React Server Components (Alpha)

Server Components allow us to render React components on the server. This is fundamentally different from server-side rendering (SSR) where you're pre-generating HTML on the server. With Server Components, there's **zero client-side JavaScript needed,** making page rendering faster. This improves the user experience of your application, pairing the best parts of server-rendering with client-side interactivity.

Enable React Server Components

To use React Server Components, ensure you have the latest React installed:

```
npm install next@canary react@latest react-dom@latest
```

Then, update your `next.config.js`:

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

Using `runtime` also enables Streaming SSR. When setting `runtime` to `'edge'`, the server will be running entirely in the Edge Runtime.

Now, you can start using React Server Components in Next.js. See our example for more information.

Server Components Conventions

To run a component on the server, append `.server.js` to the end of the filename. For example, `./pages/home.server.js` will be treated as a Server Component.

For client components, append `.client.js` to the filename. For example, `./components/avatar.client.js`.

You can then import other server or client components from any server component. Note: a server component **can not** be imported by a client component. Components without "server/client" extensions will be treated as shared components and can be used and rendered by both sides, depending on where it is imported. For example:

```
// pages/home.server.js

import { Suspense } from 'react'

import Profile from '../components/profile.server'
import Content from '../components/content.client'

export default function Home() {
  return (
    <div>
      <h1>Welcome to React Server Components</h1>
```

```
      <Suspense fallback={'Loading...'}>
        <Profile />
      </Suspense>
      <Content />
    </div>
  )
}
```

The `<Home>` and `<Profile>` components will always be server-side rendered and streamed to the client, and will not be included by the client-side JavaScript. However, `<Content>` will still be hydrated on the client-side, like normal React components.

> *Make sure you're using default imports and exports for server components (`.server.js`). The support of named exports are a work in progress!*

To see a full example, check out the vercel/next-react-server-components demo.

Supported Next.js APIs

`next/link` and `next/image`

You can use `next/link` and `next/image` like before and they will be treated as client components to keep the interaction on client side.

`next/document`

If you have a custom `_document`, you have to change your `_document` to a functional component like below to use server components. If you don't have one, Next.js will use the default `_document` component for you.

```
// pages/_document.js
import { Html, Head, Main, NextScript } from 'next/document'

export default function Document() {
  return (
    <Html>
      <Head />
      <body>
        <Main />
        <NextScript />
      </body>
    </Html>
  )
}
```

`next/app`

If you're using `_app.js`, the usage is the same as Custom App. If you're using `_app.server.js` as a server component, see the example below where it only receives the `children` prop as React elements. You can wrap any other client or server components around `children` to customize the layout of your app.

```
// pages/_app.server.js
export default function App({ children }) {
  return children
}
```

Routing

Both basic routes with path and queries and dynamic routes are supported. If you need to access the router in server components(`.server.js`), they will receive `router` instance as a prop so that you can directly access them without using the `useRouter()` hook.

```
// pages/index.server.js

export default function Index({ router }) {
  // You can access routing information by `router.pathname`, etc.
  return 'hello'
}
```

Unsupported Next.js APIs

While RSC and SSR streaming are still in the alpha stage, not all Next.js APIs are supported. The following Next.js APIs have limited functionality within Server Components. React 18 use without SSR streaming is not affected.

React internals

Most React hooks, such as `useContext` , `useState` , `useReducer` , `useEffect` and `useLayoutEffect` , are not supported as of today since server components are executed per request and aren't stateful.

Data Fetching & Styling

Like streaming SSR, styling and data fetching within `Suspense` on the server side are not well supported. We're still working on them.

Page level exported methods like `getInitialProps` , `getStaticProps` and `getStaticPaths` are not supported.

`next/head` and I18n

We are still working on support for these features.