Add Webpack Hot Module Replacement (HMR) to a Drupal Theme

Last updated May 18, 2020

Theming Module Development 8.9.x/9.0.x

Hot Module Replacment (HMR) is a technique for using Webpack to update the code that your browser renders without requiring a page refresh. It's similar to LiveReload with some additional features that make it better for working with React. With HMR configured it's possible to edit your JavaScript (and CSS) files in your IDE and have the browser update the page without requiring a refresh, allowing you to effectively see the results of your changes in near real time. If you're editing JavaScript or CSS it's amazing. And, it's one of the reasons people love the developer experience of working with React so much.

If you're writing React code as part of a Drupal theme it's possible configure HMR to work with Drupal. Doing so will allow live reloading of any JavaScript and CSS processed by Webpack.

In this tutorial we'll:

- Walk through configuring Webpack hot module replacement for a Drupal theme
- Add an `npm run start:hmr` command that will start the *webpack-dev-server* in HMR mode
- Configure the *webpack-dev-server* to proxy requests to Drupal so we can view our normal Drupal pages

By the end of this tutorial you should know how to add Webpack's hot module replacement feature to your Drupal theme and preview changes to your React code in real time.

Goal

Configure Webpack hot module replacement for use with a Drupal theme.

Prerequisites

- Connect React to a Drupal Theme or Module

Hot module replacement

There's a detailed technical explanation of what HMR is, and how it works in this post on Stack Overflow.

Our implementation consists of the following components:

- webpack-dev-server to create a development server that can handle the requirements of HMR. Standard Apache for example can not.
- react-hot-loader to integrate Webpack's HMR features with React

Note: We're aware of React Fast Refresh, but at this time it's integration with Webpack isn't great. We'll keep an eye on things though and once it becomes the community supported way to do HMR with Webpack we'll update this tutorial.

Add Webpack HMR to a Drupal theme

These steps assume that you've setup and configured Webpack using the guidelines in Connect React to a Drupal Theme or Module.

1 Install the required modules

Install `webpack-dev-server`, `react-hot-loader`, and `@hot-loader/react-dom`.

From the root directory of your Drupal theme run:

```
npm install --save-dev webpack-dev-server
npm install react-hot-loader @hot-loader/react-dom
```

2 Update your React code's *index.jsx*

Update the *index.jsx* file, or whichever file is the main entry point for your React application, to make it as *hot*. This requires adding `import { hot } from 'react-hot-loader/root';` and then wrapping your *Main* component with the imported `hot()` function.

Example:

```
import React from 'react'
import ReactDOM from 'react-dom'
import { hot } from 'react-hot-loader/root';

/* Import Components */
import NodeReadWrite from "./components/NodeReadWrite";

const Main = hot(() => (
  <NodeReadWrite/>
));

ReactDOM.render(<Main/>, document.getElementById('react-app'));
```

Note: this is safe to deploy to production so it's okay to commit these changes to your application.

3 Update webpack.config.js

Edit the *webpack.config.js* file, here's the final version, we'll explain the changes in detail below:

```
const path = require('path');
const isDevMode = process.env.NODE_ENV !== 'production';

const PROXY = 'https://react-tutorials-2.ddev.site/';
const PUBLIC_PATH = '/themes/react_example_theme/js/dist_dev/';

const config = {
  entry: {
    main: [
      "react-hot-loader/patch",
      "./js/src/index.jsx"
    ]
  },
  devtool: (isDevMode) ? 'source-map' : false,
  mode: (isDevMode) ? 'development' : 'production',
  output: {
    path: isDevMode ? path.resolve(__dirname, "js/dist_dev") : path.resolve(__dirname, "js/dist"),
    filename: '[name].min.js',
    publicPath: PUBLIC_PATH
  },
  resolve: {
    extensions: ['.js', '.jsx'],
    alias: {
      'react-dom': '@hot-loader/react-dom',
    },
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        loader: 'babel-loader',
        exclude: /node_modules/,
        include: path.join(__dirname, 'js/src'),
        options: {
          // This is a feature of `babel-loader` for webpack (not Babel itself).
```

```
          // It enables caching results in ./node_modules/.cache/babel-loader/
          // directory for faster rebuilds.
          cacheDirectory: true,
          plugins: ['react-hot-loader/babel'],
        },
      }
    ],
  },
  devServer: {
    port: 8181,
    hot: true,
    https: true,
    writeToDisk: true,
    headers: { 'Access-Control-Allow-Origin': '*' },
    // Settings for http-proxy-middleware.
    proxy: {
      '/': {
        index: '',
        context: () => true,
        target: PROXY,
        publicPath: PUBLIC_PATH,
        secure: false,
        // These settings allow Drupal authentication to work, so you can sign
        // in to your Drupal site via the proxy. They require some corresponding
        // configuration in Drupal's settings.php.
        changeOrigin: true,
        xfwd: true
      }
    }
  },
};

module.exports = config;
```

In comparison to the *webpack.config.js* from Connect React to a Drupal Theme or Module here are the things we've modified:

- Updated the Webpack entry point configuration, and included *react-hot-loader/patch*, telling Webpack to make sure *react-hot-loader* is required before *react* and *react-dom* so that it can do some low-level patching of that code.
- Use a Webpack resolver alias to replace instances of `react-dom` with `@hot-loader/react-dom` to enable support for React hooks.
- Change the `babel-loader` configuration, and tell it to enable caching, and to use the `react-hot-loader/babel` babel plugin so that babel can inject the required code for hot module replacement.
- Add the `devServer` configuration for `webpack-dev-server`. Configuration options.

4 Tell webpack-dev-server to proxy requests to Drupal

For HMR to work we need to view the pages of our site through the lens of the webpack-dev server. However, in this scenario we only want webpack-dev-server to serve the JavaScript files that are processed by Webpack. And for everything else to come from Drupal.

You request `/node/42` from the webpack-dev-server it needs to be able to recognize that this is a request it's not configured to handle, and instead pass the request to Drupal. Drupal can then process the request, and pass the resulting HTML page back to webpack-dev-server, which in turn passes it back to your browser.

The resulting page includes tags like `<img>`, `<link>`, and `<script>` that instruct your browser to retrieve additional resources. Like this:

```
<script src="/themes/react_example_theme/js/dist_dev/main.min.js?v=8.8.2"></script>
```

So the browser requests that file from webpack-dev-server. The dev server recognizes this as file that it is responsible for, and instead of proxying the request to Drupal it returns the version it has.

In the end it'll look just like you're browsing your normal Drupal site, but some of the files will actually be coming from *webpack-dev-server*.

Change the following variables in your *webpack.config.js*:

```
// The base path of your Drupal development server. This is what you would
// normally navigate to in your browser when working on the site.
const PROXY = 'https://react-tutorials-2.ddev.site/';
// The absolute path to the directory, relative to the PROXY path above, to the
// directory that contains the files that you want webpack-dev-server to NOT
```

```
// pass on to Drupal.
// For example, if your .js file is normally accessed via http://a.com/src/script.js
// and you want webpack-dev-server to handle all the .js files in the src
// directory set this to '/src/'.
const PUBLIC_PATH = '/themes/react_example_theme/js/dist_dev/';
```

Behind the scenes this uses the powerful http-proxy-middleware. And you can find more options in its documentation. In our testing we've used DDEV-local for hosting our Drupal development site, you may need to change the settings a bit depending on the specifics of your development environment.

5 Configure Drupal to accept requests from the proxy

Edit your Drupal site's *settings.php*, or *settings.local.php* file and add the following options:

```
$settings['reverse_proxy'] = TRUE;
$settings['reverse_proxy_addresses'] = array($_SERVER['REMOTE_ADDR']);
```

6 Add a helper script to start the development server

Update your *package.json* file to include a helper script for starting the *webpack-dev-server*. The command to do so is: `webpack-dev-server --hot --progress --colors`. The final *package.json* with all the necessary packages and changes looks like this:

```
{
  "name": "react_example_theme",
  "version": "1.0.0",
  "description": "",
  "main": "js/src/index.jsx",
  "scripts": {
    "build": "NODE_ENV=production webpack --mode=production",
    "build:dev": "webpack",
    "start": "webpack --watch",
    "start:hmr": "webpack-dev-server --hot --progress --colors"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@babel/core": "^7.8.4",
    "@babel/preset-env": "^7.8.4",
    "@babel/preset-react": "^7.8.3",
    "babel-loader": "^8.0.6",
    "webpack": "^4.41.6",
    "webpack-cli": "^3.3.11",
    "webpack-dev-server": "^3.10.3"
  },
  "dependencies": {
    "@hot-loader/react-dom": "^16.13.0",
    "react": "^16.12.0",
    "react-dom": "^16.12.0",
    "react-hot-loader": "^4.12.19"
  }
}
```

7 Start the development server and test it out

From the root directory of your theme, where the *package.json* file is run:

```
npm run start:hmr
```

This will output something like the following:

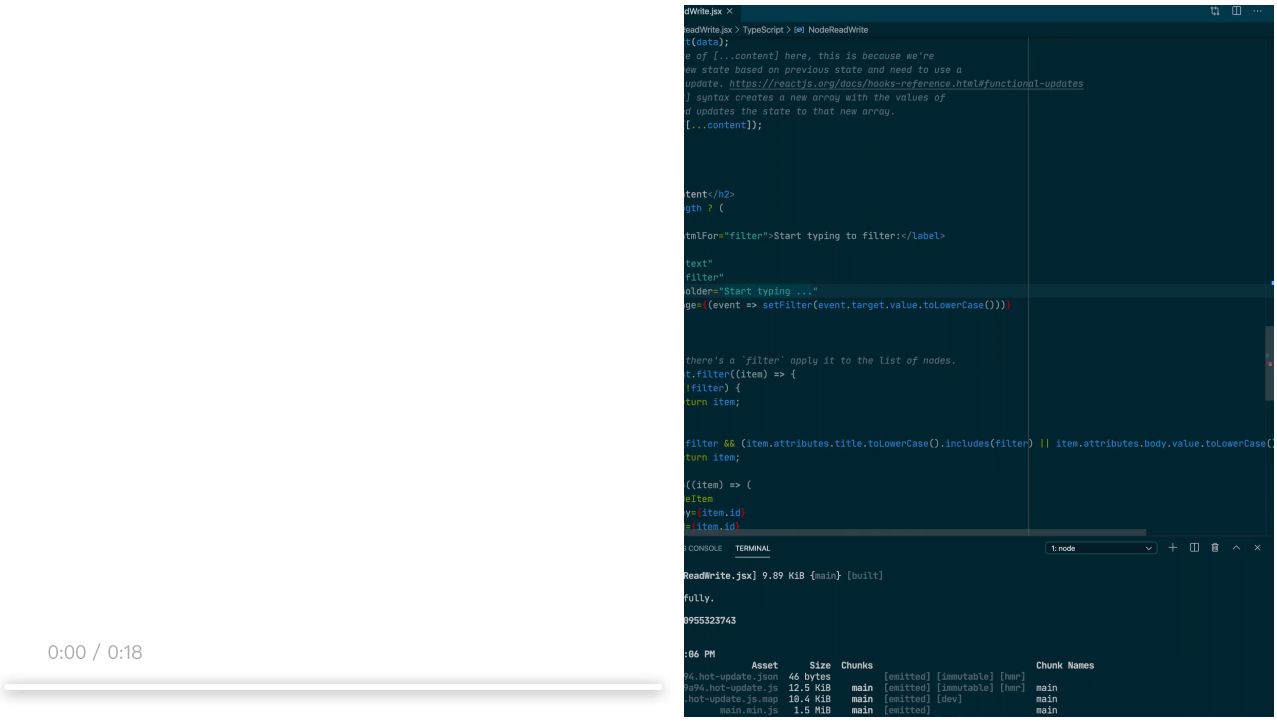
```
> react_example_theme@1.0.0 start:hmr /Users/joe/Sites/demos/react-tutorials-2/drupal/web/themes/react_example_theme
> webpack-dev-server --hot --colors

i [wds]: Project is running at https://localhost:8181/
i [wds]: webpack output is served from /themes/react_example_theme/js/dist_dev/
i [wds]: Content not from webpack is served from /Users/joe/Sites/demos/react-tutorials-2/drupal/web/themes/react_example_theme
i [wdm]: Hash: f03825bda996b935ae79
Version: webpack 4.41.6
Time: 1334ms
```

```
Built at: 05/14/2020 12:22:47 PM
          Asset      Size  Chunks                    Chunk Names
    main.min.js   1.5 MiB    main  [emitted]         main
main.min.js.map  1.71 MiB    main  [emitted] [dev]   main
Entrypoint main = main.min.js main.min.js.map
```

In which you can see that the development server is now accessible at `https://localhost:8181`. If you visit the address you should see your Drupal site.

If you make changes to any of the React code in your theme those changes should update on the site in near real time.

Example of hot module replacement in a Drupal theme:



Recap

In this tutorial we learned how to perform live reloading of changes to our React code without requiring a page refresh. We configured Webpack, and the *webpack-dev-server* to perform hot module reloading, and integrated it with *react-hot-loader*. Then we set things up so our Webpack development server can proxy requests to our Drupal development environment and we can see changes to our Drupal theme's JavaScript in near real time.

Further your understanding

- Can you update your Webpack toolchain to compile Sass/SCSS files and perform hot reloading of those?
- You still need to build the final production assets using `npm run build` when using this approach. Why?

Additional resources

- webpack-dev-server documentation (webpack.js.org)
- react-hot-loader documentation (github.com)
- An example showing HMR for CSS in a Drupal theme (github.com)

Get Started Using React and Drupal Together