Demystifying Webpack

Turku❤️Frontend
26 April 2017

Hello! 👋

- I'm Aarni Koskela (@akx)
- I've been writing code since I was about 6 years old
- … and doing web stuff since IE 6 was a Good Thing
- Now working at Valohai on a cutting-edge machine learning infrastructure SaaS (that's a mouthful)

A Disclaimer

- I'm not going to cover all the more advanced features Webpack supports in this presentation.

- Juho Vepsäläinen (@bebraw) has a book and a presentation on that! https://leanpub.com/survivejs-webpack

And by the way

Please feel free to interrupt at any time!

So, what is Webpack anyway?

- Webpack is a module bundler
  (like its predecessor Browserify)

- It . . .
  - takes one or more files
    (primarily, but not only, JavaScript)
  - transforms them
  - produces one or more bundle files
    (again, primarily but not necessarily only JavaScript)

Why Webpack?

- If you want to use ES6/JSX syntax in all browsers (e.g. using Babel or Bublé)
- If you want to optimize and minify your code before delivering it to clients (built-in!)
- If you want nicer encapsulation and componentization (CSS within your bundle, for instance)
- If you want live code reloading (lovely for development)

But isn't Webpack impossible to configure?

- That's the myth!
- Webpack is pretty easy to get going with (with a basic configuration)
- Of course, more complex configurations (splitting, per-module hot reloading, ...) get more complicated

Let's do this!

- Let's set up a simple SPA (single page application) from scratch with
  - Webpack
  - React
  - Stylus-based stylesheets
    (could just as well be SASS/SCSS or LESS)
  - Photos of cats, because of course :)

- (in reality, you might want to start something like this using create-react-app, but for the sake of practice…)

First: installing stuff!

- The packages we need are:
  - webpack (duh)
  - react (React itself)
  - react-dom (React's browser interface)
  - file-loader (allows **require**ing binary files)
  - style-loader (injects <style> tags for CSS)
  - css-loader (loads and minifies CSS)
  - stylus-loader (loads Stylus style sheets)
  - stylus (a dependency of stylus-loader)
  - babel-loader (Webpack loader for Babel)
  - babel-core (Babel's core without a CLI)
  - babel-preset-env (a general ES6 Babel preset)
  - babel-preset-react (a Babel preset for React)
- Also, for development, let's install
  - webpack-dev-server (development server)

Yarrrrn!

- We could just as well use npm, but yarn is faster, and fast is good (in this case).

- yarn init

- yarn add webpack react react-dom url-loader file-loader style-loader css-loader stylus-loader stylus babel-loader babel-core babel-preset-react babel-preset-env

- yarn add --dev webpack-dev-server

artwork from yarnpkg.com

Second: the configuration!

- Webpack can be run without a configuration file, but no one does that.

- Webpack is configured by way of a JavaScript configuration file, by default **webpack.config.js**.

- The configuration file is Real, Actual JavaScript™, which will be executed by Node.js, so you can do whatever you like in it, as long as you **module.exports =** a configuration object at the end.

- We'll just keep it reasonably simple for now…

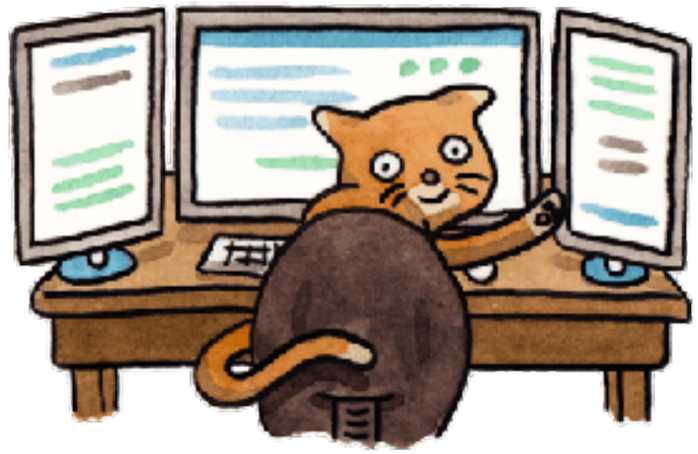
Simplest configuration

```
module.exports = {
  entry: './src/index.js',
  output: {
    path: __dirname + '/dist',
    filename: 'bundle.js',
  },
};
```

- This won't do anything fancy since no loaders are set
- But it'll bundle things and let us to use **require()** already!

Let's write some code!

- This is **src/index.js**, our entry point.



```
import React from 'react';
import ReactDOM from 'react-dom';

document.write('herlo worl');
```

artwork from yarnpkg.com

Let's rev up the dev server!

- Let's add an incantation to open the dev server to our **package.json** file, so it's easier to start now and later.

```
"scripts": {
   "dev": "webpack-dev-server"
}
```

- Now all we need to do is **yarn dev** . . .

- and navigate to http://localhost:8080/bundle – a virtual HTML file that loads our bundle

Let's add loaders!

- Anything other than plain JavaScript must be transformed by way of **loaders**; we've already installed a bunch of them, so let's configure them…

Oooh, configuration

```
module: {
  rules: [
    {
      test: /.styl$/,
      use: ['style-loader', 'css-loader', 'stylus-loader'],
    },
    {
      test: /.(jpg|jpeg|png)$/,
      use: ['file-loader'],
    },
    {
      test: /.jsx?$/,
      use: {
        loader: 'babel-loader',
        options: {
          presets: ['env', 'react'],
        },
      },
      exclude: /node_modules/,
    },
  ],
}
```

Let's write some React!

```
import React from 'react';
import ReactDOM from 'react-dom';
import CatImageURL from './cat.png';

class CatApp extends React.Component {
   constructor() {
      super();
      this.state = {number: 1};
   }
   render() {
      const cats = [];
      for(let i = 0; i < this.state.number; i++) {
         cats.push(<img src={CatImageURL} key={i} />);
      }
      return (
         <div>
            <h1>Cat app</h1>
            <button onClick={(e) => this.setState({number: this.state.number + 1})}>More mew!</button>
            <br />
            {cats}
         </div>
      );
   }
}

ReactDOM.render(<CatApp />, document.body);
```

Note how any edits reload the application in real time! 😎

Let's also add some style!

- Since **style-loader** supports hot reloading, so we can tweak styles even faster!
  - (That is, if we're running the dev server with **--hot**)
- `import style from './style.styl';`

We're done developing, let's go to production!

- The **-p** (for production) flag to **webpack** (the CLI tool) enables minification and other optimizations
- The **--progress** flag lets us watch exciting things happen
- So let's run:
  ./node_modules/.bin/webpack -p --progress
- And lo and behold; there's a bundle.js and a PNG file in **dist/**
  - Look ma, no CSS file though!
    (It's embedded in the JS bundle.)
- You can serve the built files with any old web server, and/or embed them in your apps.

That's it!

Simple, no?

Thank you!
Questions?

@akx – aarni@valohai.com