Analytics

Next.js Speed Insights allows you to analyze and measure the performance of pages using different metrics.

You can start collecting your Real Experience Score with zero-configuration on Vercel deployments.

The rest of this documentation describes the built-in relayer Next.js Speed Insights uses.

Build Your Own

First, you will need to create a custom App component and define a `reportWebVitals` function:

pages/_app.js

```
export function reportWebVitals(metric) {
  console.log(metric);
}

function MyApp({ Component, pageProps }) {
  return <Component {...pageProps} />;
}

export default MyApp;
```

This function is fired when the final values for any of the metrics have finished calculating on the page. You can use to log any of the results to the console or send to a particular endpoint.

The `metric` object returned to the function consists of a number of properties:

- `id` : Unique identifier for the metric in the context of the current page load
- `name` : Metric name

- `startTime`: First recorded timestamp of the performance entry in milliseconds ↗ (if applicable)
- `value`: Value, or duration in milliseconds ↗, of the performance entry
- `label`: Type of metric (`web-vital` or `custom`)

There are two types of metrics that are tracked:

- Web Vitals
- Custom metrics

Web Vitals

Web Vitals ↗ are a set of useful metrics that aim to capture the user experience of a web page. The following web vitals are all included:

- Time to First Byte ↗ (TTFB)
- First Contentful Paint ↗ (FCP)
- Largest Contentful Paint ↗ (LCP)
- First Input Delay ↗ (FID)
- Cumulative Layout Shift ↗ (CLS)
- Interaction to Next Paint ↗ (INP) *(experimental)*

You can handle all the results of these metrics using the `web-vital` label:

```
export function reportWebVitals(metric) {
  if (metric.label === 'web-vital') {
    console.log(metric); // The metric object ({ id, name, startTime, value, label }) is
  }
}
```

There's also the option of handling each of the metrics separately:

```
export function reportWebVitals(metric) {
  switch (metric.name) {
    case 'FCP':
      // handle FCP results
      break;
    case 'LCP':
      // handle LCP results
      break;
```

```
    case 'CLS':
      // handle CLS results
      break;
    case 'FID':
      // handle FID results
      break;
    case 'TTFB':
      // handle TTFB results
      break;
    case 'INP':
      // handle INP results (note: INP is still an experimental metric)
      break;
    default:
      break;
  }
}
```

A third-party library, web-vitals↗, is used to measure these metrics. Browser compatibility depends on the particular metric, so refer to the Browser Support↗ section to find out which browsers are supported.

Custom metrics

In addition to the core metrics listed above, there are some additional custom metrics that measure the time it takes for the page to hydrate and render:

- `Next.js-hydration`: Length of time it takes for the page to start and finish hydrating (in ms)
- `Next.js-route-change-to-render`: Length of time it takes for a page to start rendering after a route change (in ms)
- `Next.js-render`: Length of time it takes for a page to finish render after a route change (in ms)

You can handle all the results of these metrics using the `custom` label:

```
export function reportWebVitals(metric) {
  if (metric.label === 'custom') {
    console.log(metric); // The metric object ({ id, name, startTime, value, label }) is
  }
}
```

There's also the option of handling each of the metrics separately:

```
export function reportWebVitals(metric) {
  switch (metric.name) {
    case 'Next.js-hydration':
```

```
      // handle hydration results
      break;
    case 'Next.js-route-change-to-render':
      // handle route-change to render results
      break;
    case 'Next.js-render':
      // handle render results
      break;
    default:
      break;
  }
}
```

These metrics work in all browsers that support the User Timing API ↗.

Sending results to external systems

With the relay function, you can send results to any endpoint to measure and track real user performance on your site. For example:

```
export function reportWebVitals(metric) {
  const body = JSON.stringify(metric);
  const url = 'https://example.com/analytics';

  // Use `navigator.sendBeacon()` if available, falling back to `fetch()`.
  if (navigator.sendBeacon) {
    navigator.sendBeacon(url, body);
  } else {
    fetch(url, { body, method: 'POST', keepalive: true });
  }
}
```

Note: If you use Google Analytics ↗, using the `id` value can allow you to construct metric distributions manually (to calculate percentiles, etc.)

```
export function reportWebVitals({ id, name, label, value }) {
  // Use `window.gtag` if you initialized Google Analytics as this example:
  // https://github.com/vercel/next.js/blob/canary/examples/with-google-analytics/page
  window.gtag('event', name, {
    event_category:
      label === 'web-vital' ? 'Web Vitals' : 'Next.js custom metric',
    value: Math.round(name === 'CLS' ? value * 1000 : value), // values must be intege
    event_label: id, // id unique to current page load
    non_interaction: true, // avoids affecting bounce rate.
  });
}
```

Read more about sending results to Google Analytics ↗.

TypeScript

If you are using TypeScript, you can use the built-in type `NextWebVitalsMetric`:

pages/_app.tsx

```
import type { AppProps, NextWebVitalsMetric } from 'next/app';

function MyApp({ Component, pageProps }: AppProps) {
  return <Component {...pageProps} />;
}

export function reportWebVitals(metric: NextWebVitalsMetric) {
  console.log(metric);
}

export default MyApp;
```