



[Nginx, Inc.](#)

NGINX Plus Reference Guide

NGINX Plus - release 22, based on 1.19.0 core

June 4, 2020

Copyright Notice

© 2012-2020 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® (“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

Contents

Title	1
Preface	2
Table of Contents	3
1 Core modules	6
1.1 Core functionality	6
1.2 Setting up hashes	16
1.3 Connection processing methods	17
1.4 Logging to syslog	18
2 HTTP server modules	19
2.1 Module ngx_http_core_module	19
2.2 Module ngx_http_access_module	58
2.3 Module ngx_http_addition_module	60
2.4 Module ngx_http_api_module	62
2.5 Module ngx_http_auth_basic_module	109
2.6 Module ngx_http_auth_jwt_module	111
2.7 Module ngx_http_auth_request_module	115
2.8 Module ngx_http_autoindex_module	117
2.9 Module ngx_http_browser_module	119
2.10 Module ngx_http_charset_module	121
2.11 Module ngx_http_dav_module	124
2.12 Module ngx_http_empty_gif_module	127
2.13 Module ngx_http_f4f_module	128
2.14 Module ngx_http_fastcgi_module	129
2.15 Module ngx_http_flv_module	150
2.16 Module ngx_http_geo_module	151
2.17 Module ngx_http_geoip_module	154
2.18 Module ngx_http_grpc_module	157
2.19 Module ngx_http_gunzip_module	166
2.20 Module ngx_http_gzip_module	167
2.21 Module ngx_http_gzip_static_module	171
2.22 Module ngx_http_headers_module	172
2.23 Module ngx_http_hls_module	175
2.24 Module ngx_http_image_filter_module	179

2.25	Module ngx_http_index_module	183
2.26	Module ngx_http_js_module	184
2.27	Module ngx_http_keyval_module	187
2.28	Module ngx_http_limit_conn_module	189
2.29	Module ngx_http_limit_req_module	193
2.30	Module ngx_http_log_module	197
2.31	Module ngx_http_map_module	201
2.32	Module ngx_http_memcached_module	204
2.33	Module ngx_http_mirror_module	209
2.34	Module ngx_http_mp4_module	211
2.35	Module ngx_http_perl_module	214
2.36	Module ngx_http_proxy_module	220
2.37	Module ngx_http_random_index_module	248
2.38	Module ngx_http_realip_module	249
2.39	Module ngx_http_referer_module	251
2.40	Module ngx_http_rewrite_module	253
2.41	Module ngx_http_scgi_module	259
2.42	Module ngx_http_secure_link_module	277
2.43	Module ngx_http_session_log_module	280
2.44	Module ngx_http_slice_module	282
2.45	Module ngx_http_split_clients_module	284
2.46	Module ngx_http_ssi_module	285
2.47	Module ngx_http_ssl_module	291
2.48	Module ngx_http_status_module	304
2.49	Module ngx_http_stub_status_module	314
2.50	Module ngx_http_sub_module	316
2.51	Module ngx_http_upstream_module	318
2.52	Module ngx_http_upstream_conf_module	334
2.53	Module ngx_http_upstream_hc_module	338
2.54	Module ngx_http_userid_module	342
2.55	Module ngx_http_uwsgi_module	345
2.56	Module ngx_http_v2_module	366
2.57	Module ngx_http_xslt_module	370
3	Stream server modules	373
3.1	Module ngx_stream_core_module	373
3.2	Module ngx_stream_access_module	381
3.3	Module ngx_stream_geo_module	382
3.4	Module ngx_stream_geoip_module	384
3.5	Module ngx_stream_js_module	387
3.6	Module ngx_stream_keyval_module	391
3.7	Module ngx_stream_limit_conn_module	393
3.8	Module ngx_stream_log_module	396
3.9	Module ngx_stream_map_module	399
3.10	Module ngx_stream_proxy_module	402
3.11	Module ngx_stream_realip_module	410

3.12	Module ngx_stream_return_module	411
3.13	Module ngx_stream_split_clients_module	412
3.14	Module ngx_stream_ssl_module	413
3.15	Module ngx_stream_ssl_preread_module	422
3.16	Module ngx_stream_upstream_module	424
3.17	Module ngx_stream_upstream_hc_module	432
3.18	Module ngx_stream_zone_sync_module	436
4	Mail server modules	443
4.1	Module ngx_mail_core_module	443
4.2	Module ngx_mail_auth_http_module	449
4.3	Module ngx_mail_proxy_module	453
4.4	Module ngx_mail_ssl_module	455
4.5	Module ngx_mail_imap_module	463
4.6	Module ngx_mail_pop3_module	465
4.7	Module ngx_mail_smtp_module	466
5	Miscellaneous	468
5.1	High Availability support for NGINX Plus	468
5.2	Command-line parameters	474
A	Changelog for NGINX Plus	475
B	Legal Notices	484
	Index	489

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 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 named thread pools 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_timeout	31
large_client_header_buffers	31
limit_except	32

limit_rate	32
limit_rate_after	33
lingering_close	33
lingering_time	33
lingering_timeout	34
listen	34
location	37
log_not_found	39
log_subrequest	39
max_ranges	39
merge_slashes	39
msie_padding	40
msie_refresh	40
open_file_cache	40
open_file_cache_errors	41
open_file_cache_min_uses	41
open_file_cache_valid	41
output_buffers	42
port_in_redirect	42
postpone_output	42
read_ahead	42
recursive_error_pages	42
request_pool_size	43
reset_timedout_connection	43
resolver	43
resolver_timeout	44
root	44
satisfy	45
send_lowat	45
send_timeout	45
sendfile	45
sendfile_max_chunk	46
server	46
server_name	46
server_name_in_redirect	48
server_names_hash_bucket_size	48
server_names_hash_max_size	49
server_tokens	49
subrequest_output_buffer_size	49
tcp_nodelay	49
tcp_nopush	50
try_files	50
types	52
types_hash_bucket_size	52
types_hash_max_size	53
underscores_in_headers	53

variables_hash_bucket_size	53
variables_hash_max_size	53
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.

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, specified in the Content-Length request header field. 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 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, its value is only used if a server is a default one. The value specified also applies to all virtual servers listening on the same address and port.

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 100

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.

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.

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.

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`, `backlog`, `rcvbuf`, `sndbuf`, `accept_filter`, `deferred`, `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 (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, its value is only used if a server is a default one. The value specified also applies to all virtual servers listening on the same address and port.

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 `fcntl(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*] [*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 IPv6 addresses is not desired, 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 0
CONTEXT: http, server, location

When set to a non-zero value, 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.

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 jpg;

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, its value is only used if a server is a default one. The value specified also applies to all virtual servers listening on the same address and port.

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)

\$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.

\$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	58
2.2.2	Example Configuration	58
2.2.3	Directives	58
	allow	58
	deny	58

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	60
2.3.2	Example Configuration	60
2.3.3	Directives	60
	add_before_body	60
	add_after_body	60
	addition_types	61

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	62
2.4.2	Example Configuration	62
2.4.3	Directives	63
	api	63
	status_zone	64
2.4.4	Compatibility	64
2.4.5	Endpoints	65
2.4.6	Response Objects	87

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/6/
http://127.0.0.1/api/6/nginx
http://127.0.0.1/api/6/connections
http://127.0.0.1/api/6/http/requests
http://127.0.0.1/api/6/http/server_zones/server_backend
http://127.0.0.1/api/6/http/caches/cache_backend
http://127.0.0.1/api/6/http/upstreams/backend
http://127.0.0.1/api/6/http/upstreams/backend/servers/
http://127.0.0.1/api/6/http/upstreams/backend/servers/1
http://127.0.0.1/api/6/http/keyvals/one?key=arg1
http://127.0.0.1/api/6/stream/
http://127.0.0.1/api/6/stream/server_zones/server_backend
http://127.0.0.1/api/6/stream/upstreams/
http://127.0.0.1/api/6/stream/upstreams/backend
http://127.0.0.1/api/6/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 “6”.

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

```
http://127.0.0.1/api/6/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 [/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 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 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" : "1.17.3",
    "build" : "nginx-plus-r19",
    "address" : "206.251.255.64",
    "generation" : 6,
    "load_timestamp" : "2019-10-01T11:15:44.467Z",
    "timestamp" : "2019-10-01T09:26:07.305Z",
    "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.

Example:

```

{
  "handshakes" : 79572,
  "handshakes_failed" : 21025,
  "session_reuses" : 15762
}

```

- Shared memory 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 and the number of responses with status codes “1xx”, “2xx”, “3xx”, “4xx”, and “5xx”.

`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.

`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:

```
{
  "processing" : 1,
  "requests" : 706690,
  "responses" : {
    "1xx" : 0,
    "2xx" : 699482,
    "3xx" : 4522,
    "4xx" : 907,
    "5xx" : 266,
    "total" : 705177
  },
  "discarded" : 1513,
  "received" : 172711587,
  "sent" : 19415530115
}
```

- HTTP Location Zone:

`requests (integer)`

The total number of client requests received from clients.

`responses`

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

`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.
`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,
    "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.

`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.

`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,
        "max_conns" : 20,
        "requests" : 667231,
        "header_time" : 20,
        "response_time" : 36,
        "responses" : {
```

```

        "1xx" : 0,
        "2xx" : 666310,
        "3xx" : 0,
        "4xx" : 915,
        "5xx" : 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" : "2019-10-01T11:09:21.602Z",
    "selected" : "2019-10-01T15:01:25.000Z"
},
{
    "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,
        "total" : 0
    },
    "sent" : 0,
    "received" : 0,
    "fails" : 0,
    "unavail" : 0,
    "health_checks" : {
        "checks" : 26284,
        "fails" : 26284,
        "unhealthy" : 1,
        "last_passed" : false
    },
    "downtime" : 262925617,
    "downstart" : "2019-10-01T11:09:21.602Z",
    "selected" : "2019-10-01T15:01:25.000Z"
},
],
"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.

Example:

```
{
  "dns" : {
    "processing" : 1,
    "connections" : 155569,
    "sessions" : {
      "2xx" : 155564,
      "4xx" : 0,
      "5xx" : 0,
      "total" : 155569
    },
    "discarded" : 0,
    "received" : 4200363,
    "sent" : 20489184
  }
}
```

- 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.

`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,
        "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" : "2019-10-01T11:09:21.602Z",
        "selected" : "2019-10-01T15:01:25.000Z"
      },
      {
        "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" : "2019-10-01T11:09:21.602Z",
        "selected" : "2019-10-01T15:01:25.000Z"
    }
},
"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	109
2.5.2	Example Configuration	109
2.5.3	Directives	109
	auth_basic	109
	auth_basic_user_file	109

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` allows cancelling 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	111
2.6.2	Example Configuration	111
2.6.3	Directives	112
	auth_jwt	112
	auth_jwt_claim_set	112
	auth_jwt_header_set	112
	auth_jwt_key_file	113
	auth_jwt_key_request	113
	auth_jwt_leeway	113
2.6.4	Embedded Variables	113

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. JWT claims must be encoded in a [JSON Web Signature](#) (JWS) structure. 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.

The module supports the following cryptographic [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.

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

2.6.2 Example Configuration

```
location / {
    auth_jwt          "closed site";
    auth_jwt_key_file conf/keys.json;
}
```


2.6.3 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.

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_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.

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. 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;
    }
}
```

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.

2.6.4 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.

2.7 Module ngx_http_auth_request_module

2.7.1	Summary	115
2.7.2	Example Configuration	115
2.7.3	Directives	115
	<code>auth_request</code>	115
	<code>auth_request_set</code>	116

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	117
2.8.2	Example Configuration	117
2.8.3	Directives	117
	autoindex	117
	autoindex_exact_size	117
	autoindex_format	117
	autoindex_localtime	118

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	119
2.9.2	Example Configuration	119
2.9.3	Directives	120
	ancient_browser	120
	ancient_browser_value	120
	modern_browser	120
	modern_browser_value	120

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	121
2.10.2 Example Configuration	121
2.10.3 Directives	121
charset	121
charset_map	122
charset_types	123
override_charset	123
source_charset	123

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/win-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	124
2.11.2 Example Configuration	124
2.11.3 Directives	124
create_full_put_path	124
dav_access	125
dav_methods	125
min_delete_depth	125

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	127
2.12.2 Example Configuration	127
2.12.3 Directives	127
empty_gif	127

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	128
2.13.2 Example Configuration	128
2.13.3 Directives	128
f4f	128
f4f_buffer_size	128

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	130
2.14.2	Example Configuration	130
2.14.3	Directives	130
	fastcgi_bind	130
	fastcgi_buffer_size	131
	fastcgi_buffering	131
	fastcgi_buffers	131
	fastcgi_busy_buffers_size	132
	fastcgi_cache	132
	fastcgi_cache_background_update	132
	fastcgi_cache_bypass	132
	fastcgi_cache_key	133
	fastcgi_cache_lock	133
	fastcgi_cache_lock_age	133
	fastcgi_cache_lock_timeout	133
	fastcgi_cache_max_range_offset	134
	fastcgi_cache_methods	134
	fastcgi_cache_min_uses	134
	fastcgi_cache_path	134
	fastcgi_cache_purge	136
	fastcgi_cache_revalidate	137
	fastcgi_cache_use_stale	137
	fastcgi_cache_valid	137
	fastcgi_catch_stderr	138
	fastcgi_connect_timeout	139
	fastcgi_force_ranges	139
	fastcgi_hide_header	139
	fastcgi_ignore_client_abort	139
	fastcgi_ignore_headers	140
	fastcgi_index	140
	fastcgi_intercept_errors	140
	fastcgi_keep_conn	141
	fastcgi_limit_rate	141
	fastcgi_max_temp_file_size	141
	fastcgi_next_upstream	141
	fastcgi_next_upstream_timeout	142
	fastcgi_next_upstream_tries	143
	fastcgi_no_cache	143
	fastcgi_param	143
	fastcgi_pass	144
	fastcgi_pass_header	144
	fastcgi_pass_request_body	144
	fastcgi_pass_request_headers	145
	fastcgi_read_timeout	145

fastcgi_request_buffering	145
fastcgi_send_lowat	145
fastcgi_send_timeout	146
fastcgi_socket_keepalive	146
fastcgi_split_path_info	146
fastcgi_store	146
fastcgi_store_access	147
fastcgi_temp_file_write_size	148
fastcgi_temp_path	148
2.14.4 Parameters Passed to a FastCGI Server	148
2.14.5 Embedded Variables	148

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*] [*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. When this size is exceeded, 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 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	150
2.15.2 Example Configuration	150
2.15.3 Directives	150
flv	150

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	151
2.16.2 Example Configuration	151
2.16.3 Directives	151
geo	151

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	154
2.17.2 Example Configuration	154
2.17.3 Directives	154
geoip_country	154
geoip_city	155
geoip_org	156
geoip_proxy	156
geoip_proxy_recursive	156

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	157
2.18.2	Example Configuration	157
2.18.3	Directives	158
	grpc_bind	158
	grpc_buffer_size	158
	grpc_connect_timeout	158
	grpc_hide_header	158
	grpc_ignore_headers	159
	grpc_intercept_errors	159
	grpc_next_upstream	159
	grpc_next_upstream_timeout	160
	grpc_next_upstream_tries	160
	grpc_pass	161
	grpc_pass_header	161
	grpc_read_timeout	161
	grpc_send_timeout	162
	grpc_set_header	162
	grpc_socket_keepalive	162
	grpc_ssl_certificate	162
	grpc_ssl_certificate_key	163
	grpc_ssl_ciphers	163
	grpc_ssl_crl	163
	grpc_ssl_name	163
	grpc_ssl_password_file	163
	grpc_ssl_protocols	164
	grpc_ssl_server_name	164
	grpc_ssl_session_reuse	164
	grpc_ssl_trusted_certificate	164
	grpc_ssl_verify	164
	grpc_ssl_verify_depth	164

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 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.

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*.

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_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	166
2.19.2 Example Configuration	166
2.19.3 Directives	166
gunzip	166
gunzip_buffers	166

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	167
2.20.2	Example Configuration	167
2.20.3	Directives	167
	gzip	167
	gzip_buffers	168
	gzip_comp_level	168
	gzip_disable	168
	gzip_http_version	168
	gzip_min_length	168
	gzip_proxied	169
	gzip_types	169
	gzip_vary	170
2.20.4	Embedded Variables	170

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	171
2.21.2 Example Configuration	171
2.21.3 Directives	171
gzip_static	171

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	172
2.22.2 Example Configuration	172
2.22.3 Directives	172
add_header	172
add_trailer	172
expires	173

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 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 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	175
2.23.2 Example Configuration	175
2.23.3 Directives	176
hls	176
hls_buffers	176
hls_forward_args	176
hls_fragment	177
hls_mp4_buffer_size	177
hls_mp4_max_buffer_size	178

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-
/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	179
2.24.2 Example Configuration	179
2.24.3 Directives	180
image_filter	180
image_filter_buffer	181
image_filter_interlace	181
image_filter_jpeg_quality	181
image_filter_sharpen	181
image_filter_transparency	181
image_filter_webp_quality	182

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	183
2.25.2 Example Configuration	183
2.25.3 Directives	183
index	183

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	184
2.26.2 Example Configuration	184
2.26.3 Directives	185
js_content	185
js_import	185
js_include	186
js_path	186
js_set	186
2.26.4 Request Argument	186

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;

    server {
        listen 8000;

        location / {
            add_header X-Foo $foo;
            js_content http.baz;
        }

        location = /summary {
            return 200 $summary;
        }

        location = /hello {
            js_content http.hello;
        }
    }
}
```

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!");
}

export default {foo, summary, baz, hello};

```

2.26.3 Directives

js_content

SYNTAX: **js_content** *function* | *module.function*;

DEFAULT —

CONTEXT: location, limit_except

Sets an `njs` function as a location content handler. Since 0.4.0, a module function can be referenced.

js_import

SYNTAX: **js_import** *module.js* | *export_name from module.js*;

DEFAULT —

CONTEXT: http

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 contains `foo()`, `http.foo` is used to refer to it.

Several `js_import` directives can be specified.

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 is deprecated since 0.4.0, the [js_import](#) directive should be used instead.

js_path

SYNTAX: **js_path** *path*;

DEFAULT —

CONTEXT: http

THIS DIRECTIVE APPEARED IN VERSION 0.3.0.

Sets an additional path for njs modules.

js_set

SYNTAX: **js_set** *\$variable function | module.function*;

DEFAULT —

CONTEXT: http

Sets an njs function for the specified variable. Since 0.4.0, a module function can be referenced.

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	187
2.27.2 Example Configuration	187
2.27.3 Directives	187
keyval	187
keyval_zone	188

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.

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	189
2.28.2 Example Configuration	189
2.28.3 Directives	189
limit_conn	189
limit_conn_dry_run	190
limit_conn_log_level	190
limit_conn_status	191
limit_conn_zone	191
limit_zone	191
2.28.4 Embedded Variables	192

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 level if and only if there are no `limit_conn` directives 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	193
2.29.2 Example Configuration	193
2.29.3 Directives	193
limit_req	193
limit_req_dry_run	194
limit_req_log_level	194
limit_req_status	195
limit_req_zone	195
2.29.4 Embedded Variables	196

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 level if and only if there are no `limit_req` directives 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	197
2.30.2 Example Configuration	197
2.30.3 Directives	197
access_log	197
log_format	199
open_log_file_cache	200

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 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 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	201
2.31.2 Example Configuration	201
2.31.3 Directives	201
map	201
map_hash_bucket_size	203
map_hash_max_size	203

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	204
2.32.2	Example Configuration	204
2.32.3	Directives	204
	memcached_bind	204
	memcached_buffer_size	205
	memcached_connect_timeout	205
	memcached_force_ranges	205
	memcached_gzip_flag	206
	memcached_next_upstream	206
	memcached_next_upstream_timeout	206
	memcached_next_upstream_tries	207
	memcached_pass	207
	memcached_read_timeout	207
	memcached_send_timeout	207
	memcached_socket_keepalive	208
2.32.4	Embedded Variables	208

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_force_ranges

SYNTAX: **memcached_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 memcached server regardless of the `Accept-Ranges` field in these responses.

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	209
2.33.2 Example Configuration	209
2.33.3 Directives	209
mirror	209
mirror_request_body	209

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 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	211
2.34.2 Example Configuration	212
2.34.3 Directives	212
mp4	212
mp4_buffer_size	212
mp4_max_buffer_size	212
mp4_limit_rate	213
mp4_limit_rate_after	213

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 Flash 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 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](#).

2.35 Module ngx_http_perl_module

2.35.1 Summary	214
2.35.2 Known Issues	214
2.35.3 Example Configuration	215
2.35.4 Directives	215
perl	215
perl_modules	216
perl_require	216
perl_set	216
2.35.5 Calling Perl from SSI	216
2.35.6 The \$r Request Object Methods	216

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	221
2.36.2	Example Configuration	221
2.36.3	Directives	222
	proxy_bind	222
	proxy_buffer_size	222
	proxy_buffering	222
	proxy_buffers	223
	proxy_busy_buffers_size	223
	proxy_cache	223
	proxy_cache_background_update	223
	proxy_cache_bypass	224
	proxy_cache_convert_head	224
	proxy_cache_key	224
	proxy_cache_lock	224
	proxy_cache_lock_age	225
	proxy_cache_lock_timeout	225
	proxy_cache_max_range_offset	225
	proxy_cache_methods	225
	proxy_cache_min_uses	226
	proxy_cache_path	226
	proxy_cache_purge	228
	proxy_cache_revalidate	228
	proxy_cache_use_stale	228
	proxy_cache_valid	229
	proxy_connect_timeout	230
	proxy_cookie_domain	230
	proxy_cookie_path	231
	proxy_force_ranges	232
	proxy_headers_hash_bucket_size	232
	proxy_headers_hash_max_size	232
	proxy_hide_header	232
	proxy_http_version	233
	proxy_ignore_client_abort	233
	proxy_ignore_headers	233
	proxy_intercept_errors	233
	proxy_limit_rate	234
	proxy_max_temp_file_size	234
	proxy_method	234
	proxy_next_upstream	235
	proxy_next_upstream_timeout	236
	proxy_next_upstream_tries	236
	proxy_no_cache	236
	proxy_pass	236
	proxy_pass_header	238

proxy_pass_request_body	238
proxy_pass_request_headers	238
proxy_read_timeout	239
proxy_redirect	239
proxy_request_buffering	240
proxy_send_lowat	241
proxy_send_timeout	241
proxy_set_body	241
proxy_set_header	241
proxy_socket_keepalive	242
proxy_ssl_certificate	242
proxy_ssl_certificate_key	243
proxy_ssl_ciphers	243
proxy_ssl_crl	243
proxy_ssl_name	243
proxy_ssl_password_file	244
proxy_ssl_protocols	244
proxy_ssl_server_name	244
proxy_ssl_session_reuse	244
proxy_ssl_trusted_certificate	244
proxy_ssl_verify	245
proxy_ssl_verify_depth	245
proxy_store	245
proxy_store_access	246
proxy_temp_file_write_size	246
proxy_temp_path	247
2.36.4 Embedded Variables	247

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*] [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. When this size is exceeded, 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;
```

There could be several `proxy_cookie_domain` directives:

```
proxy_cookie_domain localhost example.org;  
proxy_cookie_domain ~\.[a-z]+\.[a-z]+)$ $1;
```

The `off` parameter cancels the effect of all `proxy_cookie_domain` directives on the current level:

```
proxy_cookie_domain off;  
proxy_cookie_domain localhost example.org;  
proxy_cookie_domain www.example.org example.org;
```

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;
```

There could be several `proxy_cookie_path` directives:

```
proxy_cookie_path /one/ /;  
proxy_cookie_path / /two/;
```


The `off` parameter cancels the effect of all `proxy_cookie_path` directives on the current level:

```
proxy_cookie_path off;
proxy_cookie_path /two/ /;
proxy_cookie_path ~*^/user/([^/]+) /u/$1;
```

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;
```

There could be several `proxy_redirect` directives:

```
proxy_redirect default;
proxy_redirect http://localhost:8000/ /;
proxy_redirect http://www.example.com/ /;
```

The `off` parameter cancels the effect of all `proxy_redirect` directives on the current level:

```
proxy_redirect off;
proxy_redirect default;
proxy_redirect http://localhost:8000/ /;
proxy_redirect http://www.example.com/ /;
```

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 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.

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*.

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_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_random_index_module

2.37.1 Summary	248
2.37.2 Example Configuration	248
2.37.3 Directives	248
random_index	248

2.37.1 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.

2.37.2 Example Configuration

```
location / {
    random_index on;
}
```

2.37.3 Directives

random_index

SYNTAX: **random_index** on | off;

DEFAULT off

CONTEXT: location

Enables or disables module processing in a surrounding location.

2.38 Module ngx_http_realip_module

2.38.1 Summary	249
2.38.2 Example Configuration	249
2.38.3 Directives	249
set_real_ip_from	249
real_ip_header	249
real_ip_recursive	250
2.38.4 Embedded Variables	250

2.38.1 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.

2.38.2 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;
```

2.38.3 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.

2.38.4 Embedded Variables

\$realip_remote_addr

keeps the original client address (1.9.7)

\$realip_remote_port

keeps the original client port (1.11.0)

2.39 Module ngx_http_referer_module

2.39.1 Summary	251
2.39.2 Example Configuration	251
2.39.3 Directives	251
referer_hash_bucket_size	251
referer_hash_max_size	251
valid_referers	252
2.39.4 Embedded Variables	252

2.39.1 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.

2.39.2 Example Configuration

```
valid_referers none blocked server_names
               *.example.com example.* www.example.org/galleries/
               ~\.google\.;

if ($invalid_referer) {
    return 403;
}
```

2.39.3 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\.;
```

2.39.4 Embedded Variables

`$invalid_referer`

Empty string, if the `Referer` request header field value is considered [valid](#), otherwise “1”.

2.40 Module ngx_http_rewrite_module

2.40.1 Summary	253
2.40.2 Directives	253
break	253
if	254
return	255
rewrite	255
rewrite_log	257
set	257
uninitialized_variable_warn	257
2.40.3 Internal Implementation	257

2.40.1 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.

2.40.2 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.

2.40.3 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
```

2.41 Module ngx_http_scgi_module

2.41.1	Summary	260
2.41.2	Example Configuration	260
2.41.3	Directives	260
	scgi_bind	260
	scgi_buffer_size	260
	scgi_buffering	261
	scgi_buffers	261
	scgi_busy_buffers_size	261
	scgi_cache	262
	scgi_cache_background_update	262
	scgi_cache_bypass	262
	scgi_cache_key	262
	scgi_cache_lock	262
	scgi_cache_lock_age	263
	scgi_cache_lock_timeout	263
	scgi_cache_max_range_offset	263
	scgi_cache_methods	263
	scgi_cache_min_uses	264
	scgi_cache_path	264
	scgi_cache_purge	266
	scgi_cache_revalidate	266
	scgi_cache_use_stale	266
	scgi_cache_valid	267
	scgi_connect_timeout	268
	scgi_force_ranges	268
	scgi_hide_header	268
	scgi_ignore_client_abort	269
	scgi_ignore_headers	269
	scgi_intercept_errors	269
	scgi_limit_rate	269
	scgi_max_temp_file_size	270
	scgi_next_upstream	270
	scgi_next_upstream_timeout	271
	scgi_next_upstream_tries	271
	scgi_no_cache	272
	scgi_param	272
	scgi_pass	272
	scgi_pass_header	273
	scgi_pass_request_body	273
	scgi_pass_request_headers	273
	scgi_read_timeout	273
	scgi_request_buffering	274
	scgi_send_timeout	274
	scgi_socket_keepalive	274

scgi_store	274
scgi_store_access	275
scgi_temp_file_write_size	275
scgi_temp_path	276

2.41.1 Summary

The `ngx_http_scgi_module` module allows passing requests to an SCGI server.

2.41.2 Example Configuration

```
location / {
    include    scgi_params;
    scgi_pass  localhost:9000;
}
```

2.41.3 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*]
[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. When this size is exceeded, 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 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.

2.42 Module ngx_http_secure_link_module

2.42.1 Summary	277
2.42.2 Directives	277
secure_link	277
secure_link_md5	278
secure_link_secret	278
2.42.3 Embedded Variables	279

2.42.1 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.

2.42.2 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
```

2.42.3 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.

2.43 Module ngx_http_session_log_module

2.43.1 Summary	280
2.43.2 Example Configuration	280
2.43.3 Directives	280
session_log	280
session_log_format	280
session_log_zone	281
2.43.4 Embedded Variables	281

2.43.1 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](#).

2.43.2 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;
}
```

2.43.3 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 all `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.

2.43.4 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).

2.44 Module ngx_http_slice_module

2.44.1 Summary	282
2.44.2 Example Configuration	282
2.44.3 Directives	282
slice	282
2.44.4 Embedded Variables	282

2.44.1 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.

2.44.2 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.

2.44.3 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](#).

2.44.4 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.

2.45 Module ngx_http_split_clients_module

2.45.1 Summary	284
2.45.2 Example Configuration	284
2.45.3 Directives	284
split_clients	284

2.45.1 Summary

The `ngx_http_split_clients_module` module creates variables suitable for A/B testing, also known as split testing.

2.45.2 Example Configuration

```
http {
    split_clients "${remote_addr}AAA" $variant {
        0.5% .one;
        2.0% .two;
        *    "";
    }

    server {
        location / {
            index index${variant}.html;
        }
    }
}
```

2.45.3 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).

2.46 Module ngx_http_ssi_module

2.46.1	Summary	285
2.46.2	Example Configuration	285
2.46.3	Directives	285
	ssi	285
	ssi_last_modified	285
	ssi_min_file_chunk	286
	ssi_silent_errors	286
	ssi_types	286
	ssi_value_length	286
2.46.4	SSI Commands	286
2.46.5	Embedded Variables	290

2.46.1 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.

2.46.2 Example Configuration

```
location / {
    ssi on;
    ...
}
```

2.46.3 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.

2.46.4 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.

2.46.5 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.

2.47 Module ngx_http_ssl_module

2.47.1	Summary	291
2.47.2	Example Configuration	292
2.47.3	Directives	292
	ssl	292
	ssl_buffer_size	292
	ssl_certificate	293
	ssl_certificate_key	294
	ssl_ciphers	294
	ssl_client_certificate	294
	ssl_crl	294
	ssl_dhparam	295
	ssl_early_data	295
	ssl_ecdh_curve	295
	ssl_ocsp	296
	ssl_ocsp_cache	296
	ssl_ocsp_responder	296
	ssl_password_file	297
	ssl_prefer_server_ciphers	297
	ssl_protocols	297
	ssl_session_cache	298
	ssl_session_ticket_key	298
	ssl_session_tickets	299
	ssl_session_timeout	299
	ssl_stapling	299
	ssl_stapling_file	299
	ssl_stapling_responder	300
	ssl_stapling_verify	300
	ssl_trusted_certificate	300
	ssl_verify_client	300
	ssl_verify_depth	301
2.47.4	Error Processing	301
2.47.5	Embedded Variables	301

2.47.1 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.

2.47.2 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;

        ...
    }
}
```

2.47.3 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_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.

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 built with TLSv1.3 support is used.

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.

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.

2.47.4 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.

2.47.5 Embedded Variables

The `ngx_http_ssl_module` module supports embedded variables:

`$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_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).

2.48 Module ngx_http_status_module

2.48.1 Summary	304
2.48.2 Example Configuration	304
2.48.3 Directives	305
status	305
status_format	305
status_zone	306
2.48.4 Data	306
2.48.5 Compatibility	313

2.48.1 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.

2.48.2 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.

2.48.3 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.

2.48.4 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.

2.48.5 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](#).

2.49 Module ngx_http_stub_status_module

2.49.1 Summary	314
2.49.2 Example Configuration	314
2.49.3 Directives	314
stub_status	314
2.49.4 Data	315
2.49.5 Embedded Variables	315

2.49.1 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.

2.49.2 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
```

2.49.3 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`”.

2.49.4 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.

2.49.5 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.

2.50 Module ngx_http_sub_module

2.50.1 Summary	316
2.50.2 Example Configuration	316
2.50.3 Directives	316
sub_filter	316
sub_filter_last_modified	316
sub_filter_once	317
sub_filter_types	317

2.50.1 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.

2.50.2 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;
}
```

2.50.3 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 one configuration level (1.9.4). These directives are inherited from the previous 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).

2.51 Module ngx_http_upstream_module

2.51.1 Summary	318
2.51.2 Example Configuration	318
2.51.3 Directives	319
upstream	319
server	319
zone	322
state	322
hash	323
ip_hash	323
keepalive	324
keepalive_requests	325
keepalive_timeout	326
ntlm	326
least_conn	327
least_time	327
queue	327
random	328
resolver	328
resolver_timeout	329
sticky	329
sticky_cookie_insert	332
2.51.4 Embedded Variables	332

2.51.1 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.

2.51.2 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;
    }
}
```

2.51.3 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.

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;
        ...
    }
}
```

When using load balancer methods other than the default round-robin method, it is necessary to activate them before the `keepalive` directive.

SCGI and uwsgi protocols do not have a notion of keepalive connections.

keepalive_requests

SYNTAX: **keepalive_requests** *number*;

DEFAULT 100

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.

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*] [*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 IPv6 addresses is not desired, 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*] [*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.

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).

`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];
```

2.51.4 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).

2.52 Module ngx_http_upstream_conf_module

2.52.1 Summary	334
2.52.2 Example Configuration	334
2.52.3 Directives	334
upstream_conf	334

2.52.1 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.

2.52.2 Example Configuration

```
upstream backend {
    zone upstream_backend 64k;

    ...
}

server {
    location /upstream_conf {
        upstream_conf;
        allow 127.0.0.1;
        deny all;
    }
}
```

2.52.3 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.

2.53 Module ngx_http_upstream_hc_module

2.53.1 Summary	338
2.53.2 Example Configuration	338
2.53.3 Directives	339
health_check	339
match	340

2.53.1 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](#).

2.53.2 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.

2.53.3 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

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.

`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.

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;  
}
```

2.54 Module ngx_http_userid_module

2.54.1 Summary	342
2.54.2 Example Configuration	342
2.54.3 Directives	342
userid	342
userid_domain	343
userid_expires	343
userid_mark	343
userid_name	343
userid_p3p	344
userid_path	344
userid_service	344
2.54.4 Embedded Variables	344

2.54.1 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.

2.54.2 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"';
```

2.54.3 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_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.

2.54.4 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.

2.55 Module ngx_http_uwsgi_module

2.55.1	Summary	346
2.55.2	Example Configuration	346
2.55.3	Directives	346
	uwsgi_bind	346
	uwsgi_buffer_size	347
	uwsgi_buffering	347
	uwsgi_buffers	347
	uwsgi_busy_buffers_size	348
	uwsgi_cache	348
	uwsgi_cache_background_update	348
	uwsgi_cache_bypass	348
	uwsgi_cache_key	349
	uwsgi_cache_lock	349
	uwsgi_cache_lock_age	349
	uwsgi_cache_lock_timeout	349
	uwsgi_cache_max_range_offset	350
	uwsgi_cache_methods	350
	uwsgi_cache_min_uses	350
	uwsgi_cache_path	350
	uwsgi_cache_purge	352
	uwsgi_cache_revalidate	353
	uwsgi_cache_use_stale	353
	uwsgi_cache_valid	353
	uwsgi_connect_timeout	354
	uwsgi_force_ranges	355
	uwsgi_hide_header	355
	uwsgi_ignore_client_abort	355
	uwsgi_ignore_headers	355
	uwsgi_intercept_errors	356
	uwsgi_limit_rate	356
	uwsgi_max_temp_file_size	356
	uwsgi_modifier1	356
	uwsgi_modifier2	357
	uwsgi_next_upstream	357
	uwsgi_next_upstream_timeout	358
	uwsgi_next_upstream_tries	358
	uwsgi_no_cache	358
	uwsgi_param	358
	uwsgi_pass	359
	uwsgi_pass_header	359
	uwsgi_pass_request_body	360
	uwsgi_pass_request_headers	360
	uwsgi_read_timeout	360
	uwsgi_request_buffering	360

uwsgi_send_timeout	361
uwsgi_socket_keepalive	361
uwsgi_ssl_certificate	361
uwsgi_ssl_certificate_key	361
uwsgi_ssl_ciphers	361
uwsgi_ssl_crl	362
uwsgi_ssl_name	362
uwsgi_ssl_password_file	362
uwsgi_ssl_protocols	362
uwsgi_ssl_server_name	363
uwsgi_ssl_session_reuse	363
uwsgi_ssl_trusted_certificate	363
uwsgi_ssl_verify	363
uwsgi_ssl_verify_depth	363
uwsgi_store	364
uwsgi_store_access	364
uwsgi_temp_file_write_size	365
uwsgi_temp_path	365

2.55.1 Summary

The `ngx_http_uwsgi_module` module allows passing requests to a uwsgi server.

2.55.2 Example Configuration

```
location / {
    include    uwsgi_params;
    uwsgi_pass localhost:9000;
}
```

2.55.3 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*] [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. When this size is exceeded, 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 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.

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*.

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_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.

2.56 Module ngx_http_v2_module

2.56.1	Summary	366
2.56.2	Known Issues	366
2.56.3	Example Configuration	366
2.56.4	Directives	367
	http2_body_preread_size	367
	http2_chunk_size	367
	http2_idle_timeout	367
	http2_max_concurrent_pushes	367
	http2_max_concurrent_streams	367
	http2_max_field_size	368
	http2_max_header_size	368
	http2_max_requests	368
	http2_push	368
	http2_push_preload	369
	http2_recv_buffer_size	369
	http2_recv_timeout	369
2.56.5	Embedded Variables	369

2.56.1 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.

2.56.2 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.

2.56.3 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 only since [OpenSSL](#) version 1.0.2. Using the “Next Protocol Negotiation” (NPN) TLS extension for this purpose (available since OpenSSL version 1.0.1) is not guaranteed to work.

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.

2.56.4 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

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

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

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.

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

Sets the timeout for expecting more data from the client, after which the connection is closed.

2.56.5 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.

2.57 Module ngx_http_xslt_module

2.57.1 Summary	370
2.57.2 Example Configuration	370
2.57.3 Directives	370
xml_entities	370
xslt_last_modified	371
xslt_param	371
xslt_string_param	371
xslt_stylesheet	371
xslt_types	372

2.57.1 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.

2.57.2 Example Configuration

```
location / {
    xml_entities    /site/dtd/entities.dtd;
    xslt_stylesheet /site/xslt/one.xslt param=value;
    xslt_stylesheet /site/xslt/two.xslt;
}
```

2.57.3 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 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 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	373
3.1.2	Example Configuration	373
3.1.3	Directives	374
	listen	374
	preread_buffer_size	376
	preread_timeout	376
	proxy_protocol_timeout	377
	resolver	377
	resolver_timeout	377
	server	378
	stream	378
	tcp_nodelay	378
	variables_hash_bucket_size	378
	variables_hash_max_size	378
3.1.4	Embedded Variables	379

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*]
 [*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.

`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 `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.

`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_KEEPIIDLE`, `TCP_KEEPIINTVL`, 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.

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*] [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 IPv6 addresses is not desired, 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.

\$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	381
3.2.2	Example Configuration	381
3.2.3	Directives	381
	allow	381
	deny	381

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	382
3.3.2	Example Configuration	382
3.3.3	Directives	382
	geo	382

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	384
3.4.2	Example Configuration	384
3.4.3	Directives	384
	geoip_country	384
	geoip_city	385
	geoip_org	386

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	387
3.5.2	Example Configuration	387
3.5.3	Directives	388
	js_access	388
	js_filter	389
	js_import	389
	js_include	389
	js_path	389
	js_preread	390
	js_set	390
3.5.4	Session Object Properties	390

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.abort();
        return;
    }

    s.allow();
}

export default {bar, preread, req_line, 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.

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.

js_import

SYNTAX: **js_import** *module.js* | *export_name from module.js*;

DEFAULT —

CONTEXT: stream

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 contains `foo()`, `stream.foo` is used to refer to it.

Several `js_import` directives can be specified.

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 is deprecated since 0.4.0, the [js_import](#) directive should be used instead.

js_path

SYNTAX: **js_path** *path*;

DEFAULT —

CONTEXT: http

THIS DIRECTIVE APPEARED IN VERSION 0.3.0.

Sets an additional path for njs modules.

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.

js_set

SYNTAX: **js_set** *\$variable function* | *module.function*;

DEFAULT —

CONTEXT: stream

Sets an njs function for the specified variable. Since 0.4.0, a module function can be referenced.

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	391
3.6.2	Example Configuration	391
3.6.3	Directives	391
	keyval	391
	keyval_zone	392

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](#).

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.

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	393
3.7.2	Example Configuration	393
3.7.3	Directives	393
	limit_conn	393
	limit_conn_dry_run	394
	limit_conn_log_level	394
	limit_conn_zone	394
3.7.4	Embedded Variables	395

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.

The directives are inherited from the previous level if and only if there are no `limit_conn` directives 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	396
3.8.2	Example Configuration	396
3.8.3	Directives	396
	access_log	396
	log_format	397
	open_log_file_cache	398

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 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	399
3.9.2	Example Configuration	399
3.9.3	Directives	399
	map	399
	map_hash_bucket_size	400
	map_hash_max_size	401

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	402
3.10.2	Example Configuration	402
3.10.3	Directives	403
	proxy_bind	403
	proxy_buffer_size	403
	proxy_connect_timeout	404
	proxy_download_rate	404
	proxy_next_upstream	404
	proxy_next_upstream_timeout	404
	proxy_next_upstream_tries	405
	proxy_pass	405
	proxy_protocol	405
	proxy_requests	405
	proxy_responses	406
	proxy_session_drop	406
	proxy_socket_keepalive	406
	proxy_ssl	407
	proxy_ssl_certificate	407
	proxy_ssl_certificate_key	407
	proxy_ssl_ciphers	407
	proxy_ssl_crl	407
	proxy_ssl_name	407
	proxy_ssl_password_file	408
	proxy_ssl_protocols	408
	proxy_ssl_server_name	408
	proxy_ssl_session_reuse	408
	proxy_ssl_trusted_certificate	408
	proxy_ssl_verify	409
	proxy_ssl_verify_depth	409
	proxy_timeout	409
	proxy_upload_rate	409

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_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.

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.

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_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_realip_module

3.11.1 Summary	410
3.11.2 Example Configuration	410
3.11.3 Directives	410
set_real_ip_from	410
3.11.4 Embedded Variables	410

3.11.1 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.

3.11.2 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;
```

3.11.3 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.

3.11.4 Embedded Variables

\$realip_remote_addr

keeps the original client address

\$realip_remote_port

keeps the original client port

3.12 Module ngx_stream_return_module

3.12.1 Summary	411
3.12.2 Example Configuration	411
3.12.3 Directives	411
return	411

3.12.1 Summary

The ngx_stream_return_module module (1.11.2) allows sending a specified value to the client and then closing the connection.

3.12.2 Example Configuration

```
server {
    listen 12345;
    return $time_iso8601;
}
```

3.12.3 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.

3.13 Module ngx_stream_split_clients_module

3.13.1 Summary	412
3.13.2 Example Configuration	412
3.13.3 Directives	412
split_clients	412

3.13.1 Summary

The `ngx_stream_split_clients_module` module (1.11.3) creates variables suitable for A/B testing, also known as split testing.

3.13.2 Example Configuration

```
stream {
    ...
    split_clients "${remote_addr}AAA" $upstream {
        0.5%      feature_test1;
        2.0%      feature_test2;
        *         production;
    }

    server {
        ...
        proxy_pass $upstream;
    }
}
```

3.13.3 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).

3.14 Module ngx_stream_ssl_module

3.14.1	Summary	413
3.14.2	Example Configuration	413
3.14.3	Directives	414
	ssl_certificate	414
	ssl_certificate_key	415
	ssl_ciphers	415
	ssl_client_certificate	415
	ssl_crl	416
	ssl_dhparam	416
	ssl_ecdh_curve	416
	ssl_handshake_timeout	416
	ssl_password_file	417
	ssl_prefer_server_ciphers	417
	ssl_protocols	417
	ssl_session_cache	418
	ssl_session_ticket_key	418
	ssl_session_tickets	419
	ssl_session_timeout	419
	ssl_trusted_certificate	419
	ssl_verify_client	419
	ssl_verify_depth	420
3.14.4	Embedded Variables	420

3.14.1 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.

3.14.2 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;

    ...
}
```

3.14.3 Directives

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_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.

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 built with TLSv1.3 support 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.

3.14.4 Embedded Variables

The `ngx_stream_ssl_module` module supports variables since 1.11.2.

\$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_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.

3.15 Module ngx_stream_ssl_preread_module

3.15.1 Summary	422
3.15.2 Example Configuration	422
3.15.3 Directives	423
ssl_preread	423
3.15.4 Embedded Variables	423

3.15.1 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.

3.15.2 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;
}
```

3.15.3 Directives

ssl_preread

SYNTAX: **ssl_preread** on | off;

DEFAULT off

CONTEXT: stream, server

Enables extracting information from the ClientHello message at the preread phase.

3.15.4 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.

3.16 Module ngx_stream_upstream_module

3.16.1 Summary	424
3.16.2 Example Configuration	424
3.16.3 Directives	425
upstream	425
server	425
zone	427
state	428
hash	428
least_conn	429
least_time	429
random	429
resolver	430
resolver_timeout	431
3.16.4 Embedded Variables	431

3.16.1 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.

3.16.2 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;
}
```

3.16.3 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*] [*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 IPv6 addresses is not desired, 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](#).

3.16.4 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.

3.17 Module ngx_stream_upstream_hc_module

3.17.1 Summary	432
3.17.2 Example Configuration	432
3.17.3 Directives	433
health_check	433
health_check_timeout	434
match	434

3.17.1 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](#).

3.17.2 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.

3.17.3 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

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.

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.

3.18 Module ngx_stream_zone_sync_module

3.18.1 Summary	436
3.18.2 Example Configuration	436
3.18.3 Directives	437
zone_sync	437
zone_sync_buffers	438
zone_sync_connect_retry_interval	438
zone_sync_connect_timeout	438
zone_sync_interval	438
zone_sync_recv_buffer_size	438
zone_sync_server	439
zone_sync_ssl	439
zone_sync_ssl_certificate	439
zone_sync_ssl_certificate_key	440
zone_sync_ssl_ciphers	440
zone_sync_ssl_crl	440
zone_sync_ssl_name	440
zone_sync_ssl_password_file	440
zone_sync_ssl_protocols	441
zone_sync_ssl_server_name	441
zone_sync_ssl_trusted_certificate	441
zone_sync_ssl_verify	441
zone_sync_ssl_verify_depth	441
zone_sync_timeout	441
3.18.4 API endpoints	442
3.18.5 Starting, stopping, removing a cluster node	442

3.18.1 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](#).

3.18.2 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;
    }
}

```

3.18.3 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_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.

3.18.4 API endpoints

The synchronization status of a node is available via the [/stream/zone-sync/](#) endpoint of the API which returns the [following](#) metrics.

3.18.5 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	443
4.1.2	Example Configuration	443
4.1.3	Directives	444
	listen	444
	mail	446
	protocol	446
	resolver	446
	resolver_timeout	447
	server	447
	server_name	447
	timeout	448

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 1;

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*] [*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 *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 `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_KEEPIDLE`) to 30 minutes, leave the probe interval (`TCP_KEEPINTVL`) 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.

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*] [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 IPv6 addresses is not desired, 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	449
auth_http	449
auth_http_header	449
auth_http_pass_client_cert	449
auth_http_timeout	449
4.2.2 Protocol	450

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 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-Verify: SUCCESS
Auth-SSL-Subject: /CN=example.com
Auth-SSL-Issuer: /CN=example.com
Auth-SSL-Serial: C07AD56B846B5BFF
Auth-SSL-Fingerprint: 29d6a80a123d13355ed16b4b04605e29cb55a5ad
```

4.3 Module ngx_mail_proxy_module

4.3.1 Directives	453
proxy_buffer	453
proxy_pass_error_message	453
proxy_timeout	453
xclient	454

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_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_ssl_module

4.4.1	Summary	455
4.4.2	Example Configuration	455
4.4.3	Directives	456
	ssl	456
	ssl_certificate	456
	ssl_certificate_key	457
	ssl_ciphers	457
	ssl_client_certificate	457
	ssl_crl	458
	ssl_dhparam	458
	ssl_ecdh_curve	458
	ssl_password_file	458
	ssl_prefer_server_ciphers	459
	ssl_protocols	459
	ssl_session_cache	459
	ssl_session_ticket_key	460
	ssl_session_tickets	461
	ssl_session_timeout	461
	ssl_trusted_certificate	461
	ssl_verify_client	461
	ssl_verify_depth	461
	starttls	462

4.4.1 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.

4.4.2 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;

        ...
    }
}
```

4.4.3 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_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.

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 built with TLSv1.3 support 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.

4.5 Module ngx_mail_imap_module

4.5.1 Directives	463
imap_auth	463
imap_capabilities	463
imap_client_buffer	463

4.5.1 Directives

imap_auth

SYNTAX: **imap_auth** *method* ...;

DEFAULT plain

CONTEXT: mail, server

Sets permitted methods of authentication for IMAP clients. Supported methods are:

login

[AUTH=LOGIN](#)

plain

[AUTH=PLAIN](#)

cram-md5

[AUTH=CRAM-MD5](#). In order for this method to work, the password must be stored unencrypted.

external

[AUTH=EXTERNAL](#) (1.11.6).

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.

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.

4.6 Module ngx_mail_pop3_module

4.6.1 Directives	465
pop3_auth	465
pop3_capabilities	465

4.6.1 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](#). It is not possible to disable these methods.

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).

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.

4.7 Module ngx_mail_smtp_module

4.7.1 Directives	466
smtp_auth	466
smtp_capabilities	466
smtp_client_buffer	467
smtp_greeting_delay	467

4.7.1 Directives

smtp_auth

SYNTAX: **smtp_auth** *method* ...;

DEFAULT login plain

CONTEXT: mail, server

Sets permitted methods of [SASL authentication](#) for SMTP clients. Supported methods are:

login

[AUTH LOGIN](#)

plain

[AUTH PLAIN](#)

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.

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.

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 High Availability support for NGINX Plus

5.1.1	High Availability support	468
5.1.2	Configuring HA setup	469
5.1.3	Check scripts	470
5.1.4	Checking the status of HA setup	471
5.1.5	Forcing state change	471
5.1.6	Adding more virtual IP addresses	471
5.1.7	Troubleshooting keepalived and VRRP	472
5.1.8	Miscellaneous	473

5.1.1 High Availability support

NGINX-HA-Keepalived is a solution for fast and easy configuration of NGINX Plus in an active-passive high-availability (HA) setup. It is based on keepalived.

The keepalived project provides a keepalive facility for Linux servers, an implementation of the VRRP protocol to manage virtual routers (virtual IP addresses), and a health check facility to determine if a service (web server, PHP back end, database server, etc.) is up and operational. If a service on a node fails a configurable number of health checks, keepalived reassigns the virtual IP address of the node to a secondary node.

The VRRP protocol ensures that one of participating nodes is master. The backup node listens for VRRP advertisement packets from the master node. If it does not receive an advertisement packet for a period longer than three times the configured advertisement interval, the backup node takes over as master and assigns the configured virtual IP addresses to itself.

5.1.2 Configuring HA setup

Run the `nginx-ha-setup` script (available in the `nginx-ha-keepalived` package, must be installed separately) on both nodes as the root user.

The script configures a high-availability NGINX Plus environment with an active-passive pair of nodes acting as master and backup. It prompts for the following data:

- IP address of the local and remote nodes (one of which will be configured as a master, the other one as a backup).
- One free IP address to be used as the cluster endpoint's (floating) virtual IP address.

The configuration of the keepalived daemon is recorded in a text file, `/etc-keepalived/keepalived.conf`. The configuration blocks in the file control notification settings, the virtual IP addresses to manage, and the health checks to use to test the services that rely on virtual IP addresses. Following is the configuration created by the `nginx-ha-setup` script on a CentOS 7 machine:

```
vrrp_script chk_nginx_service {
    script "/usr/libexec/keepalived/nginx-ha-check"
    interval 3
    weight 50
}

vrrp_instance VI_1 {
    interface eth0
    state BACKUP
    priority 101
    virtual_router_id 51
    advert_int 1
    unicast_src_ip 192.168.100.100
    unicast_peer {
        192.168.100.101
    }
    authentication {
        auth_type PASS
        auth_pass f8f0e5114cbe031a3e1e622daf18f82a
    }
    virtual_ipaddress {
        192.168.100.150
    }
    track_script {
        chk_nginx_service
    }
    notify "/usr/libexec/keepalived/nginx-ha-notify"
}
```

The configuration shown above is self-explanatory, but a few items are worth noting:

- Each node in the HA setup needs its own copy of the configuration file, with values for the `priority`, `unicast_src_ip`, and `unicast_peer` directives that are appropriate to the node's status (master or backup).

- The `priority` directive controls which host becomes the master, as explained in the next section.
- The `notify` directive names the notification script included in the distribution, which can be used to generate syslog messages (or other notifications) when a state transition or fault occurs.
- The value 51 for the `virtual_router_id` directive in the `vrrp_instance VI_1` block is a sample value.
- If you have multiple pairs of keepalived instances (or other VRRP instances) running in your local network, create a `vrrp_instance` block for each one, with a unique name (like `VI_1` in the sample) and `virtual_router_id` number.

5.1.3 Check scripts

There is no fencing mechanism in keepalived. If the two nodes in a pair are not aware of each other, each assumes it is the master and assigns the virtual IP address to itself. To prevent this situation, the `chk_nginx_service` script is executed regularly to check its exit code and adjust the node's priority as necessary. Code 0 indicates correct operation, and code 1 (or any nonzero code) indicates an error.

In the default configuration of the `chk_nginx_service` script, the `weight` directive is set to 50, which means that when the check script succeeds:

- The priority of the first node (which has a base priority of 101) is set to 151.
- The priority of the second node (which has a base priority of 100) is set to 150.

The first node has higher priority (151 in this case) and becomes master.

Use the `interval` directive to specify how often the check script executes, in seconds (it is set to 3 in the default configuration). Note that the check also fails when the timeout is reached (by default, the timeout is the same as the check interval).

Use the `rise` and `fall` directives to specify how many times the script must succeed or fail before action is taken (they are not set in the default configuration).

The default script provided with the `nginx-ha-keepalived` package checks if nginx is up. We recommend creating additional scripts as appropriate for your local setup.

5.1.4 Checking the status of HA setup

To see which node is currently the master for a given virtual IP address, run the `ip addr show` command for the interface on which the `vrp_instance` is defined (in the following commands, interface `eth0`):

```
centos7-1 # ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
    link/ether 52:54:00:33:a5:a5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.100/24 brd 192.168.122.255 scope global dynamic eth0
        valid_lft 3071sec preferred_lft 3071sec
    inet 192.168.100.150/32 scope global eth0
        valid_lft forever preferred_lft forever

centos7-2 # ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
    link/ether 52:54:00:33:a5:87 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.101/24 brd 192.168.122.255 scope global eth0
        valid_lft forever preferred_lft forever
```

In this output, the defined virtual IP address (192.168.100.150) is currently assigned to the host with real IP address of 192.168.100.100.

When a host's HA state changes, `nginx-ha-keepalived` writes it to the `/var/run/nginx-ha-keepalived.state` file:

```
centos7-1 # cat /var/run/nginx-ha-keepalived.state
STATE=MASTER

centos7-2 # cat /var/run/nginx-ha-keepalived.state
STATE=BACKUP
```

5.1.5 Forcing state change

To force the master node to switch to backup state, run the following command on it:

```
# service keepalived stop
```

As it shuts down, `keepalived` sends a VRRP packet with priority 0 to the backup node, which causes the backup node to take over the virtual IP address.

5.1.6 Adding more virtual IP addresses

The configuration created by `nginx-ha-setup` is very basic, and makes a single IP address highly available. To make more than one IP address highly available, add each new IP address to the `virtual_ipaddress` block in the `/etc/keepalived/keepalived.conf` configuration file. Then run the `service keepalived reload` command on both nodes to reload the `keepalived` service:

```
virtual_ipaddress {
    192.168.100.150
```



```

    192.168.100.200
    1234:5678:9abc:def::1/64
}

```

As indicated in this example, keepalived can be utilized in dual-stack IPv4/IPv6 environments to fail over both IPv4 and IPv6 addresses.

The syntax in the `virtualipaddress` block replicates the syntax of the `ip` utility.

5.1.7 Troubleshooting keepalived and VRRP

The keepalived daemon logs to syslog. On CentOS, RHEL, and SLES-based systems, the output is typically written to `/var/log/messages`, whereas on Ubuntu and Debian-based systems it is written to `/var/log/syslog`. Log entries record events such as startup of the keepalived daemon and state transitions. Here are a few sample entries that show the keepalived daemon starting up, and the node transitioning a VRRP instance to the master state:

```

Feb 27 14:42:04 centos7-1 systemd: Starting LVS and VRRP High Availability
Monitor...
Feb 27 14:42:04 centos7-1 Keepalived[19242]: Starting Keepalived v1.2.15
(02/26,2015)
Feb 27 14:42:04 centos7-1 Keepalived[19243]: Starting VRRP child process, pid
=19244
Feb 27 14:42:04 centos7-1 Keepalived_vrrp[19244]: Registering Kernel netlink
reflector
Feb 27 14:42:04 centos7-1 Keepalived_vrrp[19244]: Registering Kernel netlink
command channel
Feb 27 14:42:04 centos7-1 Keepalived_vrrp[19244]: Registering gratuitous ARP
shared channel
Feb 27 14:42:05 centos7-1 systemd: Started LVS and VRRP High Availability
Monitor.
Feb 27 14:42:05 centos7-1 Keepalived_vrrp[19244]: Opening file '/etc/keepalived
/keepalived.conf'.
Feb 27 14:42:05 centos7-1 Keepalived_vrrp[19244]: Truncating auth_pass to 8
characters
Feb 27 14:42:05 centos7-1 Keepalived_vrrp[19244]: Configuration is using :
64631 Bytes
Feb 27 14:42:05 centos7-1 Keepalived_vrrp[19244]: Using LinkWatch kernel
netlink reflector...
Feb 27 14:42:05 centos7-1 Keepalived_vrrp[19244]: VRRP_Instance(VI_1) Entering
BACKUP STATE
Feb 27 14:42:05 centos7-1 Keepalived_vrrp[19244]: VRRP sockpool: [ifindex(2),
proto(112), unicast(1), fd(14,15)]
Feb 27 14:42:05 centos7-1 nginx-ha-keepalived: Transition to state 'BACKUP' on
VRRP instance 'VI_1'.
Feb 27 14:42:05 centos7-1 Keepalived_vrrp[19244]: VRRP_Script(chk_nginx_service
) succeeded
Feb 27 14:42:06 centos7-1 Keepalived_vrrp[19244]: VRRP_Instance(VI_1) forcing a
new MASTER election
Feb 27 14:42:06 centos7-1 Keepalived_vrrp[19244]: VRRP_Instance(VI_1) forcing a
new MASTER election
Feb 27 14:42:07 centos7-1 Keepalived_vrrp[19244]: VRRP_Instance(VI_1)
Transition to MASTER STATE
Feb 27 14:42:08 centos7-1 Keepalived_vrrp[19244]: VRRP_Instance(VI_1) Entering
MASTER STATE
Feb 27 14:42:08 centos7-1 Keepalived_vrrp[19244]: VRRP_Instance(VI_1) setting
protocol VIPs.
Feb 27 14:42:08 centos7-1 Keepalived_vrrp[19244]: VRRP_Instance(VI_1) Sending
gratuitous ARPs on eth0 for 192.168.100.150
Feb 27 14:42:08 centos7-1 nginx-ha-keepalived: Transition to state 'MASTER' on
VRRP instance 'VI_1'.

```

```
Feb 27 14:42:13 centos7-1 Keepalived_vrrp[19244]: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 for 192.168.100.150
```

If the system log does not explain the source of a problem, run the `tcpdump` command with the following parameters to display the VRRP advertisements that are sent on the local network:

```
# tcpdump -vvv -ni eth0 proto vrrp
```

If you have multiple VRRP instances on the local network and want to filter the traffic for select hosts, include the `host` parameter to specify the IP address that is defined in the `unicast_peer` block, as in the following example:

```
centos7-1 # tcpdump -vvv -ni eth0 proto vrrp and host 192.168.100.101
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
14:48:27.188100 IP (tos 0xc0, ttl 255, id 382, offset 0, flags [none], proto VRRP (112), length 40)
  192.168.100.100 > 192.168.100.101: vrrp 192.168.100.100 > 192.168.100.101: VRRPv2, Advertisement, vrid 51, prio 151, authtype simple, intvl 1s, length 20, addrs: 192.168.100.150 auth "f8f0e511"
```

Several fields in the output are useful for debugging:

- `authtype` - the type of authentication in use (`authentication directive`)
- `vrid` - the virtual router ID (`virtual_router_id directive`)
- `prio` - the node's priority (`priority directive`)
- `intvl` - the frequency at which advertisements are sent (`advert_int directive`)
- `auth` - the authentication token sent (`auth_pass directive`)

5.1.8 Miscellaneous

Note that NGINX configuration files on both nodes must define the services that are being made highly available. Keeping the configuration files in sync is outside the scope of the provided clustering software.

The `nginx-ha-keepalived` package comes with numerous configuration examples, in the `/usr/share/doc/nginx-ha-keepalived/` directory. They show how to configure numerous aspects of an HA setup.

5.2 Command-line parameters

5.2.1 Overview 474

5.2.1 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.
- `-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 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 `$ssl_client_i_dn` and `$ssl_client_s_dn` variables are now compliant with [RFC 2253](#); legacy variants are available as `$ssl_client_i_dn_legacy` and `$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 healthchecks.
- 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` (1.19.0), distributed under 2-clause BSD license.
<https://nginx.org/>

Copyright © 2002-2020 Igor Sysoev

Copyright © 2011-2020 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 (used only if no system MD5 support was found), 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:

- autoprefixer, a PostCSS plugin to parse CSS and add vendor prefixes to CSS rules (7.2.6).
<https://github.com/postcss/autoprefixer>
Copyright © 2013 Andrey Sitnik <andrey@sitnik.ru>
- babel-core, Babel compiler core (6.26.3).
<https://github.com/babel/babel/tree/master/packages/babel-core>
Copyright © 2014-present Sebastian McKenzie and other contributors
- babel-loader, allows transpiling JavaScript files using Babel and webpack (7.1.5).
<https://github.com/babel/babel-loader>
Copyright © 2014-2019 Luís Couto <hello@luiscouto.pt>
- babel-plugin-transform-object-rest-spread, produces spec-compliant code by using Babel's objectSpread helper (6.26.0).
<https://github.com/babel/babel/tree/master/packages/babel-plugin-transform-object-rest-spread>
- babel-plugin-transform-runtime, makes helpers reference the module babel-runtime to avoid duplication across your compiled output (6.23.0).
<https://github.com/babel/babel/tree/master/packages/babel-plugin-transform-runtime>
- babel-polyfill, provides polyfills necessary for a full ES2015+ environment (6.26.0).
<https://github.com/babel/babel/tree/master/packages/babel-polyfill>
Copyright © 2014-2017 Sebastian McKenzie <sebmck@gmail.com>
- css-loader, interprets @import and url() like import/require() and will resolve them (0.28.11).
<https://github.com/webpack-contrib/css-loader>
Copyright © JS Foundation and other contributors

- `cssnano`, a modular minifier, built on top of the PostCSS ecosystem (3.10.0).
<https://github.com/cssnano/cssnano>
Copyright © Ben Briggs <beneb.info@gmail.com>
- `extract-text-webpack-plugin`, a lightweight CSS extraction plugin (3.0.2).
<https://github.com/webpack-contrib/extract-text-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 2016-2018
- `html-webpack-harddisk-plugin`, writes html files to hard disk even when using the webpack dev server or middleware (0.1.0).
<https://github.com/jantimon/html-webpack-harddisk-plugin>
Copyright © 2016 Jan Nicklas
- `html-webpack-inline-source-plugin`, allows embedding javascript and css sources inline (0.0.9).
<https://github.com/dustinjackson/html-webpack-inline-source-plugin>
Copyright © 2016 Jan Nicklas
- `html-webpack-plugin`, simplifies creation of HTML files to serve your webpack bundles (2.30.1).
<https://github.com/jantimon/html-webpack-plugin>
Copyright © JS Foundation and other contributors
- `postcss-import`, PostCSS plugin to inline `@import` rules content (11.1.0).
<https://github.com/postcss/postcss-import>
Copyright © 2014 Maxime Thirouin, Jason Campbell & Kevin Martensson
- `postcss-inline-svg`, PostCSS plugin to reference an SVG file and control its attributes with CSS syntax (3.1.1).
<https://github.com/TrySound/postcss-inline-svg>
Copyright © Bogdan Chadkin <trysound@yandex.ru>
- `postcss-loader`, PostCSS loader for webpack (2.1.6).
<https://github.com/postcss/postcss-loader>
Copyright © 2017 Andrey Sitnik <andrey@sitnik.ru>
- `postcss-svgo`, a modular minifier, built on top of the PostCSS ecosystem (2.1.6).
<https://github.com/cssnano/cssnano>
Copyright © Ben Briggs <beneb.info@gmail.com>
- `postcss-url`, PostCSS plugin to rebase `url()`, inline or copy asset (7.3.2).
<https://github.com/postcss/postcss-url>
Copyright © 2014 Maxime Thirouin
- `preact`, fast 3kb React alternative with the same ES6 API (8.5.3).
<https://github.com/developit/preact>
Copyright © 2015-present Jason Miller
- `preact-compat`, React compatibility layer for Preact (3.19.0).
<https://github.com/preactjs/preact-compat>
Copyright © 2016 Jason Miller
- `style-loader`, injects CSS into the DOM (0.18.2).
<https://github.com/webpack-contrib/style-loader>
Copyright © JS Foundation and other contributors

- uglifyjs-webpack-plugin, UglifyJS Plugin (0.4.6).
<https://github.com/webpack-contrib/uglifyjs-webpack-plugin>
Copyright © JS Foundation and other contributors
- webpack, a bundler for javascript and friends (3.12.0).
<https://github.com/webpack/webpack>
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:

- babel-plugin-react-css-modules, transforms styleName to className using compile time CSS module resolution (3.4.2), distributed under 3-clause BSD license.
<https://github.com/gajus/babel-plugin-react-css-modules>

Copyright © 2016, Gajus Kuizinas (<http://gajus.com/>)

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 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.
- Neither the name of the Gajus Kuizinas (<http://gajus.com/>) nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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 ANUARY 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.

- 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.

The following component (Swagger UI) is distributed as a set of standalone files in the /usr/share/nginx/html/swagger-ui directory:

- Swagger UI (3.25.5), distributed under Apache 2.0 license.

<https://github.com/swagger-api/swagger-ui>

Copyright 2020 SmartBear Software

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, [197](#), [396](#)
add_after_body, [60](#)
add_before_body, [60](#)
add_header, [172](#)
add_trailer, [172](#)
addition_types, [61](#)
aio, [21](#)
aio_write, [22](#)
alias, [22](#)
allow, [58](#), [381](#)
ancient_browser, [120](#)
ancient_browser_value, [120](#)
api, [63](#)
auth_basic, [109](#)
auth_basic_user_file, [109](#)
auth_delay, [23](#)
auth_http, [449](#)
auth_http_header, [449](#)
auth_http_pass_client_cert, [449](#)
auth_http_timeout, [449](#)
auth_jwt, [112](#)
auth_jwt_claim_set, [112](#)
auth_jwt_header_set, [112](#)
auth_jwt_key_file, [113](#)
auth_jwt_key_request, [113](#)
auth_jwt_leeway, [113](#)
auth_request, [115](#)
auth_request_set, [116](#)
autoindex, [117](#)
autoindex_exact_size, [117](#)
autoindex_format, [117](#)
autoindex_localtime, [118](#)

break, [253](#)

charset, [121](#)
charset_map, [122](#)

charset_types, [123](#)
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, [124](#)

daemon, [7](#)
dav_access, [125](#)
dav_methods, [125](#)
debug_connection, [8](#)
debug_points, [8](#)
default_type, [26](#)
deny, [58](#), [381](#)
directio, [26](#)
directio_alignment, [26](#)
disable_symlinks, [27](#)

empty_gif, [127](#)
env, [8](#)
error_log, [9](#)
error_page, [28](#)
etag, [29](#)
events, [10](#)
expires, [173](#)

f4f, [128](#)
f4f_buffer_size, [128](#)
fastcgi_bind, [130](#)
fastcgi_buffer_size, [131](#)
fastcgi_buffering, [131](#)
fastcgi_buffers, [131](#)
fastcgi_busy_buffers_size, [132](#)
fastcgi_cache, [132](#)

fastcgi_cache_background_update, 132
fastcgi_cache_bypass, 132
fastcgi_cache_key, 133
fastcgi_cache_lock, 133
fastcgi_cache_lock_age, 133
fastcgi_cache_lock_timeout, 133
fastcgi_cache_max_range_offset, 134
fastcgi_cache_methods, 134
fastcgi_cache_min_uses, 134
fastcgi_cache_path, 134
fastcgi_cache_purge, 136
fastcgi_cache_revalidate, 137
fastcgi_cache_use_stale, 137
fastcgi_cache_valid, 137
fastcgi_catch_stderr, 138
fastcgi_connect_timeout, 139
fastcgi_force_ranges, 139
fastcgi_hide_header, 139
fastcgi_ignore_client_abort, 139
fastcgi_ignore_headers, 140
fastcgi_index, 140
fastcgi_intercept_errors, 140
fastcgi_keep_conn, 141
fastcgi_limit_rate, 141
fastcgi_max_temp_file_size, 141
fastcgi_next_upstream, 141
fastcgi_next_upstream_timeout, 142
fastcgi_next_upstream_tries, 143
fastcgi_no_cache, 143
fastcgi_param, 143
fastcgi_pass, 144
fastcgi_pass_header, 144
fastcgi_pass_request_body, 144
fastcgi_pass_request_headers, 145
fastcgi_read_timeout, 145
fastcgi_request_buffering, 145
fastcgi_send_lowat, 145
fastcgi_send_timeout, 146
fastcgi_socket_keepalive, 146
fastcgi_split_path_info, 146
fastcgi_store, 146
fastcgi_store_access, 147
fastcgi_temp_file_write_size, 148
fastcgi_temp_path, 148
flv, 150
geo, 151, 382
geoip_city, 155, 385
geoip_country, 154, 384
geoip_org, 156, 386
geoip_proxy, 156
geoip_proxy_recursive, 156
grpc_bind, 158
grpc_buffer_size, 158
grpc_connect_timeout, 158
grpc_hide_header, 158
grpc_ignore_headers, 159
grpc_intercept_errors, 159
grpc_next_upstream, 159
grpc_next_upstream_timeout, 160
grpc_next_upstream_tries, 160
grpc_pass, 161
grpc_pass_header, 161
grpc_read_timeout, 161
grpc_send_timeout, 162
grpc_set_header, 162
grpc_socket_keepalive, 162
grpc_ssl_certificate, 162
grpc_ssl_certificate_key, 163
grpc_ssl_ciphers, 163
grpc_ssl_crl, 163
grpc_ssl_name, 163
grpc_ssl_password_file, 163
grpc_ssl_protocols, 164
grpc_ssl_server_name, 164
grpc_ssl_session_reuse, 164
grpc_ssl_trusted_certificate, 164
grpc_ssl_verify, 164
grpc_ssl_verify_depth, 164
gunzip, 166
gunzip_buffers, 166
gzip, 167
gzip_buffers, 168
gzip_comp_level, 168
gzip_disable, 168
gzip_http_version, 168
gzip_min_length, 168
gzip_proxied, 169
gzip_static, 171
gzip_types, 169
gzip_vary, 170
hash, 323, 428
health_check, 339, 433

health_check_timeout, 434
hls, 176
hls_buffers, 176
hls_forward_args, 176
hls_fragment, 177
hls_mp4_buffer_size, 177
hls_mp4_max_buffer_size, 178
http, 29
http2_body_preread_size, 367
http2_chunk_size, 367
http2_idle_timeout, 367
http2_max_concurrent_pushes, 367
http2_max_concurrent_streams, 367
http2_max_field_size, 368
http2_max_header_size, 368
http2_max_requests, 368
http2_push, 368
http2_push_preload, 369
http2_recv_buffer_size, 369
http2_recv_timeout, 369

if, 254
if_modified_since, 29
ignore_invalid_headers, 29
image_filter, 180
image_filter_buffer, 181
image_filter_interlace, 181
image_filter_jpeg_quality, 181
image_filter_sharpen, 181
image_filter_transparency, 181
image_filter_webp_quality, 182
imap_auth, 463
imap_capabilities, 463
imap_client_buffer, 463
include, 10
index, 183
internal, 30
ip_hash, 323

js_access, 388
js_content, 185
js_filter, 389
js_import, 185, 389
js_include, 186, 389
js_path, 186, 389
js_preread, 390
js_set, 186, 390

keepalive, 324
keepalive_disable, 30
keepalive_requests, 31, 325
keepalive_timeout, 31, 326
keyval, 187, 391
keyval_zone, 188, 392

large_client_header_buffers, 31
least_conn, 327, 429
least_time, 327, 429
limit_conn, 189, 393
limit_conn_dry_run, 190, 394
limit_conn_log_level, 190, 394
limit_conn_status, 191
limit_conn_zone, 191, 394
limit_except, 32
limit_rate, 32
limit_rate_after, 33
limit_req, 193
limit_req_dry_run, 194
limit_req_log_level, 194
limit_req_status, 195
limit_req_zone, 195
limit_zone, 191
lingering_close, 33
lingering_time, 33
lingering_timeout, 34
listen, 34, 374, 444
load_module, 10
location, 37
lock_file, 10
log_format, 199, 397
log_not_found, 39
log_subrequest, 39

mail, 446
map, 201, 399
map_hash_bucket_size, 203, 400
map_hash_max_size, 203, 401
master_process, 11
match, 340, 434
max_ranges, 39
memcached_bind, 204
memcached_buffer_size, 205
memcached_connect_timeout, 205
memcached_force_ranges, 205
memcached_gzip_flag, 206

- memcached_next_upstream, 206
- memcached_next_upstream_timeout, 206
- memcached_next_upstream_tries, 207
- memcached_pass, 207
- memcached_read_timeout, 207
- memcached_send_timeout, 207
- memcached_socket_keepalive, 208
- merge_slashes, 39
- min_delete_depth, 125
- mirror, 209
- mirror_request_body, 209
- modern_browser, 120
- modern_browser_value, 120
- mp4, 212
- mp4_buffer_size, 212
- mp4_limit_rate, 213
- mp4_limit_rate_after, 213
- mp4_max_buffer_size, 212
- msie_padding, 40
- msie_refresh, 40
- multi_accept, 11
- ntlm, 326
- open_file_cache, 40
- open_file_cache_errors, 41
- open_file_cache_min_uses, 41
- open_file_cache_valid, 41
- open_log_file_cache, 200, 398
- output_buffers, 42
- override_charset, 123
- pcre_jit, 11
- perl, 215
- perl_modules, 216
- perl_require, 216
- perl_set, 216
- pid, 11
- pop3_auth, 465
- pop3_capabilities, 465
- port_in_redirect, 42
- postpone_output, 42
- preread_buffer_size, 376
- preread_timeout, 376
- protocol, 446
- proxy_bind, 222, 403
- proxy_buffer, 453
- proxy_buffer_size, 222, 403
- proxy_buffering, 222
- proxy_buffers, 223
- proxy_busy_buffers_size, 223
- proxy_cache, 223
- proxy_cache_background_update, 223
- proxy_cache_bypass, 224
- proxy_cache_convert_head, 224
- proxy_cache_key, 224
- proxy_cache_lock, 224
- proxy_cache_lock_age, 225
- proxy_cache_lock_timeout, 225
- proxy_cache_max_range_offset, 225
- proxy_cache_methods, 225
- proxy_cache_min_uses, 226
- proxy_cache_path, 226
- proxy_cache_purge, 228
- proxy_cache_revalidate, 228
- proxy_cache_use_stale, 228
- proxy_cache_valid, 229
- proxy_connect_timeout, 230, 404
- proxy_cookie_domain, 230
- proxy_cookie_path, 231
- proxy_download_rate, 404
- proxy_force_ranges, 232
- proxy_headers_hash_bucket_size, 232
- proxy_headers_hash_max_size, 232
- proxy_hide_header, 232
- proxy_http_version, 233
- proxy_ignore_client_abort, 233
- proxy_ignore_headers, 233
- proxy_intercept_errors, 233
- proxy_limit_rate, 234
- proxy_max_temp_file_size, 234
- proxy_method, 234
- proxy_next_upstream, 235, 404
- proxy_next_upstream_timeout, 236, 404
- proxy_next_upstream_tries, 236, 405
- proxy_no_cache, 236
- proxy_pass, 236, 405
- proxy_pass_error_message, 453
- proxy_pass_header, 238
- proxy_pass_request_body, 238
- proxy_pass_request_headers, 238

- proxy_protocol, 405
- proxy_protocol_timeout, 377
- proxy_read_timeout, 239
- proxy_redirect, 239
- proxy_request_buffering, 240
- proxy_requests, 405
- proxy_responses, 406
- proxy_send_lowat, 241
- proxy_send_timeout, 241
- proxy_session_drop, 406
- proxy_set_body, 241
- proxy_set_header, 241
- proxy_socket_keepalive, 242, 406
- proxy_ssl, 407
- proxy_ssl_certificate, 242, 407
- proxy_ssl_certificate_key, 243, 407
- proxy_ssl_ciphers, 243, 407
- proxy_ssl_crl, 243, 407
- proxy_ssl_name, 243, 407
- proxy_ssl_password_file, 244, 408
- proxy_ssl_protocols, 244, 408
- proxy_ssl_server_name, 244, 408
- proxy_ssl_session_reuse, 244, 408
- proxy_ssl_trusted_certificate, 244, 408
- proxy_ssl_verify, 245, 409
- proxy_ssl_verify_depth, 245, 409
- proxy_store, 245
- proxy_store_access, 246
- proxy_temp_file_write_size, 246
- proxy_temp_path, 247
- proxy_timeout, 409, 453
- proxy_upload_rate, 409
- queue, 327
- random, 328, 429
- random_index, 248
- read_ahead, 42
- real_ip_header, 249
- real_ip_recursive, 250
- recursive_error_pages, 42
- referer_hash_bucket_size, 251
- referer_hash_max_size, 251
- request_pool_size, 43
- reset_timedout_connection, 43
- resolver, 43, 328, 377, 430, 446
- resolver_timeout, 44, 329, 377, 431, 447
- return, 255, 411
- rewrite, 255
- rewrite_log, 257
- root, 44
- satisfy, 45
- scgi_bind, 260
- scgi_buffer_size, 260
- scgi_buffering, 261
- scgi_buffers, 261
- scgi_busy_buffers_size, 261
- scgi_cache, 262
- scgi_cache_background_update, 262
- scgi_cache_bypass, 262
- scgi_cache_key, 262
- scgi_cache_lock, 262
- scgi_cache_lock_age, 263
- scgi_cache_lock_timeout, 263
- scgi_cache_max_range_offset, 263
- scgi_cache_methods, 263
- scgi_cache_min_uses, 264
- scgi_cache_path, 264
- scgi_cache_purge, 266
- scgi_cache_revalidate, 266
- scgi_cache_use_stale, 266
- scgi_cache_valid, 267
- scgi_connect_timeout, 268
- scgi_force_ranges, 268
- scgi_hide_header, 268
- scgi_ignore_client_abort, 269
- scgi_ignore_headers, 269
- scgi_intercept_errors, 269
- scgi_limit_rate, 269
- scgi_max_temp_file_size, 270
- scgi_next_upstream, 270
- scgi_next_upstream_timeout, 271
- scgi_next_upstream_tries, 271
- scgi_no_cache, 272
- scgi_param, 272
- scgi_pass, 272
- scgi_pass_header, 273
- scgi_pass_request_body, 273
- scgi_pass_request_headers, 273
- scgi_read_timeout, 273
- scgi_request_buffering, 274

scgi_send_timeout, 274
scgi_socket_keepalive, 274
scgi_store, 274
scgi_store_access, 275
scgi_temp_file_write_size, 275
scgi_temp_path, 276
secure_link, 277
secure_link_md5, 278
secure_link_secret, 278
send_lowat, 45
send_timeout, 45
sendfile, 45
sendfile_max_chunk, 46
server, 46, 319, 378, 425, 447
server_name, 46, 447
server_name_in_redirect, 48
server_names_hash_bucket_size, 48
server_names_hash_max_size, 49
server_tokens, 49
session_log, 280
session_log_format, 280
session_log_zone, 281
set, 257
set_real_ip_from, 249, 410
slice, 282
smtp_auth, 466
smtp_capabilities, 466
smtp_client_buffer, 467
smtp_greeting_delay, 467
source_charset, 123
split_clients, 284, 412
ssi, 285
ssi_last_modified, 285
ssi_min_file_chunk, 286
ssi_silent_errors, 286
ssi_types, 286
ssi_value_length, 286
ssl, 292, 456
ssl_buffer_size, 292
ssl_certificate, 293, 414, 456
ssl_certificate_key, 294, 415, 457
ssl_ciphers, 294, 415, 457
ssl_client_certificate, 294, 415, 457
ssl_crl, 294, 416, 458
ssl_dhparam, 295, 416, 458
ssl_early_data, 295
sslecdh_curve, 295, 416, 458
sslEngine, 11
ssl_handshake_timeout, 416
ssl_ocsp, 296
ssl_ocsp_cache, 296
ssl_ocsp_responder, 296
ssl_password_file, 297, 417, 458
ssl_prefer_server_ciphers, 297, 417, 459
ssl_preread, 423
ssl_protocols, 297, 417, 459
ssl_session_cache, 298, 418, 459
ssl_session_ticket_key, 298, 418, 460
ssl_session_tickets, 299, 419, 461
ssl_session_timeout, 299, 419, 461
ssl_stapling, 299
ssl_stapling_file, 299
ssl_stapling_responder, 300
ssl_stapling_verify, 300
ssl_trusted_certificate, 300, 419, 461
ssl_verify_client, 300, 419, 461
ssl_verify_depth, 301, 420, 461
starttls, 462
state, 322, 428
status, 305
status_format, 305
status_zone, 64, 306
sticky, 329
sticky_cookie_insert, 332
stream, 378
stub_status, 314
sub_filter, 316
sub_filter_last_modified, 316
sub_filter_once, 317
sub_filter_types, 317
subrequest_output_buffer_size, 49

tcp_nodelay, 49, 378
tcp_nopush, 50
thread_pool, 12
timeout, 448
timer_resolution, 12
try_files, 50
types, 52
types_hash_bucket_size, 52
types_hash_max_size, 53
underscores_in_headers, 53

- uninitialized_variable_warn, 257
- upstream, 319, 425
- upstream_conf, 334
- use, 12
- user, 13
- userid, 342
- userid_domain, 343
- userid_expires, 343
- userid_mark, 343
- userid_name, 343
- userid_p3p, 344
- userid_path, 344
- userid_service, 344
- uwsgi_bind, 346
- uwsgi_buffer_size, 347
- uwsgi_buffering, 347
- uwsgi_buffers, 347
- uwsgi_busy_buffers_size, 348
- uwsgi_cache, 348
- uwsgi_cache_background_update, 348
- uwsgi_cache_bypass, 348
- uwsgi_cache_key, 349
- uwsgi_cache_lock, 349
- uwsgi_cache_lock_age, 349
- uwsgi_cache_lock_timeout, 349
- uwsgi_cache_max_range_offset, 350
- uwsgi_cache_methods, 350
- uwsgi_cache_min_uses, 350
- uwsgi_cache_path, 350
- uwsgi_cache_purge, 352
- uwsgi_cache_revalidate, 353
- uwsgi_cache_use_stale, 353
- uwsgi_cache_valid, 353
- uwsgi_connect_timeout, 354
- uwsgi_force_ranges, 355
- uwsgi_hide_header, 355
- uwsgi_ignore_client_abort, 355
- uwsgi_ignore_headers, 355
- uwsgi_intercept_errors, 356
- uwsgi_limit_rate, 356
- uwsgi_max_temp_file_size, 356
- uwsgi_modifier1, 356
- uwsgi_modifier2, 357
- uwsgi_next_upstream, 357
- uwsgi_next_upstream_timeout, 358
- uwsgi_next_upstream_tries, 358
- uwsgi_no_cache, 358
- uwsgi_param, 358
- uwsgi_pass, 359
- uwsgi_pass_header, 359
- uwsgi_pass_request_body, 360
- uwsgi_pass_request_headers, 360
- uwsgi_read_timeout, 360
- uwsgi_request_buffering, 360
- uwsgi_send_timeout, 361
- uwsgi_socket_keepalive, 361
- uwsgi_ssl_certificate, 361
- uwsgi_ssl_certificate_key, 361
- uwsgi_ssl_ciphers, 361
- uwsgi_ssl_crl, 362
- uwsgi_ssl_name, 362
- uwsgi_ssl_password_file, 362
- uwsgi_ssl_protocols, 362
- uwsgi_ssl_server_name, 363
- uwsgi_ssl_session_reuse, 363
- uwsgi_ssl_trusted_certificate, 363
- uwsgi_ssl_verify, 363
- uwsgi_ssl_verify_depth, 363
- uwsgi_store, 364
- uwsgi_store_access, 364
- uwsgi_temp_file_write_size, 365
- uwsgi_temp_path, 365
- valid_referers, 252
- variables_hash_bucket_size, 53, 378
- variables_hash_max_size, 53, 378
- 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, 454
- xml_entities, 370
- xslt_last_modified, 371
- xslt_param, 371
- xslt_string_param, 371
- xslt_stylesheet, 371

xslt_types, [372](#)

zone, [322](#), [427](#)

zone_sync, [437](#)

zone_sync_buffers, [438](#)

zone_sync_connect_retry_interval, [438](#)

zone_sync_connect_timeout, [438](#)

zone_sync_interval, [438](#)

zone_sync_recv_buffer_size, [438](#)

zone_sync_server, [439](#)

zone_sync_ssl, [439](#)

zone_sync_ssl_certificate, [439](#)

zone_sync_ssl_certificate_key, [440](#)

zone_sync_ssl_ciphers, [440](#)

zone_sync_ssl_crl, [440](#)

zone_sync_ssl_name, [440](#)

zone_sync_ssl_password_file, [440](#)

zone_sync_ssl_protocols, [441](#)

zone_sync_ssl_server_name, [441](#)

zone_sync_ssl_trusted_certificate, [441](#)

zone_sync_ssl_verify, [441](#)

zone_sync_ssl_verify_depth, [441](#)

zone_sync_timeout, [441](#)