

Webpack DeMystified

And why even build?

Game Plan:

How Long	What
5	Warmup
20	Hands on
5	Tomato
25	Hands on
5	Tomato
25	Over- and Re-view

Warm up

Turn and talk

What has your experience been like so far with linking up HTML, JavaScript, and CSS?

Quick note about Webpack 2

Hi, I'm a Webpack 2 user and I've been using it for a while now. I've found it to be a very powerful and flexible tool for bundling JavaScript and CSS. I've been using it to build my web applications and I've found it to be a very reliable and easy to use tool. I've been using it to build my web applications and I've found it to be a very reliable and easy to use tool.

Part 1: Big Picture

What is the overall goal of the project?

What are the key deliverables?

What are the main risks?

What are the key stakeholders?

What are the key dependencies?

What are the key milestones?

What are the key metrics?

What are the key assumptions?

What are the key conclusions?

The Purpose of Build Tools

Let's say you're building a game in JavaScript and you're new to this whole coding thing. Then let's say that your HTML file looks like this:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Webpack Demystified</title>
</head>
<body>
  <h1>Luke, I am your father.</h1>
  <script src="bundle.js"></script>
  <script src="game.js"></script>
  <script src="thing.js"></script>
  <script src="jquery.js"></script>
  <script src="bootstrap.js"></script>
  <script src="other-thing.js"></script>
  <script src="something-else.js"></script>
  <script src="stuff.js"></script>
  <script src="ridiculous.js"></script>
  <script src="ughhhhh.js"></script>
</body>
</html>
```

JavaScript

CSS

Assets

Oh

my!

Gulp, Grunt, Webpack

== ["Package Manager", "Bundler", "Task Runner"]

Primary responsibility:

Scan through all of your projects' dependencies, handle any task that needs to be done prior to production, and ship it off in a neat little package that is easy for a browser to digest.

How do our needs differ in different environments

We've talked about development, test and production environments before. The environment that's best for us to code in is not the best environment for the browser to execute our code. Let's dive into how we treat each of these environments differently.

Our Wants

- We love whitespace
- CSS is kind of cumbersome. Would love to use SASS or LESS.
- We use other people's code
- We write tests

Browser Needs

- Has to download files
- Can only process CSS and ES6, in some cases only ES5. JS is evolving faster than browsers are implementing it
- There's lots of browsers, and their needs differ

How build tools bridge the gap

- Transpilers, like Babel, translate our easier to maintain code into code that the browser can interpret
- CSS Preprocessors, like SASS or LESS, evaluate our SCSS files and write CSS the browser can interpret
- Minifiers remove whitespace, and can reduce variable name length
- They also combine everything into one file, which

So What is Webpack?

Webpack is a node package that digs through your asset files, finds any dependencies, and spits out a single JS file that is ready for production.

"Loaders" pre-process your assets (like Fonts, SASS, Images, CSS, SVGs etc) and output exactly what your browser needs to know in the smallest package possible.

webpack-dev-server and hot-module-replacement

Checks for understanding

Turn and talk

1. Name a few things that build tools do? Why do we need those things.
2. What are some build tools you've used before?
Think about code that has been translated between environments.
3. What kinds of things would we be forced to do in development if we didn't have build tools?

Part 2: Webpack Tour

1. Instructor led
2. Small group exploration
3. Solo

Install Some Command Line Tools

```
npm install -g webpack webpack-dev-server mocha
```

Clone down your starter kit

1. Clone down the starter kit

2. Install the dependencies

3. Run the application

4. Deploy the application

5. Monitor the application

6. Scale the application

7. Secure the application

8. Update the application

Webpack Config

So what does Webpack configuration look like?

At its most basic level, you'll see something like this:

```
module.exports = {  
  entry: {  
    main: "./index.js",  
  },  
  output: {  
    filename: "main.bundle.js"  
  },  
  module: {  
    loaders: [  
      {test: /\.js$/, loaders: ['babel'], exclude: /node_modules/},  
    ]  
  }  
}
```

Breaking Down Loaders

Loaders take an array of objects that use regex to specify what file extensions to look for and what loaders are needed to make things happen.

```
module: {  
  loaders: [  
    {test: /\.js$/, loaders: ['babel'], exclude: /node_modules/},  
    {test: /\.css$/, loaders: ['style', 'css']},  
    {test: /\.scss$/, loader: "style!css!sass" },  
  ],  
}
```

PRO TIP: Loaders run from right to left! So it will run the "sass" loader first, then the "css" loader, then the "style" loader and do all the things.

HTML files

Let's open up `index.html` and see what that looks like.

Notice the single `<script>` tag located before the closing `</body>` tag. This is the epicenter of webpack.

`foods.html` is very similar. We'll talk about `test.html` after this quick demo of Webpack in action.

Brief Demo: Webpack In Action

Make a new file in your lib directory called alert.js and export a simple alert function.

```
touch lib/alert.js
```

```
alert.js
```

```
module.exports = function() {  
  alert('ITS A TRAP!!!!!!!!!!!!');  
}
```


Then require said file and call the function in our entry index.js file.

index.js

```
var newAlert = require('./alert');  
newAlert();
```

Now open up index.html

open index.html

...buzzkill. No alert. We need to build!

webpack

This will run webpack using the defined config, and modify our main.bundle.js. Then try refreshing the page, or just type open index.html again.

Time to Automate!

Do we want to have to run webpack and refresh the page every time we want to see our code changes in the browser?

Enter webpack-dev-server. This will boot up a development server and run our configuration file and reload our changes anytime we refresh our browser.

Try it out!

Writing Tests

In test/index.js, write a simple test.

```
const assert = require('chai').assert

describe('our test bundle', function () {
  it('should work', function () {
    assert(true)
  })
})
```

Running Tests

There are some times when you want your tests to run in the browser instead of in node.

With your webpack-dev-server running, visit `http://localhost:8080/test.html`. This will run the same tests, but in the browser environment instead. Let's poke around `test.html` to see if we can figure out how this works.

Also note that just like in your `lib/index.js` file, you can require other test files within the entry point `test/index.js` file and Webpack will bundle for you.

As long as test files all get required in `test/index.js`, then you can run your tests from the browser or from the terminal.

Using package.json scripts

package.json makes it easier to run commands. Let's make a few changes so we can keep shortcut some terminal commands.

```
// package.json
...
"scripts": {
  "start": "webpack-dev-server --hot --inline",
  "build": "webpack",
  "test": "mocha"
},
...
```


This lets us use the commands `npm start` to fire up `webpack-dev-server`, `npm run build` to package everything for production, and `npm test` to execute our testing suite.

The `--hot` `--inline` flags tell `npm` to watch for any changes and reload automatically so we can stop typing stuff into our terminal.

Styling

As a quick note, webpack also allows you to require styling, like .css and .sass files the same way you would any other js file. Behind the scenes, it's taking all your CSS, and appending it to your HTML at the time the JS file is read in. It's kind of a hack, but it allows you to have just a single .js file that loads all your logic and styling in one go.

Try creating a really simple .css file in your lib folder, and requiring it in lib/index.js. You won't

Deployment

Let's look at the [Github Pages section](#) of the starter kit README.

Checks for understanding

Turn and talk with someone new!

Answer the following in the context of using Webpack:

1. What is your development process? What steps do you need to take before you can start developing?
2. What is your test process?
3. What is your deployment process?