Using Lighthouse To Improve Page Load Performance

By Jeremy Wagner

Jeremy is a contributor to Web**Fundamentals**

Lighthouse is an automated tool for improving the quality of your site. You give it a URL, and it provides a list of recommendations on how to improve page performance, make pages more accessible, adhere to best practices and more. You can run it from within Chrome DevTools, as a Chrome Extension, or even as a Node module, which is useful for continuous integration.

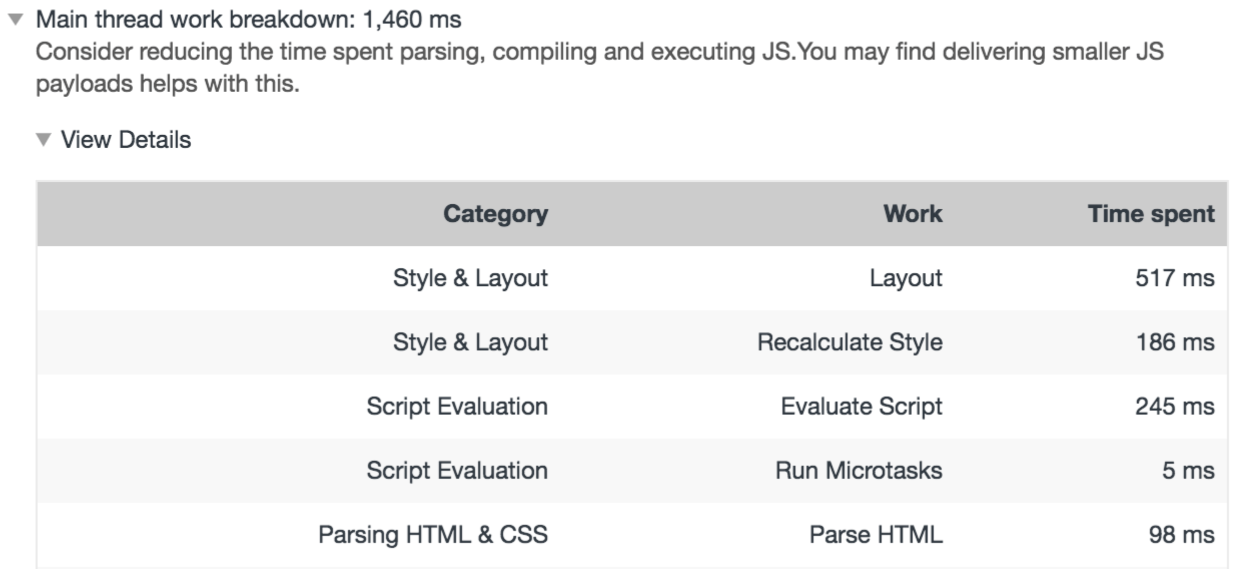
For a while now, Lighthouse has provided many tips for improving page load performance, such as enabling text compression or reducing render-blocking scripts. The Lighthouse team continues to ship new audits to give you even more useful advice for making your sites faster. This post is a roundup of useful performance audits that you may not be aware of, such as:

- Main Thread Work Breakdown
- Preload Key Requests
- JavaScript Boot-up Time is High

- Avoids Page Redirects
- Unused JavaScript
- Uses Inefficient Cache Policy on Static Assets
- Avoid Costly Multiple Round-Trips to Any Origin
- Use Video Formats for Animated Content
- All text remains visible during webfont loads
- Unminified CSS & JavaScript
- Unused CSS Rules

Main Thread Work Breakdown

If you've ever used the performance panel in DevTools, you know it can be a bit of a chore to get a breakdown of where CPU time was spent loading a page. We're pleased to announce that this information is now readily and conveniently available via the new **Main Thread Work Breakdown** audit.



▼ Main thread work breakdown: 1,460 ms
Consider reducing the time spent parsing, compiling and executing JS.You may find delivering smaller JS payloads helps with this.

▼ View Details

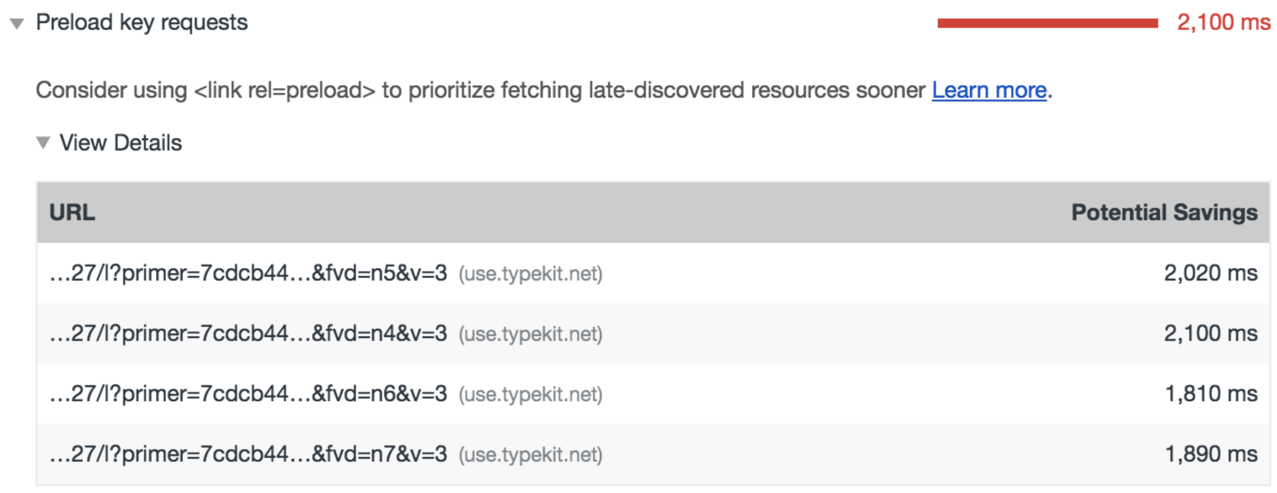
Category	Work	Time spent
Style & Layout	Layout	517 ms
Style & Layout	Recalculate Style	186 ms
Script Evaluation	Evaluate Script	245 ms
Script Evaluation	Run Microtasks	5 ms
Parsing HTML & CSS	Parse HTML	98 ms

Figure 1. A breakdown of main thread activity in Lighthouse.

This new diagnostic evaluates how much and what kind of activity occurs during page load, which you can use to get a handle on loading performance issues related to layout, script eval, parsing, and other activity.

Preload Key Requests

When browsers retrieve resources, they do so as they find references to them within the document and its subresources. This can be suboptimal at times, because some critical resources are discovered rather late in the page load process. Thankfully, `rel=preload` gives developers the ability to hint to compliant browsers which resources should be fetched as soon as possible. The new **Preload Key Requests** audit lets developers know what resources could benefit from being loaded sooner by `rel=preload`.



▼ Preload key requests 2,100 ms

Consider using <link rel=preload> to prioritize fetching late-discovered resources sooner Learn more.

▼ View Details

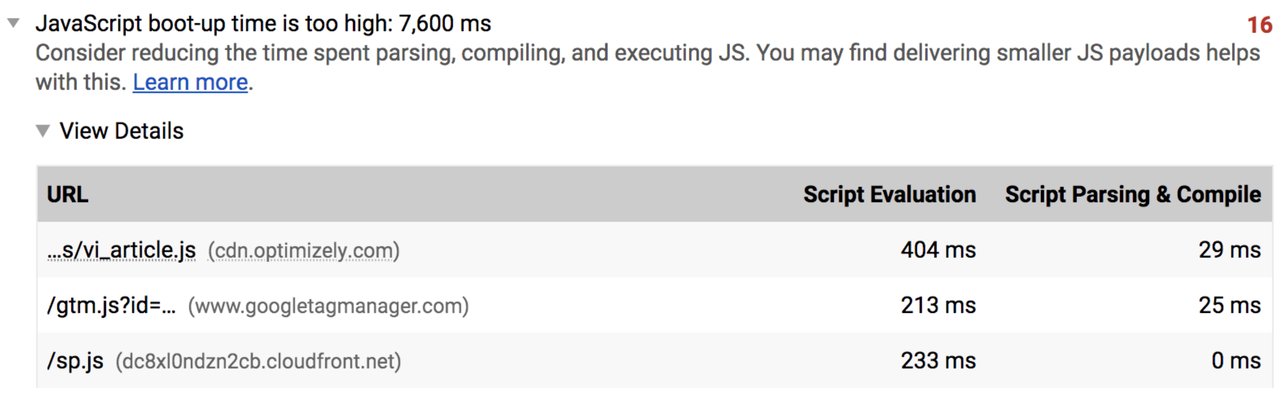
URL	Potential Savings
…27/l?primer=7cdcb44…&fvd=n5&v=3 (use.typekit.net)	2,020 ms
…27/l?primer=7cdcb44…&fvd=n4&v=3 (use.typekit.net)	2,100 ms
…27/l?primer=7cdcb44…&fvd=n6&v=3 (use.typekit.net)	1,810 ms
…27/l?primer=7cdcb44…&fvd=n7&v=3 (use.typekit.net)	1,890 ms

Figure 2. The Preload Key Requests Lighthouse audit recommending a list of resources to consider preloading.

It's super important you test and compare performance changes with and without `rel=preload`, as it can affect loading performance in ways you might not expect. For example, preloading a large image could delay initial render, but the tradeoff is that the preloaded image will appear sooner in the layout. Always make sure you're cool with the results!

JavaScript Boot-up Time is High

When too much JavaScript is loaded, the page can become unresponsive as the browser parses, compiles, and executes it. 3rd-party scripts and advertisements are a particular source of excessive script activity that can bog down even powerful devices. The new **JavaScript Boot-up Time is High** audit reveals how much CPU time each script on a page consumes, along with its URL:



JavaScript boot-up time is too high: 7,600 ms 16
Consider reducing the time spent parsing, compiling, and executing JS. You may find delivering smaller JS payloads helps with this. Learn more.

View Details

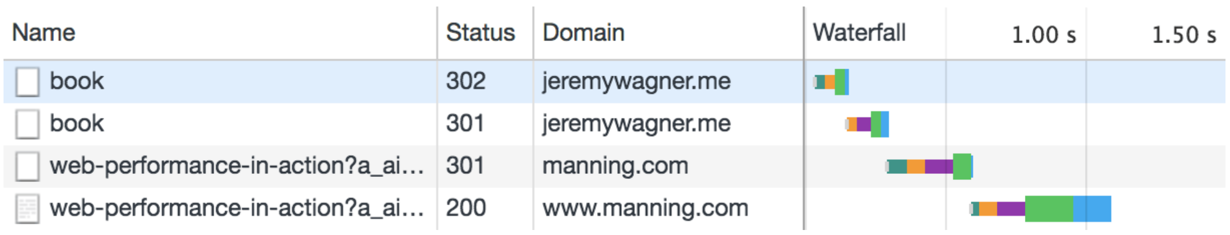
URL	Script Evaluation	Script Parsing & Compile
…s/vi_article.js (cdn.optimizely.com)	404 ms	29 ms
/gtm.js?id=… (www.googletagmanager.com)	213 ms	25 ms
/sp.js (dc8xl0ndzn2cb.cloudfront.net)	233 ms	0 ms

Figure 3. Lighthouse displaying the amount of evaluation, parsing, and compiling time for scripts on a page.

When you run this audit, you can also enable third party badges in the network panel and filter the list to identify third party script resources. With the data from this audit, you'll be better equipped to find sources of excessive JavaScript activity that turn pages from snappy to laggy. For scripts specific to your application, you can employ techniques like code splitting and tree shaking to limit the amount of JavaScript on each page of your site.

Avoids Page Redirects

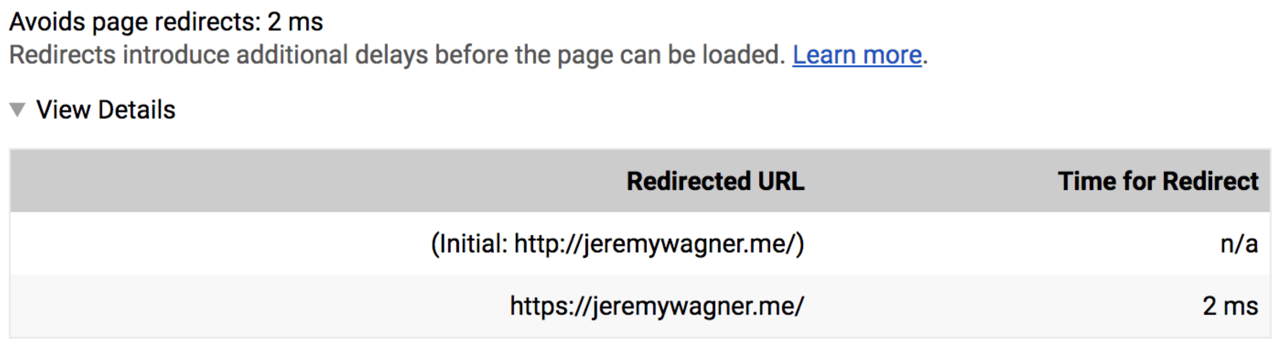
Sometimes when a browser requests a URL, the server can respond with a 300-level status code. This causes the browser to redirect to another URL. While redirects are necessary for SEO and convenience purposes, they add latency to requests. This is especially true if they redirect to other origins, which can incur additional DNS lookup and connection/TLS negotiation time.



Name	Status	Domain	Waterfall
book	302	jeremywagner.me	
book	301	jeremywagner.me	
web-performance-in-action?a_ai…	301	manning.com	
web-performance-in-action?a_ai…	200	www.manning.com	

Figure 4. A redirect chain as seen in the network panel of Chrome's developer tools.

Redirects are undesirable for landing pages on your site. To help you reduce latency and improve loading performance, Lighthouse now offers the **Avoids Page Redirects** audit, which lets you know when a navigation triggers any redirects.


Avoids page redirects: 2 ms
Redirects introduce additional delays before the page can be loaded. Learn more.

▼ View Details

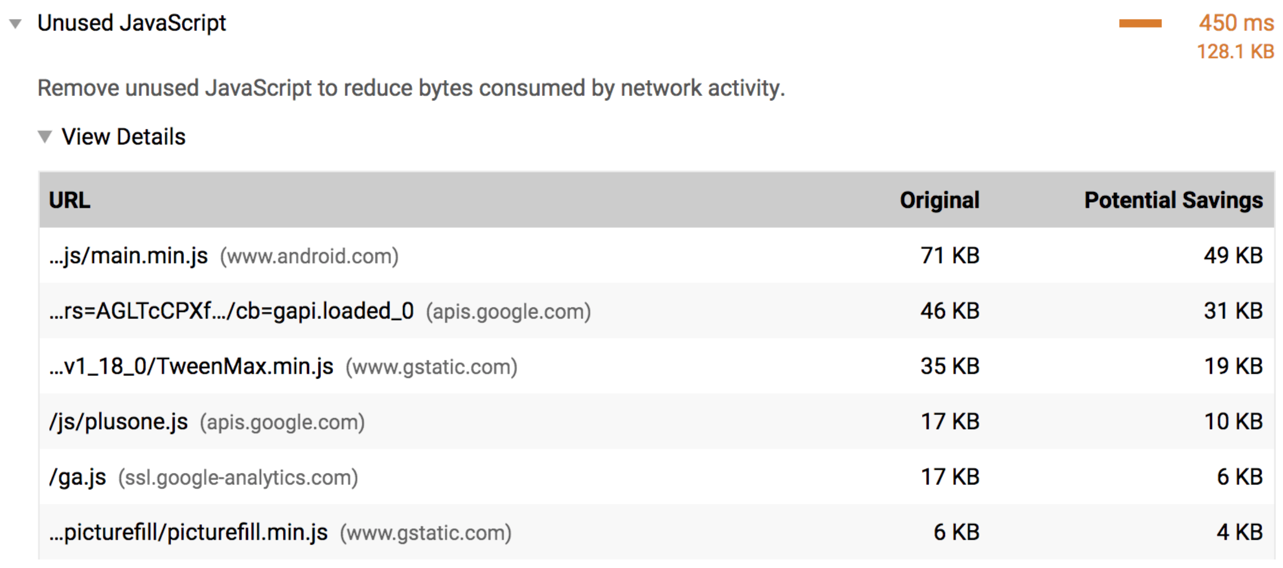
Redirected URL	Time for Redirect
(Initial: http://jeremywagner.me/)	n/a
https://jeremywagner.me/	2 ms

Figure 5. A list of page redirects in Lighthouse.

Note that this audit is difficult to trigger in the DevTools version of Lighthouse, because it analyzes the current URL in the address bar of the page, which reflects the resolution of all redirects. You're likeliest to see this audit populated in the Node CLI.

Unused JavaScript

Dead code can be a serious problem in JavaScript-heavy applications. While it doesn't pose execution costs as it's never invoked, it does carry other undesirable effects. Dead code is still downloaded, parsed, and compiled by the browser. This affects loading performance and JavaScript boot-up time. Similar to the coverage panel in DevTools, the **Unused JavaScript** audit reveals JavaScript downloaded by the current page, but is never used.


▼ Unused JavaScript 450 ms 128.1 KB

Remove unused JavaScript to reduce bytes consumed by network activity.

▼ View Details

URL	Original	Potential Savings
…js/main.min.js (www.android.com)	71 KB	49 KB
…rs=AGLTcCPXf…/cb=gapi.loaded_0 (apis.google.com)	46 KB	31 KB
…v1_18_0/TweenMax.min.js (www.gstatic.com)	35 KB	19 KB
/js/plusone.js (apis.google.com)	17 KB	10 KB
/ga.js (ssl.google-analytics.com)	17 KB	6 KB
…picturefill/picturefill.min.js (www.gstatic.com)	6 KB	4 KB

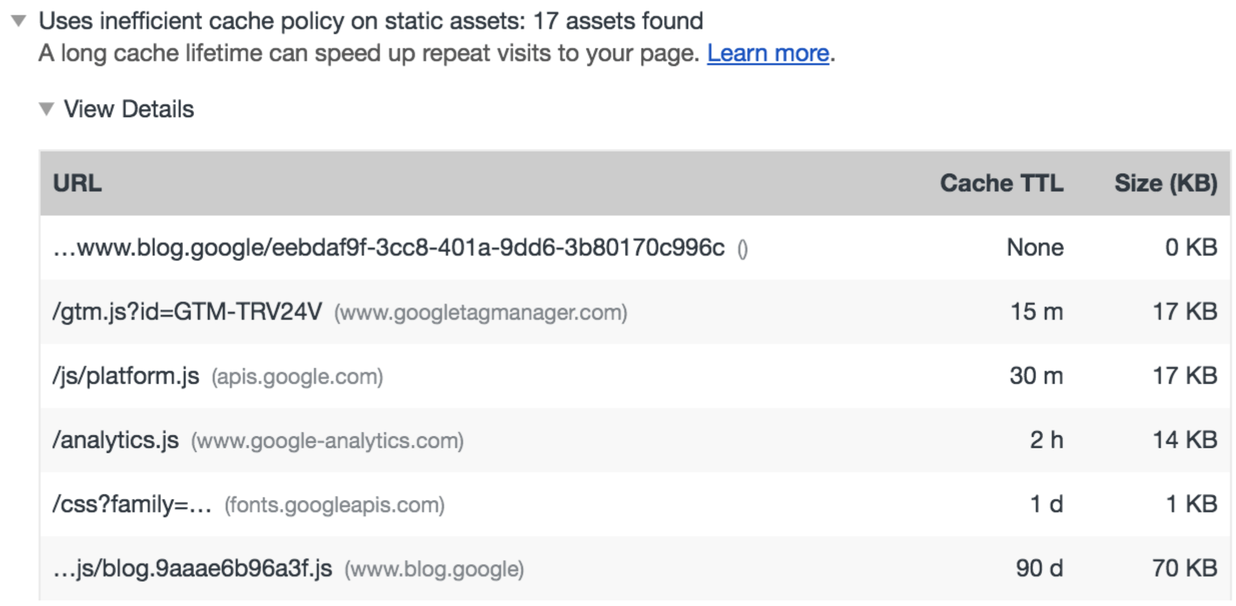
Figure 6. Lighthouse displaying the amount of unused JavaScript on a page.

With this audit, you can identify dead code in your applications and remove it to improve loading performance and reduce system resource usage. Pro tip: You can also use the code coverage panel in Chrome's DevTools to find this information!

Note: This audit is off by default! It can be enabled in the Node CLI by using the `lighthouse:full` configuration profile.

Uses Inefficient Cache Policy on Static Assets

While much performance advice tends to focus on boosting the speed of a website for first time users, it's also important to use caching to improve loading performance for returning users. **The Uses Inefficient Cache Policy on Static Assets** audit inspects caching headers for network resources, and notifies you if cache policies for static resources are substandard.



▼ Uses inefficient cache policy on static assets: 17 assets found
A long cache lifetime can speed up repeat visits to your page. Learn more.

▼ View Details

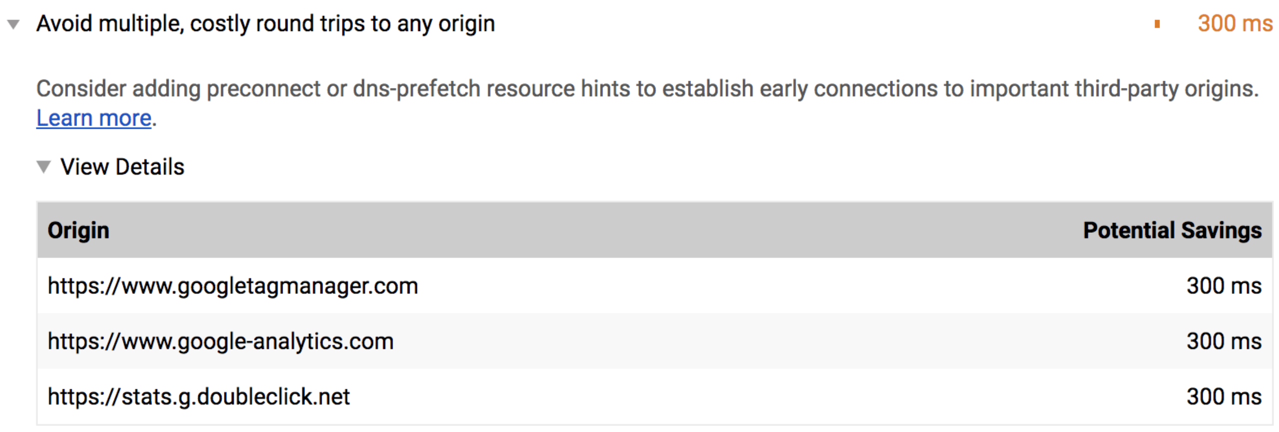
URL	Cache TTL	Size (KB)
…www.blog.google/eebdaf9f-3cc8-401a-9dd6-3b80170c996c ()	None	0 KB
/gtm.js?id=GTM-TRV24V (www.googletagmanager.com)	15 m	17 KB
/js/platform.js (apis.google.com)	30 m	17 KB
/analytics.js (www.google-analytics.com)	2 h	14 KB
/css?family=… (fonts.googleapis.com)	1 d	1 KB
…js/blog.9aaae6b96a3f.js (www.blog.google)	90 d	70 KB

Figure 7.

With the help of this audit, you'll be able to easily find and fix problems with your current cache policy. This will greatly improve performance for returning users, and they'll appreciate the extra speed!

Avoid Costly Multiple Round-Trips to Any Origin

When browsers retrieve resources from a server, it can take significant time to perform a DNS lookup and establish a connection to a server. `rel=preconnect` allows developers to mask this latency by establishing connections to other servers before the browser would in due course. The **Avoid Costly Multiple Round-Trips to Any Origin** audit will help you discover opportunities to use `rel=preconnect`!


Avoid multiple, costly round trips to any origin — 300 ms

Consider adding preconnect or dns-prefetch resource hints to establish early connections to important third-party origins. Learn more.

View Details

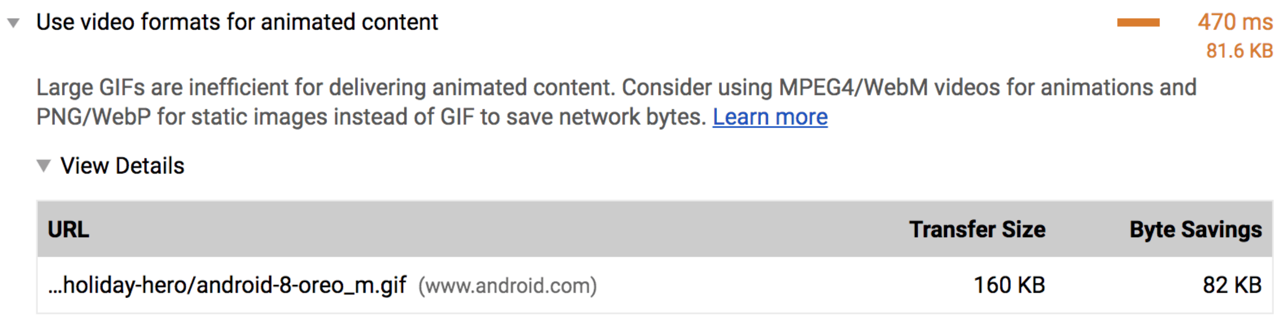
Origin	Potential Savings
https://www.googletagmanager.com	300 ms
https://www.google-analytics.com	300 ms
https://stats.g.doubleclick.net	300 ms

Figure 8. A list of origins recommended for `rel=preconnect` in Lighthouse.

When latency for cross-origin assets is reduced, users will perceive that things are moving along a bit quicker. With this new Lighthouse audit, you'll learn of new opportunities to use `rel=preconnect` to do just that.

Use Video Formats for Animated Content

Animated GIFs are *huge*, often consuming at least several hundred kilobytes if not several megabytes of data. If you care about loading performance, converting those GIFs to video is the way to go. Thankfully, the **Use Video Formats for Animated Content** audit has your back.


Use video formats for animated content — 470 ms 81.6 KB

Large GIFs are inefficient for delivering animated content. Consider using MPEG4/WebM videos for animations and PNG/WebP for static images instead of GIF to save network bytes. Learn more

View Details

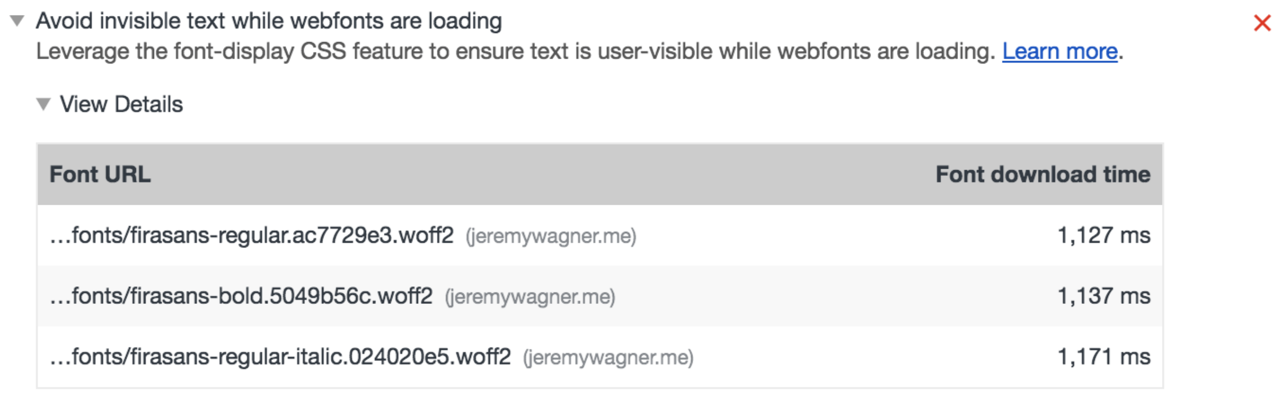
URL	Transfer Size	Byte Savings
…holiday-hero/android-8-oreo_m.gif (www.android.com)	160 KB	82 KB

Figure 9. A recommendation to convert a GIF to video in Lighthouse.

If your site has any GIFs that are over 100 KB, this audit will automatically flag them and direct you to some guidance on how to convert them to video and embed them. Sites like Imgur have significantly improved loading performance by converting their GIFs to video. Additionally, if your site is on a hosting plan with metered bandwidth, the potential cost savings alone should be enough to persuade you!

All Text Remains Visible During Web Font Loads

When we load web fonts for pages, browsers often render invisible text until the font loads. This phenomenon, known as the Flash of Invisible Text (FOIT)), may be preferable to you from a design standpoint, but it's actually a problem. Text that's blocked from rendering can't be read until it renders and becomes visible. On high latency and/or high bandwidth connections, this means a core part of your user experience is missing. It can also be a sort of perceptual performance issue in that the page is not rendering meaningful content as quickly as it otherwise could. Thankfully, the **All Text Remains Visible During Web Font Loads** audit helps you to find opportunities to fix this on your site!


▼ Avoid invisible text while webfonts are loading ✕
Leverage the font-display CSS feature to ensure text is user-visible while webfonts are loading. Learn more.

▼ View Details

Font URL	Font download time
…fonts/firasans-regular.ac7729e3.woff2 (jeremywagner.me)	1,127 ms
…fonts/firasans-bold.5049b56c.woff2 (jeremywagner.me)	1,137 ms
…fonts/firasans-regular-italic.024020e5.woff2 (jeremywagner.me)	1,171 ms

Figure 10. Lighthouse giving recommendations for improving font rendering.

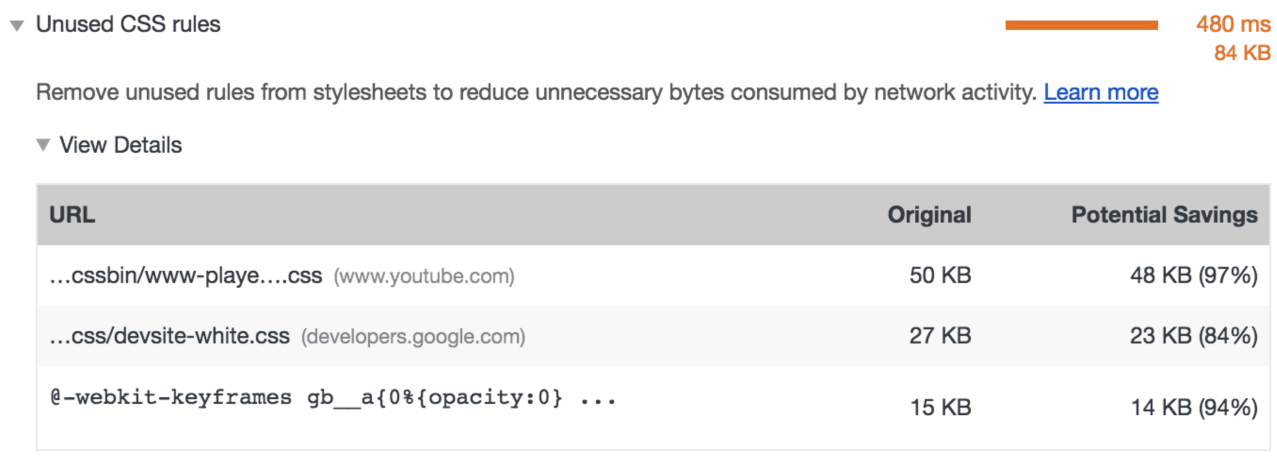
If Lighthouse finds web fonts in your application that are delaying text rendering, there's a few potential remedies. You can control text rendering with the `font-display` CSS property, and/or the Font Loading API. If you want to dig deeper, consider reading *A Comprehensive Guide to Font Loading Strategies*, an excellent guide by Zach Leatherman which is an excellent resource for optimal font loading.

Unminified CSS & JavaScript

Minification has been a suggested technique since web performance has been a thing, and for good reason. It significantly reduces the size of text-based resources, which in turn is good for loading performance. However, it's easy to overlook this optimization, especially if build processes don't take care of it for you. The **Minify CSS** and **Minify JavaScript** audits will compile a list of unminified resources it finds on the current page. From there, you can take action by minifying those files manually, or augmenting your build system to do it for you.

Unused CSS Rules

As a site gets a bit long in the tooth, it's inevitable that the cruft left over from refactoring begins to build up. One such source of cruft comes in the form of unused CSS rules which are no longer necessary for the site to function, yet still consume bandwidth. For your convenience, the **Unused CSS Rules** audit reveals which CSS resources on the page contain unused CSS.



▼ Unused CSS rules — 480 ms, 84 KB

Remove unused rules from stylesheets to reduce unnecessary bytes consumed by network activity. Learn more

▼ View Details

URL	Original	Potential Savings
…cssbin/www-playe….css (www.youtube.com)	50 KB	48 KB (97%)
…css/devsite-white.css (developers.google.com)	27 KB	23 KB (84%)
`@-webkit-keyframes gb__a{0%{opacity:0} ...`	15 KB	14 KB (94%)

Figure 11. Lighthouse showing a list of CSS resources containing unused CSS rules.

If Lighthouse finds unused CSS on the page, there are ways to get rid of it. UnCSS is one such utility that does this for you automatically (although it must be used with care). A more manual method involves using the code coverage panel in DevTools. Just remember, though, that unused CSS on one page could be necessary on another. Another approach may be to split your CSS into template-specific files that are only loaded where necessary. Whatever you decide to do, Lighthouse will be there to let you know if your CSS cruft is getting to be a bit much.

Give Lighthouse a try!

If you're excited about these new audits, update Lighthouse and give them a try!

- The Lighthouse Chrome extension should automatically update, but you can manually update it via `chrome://extensions`.
- In DevTools, you can run Lighthouse in the audits panel. Chrome updates to a new version about every 6 weeks, so some newer audits may not be available. If you're antsy to use the latest audits available, you can run the latest Chrome code by downloading Chrome Canary.
- For Node users: Run `npm update lighthouse`, or `npm update lighthouse -g` if you installed Lighthouse globally.

Special thanks to Kayce Basques, Patrick Hulce, Addy Osmani, and Vinamrata Singal for their valuable feedback, which significantly improved the quality of this article.

Last updated July 2, 2018.