The Nginx proxy module

Similarly to the previous chapter, the first step towards establishing the new architecture will be to discover the appropriate module. The default Nginx build comes with the proxy module, which allows forwarding of HTTP requests from the client to a backend server. We will be configuring multiple aspects of the module:

- Basic address and port information of the backend server
- Caching, buffering, and temporary file options
- Limits, timeout, and error behavior
- Other miscellaneous options

All these options are available via directives, which we will learn to configure throughout this section.

Main directives

The first set of directives will allow you to establish the basic configuration, such as the location of the backend server, information to be passed, and how it should be passed.

Directive	Description

`proxy_pass`
Context: `location`, `if`
	Specifies that the request should be forwarded to the backend server by indicating its location:
	For regular HTTP forwarding, the syntax is `proxy_pass http://hostname:port;`
	For Unix domain sockets, the syntax is: `proxy_pass http://unix:/path/to/file.socket;`
	You may also refer to upstream blocks: `proxy_pass http://myblock;`
	Instead of `http://`, you can use `https://` for secure traffic. Additional URI parts as well as the use of variables are allowed.
	Examples:
	`proxy_pass http://localhost:8080;`
	`proxy_pass http://127.0.0.1:8080;`
	`proxy_pass http://unix:/tmp/nginx.sock;`
	`proxy_pass https://192.168.0.1;`
	`proxy_pass http://localhost:8080/uri/;`
	`proxy_pass http://unix:/tmp/nginx.sock:/uri/;`
	`proxy_pass http://$server_name:8080;`

```
# Using an upstream block
upstream backend {
    server 127.0.0.1:8080;
    server 127.0.0.1:8081;
}
location ~* \.php$
{
    proxy_pass http://backend;
}
```

Directive	Description
`proxy_method` Context: `http`, `server`, `location`	Allows overriding the HTTP method of the request to be forwarded to the backend server. If you specify POST, for example, all requests forwarded to the backend server will be POST requests. Syntax: `proxy_method method;` Example: `proxy_method POST;`
`proxy_hide_header` Context: `http`, `server`, `location`	By default, as Nginx prepares the response received from the backend server to be forwarded back to the client, it ignores some of the headers, such as `Date`, `Server`, `X-Pad`, and `X-Accel-*`. With this directive, you can specify an additional header line to be hidden from the client. You may insert this directive multiple times with one header name for each. Syntax: `proxy_hide_header header_name;` Example: `proxy_hide_header Cache-Control;`
`proxy_pass_header` Context: `http`, `server`, `location`	Related to the preceding directive, this directive forces some of the ignored headers to be passed on to the client. Syntax: `proxy_pass_header headername;` Example: `proxy_pass_header Date;`

`proxy_pass_request_body`
`proxy_pass_request_headers`
Context: `http`, `server`, `location`
	Defines whether or not the request body and extra request headers should be passed on to the backend server.
	Syntax: `on` or `off`;
	Default: `on`

`proxy_redirect`
Context: `http`, `server`, `location`
	Allows you to rewrite the URL appearing in the Location HTTP header on redirections triggered by the backend server.
	Syntax: `off`, `default`, or the URL of your choice
	`off`: Redirections are forwarded **as it is**.
	`default`: The value of the `proxy_pass` directive is used as the hostname, and the current path of the document is appended. Note that the `proxy_redirect` directive must be inserted after the `proxy_pass` directive as the configuration is parsed sequentially.
	URL: Replace a part of the URL with another.
	Additionally, you may use variables in the rewritten URL.
	Examples:
	`proxy_redirect off;`
	`proxy_redirect default;`
	`proxy_redirect http://localhost:8080/ http://example.com/;`
	`proxy_redirect http://localhost:8080/wiki/ /w/;`
	`proxy_redirect http://localhost:8080/ http://$host/;`

Directive	Description

proxy_next_upstream
Context: http, server, location
	When proxy_pass is connected to an upstream block, this directive defines the cases where the requests should be abandoned and resent to the next upstream server of the block. The directive accepts a combination of values among the following:
	error : An error occurred while communicating or attempting to communicate with the server
	timeout : A timeout occurs during transfers or connection attempts
	invalid_header : The backend server returned an empty or invalid response
	http_500 , http_502 , http_503 , http_504 , http_404 : In case such HTTP errors occur, Nginx switches to the next upstream server
	off : Forbids the use of the next upstream server
	Examples:
	`proxy_next_upstream error timeout http_504;`
	`proxy_next_upstream timeout invalid_header;`

proxy_next_upstream_timeout Context: http, server, location	Defines the timeout to be used in conjunction with proxy_next_upstream . Setting this directive to 0 disables it. Syntax: Time value (in seconds)
proxy_next_upstream_tries Context: http, server, location	Defines the maximum number of upstream servers to be tried before returning an error message; to be used in conjunction with proxy_next_upstream . Syntax: Numeric value (default: 0)

Caching, buffering, and temporary files

Ideally, as much as possible, you should reduce the number of requests being forwarded to the backend server. The following directive will help you build a caching system as well as control the buffering options and the way Nginx handles the temporary files:

Directive	Description
proxy_buffer_size Context: http, server, location	Sets the size of the buffer for reading the beginning of the response from the backend server, which usually contains simple header data. The default value corresponds to the size of 1 buffer, as defined by the directive given previously (proxy_buffers). Syntax: Numeric value (size) Example: `proxy_buffer_size 4k;`

Directive	Description
`proxy_buffering`, `proxy_request_buffering` Context: `http`, `server`, `location`	Defines whether or not the response from the backend server should be buffered (or client requests, in the case of `proxy_request_buffering`). If set to `on`, Nginx will store the response data in memory using the memory space offered by the buffers. If the buffers are full, the response data will be stored as a temporary file. If the directive is set to `off`, the response is directly forwarded to the client. Syntax: `on` or `off` Default: `on`
`proxy_buffers` Context: `http`, `server`, `location`	Sets the amount and size of buffers that will be used for reading the response data from the backend server. Syntax: `proxy_buffers amount size;` Default: 8 buffers, 4 k or 8 k each depending on platform Example: `fastcgi_buffers 8 4k;`
`proxy_busy_buffers_size` Context: `http`, `server`, `location`	When the backend-received data accumulated in the buffers exceeds the specified value, the buffers are flushed and data is sent to the client. Syntax: `Numeric value` (size) Default: `2 * proxy_buffer_size`
`proxy_cache` Context: `http`, `server`, `location`	Defines a cache zone. The identifier given to the zone is to be reused in further directives. Syntax: `proxy_cache zonename;` Example: `proxy_cache cache1;`
`proxy_cache_key` Context: `http`, `server`, `location`	This directive defines the cache key; in other words, it differentiates one cache entry from another. If the cache key is set to `$uri`, as a result all requests with this `$uri` will work as a single cache entry. But that's not enough for most dynamic websites. You also need to include the query string arguments in the cache key so that `/index.php` and `/index.php?page=contact` do not point to the same cache entry. Syntax: `proxy_cache_key key;` Example: `proxy_cache_key "$scheme$host$request_uri $cookie_user";`
`proxy_cache_path` Context: `http`	Indicates the directory for storing the cached files as well as the other parameters. Syntax: `proxy_cache_path path [use_temp_path=on|off] [levels=numbers keys_zone=name:size inactive=time max_size=size];` The additional parameters are: `use_temp_path`: set this flag to `on` if you want to use the path defined via the `proxy_temp_path` directive. `levels`: Indicates the depth level of the subdirectories (usually 1:2 is enough) `keys_zone`: Lets you make use of the zone that you previously declared with the `proxy_cache` directive, and indicates the size that it will occupy in memory `inactive`: If a cached response is not used within the specified time frame, it is removed from the cache `max_size`: Defines the maximum size of the entire cache Example: `proxy_cache_path /tmp/nginx_cache levels=1:2 zone=zone1:10m inactive=10m max_size=200M;`

Directive	Description
`proxy_cache_methods` Context: `http`, `server`, `location`	Defines the HTTP methods eligible for caching. `GET` and `HEAD` are included by default, and cannot be disabled. Syntax: `proxy_cache_methods METHOD;` Example: `proxy_cache_methods OPTIONS;`
`proxy_cache_min_uses` Context: `http`, `server`, `location`	Defines the minimum amount of hits before a request is eligible for caching. By default, the response of a request is cached after one hit (subsequent requests with the same cache key will receive the cached response). Syntax: `Numeric value` Example: `proxy_cache_min_uses 1;`
`proxy_cache_valid` Context: `http`, `server`, `location`	This directive allows you to customize the caching time for different kinds of response codes. You may cache the responses associated with the `404` error codes for `1` minute, and on the opposite cache, `200 OK` responses for `10` minutes or more. This directive can be inserted more than once: `proxy_cache_valid 404 1m ;` `proxy_cache_valid 500 502 504 5m;` `proxy_cache_valid 200 10;` Syntax: `proxy_cache_valid code1 [code2…] time;`
`proxy_cache_use_stale` Context: `http`, `server`, `location`	Defines whether or not Nginx should serve stale cached data in certain circumstances (with regard to the gateway). If you use `proxy_cache_use_stale timeout` , and if the gateway times out, then Nginx will serve cached data. Syntax: `proxy_cache_use_stale [updating] [error] [timeout] [invalid_header] [http_500];` Example: `proxy_cache_use_stale error timeout;`
`proxy_max_temp_file_size` Context: `http`, `server`, `location`	Set this directive to `0` to disable the use of temporary files for requests eligible for proxy forwarding, or to specify a maximum file size. Syntax: `Size value` Default value: `1 GB` Example: `proxy_max_temp_file_size 5m;`
`proxy_temp_file_write_size` Context: `http`, `server`, `location`	Sets the write buffer size when saving temporary files to the storage device. Syntax: `Size value` Default value: `2 * proxy_buffer_size`

Directive	Description

`proxy_temp_path` Context: `http`, `server`, `location`	Sets the path of the temporary and cache store files.
	Syntax: `proxy_temp_path path [level1 [level2…]]`
	Examples:
	`proxy_temp_path /tmp/nginx_proxy;`
	`proxy_temp_path /tmp/cache 1 2;`

Limits, timeouts, and errors

The following directives will help you define the timeout behavior as well as various limitations regarding communication with the backend server:

Directive	Description
`proxy_connect_timeout` Context: `http`, `server`, `location`	Defines the backend server connection timeout. This is different from the read/send timeout. If Nginx is already connected to the backend server, the `proxy_connect_timeout` is not applicable. Syntax: `Time value` (in seconds) Example: `proxy_connect_timeout 15;`
`proxy_read_timeout` Context: `http`, `server`, `location`	The timeout for reading the data from the backend server. This timeout isn't applied to the entire response delay, but between two read operations instead. Syntax: `Time value` (in seconds) Default value: `60` Example: `proxy_read_timeout 60;`
`proxy_send_timeout` Context: `http`, `server`, `location`	This timeout is for sending data to the backend server. The timeout isn't applied to the entire response delay, but between two write operations instead. Syntax: `Time value` (in seconds) Default value: `60` Example: `proxy_send_timeout 60;`
`proxy_ignore_client_abort` Context: `http`, `server`, `location`	If set to `on`, Nginx will continue processing the proxy request, even if the client aborts its request. In the other case (`off`), when the client aborts its request, Nginx also aborts its request to the backend server. Default value: `off`
`proxy_intercept_errors` Context: `http`, `server`, `location`	By default, Nginx returns all error pages (HTTP status code 400 and higher) sent by the backend server directly to the client. If you set this directive to `on`, the error code is parsed and can be matched against the values specified in the `error_page` directive. Default value: `off`

Directive	Description
`proxy_send_lowat` Context: `http`, `server`, `location`	An option allowing you to make use of the `SO_SNDLOWAT` flag for TCP sockets under the BSD-based operating systems only. This value defines the minimum number of bytes in the buffer for output operations. Syntax: `Numeric value` (size) Default value: `0`
`proxy_limit_rate` Context: `http`, `server`, `location`	Allows you to limit the rate at which Nginx downloads the response from the backend proxy. Syntax: `Numeric value` (bytes per second)

SSL-related directives

If you are going to be working with SSL backend servers, the following directives will be useful to you:

Directive	Description
`proxy_ssl_certificate` Context: `http`, `server`, `location`	Sets the path of a PEM file that contains a certificate of authentication to an SSL backend. Syntax: `file path` Default value: `none`
`proxy_ssl_certificate_key` Context: `http`, `server`, `location`	Sets the path of the secret key file (PEM format) for authentication to an SSL backend. Syntax: `file path` Default value: `none`
`proxy_ssl_ciphers` Context: `http`, `server`, `location`	Sets the ciphers for SSL communication with the backend server. Run the following shell command to get the list of the available ciphers on your server: `openssl ciphers` . Syntax: `cipher names` Default value: `DEFAULT`
`proxy_ssl_crl` Context: `http`, `server`, `location`	Sets the path of the CRL (Certificate Revocation List) file in the PEM format allowing Nginx to verify the revocation state of the backend server's SSL certificate. Syntax: `file path` Default value: `-`

Directive	Description

`proxy_ssl_name`
Context:
`http`, `server`,
`location`
	Use this directive to override the server name when verifying the revocation state of the backend server's SSL certificate.
	Syntax: `character string`
	Default value: `equal to $proxy_host`

`proxy_ssl_password_file`
Context:
`http`, `server`,
`location`
	Sets the path of a file containing passphrases (one per line) which are tried in turn when loading the certificate key.
	Syntax: `file path`
	Default value: `-`

`proxy_ssl_server_name`
Context:
`http`, `server`,
`location`
	If you set this directive to `on` (as it is `off` by default), your server name will be communicated to the backend server as per the SNI protocol (Server Name Indication).
	Syntax: `on` or `off`
	Default value: `off`

`proxy_ssl_session_reuse`
Context:
`http`, `server`,
`location`
	This directive instructs Nginx to reuse the existing SSL sessions when communicating with the backend (thus reducing overhead). The official documentation recommends disabling this if the following errors start to show up in server logs: `SSL3_GET_FINISHED:digest check failed` .
	Syntax: `on` or `off`
	Default value: `on`

`proxy_ssl_protocols`
Context:
`http`, `server`,
`location`
	Sets the protocols to be used when communicating with SSL backends.
	Syntax: `proxy_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];`
	Default value: `TLSv1 TLSv1.1 TLSv1.2`

`proxy_ssl_trusted_certificate`
Context:
`http`, `server`,
`location`
	Sets the path of your trusted CA certificates (in the PEM format).
	Syntax: `file path`
	Default value: `-`

`proxy_ssl_verify`
Context:
`http`, `server`,
`location`
	If set to on, Nginx will verify the certificate of the SSL backend server.
	Syntax: `on` or `off`
	Default value: `off`

Directive	Description
`proxy_ssl_verify_depth` Context: `http` , `server` , `location`	If the `proxy_ssl_verify` directive is set to on, this sets the certificate chain verification depth. Syntax: `Numeric value` Default value: `1`

Other directives

Finally, the last set of directives available in the proxy module is uncategorized, and is as follows:

Directive	Description
`proxy_headers_hash_max_size` Context: `http` , `server` , `location`	Sets the maximum size for the proxy header's hash tables. Syntax: `Numeric value` Default value: `512`
`proxy_headers_hash_bucket_size` Context: `http` , `server` , `location`	Sets the bucket size for the proxy header's hash tables. Syntax: `Numeric value` Default value: `64`
`proxy_force_ranges` Context: `http` , `server` , `location`	When set to `on` , Nginx will enable byte-range support on the responses from the backend proxy. Syntax: `on` or `off` Default value: `off`
`proxy_ignore_headers` Context: `http` , `server` , `location`	Prevents Nginx from processing one of the following four headers from the backend server response: `X-Accel-Redirect` , `X-Accel-Expires` , `Expires` , and `Cache-Control` . Syntax: `proxy_ignore_headers header1 [header2…];`
`proxy_set_body` Context: `http` , `server` , `location`	Allows you to set a static request body for debugging purposes. Variables may be used in the directive value. Syntax: `String value` (any value) Example: `proxy_set_body test;`
`proxy_set_header` Context: `http` , `server` , `location`	This directive allows you to redefine the header values to be transferred to the backend server. It can be declared multiple times. Syntax: `proxy_set_header Header Value;` Example: `proxy_set_header Host $host;`

`proxy_store`
Context: `http` , `server` , `location`
	Specifies whether or not the backend server response should be stored as a file. Stored response files can be reused for serving other requests.
	Possible values: `on` , `off` , or a path relative to the document root (or alias). You may also set this to `on` and define the `proxy_temp_path` directive.
	Examples:
	`proxy_store on;`
	`proxy_temp_path /temp/store;`

Directive	Description
`proxy_store_access` Context: `http`, `server`, `location`	This directive defines file access permissions for the stored response files. Syntax: `proxy_store_access [ user: [ r | w | rw ]][ group: [ r | w | rw ]][ all: [ r | w | rw ]] ;` Example: `proxy_store_access user:rw group:rw all:r;`
`proxy_http_version` Context: `http`, `server`, `location`	Sets the HTTP version to be used for communicating with the proxy backend. HTTP 1.0 is the default value, but if you are going to enable `keepalive` connections, you might want to set this directive to `1.1` . Syntax: `proxy_http_version 1.0 | 1.1;`

`proxy_cookie_domain`
`proxy_cookie_path`
Context: `http`, `server`, `location`
	Applies an on-the-fly modification to the domain or path attributes of a cookie (case insensitive).
	Syntaxes: `proxy_cookie_domain off | domain replacement;`
	`proxy_cookie_path off | domain replacement ;`

Variables

The proxy module offers several variables that can be inserted in various locations, for example, in the `proxy_set_header` directive or in the logging-related directives such as `log_format` . The available variables are:

- `$proxy_host` : Contains the hostname of the backend server used for the current request.
- `$proxy_port` : Contains the port of the backend server used for the current request.
- `$proxy_add_x_forwarded_for` : This variable contains the value of the `X-Forwarded-For` request header, followed by the remote address of the client. Both values are separated by a comma. If the `X-Forwarded-For` request header is unavailable, the variable only contains the client remote address.
- `$proxy_internal_body_length` : Length of the request body (set with the `proxy_set_body` directive or `0`).

Next Section (/mapt/book/networking-and-servers/9781785280337/6/ch06lvl1sec34/configuring-apache-and-nginx)