# Webpack

## What is webpack?

It is a module bundler for “advanced Javascript” and other asset files. It can be used as a standalone or with tools such as Gulp, which is a Javascript task runner.

## What does it do?

It recursively builds a dependency graph that includes every module that your application needs. It then packages all of those modules into a small number of “bundles” to be then loaded by the browser.

* The output of the bundling process and be one, or several bundles, depending on the configuration file.

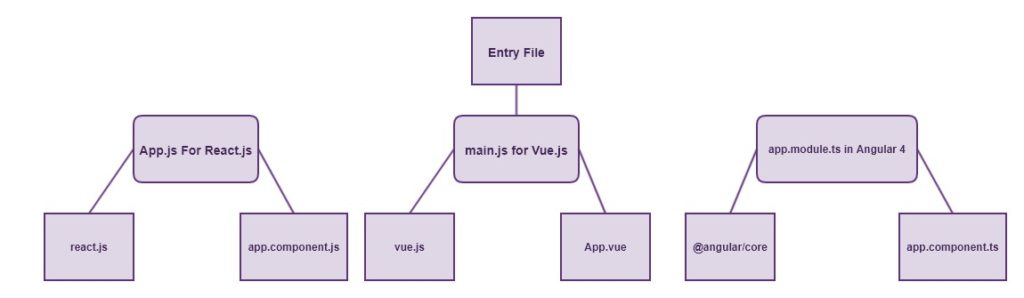
## webpack.config.js

The main configuration file that is used to set parameters around how Webpack should behave.

## Key concepts

### Entry

Is the entry point for the dependencies graph, where to start the search from to discover what should be bundled. All of the dependencies should be included directly or indirectly within this file. An indirect reference refers to a dependency that is not explicitly included in the entry file but is required by one of the dependencies that are included.



<https://appdividend.com/2017/08/17/webpack-3-tutorial-from-scratch/>

Webpack will search node by node within the dependency graph then output a bundle.js file once it has completed its recursive search.

#### Example

// webpack.config.js

module.exports = {

Entry: ‘./path/to/my/entry/file.js’

};

…There are multiple ways to declare your entry property that is unique to your application’s needs. See <https://webpack.js.org/concepts/entry-points/> for more information.

### Output

Is how the bundled code is treated, i.e. where should it be stored.

#### Example

// webpack.config.js

const path = require('path');

module.exports = {

entry: './path/to/my/entry/file.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'my-first-webpack.bundle.js'

}

};

This example uses the Node.js path module to specify the current directory, then inside a folder called “dist”.

### Loaders

Webpack only understand Javascript files, so all vendor specific Javascript files (such as TypeScript, ES#, or Vue) need to be interpreted then converted.

Depending on the language you are going to use, you will need to include a language specific loader module such as babel-loader for ES6 to ES5 transpilation or vue-loader for .vue files. These loaders help with getting new features of Javascript into older browsers.

Loaders have two purposes:

1. Identify which files should be transformed, using the *test* property.
2. Transform the identified files so that they can be added to the dependency graph, using the *use* property.

#### Example

// webpack.config.js

const path = require('path');

const config = {

entry: './path/to/my/entry/file.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'my-first-webpack.bundle.js'

},

module: {

// Special compilation rules

loaders: [

{

// Ask webpack to check: If this file ends with .js, then apply some transforms

test: /\.js$/,

// Transform it with babel

loader: 'babel-loader',

// don't transform node\_modules folder (which don't need to be compiled)

exclude: /node\_modules/,

include: [

path.resolve(\_\_dirname, '../src')

]

},

{

// Ask webpack to check: If this file ends with .vue, then apply some transforms

test: /\.vue$/,

// don't transform node\_modules folder (which don't need to be compiled)

exclude: /(node\_modules|bower\_components)/,

// Transform it with vue

use: {

loader: 'vue-loader'

}

}

]

}

};

module.exports = config;

### Plugins

Plugins allow us to extend the functionality provided by Webpack, or do things that a loader cannot do. Frequently used plugins are: UglifyJsPlugin for minifying Javascript code, EnvironmentPlugin for referencing environmental variables, HotModulReplacementPlugin for generating hot chunks of updated code during the development process.

In order to use a plugin you need to include a reference to it using the require syntax, and include it within the plugins array. You can use a plugin multiple times for different purposes, if you want to do this then you will need to create a new instance of it using the *new* keyword.

#### Example

const HtmlWebpackPlugin = require('html-webpack-plugin'); //installed via npm

const webpack = require('webpack'); //to access built-in plugins

const path = require('path');

const config = {

entry: './path/to/my/entry/file.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'my-first-webpack.bundle.js'

},

module: {

rules: [

{ test: /\.txt$/, use: 'raw-loader' }

]

},

plugins: [

new webpack.optimize.UglifyJsPlugin(),

new HtmlWebpackPlugin({template: './src/index.html'})

]

};

module.exports = config;

## Options

Watch  
Turn on watch mode. This means that after the initial build, Webpack will continue to watch for changes in any of the resolved files.

Cache  
Cache the generated Webpack modules and chunks to improve build speed.

Devtool  
This option controls if and how Source Maps are generated. See here the different options.

Context  
The base directory, an absolute path, for resolving entry points and loaders from configuration.

Entry  
The point or points to enter the application. At this point the application starts executing. The key will be the name of the compiled file. eg: home: app.js become home.bundle.js

Output  
Output options tell Webpack how to write the compiled files to disk. Note, that while there can be multiple entry points, only one output configuration is specified (see output documentation).

Resolve  
Configure how modules are resolved.

Modules  
Tell webpack what directories should be searched when resolving modules.

Alias  
Create aliases to import or require certain modules more easily. The documentation explains very well this parameter.

Plugins  
A list of webpack plugins. For example, when multiple bundles share some of the same dependencies, the CommonsChunkPlugin could be useful to extract those dependencies into a shared bundle to avoid duplication.

Example  
var webpack = require('webpack'),

path = require('path');

var srcPath = path.join(\_\_dirname, '/src/js'),

distPath = path.join(\_\_dirname, '/dist/js');

module.exports = {

watch: true,

cache: true,

devtool: '#cheap-module-eval-source-map',

context: srcPath,

entry: {

app: './app.js',

},

output: {

path: distPath,

filename: '[name].bundle.js',

},

resolve: {

modules: ["node\_modules"],

},

plugins: [

new webpack.NoEmitOnErrorsPlugin()

]

};