2020

NGINX Caching E-learning Lab Guide 1

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 1: Using the Browser Cache

Learning Objectives

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

- View request and response headers in your browser

Exercise 1: View caching response headers

Overview

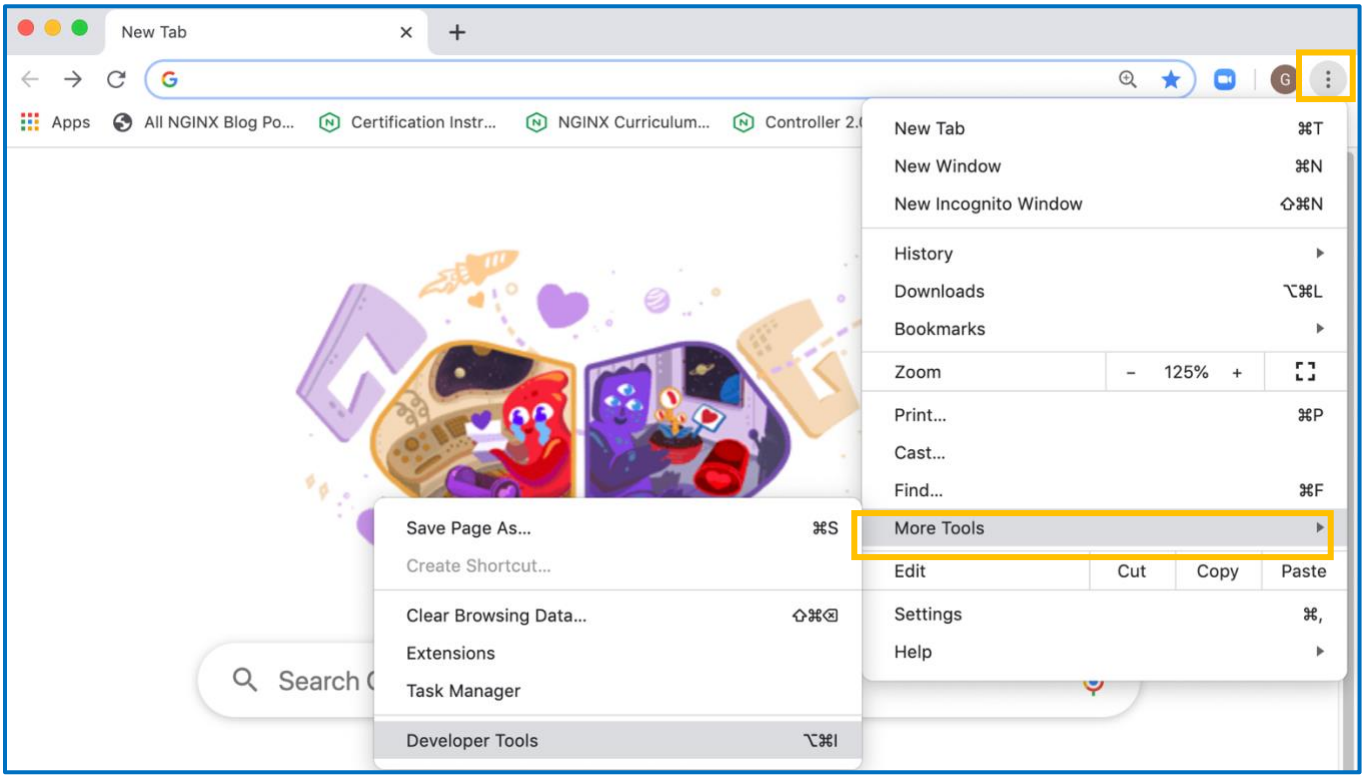
In this exercise, you use your browser with developer tools to send HTTP requests to a web site. You observe how response headers are used to control caching behavior in the browser.

We recommend the Chrome browser. Different browsers implement caching in slightly different ways. The following steps assume you are using Chrome.

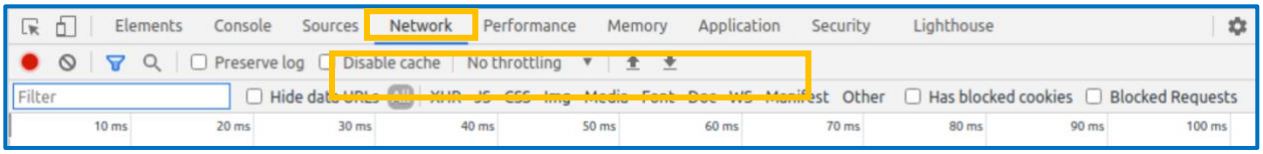
Steps

1. Open the Chrome browser and click the "**kebab**" menu to choose **More Tools** and then choose **Developer Tools**.

   Note: Lab 1 can be done on your local system using a browser, or you can use the Chrome browser that is on your first lab system.

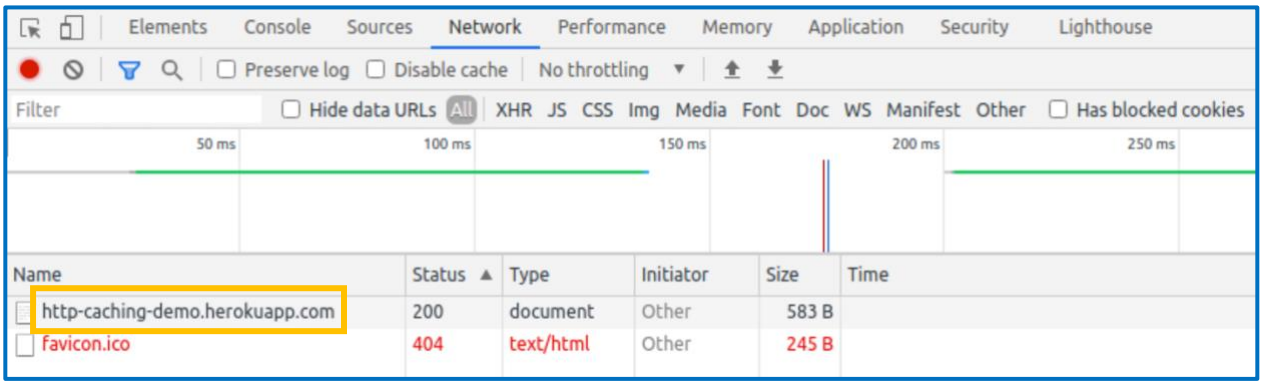


2. Click the **Network** tab.



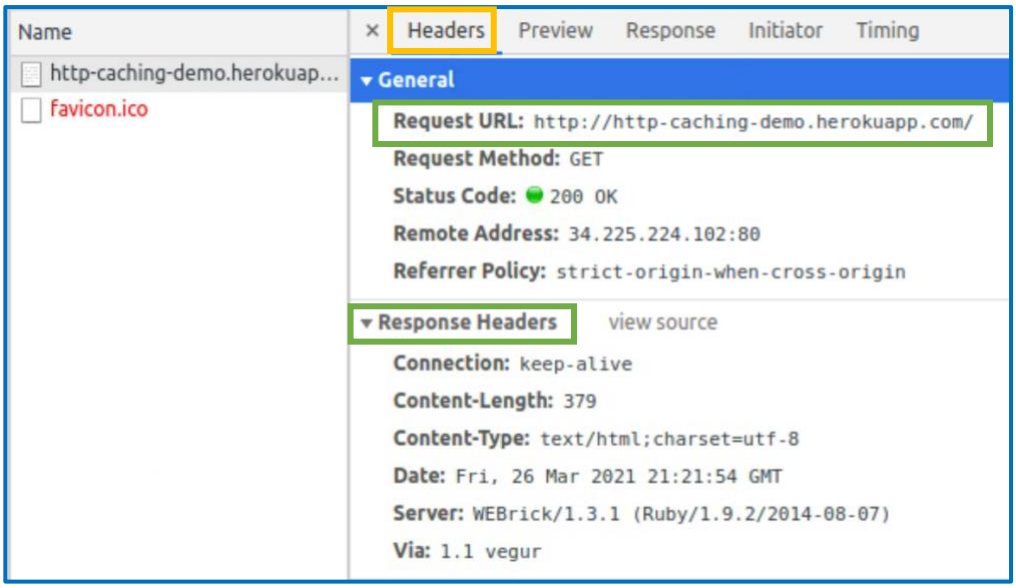
3. Enter the following in the browser's address bar:

   `http://http-caching-demo.herokuapp.com/`

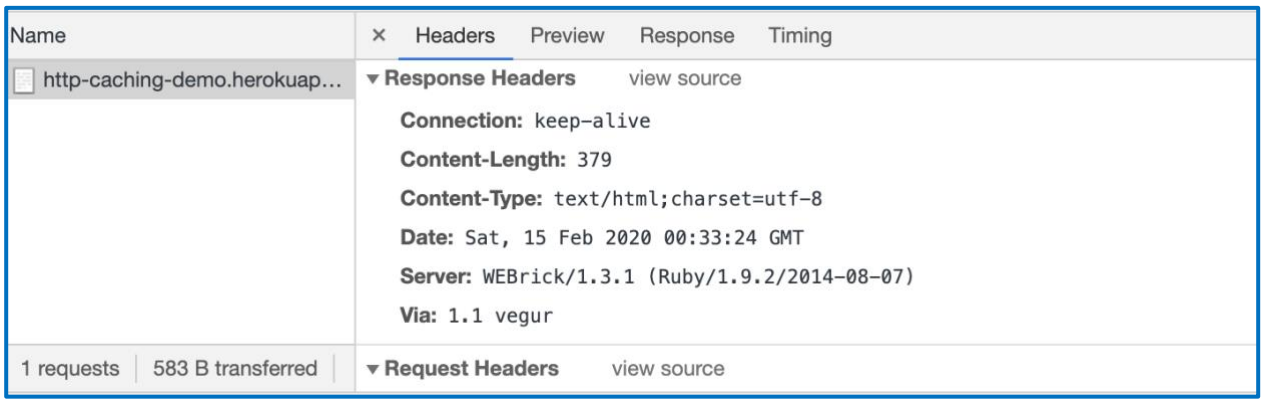


Note: You should have one response with a status code of 200. You may also have another response with 404 for a favicon.ico and that is fine, you can ignore it.

4. Click the response object under the **Name** column for http-caching-demo.herokuapp.com. This opens the response details window where you can examine the request and response headers.



5. Scroll down to see the **Response Headers** section. Note: there is no **`Cache-Control`** header in the response headers.



6. Now repeat steps 1-5. You get the same result. There is no caching involved in your request.

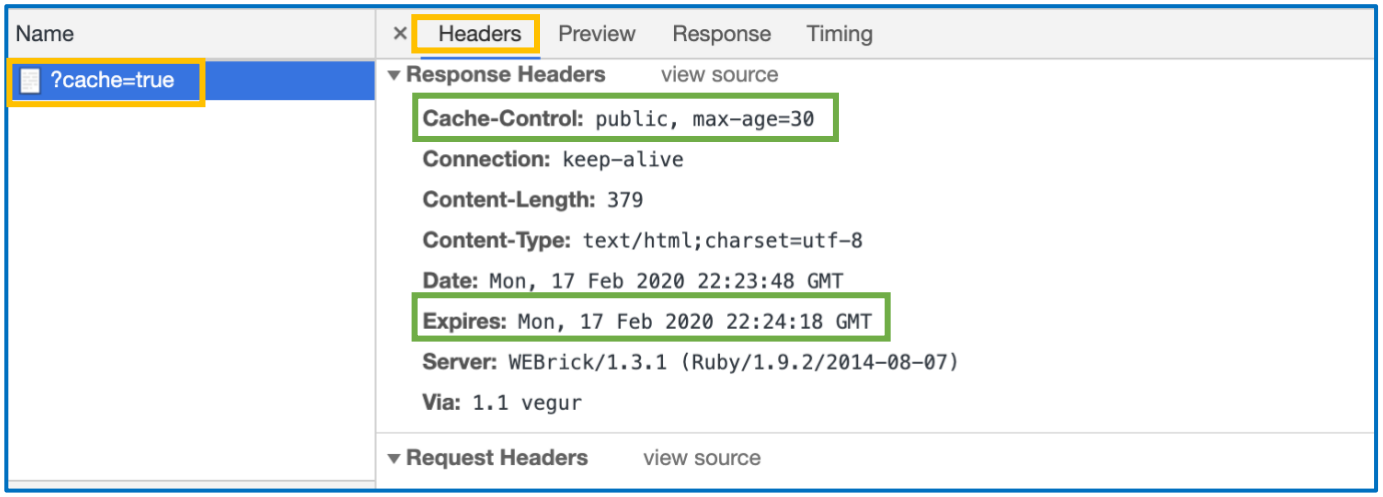
7. In your browser:

   a. Enter the following request:

      `http://http-caching-demo.herokuapp.com/?cache=true`

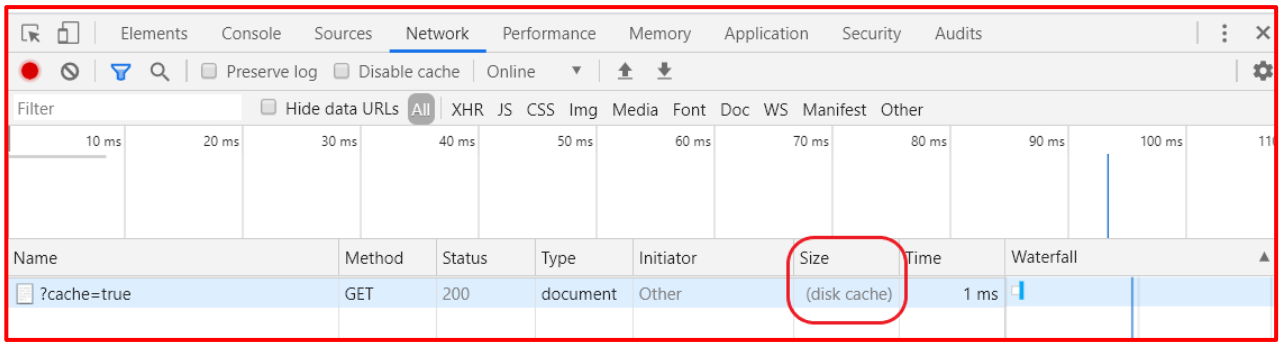
   b. Then ***click ?cache=true*** to view the headers.

   c. Make sure ***Headers*** is selected.



Note: You should see the **`Cache-Control`** header in the response headers with a max-age directive. There should also be an **`Expires`** header. Both of these can be used to control caching, but **`Cache-Control`** is more often used for HTTP/1.1. If both headers are present, the **`Cache-Control`** header takes precedence.

8. REMOVE!!!! Open a new browser tab, open the developer tools, and hit the same URL again (`http://http-caching-demo.herokuapp.com/?cache=true`). Look at the **Size** column in the developer tools. What do you notice?



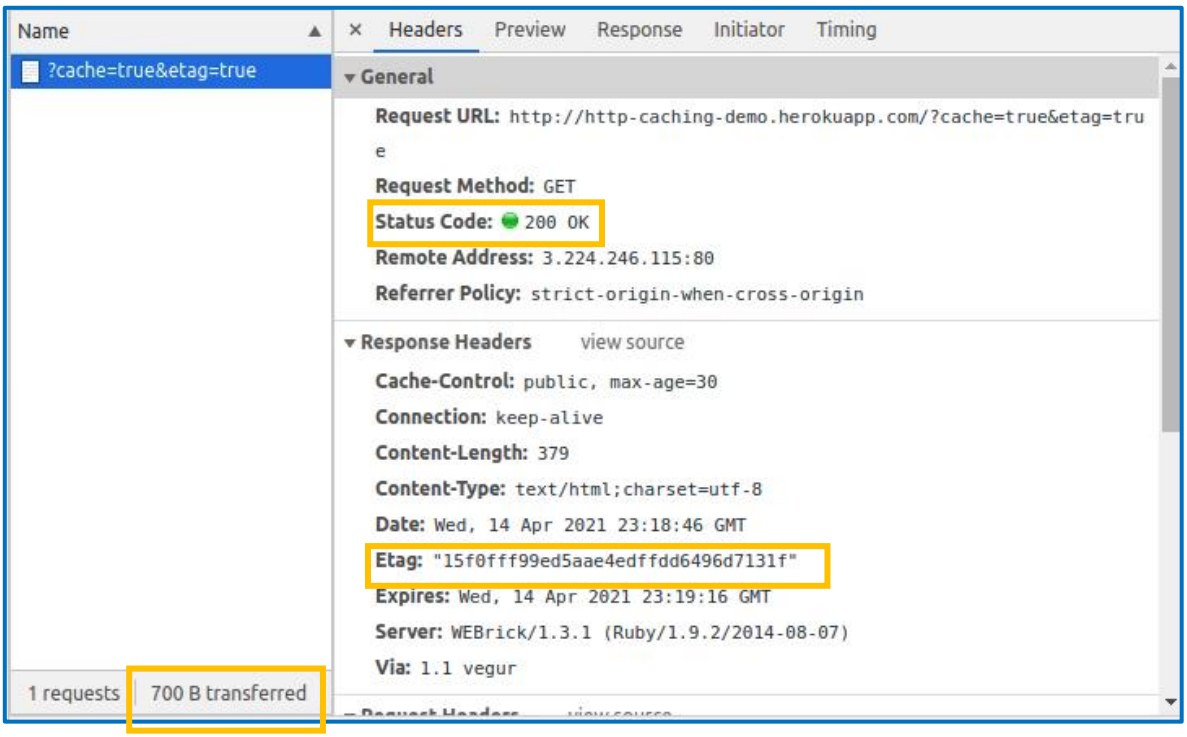
The size is zero. The browser is serving content from its local cached copy, so no data is transmitted from the server.

9. REMOVE!!! Wait 30 seconds for the **`max-age`** parameter to timeout. Send a request to the same URL.

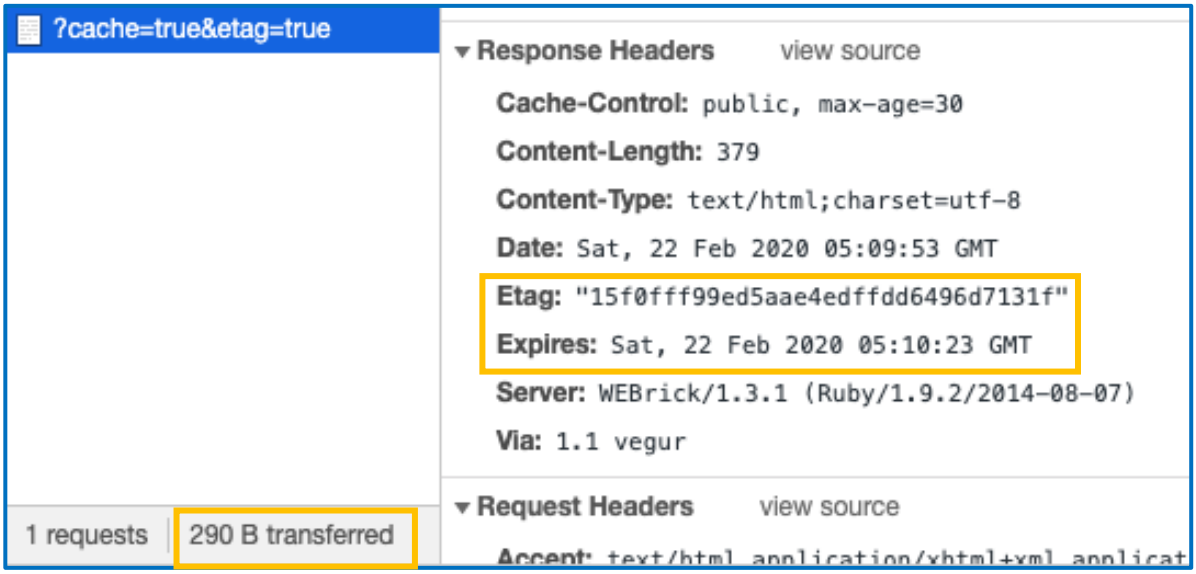
This time you see a server response because the Cache-Control header directs the browser to cache the response for a maximum of 30 seconds. After this, the browser requests a fresh copy of the data.

10. Change the request in the browser to include both cache and etag parameters. Look at the response and note the presence of the Etag header and its value. Also take note of the response size.

`http://http-caching-demo.herokuapp.com/?cache=true&etag=true`



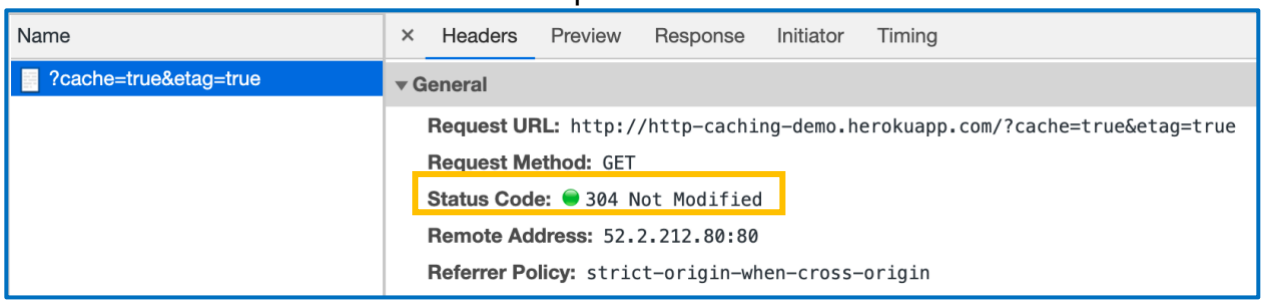
11. Open a new tab (along with the Developer tools) and send the same request. You should observe that the browser serves a cached copy of the response.

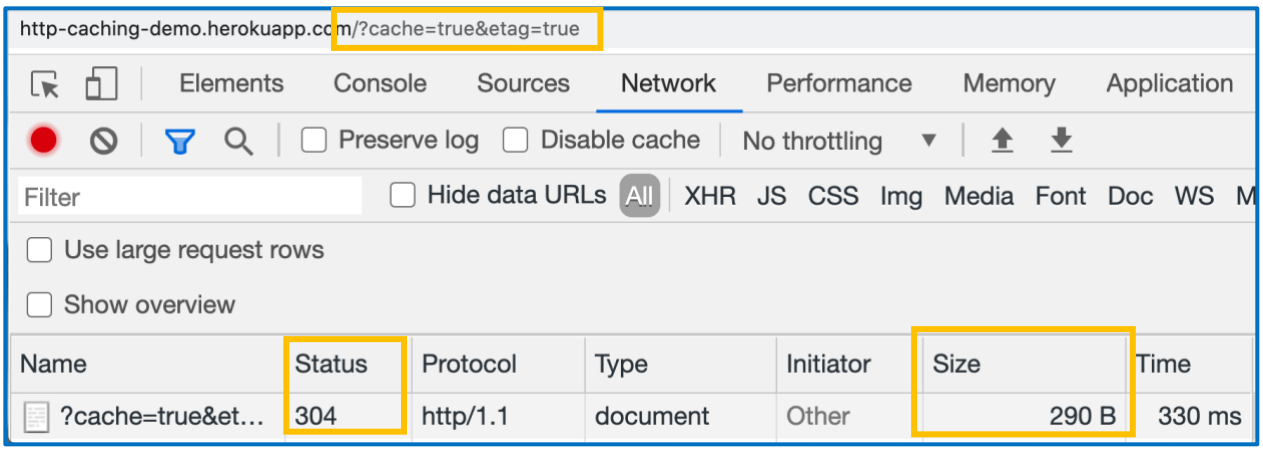


Note that the response size is also much smaller.

12. Wait 30 seconds and repeat step 11.

You should have a **304 Not Modified** response code.

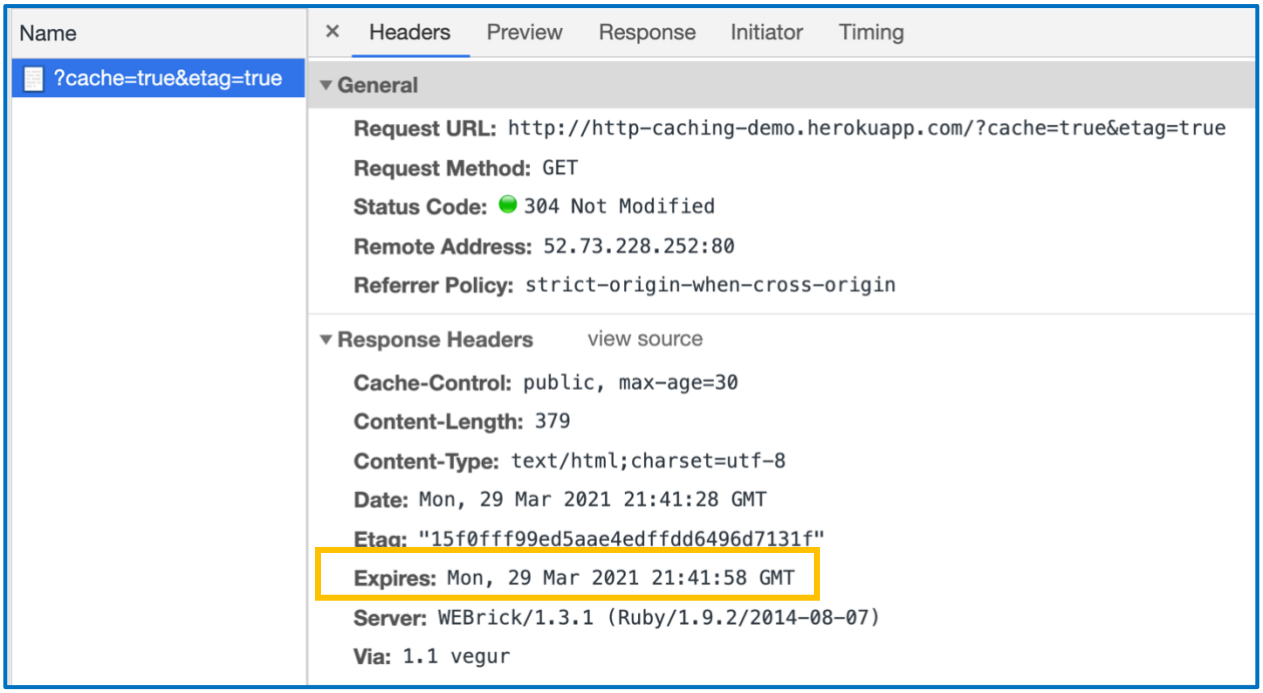


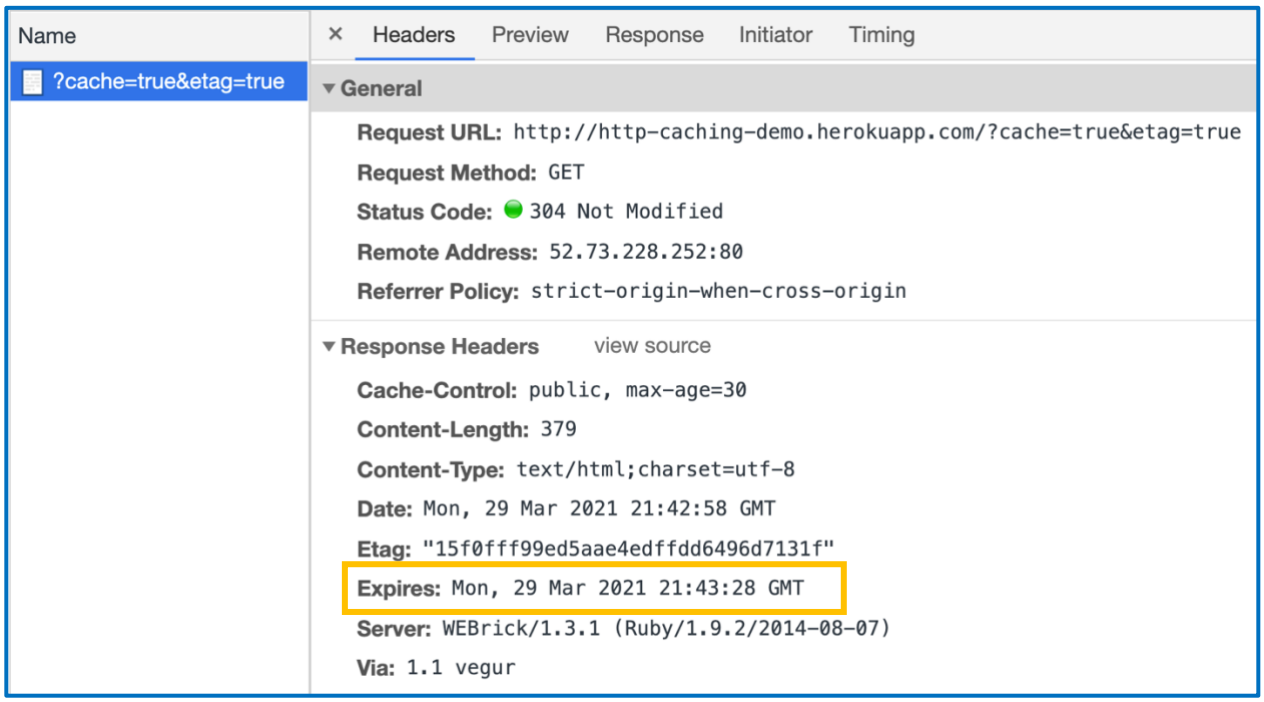


Previously the browser requested a new copy of the resource once the cache entry expired after 30 seconds. This is not an efficient caching strategy because if the resource has not been changed, we are wasting network bandwidth by having the server send a fresh copy.

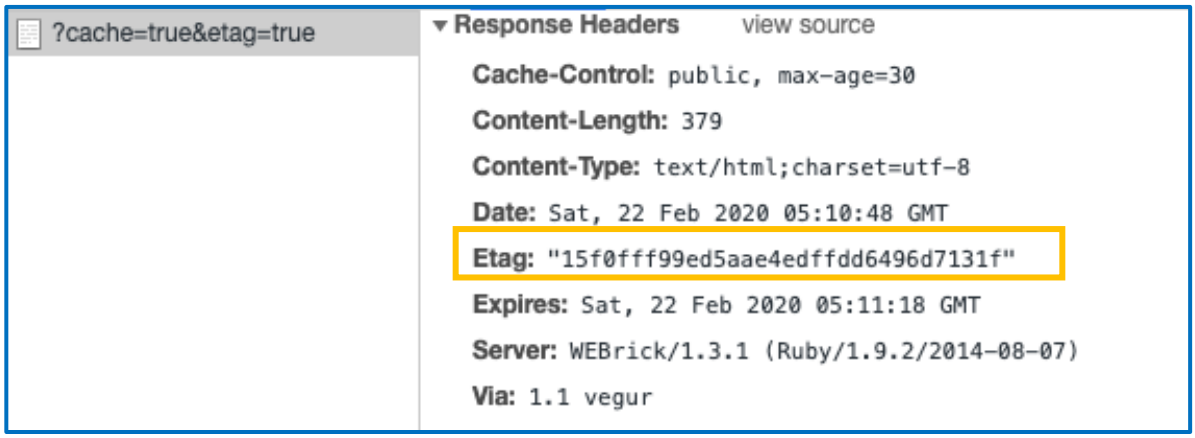
Instead, we should check to see if the resource has been modified. In this case the file has not changed compared to our cached copy, so the server responds with a HTTP 304 Not Modified. This tells our browser that although the cache time has expired, we can continue using the cached copy since the server copy is still the same. It will also reset the expiry time on the cache entry. The reduced response size is due to the fact that we are only receiving the response headers. There is no response body because we can use the cached version instead.

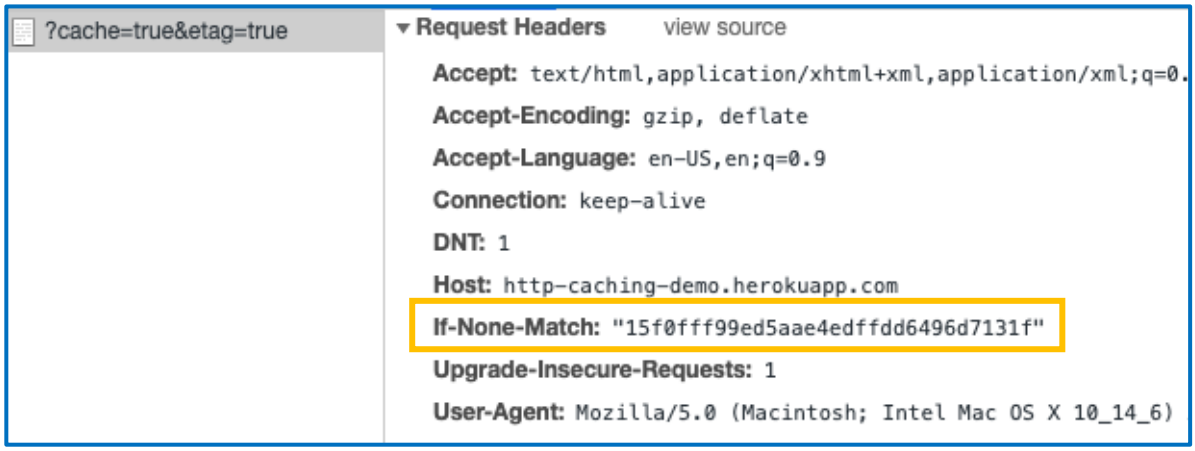
In this example the previous expiration value was 21:41:58 versus the new expiration of 21:43:28.




Name
?cache=true&etag=true
Headers
Preview
Response
Initiator
Timing
General
Request URL: http://http-caching-demo.herokuapp.com/?cache=true&etag=true
Request Method: GET
Status Code: 304 Not Modified
Remote Address: 52.73.228.252:80
Referrer Policy: strict-origin-when-cross-origin
Response Headers
view source
Cache-Control: public, max-age=30
Content-Length: 379
Content-Type: text/html;charset=utf-8
Date: Mon, 29 Mar 2021 21:42:58 GMT
Etag: "15f0fff99ed5aae4edffdd6496d7131f"
Expires: Mon, 29 Mar 2021 21:43:28 GMT
Server: WEBrick/1.3.1 (Ruby/1.9.2/2014-08-07)
Via: 1.1 vegur

To validate the file, the server looks for the **`If-None-Match`** request header and compares it the **`Etag`** value of the resource. Let's examine our response and request headers here.





Note: if the value of **`If-None-Match`** on the request is the same as the **`Etag`** value on the response, the server knows that the file has not changed. Since the **`Etag`** is a hash digest value of the file contents, any change in the content will cause the **`Etag`** value to differ.