ON THIS PAGE

- What does the compiler do?
- What does the compiler assume?
- Should I try out the compiler?

Getting Started

- Checking compatibility
- Installing eslint-plugin-react-compiler
- Rolling out the compiler to your codebase

Usage

- Babel
- Vite
- Next.js
- Remix
- Webpack
- Expo
- Metro (React Native)

Troubleshooting

- `(0 , _c) is not a function` error
- How do I know my components have been optimized?
- Something is not working after compilation
- Other issues

React Compiler

This page will give you an introduction to the new experimental React Compiler and how to try it out successfully.

Under Construction

These docs are still a work in progress. More documentation is available in the React Compiler Working Group repo, and will be upstreamed into these docs when they are more stable.

You will learn

- Getting started with the compiler
- Installing the compiler and eslint plugin
- Troubleshooting

Note

React Compiler is a new experimental compiler that we've open sourced to get early feedback from the community. It still has rough edges and is not yet fully ready for production.

React Compiler requires React 19 RC. If you are unable to upgrade to React 19, you may try a userspace implementation of the cache function as described in the Working Group. However, please note that this is not recommended and you should upgrade to React 19 when possible.

React Compiler is a new experimental compiler that we've open sourced to get early feedback from the community. It is a build-time only tool that automatically optimizes your React app. It works with plain JavaScript, and understands the Rules of React, so you don't need to rewrite any code to use it.

The compiler also includes an eslint plugin that surfaces the analysis from the compiler right in your editor. The plugin runs independently of the compiler and can be used even if you aren't using the compiler in your app. We recommend all React developers to use this eslint plugin to help improve the quality of your codebase.

What does the compiler do?

In order to optimize applications, React Compiler automatically memoizes your code. You may be familiar today with memoization through APIs such as `useMemo`, `useCallback`, and `React.memo`. With these APIs you can tell React that certain parts of your application don't need to recompute if their inputs haven't changed, reducing work on updates. While powerful, it's easy to forget to apply memoization or apply them incorrectly. This can lead to inefficient updates as React has to check parts of your UI that don't have any *meaningful* changes.

The compiler uses its knowledge of JavaScript and React's rules to automatically memoize values or groups of values within your components and hooks. If it detects breakages of the rules, it will automatically skip over just those components or hooks, and continue safely compiling other code.

If your codebase is already very well-memoized, you might not expect to see major performance improvements with the compiler. However, in practice memoizing the correct dependencies that cause performance issues is tricky to get right by hand.

DEEP DIVE

What kind of memoization does React Compiler add?

Show Details

What does the compiler assume?

React Compiler assumes that your code:

1. Is valid, semantic JavaScript
2. Tests that nullable/optional values and properties are defined before accessing them (for example, by enabling `strictNullChecks` if using TypeScript), i.e., `if (object.nullableProperty) { object.nullableProperty.foo }` or with optional-chaining `object.nullableProperty?.foo`
3. Follows the Rules of React

React Compiler can verify many of the Rules of React statically, and will safely skip compilation when it detects an error. To see the errors we recommend also installing eslint-plugin-react-compiler.

Should I try out the compiler?

Please note that the compiler is still experimental and has many rough edges. While it has been used in production at companies like Meta, rolling out the compiler to production for your app will depend on the health of your codebase and how well you've followed the Rules of React.

You don't have to rush into using the compiler now. It's okay to wait until it reaches a stable release before adopting it. However, we do appreciate trying it out in small experiments in your apps so that you can provide feedback to us to help make the compiler better.

Getting Started

In addition to these docs, we recommend checking the React Compiler Working Group for additional information and discussion about the compiler.

Checking compatibility

Prior to installing the compiler, you can first check to see if your codebase is compatible:

Terminal

```
npx react-compiler-healthcheck@latest
```

This script will:

- Check how many components can be successfully optimized: higher is better
- Check for `<StrictMode>` usage: having this enabled and followed means a higher chance that the Rules of React are followed
- Check for incompatible library usage: known libraries that are incompatible with the compiler

As an example:

Terminal Copy

```
Successfully compiled 8 out of 9 components.
StrictMode usage not found.
Found no usage of incompatible libraries.
```

Installing eslint-plugin-react-compiler

React Compiler also powers an eslint plugin. The eslint plugin can be used **independently** of the compiler, meaning you can use the eslint plugin even if you don't use the compiler.

Terminal Copy

```
npm install eslint-plugin-react-compiler
```

Then, add it to your eslint config:

```
module.exports = {
  plugins: [
    'eslint-plugin-react-compiler',
  ],
  rules: {
    'react-compiler/react-compiler': "error",
  },
}
```

The eslint plugin will display any violations of the rules of React in your editor. When it does this, it means that the compiler has skipped over optimizing that component or hook. This is perfectly okay, and the compiler can recover and continue optimizing other components in your codebase.

You don't have to fix all eslint violations straight away. You can address them at your own pace to increase the amount of components and hooks being optimized, but it is not required to fix everything before you can use the compiler.

Rolling out the compiler to your codebase

Existing projects

The compiler is designed to compile functional components and hooks that follow the Rules of React. It can also handle code that breaks those rules by bailing out (skipping over) those components or hooks. However, due to the flexible nature of JavaScript, the compiler cannot catch every possible violation and may compile with false negatives: that is, the compiler may accidentally compile a component/hook that breaks the Rules of React which can lead to undefined behavior.

For this reason, to adopt the compiler successfully on existing projects, we recommend running it on a small directory in your product code first. You can do this by configuring the compiler to only run on a specific set of directories:

```
const ReactCompilerConfig = {
  sources: (filename) => {
    return filename.indexOf('src/path/to/dir') !== -1;
  },
};
```

In rare cases, you can also configure the compiler to run in "opt-in" mode using the `compilationMode: "annotation"` option. This makes it so the compiler will only compile components and hooks annotated with a `"use memo"` directive. Please note that the `annotation` mode is a temporary one to aid early adopters, and that we don't intend for the `"use memo"` directive to be used for the long term.

```
const ReactCompilerConfig = {
  compilationMode: "annotation",
};

// src/app.jsx
export default function App() {
  "use memo";
  // ...
}
```

When you have more confidence with rolling out the compiler, you can expand coverage to other directories as well and slowly roll it out to your whole app.

New projects

If you're starting a new project, you can enable the compiler on your entire codebase, which is the default behavior.

Usage

Babel

Terminal Copy

```
npm install babel-plugin-react-compiler
```

The compiler includes a Babel plugin which you can use in your build pipeline to run the compiler.

After installing, add it to your Babel config. Please note that it's critical that the compiler run **first** in the pipeline:

```
// babel.config.js
const ReactCompilerConfig = { /* ... */ };

module.exports = function () {
  return {
    plugins: [
      ['babel-plugin-react-compiler', ReactCompilerConfig], // must run first!
      // ...
    ],
  };
};
```

`babel-plugin-react-compiler` should run first before other Babel plugins as the compiler requires the input source information for sound analysis.

Vite

If you use Vite, you can add the plugin to vite-plugin-react:

```
// vite.config.js
const ReactCompilerConfig = { /* ... */ };

export default defineConfig(() => {
  return {
    plugins: [
      react({
        babel: {
          plugins: [
            ["babel-plugin-react-compiler", ReactCompilerConfig],
          ],
        },
      }),
    ],
    // ...
  };
});
```

Next.js

Next.js has an experimental configuration to enable the React Compiler. It automatically ensures Babel is set up with `babel-plugin-react-compiler`.

- Install Next.js canary, which uses React 19 Release Candidate
- Install `babel-plugin-react-compiler`

Terminal

```
npm install next@canary babel-plugin-react-compiler
```

Then configure the experimental option in `next.config.js`:

```js
// next.config.js
/** @type {import('next').NextConfig} */
const nextConfig = {
  experimental: {
    reactCompiler: true,
  },
};

module.exports = nextConfig;
```

Using the experimental option ensures support for the React Compiler in:

- App Router
- Pages Router
- Webpack (default)
- Turbopack (opt-in through `--turbo`)

Remix

Install `vite-plugin-babel`, and add the compiler's Babel plugin to it:

Terminal

```
npm install vite-plugin-babel
```

```
// vite.config.js
import babel from "vite-plugin-babel";

const ReactCompilerConfig = { /* ... */ };

export default defineConfig({
  plugins: [
    remix({ /* ... */}),
    babel({
      filter: /\.[jt]sx?$/,
      babelConfig: {
        presets: ["@babel/preset-typescript"], // if you use TypeScript
        plugins: [
          ["babel-plugin-react-compiler", ReactCompilerConfig],
        ],
      },
    }),
  ],
});
```

Webpack

You can create your own loader for React Compiler, like so:

```
const ReactCompilerConfig = { /* ... */ };
const BabelPluginReactCompiler = require('babel-plugin-react-compiler');

function reactCompilerLoader(sourceCode, sourceMap) {
  // ...
  const result = transformSync(sourceCode, {
    // ...
    plugins: [
      [BabelPluginReactCompiler, ReactCompilerConfig],
    ],
  // ...
  });

  if (result === null) {
    this.callback(
      Error(
        `Failed to transform "${options.filename}"`
      )
    );
    return;
  }

  this.callback(
    null,
    result.code
    result.map === null ? undefined : result.map
  );
}

module.exports = reactCompilerLoader;
```

Expo

Expo uses Babel via Metro, so refer to the Usage with Babel section for installation instructions.

Metro (React Native)

React Native uses Babel via Metro, so refer to the Usage with Babel section for installation instructions.

Troubleshooting

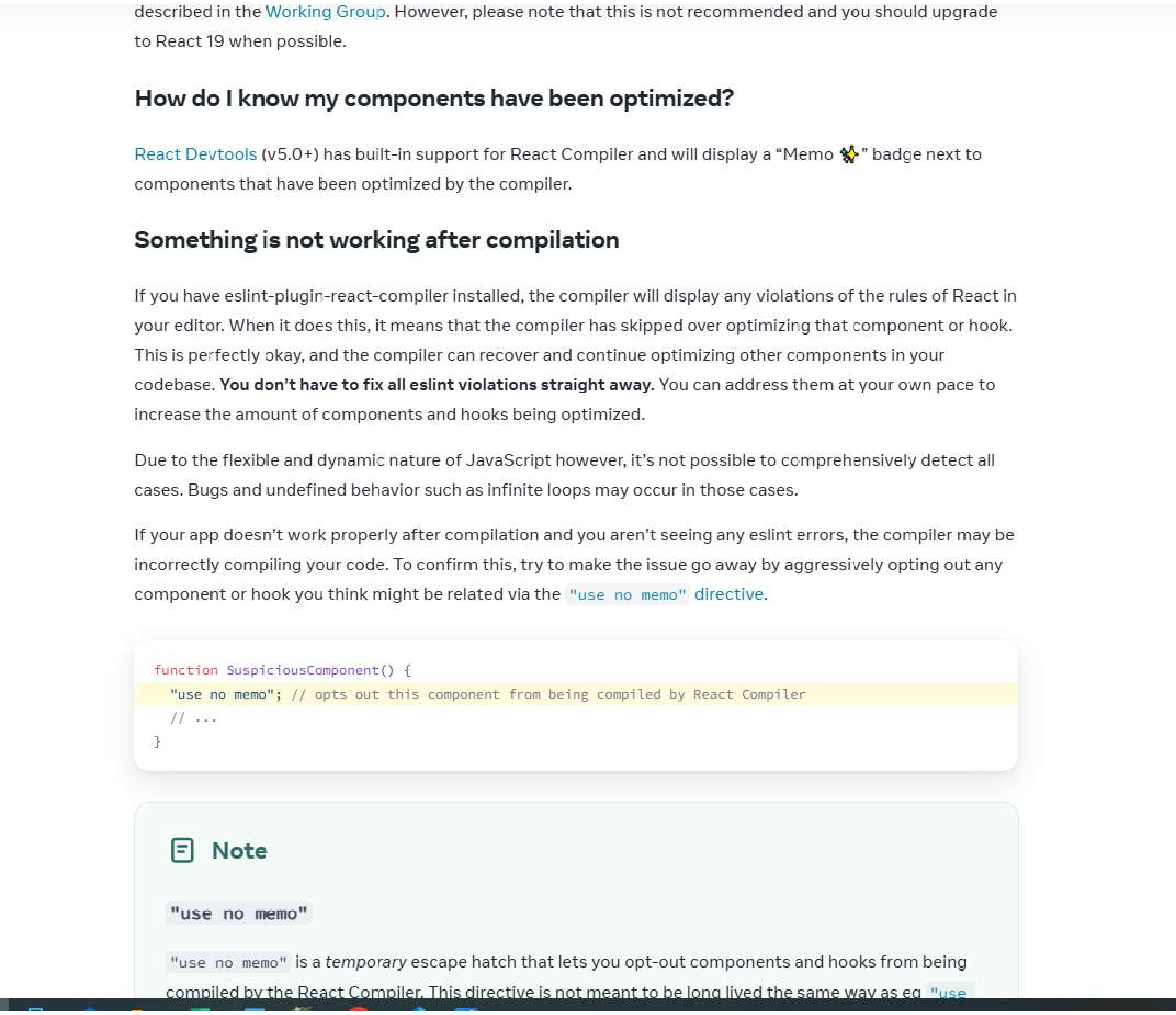
To report issues, please first create a minimal repro on the React Compiler Playground and include it in your bug report. You can open issues in the facebook/react repo.

You can also provide feedback in the React Compiler Working Group by applying to be a member. Please see the README for more details on joining.

`(0 , _c) is not a function` error

This occurs if you are not using React 19 RC and up. To fix this, upgrade your app to React 19 RC first.

If you are unable to upgrade to React 19, you may try a userspace implementation of the cache function as



described in the Working Group. However, please note that this is not recommended and you should upgrade to React 19 when possible.

How do I know my components have been optimized?

React Devtools (v5.0+) has built-in support for React Compiler and will display a “Memo ✨” badge next to components that have been optimized by the compiler.

Something is not working after compilation

If you have eslint-plugin-react-compiler installed, the compiler will display any violations of the rules of React in your editor. When it does this, it means that the compiler has skipped over optimizing that component or hook. This is perfectly okay, and the compiler can recover and continue optimizing other components in your codebase. **You don’t have to fix all eslint violations straight away.** You can address them at your own pace to increase the amount of components and hooks being optimized.

Due to the flexible and dynamic nature of JavaScript however, it’s not possible to comprehensively detect all cases. Bugs and undefined behavior such as infinite loops may occur in those cases.

If your app doesn’t work properly after compilation and you aren’t seeing any eslint errors, the compiler may be incorrectly compiling your code. To confirm this, try to make the issue go away by aggressively opting out any component or hook you think might be related via the `"use no memo"` directive.

```
function SuspiciousComponent() {
  "use no memo"; // opts out this component from being compiled by React Compiler
  // ...
}
```

Note

`"use no memo"`

`"use no memo"` is a *temporary* escape hatch that lets you opt-out components and hooks from being compiled by the React Compiler. This directive is not meant to be long lived the same way as eg `"use`

Note

`"use no memo"`

`"use no memo"` is a *temporary* escape hatch that lets you opt-out components and hooks from being compiled by the React Compiler. This directive is not meant to be long lived the same way as eg `"use client"` is.

It is not recommended to reach for this directive unless it's strictly necessary. Once you opt-out a component or hook, it is opted-out forever until the directive is removed. This means that even if you fix the code, the compiler will still skip over compiling it unless you remove the directive.

When you make the error go away, confirm that removing the opt out directive makes the issue come back. Then share a bug report with us (you can try to reduce it to a small repro, or if it's open source code you can also just paste the entire source) using the React Compiler Playground so we can identify and help fix the issue.

Other issues

Please see https://github.com/reactwg/react-compiler/discussions/7.