Configuring NGINX Proxy

In the preceding section, we developed a Python application running on the `5000` port. Now, we want to configure NGINX to Proxy the server using the following directives. We will also try to cache the response from the Python server by enabling the proxy cache.

In addition to the directives, the NGINX Proxy module also creates the `$proxy_host` , `$proxy_port` , and `$proxy_add_x_forward_for` variables that can be used at various places such as Proxy headers.

proxy_pass

This directive can be used to specify the protocol, address, and URL (optional) of the upstream server. The protocol takes the value `http` or `https` . The address can be in any of the following forms:

- It can be a domain name or IP address along with the port, that is, `location:port`
- It can be a `unix` socket address specified with a `unix:` prefix and enclosed within colons, for example, `unix:/var/run/server.sock:`
- It can be a group of servers created using the NGINX upstream directive

This directive is only available under the `location and if` and `in location` sections of an NGINX configuration. Here's an example:

Copy

```
location /myloc/{
proxy_pass http://unix:/var/run/server.sock:/loc;
}
```

proxy_method

This directive sets the HTTP method passed to the proxy server instead of the version specified by the client. The directive is available under the `http` , `server` , and `location` sections of an NGINX configuration.

proxy_set_header

This directive is used to define header fields passed in the upstream request. The fields can have values in the form of variables and text. If there is a field with an empty string (`""`) as its value, it will not be passed in the upstream request. The directive is available under the `http` , `server` , and `location` sections of an NGINX configuration. Here's an example:

Copy

```
location /myloc/{
proxy_set_header Host $proxy_host;
proxy_set_header Accept-Charset UTF-8;
}
```

By default, NGINX sets only two fields, namely `Host` with the `$proxy_host` value and `connection` with the `close` value.

proxy_http_version

This directive sets the version of HTTP for requests to the upstream server. By default, the value is 1.0. Version 1.1 should be used for `keepalive` connections. The directive is available under the `http` , `server` , and `location` sections of an NGINX configuration.

> **Note**
>
> In order to work with upstream `keepalive` connections, the connection header needs to be set. NGINX, by default, sets the header field to close. Use `proxy_set_header` to set the `keepalive` value in the connection header. Here's an example:

Copy

```
location /myloc/{
      proxy_set_header Connection keep-alive;
      proxy_http_version 1.1
}
```

proxy_pass_request_headers / proxy_pass_request_body

These directives indicate whether the request header and body should be passed to the upstream server or not. By default, both the directives are set to `on` . The directives are available under the `http` , `server` , and `location` sections of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_pass_request_headers on;
proxy_pass_request_body on;
}
```

proxy_ignore_headers

This directive enables NGINX to ignore the processing of certain upstream response headers, namely `X-Accel-Expires` , `Expires` , `Cache-Control` , `Set-Cookie` , `Vary` , `X-Accel-Redirect` , `X-Accel-Charset` , `X-Accel-Buffering` , and `X-Accel-Limit-Rate` . If not disabled, the fields cause the following behavior:

- The caching is controlled by `X-Accel-Expires` , `Expires` , `Cache-Control` , `Set_Cookie` , and `Vary` header fields
- An internal redirect is performed using `X-Accel-Redirect`
- The response character set is controlled by `X-Accel-Charset`
- The buffering of responses is controlled by `X-Accel-Buffering`
- The rate of response transmission to the client is controlled by `X-Accel-Limit-Rate`

This directive is available under the `http` , `server` , and `location` sections of an NGINX configuration.

proxy_connect_timeout / proxy_send_timeout / proxy_read_timeout

The `proxy_connect_timeout` directive sets the timeout to establish a connection between NGINX and the Proxy server.

The `proxy_send_timeout` directive sets the timeout to write a request to the Proxy server. The `proxy_read_timeout` directive sets the timeout to read a response from the Proxy server.

All the directives have a default value of 60 seconds and are available under the `http`, `server`, and `location` sections of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_send_timeout 30s;
proxy_connect_timeout 30s;
proxy_read_timeout 30s;
}
```

It is important to note that the connection will be closed if the timeout specifies any `proxy_send_timeout` directives and `proxy_read_timeoutdirective` expires.

proxy_store / proxy_store_access

The `proxy_store` directive enables saving responses from upstream to the disk as files. By default, the directive is `off`. The directive can be turned `on`, which will save the response in the directive's root location. The directive can also specify a path, using variables, which is used to determine the file's location. The directive is available under the `http`, `server`, and `location` sections of an NGINX configuration:

Copy

```
location /myloc{
proxy_store on;
proxy_store_access user:rw group:rw all:r;
root /location;
}
```

The `proxy_store_access` directive can be used to specify the permissions for the created files. By default, the directive grants read-write permissions to the file owner only. This directive is available under the `http`, `server`, and `location` sections of an NGINX configuration.

proxy_cache_path

This directive is used to define a cache. The directive has a couple of arguments that can be used to configure the cache behavior. It is mandatory to specify the disk location, cache name, and cache size. The cache also has an `inactive` time period, that is, the time after which data will be purged from the cache. The `levels` parameter can be used to define the hierarchy while writing data to the cache. The directive is only available under the `http` section of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_cache_path /var/cache/NGINX keys_zone=mycache:10m inactive=15m;
}
```

By default, the `inactive` period is set to `10` minutes. NGINX also runs a cache manager, which will remove the oldest entry once the cache reaches its maximum size as defined by the optional `max_size` parameter.

Cache loading is accomplished by a cache loader process. The option parameter specified by the directive, namely, `loader_files`, `loader_sleep`, and `loader_threshold` can alter the cache loading behavior.

proxy_cache_key

This directive defines the key for cache lookup. The directive is available under the `http`, `server`, and `location` sections of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_cache_key "$request_method$request_uri";
}
```

proxy_cache

This directive enables the use of a previously defined memory cache using the proxy `cache_path` directive. The `proxy_cache` directive identifies the memory zone by the name specified in the configuration. By default, the cache is set to `off`. The directive is available under the `http`, `server`, and `location` sections of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_cache mycache;
}
```

proxy_cache_valid

This directive enables NGINX to define the cache time period depending on the HTTP response code. If the HTTP response code is not specified, then only the `200`, `301`, and `302` response codes are cached. The directive also specifies any parameter that can be used to cache all response code files. The directive is available under the `http`, `server`, and `location` sections of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_cache_valid 200 302 10m;
proxy_cache_valid any 1m;
}
```

Cache control fields in the response header have a higher precedence than the directive. The processing of the response header can be turned `off` using the `proxy_ignore_headers` directive.

proxy_no_cache

This directive defines the conditions under which the response will not be saved to the cache. The directive takes a list of parameters that are evaluated at runtime. If any of them is nonempty and nonzero, the response will not be cached. The directive is available under the `http`, `server`, and `location` sections of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_no_cache $http_pragma $cookie_nocache;
}
```

The preceding configuration will not cache the response if the header contains the Pragma field, or if a `no-cache` cookie has been set.

proxy_cache_bypass

This directive defines the conditions in which NGINX will not perform cache lookup. The directive takes a list of parameters that are evaluated at runtime. If either of them is nonempty and nonzero, the response will not be cached. The directive is available under the `http`, `server`, and `location` sections of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_bypass_cache $http_pragma $cookie_nocache;
}
```

The preceding configuration will not perform cache lookup if the header contains the Pragma field, or if a no-cache cookie has been set.

proxy_cache_methods

This directive defines the list of HTTP methods that will be cached. By default, the `GET` and `HEAD` methods are cached. The directive is available under the `http`, `server`, and `location` sections of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_cache_methods GET HEAD;
}
```

proxy_cache_use_stale

This directive defines the error conditions under which a stale response can be used from the cache. The directive defines all error conditions served by NGINX as parameters. By default, the directive is set to `off`. This directive is available under the `http`, `server`, and `location` sections of an NGINX configuration. Here's an example:

Copy

```
http{
proxy_cache_use_stale http_500 http_503;
}
```

Setting up the server

The following configuration enables NGINX to serve content from the Flask application. NGINX is also configured to serve the CSS file, which is a part of the Python application. The cache directives are used to enable the support of the Proxy cache for the Python application. Here's how NGINX is configured for Flash:

Copy

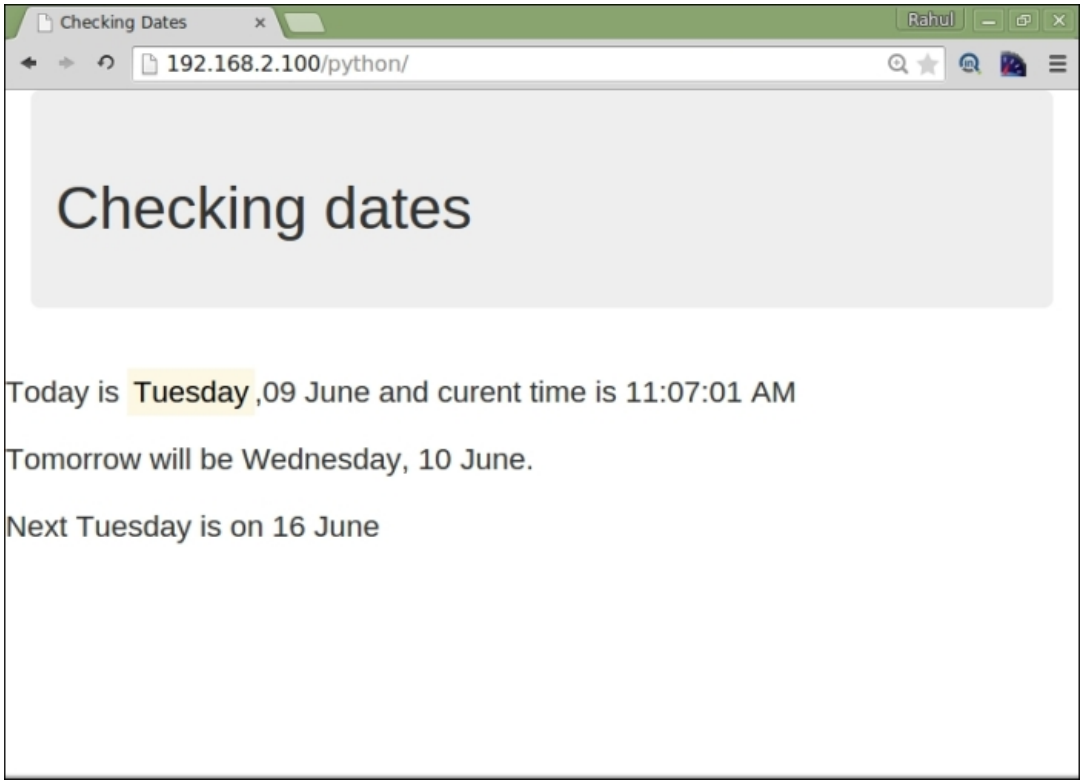
```
http{
proxy_cache_path /etc/NGINX/pythoncachekeys_zone=pythonCache:100m inactive=60m;
proxy_cache_key "$request_method$host$request_uri";
server{
location /python/css/ {
alias "/code-path/css/";
}

location /python/ {
proxy_pass http://127.0.0.1:5000/;
proxy_cache pythonCache;
proxy_cache_valid any 1m;
add_header X-Proxy-Cache $upstream_cache_status;
}
# Rest NGINX configuration omitted for brevity
}
}
```

The preceding configuration does the following things:

- The `proxy_cache_path` directive creates a cache for the specified size
- The `proxy_cache_key` directive defines the key format for cache lookup
- The location directive for `/python/css` enables NGINX to serve all static content
- The location directive for `/python/` forwards all requests to the server running at `127.0.0.1:5000`
- The `proxy_cache` directive in the location section enables the cache for all requests served by the block
- The `proxy_cache_valid` directive caches a successful response for 1 minute

Additionally, the server adds the `X-Proxy-Cache` field in the response header, indicating the status of a cache hit.

Now, access the location `http://server/python/`. A screenshot similar to the following one will be displayed:



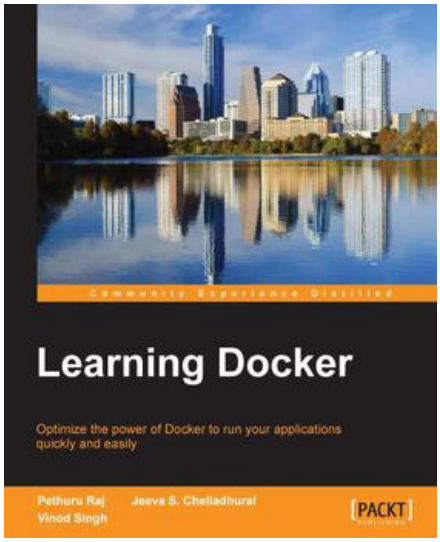
Since we have added a field in the header, it can be used to validate whether the request has been served from the cache:

Copy

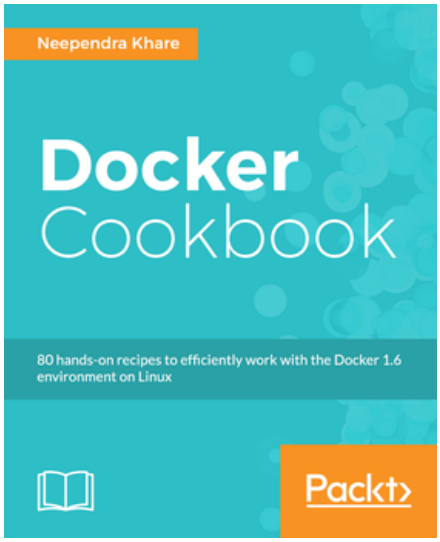
```
$ curl -I http://192.168.2.100/python/
HTTP/1.1 200 OK
Server: NGINX/1.7.12 (Ubuntu)
Date: Thu, 30 Apr 2015 08:50:30 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Proxy-Cache: MISS

$ curl -I http://192.168.2.100/python/
.. .. .. ..
X-Proxy-Cache: HIT
```

Next Section (/mapt/book/networking-and-servers/9781785281839/6/ch06lvl1sec37/using-memcache)

Related titles



(/mapt/book/virtualization_and_cloud/9781784397937)



(/mapt/book/virtualization_and_cloud/9781783984862/1/ch01lvl1sec09/introduction)