**Serve static assets with an efficient cache policy**

May 2, 2019 • Updated Oct 4, 2019

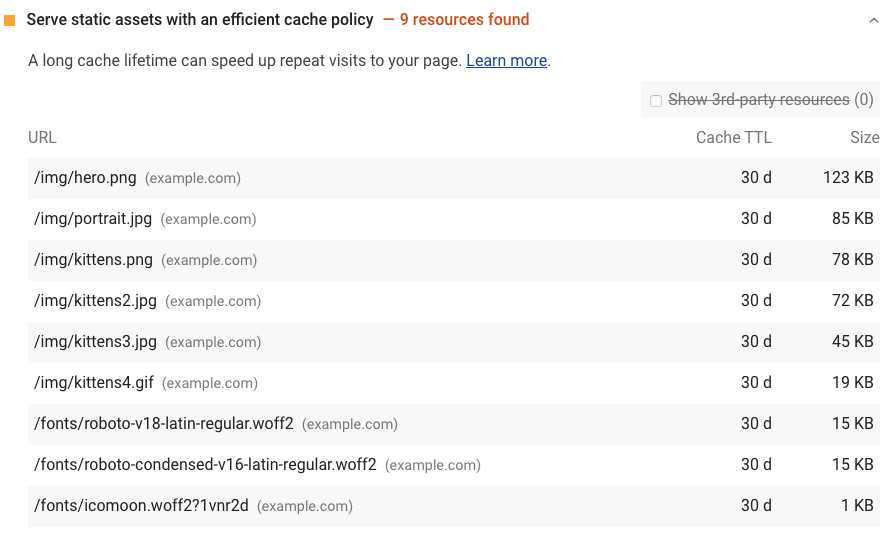
Appears in: [Performance audits](https://web.dev/lighthouse-performance)

HTTP caching can speed up your page load time on repeat visits.

When a browser requests a resource, the server providing the resource can tell the browser how long it should temporarily store or *cache* the resource. For any subsequent request for that resource, the browser uses its local copy rather than getting it from the network.

**How the Lighthouse cache policy audit fails** [**#**](https://web.dev/uses-long-cache-ttl/?utm_source=lighthouse&utm_medium=devtools#how-the-lighthouse-cache-policy-audit-fails)

[Lighthouse](https://developers.google.com/web/tools/lighthouse/) flags all static resources that aren't cached:



Lighthouse considers a resource cacheable if all the following conditions are met:

* The resource is a font, image, media file, script, or stylesheet.
* The resource has a 200, 203, or 206 [HTTP status code](https://developer.mozilla.org/docs/Web/HTTP/Status).
* The resource doesn't have an explicit no-cache policy.

When a page fails the audit, Lighthouse lists the results in a table with three columns:

|  |  |
| --- | --- |
| **URL** | The location of the cacheable resource |
| **Cache TTL** | The current cache duration of the resource |
| **Size** | An estimate of the data your users would save if the flagged resource had been cached |

See the [Lighthouse performance scoring](https://web.dev/performance-scoring) post to learn how your page's overall performance score is calculated.

**How to cache static resources using HTTP caching** [**#**](https://web.dev/uses-long-cache-ttl/?utm_source=lighthouse&utm_medium=devtools#how-to-cache-static-resources-using-http-caching)

Configure your server to return the Cache-Control HTTP response header:

Cache-Control: max-age=31536000

The max-age directive tells the browser how long it should cache the resource in seconds. This example sets the duration to 31536000, which corresponds to 1 year: 60 seconds × 60 minutes × 24 hours × 365 days = 31536000 seconds.

When possible, cache immutable static assets for a long time, such as a year or longer.

One risk of long cache durations is that your users won't see updates to static files. You can avoid this issue by configuring your build tool to embed a hash in your static asset filenames so that each version is unique, prompting the browser to fetch the new version from the server. (To learn how to embed hashes using webpack, see webpack's [Caching](https://webpack.js.org/guides/caching/) guide.)

Use no-cache if the resource changes and freshness matters, but you still want to get some of the speed benefits of caching. The browser still caches a resource that's set to no-cache but checks with the server first to make sure that the resource is still current.

A longer cache duration isn't always better. Ultimately, it's up to you to decide what the optimal cache duration is for your resources.

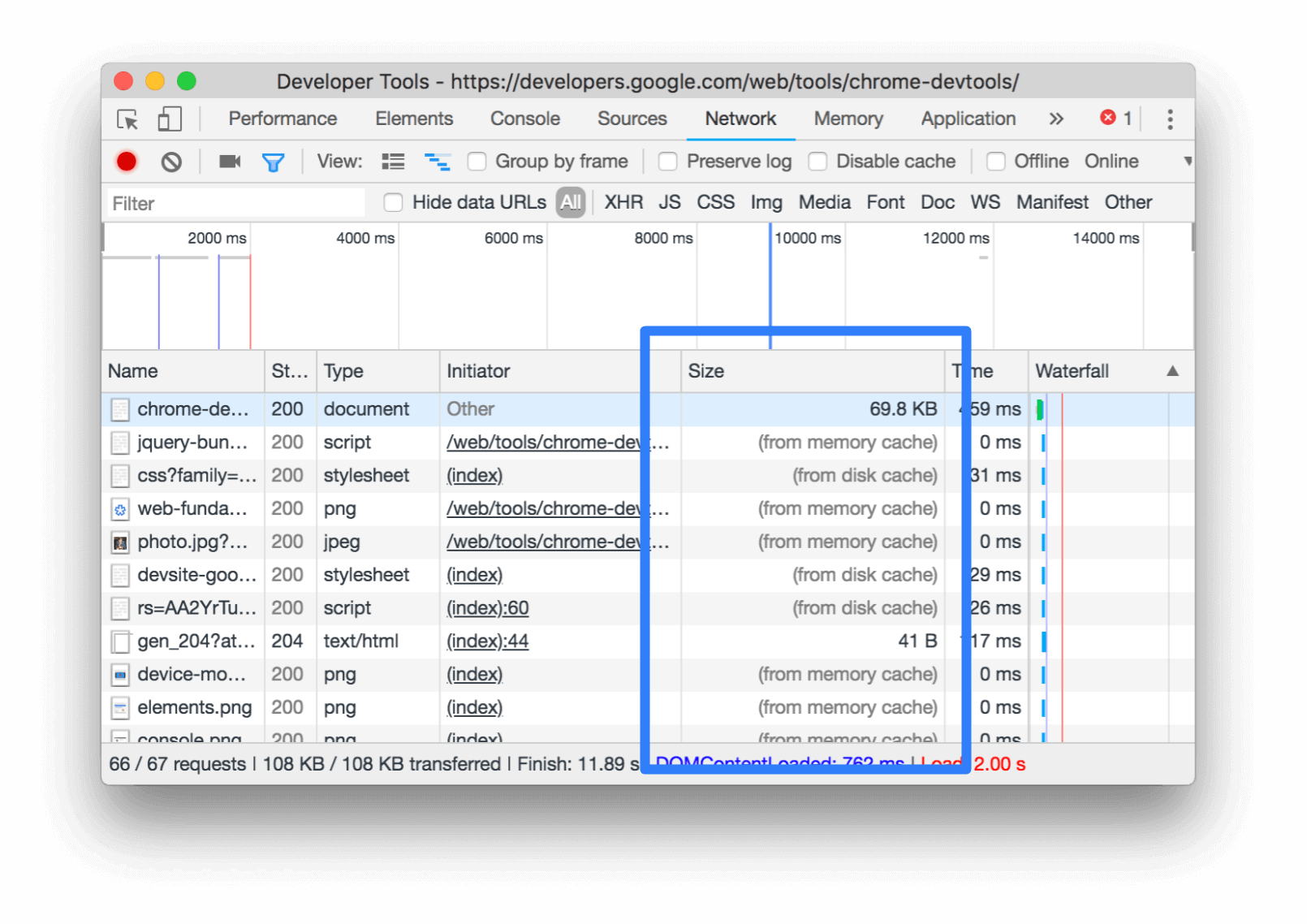
There are many directives for customizing how the browser caches different resources. Learn more about caching resources in [The HTTP cache: your first line of defense guide](https://web.dev/http-cache) and [Configuring HTTP caching behavior codelab](https://web.dev/codelab-http-cache).

**How to verify cached responses in Chrome DevTools** [**#**](https://web.dev/uses-long-cache-ttl/?utm_source=lighthouse&utm_medium=devtools#how-to-verify-cached-responses-in-chrome-devtools)

To see which resources the browser is getting from its cache, open the **Network** tab in Chrome DevTools:

1. Press `Control+Shift+J` (or `Command+Option+J` on Mac) to open DevTools.
2. Click the **Network** tab.

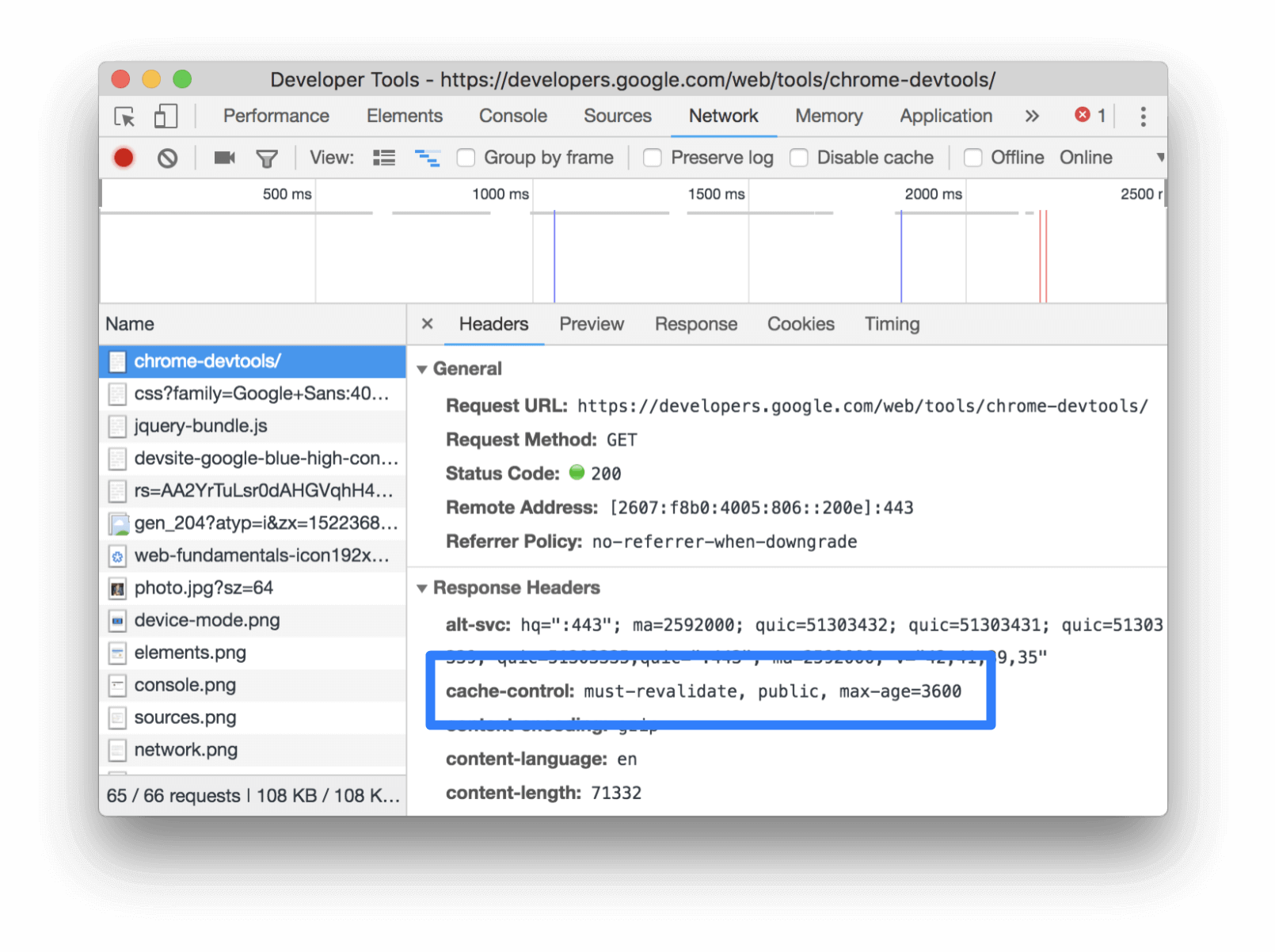
The **Size** column in Chrome DevTools can help you verify that a resource has been cached:



Chrome serves the most requested resources from the memory cache, which is very fast, but is cleared when the browser is closed.

To verify a resource's Cache-Control header is set as expected, check its HTTP header data:

1. Click the URL of the request, under the **Name** column of the Requests table.
2. Click the **Headers** tab.

Inspecting the Cache-Control header via the **Headers** tab.

**Stack-specific guidance** [**#**](https://web.dev/uses-long-cache-ttl/?utm_source=lighthouse&utm_medium=devtools#stack-specific-guidance)

**Drupal** [**#**](https://web.dev/uses-long-cache-ttl/?utm_source=lighthouse&utm_medium=devtools#drupal)

Set the **Browser and proxy cache maximum age** in the **Administration** > **Configuration** > **Development** page. See [Drupal performance resources](https://www.drupal.org/docs/7/managing-site-performance-and-scalability/caching-to-improve-performance/caching-overview#s-drupal-performance-resources).

**Joomla** [**#**](https://web.dev/uses-long-cache-ttl/?utm_source=lighthouse&utm_medium=devtools#joomla)

See [Cache](https://docs.joomla.org/Cache).

**WordPress** [**#**](https://web.dev/uses-long-cache-ttl/?utm_source=lighthouse&utm_medium=devtools#wordpress)

See [Browser Caching](https://wordpress.org/support/article/optimization/#browser-caching).

**Browser caching** can help to reduce server load by reducing the number of requests per page. For example, by setting the correct file headers on files that don’t change (static files like images, CSS, JavaScript etc) browsers will then cache these files on the user’s computer. This technique allows the browser to check to see if files have changed, instead of simply requesting them. The result is your web server can answer many more 304 responses, confirming that a file is unchanged, instead of 200 responses, which require the file to be sent.

Look into HTTP Cache-Control (specifically max-age) and Expires headers, as well as [Entity Tags](http://en.wikipedia.org/wiki/HTTP_ETag) for more information.

W3 Total Cache integrates support for browser caching and ETags.

https://wordpress.org/plugins/w3-total-cache/