\_ssl\_ciphers

SYNTAX: **zone\_sync\_ssl\_ciphers** *ciphers*;

DEFAULT DEFAULT

CONTEXT: stream, server

Specifies the enabled ciphers for connections to another cluster server. The ciphers are specified in the format understood by the OpenSSL library.

The full list can be viewed using the “`openssl ciphers`” command.

### zone\_sync\_ssl\_conf\_command

SYNTAX: **zone\_sync\_ssl\_conf\_command** *name value*;

DEFAULT —

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

Sets arbitrary OpenSSL configuration [commands](#) when establishing a connection with another cluster server.

The directive is supported when using OpenSSL 1.0.2 or higher.

Several `zone_sync_ssl_conf_command` directives can be specified on the same level. These directives are inherited from the previous configuration level if and only if there are no `zone_sync_ssl_conf_command` directives defined on the current level.

Note that configuring OpenSSL directly might result in unexpected behavior.

### zone\_sync\_ssl\_crl

SYNTAX: **zone\_sync\_ssl\_crl** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with revoked certificates (CRL) in the PEM format used to [verify](#) the certificate of another cluster server.

### zone\_sync\_ssl\_name

SYNTAX: **zone\_sync\_ssl\_name** *name*;

DEFAULT host from zone\_sync\_server

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.15.7.

Allows overriding the server name used to [verify](#) the certificate of a cluster server and to be [passed through SNI](#) when establishing a connection with the cluster server.

By default, the host part of the [zone\\_sync\\_server](#) address is used, or resolved IP address if the resolve parameter is specified.

### zone\_sync\_ssl\_password\_file

SYNTAX: **zone\_sync\_ssl\_password\_file** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with passphrases for [secret keys](#) where each passphrase is specified on a separate line. Passphrases are tried in turn when loading the key.

### zone\_sync\_ssl\_protocols

SYNTAX: **zone\_sync\_ssl\_protocols** [SSLv2] [SSLv3] [TLSv1] [TLSv1.1]  
[TLSv1.2] [TLSv1.3];

DEFAULT TLSv1 TLSv1.1 TLSv1.2

CONTEXT: stream, server

Enables the specified protocols for connections to another cluster server.

### zone\_sync\_ssl\_server\_name

SYNTAX: **zone\_sync\_ssl\_server\_name** on | off;

DEFAULT off

CONTEXT: stream, server

THIS DIRECTIVE APPEARED IN VERSION 1.15.7.

Enables or disables passing of the server name through [TLS Server Name Indication extension](#) (SNI, RFC 6066) when establishing a connection with another cluster server.

### zone\_sync\_ssl\_trusted\_certificate

SYNTAX: **zone\_sync\_ssl\_trusted\_certificate** *file*;

DEFAULT —

CONTEXT: stream, server

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) the certificate of another cluster server.

### zone\_sync\_ssl\_verify

SYNTAX: **zone\_sync\_ssl\_verify** on | off;

DEFAULT off

CONTEXT: stream, server

Enables or disables verification of another cluster server certificate.

### zone\_sync\_ssl\_verify\_depth

SYNTAX: **zone\_sync\_ssl\_verify\_depth** *number*;

DEFAULT 1

CONTEXT: stream, server

Sets the verification depth in another cluster server certificates chain.

### zone\_sync\_timeout

SYNTAX: **zone\_sync\_timeout** *timeout*;

DEFAULT 5s

CONTEXT: stream, server

Sets the *timeout* between two successive read or write operations on connection to another cluster node. If no data is transmitted within this time, the connection is closed.

## 3.20.4 API endpoints

The synchronization status of a node is available via the [/stream/zone-sync/](#) endpoint of the API which returns the [following](#) metrics.

## 3.20.5 Starting, stopping, removing a cluster node

To start a new node, update a DNS record of a cluster hostname with the IP address of the new node and start an instance. The new node will discover other nodes from DNS or static configuration and will start sending updates to them. Other nodes will eventually discover the new node using DNS and start pushing updates to it. In case of static configuration, other nodes need to be reloaded in order to send updates to the new node.

To stop a node, send the QUIT signal to the instance. The node will finish zone synchronization and gracefully close open connections.

To remove a node, update a DNS record of a cluster hostname and remove the IP address of the node. All other nodes will eventually discover that the node is removed, close connections to the node, and will no longer try to connect to it. After the node is removed, it can be stopped as described above. In case of static configuration, other nodes need to be reloaded in order to stop sending updates to the removed node.



# Chapter 4

## Mail server modules

### 4.1 Module ngx\_mail\_core\_module

4.1.1	Summary	472
4.1.2	Example Configuration	472
4.1.3	Directives	473
	listen	473
	mail	475
	max_errors	475
	protocol	475
	resolver	475
	resolver_timeout	476
	server	477
	server_name	477
	timeout	477

#### 4.1.1 Summary

This module is not built by default, it should be enabled with the `--with-mail` configuration parameter.

#### 4.1.2 Example Configuration

```
worker_processes auto;

error_log /var/log/nginx/error.log info;

events {
    worker_connections 1024;
}

mail {
    server_name      mail.example.com;
    auth_http        localhost:9000/cgi-bin/nginxauth.cgi;

    imap_capabilities IMAP4rev1 UIDPLUS IDLE LITERAL+ QUOTA;

    pop3_auth        plain apop cram-md5;
    pop3_capabilities LAST TOP USER PIPELINING UIDL;
```

```

smtp_auth      login plain cram-md5;
smtp_capabilities "SIZE 10485760" ENHANCEDSTATUSCODES 8BITMIME DSN;
xclient        off;

server {
    listen 25;
    protocol smtp;
}
server {
    listen 110;
    protocol pop3;
    proxy_pass_error_message on;
}
server {
    listen 143;
    protocol imap;
}
server {
    listen 587;
    protocol smtp;
}
}

```

### 4.1.3 Directives

#### listen

SYNTAX: **listen** *address:port* [ssl] [proxy\_protocol] [backlog=*number*]  
 [rcvbuf=*size*] [sndbuf=*size*] [bind] [ipv6only=on|off]  
 [so\_keepalive=on|off][*keepidle*]:[*keepintvl*]:[*keepcnt*];

DEFAULT —

CONTEXT: server

Sets the *address* and *port* for the socket on which the server will accept requests. It is possible to specify just the port. The address can also be a hostname, for example:

```

listen 127.0.0.1:110;
listen *:110;
listen 110;      # same as *:110
listen localhost:110;

```

IPv6 addresses (0.7.58) are specified in square brackets:

```

listen [::1]:110;
listen [::]:110;

```

UNIX-domain sockets (1.3.5) are specified with the “unix:” prefix:

```

listen unix:/var/run/nginx.sock;

```

Different servers must listen on different *address:port* pairs.

The `ssl` parameter allows specifying that all connections accepted on this port should work in SSL mode.

The `proxy_protocol` parameter (1.19.8) allows specifying that all connections accepted on this port should use the [PROXY protocol](#). Obtained

information is passed to the authentication server and can be used to [change the client address](#).

The `listen` directive can have several additional parameters specific to socket-related system calls.

`backlog=number`

sets the `backlog` parameter in the `listen` call that limits the maximum length for the queue of pending connections (1.9.2). By default, `backlog` is set to -1 on FreeBSD, DragonFly BSD, and macOS, and to 511 on other platforms.

`rcvbuf=size`

sets the receive buffer size (the `SO_RCVBUF` option) for the listening socket (1.11.13).

`sndbuf=size`

sets the send buffer size (the `SO_SNDBUF` option) for the listening socket (1.11.13).

`bind`

this parameter instructs to make a separate `bind` call for a given address:port pair. The fact is that if there are several `listen` directives with the same port but different addresses, and one of the `listen` directives listens on all addresses for the given port (`*:port`), nginx will `bind` only to `*:port`. It should be noted that the `getsockname` system call will be made in this case to determine the address that accepted the connection. If the `backlog`, `rcvbuf`, `sndbuf`, `ipv6only`, or `so_keepalive` parameters are used then for a given `address:port` pair a separate `bind` call will always be made.

`ipv6only=on|off`

this parameter determines (via the `IPV6_V6ONLY` socket option) whether an IPv6 socket listening on a wildcard address `:::` will accept only IPv6 connections or both IPv6 and IPv4 connections. This parameter is turned on by default. It can only be set once on start.

`so_keepalive=on|off[[keepidle]:[keepintvl]:[keepcnt]`

this parameter configures the “TCP keepalive” behavior for the listening socket. If this parameter is omitted then the operating system’s settings will be in effect for the socket. If it is set to the value “on”, the `SO_KEEPALIVE` option is turned on for the socket. If it is set to the value “off”, the `SO_KEEPALIVE` option is turned off for the socket. Some operating systems support setting of TCP keepalive parameters on a per-socket basis using the `TCP_KEEPIDLE`, `TCP_KEEPINTVL`, and `TCP_KEEPCNT` socket options. On such systems (currently, Linux 2.4+, NetBSD 5+, and FreeBSD 9.0-STABLE), they can be configured using the `keepidle`, `keepintvl`, and `keepcnt` parameters. One or two parameters may be omitted, in which case the system default setting for the corresponding socket option will be in effect. For example,

```
so_keepalive=30m::10
```

will set the idle timeout (TCP\_KEEPIIDLE) to 30 minutes, leave the probe interval (TCP\_KEEPIINTVL) at its system default, and set the probes count (TCP\_KEEPCNT) to 10 probes.

## mail

SYNTAX: **mail** { ... }

DEFAULT —

CONTEXT: main

Provides the configuration file context in which the mail server directives are specified.

## max\_errors

SYNTAX: **max\_errors** *number*;

DEFAULT 5

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.21.0.

Sets the number of protocol errors after which the connection is closed.

## protocol

SYNTAX: **protocol** imap | pop3 | smtp;

DEFAULT —

CONTEXT: server

Sets the protocol for a proxied server. Supported protocols are [IMAP](#), [POP3](#), and [SMTP](#).

If the directive is not set, the protocol can be detected automatically based on the well-known port specified in the [listen](#) directive:

- imap: 143, 993
- pop3: 110, 995
- smtp: 25, 587, 465

Unnecessary protocols can be disabled using the [configuration](#) parameters `--without-mail_imap_module`, `--without-mail_pop3_module`, and `--without-mail_smtp_module`.

## resolver

SYNTAX: **resolver** *address* ... [valid=*time*] [ipv4=on|off] [ipv6=on|off]  
[status\_zone=*zone*];

SYNTAX: **resolver** off;

DEFAULT off

CONTEXT: mail, server

Configures name servers used to find the client's hostname to pass it to the [authentication server](#), and in the [XCLIENT](#) command when proxying SMTP. For example:

```
resolver 127.0.0.1 [::1]:5353;
```

The address can be specified as a domain name or IP address, with an optional port (1.3.1, 1.2.2). If port is not specified, the port 53 is used. Name servers are queried in a round-robin fashion.

Before version 1.1.7, only a single name server could be configured. Specifying name servers using IPv6 addresses is supported starting from versions 1.3.1 and 1.2.2.

By default, nginx will look up both IPv4 and IPv6 addresses while resolving. If looking up of IPv4 or IPv6 addresses is not desired, the `ipv4=off` (1.23.1) or the `ipv6=off` parameter can be specified.

Resolving of names into IPv6 addresses is supported starting from version 1.5.8.

By default, nginx caches answers using the TTL value of a response. An optional `valid` parameter allows overriding it:

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

Before version 1.1.9, tuning of caching time was not possible, and nginx always cached answers for the duration of 5 minutes.

To prevent DNS spoofing, it is recommended configuring DNS servers in a properly secured trusted local network.

The optional `status_zone` parameter (1.17.1) enables [collection](#) of DNS server statistics of requests and responses in the specified *zone*. The parameter is available as part of our [commercial subscription](#).

The special value `off` disables resolving.

### **resolver\_timeout**

SYNTAX: **resolver\_timeout** *time*;

DEFAULT 30s

CONTEXT: mail, server

Sets a timeout for DNS operations, for example:

```
resolver_timeout 5s;
```

**server**SYNTAX: **server** { ... }

DEFAULT —

CONTEXT: mail

Sets the configuration for a server.

**server\_name**SYNTAX: **server\_name** *name*;DEFAULT `hostname`

CONTEXT: mail, server

Sets the server name that is used:

- in the initial POP3/SMTP server greeting;
- in the salt during the SASL CRAM-MD5 authentication;
- in the EHLO command when connecting to the SMTP backend, if the passing of the [XCLIENT](#) command is enabled.

If the directive is not specified, the machine's hostname is used.

**timeout**SYNTAX: **timeout** *time*;DEFAULT `60s`

CONTEXT: mail, server

Sets the timeout that is used before proxying to the backend starts.

## 4.2 Module ngx\_mail\_auth\_http\_module

4.2.1 Directives . . . . .	478
auth_http . . . . .	478
auth_http_header . . . . .	478
auth_http_pass_client_cert . . . . .	478
auth_http_timeout . . . . .	478
4.2.2 Protocol . . . . .	479

### 4.2.1 Directives

#### auth\_http

SYNTAX: **auth\_http** *URL*;

DEFAULT —

CONTEXT: mail, server

Sets the URL of the HTTP authentication server. The protocol is described [below](#).

#### auth\_http\_header

SYNTAX: **auth\_http\_header** *header value*;

DEFAULT —

CONTEXT: mail, server

Appends the specified header to requests sent to the authentication server. This header can be used as the shared secret to verify that the request comes from nginx. For example:

```
auth_http_header X-Auth-Key "secret_string";
```

#### auth\_http\_pass\_client\_cert

SYNTAX: **auth\_http\_pass\_client\_cert** on | off;

DEFAULT off

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Appends the Auth-SSL-Cert header with the [client](#) certificate in the PEM format (urlencoded) to requests sent to the authentication server.

#### auth\_http\_timeout

SYNTAX: **auth\_http\_timeout** *time*;

DEFAULT 60s

CONTEXT: mail, server

Sets the timeout for communication with the authentication server.

## 4.2.2 Protocol

The HTTP protocol is used to communicate with the authentication server. The data in the response body is ignored, the information is passed only in the headers.

Examples of requests and responses:

Request:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: plain # plain/apop/cram-md5/external
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap # imap/pop3/smtp
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
```

Good response:

```
HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 198.51.100.1
Auth-Port: 143
```

Bad response:

```
HTTP/1.0 200 OK
Auth-Status: Invalid login or password
Auth-Wait: 3
```

If there is no Auth-Wait header, an error will be returned and the connection will be closed. The current implementation allocates memory for each authentication attempt. The memory is freed only at the end of a session. Therefore, the number of invalid authentication attempts in a single session must be limited — the server must respond without the Auth-Wait header after 10-20 attempts (the attempt number is passed in the Auth-Login-Attempt header).

When the APOP or CRAM-MD5 are used, request-response will look as follows:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: apop
Auth-User: user
Auth-Salt: <238188073.1163692009@mail.example.com>
Auth-Pass: auth_response
Auth-Protocol: imap
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
```

Good response:

```
HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 198.51.100.1
```



```
Auth-Port: 143
Auth-Pass: plain-text-pass
```

If the `Auth-User` header exists in the response, it overrides the username used to authenticate with the backend.

For the SMTP, the response additionally takes into account the `Auth-Error-Code` header — if exists, it is used as a response code in case of an error. Otherwise, the 535 5.7.0 code will be added to the `Auth-Status` header.

For example, if the following response is received from the authentication server:

```
HTTP/1.0 200 OK
Auth-Status: Temporary server problem, try again later
Auth-Error-Code: 451 4.3.0
Auth-Wait: 3
```

then the SMTP client will receive an error

```
451 4.3.0 Temporary server problem, try again later
```

If proxying SMTP does not require authentication, the request will look as follows:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: none
Auth-User:
Auth-Pass:
Auth-Protocol: smtp
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
Auth-SMTP-Helo: client.example.org
Auth-SMTP-From: MAIL FROM: <>
Auth-SMTP-To: RCPT TO: <postmaster@mail.example.com>
```

For the SSL/TLS client connection (1.7.11), the `Auth-SSL` header is added, and `Auth-SSL-Verify` will contain the result of client certificate verification, if [enabled](#): “SUCCESS”, “FAILED:*reason*”, and “NONE” if a certificate was not present.

Prior to version 1.11.7, the “FAILED” result did not contain the *reason* string.

When the client certificate was present, its details are passed in the following request headers: `Auth-SSL-Subject`, `Auth-SSL-Issuer`, `Auth-SSL-Serial`, and `Auth-SSL-Fingerprint`. If [auth\\_http\\_pass-client\\_cert](#) is enabled, the certificate itself is passed in the `Auth-SSL-Cert` header. The protocol and cipher of the established connection are passed in the `Auth-SSL-Protocol` and `Auth-SSL-Cipher` headers (1.21.2). The request will look as follows:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: plain
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Auth-SSL: on
Auth-SSL-Protocol: TLSv1.3
Auth-SSL-Cipher: TLS_AES_256_GCM_SHA384
Auth-SSL-Verify: SUCCESS
Auth-SSL-Subject: /CN=example.com
Auth-SSL-Issuer: /CN=example.com
Auth-SSL-Serial: C07AD56B846B5BFF
Auth-SSL-Fingerprint: 29d6a80a123d13355ed16b4b04605e29cb55a5ad
```

When the PROXY protocol is used, its details are passed in the following request headers: `Proxy-Protocol-Addr`, `Proxy-Protocol-Port`, `Proxy-Protocol-Server-Addr`, and `Proxy-Protocol-Server-Port` (1.19.8).

## 4.3 Module ngx\_mail\_proxy\_module

4.3.1 Directives	482
<a href="#">proxy_buffer</a>	482
<a href="#">proxy_pass_error_message</a>	482
<a href="#">proxy_protocol</a>	482
<a href="#">proxy_smtp_auth</a>	483
<a href="#">proxy_timeout</a>	483
<a href="#">xclient</a>	483

### 4.3.1 Directives

#### proxy\_buffer

SYNTAX: **proxy\_buffer** *size*;

DEFAULT 4k | 8k

CONTEXT: mail, server

Sets the size of the buffer used for proxying. By default, the buffer size is equal to one memory page. Depending on a platform, it is either 4K or 8K.

#### proxy\_pass\_error\_message

SYNTAX: **proxy\_pass\_error\_message** on | off;

DEFAULT off

CONTEXT: mail, server

Indicates whether to pass the error message obtained during the authentication on the backend to the client.

Usually, if the authentication in nginx is a success, the backend cannot return an error. If it nevertheless returns an error, it means some internal error has occurred. In such case the backend message can contain information that should not be shown to the client. However, responding with an error for the correct password is a normal behavior for some POP3 servers. For example, CommuniGatePro informs a user about [mailbox overflow](#) or other events by periodically outputting the [authentication error](#). The directive should be enabled in this case.

#### proxy\_protocol

SYNTAX: **proxy\_protocol** on | off;

DEFAULT off

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.19.8.

Enables the [PROXY protocol](#) for connections to a backend.

## proxy\_smtp\_auth

SYNTAX: **proxy\_smtp\_auth** on | off;

DEFAULT off

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

Enables or disables user authentication on the SMTP backend using the AUTH command.

If [XCLIENT](#) is also enabled, then the XCLIENT command will not send the LOGIN parameter.

## proxy\_timeout

SYNTAX: **proxy\_timeout** *timeout*;

DEFAULT 24h

CONTEXT: mail, server

Sets the *timeout* between two successive read or write operations on client or proxied server connections. If no data is transmitted within this time, the connection is closed.

## xclient

SYNTAX: **xclient** on | off;

DEFAULT on

CONTEXT: mail, server

Enables or disables the passing of the [XCLIENT](#) command with client parameters when connecting to the SMTP backend.

With XCLIENT, the MTA is able to write client information to the log and apply various limitations based on this data.

If XCLIENT is enabled then nginx passes the following commands when connecting to the backend:

- EHLO with the [server name](#)
- XCLIENT
- EHLO or HELO, as passed by the client

If the name [found](#) by the client IP address points to the same address, it is passed in the NAME parameter of the XCLIENT command. If the name could not be found, points to a different address, or [resolver](#) is not specified, the [UNAVAILABLE] is passed in the NAME parameter. If an error has occurred in the process of resolving, the [TEMPUNAVAIL] value is used.

If XCLIENT is disabled then nginx passes the EHLO command with the [server name](#) when connecting to the backend if the client has passed EHLO, or HELO with the server name, otherwise.

## 4.4 Module ngx\_mail\_realip\_module

4.4.1	Summary	484
4.4.2	Example Configuration	484
4.4.3	Directives	484
	set_real_ip_from	484

### 4.4.1 Summary

The ngx\_mail\_realip\_module module is used to change the client address and port to the ones sent in the PROXY protocol header (1.19.8). The PROXY protocol must be previously enabled by setting the proxy\_protocol parameter in the listen directive.

### 4.4.2 Example Configuration

```
listen 110 proxy_protocol;  
  
set_real_ip_from 192.168.1.0/24;  
set_real_ip_from 192.168.2.1;  
set_real_ip_from 2001:0db8::/32;
```

### 4.4.3 Directives

#### set\_real\_ip\_from

SYNTAX: **set\_real\_ip\_from** *address* | *CIDR* | *unix::*;

DEFAULT —

CONTEXT: mail, server

Defines trusted addresses that are known to send correct replacement addresses. If the special value `unix:` is specified, all UNIX-domain sockets will be trusted.

## 4.5 Module ngx\_mail\_ssl\_module

4.5.1	Summary	485
4.5.2	Example Configuration	485
4.5.3	Directives	486
	ssl	486
	ssl_certificate	486
	ssl_certificate_key	487
	ssl_ciphers	487
	ssl_client_certificate	487
	ssl_conf_command	488
	ssl_crl	488
	ssl_dhparam	488
	ssl_ecdh_curve	488
	ssl_password_file	489
	ssl_prefer_server_ciphers	489
	ssl_protocols	490
	ssl_session_cache	490
	ssl_session_ticket_key	491
	ssl_session_tickets	491
	ssl_session_timeout	491
	ssl_trusted_certificate	491
	ssl_verify_client	492
	ssl_verify_depth	492
	starttls	492

### 4.5.1 Summary

The `ngx_mail_ssl_module` module provides the necessary support for a mail proxy server to work with the SSL/TLS protocol.

This module is not built by default, it should be enabled with the `--with-mail_ssl_module` configuration parameter.

This module requires the [OpenSSL](#) library.

### 4.5.2 Example Configuration

To reduce the processor load, it is recommended to

- set the number of [worker processes](#) equal to the number of processors,
- enable the [shared](#) session cache,
- disable the [built-in](#) session cache,
- and possibly increase the session [lifetime](#) (by default, 5 minutes):

```
worker_processes auto;

mail {
    ...

    server {
        listen          993 ssl;

        ssl_protocols   TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers      AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate   /usr/local/nginx/conf/cert.pem;
        ssl_certificate_key /usr/local/nginx/conf/cert.key;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 10m;

        ...
    }
}
```

### 4.5.3 Directives

#### ssl

SYNTAX: **ssl** on | off;

DEFAULT off

CONTEXT: mail, server

This directive was made obsolete in version 1.15.0. The `ssl` parameter of the [listen](#) directive should be used instead.

#### ssl\_certificate

SYNTAX: **ssl\_certificate** *file*;

DEFAULT —

CONTEXT: mail, server

Specifies a *file* with the certificate in the PEM format for the given server. If intermediate certificates should be specified in addition to a primary certificate, they should be specified in the same file in the following order: the primary certificate comes first, then the intermediate certificates. A secret key in the PEM format may be placed in the same file.

Since version 1.11.0, this directive can be specified multiple times to load certificates of different types, for example, RSA and ECDSA:

```
server {
    listen          993 ssl;

    ssl_certificate   example.com.rsa.crt;
    ssl_certificate_key example.com.rsa.key;

    ssl_certificate   example.com.ecdsa.crt;
    ssl_certificate_key example.com.ecdsa.key;

    ...
}
```

Only OpenSSL 1.0.2 or higher supports separate certificate chains for different certificates. With older versions, only one certificate chain can be used.

The value `data:certificate` can be specified instead of the *file* (1.15.10), which loads a certificate without using intermediate files. Note that inappropriate use of this syntax may have its security implications, such as writing secret key data to [error log](#).

### ssl\_certificate\_key

SYNTAX: **ssl\_certificate\_key** *file*;

DEFAULT —

CONTEXT: mail, server

Specifies a *file* with the secret key in the PEM format for the given server.

The value `engine:name:id` can be specified instead of the *file* (1.7.9), which loads a secret key with a specified *id* from the OpenSSL engine *name*.

The value `data:key` can be specified instead of the *file* (1.15.10), which loads a secret key without using intermediate files. Note that inappropriate use of this syntax may have its security implications, such as writing secret key data to [error log](#).

### ssl\_ciphers

SYNTAX: **ssl\_ciphers** *ciphers*;

DEFAULT HIGH:!aNULL:!MD5

CONTEXT: mail, server

Specifies the enabled ciphers. The ciphers are specified in the format understood by the OpenSSL library, for example:

```
ssl_ciphers ALL:!aNULL:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
```

The full list can be viewed using the “`openssl ciphers`” command.

The previous versions of nginx used [different](#) ciphers by default.

### ssl\_client\_certificate

SYNTAX: **ssl\_client\_certificate** *file*;

DEFAULT —

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) client certificates.

The list of certificates will be sent to clients. If this is not desired, the [ssl\\_trusted\\_certificate](#) directive can be used.



## ssl\_conf\_command

SYNTAX: **ssl\_conf\_command** *name value*;

DEFAULT —

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.19.4.

Sets arbitrary OpenSSL configuration [commands](#).

The directive is supported when using OpenSSL 1.0.2 or higher.

Several `ssl_conf_command` directives can be specified on the same level:

```
ssl_conf_command Options PrioritizeChaCha;  
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

These directives are inherited from the previous configuration level if and only if there are no `ssl_conf_command` directives defined on the current level.

Note that configuring OpenSSL directly might result in unexpected behavior.

## ssl\_crl

SYNTAX: **ssl\_crl** *file*;

DEFAULT —

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Specifies a *file* with revoked certificates (CRL) in the PEM format used to [verify](#) client certificates.

## ssl\_dhparam

SYNTAX: **ssl\_dhparam** *file*;

DEFAULT —

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 0.7.2.

Specifies a *file* with DH parameters for DHE ciphers.

By default no parameters are set, and therefore DHE ciphers will not be used.

Prior to version 1.11.0, builtin parameters were used by default.

## ssl\_ecdh\_curve

SYNTAX: **ssl\_ecdh\_curve** *curve*;

DEFAULT auto

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSIONS 1.1.0 AND 1.0.6.

Specifies a *curve* for ECDHE ciphers.

When using OpenSSL 1.0.2 or higher, it is possible to specify multiple curves (1.11.0), for example:

```
ssl_ecdh_curve prime256v1:secp384r1;
```

The special value `auto` (1.11.0) instructs nginx to use a list built into the OpenSSL library when using OpenSSL 1.0.2 or higher, or `prime256v1` with older versions.

Prior to version 1.11.0, the `prime256v1` curve was used by default.

When using OpenSSL 1.0.2 or higher, this directive sets the list of curves supported by the server. Thus, in order for ECDSA certificates to work, it is important to include the curves used in the certificates.

## ssl\_password\_file

SYNTAX: **ssl\_password\_file** *file*;

DEFAULT —

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.7.3.

Specifies a *file* with passphrases for [secret keys](#) where each passphrase is specified on a separate line. Passphrases are tried in turn when loading the key.

Example:

```
mail {
    ssl_password_file /etc/keys/global.pass;
    ...

    server {
        server_name mail1.example.com;
        ssl_certificate_key /etc/keys/first.key;
    }

    server {
        server_name mail2.example.com;

        # named pipe can also be used instead of a file
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

## ssl\_prefer\_server\_ciphers

SYNTAX: **ssl\_prefer\_server\_ciphers** on | off;

DEFAULT off

CONTEXT: mail, server

Specifies that server ciphers should be preferred over client ciphers when the SSLv3 and TLS protocols are used.

### ssl\_protocols

SYNTAX: **ssl\_protocols** [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2]  
[TLSv1.3];  
DEFAULT TLSv1 TLSv1.1 TLSv1.2  
CONTEXT: mail, server

Enables the specified protocols.

The TLSv1.1 and TLSv1.2 parameters (1.1.13, 1.0.12) work only when OpenSSL 1.0.1 or higher is used.

The TLSv1.3 parameter (1.13.0) works only when OpenSSL 1.1.1 or higher is used.

### ssl\_session\_cache

SYNTAX: **ssl\_session\_cache** off | none | [builtin[:size]]  
[shared:name:size];  
DEFAULT none  
CONTEXT: mail, server

Sets the types and sizes of caches that store session parameters. A cache can be of any of the following types:

off

the use of a session cache is strictly prohibited: nginx explicitly tells a client that sessions may not be reused.

none

the use of a session cache is gently disallowed: nginx tells a client that sessions may be reused, but does not actually store session parameters in the cache.

builtin

a cache built in OpenSSL; used by one worker process only. The cache size is specified in sessions. If size is not given, it is equal to 20480 sessions. Use of the built-in cache can cause memory fragmentation.

shared

a cache shared between all worker processes. The cache size is specified in bytes; one megabyte can store about 4000 sessions. Each shared cache should have an arbitrary name. A cache with the same name can be used in several servers.

Both cache types can be used simultaneously, for example:

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

but using only shared cache without the built-in cache should be more efficient.

### ssl\_session\_ticket\_key

SYNTAX: **ssl\_session\_ticket\_key** *file*;

DEFAULT —

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.5.7.

Sets a *file* with the secret key used to encrypt and decrypt TLS session tickets. The directive is necessary if the same key has to be shared between multiple servers. By default, a randomly generated key is used.

If several keys are specified, only the first key is used to encrypt TLS session tickets. This allows configuring key rotation, for example:

```
ssl_session_ticket_key current.key;  
ssl_session_ticket_key previous.key;
```

The *file* must contain 80 or 48 bytes of random data and can be created using the following command:

```
openssl rand 80 > ticket.key
```

Depending on the file size either AES256 (for 80-byte keys, 1.11.8) or AES128 (for 48-byte keys) is used for encryption.

### ssl\_session\_tickets

SYNTAX: **ssl\_session\_tickets** on | off;

DEFAULT on

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.5.9.

Enables or disables session resumption through [TLS session tickets](#).

### ssl\_session\_timeout

SYNTAX: **ssl\_session\_timeout** *time*;

DEFAULT 5m

CONTEXT: mail, server

Specifies a time during which a client may reuse the session parameters.

### ssl\_trusted\_certificate

SYNTAX: **ssl\_trusted\_certificate** *file*;

DEFAULT —

CONTEXT: mail, server

THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Specifies a *file* with trusted CA certificates in the PEM format used to [verify](#) client certificates.

In contrast to the certificate set by [ssl\\_client\\_certificate](#), the list of these certificates will not be sent to clients.

### ssl\_verify\_client

SYNTAX: **ssl\_verify\_client** on | off | optional | optional\_no\_ca;  
DEFAULT off  
CONTEXT: mail, server  
THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Enables verification of client certificates. The verification result is passed in the Auth-SSL-Verify header of the [authentication](#) request.

The `optional` parameter requests the client certificate and verifies it if the certificate is present.

The `optional_no_ca` parameter requests the client certificate but does not require it to be signed by a trusted CA certificate. This is intended for the use in cases when a service that is external to nginx performs the actual certificate verification. The contents of the certificate is accessible through requests [sent](#) to the authentication server.

### ssl\_verify\_depth

SYNTAX: **ssl\_verify\_depth** *number*;  
DEFAULT 1  
CONTEXT: mail, server  
THIS DIRECTIVE APPEARED IN VERSION 1.7.11.

Sets the verification depth in the client certificates chain.

### starttls

SYNTAX: **starttls** on | off | only;  
DEFAULT off  
CONTEXT: mail, server

on

allow usage of the STLS command for the POP3 and the STARTTLS command for the IMAP and SMTP;

off

deny usage of the STLS and STARTTLS commands;

only

require preliminary TLS transition.

## 4.6 Module ngx\_mail\_imap\_module

4.6.1 Directives . . . . .	493
imap_auth . . . . .	493
imap_capabilities . . . . .	493
imap_client_buffer . . . . .	494

### 4.6.1 Directives

#### imap\_auth

SYNTAX: **imap\_auth** *method* ...;

DEFAULT plain

CONTEXT: mail, server

Sets permitted methods of authentication for IMAP clients. Supported methods are:

plain

[LOGIN](#), [AUTH=PLAIN](#)

login

[AUTH=LOGIN](#)

cram-md5

[AUTH=CRAM-MD5](#). In order for this method to work, the password must be stored unencrypted.

external

[AUTH=EXTERNAL](#) (1.11.6).

Plain text authentication methods (the [LOGIN](#) command, [AUTH=PLAIN](#), and [AUTH=LOGIN](#)) are always enabled, though if the `plain` and `login` methods are not specified, [AUTH=PLAIN](#) and [AUTH=LOGIN](#) will not be automatically included in [imap\\_capabilities](#).

#### imap\_capabilities

SYNTAX: **imap\_capabilities** *extension* ...;

DEFAULT IMAP4 IMAP4rev1 UIDPLUS

CONTEXT: mail, server

Sets the [IMAP protocol](#) extensions list that is passed to the client in response to the `CAPABILITY` command. The authentication methods specified in the [imap\\_auth](#) directive and [STARTTLS](#) are automatically added to this list depending on the [starttls](#) directive value.

It makes sense to specify the extensions supported by the IMAP backends to which the clients are proxied (if these extensions are related to commands used after the authentication, when nginx transparently proxies a client connection to the backend).

The current list of standardized extensions is published at [www.iana.org](http://www.iana.org).

## imap\_client\_buffer

SYNTAX: **imap\_client\_buffer** *size*;

DEFAULT 4k | 8k

CONTEXT: mail, server

Sets the *size* of the buffer used for reading IMAP commands. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

## 4.7 Module ngx\_mail\_pop3\_module

4.7.1 Directives . . . . .	495
<a href="#">pop3_auth</a> . . . . .	495
<a href="#">pop3_capabilities</a> . . . . .	495

### 4.7.1 Directives

#### pop3\_auth

SYNTAX: **pop3\_auth** *method* ...;

DEFAULT plain

CONTEXT: mail, server

Sets permitted methods of authentication for POP3 clients. Supported methods are:

plain

[USER/PASS](#), [AUTH PLAIN](#), [AUTH LOGIN](#)

apop

[APOP](#). In order for this method to work, the password must be stored unencrypted.

cram-md5

[AUTH CRAM-MD5](#). In order for this method to work, the password must be stored unencrypted.

external

[AUTH EXTERNAL](#) (1.11.6).

Plain text authentication methods ([USER/PASS](#), [AUTH PLAIN](#), and [AUTH LOGIN](#)) are always enabled, though if the `plain` method is not specified, [AUTH PLAIN](#) and [AUTH LOGIN](#) will not be automatically included in [pop3\\_capabilities](#).

#### pop3\_capabilities

SYNTAX: **pop3\_capabilities** *extension* ...;

DEFAULT TOP USER UIDL

CONTEXT: mail, server

Sets the [POP3 protocol](#) extensions list that is passed to the client in response to the CAPA command. The authentication methods specified in the [pop3\\_auth](#) directive ([SASL](#) extension) and [STLS](#) are automatically added to this list depending on the [starttls](#) directive value.

It makes sense to specify the extensions supported by the POP3 backends to which the clients are proxied (if these extensions are related to commands used after the authentication, when nginx transparently proxies the client connection to the backend).

The current list of standardized extensions is published at [www.iana.org](http://www.iana.org).



## 4.8 Module ngx\_mail\_smtp\_module

4.8.1 Directives . . . . .	496
<a href="#">smtp_auth</a> . . . . .	496
<a href="#">smtp_capabilities</a> . . . . .	496
<a href="#">smtp_client_buffer</a> . . . . .	497
<a href="#">smtp_greeting_delay</a> . . . . .	497

### 4.8.1 Directives

#### smtp\_auth

SYNTAX: **smtp\_auth** *method* ...;

DEFAULT plain login

CONTEXT: mail, server

Sets permitted methods of [SASL authentication](#) for SMTP clients. Supported methods are:

plain

[AUTH PLAIN](#)

login

[AUTH LOGIN](#)

cram-md5

[AUTH CRAM-MD5](#). In order for this method to work, the password must be stored unencrypted.

external

[AUTH EXTERNAL](#) (1.11.6).

none

Authentication is not required.

Plain text authentication methods ([AUTH PLAIN](#) and [AUTH LOGIN](#)) are always enabled, though if the `plain` and `login` methods are not specified, [AUTH PLAIN](#) and [AUTH LOGIN](#) will not be automatically included in [smtp\\_capabilities](#).

#### smtp\_capabilities

SYNTAX: **smtp\_capabilities** *extension* ...;

DEFAULT —

CONTEXT: mail, server

Sets the SMTP protocol extensions list that is passed to the client in response to the EHLO command. The authentication methods specified in the [smtp\\_auth](#) directive and [STARTTLS](#) are automatically added to this list depending on the [starttls](#) directive value.

It makes sense to specify the extensions supported by the MTA to which the clients are proxied (if these extensions are related to commands used after

the authentication, when nginx transparently proxies the client connection to the backend).

The current list of standardized extensions is published at [www.iana.org](http://www.iana.org).

### **smtp\_client\_buffer**

SYNTAX: **smtp\_client\_buffer** *size*;  
DEFAULT 4k | 8k  
CONTEXT: mail, server

Sets the *size* of the buffer used for reading SMTP commands. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

### **smtp\_greeting\_delay**

SYNTAX: **smtp\_greeting\_delay** *time*;  
DEFAULT 0  
CONTEXT: mail, server

Allows setting a delay before sending an SMTP greeting in order to reject clients who fail to wait for the greeting before sending SMTP commands.

# Chapter 5

## Miscellaneous

### 5.1 Command-line parameters

5.1.1 Overview	498
----------------	-----

#### 5.1.1 Overview

nginx supports the following command-line parameters:

- `-?` | `-h` — print help for command-line parameters.
- `-c file` — use an alternative configuration *file* instead of a default file.
- `-e file` — use an alternative error log *file* to store the log instead of a default file (1.19.5). The special value `stderr` selects the standard error file.
- `-g directives` — set [global configuration directives](#), for example,

```
nginx -g "pid /var/run/nginx.pid; worker_processes `sysctl -n hw.ncpu`;"
```
- `-p prefix` — set nginx path prefix, i.e. a directory that will keep server files (default value is `/usr/local/nginx`).
- `-q` — suppress non-error messages during configuration testing.
- `-s signal` — send a *signal* to the master process. The argument *signal* can be one of:
  - `stop` — shut down quickly
  - `quit` — shut down gracefully
  - `reload` — reload configuration, start the new worker process with a new configuration, gracefully shut down old worker processes.
  - `reopen` — reopen log files
- `-t` — test the configuration file: nginx checks the configuration for correct syntax, and then tries to open files referred in the configuration.

- `-T` — same as `-t`, but additionally dump configuration files to standard output (1.9.2).
- `-v` — print nginx version.
- `-V` — print nginx version, compiler version, and configure parameters.

# Appendix A

## Changelog for NGINX Plus

This appendix contains the most important changes that may apply to both NGINX Plus and nginx/OSS. Full changelog for nginx/OSS is available in the packages and by the following link: <https://nginx.org/en/CHANGES>

- NGINX Plus R28 (1.23.2), released November 29, 2022
  - API version 8 update:
    - \* SSL extended statistics for each HTTP [upstream](#) and stream [upstream](#)
    - \* SSL extended statistics for each HTTP [server zone](#) and stream [server zone](#)
    - \* Extended statistics for [SSL](#)
  - PROXY protocol v2 TLV variables for Amazon Web Services, Google Cloud Platform, and Microsoft Azure in [HTTP](#) and [stream](#)
  - The `proxy_protocol_tlv_` variable for [HTTP](#) and [stream](#)
  - Variables support for sticky cookie [samesite](#) flag in the [HTTP upstream](#) module
  - Nginx Plus Dashboard now support extended HTTP codes and extended SSL counters
  - TLS session tickets encryption keys are now automatically rotated when using shared memory in the [ssl\\_session\\_cache](#) directive.
  - Looking up of IPv4 addresses while resolving now can be disabled with the `ipv4=off` parameter of the [resolver](#) directive.
  - Processing of a specially crafted mp4 file by the [ngx\\_http\\_mp4\\_module](#) might cause a worker process crash, worker process memory disclosure, or might have potential other impact (CVE-2022-41741, CVE-2022-41742).
- NGINX Plus R27 (1.21.6), released June 28, 2022
  - API version 8:
    - \* SSL statistics for each HTTP [upstream](#) and stream [upstream](#)
    - \* SSL statistics for each HTTP [server zone](#) and stream [server zone](#)
  - HTTP health checks: introduced the [keepalive\\_time](#) parameter of the [health\\_check](#) directive that enables keepalive connections for health checks and specifies the time during which requests can be processed through one keepalive connection
  - JWT Authentication: error code can be customized with the `error` parameter of the [auth\\_jwt\\_require](#) directive if any additional condition of JWT validation fails
  - The `njs-prometheus-module` now supports version 7 of the API, including `/stream/limit_conns/`, `/http/limit_conns/`, `/http/limit_req/` data, and HTTP status code statistics for [upstreams](#), [server zones](#), and [location zones](#)

- NGINX Plus R26 (1.21.5), released Feb 15, 2022
  - JWT key caching with the [auth\\_jwt\\_key\\_cache](#) directive
  - Enhanced ALPN support with the [ssl\\_alpn](#) directive for stream, and the `$ssl_alpn_protocol` variable for [HTTP](#) and [stream](#)
  - The [\\$ssl\\_curve](#) variable that returns the negotiated curve used for SSL handshake key exchange process
  - The [proxy\\_half\\_close](#) directive for stream that allows closing one side of a connection while the data is still transmitted
  - The [mp4\\_start\\_key\\_frame](#) directive in the MP4 module that forces a video to always start with a key frame
- NGINX Plus R25 (1.21.3), released Sep 21, 2021
  - JWT authentication:
    - \* support for signed and then encrypted Nested JWT with the `nested` parameter of the [auth\\_jwt\\_type](#) directive
    - \* additional conditions for JWT validation can be specified with the [auth\\_jwt\\_require](#) directive
    - \* the [\\$jwt\\_payload](#) variable that returns either enclosed JWS token for Nested JWT, or JSON with claims for JWE
    - \* multiple sources of JSON Web Key Sets can be specified in [auth\\_jwt\\_key\\_file](#) and [auth\\_jwt\\_key\\_request](#) directives
    - \* asymmetric RSA-OAEP cryptographic algorithms for JWE
  - API version 7: HTTP status code statistics are now collected per-code, in addition to aggregation per-class, for [upstreams](#), [server zones](#), and [location zones](#).
  - Stream health checks: introduced the `persistent` parameter of the [health\\_check](#) directive that enables persistence to mandatory health checks after reload.
  - TCP Fast Open support with the `fastopen` parameter of the [listen](#) directive in the stream module.
  - Mail proxy:
    - \* the number of errors before closing the connection can be specified with the [max\\_errors](#) directive to mitigate against ALPACA attack
    - \* support for POP3 and IMAP pipelining
    - \* the `Auth-SSL-Protocol` and `Auth-SSL-Cipher` header lines are now [passed](#) to the mail proxy authentication server
  - Security hardening of HTTP request parsing. NGINX Plus will return an error if:
    - \* spaces or control characters are found in the request line, header names, or the Host request header line
    - \* the CONNECT method is used
    - \* both Content-Length and Transfer-Encoding header lines are present in the request
  - Request body filters API now permits buffering of the data being processed.
  - Added support for dynamic SSL certificate loading for [http](#), [grpc](#) and [uwsgi](#) backends.
- NGINX Plus R24 (1.19.10), released Apr 27, 2021
  - Added support for JSON Web Encryption to the [JSON Web Token authorization module](#) (the [auth\\_jwt\\_type](#) directive).
  - HTTP health checks: introduced the `persistent` parameter of the [health\\_check](#) directive that enables persistence to mandatory health checks after reload.

- Flags in the [proxy\\_cookie\\_flags](#) directive can now contain variables.
  - Added [PROXY protocol](#) support for mail (the [proxy\\_protocol](#) parameter of the [listen](#) directive, the [proxy\\_protocol](#) and [set\\_real\\_ip\\_from](#) directives).
  - If free worker connections are exhausted, NGINX Plus starts closing not only keepalive connections, but also connections in [lingering close](#).
  - The maximum duration of a persistent connection can be limited with the [keepalive\\_time](#) directive for [http](#) and [upstream](#) servers.
  - New variable, [\\$connection\\_time](#), contains connection time.
- NGINX Plus R23 (1.19.5), released Dec 8, 2020
    - gRPC health checks: introduced the [type=grpc](#) parameter in the [health\\_check](#) directive that enables active health checks of gRPC upstream servers.
    - [Sticky](#) cookie now can accept the [SameSite](#) attribute with [Strict](#), [Lax](#), or [None](#) values.
    - Support for cookie flags with the [proxy\\_cookie\\_flags](#) and [userid\\_flags](#) directives.
    - Introduced script that performs [unprivileged installation](#) of NGINX Plus.
    - New command-line switch to redefine an error log file: [-e](#).
    - New [set](#) directive for stream that allows setting a value for a variable.
    - Added support for arbitrary OpenSSL configuration [commands](#) with the [ssl\\_conf\\_command](#) directive.
    - The [ssl\\_reject\\_handshake](#) directive that allows rejecting the SSL handshake in the server block.
    - Support for [user authentication](#) on the SMTP backend in mail proxy.
    - Cache manager improved to monitor the minimum amount of free space (see the [min\\_free](#) parameter of the [proxy\\_cache\\_path](#) directive).
- NGINX Plus R22 (1.19.0), released Jun 9, 2020
    - Client certificate validation with OCSP.
    - Unauthorized requests can now be delayed with the [auth\\_delay](#) directive.
    - Updated status dashboard with new visualizations of [limit\\_conn](#) and [limit\\_req](#) statistics.
- NGINX Plus R21 (1.17.9), released Apr 7, 2020
    - The [grpc\\_pass](#) directive can now contain variables.
- NGINX Plus R20 (1.17.6), released Dec 3, 2019
    - Key-value storage: added support for matching a key based on a substring (the [type](#) parameter of the [keyval\\_zone](#) directive was extended with the [prefix](#) option).
    - DNS resolvers can now be specified in [upstream](#) context.
    - Rate limiting: the [\\$limit\\_req\\_status](#) variable keeps the result of limiting a request processing rate. Corresponding metrics are available through the [/http/limit\\_reqs/](#) endpoint in the [API module](#).
    - The [connection limiting](#) facility now has [dry run](#) mode. The [\\$limit\\_conn\\_status](#) variable keeps the result of limiting a number of connections. Corresponding metrics are available through the [/http/limit\\_conns/](#) endpoint in the [API module](#).
    - Additional variables were introduced to keep original IP address and port of the server that a client originally connected to when PROXY protocol was used: [\\$proxy\\_protocol\\_server\\_addr](#), [\\$proxy\\_protocol\\_server\\_port](#).

- NGINX Plus R19 (1.17.3), released Sep 3, 2019
  - Metrics provided by the [status\\_zone](#) directive can now be collected per [location](#).
  - A [set of metrics](#) related to DNS resolver functionality is now available (the `status_zone` parameter of the [resolver](#) directive).
  - The [request limiting](#) facility now has [dry run](#) mode.
  - Key-value storage now has optimized mode for storing IP addresses in CIDR notation (the `type` parameter of the [keyval\\_zone](#) directive).
  - Each key-value pair can now have its own custom expiration timer, either set at [creation](#) time for new entry, or [modified](#) for existing entry.
  - The [limit\\_rate](#) and [limit\\_rate\\_after](#) directives can now contain variables.
  - The [proxy\\_download\\_rate](#) and [proxy\\_upload\\_rate](#) directives can now contain variables.
  - Updated status dashboard with new visualizations for per-location statistic, resolver metrics, and [zone sync](#) status.
- NGINX Plus R18 (1.15.10), released Apr 9, 2019
  - Added support for dynamic SSL certificate loading, either from [file](#) or from [key-value](#) storage (variable should be prefixed with `data:` for the latter case).
  - Active health checks extended with additional logic of verifying arbitrary variables (the `require` parameter of the [match](#) directive).
  - Clustering enhancement: a single [zone sync](#) configuration can now be used for all instances in a cluster with the help of wildcard support in the [listen](#) directive.
  - Port ranges can now be used in the [listen](#) directive.
  - Stream proxy: added an ability to explicitly close existing connections to the particular upstream server after it was removed from the group due to health check failure, API call, or re-resolve action (the [proxy\\_session\\_drop](#) directive).
  - New variable, [\\$upstream\\_bytes\\_sent](#), contains number of bytes sent to an upstream server.
- NGINX Plus R17 (1.15.7), released Dec 11, 2018
  - Added [support](#) for TLS v1.3 [early data](#). Check out the [\\$ssl\\_early\\_data](#) variable which can be used to protect against [replay attacks](#) at the application layer.
  - Introduced two-stage rate limiting (the `delay` parameter of the [limit\\_req](#) directive).
  - Added [support](#) for retrieving JSON Web Key (JWK) set from a subrequest.
  - Added support for the Ed25519 and Ed448 cryptographic [algorithms](#) to the [JSON Web Token authorization module](#).
  - Added an option to enable TCP keepalives for outgoing connections to proxied servers (the [proxy\\_socket\\_keepalive](#) and friends).
  - Fine-grained control over persistent connections to upstreams with the [keepalive\\_timeout](#) and [keepalive\\_requests](#) directives.
  - Added ability to restrict UDP session to a particular number of packets with the [proxy\\_requests](#) directive.
- NGINX Plus R16 (1.15.2), released Sep 5, 2018
  - IMPORTANT: [status](#) and [upstream\\_conf](#) modules deprecated since R13 were finally removed in favor to the [API](#) module.
  - Shared zones synchronization extended to support [keyval](#) and [limit\\_req](#) modules.



- Key-value pairs can now be expired after configured timeout (the `timeout` parameter of the `keyval_zone` directive).
  - Introduced new load balancing algorithm: `random` with two choices (optional).
  - UDP load balancing extended to support multiple incoming datagrams from a client within a single session, thus allowing to proxy/load balance of more complex applications.
  - Added support for PROXY protocol version 2 in `HTTP` and `stream` modules.
  - New variable, `$ssl_preread_protocol`, of stream module contains the highest SSL protocol version supported by the client.
- NGINX Plus R15 (1.13.10), released Apr 10, 2018
    - Added `gRPC proxy` support.
    - Added `HTTP/2 push` support.
    - Introduced `shared zone synchronization` between NGINX Plus nodes. Currently, it is possible to synchronize `sticky` sessions when using `learn` method.
    - Added an ability to completely disable any escaping in access log (the `escape=none` parameter of the `log_format` directive).
    - It is no longer required to run nginx under superuser when using the `proxy_bind` directive with `transparent` parameter on Linux, as worker processes now inherit the `CAP_NET_RAW` capability from the master process.
    - Added the `auth_jwt_leeway` directive used to configure a leeway to account for clock skew when verifying exp and nbf claims, as per [RFC 7519](#).
    - Stream `SSL pre-read` module now can extract list of protocols advertised by the client through ALPN (the `$ssl_preread_alpn_protocols` variable).
    - New variable, `$upstream_queue_time`, contains time the request spent in the upstream `queue`.
  - NGINX Plus R14 (1.13.7), released Dec 12, 2017
    - Added refactored status dashboard v2 (`dashboard.html` in packages) that uses recently introduced `API` subsystem instead of older `status` and `upstream_conf` interfaces. Previous dashboard v1 is still available (`status.html` in packages). Please note that older status and upstream\_conf interfaces, as well as dashboard v1, are going to be removed in NGINX Plus R16.
    - Dynamic key-value pairs support added to the `stream` proxy. `API` version has changed to 2 (see the “[Compatibility](#)” section for details).
    - The `auth_jwt_header_set` and `auth_jwt_claim_set` directives can now handle multiple values, providing complex JWT claims support.
    - Additional cryptographic `algorithms` were introduced in the `JSON Web Token authorization module`.
    - Upstream servers configured with the `resolve` parameter are now being pre-resolved on configuration reload.
    - The `drain` parameter of the `server` directive can now be specified in configuration.
    - New variable: `$ssl_client_escaped_cert`.
    - Swagger UI was adjusted to support custom ports in URLs.
  - NGINX Plus R13 (1.13.4), released Aug 29, 2017
    - Introduced new `API` for accessing various status information, configuring upstream server groups on-the-fly, and managing key-value pairs. Swagger specification and UI is bundled in the nginx-plus packages for easy try-out.

- Introduced new module that creates variables based on [key-value pairs](#), dynamically managed by the new [API](#).
  - Introduced new module for [mirroring](#) requests by creating background mirror subrequests and ignoring their responses.
  - Added support for HTTP [trailers](#).
  - New [worker\\_shutdown\\_timeout](#) directive allows configuring a timeout for graceful shutdown of worker processes.
  - [Sticky](#) cookie with `expires` parameter now also includes the `max-age` attribute defined in [RFC 6265](#).
  - [Sticky learn](#) session affinity mechanism extended with the `header` parameter used to indicate that a session should be created immediately after receiving response headers from upstream server.
  - The [proxy\\_next\\_upstream](#) directive extended with the new `http_429` parameter ("Too Many Requests").
  - Added the ability to specify network buffer sizes in [stream](#) and [mail](#) modules.
  - SSL renegotiation is now allowed on backend connections.
  - Added initial support for TLS 1.3.
- NGINX Plus R12 (1.11.10), released Mar 14, 2017
    - Status module [dataset](#) updated with nginx build name (`nginx_build`), shared zones usage statistics (the `slabs/ subtree`), and additional upstream fields (`name`, `service`).
    - Status dashboard now shows NGINX Plus version, response time metrics, shared zones memory usage, and server names in upstreams.
    - Added [support](#) for the `stale-while-revalidate` and `stale-if-error` Cache-Control extensions, as defined by [RFC 5861](#).
    - Introduced flexible control of caching byte-range responses (the [proxy\\_cache\\_max\\_range\\_offset](#) directive).
    - Cache header `Vary` and `ETag` lengths increased to 128 bytes. Note that the on-disk cache format has changed, so cached content will be invalidated after the upgrade.
    - Introduced the mandatory parameter in the [health\\_check](#) directive, requiring all new servers to pass the associated health check before accepting real traffic.
    - UDP [health checks](#) now may be configured without specifying `match` block.
    - Stream module now supports client SSL certificates [verification](#).
    - Added a number of [SSL variables](#) representing various details about client certificates and capabilities (`$ssl_client_v_end`, `$ssl_client_v_start`, `$ssl_client_v_remain`, `$ssl_curves`, `$ssl_ciphers`). The `$ssl_client_verify` variable was extended to include a reason of failure.
    - The `$ssl_client_idn` and `$ssl_client_s_dn` variables are now compliant with [RFC 2253](#); legacy variants are available as `$ssl_client_idn_legacy` and `$ssl_client_s_dn_legacy`, accordingly.
    - Support for accessing arbitrary JWT fields as [variables](#).
    - Added support for JSON escaping in access logs (the `escape` parameter of the [log\\_format](#) directive).
    - WebP support added to the [image filter](#) module.
    - Duplicate configuration parts excluded from `nginx -T` output.
    - Various improvements in memory usage and performance, including upstream [queue](#) optimization.
  - NGINX Plus R11 (1.11.5), released Oct 25, 2016
    - Introduced dynamic modules binary compatibility between NGINX Plus and corresponding version of nginx/OSS.

- Stream module enhancements (custom [logging](#) with a number of additional [variables](#), [PROXY protocol](#) support for incoming connections, support for [obtaining](#) real IP address and port from PROXY protocol header, ability to [extract server name](#) from SNI to a variable for various purposes, e.g. custom routing).
  - Status module [dataset](#) updated with additional stream metrics ([sessions](#), [discarded](#)).
  - Cache manager improved to support iterative operations mode when deleting old cache files, reducing the disk load (see the [manager\\_files](#), [manager\\_threshold](#), and [manager\\_sleep](#) parameters of the [proxy\\_cache\\_path](#) directive).
  - Added support for using variables in the domain parameter of the [sticky](#) directive.
  - New variable: [\\$upstream\\_bytes\\_received](#).
- NGINX Plus R10 (1.11.3), released Aug 23, 2016
    - New dynamic module: [ModSecurity](#) (package name is `nginx-plus-module-modsecurity`). This is the early release candidate of ModSecurity 3.0.
    - New dynamic module: [nginScript](#) (package name is `nginx-plus-module-njs`).
    - Support for client authorization using the [JSON Web Token](#) (JWT).
    - Stream module enhancements (embedded [variables](#), [resolver](#) support, [map](#) module, [geo](#) and [geoip](#) modules, [A/B testing](#) support).
    - Support for multiple [SSL certificate](#) types per SSL server or SNI name (e.g., RSA and ECDSA).
    - Transparent proxy mode support (the [transparent](#) parameter of the [proxy\\_bind](#) directive).
    - Support for the `IP_BIND_ADDRESS_NO_PORT` socket option where available, allowing for many more upstream connections.
    - HTTP/2 improvements: unbuffered upload support, general bugfixes.
    - New variables: [\\$request\\_id](#), [\\$proxy\\_protocol\\_port](#), [\\$realip\\_remote\\_port](#).
    - Lua module updated to version 0.10.6 (`nginx-plus-extras`, `nginx-plus-module-lua`).
    - Passenger module updated to version 5.0.30 (`nginx-plus-extras`, `nginx-plus-module-passenger`).
    - `headers-more` module updated to version 0.31 (`nginx-plus-extras`, `nginx-plus-module-headers-more`).
    - `set-misc` module updated to version 0.31 (`nginx-plus-extras`, `nginx-plus-module-headers-more`).

NGINX Plus R10 will be the last release to provide the NGINX Plus Extras package. Users should migrate to the NGINX Plus package and use the equivalent dynamic modules.

- NGINX Plus R9 (1.9.13), released Apr 12, 2016
  - Introduced a number of standalone packages with dynamic modules for NGINX Plus (both official and third-party). Packages with official modules:
    - \* `nginx-plus-module-geoip` ([doc](#))
    - \* `nginx-plus-module-image-filter` ([doc](#))
    - \* `nginx-plus-module-perl` ([doc](#))
    - \* `nginx-plus-module-xslt` ([doc](#))

Packages with third-party modules:

- \* nginx-plus-module-headers-more ([site](#))
  - \* nginx-plus-module-lua ([site](#))
  - \* nginx-plus-module-passenger ([site](#))
  - \* nginx-plus-module-rtmp ([site](#))
  - \* nginx-plus-module-set-misc ([site](#))
  - UDP proxy support added to the [stream](#) module.
  - Added support for retrieving upstream servers configuration via DNS SRV records (the [service](#) parameter of the [server](#) directive).
  - Resolver: added support for TCP fallback on retrieving large DNS responses.
  - Change: requests with [non-idempotent](#) method (POST, LOCK, PATCH) are not passed to the next server in upstream group if a request has already been sent to an upstream server. Enabling the `non_idempotent` option in the [proxy\\_next\\_upstream](#) directive explicitly allows retrying such requests.
  - Cache: improved meta-data accounting.
  - Automatic binding of worker processes to available CPUs (the `auto` parameter of the [worker\\_cpu\\_affinity](#) directive).
  - Some write operations can now be [offloaded](#) to [thread pools](#).
  - Added support for [customizing](#) the Server response header field, as well as the signature in standard error messages.
  - Lua module updated to version 0.10.2 (nginx-plus-extras, nginx-plus-module-lua).
  - Passenger module updated to version 5.0.26 (nginx-plus-extras, nginx-plus-module-passenger).
  - headers-more module updated to version 0.29 (nginx-plus-extras, nginx-plus-module-headers-more).
  - Updated status dashboard.
- NGINX Plus R8 (1.9.9), released Dec 29, 2015
    - [HTTP/2](#) support is now included into the nginx-plus and nginx-plus-extras packages. The nginx-plus-http2 and nginx-plus-lua packages are deprecated.
    - Caching improvements, including support of caching [HEAD](#) requests and more effective caching of big responses with the [slice](#) module.
    - Dynamically configured upstream groups now can be configured to [keep states](#) between reloads.
    - Support for arbitrary port in health check requests (the `port` parameter of the [health\\_check](#) directive).
    - Enhancement in the [real IP](#) module: the `$realip_remote_addr` variable.
    - Enhancement in [syslog](#) logging: the `nohostname` parameter.
    - Lua module updated to version 0.9.20 (nginx-plus-extras).
    - The lua-resty-redis Lua module updated to version 0.21 (nginx-plus-extras).
    - Passenger module updated to version 5.0.22 (nginx-plus-extras).
    - headers-more module updated to version 0.28 (nginx-plus-extras).
    - Updated status dashboard.
  - NGINX Plus R7 (1.9.4), released Sep 15, 2015

- Introduced separate family of `nginx-plus-http2` packages with HTTP/2 support included in favor of SPDY. General `nginx-plus` packages still have SPDY support. Please refer to the [listen](#) directive documentation for the instructions on how to enable HTTP/2.
  - [TCP proxy](#) enhancements ([access](#) control; connection [limiting](#); [upload](#) and [download](#) bandwidth control; client-side [PROXY](#) protocol support; ability to [choose](#) local IP address for outgoing connections; the `backlog` parameter of the [listen](#) directive; the [tcp\\_nodelay](#) directive).
  - More efficient connections distribution between worker processes (the `reuseport` parameter of the [listen](#) directive).
  - Introduced [thread pools](#) used for multi-threaded reading and sending files without blocking worker processes.
  - Enhanced support for [modifying HTTP responses](#) (multiple substitutions support, variables support in search strings).
  - A number of additional metrics in the new version (6) of the [status dataset](#) (SSL handshakes and upstream queue overflows in particular).
  - Updated status dashboard.
  - Additional arguments to playlists in the [HLS module](#) (`start`, `end` and `offset`).
  - Support for proxying requests with [NTLM authentication](#).
  - New command-line switch to dump configuration to standard output: `-T`.
  - Added `lua-resty-redis` Lua module (`nginx-plus-extras`).
  - Lua module updated to version 0.9.16 (`nginx-plus-lua`, `nginx-plus-extras`).
  - Passenger module updated to version 5.0.15 (`nginx-plus-extras`).
  - `headers-more` module updated to version 0.26 (`nginx-plus-extras`).
  - `set-misc` module updated to version 0.29 (`nginx-plus-extras`).
- NGINX Plus R6 (1.7.11), released Apr 14, 2015
    - [TCP proxy](#) enhancements (health checks, dynamic reconfiguration, SSL support, logging, status counters).
    - New [least\\_time](#) load balancing method.
    - Unbuffered upload support ([proxy\\_request\\_buffering](#) and friends).
    - Proxy SSL authentication support for [http](#) and [uwsgi](#).
    - Proxy cache enhancements (variables support in [proxy\\_cache](#), `use_temp_path` parameter in [proxy\\_cache\\_path](#)).
    - [Client SSL certificates](#) support in mail proxy.
    - Autoindex module enhancement (the [autoindex\\_format](#) directive).
    - New status dashboard.
    - Lua module updated to version 0.9.16rc1 (`nginx-plus-lua`, `nginx-plus-extras`).
    - Passenger module updated to version 4.0.59 (`nginx-plus-extras`).
    - `set-misc` module updated to version 0.28 (`nginx-plus-extras`).
  - NGINX Plus R5 (1.7.7), released Dec 1, 2014
    - New TCP proxying and load balancing mode (the [stream](#) module).
    - [Sticky](#) session timeout now applies from the most recent request in the session.

- Upstream “draining” can be used to remove an upstream server without interrupting any user sessions (the `drain` command of the [upstream\\_conf](#) dynamic configuration interface).
  - Improved control over request retries in the event of failure, based on [number of tries](#) and [time](#). Also available for `fastcgi`, `uwsgi`, `scgi` and `memcached` modules.
  - Caching: the `Vary` response header is correctly handled (multiple variants of the same resource can be cached). Note that the on-disk cache format has changed, so cached content will be invalidated after the upgrade.
  - Caching: improved support for [byte-range](#) requests.
  - Ability to control upstream bandwidth with the [proxy\\_limit\\_rate](#) directive.
  - Lua module updated to version 0.9.13 (`nginx-plus-lua`, `nginx-plus-extras`).
  - Passenger module updated to version 4.0.53 (`nginx-plus-extras`).
- NGINX Plus R4 (1.7.3), released Jul 22, 2014
    - [MP4](#) module now supports the `end` query argument which sets the end point of playback.
    - Added the ability to [verify](#) backend SSL certificates.
    - Added support for [SNI](#) while working with SSL backends.
    - Added conditional logging for requests (the `if` parameter of the [access\\_log](#) directive).
    - New load balancing method based on [user-defined keys](#) with optional consistency.
    - Cache revalidation now uses `If-None-Match` header if possible.
    - Passphrases for SSL private keys can now be stored in an [external file](#).
    - Introduced a new session affinity mechanism ([sticky learn](#)) based on server-initiated sessions.
    - Added the ability to retrieve a subset of the [extended status](#) data.
    - Lua module updated to version 0.9.10 (`nginx-plus-lua`, `nginx-plus-extras`).
    - Passenger module updated to version 4.0.45 (`nginx-plus-extras`).
- NGINX Plus R3 (1.5.12), released Apr 2, 2014
    - SPDY protocol updated to version 3.1. SPDY/2 is no longer supported.
    - Added [PROXY protocol](#) support (the `proxy_protocol` parameter of the [listen](#) directive).
    - IPv6 support added to [resolver](#).
    - DNS names in upstream groups are periodically re-resolved (the `resolve` parameter of the [server](#) directive).
    - Introduced limiting connections to [upstream servers](#) (the `max_conns` parameter) with optional support for [connections queue](#).
- NGINX Plus R2 (1.5.7), released Dec 12, 2013
    - Enhanced [sticky routing](#) support.
    - Additional [status metrics](#) for virtual hosts and cache zones.
    - [Cache purge](#) support (also available for [FastCGI](#)).
    - Added support for [cache revalidation](#).

- New module: [ngx\\_http\\_auth\\_request\\_module](#) (authorization based on the result of a subrequest).
- NGINX Plus R1 (1.5.3), released Aug 12, 2013
  - Enhanced [status](#) monitoring.
  - Load balancing: [slow start](#) feature.
  - Added syslog support for both [error\\_log](#) and [access\\_log](#).
  - Support for [Apple HTTP Live Streaming](#).
- NGINX Plus 1.5.0-2, released May 27, 2013
  - Added support for active [health checks](#).
- NGINX Plus 1.5.0, released May 7, 2013
  - Security: fixed CVE-2013-2028.
- NGINX Plus 1.3.16, released Apr 19, 2013
  - Added SPDY support.
- NGINX Plus 1.3.13, released Feb 22, 2013
  - Added [sticky](#) sessions support.
  - Added support for proxying WebSocket connections.
- NGINX Plus 1.3.11, released Jan 18, 2013
  - Added base module [ngx\\_http\\_gunzip\\_module](#).
  - New extra module: [ngx\\_http\\_f4f\\_module](#) (Adobe HDS Dynamic Streaming).
  - New extra module: [ngx\\_http\\_session\\_log\\_module](#) (aggregated session logging).
- NGINX Plus 1.3.9-2, released Dec 20, 2012
  - License information updated.
  - End-User License Agreement added to the package.
- NGINX Plus 1.3.9, released Nov 27, 2012
  - Added [dynamic upstream management](#) feature.
  - PDF documentation bundled into package.
- NGINX Plus 1.3.7, released Oct 18, 2012
  - Initial release of NGINX Plus package.

# Appendix B

## Legal Notices

Open source components included in the NGINX Plus (package name is `nginx-plus`) are:

- `nginx/OSS` (1.23.2), distributed under 2-clause BSD license.  
<https://nginx.org/>

Copyright © 2002-2021 Igor Sysoev

Copyright © 2011-2022 Nginx, Inc.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- Internal MD5 implementation based on Alexander Peslyak’s public domain implementation:

This is an OpenSSL-compatible implementation of the RSA Data Security, Inc. MD5 Message-Digest Algorithm (RFC 1321).

Homepage:

<http://openwall.info/wiki/people/solar/software/public-domain-source-code/md5>

Author: Alexander Peslyak, better known as Solar Designer <solar at openwall.com>

This software was written by Alexander Peslyak in 2001. No copyright is claimed, and the software is hereby placed in the public domain. In case this attempt to disclaim copyright and place the software in the public domain is deemed null and void, then the software is Copyright © 2001 Alexander Peslyak and it is hereby released to the general public under the following terms:



1. Redistribution and use in source and binary forms, with or without modification, are permitted.
2. There's ABSOLUTELY NO WARRANTY, express or implied.

(This is a heavily cut-down "BSD license".)

- MurmurHash algorithm (version 2), distributed under MIT license.  
<https://sites.google.com/site/murmurhash/>

Copyright © Austin Appleby

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Components used in status monitoring dashboard v2 only (dashboard.html in nginx-plus package) and distributed under MIT license:

- @babel-core, Babel compiler core (7.18.2).  
<https://github.com/babel/babel/tree/master/packages/babel-core>  
Copyright © 2014-present Sebastian McKenzie and other contributors
- @babel/plugin-proposal-object-rest-spread, produces spec-compliant code by using Babel's objectSpread helper (7.18.0).  
<https://github.com/babel/babel/tree/master/packages/babel-plugin-proposal-object-rest-spread>
- @babel/plugin-transform-runtime, makes helpers reference the module babel-runtime to avoid duplication across your compiled output (7.18.2).  
<https://github.com/babel/babel/tree/master/packages/babel-plugin-transform-runtime>
- @babel/preset-env, a Babel preset for each environment (7.18.2).  
<https://github.com/babel/babel/tree/master/packages/babel-preset-env>
- @babel/preset-react, a Babel preset for all React plugins (7.17.12).  
<https://github.com/babel/babel/tree/master/packages/babel-preset-react>
- autoprefixer, a PostCSS plugin to parse CSS and add vendor prefixes to CSS rules (10.4.7).  
<https://github.com/postcss/autoprefixer>  
Copyright © 2013 Andrey Sitnik <andrey@sitnik.ru>
- babel-loader, allows transpiling JavaScript files using Babel and webpack (8.2.5).  
<https://github.com/babel/babel-loader>  
Copyright © 2014-2019 Luís Couto <hello@luiscouto.pt>
- babel-plugin-istanbul, a babel plugin that adds istanbul instrumentation to ES6 code (6.1.1).  
<https://github.com/istanbuljs/babel-plugin-istanbul>  
Copyright © 2016, Istanbul Code Coverage

- core-js, a modular standard library for JavaScript (3.22.8).  
<https://github.com/zloirock/core-js>  
Copyright © 2014-2022 Denis Pushkarev
- css-loader, interprets `@import` and `url()` like `import/require()` and will resolve them (6.7.1).  
<https://github.com/webpack-contrib/css-loader>  
Copyright © JS Foundation and other contributors
- cssnano, a modular minifier, built on top of the PostCSS ecosystem (5.1.11).  
<https://github.com/cssnano/cssnano>  
Copyright © Ben Briggs <beneb.info@gmail.com>
- eslint, a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code (8.17.0).  
<https://www.npmjs.com/package/eslint>  
Copyright OpenJS Foundation and other contributors, <www.openjsf.org>
- eslint-config-airbnb, exports some ESLint configurations (19.0.4).  
<https://www.npmjs.com/package/eslint-config-airbnb>  
Copyright (c) 2012 Airbnb
- eslint-plugin-import, supports linting of ES2015+ (ES6+) import/export syntax and prevent issues with misspelling of file paths and import names (2.26.0).  
<https://www.npmjs.com/package/eslint-plugin-import>  
Copyright (c) 2015 Ben Mosher
- eslint-plugin-jsx-a11y, static AST checker for accessibility rules on JSX elements (6.5.1).  
<https://www.npmjs.com/package/eslint-plugin-jsx-a11y>  
Copyright (c) 2016 Ethan Cohen
- eslint-plugin-react, react specific linting rules for eslint (7.30.0).  
<https://www.npmjs.com/package/eslint-plugin-react>  
Copyright (c) 2014 Yannick Croissant
- eslint-webpack-plugin, uses eslint to find and fix problems in the JavaScript code (3.1.1).  
<https://www.npmjs.com/package/eslint-webpack-plugin>  
Copyright JS Foundation and other contributors
- history, manage session history with JavaScript (4.10.1).  
<https://github.com/ReactTraining/history>  
Copyright © React Training 2016-2020 Copyright © Remix Software 2020-2021
- html-inline-css-webpack-plugin, converts external stylesheet to embedded stylesheet, aka document stylesheet (1.11.1).  
<https://github.com/Runjuu/html-inline-css-webpack-plugin>  
Copyright © 2018 Huang
- html-webpack-plugin, simplifies creation of HTML files to serve your webpack bundles (5.5.0).  
<https://github.com/jantimon/html-webpack-plugin>  
Copyright © JS Foundation and other contributors
- mini-css-extract-plugin, extracts CSS into separate files (2.6.0).  
<https://github.com/webpack-contrib/mini-css-extract-plugin>  
Copyright © JS Foundation and other contributors
- postcss, a tool for transforming styles with JS plugins (8.4.14).  
<https://github.com/postcss/postcss>  
Copyright 2013 Andrey Sitnik <andrey@sitnik.ru>

- postcss-loader, PostCSS loader for webpack (6.2.1).  
<https://github.com/postcss/postcss-loader>  
Copyright © JS Foundation and other contributors
- postcss-url, PostCSS plugin to rebase url(), inline or copy asset (10.1.3).  
<https://github.com/postcss/postcss-url>  
Copyright © 2014 Maxime Thirouin
- preact, fast 3kb React alternative with the same ES6 API (10.7.3).  
<https://github.com/developit/preact>  
Copyright © 2015-present Jason Miller
- react-dev-utils, utilities used by Create React App (12.0.1).  
<https://github.com/facebook/create-react-app>  
Copyright © 2013-present, Facebook, Inc.
- regenerator-runtime, standalone runtime for Regenerator-compiled generator and async functions (0.13.9).  
<https://github.com/facebook/regenerator>  
Copyright © 2014-present, Facebook, Inc.
- style-loader, injects CSS into the DOM (3.3.1).  
<https://github.com/webpack-contrib/style-loader>  
Copyright © JS Foundation and other contributors
- webpack, a bundler for javascript and friends (5.73.0).  
<https://github.com/webpack/webpack>  
Copyright © JS Foundation and other contributors
- webpack-cli, provides the interface of options webpack uses in its configuration file (4.10.0).  
<https://github.com/webpack/webpack-cli>  
Copyright © JS Foundation and other contributors
- whatwg-fetch, a window.fetch JavaScript polyfill (2.0.4).  
<https://github.com/github/fetch>  
Copyright © 2014-2016 GitHub, Inc.

#### The MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Components used in status monitoring dashboard v2 only (dashboard.html in nginx-plus package) and distributed under 3-clause BSD license and Apache 2.0 license:

- npm-font-open-sans, Open Sans font family - incl. usage of CSS, SCSS, LESS (1.1.0), distributed under Apache 2.0 license.  
<https://github.com/dasrick/npm-font-open-sans>

Copyright © Steve Matteson

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:  
<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Optional add-on and third-party modules provided with NGINX Plus may include additional open-source components. The licenses for these components are included in the installation package for each module.

# Index

absolute\_redirect, [21](#)  
accept\_mutex, [7](#)  
accept\_mutex\_delay, [7](#)  
access\_log, [208](#), [420](#)  
add\_after\_body, [61](#)  
add\_before\_body, [61](#)  
add\_header, [178](#)  
add\_trailer, [178](#)  
addition\_types, [62](#)  
aio, [21](#)  
aio\_write, [22](#)  
alias, [22](#)  
allow, [59](#), [401](#)  
ancient\_browser, [126](#)  
ancient\_browser\_value, [126](#)  
api, [64](#)  
auth\_basic, [114](#)  
auth\_basic\_user\_file, [114](#)  
auth\_delay, [23](#)  
auth\_http, [478](#)  
auth\_http\_header, [478](#)  
auth\_http\_pass\_client\_cert, [478](#)  
auth\_http\_timeout, [478](#)  
auth\_jwt, [117](#)  
auth\_jwt\_claim\_set, [117](#)  
auth\_jwt\_header\_set, [118](#)  
auth\_jwt\_key\_cache, [118](#)  
auth\_jwt\_key\_file, [118](#)  
auth\_jwt\_key\_request, [119](#)  
auth\_jwt\_leeway, [119](#)  
auth\_jwt\_require, [120](#)  
auth\_jwt\_type, [120](#)  
auth\_request, [121](#)  
auth\_request\_set, [122](#)  
autoindex, [123](#)  
autoindex\_exact\_size, [123](#)  
autoindex\_format, [123](#)  
autoindex\_localtime, [124](#)  
  
break, [266](#)  
  
charset, [127](#)  
charset\_map, [128](#)  
charset\_types, [129](#)  
chunked\_transfer\_encoding, [23](#)  
client\_body\_buffer\_size, [24](#)  
client\_body\_in\_file\_only, [24](#)  
client\_body\_in\_single\_buffer, [24](#)  
client\_body\_temp\_path, [24](#)  
client\_body\_timeout, [25](#)  
client\_header\_buffer\_size, [25](#)  
client\_header\_timeout, [25](#)  
client\_max\_body\_size, [25](#)  
connection\_pool\_size, [26](#)  
create\_full\_put\_path, [130](#)  
  
daemon, [7](#)  
dav\_access, [131](#)  
dav\_methods, [131](#)  
debug\_connection, [8](#)  
debug\_points, [8](#)  
default\_type, [26](#)  
deny, [59](#), [401](#)  
directio, [26](#)  
directio\_alignment, [26](#)  
disable\_symlinks, [27](#)  
  
empty\_gif, [133](#)  
env, [8](#)  
error\_log, [9](#)  
error\_page, [28](#)  
etag, [29](#)  
events, [10](#)  
expires, [179](#)  
  
f4f, [134](#)  
f4f\_buffer\_size, [134](#)  
fastcgi\_bind, [136](#)  
fastcgi\_buffer\_size, [137](#)

`fastcgi_buffering`, 137  
`fastcgi_buffers`, 137  
`fastcgi_busy_buffers_size`, 138  
`fastcgi_cache`, 138  
`fastcgi_cache_background_update`, 138  
`fastcgi_cache_bypass`, 138  
`fastcgi_cache_key`, 139  
`fastcgi_cache_lock`, 139  
`fastcgi_cache_lock_age`, 139  
`fastcgi_cache_lock_timeout`, 139  
`fastcgi_cache_max_range_offset`, 140  
`fastcgi_cache_methods`, 140  
`fastcgi_cache_min_uses`, 140  
`fastcgi_cache_path`, 140  
`fastcgi_cache_purge`, 142  
`fastcgi_cache_revalidate`, 143  
`fastcgi_cache_use_stale`, 143  
`fastcgi_cache_valid`, 144  
`fastcgi_catch_stderr`, 144  
`fastcgi_connect_timeout`, 145  
`fastcgi_force_ranges`, 145  
`fastcgi_hide_header`, 145  
`fastcgi_ignore_client_abort`, 145  
`fastcgi_ignore_headers`, 146  
`fastcgi_index`, 146  
`fastcgi_intercept_errors`, 146  
`fastcgi_keep_conn`, 147  
`fastcgi_limit_rate`, 147  
`fastcgi_max_temp_file_size`, 147  
`fastcgi_next_upstream`, 147  
`fastcgi_next_upstream_timeout`, 148  
`fastcgi_next_upstream_tries`, 149  
`fastcgi_no_cache`, 149  
`fastcgi_param`, 149  
`fastcgi_pass`, 150  
`fastcgi_pass_header`, 150  
`fastcgi_pass_request_body`, 150  
`fastcgi_pass_request_headers`, 151  
`fastcgi_read_timeout`, 151  
`fastcgi_request_buffering`, 151  
`fastcgi_send_lowat`, 151  
`fastcgi_send_timeout`, 152  
`fastcgi_socket_keepalive`, 152  
`fastcgi_split_path_info`, 152  
`fastcgi_store`, 152  
`fastcgi_store_access`, 153  
`fastcgi_temp_file_write_size`, 154  
`fastcgi_temp_path`, 154  
`flv`, 156  
`geo`, 157, 402  
`geoip_city`, 161, 405  
`geoip_country`, 160, 404  
`geoip_org`, 162, 406  
`geoip_proxy`, 162  
`geoip_proxy_recursive`, 162  
`grpc_bind`, 164  
`grpc_buffer_size`, 164  
`grpc_connect_timeout`, 164  
`grpc_hide_header`, 165  
`grpc_ignore_headers`, 165  
`grpc_intercept_errors`, 165  
`grpc_next_upstream`, 165  
`grpc_next_upstream_timeout`, 166  
`grpc_next_upstream_tries`, 167  
`grpc_pass`, 167  
`grpc_pass_header`, 167  
`grpc_read_timeout`, 168  
`grpc_send_timeout`, 168  
`grpc_set_header`, 168  
`grpc_socket_keepalive`, 168  
`grpc_ssl_certificate`, 169  
`grpc_ssl_certificate_key`, 169  
`grpc_ssl_ciphers`, 169  
`grpc_ssl_conf_command`, 169  
`grpc_ssl_crl`, 170  
`grpc_ssl_name`, 170  
`grpc_ssl_password_file`, 170  
`grpc_ssl_protocols`, 170  
`grpc_ssl_server_name`, 170  
`grpc_ssl_session_reuse`, 171  
`grpc_ssl_trusted_certificate`, 171  
`grpc_ssl_verify`, 171  
`grpc_ssl_verify_depth`, 171  
`gunzip`, 172  
`gunzip_buffers`, 172  
`gzip`, 173  
`gzip_buffers`, 174  
`gzip_comp_level`, 174  
`gzip_disable`, 174  
`gzip_http_version`, 174  
`gzip_min_length`, 174  
`gzip_proxied`, 175

- gzip\_static, 177
- gzip\_types, 175
- gzip\_vary, 176
- hash, 337, 456
- health\_check, 354, 461
- health\_check\_timeout, 462
- hls, 182
- hls\_buffers, 182
- hls\_forward\_args, 182
- hls\_fragment, 183
- hls\_mp4\_buffer\_size, 183
- hls\_mp4\_max\_buffer\_size, 184
- http, 29
- http2\_body\_preread\_size, 385
- http2\_chunk\_size, 385
- http2\_idle\_timeout, 385
- http2\_max\_concurrent\_pushes, 385
- http2\_max\_concurrent\_streams, 386
- http2\_max\_field\_size, 386
- http2\_max\_header\_size, 386
- http2\_max\_requests, 386
- http2\_push, 387
- http2\_push\_preload, 387
- http2\_recv\_buffer\_size, 387
- http2\_recv\_timeout, 387
- if, 267
- if\_modified\_since, 29
- ignore\_invalid\_headers, 29
- image\_filter, 186
- image\_filter\_buffer, 187
- image\_filter\_interlace, 187
- image\_filter\_jpeg\_quality, 187
- image\_filter\_sharpen, 187
- image\_filter\_transparency, 187
- image\_filter\_webp\_quality, 188
- imap\_auth, 493
- imap\_capabilities, 493
- imap\_client\_buffer, 494
- include, 10
- index, 189
- internal, 30
- ip\_hash, 337
- js\_access, 409
- js\_body\_filter, 192
- js\_content, 193
- js\_fetch\_buffer\_size, 193, 409
- js\_fetch\_ciphers, 193, 409
- js\_fetch\_max\_response\_buffer\_size, 193, 409
- js\_fetch\_protocols, 194, 410
- js\_fetch\_timeout, 194, 410
- js\_fetch\_trusted\_certificate, 194, 410
- js\_fetch\_verify, 194, 410
- js\_fetch\_verify\_depth, 194, 410
- js\_filter, 411
- js\_header\_filter, 195
- js\_import, 195, 411
- js\_include, 195, 411
- js\_path, 196, 412
- js\_preload\_object, 196, 412
- js\_preread, 412
- js\_set, 196, 413
- js\_var, 197, 413
- keepalive, 338
- keepalive\_disable, 30
- keepalive\_requests, 31, 339
- keepalive\_time, 31, 340
- keepalive\_timeout, 31, 340
- keyval, 198, 415
- keyval\_zone, 199, 416
- large\_client\_header\_buffers, 32
- least\_conn, 341, 457
- least\_time, 341, 457
- limit\_conn, 200, 417
- limit\_conn\_dry\_run, 201, 418
- limit\_conn\_log\_level, 201, 418
- limit\_conn\_status, 202
- limit\_conn\_zone, 202, 418
- limit\_except, 32
- limit\_rate, 32
- limit\_rate\_after, 33
- limit\_req, 204
- limit\_req\_dry\_run, 205
- limit\_req\_log\_level, 205
- limit\_req\_status, 206
- limit\_req\_zone, 206
- limit\_zone, 202
- lingering\_close, 33
- lingering\_time, 34

- lingering\_timeout, 34
- listen, 35, 393, 473
- load\_module, 10
- location, 38
- lock\_file, 10
- log\_format, 210, 421
- log\_not\_found, 39
- log\_subrequest, 40
- mail, 475
- map, 212, 423
- map\_hash\_bucket\_size, 214, 424
- map\_hash\_max\_size, 214, 425
- master\_process, 11
- match, 355, 462
- max\_errors, 475
- max\_ranges, 40
- memcached\_bind, 215
- memcached\_buffer\_size, 216
- memcached\_connect\_timeout, 216
- memcached\_gzip\_flag, 216
- memcached\_next\_upstream, 217
- memcached\_next\_upstream\_timeout, 217
- memcached\_next\_upstream\_tries, 217
- memcached\_pass, 218
- memcached\_read\_timeout, 218
- memcached\_send\_timeout, 218
- memcached\_socket\_keepalive, 218
- merge\_slashes, 40
- min\_delete\_depth, 131
- mirror, 220
- mirror\_request\_body, 220
- modern\_browser, 126
- modern\_browser\_value, 126
- mp4, 223
- mp4\_buffer\_size, 223
- mp4\_limit\_rate, 224
- mp4\_limit\_rate\_after, 224
- mp4\_max\_buffer\_size, 224
- mp4\_start\_key\_frame, 224
- msie\_padding, 41
- msie\_refresh, 41
- multi\_accept, 11
- ntlm, 340
- open\_file\_cache, 41
- open\_file\_cache\_errors, 42
- open\_file\_cache\_min\_uses, 42
- open\_file\_cache\_valid, 42
- open\_log\_file\_cache, 211, 422
- output\_buffers, 42
- override\_charset, 129
- pcrc\_jit, 11
- perl, 226
- perl\_modules, 227
- perl\_require, 227
- perl\_set, 227
- pid, 11
- pop3\_auth, 495
- pop3\_capabilities, 495
- port\_in\_redirect, 42
- postpone\_output, 43
- preread\_buffer\_size, 395
- preread\_timeout, 396
- protocol, 475
- proxy\_bind, 233, 427
- proxy\_buffer, 482
- proxy\_buffer\_size, 233, 427
- proxy\_buffering, 233
- proxy\_buffers, 234
- proxy\_busy\_buffers\_size, 234
- proxy\_cache, 234
- proxy\_cache\_background\_update, 234
- proxy\_cache\_bypass, 235
- proxy\_cache\_convert\_head, 235
- proxy\_cache\_key, 235
- proxy\_cache\_lock, 235
- proxy\_cache\_lock\_age, 236
- proxy\_cache\_lock\_timeout, 236
- proxy\_cache\_max\_range\_offset, 236
- proxy\_cache\_methods, 236
- proxy\_cache\_min\_uses, 237
- proxy\_cache\_path, 237
- proxy\_cache\_purge, 239
- proxy\_cache\_revalidate, 239
- proxy\_cache\_use\_stale, 240
- proxy\_cache\_valid, 240
- proxy\_connect\_timeout, 241, 428
- proxy\_cookie\_domain, 241
- proxy\_cookie\_flags, 242
- proxy\_cookie\_path, 243
- proxy\_download\_rate, 428



- proxy\_force\_ranges, 243
- proxy\_half\_close, 428
- proxy\_headers\_hash\_bucket\_size, 244
- proxy\_headers\_hash\_max\_size, 244
- proxy\_hide\_header, 244
- proxy\_http\_version, 244
- proxy\_ignore\_client\_abort, 244
- proxy\_ignore\_headers, 245
- proxy\_intercept\_errors, 245
- proxy\_limit\_rate, 245
- proxy\_max\_temp\_file\_size, 246
- proxy\_method, 246
- proxy\_next\_upstream, 246, 428
- proxy\_next\_upstream\_timeout, 247, 429
- proxy\_next\_upstream\_tries, 247, 429
- proxy\_no\_cache, 248
- proxy\_pass, 248, 429
- proxy\_pass\_error\_message, 482
- proxy\_pass\_header, 249
- proxy\_pass\_request\_body, 250
- proxy\_pass\_request\_headers, 250
- proxy\_protocol, 429, 482
- proxy\_protocol\_timeout, 396
- proxy\_read\_timeout, 250
- proxy\_redirect, 250
- proxy\_request\_buffering, 252
- proxy\_requests, 430
- proxy\_responses, 430
- proxy\_send\_lowat, 252
- proxy\_send\_timeout, 252
- proxy\_session\_drop, 430
- proxy\_set\_body, 253
- proxy\_set\_header, 253
- proxy\_smtp\_auth, 483
- proxy\_socket\_keepalive, 254, 431
- proxy\_ssl, 431
- proxy\_ssl\_certificate, 254, 431
- proxy\_ssl\_certificate\_key, 254, 431
- proxy\_ssl\_ciphers, 254, 431
- proxy\_ssl\_conf\_command, 255, 432
- proxy\_ssl\_crl, 255, 432
- proxy\_ssl\_name, 255, 432
- proxy\_ssl\_password\_file, 255, 432
- proxy\_ssl\_protocols, 256, 433
- proxy\_ssl\_server\_name, 256, 433
- proxy\_ssl\_session\_reuse, 256, 433
- proxy\_ssl\_trusted\_certificate, 256, 433
- proxy\_ssl\_verify, 256, 433
- proxy\_ssl\_verify\_depth, 257, 433
- proxy\_store, 257
- proxy\_store\_access, 258
- proxy\_temp\_file\_write\_size, 258
- proxy\_temp\_path, 258
- proxy\_timeout, 434, 483
- proxy\_upload\_rate, 434
- queue, 342
- random, 342, 457
- random\_index, 261
- read\_ahead, 43
- real\_ip\_header, 262
- real\_ip\_recursive, 263
- recursive\_error\_pages, 43
- referer\_hash\_bucket\_size, 264
- referer\_hash\_max\_size, 264
- request\_pool\_size, 43
- reset\_timedout\_connection, 43
- resolver, 44, 342, 396, 458, 475
- resolver\_timeout, 45, 343, 397, 459, 476
- return, 268, 437
- rewrite, 268
- rewrite\_log, 270
- root, 45
- satisfy, 45
- scgi\_bind, 273
- scgi\_buffer\_size, 273
- scgi\_buffering, 274
- scgi\_buffers, 274
- scgi\_busy\_buffers\_size, 274
- scgi\_cache, 275
- scgi\_cache\_background\_update, 275
- scgi\_cache\_bypass, 275
- scgi\_cache\_key, 275
- scgi\_cache\_lock, 275
- scgi\_cache\_lock\_age, 276
- scgi\_cache\_lock\_timeout, 276
- scgi\_cache\_max\_range\_offset, 276
- scgi\_cache\_methods, 276
- scgi\_cache\_min\_uses, 277

scgi\_cache\_path, 277  
scgi\_cache\_purge, 279  
scgi\_cache\_revalidate, 279  
scgi\_cache\_use\_stale, 279  
scgi\_cache\_valid, 280  
scgi\_connect\_timeout, 281  
scgi\_force\_ranges, 281  
scgi\_hide\_header, 281  
scgi\_ignore\_client\_abort, 282  
scgi\_ignore\_headers, 282  
scgi\_intercept\_errors, 282  
scgi\_limit\_rate, 282  
scgi\_max\_temp\_file\_size, 283  
scgi\_next\_upstream, 283  
scgi\_next\_upstream\_timeout, 284  
scgi\_next\_upstream\_tries, 284  
scgi\_no\_cache, 285  
scgi\_param, 285  
scgi\_pass, 285  
scgi\_pass\_header, 286  
scgi\_pass\_request\_body, 286  
scgi\_pass\_request\_headers, 286  
scgi\_read\_timeout, 286  
scgi\_request\_buffering, 287  
scgi\_send\_timeout, 287  
scgi\_socket\_keepalive, 287  
scgi\_store, 287  
scgi\_store\_access, 288  
scgi\_temp\_file\_write\_size, 288  
scgi\_temp\_path, 289  
secure\_link, 290  
secure\_link\_md5, 291  
secure\_link\_secret, 291  
send\_lowat, 46  
send\_timeout, 46  
sendfile, 46  
sendfile\_max\_chunk, 47  
server, 47, 333, 397, 453, 477  
server\_name, 47, 477  
server\_name\_in\_redirect, 49  
server\_names\_hash\_bucket\_size, 49  
server\_names\_hash\_max\_size, 49  
server\_tokens, 50  
session\_log, 293  
session\_log\_format, 293  
session\_log\_zone, 294  
set, 270, 438  
set\_realip\_from, 262, 436, 484  
slice, 295  
smtp\_auth, 496  
smtp\_capabilities, 496  
smtp\_client\_buffer, 497  
smtp\_greeting\_delay, 497  
source\_charset, 129  
split\_clients, 297, 439  
ssi, 298  
ssi\_last\_modified, 298  
ssi\_min\_file\_chunk, 299  
ssi\_silent\_errors, 299  
ssi\_types, 299  
ssi\_value\_length, 299  
ssl, 305, 486  
ssl\_alpn, 441  
ssl\_buffer\_size, 305  
ssl\_certificate, 306, 441, 486  
ssl\_certificate\_key, 307, 442, 487  
ssl\_ciphers, 307, 442, 487  
ssl\_client\_certificate, 307, 443, 487  
ssl\_conf\_command, 307, 443, 488  
ssl\_crl, 308, 443, 488  
ssl\_dhparam, 308, 444, 488  
ssl\_early\_data, 308  
ssl\_ecdh\_curve, 309, 444, 488  
ssl\_engine, 11  
ssl\_handshake\_timeout, 444  
ssl\_ocsp, 309  
ssl\_ocsp\_cache, 310  
ssl\_ocsp\_responder, 310  
ssl\_password\_file, 310, 444, 489  
ssl\_prefer\_server\_ciphers, 311, 445, 489  
ssl\_preread, 451  
ssl\_protocols, 311, 445, 490  
ssl\_reject\_handshake, 311  
ssl\_session\_cache, 311, 445, 490  
ssl\_session\_ticket\_key, 312, 446, 491  
ssl\_session\_tickets, 313, 447, 491  
ssl\_session\_timeout, 313, 447, 491  
ssl\_stapling, 313  
ssl\_stapling\_file, 313  
ssl\_stapling\_responder, 314  
ssl\_stapling\_verify, 314  
ssl\_trusted\_certificate, 314, 447, 491

ssl\_verify\_client, [314](#), [447](#), [492](#)  
ssl\_verify\_depth, [315](#), [448](#), [492](#)  
starttls, [492](#)  
state, [336](#), [456](#)  
status, [319](#)  
status\_format, [319](#)  
status\_zone, [65](#), [320](#)  
sticky, [343](#)  
sticky\_cookie\_insert, [346](#)  
stream, [397](#)  
stub\_status, [328](#)  
sub\_filter, [330](#)  
sub\_filter\_last\_modified, [330](#)  
sub\_filter\_once, [331](#)  
sub\_filter\_types, [331](#)  
subrequest\_output\_buffer\_size, [50](#)  
  
tcp\_nodelay, [50](#), [397](#)  
tcp\_nopush, [50](#)  
thread\_pool, [12](#)  
timeout, [477](#)  
timer\_resolution, [12](#)  
try\_files, [51](#)  
types, [53](#)  
types\_hash\_bucket\_size, [53](#)  
types\_hash\_max\_size, [53](#)  
  
underscores\_in\_headers, [54](#)  
uninitialized\_variable\_warn, [270](#)  
upstream, [333](#), [453](#)  
upstream\_conf, [349](#)  
use, [12](#)  
user, [13](#)  
userid, [358](#)  
userid\_domain, [359](#)  
userid\_expires, [359](#)  
userid\_flags, [359](#)  
userid\_mark, [359](#)  
userid\_name, [360](#)  
userid\_p3p, [360](#)  
userid\_path, [360](#)  
userid\_service, [360](#)  
uwsgi\_bind, [363](#)  
uwsgi\_buffer\_size, [364](#)  
uwsgi\_buffering, [364](#)  
uwsgi\_buffers, [365](#)  
uwsgi\_busy\_buffers\_size, [365](#)  
uwsgi\_cache, [365](#)  
uwsgi\_cache\_background\_update, [365](#)  
uwsgi\_cache\_bypass, [365](#)  
uwsgi\_cache\_key, [366](#)  
uwsgi\_cache\_lock, [366](#)  
uwsgi\_cache\_lock\_age, [366](#)  
uwsgi\_cache\_lock\_timeout, [366](#)  
uwsgi\_cache\_max\_range\_offset, [367](#)  
uwsgi\_cache\_methods, [367](#)  
uwsgi\_cache\_min\_uses, [367](#)  
uwsgi\_cache\_path, [367](#)  
uwsgi\_cache\_purge, [369](#)  
uwsgi\_cache\_revalidate, [370](#)  
uwsgi\_cache\_use\_stale, [370](#)  
uwsgi\_cache\_valid, [371](#)  
uwsgi\_connect\_timeout, [371](#)  
uwsgi\_force\_ranges, [372](#)  
uwsgi\_hide\_header, [372](#)  
uwsgi\_ignore\_client\_abort, [372](#)  
uwsgi\_ignore\_headers, [372](#)  
uwsgi\_intercept\_errors, [373](#)  
uwsgi\_limit\_rate, [373](#)  
uwsgi\_max\_temp\_file\_size, [373](#)  
uwsgi\_modifier1, [374](#)  
uwsgi\_modifier2, [374](#)  
uwsgi\_next\_upstream, [374](#)  
uwsgi\_next\_upstream\_timeout, [375](#)  
uwsgi\_next\_upstream\_tries, [375](#)  
uwsgi\_no\_cache, [375](#)  
uwsgi\_param, [376](#)  
uwsgi\_pass, [376](#)  
uwsgi\_pass\_header, [377](#)  
uwsgi\_pass\_request\_body, [377](#)  
uwsgi\_pass\_request\_headers, [377](#)  
uwsgi\_read\_timeout, [377](#)  
uwsgi\_request\_buffering, [377](#)  
uwsgi\_send\_timeout, [378](#)  
uwsgi\_socket\_keepalive, [378](#)  
uwsgi\_ssl\_certificate, [378](#)  
uwsgi\_ssl\_certificate\_key, [378](#)  
uwsgi\_ssl\_ciphers, [379](#)  
uwsgi\_ssl\_conf\_command, [379](#)  
uwsgi\_ssl\_crl, [379](#)  
uwsgi\_ssl\_name, [379](#)  
uwsgi\_ssl\_password\_file, [380](#)  
uwsgi\_ssl\_protocols, [380](#)

uwsgi\_ssl\_server\_name, [380](#)  
uwsgi\_ssl\_session\_reuse, [380](#)  
uwsgi\_ssl\_trusted\_certificate, [380](#)  
uwsgi\_ssl\_verify, [381](#)  
uwsgi\_ssl\_verify\_depth, [381](#)  
uwsgi\_store, [381](#)  
uwsgi\_store\_access, [382](#)  
uwsgi\_temp\_file\_write\_size, [382](#)  
uwsgi\_temp\_path, [382](#)  
  
valid\_referers, [265](#)  
variables\_hash\_bucket\_size, [54](#), [397](#)  
variables\_hash\_max\_size, [54](#), [398](#)  
  
worker\_aio\_requests, [13](#)  
worker\_connections, [13](#)  
worker\_cpu\_affinity, [13](#)  
worker\_priority, [14](#)  
worker\_processes, [14](#)  
worker\_rlimit\_core, [15](#)  
worker\_rlimit\_nofile, [15](#)  
worker\_shutdown\_timeout, [15](#)  
working\_directory, [15](#)  
  
xclient, [483](#)  
xml\_entities, [389](#)  
xslt\_last\_modified, [390](#)  
xslt\_param, [390](#)  
xslt\_string\_param, [390](#)  
xslt\_stylesheet, [390](#)  
xslt\_types, [391](#)  
  
zone, [336](#), [455](#)  
zone\_sync, [466](#)  
zone\_sync\_buffers, [466](#)  
zone\_sync\_connect\_retry\_interval, [466](#)  
zone\_sync\_connect\_timeout, [466](#)  
zone\_sync\_interval, [466](#)  
zone\_sync\_recv\_buffer\_size, [467](#)  
zone\_sync\_server, [467](#)  
zone\_sync\_ssl, [467](#)  
zone\_sync\_ssl\_certificate, [468](#)  
zone\_sync\_ssl\_certificate\_key, [468](#)  
zone\_sync\_ssl\_ciphers, [468](#)  
zone\_sync\_ssl\_conf\_command, [468](#)  
zone\_sync\_ssl\_crl, [469](#)  
zone\_sync\_ssl\_name, [469](#)  
zone\_sync\_ssl\_password\_file, [469](#)  
zone\_sync\_ssl\_protocols, [469](#)  
zone\_sync\_ssl\_server\_name, [469](#)  
zone\_sync\_ssl\_trusted\_certificate, [470](#)  
zone\_sync\_ssl\_verify, [470](#)  
zone\_sync\_ssl\_verify\_depth, [470](#)  
zone\_sync\_timeout, [470](#)