Next.js Compiler

Version History

Version	Changes
`v12.1.0`	Added support for Styled Components, Jest, Relay, Remove React Properties, Legacy Decorators, Remove Console, and jsxImportSource.
`v12.0.0`	Next.js Compiler introduced.

The Next.js Compiler, written in Rust using SWC, allows Next.js to transform and minify your JavaScript code for production. This replaces Babel for individual files and Terser for minifying output bundles.

Compilation using the Next.js Compiler is 17x faster than Babel and enabled by default since Next.js version 12. If you have an existing Babel configuration or are using unsupported features, your application will opt-out of the Next.js Compiler and continue using Babel.

Why SWC?

SWC is an extensible Rust-based platform for the next generation of fast developer tools.

SWC can be used for compilation, minification, bundling, and more – and is designed to be extended. It's something you can call to perform code transformations (either built-in or custom). Running those transformations happens through higher-level tools like Next.js.

We chose to build on SWC for a few reasons:

- **Extensibility:** SWC can be used as a Crate inside Next.js, without having to fork the library or workaround design constraints.
- **Performance:** We were able to achieve ~3x faster Fast Refresh and ~5x faster builds in Next.js by switching to SWC, with more room for optimization still in progress.
- **WebAssembly:** Rust's support for WASM is essential for supporting all possible platforms and taking Next.js development everywhere.
- **Community:** The Rust community and ecosystem are amazing and still growing.

Supported Features

Styled Components

We're working to port `babel-plugin-styled-components` to the Next.js Compiler.

First, update to the latest version of Next.js: `npm install next@latest`. Then, update your `next.config.js` file:

```
// next.config.js

module.exports = {
  compiler: {
    // ssr and displayName are configured by default
    styledComponents: true,
  },
}
```

Currently, only the `ssr` and `displayName` transforms have been implemented. These two transforms are the main requirement for using `styled-components` in Next.js.

Jest

Jest support not only includes the transformation previously provided by Babel, but also simplifies configuring Jest together with Next.js including:

- Auto mocking of `.css`, `.module.css` (and their `.scss` variants), and image imports
- Automatically sets up `transform` using SWC
- Loading `.env` (and all variants) into `process.env`
- Ignores `node_modules` from test resolving and transforms
- Ignoring `.next` from test resolving
- Loads `next.config.js` for flags that enable experimental SWC transforms

First, update to the latest version of Next.js: `npm install next@latest`. Then, update your `jest.config.js` file:

```
// jest.config.js
const nextJest = require('next/jest')

// Providing the path to your Next.js app which will enable loading next.config.js and .env
const createJestConfig = nextJest({ dir })

// Any custom config you want to pass to Jest
const customJestConfig = {
  setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
}

// createJestConfig is exported in this way to ensure that next/jest can load the Next.js co
module.exports = createJestConfig(customJestConfig)
```

Relay

To enable Relay support:

```
// next.config.js
module.exports = {
  compiler: {
    relay: {
      // This should match relay.config.js
      src: './',
      artifactDirectory: './__generated__',
      language: 'typescript',
    },
  },
}
```

NOTE: In Next.js all JavaScript files in `pages` directory are considered routes. So, for `relay-compiler` you'll need to specify `artifactDirectory` configuration settings outside of the `pages`, otherwise `relay-compiler` will generate files next to the source file in the `__generated__` directory, and this file will be considered a route, which will break production builds.

Remove React Properties

Allows to remove JSX properties. This is often used for testing. Similar to `babel-plugin-react-remove-properties`.

To remove properties matching the default regex `^data-test`:

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

To remove custom properties:

```
// next.config.js
module.exports = {
  compiler: {
    // The regexes defined here are processed in Rust so the syntax is different from
    // JavaScript `RegExp`s. See https://docs.rs/regex.
    reactRemoveProperties: { properties: ['^data-custom$'] },
  },
}
```

Remove Console

This transform allows for removing all `console.*` calls in application code (not `node_modules`). Similar to `babel-plugin-transform-remove-console`.

Remove all `console.*` calls:

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

Remove `console.*` output except `console.error`:

```
// next.config.js
module.exports = {
  compiler: {
    removeConsole: {
      exclude: ['error'],
    },
  },
}
```

Legacy Decorators

Next.js will automatically detect `experimentalDecorators` in `jsconfig.json` or `tsconfig.json`. Legacy decorators are commonly used with older versions of libraries like `mobx`.

This flag is only supported for compatibility with existing applications. We do not recommend using legacy decorators in new applications.

First, update to the latest version of Next.js: `npm install next@latest`. Then, update your `jsconfig.json` or `tsconfig.json` file:

```
{
  "compilerOptions": {
    "experimentalDecorators": true
  }
}
```

importSource

Next.js will automatically detect `jsxImportSource` in `jsconfig.json` or `tsconfig.json` and apply that. This is commonly used with libraries like Theme UI.

First, update to the latest version of Next.js: `npm install next@latest`. Then, update your `jsconfig.json` or `tsconfig.json` file:

```
{
  "compilerOptions": {
    "jsxImportSource": 'preact'
  }
}
```

Experimental Features

Emotion

We're working to port `@emotion/babel-plugin` to the Next.js Compiler.

First, update to the latest version of Next.js: `npm install next@latest`. Then, update your `next.config.js` file:

```
// next.config.js

module.exports = {
  experimental: {
    emotion: boolean | {
      // default is true. It will be disabled when build type is production.
      sourceMap?: boolean,
      // default is 'dev-only'.
      autoLabel?: 'never' | 'dev-only' | 'always',
      // default is '[local]'.
      // Allowed values: `[local]` `[filename]` and `[dirname]`
      // This option only works when autoLabel is set to 'dev-only' or 'always'.
      // It allows you to define the format of the resulting label.
      // The format is defined via string where variable parts are enclosed in square brack
      // For example labelFormat: "my-classname--[local]", where [local] will be replaced w
      labelFormat?: string,
    },
  },
}
```

Only `importMap` in `@emotion/babel-plugin` is not supported for now.

Minification

You can opt-in to using the Next.js compiler for minification. This is 7x faster than Terser.

```
// next.config.js

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

If you have feedback about `swcMinify`, please share it on the feedback discussion.

Modularize Imports

Allows to modularize imports, similar to babel-plugin-transform-imports.

Transforms member style imports:

```
import { Row, Grid as MyGrid } from 'react-bootstrap'
import { merge } from 'lodash'
```

. . . into default style imports:

```
import Row from 'react-bootstrap/lib/Row'
import MyGrid from 'react-bootstrap/lib/Grid'
import merge from 'lodash/merge'
```

Config for the above transform:

```
// next.config.js
module.exports = {
  experimental: {
    modularizeImports: {
      'react-bootstrap': {
        transform: 'react-bootstrap/lib/{{member}}',
      },
      lodash: {
        transform: 'lodash/{{member}}',
      },
    },
  },
}
```

Advanced transformations:

- Using regular expressions

Similar to `babel-plugin-transform-imports`, but the transform is templated with handlebars and regular expressions are in Rust regex crate's syntax.

The config:

```
// next.config.js
module.exports = {
  experimental: {
    modularizeImports: {
      'my-library/?((((\\w*)?/?)*)': {
        transform: 'my-library/{{ matches.[1] }}/{{member}}',
      },
    },
  },
}
```

Cause this code:

```
import { MyModule } from 'my-library'
import { App } from 'my-library/components'
import { Header, Footer } from 'my-library/components/App'
```

To become:

```
import MyModule from 'my-library/MyModule'
import App from 'my-library/components/App'
import Header from 'my-library/components/App/Header'
import Footer from 'my-library/components/App/Footer'
```

- Handlebars templating

This transform uses handlebars to templating the replacement import path in the `transform` field. These variables and helper functions are available:

1. `matches`: Has type `string[]`. All groups matched by the regular expression. `matches.[0]` is the full match.
2. `member`: Has type `string`. The name of the member import.
3. `lowerCase`, `upperCase`, `camelCase`: Helper functions to convert a string to lower, upper or camel cases.

Unsupported Features

When your application has a `.babelrc` file, Next.js will automatically fall back to using Babel for transforming individual files. This ensures backwards compatibility with existing applications that leverage custom Babel plugins.

If you're using a custom Babel setup, please share your configuration. We're working to port as many commonly used Babel transformations as possible, as well as supporting plugins in the future.