



# Streaming SSR



# Reference

reactwg : New Suspense SSR Architecture in React 18	<a href="https://github.com/reactwg/react-18/discussions/37">https://github.com/reactwg/react-18/discussions/37</a>
Next.js : What is Streaming?	<a href="https://beta.nextjs.org/docs/data-fetching/streaming-and-suspense#what-is-streaming">https://beta.nextjs.org/docs/data-fetching/streaming-and-suspense#what-is-streaming</a>
Hydrogen : Streaming server-side rendering (SSR)	<a href="https://shopify.github.io/hydrogen-v1/tutorials/streaming-ssr">https://shopify.github.io/hydrogen-v1/tutorials/streaming-ssr</a>
React API Reference : renderToPipeableStream	<a href="https://react.dev/reference/react-dom/server/renderToPipeableStream">https://react.dev/reference/react-dom/server/renderToPipeableStream</a>
React Suspense Architecture Demo	<a href="https://codesandbox.io/s/kind-sammet-j56ro?file=/src/App.js">https://codesandbox.io/s/kind-sammet-j56ro?file=/src/App.js</a>



**What?**

**How?**

**Why?**



**Hmm?**



# 1. 왜 필요할까?



## 1. Why

# 기존 SSR

Data Fetching → Rendering → Load JS → Hydration

Server

Client

Time →



- Fetching data on server
- Rendering HTML on server
- Loading code on the client
- Hydrating

- TTFB Time To First Byte
- FCP First Contentful Paint
- TTI Time To Interactive



## 1. Why

# 기존 SSR

Data Fetching → Rendering

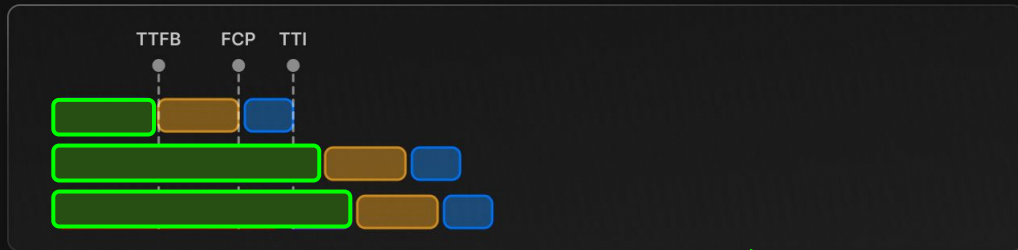
Server



Load JS → Hydration

Client

Time →



- Fetching data on server
- Rendering HTML on server
- Loading code on the client
- Hydrating



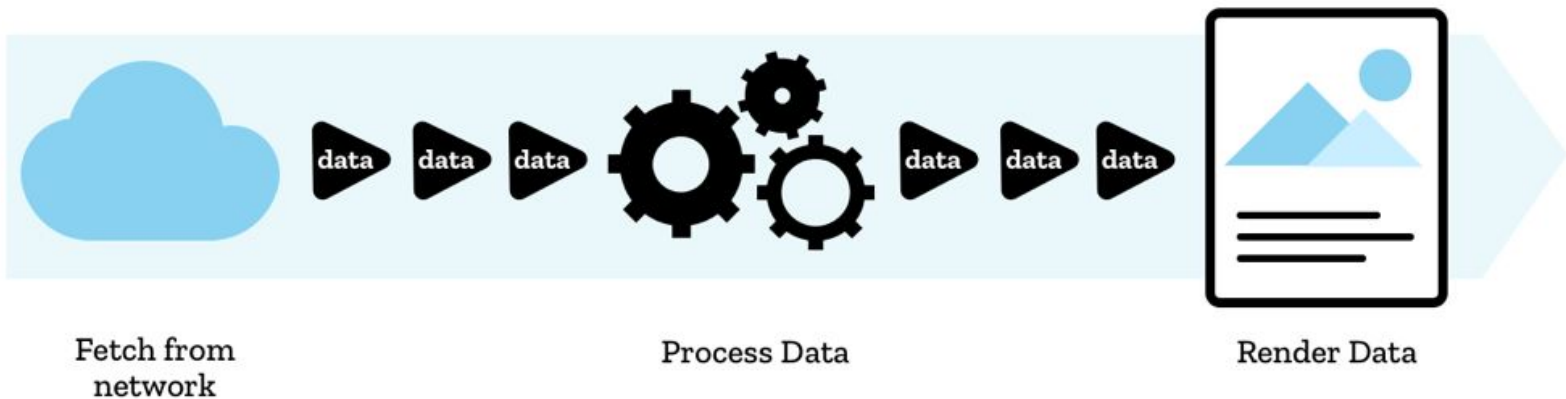
- TTFB Time To First Byte
- FCP First Contentful Paint
- TTI Time To Interactive



## 2. 무엇일까?

## 2. What

**Stream** : 시간이 지남에 따라 사용할 수 있게 되는 일련의 데이터 요소





## 2. What

Delayed by 1.5s

Delayed by 0.5s

Delayed by 1s

Delayed by 1.4s

Delayed by 0.6s

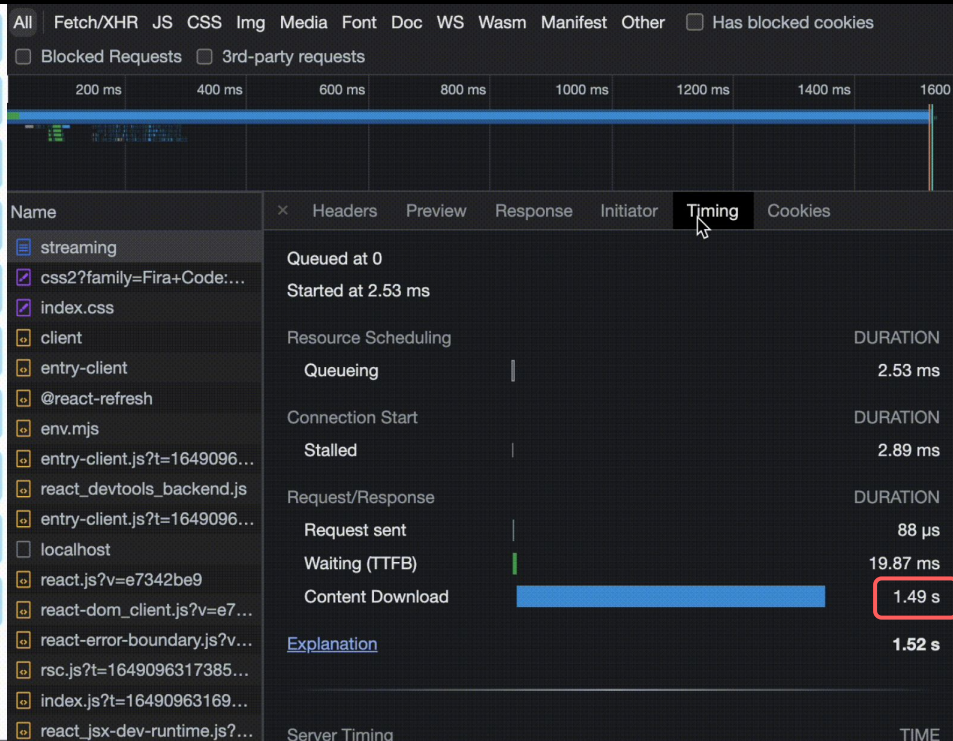
Delayed by 0.9s

Delayed by 0.7s

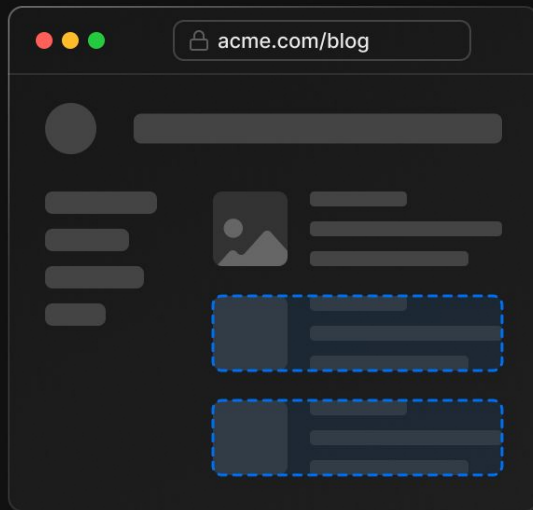
Delayed by 1.2s

Delayed by 1.3s

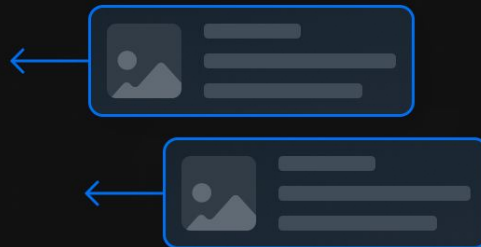
Delayed by 0.8s



# Streaming SSR : 서버에서 HTML을 여러 Chunk로 나눠 점진적으로 전송하는 것



Partial content with loading state



Suspended content  
streaming in


# Benefits of streaming SSR

---

- **Fast TTFB** : 최초 HTML을 응답 받은 후, `<script>` 태그와 함께 스트리밍 형태의 추가 콘텐츠를 받을 수 있다.
- **Progressive hydration** : 나머지 HTML과 JavaScript가 완전히 다운로드되기 전에 Hydration을 최대한 빨리 시작할 수 있게 해준다. 또한 사용자가 상호작용하는 부분에 대한 Hydration에 우선순위를 제공한다.




## 2. What

 **vercel / next.js** Public

<> Code Issues 1.3k Pull requests 258 Discussions Actions Projects Security 8 Insights


# Use HTML Streaming? #3342


Closed jlei523 opened this issue on Nov 27, 2017 · 10 comments



jlei523 commented on Nov 27, 2017


On a page with many dynamic components, it can be slow to render the page. If we can stream components down when it's ready asynchronously, we can improve the initial byte speed.  
  
See example:  
<https://github.com/aickin/react-dom-stream>

 1




andregoncalvesdev commented on Nov 27, 2017

it's available on react16: <https://reactjs.org/docs/react-dom-server.html#rendertonodestream>

Member

timneutkens commented on Nov 28, 2017 · edited

It's something we have to talk about more internally cause a lot of features won't work the way you expect them to when stream rendering (next/head etc) 🙌 Definitely one of our goals 🙌

 20

<https://github.com/vercel/next.js>


## 2. What

vercel / next.js Public

<> Code Issues 1.3k Pull requests 258 Discussions Actions Projects Security 8 Insights

# Stream rendering to reduce TTFB and CPU load #1209






Open ghost opened this issue on Feb 20, 2017 · 48 comments


 ghost commented on Feb 20, 2017 · edited by ghost

I suggest to stream-render `pages > 50KB` ([hypothetical stream overhead](#)) to reduce TTFB and CPU load.

- Inferno offers `streamAsString`, `streamAsStaticMarkup`, `streamQueueAsString`, `streamQueueAsStaticMarkup` with [inferno-server \(master\)](#). As of 1.2.1, the stream renderer was experimental and 10-15% slower than `renderToString`. Should wait for 1.3 (currently RC3).
- React with [aickin/react-dom-stream](#), which replaces `renderToString` and `renderToStaticMarkup`.
- [thysultan/dio.js](#) with `renderToStream` ([SSR/renderToStream.js](#)).

It would supersede #767. Preact does not support streaming ([preactjs/preact#23 \(comment\)](#)).

 102  18  21  19  13

 arunoda commented on Feb 20, 2017 Contributor

One thing to mention..

Stream rendering usually won't add much CPU improvements since the amount of work to be done is the same. But it'll reduce the response time.

It's a pretty good idea to provide a way to customize the SSR rendering system. But I think for now, we'll stick with the React's `renderToString()` methods by default.

This is something we could do after 2.0.



# 3. 어떻게 동작할까?



### 3. How

#### Dan Abramov's demo - Request

```
1  app.get(  
2    '/',  
3    handleErrors(async function (req, res) {  
4      await waitForWebpack();  
5      render(req.url, res);  
6    } ),  
7  );  
8
```

### 3. How

#### Dan Abramov's demo - Render


```
1 function render(url, res) {
2   let didError = false;
3   const data = createServerData();
4
5   res.send(
6     '<!DOCTYPE html>' +
7     renderToString(
8       <DataProvider data={data}>
9         <App assets={assets} />
10      </DataProvider>,
11    ),
12  );
13 }
```



```
1 function render(url, res) {
2   let didError = false;
3   const data = createServerData();
4
5   const stream = renderToPipeableStream(
6     <DataProvider data={data}>
7       <App assets={assets} />
8     </DataProvider>,
9   {
10     bootstrapScripts: [assets['main.js']],
11     onShellReady() {
12       res.statusCode = didError ? 500 : 200;
13       res.setHeader('Content-type', 'text/html');
14       stream.pipe(res);
15     },
16     onError(x) {
17       didError = true;
18       console.error(x);
19     },
20   },
21 );
22
23 setTimeout(() => stream.abort(), ABORT_DELAY);
24 };
```



### 3. How

 Search

react@18.2.0

API REFERENCE > SERVER APIS >

renderToPipeableStream

renderToPipeableStream renders a React tree to a pipeable [Node.js Stream](#).

```
const { pipe, abort } = renderToPipeableStream(reactNode, options?)
```

react-dom@18.2.0

react@18.2.0

Hooks >

Components >

APIs >

Returns

renderToPipeableStream returns an object with two methods:

- pipe outputs the HTML into the provided [Writable Node.js Stream](#). Call pipe in onShellReady if you want to enable streaming, or in onAllReady for crawlers and static generation.
- abort lets you [abort server rendering](#) and render the rest on the client.

### 3. How

```
1 import { Suspense, lazy } from 'react';
2 import { ErrorBoundary } from 'react-error-boundary';
3 import Html from './Html';
4 import Spinner from './Spinner';
5 import Layout from './Layout';
6 import NavBar from './NavBar';
7
8 const Comments = lazy(() => import('./Comments' /* webpackPrefetch: true */));
9 const Sidebar = lazy(() => import('./Sidebar' /* webpackPrefetch: true */));
10 const Post = lazy(() => import('./Post' /* webpackPrefetch: true */));
11
12 export default function App({ assets }) {
13   return (
14     <Html assets={assets} title="Hello">
15       <Suspense fallback={<Spinner />}>
16         <ErrorBoundary FallbackComponent={Error}>
17           <Content />
18         </ErrorBoundary>
19       </Suspense>
20     </Html>
21   );
22 }
23
24 function Content() {
25   return (
26     <Layout>
27       <NavBar />
28       <aside className="sidebar">
29         <Suspense fallback={<Spinner />}>
30           <Sidebar />
31         </Suspense>
32       </aside>
33       <article className="post">
34         <Suspense fallback={<Spinner />}>
35           <Post />
36         </Suspense>
37         <section className="comments">
38           <h2>Comments</h2>
39           <Suspense fallback={<Spinner />}>
40             <Comments />
41           </Suspense>
42         </section>
43         <h2>Thanks for reading!</h2>
44       </article>
45     </Layout>
46   );
47 }
48
```

### 3. How

Name	Headers	Preview	Response	Initiator	Timing	Cookies
j56ro.sse.codesandbox.io			<pre>    &lt;/script&gt;     &lt;script defer src="https://static.cloudflareinsights.com/beacon.min.js/vb26e4fa9e5134444860be286fd877185167933512 2    &lt;/body&gt; &lt;/html&gt; &lt;script src="/main.js" asynce=""&gt;&lt;/script&gt; &lt;div hidden id="S:0"&gt;   &lt;p class="comment"&gt;Wait, it doesn't wait for React to load?&lt;/p&gt;   &lt;p class="comment"&gt;How does this even work?&lt;/p&gt;   &lt;p class="comment"&gt;I like marshmallows&lt;/p&gt; &lt;/div&gt; &lt;script&gt; function \$RC(a, b) {   a = document.getElementById(a);   b = document.getElementById(b);   b.parentNode.removeChild(b);   if (a) {     a = a.previousSibling;     var f = a.parentNode;     , c = a.nextSibling;     , e = 0;     do {       if (c &amp;&amp; 8 =         var d = c;         if ("/\$"           if (0             break;           else             else               "             }             d = c.next;             f.removeChild(c);             c = d;           } while (c);           for (; b.first;             f.insertBefore(b, c);             a.data = "\$";             a._reactRetry           }         }       ;\$RC("B:0", "S:0")     &lt;/script&gt;</pre>			

Diagram illustrating the rendering process of the comments section:

- The initial HTML response contains a hidden `<div id="S:0">` containing three `<p class="comment">` elements.
- The JavaScript function `$RC(a, b)` is called, which removes the `<div id="S:0">` and inserts the rendered comments section into the document.
- The rendered comments section is shown in the final HTML output, containing the three `<p class="comment">` elements.



# Q&A