Streaming SSR (Alpha)

React 18 will include architectural improvements to React server-side rendering (SSR) performance. This means you can use `Suspense` in your React components in streaming SSR mode and React will render them on the server and send them through HTTP streams. It's worth noting that another experimental feature, React Server Components, is based on streaming. You can read more about server components related streaming APIs in `next/streaming`. However, this guide focuses on basic React 18 streaming.

Enable Streaming SSR

Enabling streaming SSR means React renders your components into streams and the client continues receiving updates from these streams even after the initial SSR response is sent. In other words, when any suspended components resolve down the line, they are rendered on the server and streamed to the client. With this strategy, the app can start emitting HTML even before all the data is ready, improving your app's loading performance. As an added bonus, in streaming SSR mode, the client will also use selective hydration strategy to prioritize component hydration which based on user interaction.

To enable streaming SSR, set the experimental option `runtime` to either `'nodejs'` or `'edge'`:

```
// next.config.js
module.exports = {
  experimental: {
    runtime: 'nodejs',
  },
}
```

This option determines the environment in which streaming SSR will be happening. When setting to `'edge'`, the server will be running entirely in the Edge Runtime.

Streaming Features

next/dynamic

Dynamic imports through `React.lazy` have better support in React 18. Previously, Next.js supported dynamic imports internally without requiring `Suspense` or `React.lazy`. Now to embrace the official APIs on the React side, we provide you with `options.suspense` in `next/dynamic`.

```
import dynamic from 'next/dynamic'
import { lazy, Suspense } from 'react'

import Content from '../components/content'
```

```
// These two ways are identical:
const Profile = dynamic(() => import('./profile'), { suspense: true })
const Footer = lazy(() => import('./footer'))

export default function Home() {
  return (
    <div>
      <Suspense fallback={<Spinner />}>
        {/* A component that uses Suspense */}
        <Content />
      </Suspense>
      <Suspense fallback={<Spinner />}>
        <Profile />
      </Suspense>
      <Suspense fallback={<Spinner />}>
        <Footer />
      </Suspense>
    </div>
  )
}
```

Check out `next/streaming` for more details on building Next.js apps in streaming SSR mode.

Important Notes

`next/head` and `next/script` Using resource tags (e.g. scripts or stylesheets) in `next/head` won't work as intended with streaming, as the loading order and timing of `next/head` tags can no longer be guaranteed once you add Suspense boundaries. We suggest moving resource tags to `next/script` with the `afterInteractive` or `lazyOnload` strategy, or to `_document`. For similar reasons, we also suggest migrating `next/script` instances with the `beforeInteractive` strategy to `_document`.

Data Fetching Currently, data fetching within `Suspense` boundaries on the server side is not fully supported, which could lead to mismatching between server and client. In the short-term, please don't try data fetching within `Suspense`.

Styling Inline styles, Global CSS, CSS modules and Next.js built-in `styled-jsx` are supported with streaming. The Next.js team is working on the guide of integrating other CSS-in-JS solutions in streaming SSR. Stay tuned for updates.

> Note: The styling code should be only placed in client components, not server components, when using React Server Components