Using Memcache

Memcache is a generic-purpose memory-caching system. It is often used to speed up slow data access. The NGINX memcached module presents various directives that can be configured to serve content directly from Memcache, thus avoiding requests to the upstream server.

In addition to the directives, the module also creates the `$memcached_key` variable, which is used to perform cache lookup. Before using the Memcache lookup, a value must be set in the `$memcached_key` variable, which is determined from the request URL.

Note

It is important to note that the NGINX Memcache module only performs the lookup for the specified `$memcached_key` variable. It does not perform any write backs to the cache for the specified key. A value corresponding to the key should be in the cache beforehand, written by means external to NGINX.

memcached_pass

This directive is used to specify the location of the memcached server. The address can be specified in any of the following ways:

- A domain name or IP address, along with an optional port
- A Unix domain socket specified with the `unix:` prefix
- A group of servers created using the NGINX upstream directive

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

Copy

```
location /myloc/{
set $memached_key $uri;
memcached_pass localhost:11211;
}
```

memcached_connect_timeout / memcached_send_timeout / memcached_read_timeout

The `memcached_connect_timeout` directive sets the timeout to establish a connection between NGINX and the memcached server.

The `memcached_send_timeout` directive sets the timeout to write a request to the memcached server. The `memcached_read_timeout` directive sets the timeout for reading a response from the memcached 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:

```
http{
memcached_send_timeout 30s;
memcached_connect_timeout 30s;
memcached_read_timeout 30s;
}
```

It is important to note that the connection will be closed if the timeout specifies any `memcached_send_timeout` directive, and `memcached_read_timeout` directive expires.

memcached_bind

This directive can be used to specify and address in order to communicate with the memcached server. By default, the directive is set to `off` . Thus, NGINX auto-assigns the local IP address for communication. The directive is available under the `http` , `server` , and `location` sections of an NGINX configuration.

Setting up the server

In this section, we will configure NGINX to serve requests from the memcached server for the Python application that was developed earlier in the chapter. When NGINX is unable to find the key in the memcached server, it forwards the request to the Python application. The application then serves the request and puts the responses in the memcached server for future requests. Here's how we configure NGINX for Memcache:

```
alias "/code/location/css/";
}

location /python/ {
set $memcached_key "$request_method$request_uri";
charset utf-8;
memcached_pass 127.0.0.1:11211;
error_page 404 502 504 = @pythonfallback;
default_type text/html;
}

location @pythonfallback {
rewrite ^/python/(.*) /$1 break;
proxy_pass http://127.0.0.1:5000;
proxy_set_header X-Cache-Key "$request_method$request_uri";
}
# Rest NGINX configuration omitted for brevity

}
```

The preceding NGINX configuration does the following things:

- Setting `$memcached_key` for cache lookup
- Specifying the address of the memcached server using the `memcached_pass` directive
- Using a fallback to serve content from Python server upstream for errors such as `404`
- Additionally requesting the Python server upstream, thus setting a header field for the lookup key

Note

In order to interact with the memcached server from Python, we would require the `python3-memcached` package. Install it using the `pip` command:

Copy

```
$ sudopython3 -m pip install python3-memcached
```

Copy

```
cache = memcache.Client(["127.0.0.1:11211"])

@app.after_request
defprocessResponse(response):
cachekey = request.headers.get('X-Cache-Key')
if(request.method=="GET"):
cache.set(cachekey,str(response.data).encode("utf8"))
return response

class AppdateTime :
def __init__(self, day, date, time):
self.day = day
self.date = date
self.time = time

@app.route("/")
def hello():

# Rest Python implementation removed for brevity
```

The aforementioned Python code does the following:

- Line 4 imports the `memcache` module
- Next, we make the connection to the `memcached` server running on `127.0.0.1:11211`
- The `processResponse` method marked with `@app.after_request` executes for every request
- The `processResponse` method adds the response to the memcached server against the key specified in the request header (`X-Cache-key`)

Measuring gains

It is time to test the changes and make sure that they have given any performance gain. It is important to note that we cannot apply the baselines developed in the previous chapter as the complete setup is quite different. We are no longer serving static content; here we are building dynamic content from a proxy server. Thus, first we need to perform a couple of tests in order to build a baseline with only the proxy server:

Copy

```
$ siege -b -c 250 -r 50 -q http://192.168.2.100/python/
done.
Transactions: 12461 hits
Availability: 99.69 %
Data transferred: 6.03 MB
Response time: 0.33 secs
Transaction rate: 381.65 trans/sec
Throughput: 0.18 MB/sec
Concurrency: 127.28
Successful transactions: 12461
Failed transactions: 39
```

Now, modify the NGINX configuration to serve content using the memcached server. Validated by the new numbers, the server will take much more load. Increase the concurrency so that you can get to know the new limits:

Copy

```
$ siege -b -c 900 -r 50 -q http://192.168.2.100/python/
done.

Transactions: 45000 hits
Availability: 100.00 %
Elapsed time: 18.15 secs
Data transferred: 23.82 MB
Response time: 0.27 secs
Transaction rate: 2479.34 trans/sec
Throughput: 1.31 MB/sec
Concurrency: 669.05
Successful transactions: 45000
Failed transactions: 0
```

Next Section (/mapt/book/networking-and-servers/9781785281839/6/ch06lvl1sec38/summary)