



Ibrahim Butt [Follow](#)  
Aspiring Software Engineer  
Mar 16 · 5 min read

## If you've ever configured Webpack, Parcel will blow your mind!

And how to hit the ground running with Parcel.



**In** this post I will walk you through using Parcel, with Pug, Sass, Babel and PostCSS. With some comparisons to Webpack, as its the most popular module bundler.

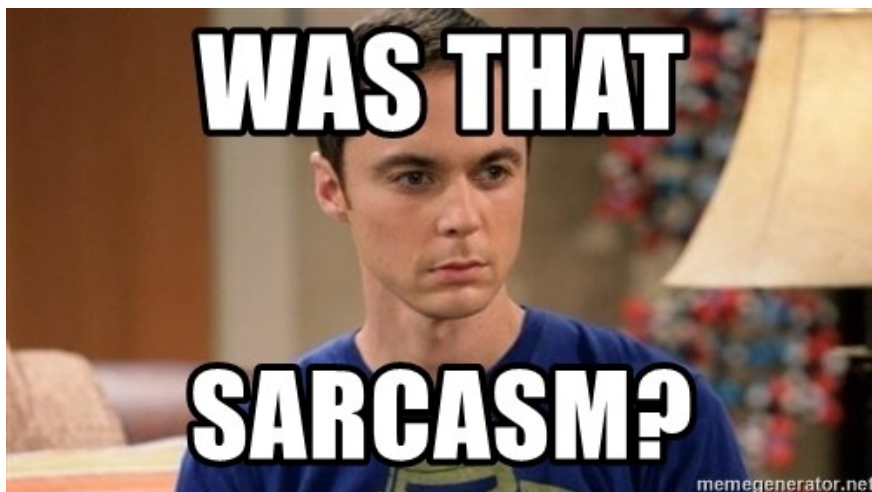
. . .

1. The problem with Webpack is its configuration. It's a nightmare. And I don't know about you, but I don't like nightmares. I prefer getting things done and not losing my mind in the process.

## Zero Configuration: Webpack Vs Parcel

### Webpack

Version 4 is ‘zero-config’.



Yes.

It’s zero-config in two ways:

You no longer have to specify an entry and output point.

Its production mode includes minification, scope hoisting, and tree-shaking.

Realistically, you’ll need a `webpack.config.js` for most projects.

**Edit:** @TheLarkInn left a response to this article and enlightened me with what zero-config means to Webpack:

*We bring a fully defaulted build out of the box because (webpack being rooted in extensibility) will allow users to define what Zero Config means to them. This keeps a slim core, allows us as a project to pivot fast and still gives every user a #OCJS experience.*

## Parcel

Parcel has native support for the following and more:

CSS

SCSS

Babel

PostCSS

PostHTML

TypeScript

JSX for React and Preact

Minification

Development Server

It doesn't have native support for scope hoisting and tree-shaking, but it will in a future version.

Parcel is the winner in the context of zero-config.

## Build Time

This table taken from Parcel's own site says it all.

Bundler	Time
browserify	22.98s
webpack	20.71s
parcel	9.98s
parcel - with cache	2.64s

Based on a reasonably sized app, containing 1726 modules, 6.5M uncompressed. Built on a 2016 MacBook Pro with 4 physical cores.

Webpack 4 uses a cache too, which helps with faster build times, but it's not completely implemented yet.

. . .

## Using Parcel: See yourself how easy it is

I'm going to walk you through setting up Parcel, to use Pug, Sass, Babel and autoprefixing via PostCSS. The first time I installed Parcel, I was actually finished in under 9 minutes.

# Lets Get To It

## 1 — Structure and Files

Make a new folder that will serve as the root directory. Inside that folder, create more folders and some files:

```
root-folder
--/ src
---/ pug
---- index.pug
---/ sass
---- main.sass
---- _background.sass
---/ js
---- main.js
---- message.js
```

You don't need to follow this structure, it's not required by Parcel. Its common practice to have a `src` folder where you write all your code. From here, it's compiled and/or transformed then output to `dist`.

## 2 — Installing Parcel and Pug

Run the following command in the root directory:

```
npm init -y && npm install --save-dev parcel-bundler parcel-plugin-pug
```

That's it. Parcel and the rest are set up, except for autoprefixer. It needs a `.postcssrc` file.

Continue with the rest of the article. keeping in mind from now on, you're not actually configuring Parcel.

## 3 — Lorem Ipsum?

Inside `index.pug`, paste in:

```
<!DOCTYPE html>
html (lang="en")
  head
    link(rel="stylesheet", href="../sass/main.sass")
  body
    script(src="../js/main.js")
```

If you chose a different structure for organising your files, change the paths.

Inside `_background.sass` , apply a background colour to `body` . I went with green, to signify Sass is working as soon as the development server loads. Import `_background.sass` to `main.sass` . To ensure PostCSS and autoprefixer are working, I added the following to

`main.sass` :

```
::placeholder
  color: grey
```

For `message.js` , what I did is write a function that prints a `string` to the console, using and ES6 arrow function:

```
export default (message) => {
  console.log(message);
};
```

For `main.js` , import `message` and use it:

```
import message from "../message";

message('Hello World!');
```

To get autoprefixer to work, we need to write a `.postcssrc` file inside the root directory:

```
{
  "plugins": {
    "autoprefixer": true
  }
}
```

This lets PostCSS know we want to use the autoprefixer plugin.

With Parcel, we don't need a `.babelrc` for Babel. If you need anything other than Babel's default functionality, you will need to create one. By default, Parcel will compile code for browsers that have more than 1% market share.

## 4 — Scripts

Parcel comes with a build mode. It minifies JavaScript, HTML and CSS using uglify-es, htmlnano and cssnano. All transformed and compiled files are output to dist inside the root.

We are going to setup two scripts. One which will start Parcel and its development server. Another for production using the build mode.

Paste this to `package.json` :

```
"scripts": {
  "start": "parcel src/pug/index.pug --open",
  "build": "parcel build src/pug/index.pug"
}
```

## 5 — Done!

Use `npm run start` to use Parcel for the first time. It'll take a little longer than usual since it's the first time.

The development server will open your default browser. The first thing you should notice is the background colour. This, of course, means Pug did transform to HTML and Sass to CSS.

Now open up the console, you will have a message.

While you're in the developer tools, check out the CSS. `::placeholder` is prefixed.

Have a look at the JavaScript file and look for `var _message`. If it's not in the first JavaScript file, it will be in the second one. In my experience, it's the second one.

If you met all the above 'conditions', congratulations. You have installed and set up Parcel to work with Pug, Sass, Babel and PostCSS.

## Conclusion


As you can see, you can hit the ground running with Parcel. Keep in mind that Parcel was released in December 2017—its still in infancy. I expect a lot more zero-config features in the future.

Of course, a lot of users will need to configure Parcel and that's expected. It can't have native support for everything. The point is, Parcel is very easy to use from the get-go. That's what I'm trying to showcase here and not just it's 'zero-config' capabilities. Hopefully, this approach to usability will be applied to its configuration too. In contrast to Webpack, that's big.

I do like Webpack, I just don't like setting it up. In the end, it's a matter of preference.

I won't be looking back at Gulp or Webpack for future projects unless I have to. What are your thoughts? Will you give Parcel a try or are you happy with Webpack?

. . .

*If you found this article helpful, please share it with all your Medium friends and hit that  button below to spread it around even more—it would help me tremendously.*

*Follow me on here and my social accounts to be notified of future posts—Instagram Twitter GitHub.*

. . .

*Other articles by me:*

How to make a pure HTML & CSS slider using  
`:checked`!

It Just Works  
medium.com

