

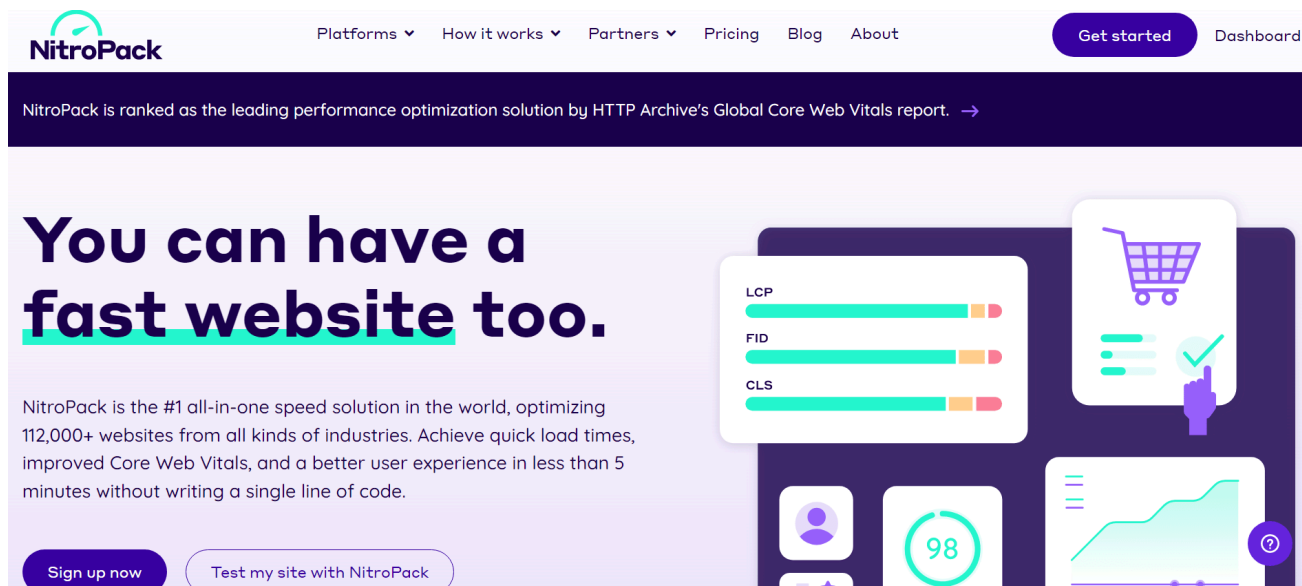
# Largest Contentful Paint and How to Improve It

**TL;DR:** Largest Contentful Paint (LCP) is a key metric measuring the time it takes for the biggest element on a webpage to load. To fix high LCP, optimize Time to First Byte (TTFB), eliminate resource load delays, reduce resource load times through image optimization, and use CDN to ensure immediate rendering of the LCP element.

## What is Largest Contentful Paint?

Largest Contentful Paint measures how long it takes for the largest above the fold element to load on a page.

Above the fold means everything that appears on your screen without scrolling. For instance, here's what I see on my laptop when I load NitroPack's home page:



The types of elements considered for Largest Contentful Paint are:

- Images;
- Image tags;
- Video thumbnails;
- Background images with CSS;
- Text elements.

Reducing your website's LCP helps users see the essential content on your website faster.

However, before you can fix potential LCP issues, you need to know how fast the largest element loads.

Here's how you can measure your LCP score.

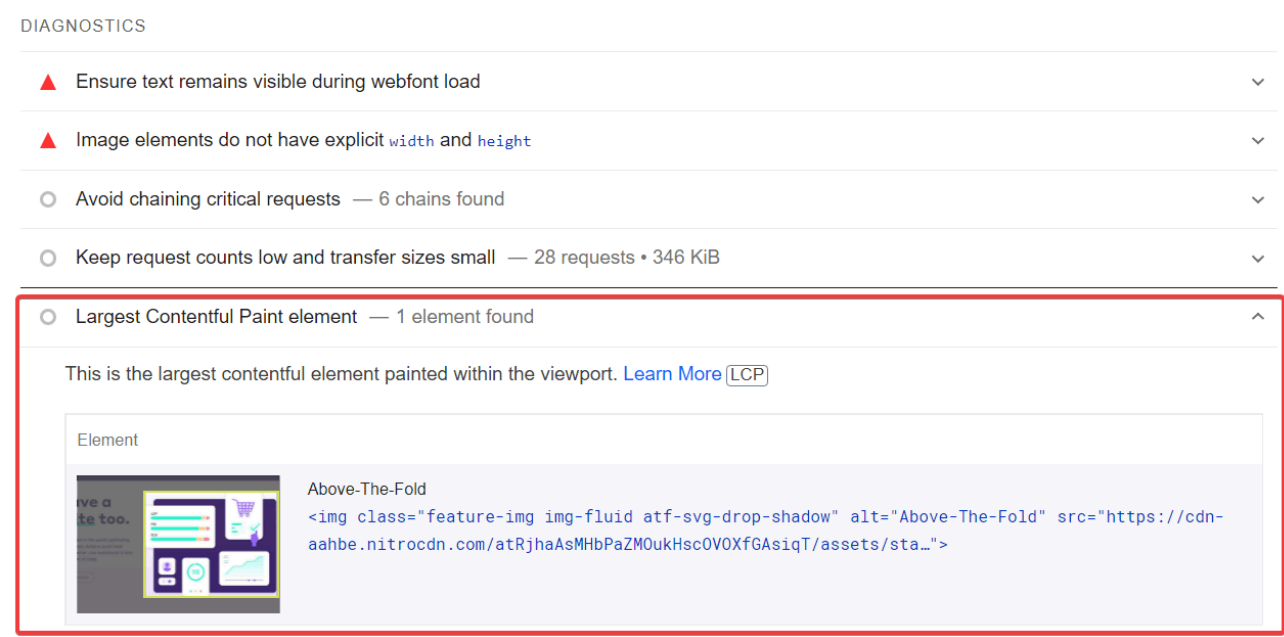
# How to measure LCP

There are a couple of tools that you can use to first diagnose the largest contentful paint element on your page and then measure its performance.

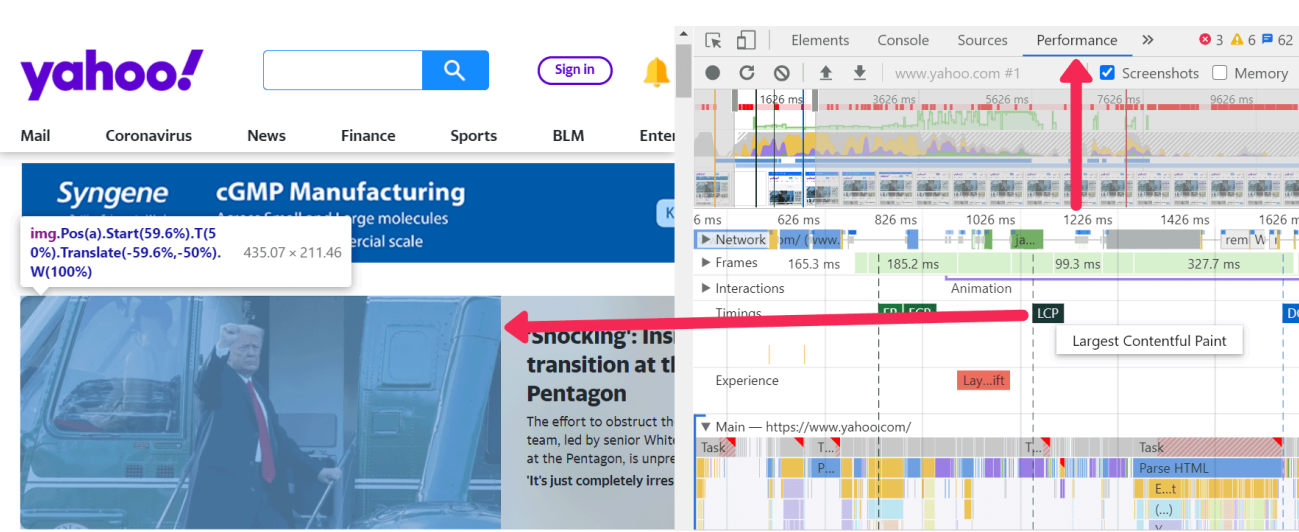
## PageSpeed Insights

You can run a PageSpeed Insights (PSI) test to see which is the largest element on your page.

The “Diagnostics” section shows you which element triggers the LCP metric.



The “Performance” panel in DevTools can also do the same job:



Just go to the page you want to test. Then, open the “Inspect” panel, find “Performance”, and refresh the page.

## WebPageTest

WebPageTest is another useful tool that allows you to test from over twenty locations while emulating a device with a specific connection.

Test Location

Frankfurt, Germany - EC2

Browser

Chrome

Start Test →

Test Settings

Advanced

Chromium

Script

Block

SPOF

Custom

Connection

Cable (5/1 Mbps 28ms RTT)

Desktop Browser Dimensions

default (1366x768)

Number of Tests to Run

Up to 9

3

Repeat View

First View and Repeat View

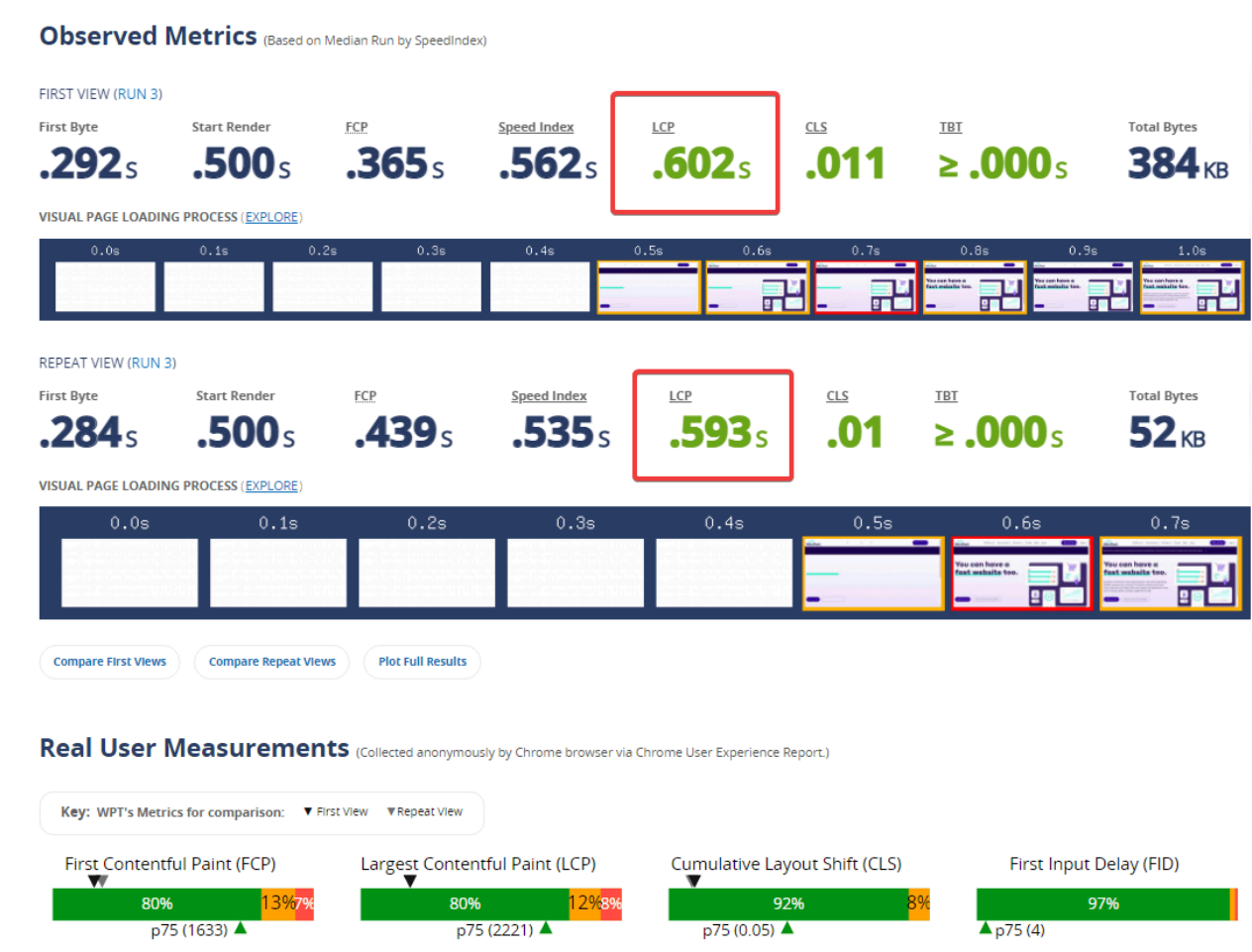
First View Only

Capture Video

All test results are configured to be private by default.

Label

After the test ends, you will see the Observed Metrics as well as Real User Measurements:



[Improve your LCP score automatically! Test your site with NitroPack →](#)

# GTMetrix

You can test your website with GTMetrix as well. After the test is complete, go to the Performance tab where you will see your LCP score:

Summary	Performance	Structure	Waterfall	Video	History
---------	-------------	-----------	-----------	-------	---------

**Performance Metrics**

The following metrics are generated using Lighthouse Performance data. Metric details ☐ OFF

First Contentful Paint ?	OK, but consider improvement <b>1.1s</b>	Time to Interactive ?	Good - Nothing to do here <b>1.1s</b>
Speed Index ?	Good - Nothing to do here <b>1.1s</b>	Total Blocking Time ?	Good - Nothing to do here <b>0ms</b>
Largest Contentful Paint ?	Good - Nothing to do here <b>1.1s</b>	Cumulative Layout Shift ?	Good - Nothing to do here <b>0.02</b>

# What is a good Largest Contentful Paint score?

According to [Google](#):

“To provide a good user experience, sites should strive to have Largest Contentful Paint of **2.5 seconds or less**. To ensure you’re hitting this target for most of your users, a good threshold to measure is the 75th percentile of page loads, segmented across mobile and desktop devices.”



So you need to aim for a result below 2.5 sec. But how to achieve it?

To improve a page’s LCP time, you have to optimize that element’s load time.

Here are five proven ways to do that.

## How to improve LCP

Five optimization categories help fix LCP problems on most websites:

- [Image optimization](#);
- [CSS and JavaScript optimization](#);
- [Faster server response time](#);
- [Limited and optimized client-side rendering](#);
- [Preloading and preconnecting](#).

All of these also help with other performance metrics like FCP, CLS, and TTl.

## Image Optimization

Image optimization is a collection of techniques that can improve all load metrics and [reduce layout shifts \(CLS\)](#).

### 1. Image compression

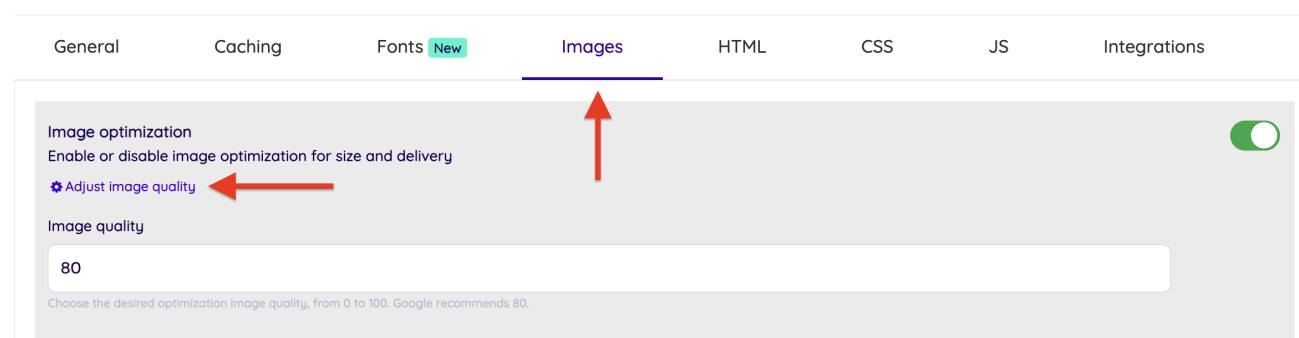
Compression means applying different algorithms to remove or group parts of an image, making it smaller in the process.

There are two types of compression - lossy and lossless.

- **Lossy compression** removes parts of data from the file, resulting in a lower quality, lightweight image. *JPEG and GIF* are examples of lossy image types.
- **Lossless compression** maintains roughly the same image quality, i.e., it doesn't remove any data. It produces heavy, high-quality images. *RAW and PNG* are lossless image types.

To find the ideal compression level for your website, you have to experiment. Fortunately, there are lots of great tools for the job:

- You can use [imagemin](#) if you're comfortable with command-line tools;
- If not, beginner-friendly tools like [Optimizilla](#) also do a great job;
- At [NitroPack](#), we also offer adjustable image quality as part of our image optimization stack.



Also, remember that as your website grows, you'll likely add more and more images. Eventually, you'll need a tool that optimizes images to your desired level automatically.

## 2. Choosing the right format

Here's the deal:

The tricky thing about choosing between image formats is finding a balance between quality and speed.

High-quality images are heavy but look great. Lower-res ones look worse but load faster.

In some cases, high-resolution images are necessary to stand out from the competition. Think photography and fashion sites.

For others (news sites and personal blogs), lower-res images are perfectly fine.

The choice here depends on your personal needs. Again, you have to run tests to see how much image quality affects your visitor's behavior.

Here's a quick checklist of rules you can use as a guide:

- Use **SVG** for images made up of simple geometric shapes like logos;
- Use **PNG** whenever you have to preserve quality while sacrificing a bit of speed;
- For an optimal balance between quality and UX, use **WebP** while keeping the original format as a backup since WebP doesn't have 100% browser support. That's the strategy we use at NitroPack.
- Also, WebP was considered risky in late 2023 due to a critical, actively exploited vulnerability (CVE-2023-4863) in the libwebp library, which could lead to remote code execution (RCE) or denial of service (DoS) by exploiting a heap buffer overflow in a maliciously crafted WebP image.

Again, don't forget to experiment with compression levels after choosing your image type.



3. Use the `srcset` attribute

A classic mistake when working with images is serving one large image to all screen sizes.

Large images look good on smaller devices, but they still have to be processed entirely. That’s a massive waste of bandwidth.

There is a better approach:

Provide different image sizes and let the browser decide which one to use based on the device. To do that, use the `srcset` attribute and specify the different widths of the image you want to serve. Here’s an example:

```

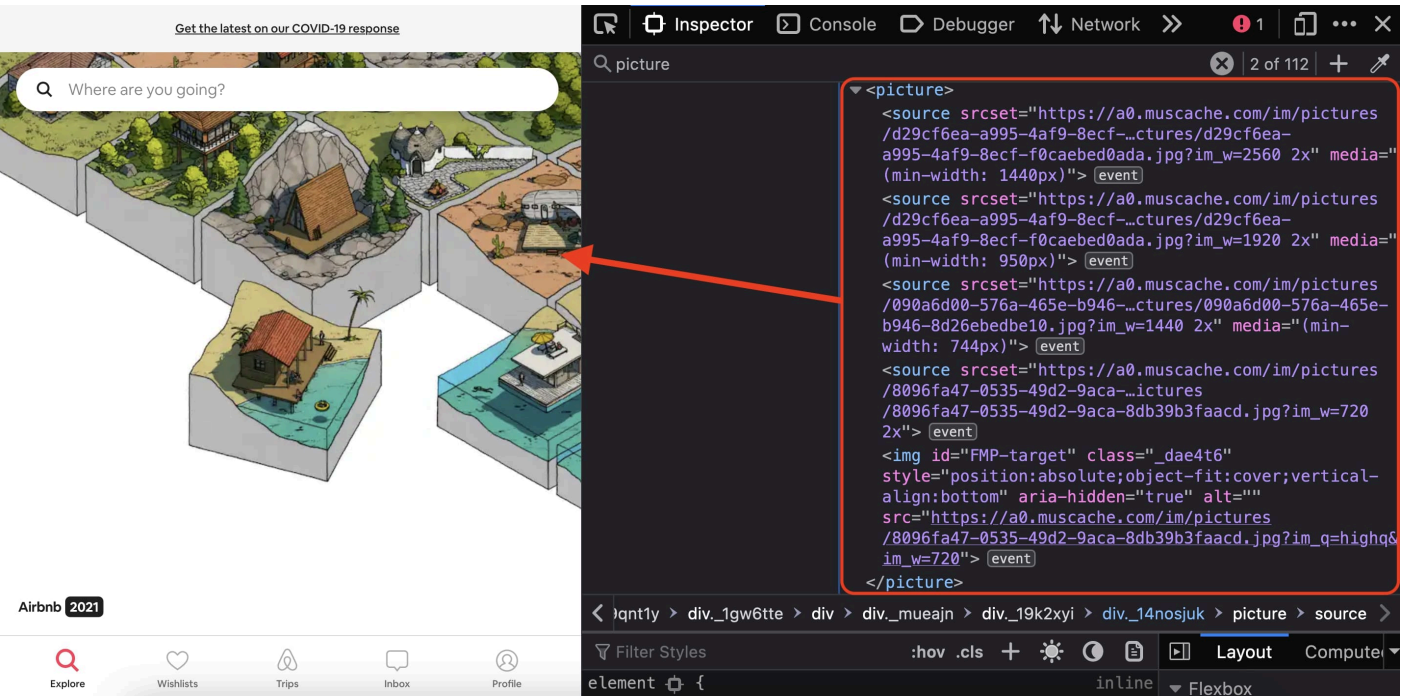
```

Width descriptors

As you can see, with `srcset`, we use w instead of px. If you want an image version to be 600px wide, you have to write 600w.

Again, this method outsources the choice of image size to the browser. You just provide the options.

Airbnb’s website utilizes this approach:

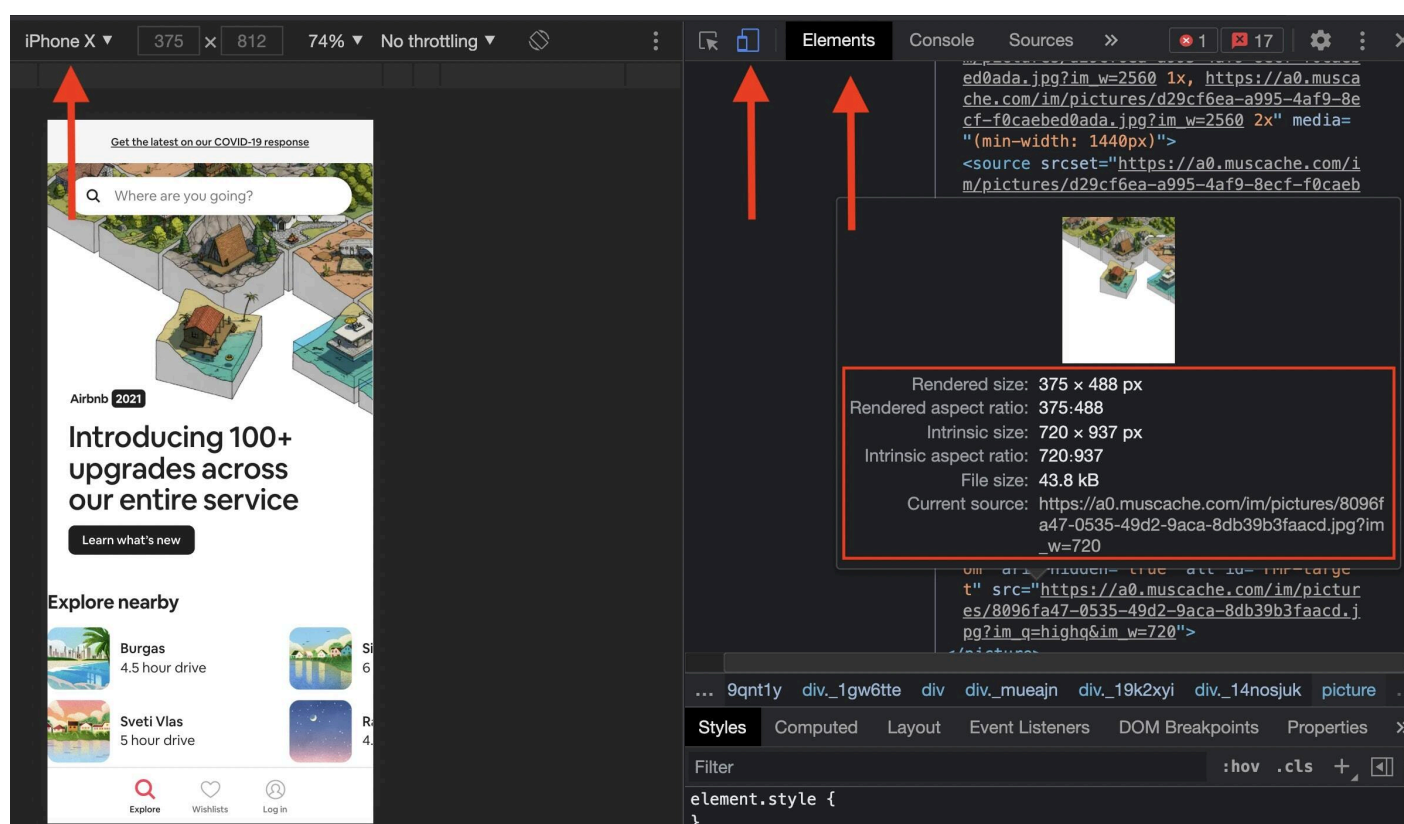


When deciding on the correct image sizes, use Google Analytics to figure out what percentage of your audience visits your site from a desktop or mobile device. The “Devices” report also has in-depth info about the specific devices your visitors use.

Audience		1,514	1,385
		% of Total: 3.30% (45,855)	% of Total: 3.15% (44,030)
Overview			
Active Users			
Lifetime Value <sup>BETA</sup>			
Cohort Analysis <sup>BETA</sup>			
Audiences			
User Explorer			
Demographics			
Interests			
Geo			
Behavior			
Technology			
Mobile			
Overview			
Devices			

<input type="checkbox"/>	1. Apple iPhone	388 (25.63%)	379 (27.36%)
<input type="checkbox"/>	2. (not set)	45 (2.97%)	41 (2.96%)
<input type="checkbox"/>	3. Huawei EML-AL00 P20	20 (1.32%)	20 (1.44%)
<input type="checkbox"/>	4. Apple iPad	19 (1.25%)	18 (1.30%)
<input type="checkbox"/>	5. Samsung SM-G975F Galaxy S10+	14 (0.92%)	12 (0.87%)
<input type="checkbox"/>	6. Huawei VOG-L29 P30 Pro	13 (0.86%)	12 (0.87%)
<input type="checkbox"/>	7. Apple iPhone 7 Plus	12 (0.79%)	12 (0.87%)
<input type="checkbox"/>	8. Xiaomi Redmi Note 7	12 (0.79%)	10 (0.72%)
<input type="checkbox"/>	9. Samsung SM-A505F Galaxy A50	9 (0.59%)	8 (0.58%)
<input type="checkbox"/>	10. Xiaomi MI 9	9 (0.59%)	6 (0.43%)

You should also use DevTools to check how images look on different viewports.



When it comes time to change image sizes use [Smart Resize](#) to resize in bulk.

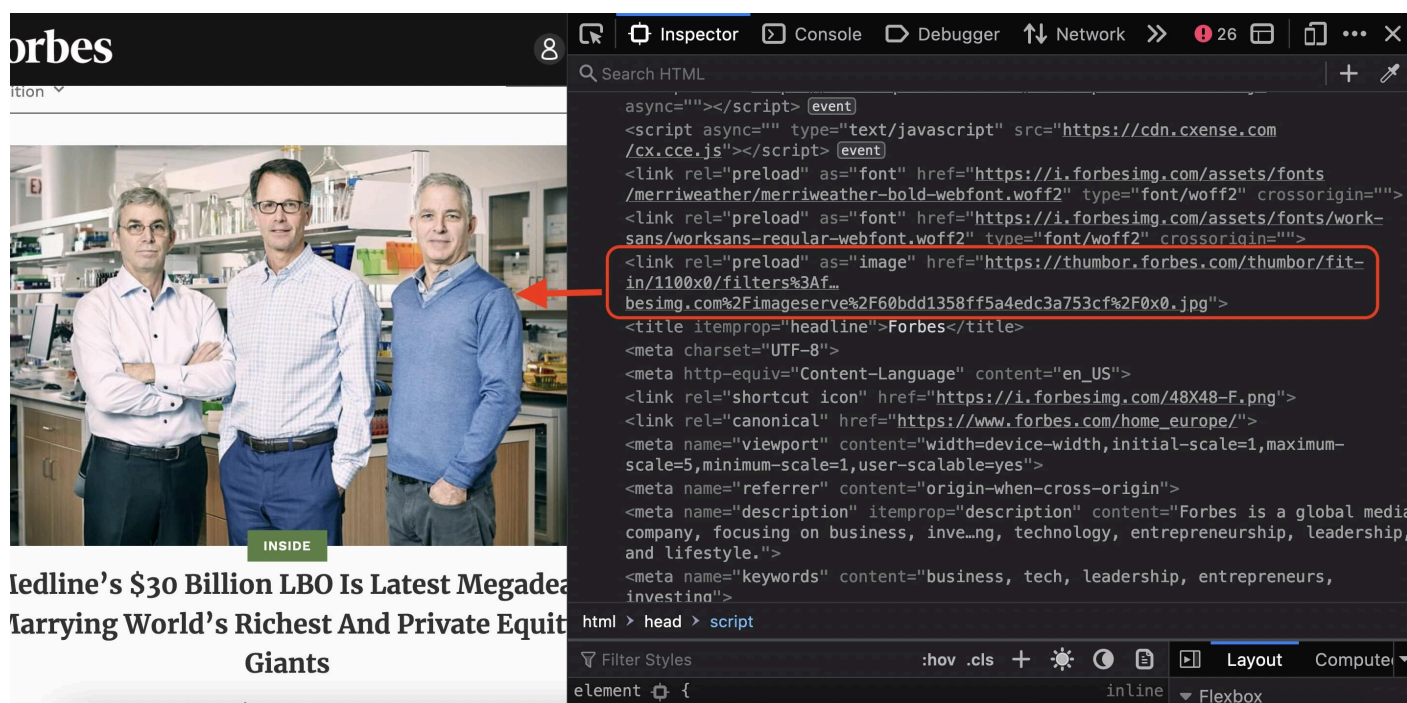
**Note for WordPress users:** Since version 4.4., WP automatically creates different versions of your images. It also adds the `srcset` attribute. If you're a WordPress user, you only need to [provide the right image sizes](#).

## 4. Preload Hero Images

This final tip is about optimizing how fast the browser discovers hero images.

Hero images are usually the most meaningful above the fold elements, so loading them faster is crucial for the user experience.

Forbes' site preloads the largest above the fold image on the homepage:



This technique tells the browser to prioritize that specific image when rendering the page.

Preloading can dramatically improve LCP, especially on pages with hero images that are loaded with:

- JavaScript;
- The `background-image` property in CSS.

Since browsers discover these images late, using `link rel=preload` can improve both actual and perceived performance.

If you're loading hero images with JS or CSS, check out [this tutorial on preloading images by Addy Osmani](#).

# CSS and JavaScript Optimization

Before the browser can render a page, it has to load, parse and execute all CSS and JavaScript files it finds while parsing the HTML.

That's why CSS and JavaScript are both render-blocking by default.

If left unoptimized, they can slow down the page load and consequently - hurt your LCP.

Here's how you can optimize them:

## 1. Minify and compress code files

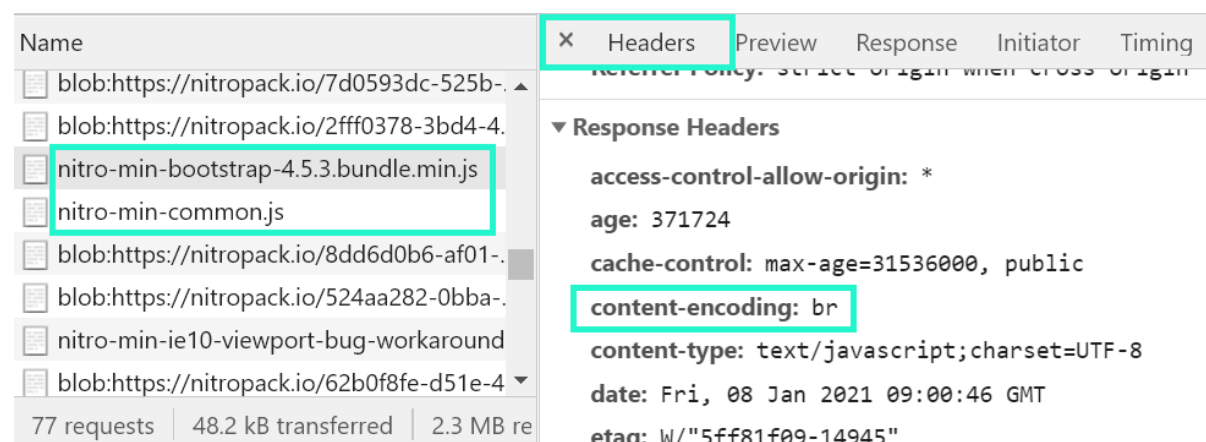
Minification removes unnecessary parts from code files like comments, whitespace, and line-breaks. It produces a small to medium file size reduction.

On the other hand, compression reduces the volume of data in the file by applying different algorithms. It typically produces a huge reduction in file size.

Both techniques are a must when it comes to performance.

Some **hosting companies** and **CDN providers** apply these techniques by default. It's worth checking to see if they're implemented on your site.

You can use the "Network" tab in DevTools and analyze the response headers for a file to see if that's the case:



Most minified files have ".min" somewhere in their name. Compressed files have a **content-encoding response header**, usually with a gzip or br value.

If your site's files aren't minified or compressed, I suggest you get on it right away. Ask your hosting company and CDN provider if they can do this for you.

If they can't, there are lots of minification and compression tools, including free ones.

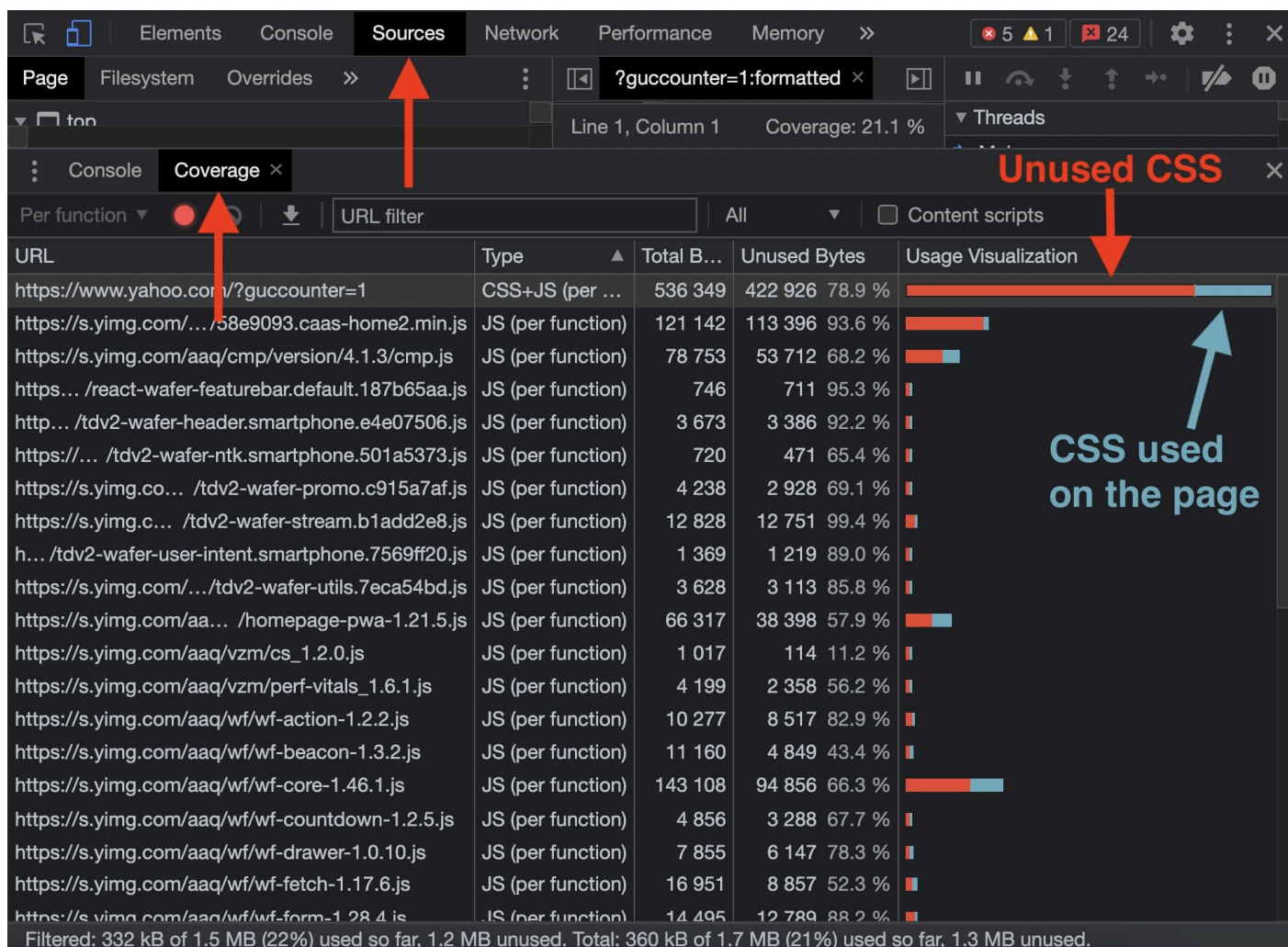
## 2. Implement Critical CSS

Implementing Critical CSS is a three-step process involving:

- Finding the CSS that styles above the fold content on different viewports;
- Placing (inlining) that CSS directly in the page's **head** tag;
- Deferring the rest of the CSS.

You can start by using the "Coverage" panel in DevTools to figure out how much of each CSS file is used on the page.





You can arrange the resources by type and go through each CSS and JS file.

Obviously, CSS that is not used on the page isn't critical. On that note, it's worth trying to remove or reduce this unused CSS, as it can slow down rendering.

Next, to extract the Critical CSS, you'll need to go through the code by hand or use a tool. Two great options for the job are [criticalCSS](#) and [critical](#).

Once extracted, inline the Critical CSS in the head tag of your page.

Finally, load the rest of the CSS asynchronously. [Google recommends](#) using `link rel="preload", as="style", a nulled onload handler` and nesting the link to the stylesheet in a `noscript` element.

```
<link rel="preload" href="styles.css" as="style"
onload="this.onload=null;this.rel='stylesheet'">
<noscript><link rel="stylesheet" href="styles.css"></noscript>
```

Also, don't forget to consider different viewports. Desktop and mobile users don't see the same above the fold content. To take full advantage of this technique, you need different Critical CSS based on the device type.

### 3. Deliver smaller JavaScript payloads

JavaScript is one of the main reasons for slow load times across the web. Like images, you have to optimize your website's JavaScript if you want great performance.

When it comes to LCP, **splitting JavaScript bundles** is a great way to improve your score.

The idea is to only send the code needed for the initial route. Everything not included in the initial bundle should be provided later on. That way, there's less JavaScript that needs to be parsed and compiled at one time.




Some popular tools for the job are [webpack](#), [Rollup](#) and [browserify](#).

For more information on code-splitting, check out [this article by web.dev](#).

## Faster Server Response Time

Reducing initial server response time is one of the most common suggestions in PageSpeed Insights.

**Opportunities** — These suggestions can help your page load faster. They don't **directly affect** the Performance score.

Opportunity	Estimated Savings
 <b>Reduce initial server response time</b>	 <b>3.8 s</b> ^
Keep the server response time for the main document short because all other requests depend on it. <a href="#">Learn more</a> .	
 Themes, plugins, and server specifications all contribute to server response time. Consider finding a more optimized theme, carefully selecting an optimization plugin, and/or upgrading your server.	

Here are some of the steps you can take to fix this issue:

- **Upgrade your hosting plan.** If you're on a cheap, shared hosting plan, you need to upgrade. It's impossible to have a fast website with a slow host server;
- **Optimize your server.** Lots of factors can impact your server's performance, especially once traffic spikes. Use [this tutorial by Katie Hempenius](#) to assess, stabilize, improve and monitor your server;
- **Reduce the site's reliance 3rd party plugins, tools and libraries.** They increase the amount of code that has to be executed on the server or browser. This increases resource consumption and hurts metrics like Time to First Byte, First Contentful Paint and LCP;
- **Take maximum advantage of caching.** **Caching is the backbone of great web performance. Many assets can be cached for months or even a year (logos, nav icons, media files).** Also, if your HTML is static, you can cache it, which can reduce TTFB significantly;
- **Use a CDN.** **A CDN reduces the distance between visitors and the content they want to access.** To make your job as easy as possible, get a caching tool with a built-in CDN;
- **Use service workers.** Service workers let you reduce the size of HTML payloads by avoiding the repetition of common elements. Once installed, service workers request the bare minimum of data from the server and transform it into a full HTML doc. Check out [this tutorial by Philip Walton](#) for more details.

## Limited and Optimized Client-Side Rendering

Client-side rendering (CSR) means using JavaScript to render pages directly in the browser.

This approach offloads tasks (data fetching, routing, etc.) away from the server to the client.

**At first, CSR might be a perfect solution, but it becomes increasingly difficult to maintain as you add more JavaScript.**

If you've implemented CSR, you need to take special care when optimizing your JavaScript. **Code splitting, compression and minification are a must.**

Also, using [HTTP/2 Server Push](#) and **link rel=preload** can help deliver critical resources sooner.

Finally, you can try **combining CSR with prerendering** or adding server-side rendering in the mix. The approach you take here depends on your website's tech stack. **The important thing is to be aware of how much work you're putting on the client and how that affects performance.**

For a deep dive into the topic, I recommend this comprehensive guide to [Rendering on the Web](#).

## Use **rel=preload**, **rel=preconnect** and **rel=dns-prefetch**

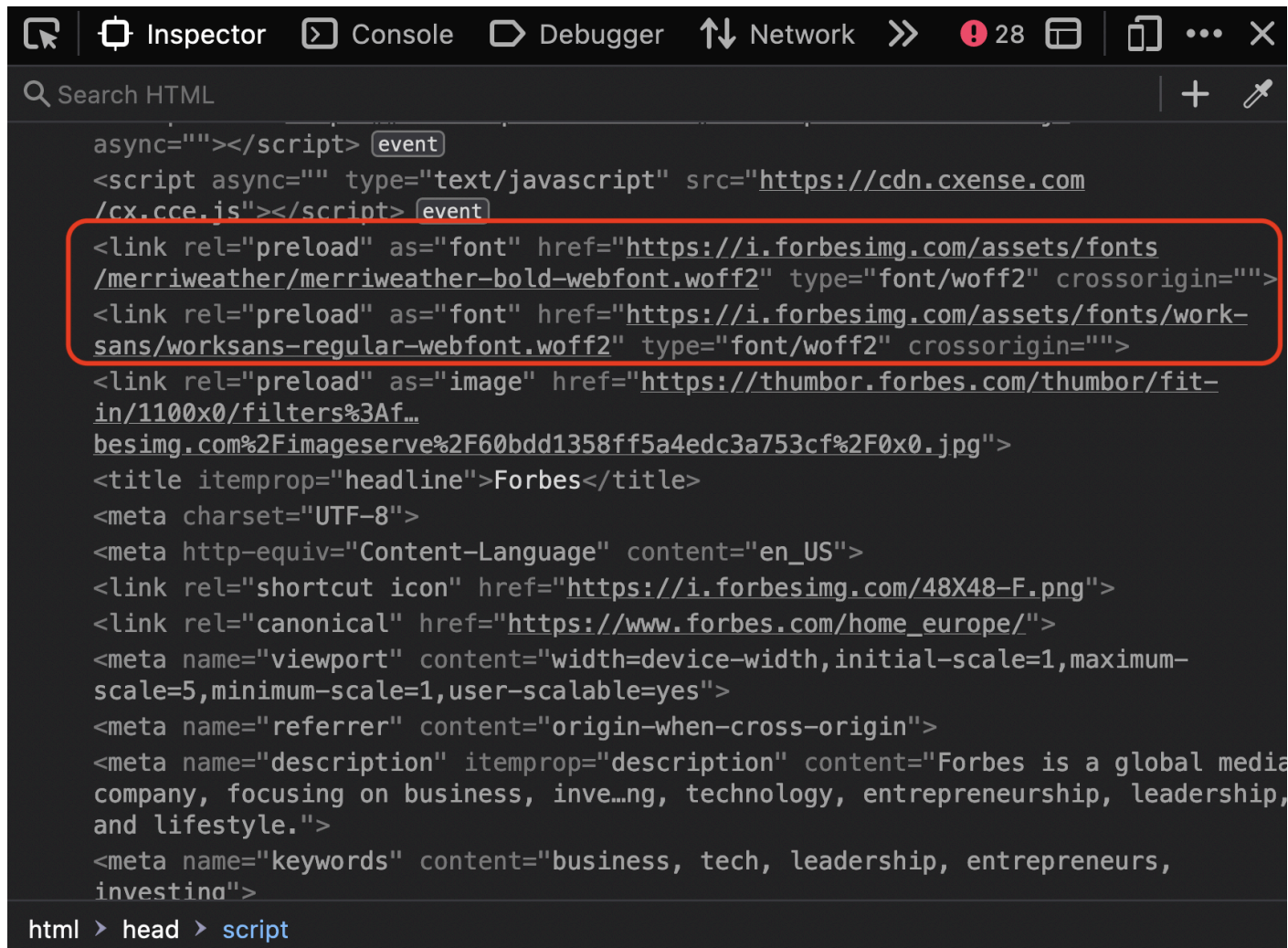
**These three attributes** help the browser by pointing it to resources and connections it needs to handle first.

First, use **rel=preload** for resources the browser should prioritize. Typically, these are above the fold images, videos, Critical CSS, or fonts. It's as simple as adding a few lines to **head tag** like this:

```
<head>
  <link rel="preload" as="font" href="https://fonturl.com/"
  type="font/woff2" crossorigin>
</head>
```

When preloading fonts, the like `as="font"`, `type="font/woff2"` and `crossorigin` help the browser prioritize resources during the rendering process. As a bonus, preloading fonts also helps them meet FCP, which reduces layout shifts.

Forbes.com uses this technique to reduce their font load time:



Next, `rel=preconnect` tells the browser that you intend to establish a connection to a domain immediately. This reduces round-trips to important domains.

Again, implementing this is very simple:

```
<head>
  <link rel="preconnect" href="https://cdnprovider.com">
</head>
```

But be very careful when preconnecting.

Just because you can preconnect to a domain doesn't mean you should. Only do so for domains you need to connect to **right away**. Using it for unneeded hosts stalls all other DNS requests, resulting in more harm than good.

Finally, to save time on the [DNS lookup](#) for connections that aren't as critical, use `rel=dns-prefetch`.

```
<head>
  <link rel="dns-prefetch" href="https://cdnprovider.com">
</head>
```

Prefetching can also be used as a fall back to preconnect.

```
<head>
  <link rel="preconnect" href="https://cdnprovider.com">
  <link rel="dns-prefetch" href="https://cdnprovider.com">
</head>
```



All of these techniques are extremely useful for improving your website's performance metrics. Implement them if you haven't already. Just be careful when selecting which resources to preload and which hosts to preconnect to.

## Reducing LCP on WordPress sites

The techniques listed above help improve LCP on all sites, including those built on WordPress (WP). However, two other tips can also help optimize this metric on WP sites:

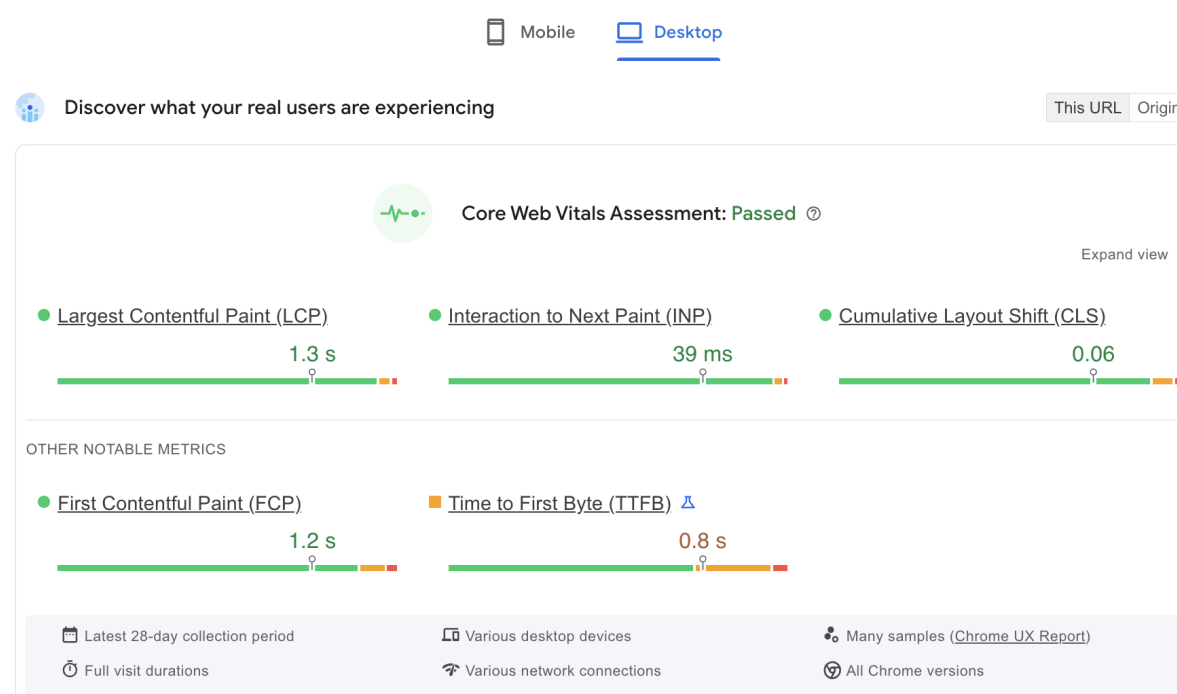
- **Choose a lightweight theme.** A bloated theme can significantly increase load times and hurt your Core Web Vitals, including LCP. That's why you should choose your theme carefully. For more details, check out [this research by Kinsta](#);
- **Don't use too many plugins.** While plugins provide useful functionalities, most of them also have a performance cost. They increase the amount of code that has to be executed, leading to higher resource consumption. Research each plugin carefully and monitor your site's performance after adding it.

Both of these tips are a must when working with WP sites. Combine them with the techniques listed above and you should see big LCP improvements.

## Other tools and best practices for monitoring the Core Web Vitals

Even if you don't have any LCP concerns, it's a good idea to periodically look at field data to detect potential problems.

Field data are gathered by the Chrome User Experience Report (CrUX). These data show how **real users** experience your site. You can find field data at the top of your PageSpeed Insights report:



Since Google evaluates a page's Core Web Vitals for the previous 28-day period, you should test important pages at least once a month.

If PageSpeed Insights doesn't display this section due to lack of data, you can use different tools to access the CrUX dataset:

- [The Chrome UX Report API](#) - requires some experience with JavaScript and JSON;
- [BigQuery](#) - requires a Google Cloud project and SQL skills;
- [The Core Web Vitals report in Google Search Console](#) - very beginner-friendly, useful for marketers, SEOs and webmasters.

Which tool you choose depends on your preference. The important thing is to be aware of any potential issues with your website's LCP (and the other [Core Web Vitals](#).)



