**Webpack 4 Tutorial: from 0 Conf to Production Mode**

**webpack 4** is out!

The popular module bundler gets a massive update.

webpack 4, what’s **new**? A massive **performance improvement**, zero configuration and sane**defaults**.



This is a living, breathing **introduction to webpack 4**. Constantly updated.

You’ll build a **working webpack 4 environment** by **following each section in order**. But feel free to jump over the tutorial!

Table of Contents

* [webpack 4 as a zero configuration module bundler](https://www.valentinog.com/blog/webpack-4-tutorial/#webpack_4_as_a_zero_configuration_module_bundler)
* [webpack 4: getting started with zero conf](https://www.valentinog.com/blog/webpack-4-tutorial/#webpack_4_getting_started_with_zero_conf)
* [webpack 4: production and development mode](https://www.valentinog.com/blog/webpack-4-tutorial/#webpack_4_production_and_development_mode)
* [webpack 4: overriding the defaults entry/output](https://www.valentinog.com/blog/webpack-4-tutorial/#webpack_4_overriding_the_defaults_entryoutput)
* [webpack 4: transpiling Javascript ES6 with Babel](https://www.valentinog.com/blog/webpack-4-tutorial/#webpack_4_transpiling_Javascript_ES6_with_Babel)
* [webpack 4: setting up webpack 4 with React](https://www.valentinog.com/blog/webpack-4-tutorial/#webpack_4_setting_up_webpack_4_with_React)
* [webpack 4: the HTML webpack plugin](https://www.valentinog.com/blog/webpack-4-tutorial/#webpack_4_the_HTML_webpack_plugin)
* [webpack 4 : the webpack dev server](https://www.valentinog.com/blog/webpack-4-tutorial/#webpack_4_the_webpack_dev_server)
* [webpack 4: resources](https://www.valentinog.com/blog/webpack-4-tutorial/#webpack_4_resources)

**webpack 4 as a zero configuration module bundler**

webpack is powerful and has a lot of unique features but one of its pain point is the**configuration file**.

Providing a configuration for webpack is not a big deal in medium to bigger projects. You can’t live without one. Yet, for smaller ones it’s kind of annoying, especially when you want to kickstart some toy app.

That’s why [Parcel gained a lot of traction](https://www.valentinog.com/blog/tutorial-react-parcel-bundler/).

Here’s the breaking news now: **webpack 4 doesn’t need a configuration file by default**!

Let’s try that out.

**webpack 4: getting started with zero conf**

* Create a new directory and move into it:

>mkdir webpack-4-quickstart && cd $\_

* Initialize a package.json by running:

>npm init -y

* and pull webpack 4 in:

>npm i webpack --save-dev

* We need **webpack-cli** also, which lives as a separate package:

>npm i webpack-cli --save-dev

* Now open up [package.json](https://docs.npmjs.com/files/package.json" \t "_blank) and add a build script:

1. **"scripts"**: {
2. **"build"**: "webpack"
3. }

Close the file and save it.

Try to run:

>npm run build

and see what happens:

1. ERROR **in** Entry module not found: Error: Can't resolve './src' in '~/webpack-4-quickstart'

webpack 4 is looking for an entry point in **./src**! If you don’t know what that means go check out my previous article on [switching from Gulp to webpack](https://www.valentinog.com/blog/from-gulp-to-webpack-quickstart/).

In brief: the **entry point** is the file webpack looks for to start building your Javascript bundle.

In the previous version of webpack the entry point has to be defined inside a configuration file named webpack.config.js.

But starting from **webpack 4 there is no need to define the entry point**: it will take**./src/index.js** as the default!

Testing the new feature is easy. Create ./src/index.js:

1. console.log(`I'm a silly entry point`);

and run the build again:

1. npm run build

You will get the bundle in ~/webpack-4-quickstart/dist/main.js.

What? Wait a moment. There is no need to define the output file? Exactly.

In **webpack 4 there is no need to define neither the entry point, nor the output file**.

I know that for a lot of people that’s not so exciting. Webpack’s main strength is code splitting. But trust me, having a zero configuration tool speeds things up.

So here is the first news: **webpack 4 doesn’t need a configuration file**.

It will look in **./src/index.js** as the default entry point. Moreover, it will spit out the bundle in**./dist/main.js**.

In the next section we’ll see another nice feature of webpack 4: **production and development mode**.

**webpack 4: production and development mode**

Having 2 configuration files is a common pattern in webpack.

A tipical project may have:

* a **configuration file for development**, for defining webpack dev server and other stuff
* a **configuration file for production**, for defining **UglifyJSPlugin**, sourcemaps and so on

While bigger projects may still need 2 files, in webpack 4 you can get by without a single line of configuration.

How so?

webpack 4 introduces **production** and **development** mode.

In fact if you pay attention at the output of npm run buildyou’ll see a nice warning:



*The ‘mode’ option has not been set. Set ‘mode’ option to ‘development’ or ‘production’ to enable defaults for this environment.*

What does that mean? Let’s see.

Open up [package.json](https://docs.npmjs.com/files/package.json" \t "_blank) and fill the script section like the following:

1. **"scripts"**: {
2. "dev": "webpack --mode development",
3. **"build"**: "webpack --mode production"
4. }

Now try to run:

1. npm run dev

and take a look at **./dist/main.js**. What do you see? Yes, I know, a boring bundle… not minified!

Now try to run:

1. npm run build

and take a look at **./dist/main.js**. What do you see now? A **minified bundle**!

Yes!

**Production mode** enables all sorts of optimizations out of the box. Including minification, scope hoisting, tree-shaking and more.

Development mode on the other hand is optimized for speed and does nothing more than providing an un-minified bundle.

So here is the second news: webpack 4 introduces **production** and **development** mode.

In webpack 4 you can get by without a single line of configuration! Just define the --modeflag and you get everything for free!

**webpack 4: overriding the defaults entry/output**

I love webpack 4 zero conf but how about overriding the default entry point? And the default output?

Configure them in package.json!

Here’s an example:

1. **"scripts"**: {
2. "dev": "webpack --mode development ./foo/src/js/index.js --output ./foo/main.js",
3. **"build"**: "webpack --mode production ./foo/src/js/index.js --output ./foo/main.js"
4. }

**webpack 4: transpiling Javascript ES6 with Babel**



Modern Javascript is mostly written in ES6.

But not every browser know how to deal with ES6. We need some kind of transformation.

This transformation step is called **transpiling**. Transpiling is the act of taking ES6 and making it understandable by older browsers.

Webpack doesn’t know how to make the transformation but has **loaders**: think of them as of transformers.

**babel-loader** is the webpack loader for transpiling ES6 and above, down to ES5.

To start using the loader we need to install a bunch of dependencies. In particular:

* babel-core
* babel-loader
* babel-preset-env for compiling Javascript ES6 code down to ES5

Let’s do it:

npm i babel-core babel-loader babel-preset-env --save-dev

Next up configure Babel by creating a new file named **.babelrc** inside the project folder:

1. {
2. **"presets"**: [
3. "env"
4. ]
5. }

At this point we have 2 options for configuring babel-loader:

* using a configuration file for webpack
* using --module-bindin your npm scripts

Yes, I know what you’re thinking. webpack 4 markets itself as a zero configuration tool. Why would you write a configuration file again?

The concept of **zero configuration in webpack 4** applies to:

* the **entry point**. Default to ./src/index.js
* the **output**. Default to ./dist/main.js
* **production and development mode** (no need to create 2 separate confs for production and development)

And it’s enough. But for using loaders in webpack 4 you still have to create a configuration file.

I’ve asked Sean about this. Will loaders in webpack 4 work the same as webpack 3? Is there any plan to provide 0 conf for common loaders like babel-loader?

His response:

“For the future (after v4, maybe 4.x or 5.0), we have already started the exploration of how a preset or addon system will help define this. What we don’t want: To try and shove a bunch of things into core as defaults What we do want: Allow other to extend”

For now you must still rely on **webpack.config.js**. Let’s take a look…

webpack 4: using babel-loader with a configuration file

Give webpack a configuration file for using babel-loader in the most classical way.

Create a new file named webpack.config.jsand configure the loader:

1. module.exports = {
2. module: {
3. rules: [
4. {
5. test: /\.js$/,
6. exclude: /node\_modules/,
7. use: {
8. loader: "babel-loader"
9. }
10. }
11. ]
12. }
13. };

There is no need to specify the entry point unless you want to customize it.

Next up open **./src/index.js** and write some ES6:

1. const arr = [1, 2, 3];
2. const iAmJavascriptES6 = () => console.log(...arr);
3. window.iAmJavascriptES6 = iAmJavascriptES6;

Finally create the bundle with:

1. npm run build

Now take a look at **./dist/main.js** to see the transpiled code.

webpack 4: using babel-loader without a configuration file

There is another method for using webpack loaders.

The --module-bindflag lets you specify loaders from the command line. Thank you Cezar for pointing this out.

The flag is not webpack 4 specific. It was there since version 3.

To use babel-loader without a configuration file configure your npm scripts in **package.json** like so:

1. **"scripts"**: {
2. "dev": "webpack --mode development --module-bind js=babel-loader",
3. **"build"**: "webpack --mode production --module-bind js=babel-loader"
4. }

And you’re ready to run the build.

I’m not a fan of this method (I don’t like fat npm scripts) but it is interesting nonetheless.

**webpack 4: setting up webpack 4 with React**



That’s easy once you’ve installed and configured babel.

Install React with:

1. npm i react react-dom --save-dev

Next up add babel-preset-react:

1. npm i babel-preset-react --save-dev

Configure the preset in **.babelrc**:

1. {
2. **"presets"**: ["env", "react"]
3. }

and you’re good to go!

To test things out you can create a dummy React component in ./src/App.js:

1. import React from "react";
2. import ReactDOM from "react-dom";
3. const App = () => {
4. **return** (
5. <div>
6. <p>React here!</p>
7. </div>
8. );
9. };
10. export default App;
11. ReactDOM.render(<App />, document.getElementById("app"));

Next up import the component in ./src/index.js:

1. import App from "./App";

and run the build again.

**webpack 4: the HTML webpack plugin**

webpack needs two additional components for processing HTML: html-webpack-plugin and html-loader.

Add the dependencies with:

1. npm i html-webpack-plugin html-loader --save-dev

Then update the webpack configuration:

1. const HtmlWebPackPlugin = require("html-webpack-plugin");
2. module.exports = {
3. module: {
4. rules: [
5. {
6. test: /\.js$/,
7. exclude: /node\_modules/,
8. use: {
9. loader: "babel-loader"
10. }
11. },
12. {
13. test: /\.html$/,
14. use: [
15. {
16. loader: "html-loader",
17. options: { minimize: true }
18. }
19. ]
20. }
21. ]
22. },
23. plugins: [
24. new HtmlWebPackPlugin({
25. template: "./src/index.html",
26. filename: "./index.html"
27. })
28. ]
29. };

Next up create an HTML file into ./src/index.html:

1. <!DOCTYPE html>
2. **<html** lang**=**"en"**>**
3. **<head>**
4. **<meta** charset**=**"utf-8"**>**
5. **<title>**webpack 4 quickstart**</title>**
6. **</head>**
7. **<body>**
8. **<div** id**=**"app"**>**
9. **</div>**
10. **</body>**
11. **</html>**

run the build with:

1. npm run build

and take a look at the ./distfolder. You should see the resulting HTML.

There’s no need to include your Javascript inside the HTML file: the bundle will be automatically injected.

Open up ./dist/index.htmlin your browser: you should see the React component working!

As you can see nothing has changed in regards of handling HTML.

webpack 4 is still a module bundler aiming at Javascript.

But there are plans for adding HTML as a module (HTML as an entry point).

**webpack 4 : the webpack dev server**

Running npm run devwhenever you make changes to your code? Far from ideal.

It takes just a minute to configure a development server with webpack.

Once configured [webpack dev server](https://github.com/webpack/webpack-dev-server" \t "_blank) will launch your application inside a browser.

It will automagically refresh the browser’s window as well, every time you change a file.

To set up webpack dev server install the package with:

1. npm i webpack-dev-server --save-dev

Next up open package.jsonand adjust the scripts like the following:

1. **"scripts"**: {
2. "start": "webpack-dev-server --mode development --open",
3. **"build"**: "webpack --mode production"
4. }

Save and close the file.

Now, by running:

1. npm run start

you’ll see webpack dev server launching your application inside the browser.

webpack dev server is great for development. (And it makes React Dev Tools work properly in your browser).