Understanding Performance

The hardest thing for me to do when working in the performance space is to understand what the metrics represent and whether we should aim for top performance or where to draw the line of “this is good enough”.

Part of this is based on the little touches we can put on our sites like animations or elements other than content, how they will affect performance and whether those effects will be felt equally across the users of the site.

The metrics I try to meet and what they mean

The best performance metrics around, I think, are Google’s Core Web Vitals. The current metrics tracked in Web Vitals are:

- **Largest Contentful Paint (LCP)** (*loading* performance) reports the render time of the largest image or text block visible within the viewport
- **First Input Delay (FID)** (*interactivity*) reports the time from when a user first interacts with a page to the time when the browser is actually able to begin processing event handlers in response to that interaction.
- **Cumulative Layout Shift (CLS)** (*visual stability*) reports the sum total of all individual layout shift scores for every unexpected layout shift that occurs during the entire lifespan of the page.

The table below, taken from Defining the Core Web Vitals metrics thresholds shows the three different web vitals metrics, their values, and the percentile.

	Good	Poor	Percentile
Largest Contentful Paint	≤2500ms	>4000ms	75
First Input Delay	≤100ms	>300ms	75
Cumulative Layout Shift	≤0.1	>0.25	75

While it’s true that a 75 percentile in Web Vitals measurements reduces the chance

of outliers affecting the result, I have to wonder where the outliers are and if they are outliers at all.

In Defining the Web Vitals metrics thresholds, Bryan McQuade writes that:

> Additionally, to classify the overall performance of a page or site, we use the 75th percentile value of all page views to that page or site. In other words, if at least 75 percent of page views to a site meet the “good” threshold, the site is classified as having “good” performance for that metric. Conversely, if at least 25 percent of page views meet the “poor” threshold, the site is classified as having “poor” performance. So, for example, a 75th percentile LCP of 2 seconds is classified as “good”, while a 75th percentile LCP of 5 seconds is classified as “poor”.

But what happens if your users come from places with poor network connectivity or are using lower-end devices? Will that penalize your Web Vitals score?

The Science Behind Web Vitals discusses the scientific reasoning behind these vital measurements.

User timings

The User Timing API allows you to create custom measurements for your page.

This example will create a measure telling us how long did `zenscroll.min.js` take to load by calculating the difference between the end mark and the starting mark we identified and record it as a measure.

```
<script>performance.mark('begin zenscroll');</script>
<script async src="./js/zenscroll.</script>/script>
<script>p<script>performance.mark('end zenscroll');
<script>performan<script>performance.measure('zenscroll', 'begin zenscrol
```

These measurements are visible in Lighthouse and the DevTools timeline.

Gathering Performance Data

We know what performance data we want to gather, let's look at what tools we can use to gather that data.

Lighthouse

One of the first tools that I used was Google's Lighthouse that runs as a Node CLI, inside DevTools, inside the Web.dev site, as part of Pagespeed Insights and Webpagetest among others.

The most frustrating aspect of the experience is that is not consistent. I will get different results running Lighthouse from Chrome Canary than what I get from running NodeJS Lighthouse from the command line on the same machine under the same network conditions.

I stopped running Lighthouse in Chrome stable because, as my everyday browser, it uses plugins and It's prone to Lighthouse failures. However, I don't run many plugins in Chrome Canary and there are no plugins to slow down Node... so what would cause such a discrepancy between two versions of the software running on the same network with the same hardware?

The lighthouse project has a Wiki page on score variability where they go into detail about the different aspects that can impact the results of a Lighthouse run and what impact may they have in different scenarios: user run, PageSpeed Insights, and controlled lab situations.

CrUX report

This is another frustrating piece of performance data.

The Chrome User Experience (CrUX) report provides a large set of data about sites on the web. It uses Real User Measurement data, ***aggregated from users who have opted-in to syncing their Chrome browsing history, have not set up a Sync passphrase, and have usage statistic reporting enabled*** (according to the CrUX page on Google Developers).

It's awesome if there is data for your site. Because it's based on voluntary aggregate data it is possible that your origin is not included in the report because not enough people who meet the criteria for inclusion visit your site.

No matter the tool I use, if it incorporates CrUX data I will see that little blank space where the data should go and, maybe, the message the CrUX doesn't have enough data for the origin.

Furthermore, CrUX only gathers aggregate data per origin and, to my knowledge, it does not give you page-level performance data.

Web Vitals Library

Google provides a web-vitals library that will measure these vitals statistics for you.

As with all node packages, the first step is to install it. Run the following command:

```
npm i web-vitals
```

Once it is installed we can use it inside a JS module either created with a bundler or build tool or with a script tag with the `type='module'` attribute somewhere in your document.

You can also serve web vitals from a CDN that can serve modules, like unpkg.

```
<script async type='module'>
import {getCLS, getFID, getLCP} from 'https://unpkg.com/web-vitals?module

getCLS(console.log);
getFID(console.log);
getLCP(console.log);
</script>
```

Once we get the values we can either send them to an analytics account for further processing or do something else with the beacon API if we have our own logger.

Creating your own measurements

The User Timing API provides methods in the `performance` object that would allow you to track the timing of events.

`performance.mark()` creates a timestamp in the browser's performance entry buffer with the given name.

`performance.measure()` creates a named timestamp in the browser's performance entry buffer between marks. When measuring between two marks, there is a start mark and end mark. The named timestamp is referred to as a measure.

In this example, there are multiple marks and measures. I've grouped them into three categories:

- Begin an event, like `begin analytics`
- End and event, `end analytics`
- A measure recording the delta between the beginning and ending of an event (`analytics`).

These marks and measurements will appear in the DevTools performance timeline and provide additional data points for performance analysis.

```
<script>performance.mark('begin analytics');</script>
 <script async src="your analytics</script>here"></script>
 <script></script>
   // gtag manager configuration goes here

 <script>performan<script>performance.mark('end analytics');</script>
 <script>performance.measure('analytics', 'begin analytics', 'end analyt

 <script>performance.mark('begin zenscroll');pt async src="./js/zenscrol
 <scri</script></script>
 <script>performance.mark('end zenscroll');</script>
 <script>performance.measure('zenscroll', 'begin zenscroll', 'end zenscr
```

This works nicely as all the custom marks and measurements appear in the DevTools timeline but it's my results and that's it.

I've asked how we can push these measurements to Google Analytics or some other analytics gathering system. When I get an answer, I will write a new post about it.