Nginx, Inc.

NGINX Plus Reference Guide

NGINX Plus - release 8, based on 1.9.9 core

December 24, 2015

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 http://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

Chapter 1

Core modules

1.1 Core functionality

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	on
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.

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.

error_log

SYNTAX: **`error_log`** *file* | `stderr` | `syslog:server=`*address*[,*parameter*=*value*] | `memory:`*size* [`debug` | `info` | `notice` | `warn` | `error` | `crit` | `alert` | `emerg`];
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).

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. 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.

The directive can be specified on the mail level starting from version 1.9.0.

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.

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;
```

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 `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* . . . ;
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 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.

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

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

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 Mac OS X.

- `epoll` — efficient method used on Linux 2.6+.

> 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, efficient method used on Solaris 10.

1.4 Logging to syslog

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

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. 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.

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;
}
```

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 `408 Request Time-out` error is returned to the client.

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 `408 Request Time-out` error is returned to the client.

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 (Mac OS X), 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. `error_page` directives are inherited from the previous level only if there are no `error_page` directives defined on the current level. A `uri` value can contain variables.

Example:

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

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 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;
```

It is also possible to use 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, and 307).

If there is no need to change URI 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.

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 and by the ngx_http_addition_module module 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 Mac OS X and Mac OS X-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.

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 and ngx_http_auth_basic_module 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.

Rate limit can also be set in the *$limit_rate* variable. It may be useful in cases where rate should be limited depending on a certain condition:

```
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.

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 `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 Mac OS X, 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), allowing a kernel to distribute incoming connections between worker processes. This currently works only on Linux 3.9+ and DragonFly BSD.

> 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 Mac OS X 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, or memcached_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 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. 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];
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;
```

An address can be specified as a domain name or IP address, and 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.

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 or ngx_http_auth_request_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 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;
DEFAULT on
CONTEXT: http, server, location

Enables or disables emitting nginx version in error messages and in the `Server` response header field.

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 only when a connection is transitioned into the keep-alive state.

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

$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, or an empty string otherwise (1.5.12)
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.

$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_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

$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

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

Access can also be limited by password or by the result of subrequest. 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

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_auth_basic_module

2.4.1 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 or by the result of subrequest. Simultaneous limitation of access by address and by password is controlled by the satisfy directive.

2.4.2 Example Configuration

```
location / {
    auth_basic           "closed site";
    auth_basic_user_file conf/htpasswd;
}
```

2.4.3 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.

2.5 Module ngx_http_auth_request_module

2.5.1 Summary

The `ngx_http_auth_request_module` 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 and ngx_http_auth_basic_module, via the satisfy directive.

> Before version 1.7.3, responses to authorization subrequests could not be cached (using proxy_cache, proxy_store, etc.).

2.5.2 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;
}
```

2.5.3 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_**.

2.6 Module ngx_http_autoindex_module

2.6.1 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.

2.6.2 Example Configuration

```
location / {
    autoindex on;
}
```

2.6.3 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.

2.7 Module ngx_http_browser_module

2.7.1 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.

2.7.2 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;
}
```

2.7.3 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.

2.8 Module ngx_http_charset_module

2.8.1 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.

2.8.2 Example Configuration

```
include        conf/koi-win;

charset        windows-1251;
source_charset koi8-r;
```

2.8.3 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, and scgi_ignore_headers directives.

charset_map

SYNTAX: **`charset_map`** *charset1 charset2* { ... }
DEFAULT —
CONTEXT: http

Describes the conversion table from one charset to another. A reverse conversion table is built using the same data. Character codes are given in hexadecimal. Missing characters in the range 80-FF are replaced with "?". When converting from UTF-8, characters missing in a one-byte charset are replaced with "`&#XXXX;`".

Example:

```
charset_map koi8-r windows-1251 {
    C0 FE ; # small yu
    C1 E0 ; # small a
    C2 E1 ; # small b
    C3 F6 ; # small ts
    ...
}
```

When describing a conversion table to UTF-8, codes for the UTF-8 charset should be given in the second column, for example:

```
charset_map koi8-r utf-8 {
    C0 D18E ; # small yu
    C1 D0B0 ; # small a
    C2 D0B1 ; # small b
    C3 D186 ; # small ts
    ...
}
```

Full conversion tables from `koi8-r` to `windows-1251`, and from `koi8-r` and `windows-1251` to `utf-8` are provided in the distribution files `conf/koi-win`, `conf/koi-utf`, and `conf/win-utf`.

charset_types

SYNTAX: **charset_types** *mime-type* ...;
DEFAULT `text/html text/xml text/plain text/vnd.wap.wml application/javascript application/rss+xml`
CONTEXT: http, server, location
THIS DIRECTIVE APPEARED IN VERSION 0.7.9.

Enables module processing in responses with the specified MIME types in addition to "`text/html`". The special value "`*`" matches any MIME type (0.8.29).

> Until version 1.5.4, "`application/x-javascript`" was used as the default MIME type instead of "`application/javascript`".

override_charset

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

Determines whether a conversion should be performed for answers received from a proxied or a FastCGI/uwsgi/SCGI 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.

2.9 Module ngx_http_dav_module

2.9.1 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.

2.9.2 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;
    }
}
```

2.9.3 Directives

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.

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.

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
```

2.10 Module ngx_http_empty_gif_module

2.10.1 Summary

The `ngx_http_empty_gif_module` module emits single-pixel transparent GIF.

2.10.2 Example Configuration

```
location = /_.gif {
    empty_gif;
}
```

2.10.3 Directives

empty_gif

SYNTAX: **empty_gif**;
DEFAULT —
CONTEXT: location

Turns on module processing in a surrounding location.

2.11 Module ngx_http_f4f_module

2.11.1 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.

2.11.2 Example Configuration

```
location /video/ {
    f4f;
    ...
}
```

2.11.3 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.

2.12 Module ngx_http_fastcgi_module

2.12.1 Summary

The `ngx_http_fastcgi_module` module allows passing requests to a FastCGI server.

2.12.2 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;
}
```

2.12.3 Directives

fastcgi_bind

SYNTAX: **`fastcgi_bind`** *address* | `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*. 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.

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_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_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*] [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. 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.

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.

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 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.

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 | 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.

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 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` | `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;

`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` and `http_503` 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 allowed to pass a request 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 a directive is specified with `if_not_empty` (1.1.11) then such a parameter will not be passed to the server until 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.

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_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.

2.12.4 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.

2.12.5 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.

2.13 Module ngx_http_flv_module

2.13.1 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.

2.13.2 Example Configuration

```
location ~ \.flv$ {
    flv;
}
```

2.13.3 Directives

flv

Syntax: **flv**;
Default —
Context: location

Turns on module processing in a surrounding location.

2.14 Module ngx_http_geo_module

2.14.1 Summary

The `ngx_http_geo_module` module creates variables with values depending on the client IP address.

2.14.2 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;
}
```

2.14.3 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;
}
```

2.15 Module ngx_http_geoip_module

2.15.1 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.

2.15.2 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;
    ...
```

2.15.3 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.

2.16 Module ngx_http_gunzip_module

2.16.1 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.

2.16.2 Example Configuration

```
location /storage/ {
    gunzip on;
    ...
}
```

2.16.3 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.

2.17 Module ngx_http_gzip_module

2.17.1 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.

2.17.2 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.

2.17.3 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_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_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_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.

2.17.4 Embedded Variables

$gzip_ratio
achieved compression ratio, computed as the ratio between the original and compressed response sizes.

2.18 Module ngx_http_gzip_static_module

2.18.1 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.

2.18.2 Example Configuration

```
gzip_static  on;
gzip_proxied expired no-cache no-store private auth;
```

2.18.3 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.

2.19 Module ngx_http_headers_module

2.19.1 Summary

The `ngx_http_headers_module` module allows adding the `Expires` and `Cache-Control` header fields, and arbitrary fields, to a response header.

2.19.2 Example Configuration

```
expires    24h;
expires    modified +24h;
expires    @24h;
expires    0;
expires    -1;
expires    epoch;
expires    $expires;
add_header Cache-Control private;
```

2.19.3 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, 204, 206, 301, 302, 303, 304, or 307. A 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.

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, 204, 206, 301, 302, 303, 304, or 307. A parameter can be a positive or negative time.

A 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 time is computed as a sum of the file's modification time and *time* specified in the directive.

In addition, it is possible to specify a time of the day using the "`@`" prefix (0.7.9, 0.6.34):

```
expires @15h30m;
```

The `epoch` parameter corresponds to the absolute time "`Thu, 01 Jan 1970 00:00:01 GMT`". 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 `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;
```

2.20 Module ngx_http_hls_module

2.20.1 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.

2.20.2 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
```

2.20.3 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
```

2.21 Module ngx_http_image_filter_module

2.21.1 Summary

The `ngx_http_image_filter_module` module (0.7.54+) is a filter that transforms images in JPEG, GIF, and PNG 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.

2.21.2 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;
}
```

2.21.3 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, or PNG 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.

2.22 Module ngx_http_index_module

2.22.1 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.

2.22.2 Example Configuration

```
location / {
    index index.$geo.html index.html;
}
```

2.22.3 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`".

2.23 Module ngx_http_limit_conn_module

2.23.1 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 processed by the server and the whole request header has already been read.

2.23.2 Example Configuration

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

        location /download/ {
            limit_conn addr 1;
        }
```

2.23.3 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 `503 Service Temporarily Unavailable` 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_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. The stored state always occupies 32 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 `503 Service Temporarily Unavailable` error to all further requests.

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*;

2.24 Module ngx_http_limit_req_module

2.24.1 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.

2.24.2 Example Configuration

```
http {
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

    ...

    server {

        ...

        location /search/ {
            limit_req zone=one burst=5;
        }
```

2.24.3 Directives

limit_req

SYNTAX: **`limit_req`** `zone=`*name* [`burst=`*number*] [`nodelay`];
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 `503 Service Temporarily Unavailable`. 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;
```

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_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*;
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, that allows decreasing the state size down to 64 bytes. One megabyte zone can keep about 16 thousand 64-byte states. If the zone storage is exhausted, the server will return the `503 Service Temporarily Unavailable` error to all further requests.

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.

2.25 Module ngx_http_log_module

2.25.1 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.

2.25.2 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;
```

2.25.3 Directives

access_log

SYNTAX: **access_log** *path* [*format* [buffer=*size* [flush=*time*]] [if=*condition*]];

SYNTAX: **access_log** *path* *format* gzip[=*level*] [buffer=*size*] [flush=*time*] [if=*condition*];

SYNTAX: **access_log** syslog:server=*address*[,*parameter*=*value*] [*format* [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 string* ...;
DEFAULT `combined "..."`
CONTEXT: http

Specifies log format.

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;
```

2.26 Module ngx_http_map_module

2.26.1 Summary

The `ngx_http_map_module` module creates variables whose values depend on values of other variables.

2.26.2 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;
}
```

2.26.3 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).

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 be a string or another variable (0.9.0).

The directive also supports three special parameters:

`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.

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.

2.27 Module ngx_http_memcached_module

2.27.1 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.

2.27.2 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;
    }
}
```

2.27.3 Directives

memcached_bind

SYNTAX: **`memcached_bind`** *address* | `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*. 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.

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_header` 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 allowed to pass a request 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.

2.27.4 Embedded Variables

$memcached_key
Defines a key for obtaining response from a memcached server.

2.28 Module ngx_http_mp4_module

2.28.1 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.

2.28.2 Example Configuration

```
location /video/ {
    mp4;
    mp4_buffer_size       1m;
    mp4_max_buffer_size   5m;
    mp4_limit_rate        on;
    mp4_limit_rate_after  30s;
}
```

2.28.3 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.

2.29 Module ngx_http_perl_module

2.29.1 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.

2.29.2 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 `-Dusemymalloc=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:usemymalloc
usemultiplicity='define';
usemymalloc='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.

2.29.3 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__
```

2.29.4 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.

2.29.5 Calling Perl from SSI

An SSI command calling Perl has the following format:

```
<!--# perl sub="module::function" arg="parameter1" arg="parameter2" ...
-->
```

2.29.6 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.

> Redirections to named locations are currently not supported.

`$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.

2.30 Module ngx_http_proxy_module

2.30.1 Summary

The `ngx_http_proxy_module` module allows passing requests to another server.

2.30.2 Example Configuration

```
location / {
    proxy_pass       http://localhost:8000;
    proxy_set_header Host      $host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

2.30.3 Directives

proxy_bind

Syntax: **`proxy_bind`** *address* | `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*. 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.

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_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_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*] [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. 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.

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.

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 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.

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` | `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 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.

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 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.

proxy_next_upstream

SYNTAX: **proxy_next_upstream** `error` | `timeout` | `invalid_header` | `http_500` | `http_502` | `http_503` | `http_504` | `http_403` | `http_404` | `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;

`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` and `http_504` 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 allowed to pass a request 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.

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.

  In this case, the directive 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.

A server name, its port and the passed URI can also be specified using variables:

```
proxy_pass http://$host$uri;
```

or even like this:

```
proxy_pass $request;
```

In this case, the server name is searched among the described server groups, and, if not found, is determined using a resolver.

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_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 to override 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_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_protocols

SYNTAX: **`proxy_ssl_protocols`** [`SSLv2`] [`SSLv3`] [`TLSv1`] [`TLSv1.1`] [`TLSv1.2`];
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_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.

2.30.4 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.

2.31 Module ngx_http_random_index_module

2.31.1 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.

2.31.2 Example Configuration

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

2.31.3 Directives

random_index

SYNTAX: **`random_index`** `on` | `off`;
DEFAULT `off`
CONTEXT: location

Enables or disables module processing in a surrounding location.

2.32 Module ngx_http_realip_module

2.32.1 Summary

The `ngx_http_realip_module` module is used to change the client address to the one 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.

2.32.2 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;
```

2.32.3 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.

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 `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.

real_ip_recursive

Syntax: **`real_ip_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 real_ip_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.

2.32.4 Embedded Variables

$realip_remote_addr
keeps the original client address (1.9.7)

2.33 Module ngx_http_referer_module

2.33.1 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.

2.33.2 Example Configuration

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

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

2.33.3 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\.;
```

2.33.4 Embedded Variables

$invalid_referer
: Empty string, if the Referer request header field value is considered valid, otherwise "1".

2.34 Module ngx_http_rewrite_module

2.34.1 Summary

The `ngx_http_rewrite_module` module is used to change request URI using regular expressions, return redirects, and conditionally select configurations.

The `ngx_http_rewrite_module` module 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.

2.34.2 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, and 307), 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.

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://`" or "`https://`", 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://`" or "`https://`";

`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.

2.34.3 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
```

2.35 Module ngx_http_scgi_module

2.35.1 Summary

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

2.35.2 Example Configuration

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

2.35.3 Directives

scgi_bind

SYNTAX: **scgi_bind** *address* | `off`;
DEFAULT —
CONTEXT: http, server, location

Makes outgoing connections to an SCGI server originate from the specified local IP *address*. 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.

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_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_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*] [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. 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.

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.

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 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.

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 | 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.

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 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 | 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;

`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` and `http_503` 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 allowed to pass a request 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 a directive is specified with `if_not_empty` (1.1.11) then such a parameter will not be passed to the server until 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.

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_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.

2.36 Module ngx_http_secure_link_module

2.36.1 Summary

The `ngx_http_secure_link_module` 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.

2.36.2 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
```

2.36.3 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.

2.37 Module ngx_http_session_log_module

2.37.1 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.

2.37.2 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;
}
```

2.37.3 Directives

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.

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.

2.37.4 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).

2.38 Module ngx_http_slice_module

2.38.1 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.

2.38.2 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.

2.38.3 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 a large number of file descriptors.

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.

2.38.4 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`.

2.39 Module ngx_http_split_clients_module

2.39.1 Summary

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

2.39.2 Example Configuration

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

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

2.39.3 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).

2.40 Module ngx_http_ssi_module

2.40.1 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.

2.40.2 Example Configuration

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

2.40.3 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.

2.40.4 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 -->
```

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 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" --> <!--# 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"
    -->
```

It should be noted that 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 can be written into variables.

`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.

2.40.5 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.

2.41 Module ngx_http_ssl_module

2.41.1 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.

2.41.2 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;

        ...
    }
```

2.41.3 Directives

ssl

SYNTAX: **ssl** on | off;
DEFAULT off
CONTEXT: http, server

Enables the HTTPS protocol for the given virtual server.

It is recommended to use the `ssl` parameter of the listen directive instead of this directive.

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.

It should be kept in mind that due to the HTTPS protocol limitations virtual servers should listen on different IP addresses:

```
server {
    listen          192.168.1.1:443;
    server_name     one.example.com;
    ssl_certificate /usr/local/nginx/conf/one.example.com.cert;
    ...
}

server {
    listen          192.168.1.2:443;
    server_name     two.example.com;
    ssl_certificate /usr/local/nginx/conf/two.example.com.cert;
    ...
}
```

otherwise the first server's certificate will be issued for the second site.

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*.

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 EDH ciphers.

ssl_ecdh_curve

Syntax: **ssl_ecdh_curve** *curve*;
Default `prime256v1`
Context: http, server
This directive appeared in versions 1.1.0 and 1.0.6.

Specifies a *curve* for ECDHE ciphers.

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];
DEFAULT TLSv1 TLSv1.1 TLSv1.2
CONTEXT: http, server

Enables the specified protocols. The TLSv1.1 and TLSv1.2 parameters work only when the OpenSSL library of version 1.0.1 or higher is used.

> The TLSv1.1 and TLSv1.2 parameters are supported starting from versions 1.1.13 and 1.0.12, so when the OpenSSL version 1.0.1 or higher is used on older nginx versions, these protocols work, but cannot be disabled.

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 48 bytes of random data and can be created using the following command:

```
openssl rand 48 > ticket.key
```

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 stored in a cache.

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.

2.41.4 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.

2.41.5 Embedded Variables

The `ngx_http_ssl_module` module supports several embedded variables:

$ssl_cipher
: returns the string of ciphers used for an established SSL connection;

$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;

$ssl_client_fingerprint
: returns the SHA1 fingerprint of the client certificate for an established SSL connection (1.7.1);

$ssl_client_raw_cert
: returns the client certificate in the PEM format for an established SSL connection;

$ssl_client_serial
: returns the serial number of the client certificate for an established SSL connection;

$ssl_client_s_dn
: returns the "subject DN" string of the client certificate for an established SSL connection;

$ssl_client_i_dn
: returns the "issuer DN" string of the client certificate for an established SSL connection;

$ssl_client_verify
: returns the result of client certificate verification: "`SUCCESS`", "`FAILED`", and "`NONE`" if a certificate was not present;

$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).

2.42 Module ngx_http_status_module

2.42.1 Summary

The `ngx_http_status_module` module provides access to various status information.

This module is available as part of our commercial subscription.

2.42.2 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.

2.42.3 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. The 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.

2.42.4 Data

The following status information is provided:

`version`
: Version of the provided data set. The current version is 6.

`nginx_version`
: Version of nginx.

`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.

`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.

`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.

`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`", 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`" and "`unhealthy`" states.

`downstart`
: The time (in milliseconds since Epoch) when the server became "`unavail`" 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). The field is available when using the least_time load balancing method.

`response_time`
The average time to get the full response from the server (1.7.10). The field is available when using the least_time load balancing method.

`keepalive`
The current number of idle keepalive connections.

`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.

`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.

`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`", or "`unhealthy`".

`active`
The current number of connections.

`connections`
The total number of client connections forwarded to this server.

`connect_time`
The average time to connect to the upstream server. The field is available when using the least_time load balancing method.

`first_byte_time`
The average time to receive the first byte of data. The field is available when using the least_time load balancing method.

`response_time`
: The average time to receive the last byte of data. The field is available 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`" and "`unhealthy`" states.

`downstart`
: The time (in milliseconds since Epoch) when the server became "`unavail`" or "`unhealthy`".

`selected`
: The time (in milliseconds since Epoch) when the server was last selected to process a connection.

2.42.5 Compatibility

- 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.

2.43 Module ngx_http_stub_status_module

2.43.1 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.

2.43.2 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
```

2.43.3 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`”.

2.43.4 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.

2.43.5 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.

2.44 Module ngx_http_sub_module

2.44.1 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.

2.44.2 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;
}
```

2.44.3 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).

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).

2.45 Module ngx_http_upstream_module

2.45.1 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, and memcached_pass directives.

2.45.2 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, 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 backup1.example.com:8080  backup;
    server backup2.example.com:8080  backup;
}

server {
    location / {
        proxy_pass http://dynamic;
        health_check;
    }
}
```

2.45.3 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_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, and memcached_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.

`down`
: marks the server as permanently unavailable.

Additionally, the following parameters are available as part of our commercial subscription:

`max_conns`=*number*
: limits the maximum *number* of simultaneous active connections to the proxied server (1.5.9). Default value is zero, meaning there is no limit.

> When keepalive connections and multiple workers are enabled, the total number of connections to the proxied server may exceed the `max_conns` value.

`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. Example:

```
http {
    resolver 10.0.0.1;

    upstream u {
        zone ...;
        ...
        server example.com resolve;
    }
}
```

`route`=*string*
: sets the server route name.

`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.

> 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 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. 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.

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;
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.

This directive is available as part of our commercial subscription.

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;

`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, "/";

`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.

For example,

```
location / {
    proxy_pass http://backend;
    health_check;
}
```

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. Client requests are not passed to unhealthy servers.

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. For example:

```
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 tells that for a health check to pass, the response to a health check request should succeed, have status 200, content type "`text/html`", and contain "`Welcome to nginx!`" in the body.

The server group must reside in the shared memory.

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.

> Please note that most of the variables will have empty values when used with health checks.

> This directive is available as part of our commercial subscription.

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!`"

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";
}
```

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, and there are the servers in the group that have reached the max_conns limit, 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.

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*];
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. 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.

`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 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 8000 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.

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*];

2.45.4 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:

$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_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.

2.46 Module ngx_http_upstream_conf_module

2.46.1 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 is available as part of our commercial subscription.

2.46.2 Example Configuration

```
upstream backend {
    zone upstream_backend 64k;

    ...
}

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

2.46.3 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.

`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 in the "draining" mode (1.7.5). In this mode, only requests of the 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.

2.47 Module ngx_http_userid_module

2.47.1 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.

2.47.2 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"';
```

2.47.3 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.

2.47.4 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.

2.48 Module ngx_http_uwsgi_module

2.48.1 Summary

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

2.48.2 Example Configuration

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

2.48.3 Directives

uwsgi_bind

SYNTAX: **`uwsgi_bind`** *address* | `off`;
DEFAULT —
CONTEXT: http, server, location

Makes outgoing connections to a uwsgi server originate from the specified local IP *address*. 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.

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_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_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*] [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. 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.

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.

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 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.

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` | `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.

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 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 | 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;

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 and http_503 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 allowed to pass a request 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 a directive is specified with `if_not_empty` (1.1.11) then such a parameter will not be passed to the server until 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.

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_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];
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.

2.49 Module ngx_http_v2_module

2.49.1 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.

2.49.2 Known Issues

The module is experimental, caveat emptor applies.

Buffering of a client request body cannot be disabled regardless of proxy_request_buffering, fastcgi_request_buffering, uwsgi_request_buffering, and scgi_request_buffering directive values.

2.49.3 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.

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.

2.49.4 Directives

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_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_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.

2.49.5 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.

2.50 Module ngx_http_xslt_module

2.50.1 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.

2.50.2 Example Configuration

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

2.50.3 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 " ">
```

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 proxy modules

3.1 Module ngx_stream_core_module

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;
    }

    server {
        listen 12345;
```

```
        proxy_connect_timeout 1s;
        proxy_timeout 3s;
        proxy_pass backend;
    }

    server {
        listen [::1]:12345;
        proxy_pass unix:/tmp/stream.socket;
    }
}
```

3.1.3 Directives

listen

SYNTAX: **listen** *address*:*port* [ssl] [backlog=*number*] [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;
```

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 Mac OS X, and to 511 on other platforms.

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), allowing a kernel to distribute incoming connections between worker processes. This currently works only on Linux 3.9+ and DragonFly BSD.

> Inappropriate use of this option may have its security implications.

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

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

will set the idle timeout (`TCP_KEEPIDLE`) to 30 minutes, leave the probe interval (`TCP_KEEPINTVL`) at its system default, and set the probes count (`TCP_KEEPCNT`) to 10 probes.

Different servers must listen on different *address*:*port* pairs.

resolver

SYNTAX: **`resolver`** *address* ... [`valid=`*time*] [`ipv6=on|off`];
DEFAULT —
CONTEXT: stream, server

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

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

An address can be specified as a domain name or IP address, and 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;
```

This directive is available as part of our commercial subscription.

resolver_timeout

SYNTAX: **`resolver_timeout`** *time*;
DEFAULT `30s`
CONTEXT: stream, server

Sets a timeout for name resolution, for example:

```
resolver_timeout 5s;
```

This directive is 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.

3.2 Module ngx_stream_access_module

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_limit_conn_module

3.3.1 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.

3.3.2 Example Configuration

```
stream {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

        limit_conn           addr 1;
        limit_conn_log_level error;
    }
}
```

3.3.3 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_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. Currently, the supported value for the *key* is the client address in the binary form specified as `$binary_remote_addr`. 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 `$binary_remote_addr` key. The size of `$binary_remote_addr` is 4 bytes. The stored state always occupies 32 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.

3.4 Module ngx_stream_proxy_module

3.4.1 Summary

The `ngx_stream_proxy_module` module (1.9.0) allows passing connections to another server over TCP and UNIX-domain sockets.

3.4.2 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 [::1]:12345;
    proxy_pass unix:/tmp/stream.socket;
}
```

3.4.3 Directives

proxy_bind

Syntax: **`proxy_bind`** *address* | `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*. 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.

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.

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.

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_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 to override 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.

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_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_protocols

SYNTAX: **proxy_ssl_protocols** [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];
DEFAULT TLSv1 TLSv1.1 TLSv1.2
CONTEXT: stream, server

Enables the specified protocols for connections to a proxied server.

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.

3.5 Module ngx_stream_ssl_module

3.5.1 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.

3.5.2 Example Configuration

To reduce the processor load, it is recommended to

- set the number of worker processes equal to the number of processors,
- enable the shared session cache,
- disable the built-in session cache,
- and possibly increase the session lifetime (by default, 5 minutes):

```
worker_processes auto;

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;

        ...
    }
```

3.5.3 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.

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*.

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_dhparam

Syntax: **ssl_dhparam** *file*;
Default —
Context: stream, server

Specifies a *file* with DH parameters for EDH ciphers.

ssl_ecdh_curve

SYNTAX: **ssl_ecdh_curve** *curve*;
DEFAULT prime256v1
CONTEXT: stream, server

Specifies a *curve* for ECDHE ciphers.

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];
Default TLSv1 TLSv1.1 TLSv1.2
Context: stream, server

Enables the specified protocols. The TLSv1.1 and TLSv1.2 parameters work only when the OpenSSL library of version 1.0.1 or higher is used.

ssl_session_cache

Syntax: **ssl_session_cache** off | none | [builtin[:*size*]] [shared:*name*:*size*];
Default none
Context: stream, server

Sets the types and sizes of caches that store session parameters. A cache can be of any of the following types:

off
: the use of a session cache is strictly prohibited: nginx explicitly tells a client that sessions may not be reused.

none
: the use of a session cache is gently disallowed: nginx tells a client that sessions may be reused, but does not actually store session parameters in the cache.

builtin
: a cache built in OpenSSL; used by one worker process only. The cache size is specified in sessions. If size is not given, it is equal to 20480 sessions. Use of the built-in cache can cause memory fragmentation.

shared
: a cache shared between all worker processes. The cache size is specified in bytes; one megabyte can store about 4000 sessions. Each shared cache should have an arbitrary name. A cache with the same name can be used in several servers.

Both cache types can be used simultaneously, for example:

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

but using only shared cache without the built-in cache should be more efficient.

ssl_session_ticket_key

Syntax: **ssl_session_ticket_key** *file*;
Default —
Context: stream, server

Sets a *file* with the secret key used to encrypt and decrypt TLS session tickets. The directive is necessary if the same key has to be shared between multiple servers. By default, a randomly generated key is used.

If several keys are specified, only the first key is used to encrypt TLS session tickets. This allows configuring key rotation, for example:

```
ssl_session_ticket_key current.key;
ssl_session_ticket_key previous.key;
```

The *file* must contain 48 bytes of random data and can be created using the following command:

```
openssl rand 48 > ticket.key
```

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 stored in a cache.

3.6 Module ngx_stream_upstream_module

3.6.1 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.

3.6.2 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, 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 backup1.example.com:12345  backup;
    server backup2.example.com:12345  backup;
}
```

```
server {
    listen 12346;
    proxy_pass dynamic;
    health_check;
}
```

3.6.3 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_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.

`down`
: marks the server as permanently unavailable.

Additionally, the following parameters are available as part of our commercial subscription:

`max_conns=`*number*
: limits the maximum *number* of simultaneous connections to the proxied server. Default value is zero, meaning there is no limit.

`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. Example:

```
stream {
    resolver 10.0.0.1;

    upstream u {
        zone ...;
        ...
        server example.com:12345 resolve;
    }
}
```

`slow_start=`*time*
: sets the *time* during which the server will recover its weight from zero to a nominal value, 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.

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 zone 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 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. 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 client-server mapping is based on the hashed *key* value. Currently, the only supported value for the `key` is the client remote address specified as `$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 server 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`;
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.

This directive is available as part of our commercial subscription.

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;

`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;

`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, only the ability to connect to the server is checked;

`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.

For example,

```
server {
    proxy_pass backend;
    health_check;
}
```

will check the ability to connect to each server in the `backend` 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. Client connections are not passed to unhealthy servers.

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.

The server group must reside in the shared memory.

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.

This directive is available as part of our commercial subscription.

health_check_timeout

SYNTAX: **`health_check_timeout`** *timeout*;
DEFAULT `5s`
CONTEXT: stream, server

Overrides the proxy_timeout value for health checks.

This directive is available as part of our commercial subscription.

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 ~` *regexp*;
: 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).

Health check is passed if:

- the connection was successfully established;
- the *string* from the `send` parameter, if specified, was sent;
- the data obtained from the server matched the 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_upstream_buffer bytes of data obtained from the server are examined.

This directive is available as part of our commercial subscription.

Chapter 4

Mail server modules

4.1 Module ngx_mail_core_module

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*] [`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 Mac OS X, and to 511 on other platforms.

`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*];
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;
```

An address can be specified as a domain name or IP address, and 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 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.

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

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
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`", and "`NONE`" if a certificate was not present. 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

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

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

Enables the SSL/TLS protocol for the given server.

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.

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*.

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 EDH ciphers.

ssl_ecdh_curve

SYNTAX: **`ssl_ecdh_curve`** *curve*;
DEFAULT `prime256v1`
CONTEXT: mail, server

This directive appeared in versions 1.1.0 and 1.0.6.

Specifies a *curve* for ECDHE ciphers.

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];
Default TLSv1 TLSv1.1 TLSv1.2
Context: mail, server

Enables the specified protocols. The TLSv1.1 and TLSv1.2 parameters work only when the OpenSSL library of version 1.0.1 or higher is used.

The `TLSv1.1` and `TLSv1.2` parameters are supported starting from versions 1.1.13 and 1.0.12 so when the OpenSSL version 1.0.1 or higher is used on older nginx versions, these protocols work, but cannot be disabled.

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 48 bytes of random data and can be created using the following command:

```
openssl rand 48 > ticket.key
```

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 stored in a cache.

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;

`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

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.

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 and STARTTLS directives are automatically added to this list if the starttls directive is enabled.

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 IMAP commands read buffer size. 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

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.

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 and (SASL extension) and STLS directives, are automatically added to this list if the starttls directive is enabled.

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

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.

`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. Authentication methods specified in the smtp_auth directive are automatically added to this list.

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.

Chapter 5

Miscellaneous

5.1 High Availability support for NGINX Plus

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 it's 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 vrrp_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 virtual_ipaddress 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

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: http://nginx.org/en/CHANGES

- 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

At the release moment of this document, there are two versions of NGINX Plus package in distribution:

- NGINX Plus (package name is `nginx-plus`)
- NGINX Plus Extras (package name is `nginx-plus-extras`)

These distributions contain a different set of various open source software components described below.

Open source components included in NGINX Plus and NGINX Plus Extras are:

- nginx/OSS (1.9.9), distributed under 2-clause BSD license.

  http://nginx.org/

- 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

- MurmurHash algorithm (version 2), distributed under MIT license.

  https://sites.google.com/site/murmurhash/

The following components (Ractive.js, opentip, Reset CSS, Web Font Loader) are used in status monitoring dashboard only:

- Ractive.js JavaScript library (0.7.3), distributed under MIT license.

  http://www.ractivejs.org/

- opentip JavaScript tooltip framework (2.4.6), distributed under MIT license.

  http://www.opentip.org/

- Reset CSS (v2), distributed under public domain.

  http://meyerweb.com/eric/tools/css/reset/

- Web Font Loader (1.6.6), distributed under Apache 2.0 license.

  https://github.com/typekit/webfontloader/

Open source components included in NGINX Plus Extras are:

- LuaJIT (2.0.4), a Just-In-Time Compiler for Lua

  http://luajit.org/

MIT license: http://www.opensource.org/licenses/mit-license.php

LuaJIT includes code from Lua 5.1/5.2, which has this license statement:

LuaJIT includes code from dlmalloc, which has this license statement:

This is a version (aka dlmalloc) of malloc/free/realloc written by Doug Lea and released to the public domain, as explained at http://creativecommons.org/licenses/publicdomain

- Nginx Development Kit module (0.2.19), distributed under BSD license.

  https://github.com/simpl/ngx_devel_kit

- lua-nginx-module (0.9.20), distributed under 2-clause BSD license.

  https://github.com/openresty/lua-nginx-module

- headers-more-nginx-module (0.28), distributed under 2-clause BSD license.
  https://github.com/openresty/headers-more-nginx-module

- set-misc-nginx-module (0.29), distributed under 2-clause BSD license.
  https://github.com/openresty/set-misc-nginx-module

- lua-resty-redis Lua module (0.21), distributed under 2-clause BSD license.

  https://github.com/openresty/lua-resty-redis

- Phusion Passenger open source module for nginx (5.0.22)

  https://www.phusionpassenger.com/

- nginx-rtmp-module (1.1.7), distributed under 2-clause BSD license.

  https://github.com/arut/nginx-rtmp-module

- 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 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 THE COPYRIGHT OWNER 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.

Index