Efficiently load third-party JavaScript

- Use async or defer
  - async
  - defer
- Establish early connections to required origins
  - preconnect
  - dns-prefetch
- Lazy-load third-party resources
- Optimize how you serve third-party scripts
  - Third-party CDN hosting
  - Self-host third-party scripts
  - Use service workers to cache scripts from third-party servers

If a third-party script is slowing down your page load, you have two options to improve performance:

- Remove it if it doesn't add clear value to your site.
- Optimize the loading process.

This post explains how to optimize the loading process of third-party scripts with the following techniques:

1. Using the `async` or `defer` attribute on `<script>` tags
2. Establishing early connections to required origins
3. Lazy loading
4. Optimizing how you serve third-party scripts

Use `async` or `defer`

Because synchronous scripts delay DOM construction and rendering, you should always load third-party scripts asynchronously unless the script has to run before the page can be rendered.

The `async` and `defer` attributes tell the browser that it may go on parsing the HTML while loading the script in the background, and then execute the script after it loads. This way, script downloads don't block DOM construction and page rendering. The result is that the user can see the page before all scripts have finished loading.

```
<script async src="script.js">


<script defer src="script.js">
```

The difference between `async` and `defer` is when they start executing the scripts.

`async`

Scripts with the `async` attribute execute at the first opportunity after they finish downloading and before the window's load event. This means it's possible (and likely) that `async` scripts will not be executed in the order in which they appear in the HTML. It also means they can interrupt DOM building if they finish downloading while the parser is still at work.

{% Img src="image/admin/tCqsJ3E7m4lpKOprXu5B.png", alt="Diagram of parser blocking script with async attribute", width="800", height="252" %}

`defer`

Scripts with the `defer` attribute execute after HTML parsing is completely finished, but before the `DOMContentLoaded` event. `defer` guarantees scripts will be executed in the order they appear in the HTML and will not block the parser.

{% Img src="image/admin/Eq0mcvDALKibHe15HspN.png", alt="Diagram of parser flow with a script with defer attribute", width="800", height="253" %}

- Use `async` if it's important to have the script run earlier in the loading process.
- Use `defer` for less critical resources. A video player that's below-the-fold, for example.

Using these attributes can significantly speed up page load. For example, Telegraph recently deferred all of their scripts, including ads and analytics, and improved the ad loading time by an average of four seconds.

{% Aside %} Analytics scripts are usually loaded early so you don't miss any valuable analytics data. Fortunately, there are patterns to initialize analytics lazily while retaining early page-load

data. {% endAside %}

Establish early connections to required origins

You can save 100–500 ms by establishing early connections to important third-party origins.

Two `<link>` types can help here:

- `preconnect`
- `dns-prefetch`

`preconnect`

`<link rel="preconnect">` informs the browser that your page intends to establish a connection to another origin, and that you'd like the process to start as soon as possible. When the request for a resource from the pre-connected origin is made, the download starts immediately.

```
<link rel="preconnect" href="https://cdn.example.com">
```

{% Aside 'caution' %} Only preconnect to critical domains you will use soon because the browser closes any connection that isn't used within 10 seconds. Unnecessary preconnecting can delay other important resources, so limit the number of preconnected domains and test the impact preconnecting makes. {% endAside %}

`dns-prefetch`

`<link rel="dns-prefetch>` handles a small subset of what is handled by `<link rel="preconnect">`. Establishing a connection involves the DNS lookup and TCP handshake, and for secure origins, TLS negotiations. `dns-prefetch` instructs the browser to only resolve the DNS of a specific domain before it has been explicitly called.

The `preconnect` hint is best used for only the most critical connections; for less critical third-party domains use `<link rel=dns-prefetch>`.

```
<link rel="dns-prefetch" href="http://example.com">
```

Browser support for `dns-prefetch` is slightly different from `preconnect` support, so `dns-prefetch` can serve as a fallback for browsers that don't support `preconnect`. Use separate link tags to implement this safely:

```
<link rel="preconnect" href="http://example.com">
<link rel="dns-prefetch" href="http://example.com">
```

Lazy-load third-party resources

Embedded third-party resources can be a big contributor to slow page speed when constructed poorly. If they aren't critical or are below the fold (that is, if users have to scroll to view them), lazy loading is a good way to improve page speed and paint metrics. This way, users will get the main page content faster and have a better experience.

> {% Img src="image/admin/uzPZzkgzfrv2Oy3UQPrN.png", alt="A diagram of a webpage shown on a mobile device with scrollable content extending beyond the screen. The content that's below-the-fold is desaturated because it's not loaded yet.", width="366", height="438" %}

One effective approach is to lazy-load third-party content after the main page content loads. Ads are a good candidate for this approach.

Ads are an important source of income for many sites, but users come for the content. By lazy loading ads and delivering the main content faster, you can increase the overall viewability percentage of an ad. For example, MediaVine switched to lazy-loading ads and saw a 200% improvement in page load speed. DoubleClick have guidance on how to lazy-load ads in their official documentation.

An alternative approach is to load third-party content only when users scroll down to that section of the page.

Intersection Observer is a browser API that efficiently detects when an element enters or exits the browser's viewport and it can be used to implement this technique. lazysizes is a popular JavaScript library for lazy loading images and `iframes`. It supports YouTube embeds and widgets. It also has optional support for IntersectionObserver.

{% Aside 'caution' %} Be careful when lazy loading resources with JavaScript. If JavaScript fails to load, perhaps due to flaky network conditions, your resources won't load at all. {% endAside %}

Using the `loading` attribute for lazy loading images and iframes is a great alternative to JavaScript techniques, and it has recently become available in Chrome 76!

Optimize how you serve third-party scripts

Third-party CDN hosting

It's common for third-party vendors to provide URLs of JavaScript files that they host, usually on a content delivery network (CDN). The benefits of this approach are that you can get started quickly—just copy and paste the URL—and there's no maintenance overhead. The third-party vendor handles server configuration and script updates.

But because they are not on the same origin as the rest of your resources, loading files from a public CDN comes with a network cost. The browser needs to perform a DNS lookup, establish a new HTTP connection, and, on secure origins, perform an SSL handshake with the vendor's server.

When you use files from third-party servers, you rarely have control over caching. Relying on someone else's caching strategy might cause scripts to be unnecessarily re-fetched from the network too often.

Self-host third-party scripts

Self-hosting third-party scripts is an option that gives you more control over a script's loading process. By self-hosting you can:

- Reduce DNS lookup and round-trip times.
- Improve HTTP caching headers.
- Take advantage of HTTP/2, or the newer HTTP/3.

For example, Casper managed to shave 1.7 seconds off load time by self-hosting an A/B testing script.

Self-hosting comes with one big downside though: scripts can go out of date and won't get automatic updates when there's an API change or a security fix.

{% Aside 'caution' %}

Manually updating scripts can add a lot of overhead to your development process and you might miss out on important updates. If you are not using CDN hosting for serving all the resources, you are also missing out on edge-caching and you have to optimize your server's compression. {% endAside%}

Use service workers to cache scripts from third-party servers

An alternative to self-hosting that allows you greater control over caching while still getting the third-party CDN benefits is using service workers to cache scripts from third-party servers. This gives you control over how often scripts are re-fetched from the network and makes it possible to create a loading strategy that throttles requests for non-essential third-party resources until the page reaches a key user moment. Using `preconnect` to establish early connections in this case can also mitigate the network costs to an extent.

□□□ #2
□ □□□ □□□ 2026-10-31 22:35:24 UTC
□ □□□ □□□ 2026-10-31 22:35:26 UTC