2020

NGINX Caching E-learning Lab Guide 3

STUDENT LAB GUIDE

NGINX, INC. | 795 Folsom Street, San Francisco, CA. 94107

Table of Contents

Getting Started

This lab guide provides step-by-step instructions for lab exercises. Each lab corresponds to a module covered in class and provides you with hands-on experience working with the NGINX Plus as a caching server.

Course Pre-requisites

This course is intended to provide training on NGINX Plus caching configurations to IT professionals who have completed the NGINX Core course. It is assumed students have familiarity with:

- IT operations
- Web servers
- Linux
- Text editor: Vim, Vi, Emacs, etc.
- Networking topologies

Log into Hosted Environment

- Open your email and find the lab systems assigned to you. Your lab systems have NGINX Plus pre-installed.

- There is a login for the machine with a username ***student*** and the password ***student***.

Lab 3: Controlling Cached Responses

Learning Objectives

By the end of the lab you will be able to:

- Configure and test the number of requests after which a response will be cached
- Enable cache revalidation
- Enable NGINX to serve stale content

Exercise 1: Delayed Caching

Overview

In this exercise we will examine the effect of only caching a response after a minimum number of requests

1. Open a terminal window on your lab system, **NGINX_Cache1** and then open the ***Snippets_Lab3*** file so it will be easy to copy and paste commands.

   Note: Only if you do not have the **Snippets_Lab3** file on your desktop then run:

   ```
   /home/student/.Lab3/startup_Lab3
   ```

2. In the `server` context of your `default.conf` configuration file, set the number of requests after which the response will be cached to 5. Add this directive after the add_header directive.

   ```
   $ sudo vim /etc/nginx/conf.d/default.conf

   server {
       ...
       proxy_cache_min_uses 5;
   }
   ```

Note: Your file should be similar to this:

```
server {
    listen 80 default_server;
    root /usr/share/nginx/html;
    index index.html;
    status_zone proxy;

    access_log /var/log/nginx/py-mongo.access.log json_log;

    add_header X-Cache-Status $cache_status;

    proxy_cache_min_uses 5;

    proxy_cache_key $scheme$host$request_uri;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    location / {
        proxy_cache py-mongo;
        proxy_cache_valid 200 5m;
        proxy_pass http://py-mongo;
    }
```

3. Save the file and reload NGINX:

   ```
   $ sudo nginx -s reload
   ```

4. Run the `curl` command. What is the value of the `X-Cache-Status` header?

   ```
   $ curl -I http://localhost/
   ```

5. Repeat the same command 5 more times. What can you observe in the response?

   We should see that the first 5 times we make the request, the value of `X-Cache-Status` is **MISS**. This is because the `proxy_cache_min_uses` directive is instructing NGINX only to cache the response to a request if there has been a minimum of 5 requests for that resource.

```
NGINX1$ curl -I http://localhost
HTTP/1.1 200 OK
Server: nginx/1.19.5
Date: Thu, 04 Mar 2021 19:30:05 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 2972
Connection: keep-alive
X-Cache-Status: MISS
```

On the **sixth** request, we should see the value change to **HIT**, as we have now met the minimum requirement. NGINX would have cached the response after the 5th request and therefore on our 6th request, the response was served from the cache.

```
NGINX1$ curl -I http://localhost
HTTP/1.1 200 OK
Server: nginx/1.19.5
Date: Thu, 04 Mar 2021 19:30:29 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 2972
Connection: keep-alive
X-Cache-Status: HIT
```

Troubleshooting Note: If you don't get 5 MISS and then 1 HIT, change the 5m to 1m in the directive: proxy_cache_valid 200 5m;
Then reload the NGINX configuration and repeat the curl command 6 times.

6. Comment out the `proxy_cache_min_uses` directive in the file /etc/nginx/conf.d/default.conf. Save the file and reload NGINX.

```
$ sudo vim /etc/nginx/conf.d/default.conf

server {
    ...
#    proxy_cache_min_uses 5;
}

$ sudo nginx -s reload
```

Exercise 2: NGINX Cache Revalidation

Overview

In this lab we will examine the use of the `proxy_cache_revalidate` directive along with the `proxy_ignore_headers` directive and how they impact the way NGINX handles cached content.

Steps

1. On your lab system run the following curl command and observe the response headers.

```
$ curl -I localhost/static/goku.gif
```

```
student1@ip-172-31-19-169:~$ curl -I localhost/static/goku.gif
HTTP/1.1 200 OK
Server: nginx/1.15.10
Date: Tue, 25 Feb 2020 05:53:27 GMT
Content-Type: image/gif
Content-Length: 236756
Connection: keep-alive
Last-Modified: Mon, 21 May 2018 18:01:50 GMT
Cache-Control: public, max-age=43200
Expires: Tue, 25 Feb 2020 17:53:27 GMT
ETag: "1526925710.0-236756-3865906422"
Accept-Ranges: bytes
X-Cache-Status: MISS
```

Notice the `Cache-Control` header with `max-age` set to 43200, as well as the `Expires` header. NGINX will prioritize these headers when determining how long to cache the content for. Therefore, even though we have `proxy_cache_valid` specified in our configuration, the value we specify won't be applied. We need to tell NGINX to ignore the `Cache-Control` and `Expires` headers.

2. Open the `default.conf` configuration file.

   `$ sudo vim /etc/nginx/conf.d/default.conf`

3. In the `server` context, after the proxy_set_header directives, add the following:

   ```
   proxy_ignore_headers Cache-Control Expires;
   ```

4. Change the `proxy_cache_valid` duration from 5 minutes to 30 seconds.

   ```
   proxy_cache_valid 200 30s;
   ```

5. Save the configuration and reload NGINX. Your file should be similar to this (this is not the whole file, just the part that changed):

```
server {
    listen 80 default_server;
    root /usr/share/nginx/html;
    index index.html;
    status_zone proxy;

    access_log /var/log/nginx/py-mongo.access.log json_log;

    add_header X-Cache-Status $cache_status;

#    proxy_cache_min_uses 5;

    proxy_cache_key $scheme$host$request_uri;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_ignore_headers Cache-Control Expires;

    location / {
        proxy_cache py-mongo;
        proxy_cache_valid 200 30s;
        proxy_pass http://py-mongo;
    }
```

```
$ sudo nginx -s reload
```

6. Open a separate terminal on your lab system machine and follow the **pymongo-app** container logs. Hit enter a few times so it is easier to know when new entries are written to the log.

```
$ docker logs -f pymongo-app
```

7. Run the following curl command. What do you see in the **`X-Cache-Status`** response header and what do you notice in the docker container log?

   ```
   $ curl -I localhost/static/goku.gif
   ```

   You should see that the value is "**MISS**".

   ```
   SYS1$ curl -I localhost/static/goku.gif
   HTTP/1.1 200 OK
   Server: nginx/1.17.9
   Date: Sat, 30 May 2020 00:20:38 GMT
   Content-Type: image/gif
   Content-Length: 236756
   Connection: keep-alive
   Last-Modified: Mon, 21 May 2018 18:01:50 GMT
   Cache-Control: public, max-age=43200
   Expires: Sat, 30 May 2020 12:20:38 GMT
   ETag: "1526925710.0-236756-3865906422"
   X-Cache-Status: MISS
   Accept-Ranges: bytes
   ```

   In the container log, we should observe one additional line representing our request. The response code should indicate "**200**".

   ```
   172.19.0.1 - - [27/Feb/2020 04:05:28] "GET /static/goku.gif HTTP/1.0" 200 -
   ```

   Troubleshooting:
   If you get a HIT then make sure to wait more than 30 seconds before trying the curl command again. You can also run “sudo nginx -s reload” again and then wait at least 30 seconds before trying the curl command. You can use the ***`date`*** command several times to make sure you are waiting at least 30 seconds.

8. Repeat the same curl command. What were the results this time?

   The `X-Cache-Status` header value should be "**HIT**". On the container logs, we don't observe any additional lines because the request is not being proxied to our application since it is being served from the cache.

```
NGINX1$ curl -I localhost/static/goku.gif
HTTP/1.1 200 OK
Server: nginx/1.19.5
Date: Tue, 27 Apr 2021 01:21:41 GMT
Content-Type: image/gif
Content-Length: 236756
Connection: keep-alive
Last-Modified: Mon, 21 May 2018 18:01:50 GMT
Cache-Control: public, max-age=43200
Expires: Tue, 27 Apr 2021 13:21:22 GMT
ETag: "1526925710.0-236756-3865906422"
X-Cache-Status: HIT
Accept-Ranges: bytes
```

Note: If you received EXPIRED instead of HIT status, then just do another curl command quickly and you should get HIT. Then move on to step 9.

9. Now wait 30 seconds and then repeat the same curl command. What can you observe this time?

   The `X-Cache-Status` will now be "**EXPIRED**". This is because we have set the `proxy_cache_valid` directive to be 30 seconds, so our content only stays fresh for that duration. After that it becomes stale.

```
NGINX1$ curl -I localhost/static/goku.gif
HTTP/1.1 200 OK
Server: nginx/1.19.5
Date: Tue, 27 Apr 2021 01:21:59 GMT
Content-Type: image/gif
Content-Length: 236756
Connection: keep-alive
Last-Modified: Mon, 21 May 2018 18:01:50 GMT
Cache-Control: public, max-age=43200
Expires: Tue, 27 Apr 2021 13:21:59 GMT
ETag: "1526925710.0-236756-3865906422"
X-Cache-Status: EXPIRED
Accept-Ranges: bytes
```

On the container log, we can also observe a request being logged with the response code of "200".

```
student1@ip-172-31-24-40:/etc/nginx/conf.d$ curl -I localhost/static/goku.gif
HTTP/1.1 200 OK
...
...
Cache-Control: public, max-age=43200
Expires: Thu, 27 Feb 2020 16:06:45 GMT
ETag: "1526925710.0-236756-3865906422"
X-Cache-Status: EXPIRED
Accept-Ranges: bytes
```

10. Edit the **default.conf** file and add in the `proxy_cache_revalidate` directive in the server context, after the proxy_ignore_headers directive.

```
server {
        listen 80 default_server;
        root /usr/share/nginx/html;
        index index.html;
        status_zone proxy;

        access_log /var/log/nginx/py-mongo.access.log json_log;

        add_header X-Cache-Status $cache_status;

        proxy_cache_min_uses 5;
        proxy_cache_key $scheme$host$request_uri;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_ignore_headers Cache-Control Expires;
        proxy_cache_revalidate on;
```

11. Save the configuration and reload NGINX.

```
$ sudo nginx -s reload
```

12. Run the curl command twice. What can you see in the response headers and on the container log?

```
$ curl -I localhost/static/goku.gif
```

You should see the `X-Cache-Status` value as "**REVALIDATED**". This indicates that NGINX sent a validation request to the origin server to check whether the cached entry is still valid, and the origin server responded with a "**304 NOT MODIFIED**" status.

On the container log, we should see that request was logged with a response code 304.

Container Log

```
172.19.0.1 - - [27/Feb/2020 05:34:20] "GET /static/goku.gif HTTP/1.0" 304 -
```

Application Response

```
student1@ip-172-31-24-40:/etc/nginx/conf.d$ curl -I localhost/static/goku.gif
HTTP/1.1 200 OK
Server: nginx/1.15.10
Date: Thu, 27 Feb 2020 05:34:20 GMT
Content-Type: image/gif
Content-Length: 236756
Connection: keep-alive
Last-Modified: Mon, 21 May 2018 18:01:50 GMT
Cache-Control: public, max-age=43200
Expires: Thu, 27 Feb 2020 17:33:20 GMT
ETag: "1526925710.0-236756-3865906422"
X-Cache-Status: REVALIDATED
Accept-Ranges: bytes
```

Exercise 3: Serving Stale Content

Overview

In this exercise, you will configure NGINX to serve stale content from the cache

Steps

1. Run the following curl command. Observe the `X-Cache-Status` response header.

   ```
   $ curl -I localhost/static/goku.gif
   ```

   You should see a value of **"MISS"** or **"REVALIDATED"** if not enough time has passed.

2. Repeat Step 1. This time the X-Cache-Status should be "**HIT".** We now have a cached copy of the goku.gif image.

3. Stop the pymongo-app Docker container

   ```
   $ sudo docker stop pymongo-app
   ```

   Note: This will stop the *docker logs -f pymongo-app* command in your second terminal.

4. Wait for 30 seconds and the repeat the curl command in Step 1. What can you observe in the response?

   We should get a **502 Bad Gateway** error because we have shut down our application container. When NGINX tries to proxy the request through it cannot reach the application.

   Note that we chose to wait for 30 seconds so that our cached entry will have expired.

```
student1@ip-172-31-29-105:/etc/nginx/conf.d$ curl -I localhost/static/goku.gif
HTTP/1.1 502 Bad Gateway
Server: nginx/1.15.10
Date: Tue, 17 Mar 2020 13:27:02 GMT
Content-Type: text/html
Content-Length: 158
Connection: keep-alive
```

5. Open the default.conf file and add the following line into the `server` context, after the proxy_cache_revalidate directive:

   ```
   proxy_cache_use_stale error;
   ```

   Your file should be similar to this (Note: This is not the whole file):

```
#    proxy_cache_min_uses 5;

    proxy_cache_key $scheme$host$request_uri;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_ignore_headers Cache-Control Expires;
    proxy_cache_revalidate on;
    proxy_cache_use_stale error;

    location / {
        proxy_cache py-mongo;
        proxy_cache_valid 30s;
        proxy_pass http://py-mongo;
    }
```

6. Save the file and reload NGINX.

7. Repeat Step 1 with the curl command. What can you see in the X-Cache-Status value?

```
student1@ip-172-31-29-105:/etc/nginx/conf.d$ curl -I localhost/static/goku.gif
HTTP/1.1 200 OK
Server: nginx/1.15.10
Date: Tue, 17 Mar 2020 13:28:11 GMT
Content-Type: image/gif
Content-Length: 236756
Connection: keep-alive
Last-Modified: Mon, 21 May 2018 18:01:50 GMT
Cache-Control: public, max-age=43200
Expires: Wed, 18 Mar 2020 01:26:12 GMT
ETag: "1526925710.0-236756-3865906422"
X-Cache-Status: STALE
Accept-Ranges: bytes
```

   The value is **STALE.** Our cached content has expired but we have told NGINX to serve stale content in the event of an error with the upstream server.

8. Start the **pymongo-app** Docker container

```
$ docker start pymongo-app
```

9. Run step 1 again. What is the value of the X-Cache-Status header?

   We should get the REVALIDATED value as NGINX is now able to reach the upstream server, which is our container application. Our previous cached entry had expired but we've previously configured NGINX to check with the upstream server to see whether the file had been updated since. In this case, the server responded with a 304 NOT MODIFIED so NGINX was able to revalidate the cache entry.