Upgrade Guide

Upgrading from 11 to 12

Minimum Node.js version

The minimum Node.js version has been bumped from 12.0.0 to 12.22.0 which is the first version of Node.js with native ES Modules support.

Upgrade React version to latest

To upgrade you can run the following command:

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

Or using `yarn`:

```
yarn add react@latest react-dom@latest
```

Upgrade Next.js version to 12

To upgrade you can run the following command in the terminal:

```
npm install next@12
```

or

```
yarn add next@12
```

SWC replacing Babel

Next.js now uses Rust-based compiler SWC to compile JavaScript/TypeScript. This new compiler is up to 17x faster than Babel when compiling individual files and up to 5x faster Fast Refresh.

Next.js provides full backwards compatibility with applications that have custom Babel configuration. All transformations that Next.js handles by default like styled-jsx and tree-shaking of `getStaticProps` / `getStaticPaths` / `getServerSideProps` have been ported to Rust.

When an application has a custom Babel configuration, Next.js will automatically opt-out of using SWC for compiling JavaScript/Typescript and will fall back to using Babel in the same way that it was used in Next.js 11.

Many of the integrations with external libraries that currently require custom Babel transformations will be ported to Rust-based SWC transforms in the near future. These include but are not limited to:

- Styled Components
- Emotion
- Relay

In order to prioritize transforms that will help you adopt SWC, please provide your `.babelrc` on the feedback thread.

SWC replacing Terser for minification

You can opt-in to replacing Terser with SWC for minifying JavaScript up to 7x faster using a flag in `next.config.js`:

```
module.exports = {
  swcMinify: true,
}
```

Minification using SWC is an opt-in flag to ensure it can be tested against more real-world Next.js applications before it becomes the default in Next.js 12.1. If you have feedback about minification, please leave it on the feedback thread.

Improvements to styled-jsx CSS parsing

On top of the Rust-based compiler we've implemented a new CSS parser based on the CSS parser that was used for the styled-jsx Babel transform. This new parser has improved handling of CSS and now errors when invalid CSS is used that would previously slip through and cause unexpected behavior.

Because of this change invalid CSS will throw an error during development and `next build`. This change only affects styled-jsx usage.

`next/image` changed wrapping element

`next/image` now renders the `<img>` inside a `<span>` instead of `<div>`.

If your application has specific CSS targeting span, for example `.container span`, upgrading to Next.js 12 might incorrectly match the wrapping element inside the `<Image>` component. You can avoid this by restricting the selector to a specific class such as `.container span.item` and updating the relevant component with that className, such as `<span className="item" />`.

If your application has specific CSS targeting the `next/image` `<div>` tag, for example `.container div`, it may not match anymore. You can update the selector `.container span`, or preferably, add a new `<div className="wrapper">` wrapping the `<Image>` component and target that instead such as `.container .wrapper`.

The `className` prop is unchanged and will still be passed to the underlying `<img>` element.

See the documentation for more info.

Next.js' HMR connection now uses a WebSocket

Previously, Next.js used a server-sent events connection to receive HMR events. Next.js 12 now uses a WebSocket connection.

In some cases when proxying requests to the Next.js dev server, you will need to ensure the upgrade request is handled correctly. For example, in `nginx` you would need to add the following configuration:

```
location /_next/webpack-hmr {
    proxy_pass http://localhost:3000/_next/webpack-hmr;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
```

For custom servers, such as `express`, you may need to use `app.all` to ensure the request is passed correctly, for example:

```
app.all('/_next/webpack-hmr', (req, res) => {
  nextjsRequestHandler(req, res)
})
```

Webpack 4 support has been removed

If you are already using webpack 5 you can skip this section.

Next.js has adopted webpack 5 as the default for compilation in Next.js 11. As communicated in the webpack 5 upgrading documentation Next.js 12 removes support for webpack 4.

If your application is still using webpack 4 using the opt-out flag you will now see an error linking to the webpack 5 upgrading documentation.

`target` option deprecated

If you do not have `target` in `next.config.js` you can skip this section.

The target option has been deprecated in favor of built-in support for tracing what dependencies are needed to run a page.

During `next build`, Next.js will automatically trace each page and its dependencies to determine all of the files that are needed for deploying a production version of your application.

If you are currently using the `target` option set to `serverless` please read the documentation on how to leverage the new output.

Upgrading from version 10 to 11

Upgrade React version to latest

Most applications already use the latest version of React, with Next.js 11 the minimum React version has been updated to 17.0.2.

To upgrade you can run the following command:

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

Or using `yarn`:

```
yarn add react@latest react-dom@latest
```

Upgrade Next.js version to 11

To upgrade you can run the following command in the terminal:

```
npm install next@11
```

or

```
yarn add next@11
```

Webpack 5

Webpack 5 is now the default for all Next.js applications. If you did not have custom webpack configuration your application is already using webpack 5. If you do have custom webpack configuration you can refer to the Next.js webpack 5 documentation for upgrading guidance.

Cleaning the `distDir` is now a default

The build output directory (defaults to `.next`) is now cleared by default except for the Next.js caches. You can refer to the cleaning `distDir` RFC for more information.

If your application was relying on this behavior previously you can disable the new default behavior by adding the `cleanDistDir: false` flag in `next.config.js`.

`PORT` is now supported for `next dev` and `next start`

Next.js 11 supports the `PORT` environment variable to set the port the application has to run on. Using `-p` / `--port` is still recommended but if you were prohibited from using `-p` in any way you can now use `PORT` as an alternative:

Example:

```
PORT=4000 next start
```

`next.config.js` customization to import images

Next.js 11 supports static image imports with `next/image`. This new feature relies on being able to process image imports. If you previously added the `next-images` or `next-optimized-images` packages you can either move to the new built-in support using `next/image` or disable the feature:

```
module.exports = {
  images: {
    disableStaticImages: true,
  },
}
```

Remove `super.componentDidCatch()` from `pages/_app.js`

The `next/app` component's `componentDidCatch` has been deprecated since Next.js 9 as it's no longer needed and has since been a no-op, in Next.js 11 it has been removed.

If your `pages/_app.js` has a custom `componentDidCatch` method you can remove `super.componentDidCatch` as it is no longer needed.

Remove `Container` from `pages/_app.js`

This export has been deprecated since Next.js 9 as it's no longer needed and has since been a no-op with a warning during development. In Next.js 11 it has been removed.

If your `pages/_app.js` imports `Container` from `next/app` you can remove `Container` as it has been removed. Learn more in the documentation.

Remove `props.url` usage from page components

This property has been deprecated since Next.js 4 and has since shown a warning during development. With the introduction of `getStaticProps` / `getServerSideProps` these methods already disallowed usage of `props.url`. In Next.js 11 it has been removed completely.

You can learn more in the documentation.

Remove `unsized` property on `next/image`

The `unsized` property on `next/image` was deprecated in Next.js 10.0.1. You can use `layout="fill"` instead. In Next.js 11 `unsized` was removed.

Remove `modules` property on `next/dynamic`

The `modules` and `render` option for `next/dynamic` have been deprecated since Next.js 9.5 showing a warning that it has been deprecated. This was done in order to make `next/dynamic` close to `React.lazy` in API surface. In Next.js 11 the `modules` and `render` options have been removed.

This option hasn't been mentioned in the documentation since Next.js 8 so it's less likely that your application is using it.

If your application does use `modules` and `render` you can refer to the documentation.

Remove `Head.rewind`

`Head.rewind` has been a no-op since Next.js 9.5, in Next.js 11 it was removed. You can safely remove your usage of `Head.rewind`.

Moment.js locales excluded by default

Moment.js includes translations for a lot of locales by default. Next.js now automatically excludes these locales by default to optimize bundle size for applications using Moment.js.

To load a specific locale use this snippet:

```
import moment from 'moment'
import 'moment/locale/ja'

moment.locale('ja')
```

You can opt-out of this new default by adding `excludeDefaultMomentLocales: false` to `next.config.js` if you do not want the new behavior, do note it's highly recommended to not disable this new optimization as it significantly reduces the size of Moment.js.

Update usage of `router.events`

In case you're accessing `router.events` during rendering, in Next.js 11 `router.events` is no longer provided during pre-rendering. Ensure you're accessing `router.events` in `useEffect`:

```
useEffect(() => {
  const handleRouteChange = (url, { shallow }) => {
    console.log(
      `App is changing to ${url} ${
        shallow ? 'with' : 'without'
      } shallow routing`
    )
  }

  router.events.on('routeChangeStart', handleRouteChange)

  // If the component is unmounted, unsubscribe
  // from the event with the `off` method:
  return () => {
    router.events.off('routeChangeStart', handleRouteChange)
  }
}, [router])
```

If your application uses `router.router.events` which was an internal property that was not public please make sure to use `router.events` as well.

React 16 to 17

React 17 introduced a new JSX Transform that brings a long-time Next.js feature to the wider React ecosystem: Not having to `import React from 'react'` when using JSX. When using React 17 Next.js will automatically use the new transform. This transform does not make the `React` variable global, which was an unintended side-effect of the previous Next.js implementation. A codemod is available to automatically fix cases where you accidentally used `React` without importing it.

Upgrading from version 9 to 10

There were no breaking changes between version 9 and 10.

To upgrade run the following command:

```
npm install next@10
```

Or using `yarn`:

```
yarn add next@10
```

Upgrading from version 8 to 9

Preamble

Production Deployment on Vercel

If you previously configured `routes` in your `vercel.json` file for dynamic routes, these rules can be removed when leveraging Next.js 9's new Dynamic Routing feature.

Next.js 9's dynamic routes are **automatically configured on Vercel** and do not require any `vercel.json` customization.

You can read more about Dynamic Routing here.

Check your Custom (`pages/_app.js`)

If you previously copied the Custom `<App>` example, you may be able to remove your `getInitialProps`.

Removing `getInitialProps` from `pages/_app.js` (when possible) is important to leverage new Next.js features!

The following `getInitialProps` does nothing and may be removed:

```
class MyApp extends App {
  // Remove me, I do nothing!
  static async getInitialProps({ Component, ctx }) {
    let pageProps = {}

    if (Component.getInitialProps) {
      pageProps = await Component.getInitialProps(ctx)
    }

    return { pageProps }
  }

  render() {
    // ... etc
  }
}
```

Breaking Changes

`@zeit/next-typescript` is no longer necessary

Next.js will now ignore usage `@zeit/next-typescript` and warn you to remove it. Please remove this plugin from your `next.config.js`.

Remove references to `@zeit/next-typescript/babel` from your custom `.babelrc` (if present).

Usage of `fork-ts-checker-webpack-plugin` should also be removed from your `next.config.js`.

TypeScript Definitions are published with the `next` package, so you need to uninstall `@types/next` as they would conflict.

The following types are different:

> *This list was created by the community to help you upgrade, if you find other differences please send a pull-request to this list to help other users.*

From:

```
import { NextContext } from 'next'
import { NextAppContext, DefaultAppIProps } from 'next/app'
```

```
import { NextDocumentContext, DefaultDocumentIProps } from 'next/document'
```

to

```
import { NextPageContext } from 'next'
import { AppContext, AppInitialProps } from 'next/app'
import { DocumentContext, DocumentInitialProps } from 'next/document'
```

The `config` key is now an export on a page

You may no longer export a custom variable named `config` from a page (i.e. `export { config }` / `export const config ...`). This exported variable is now used to specify page-level Next.js configuration like Opt-in AMP and API Route features.

You must rename a non-Next.js-purposed `config` export to something different.

`next/dynamic` no longer renders "loading..." by default while loading

Dynamic components will not render anything by default while loading. You can still customize this behavior by setting the `loading` property:

```
import dynamic from 'next/dynamic'

const DynamicComponentWithCustomLoading = dynamic(
  () => import('../components/hello2'),
  {
    loading: () => <p>Loading</p>,
  }
)
```

`withAmp` has been removed in favor of an exported configuration object

Next.js now has the concept of page-level configuration, so the `withAmp` higher-order component has been removed for consistency.

This change can be **automatically migrated by running the following commands in the root of your Next.js project:**

```
curl -L https://github.com/vercel/next-codemod/archive/master.tar.gz | tar -xz --strip=2 next-codemod-master/transforms/withamp-to-config.js npx jscodeshift -t ./withamp-to-config.js pages/**/*.js
```

To perform this migration by hand, or view what the codemod will produce, see below:

Before

```
import { withAmp } from 'next/amp'

function Home() {
  return <h1>My AMP Page</h1>
}
```

```
export default withAmp(Home)
// or
export default withAmp(Home, { hybrid: true })
```

After

```
export default function Home() {
  return <h1>My AMP Page</h1>
}

export const config = {
  amp: true,
  // or
  amp: 'hybrid',
}
```

`next export` no longer exports pages as `index.html`

Previously, exporting `pages/about.js` would result in `out/about/index.html`. This behavior has been changed to result in `out/about.html`.

You can revert to the previous behavior by creating a `next.config.js` with the following content:

```
// next.config.js
module.exports = {
  trailingSlash: true,
}
```

`./pages/api/` is treated differently

Pages in `./pages/api/` are now considered API Routes. Pages in this directory will no longer contain a client-side bundle.

Deprecated Features

`next/dynamic` has deprecated loading multiple modules at once

The ability to load multiple modules at once has been deprecated in `next/dynamic` to be closer to React's implementation (`React.lazy` and `Suspense`).

Updating code that relies on this behavior is relatively straightforward! We've provided an example of a before/after to help you migrate your application:

Before

```
import dynamic from 'next/dynamic'

const HelloBundle = dynamic({
  modules: () => {
    const components = {
```

```
      Hello1: () => import('../components/hello1').then((m) => m.default),
      Hello2: () => import('../components/hello2').then((m) => m.default),
    }

    return components
  },
  render: (props, { Hello1, Hello2 }) => (
    <div>
      <h1>{props.title}</h1>
      <Hello1 />
      <Hello2 />
    </div>
  ),
})

function DynamicBundle() {
  return <HelloBundle title="Dynamic Bundle" />
}

export default DynamicBundle
```

After

```
import dynamic from 'next/dynamic'

const Hello1 = dynamic(() => import('../components/hello1'))
const Hello2 = dynamic(() => import('../components/hello2'))

function HelloBundle({ title }) {
  return (
    <div>
      <h1>{title}</h1>
      <Hello1 />
      <Hello2 />
    </div>
  )
}

function DynamicBundle() {
  return <HelloBundle title="Dynamic Bundle" />
}

export default DynamicBundle
```