Monitor and analyze the app

By Ivan Akulov

Contracting software engineer · Blogging, open source, performance, UX

Even when you configure webpack to make the app as small as possible, it's still important to keep track of it and know what it includes. Otherwise, you can install a dependency that will make the app twice as large – and won't even notice it!

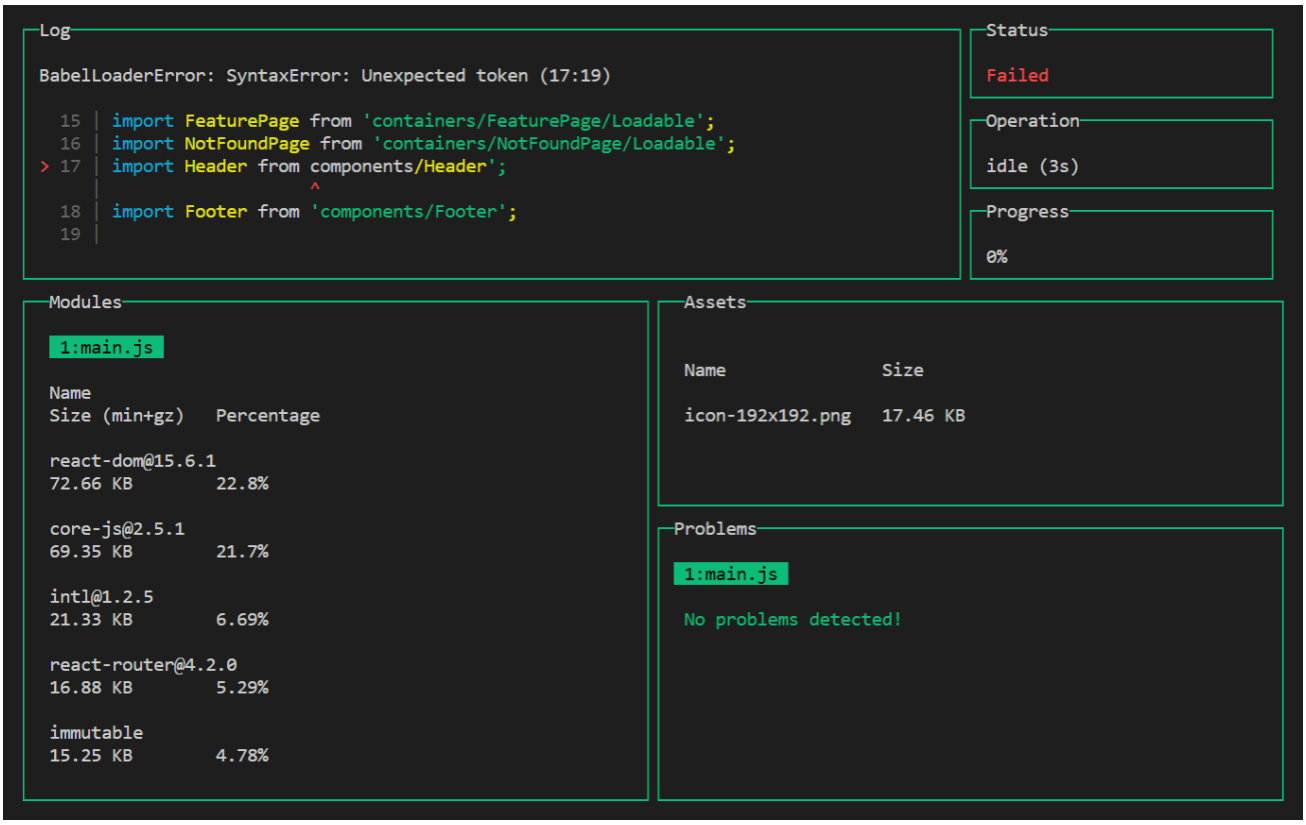
This section describes tools that help you to understand your bundle.

Keep track of the bundle size

To monitor your app size, use webpack-dashboard during development and bundlesize on CI.

webpack-dashboard

webpack-dashboard enhances webpack output with sizes of dependencies, progress and other details. Here's how it looks:



This dashboard helps to track large dependencies – if you add one, you'll immediately see it in the *Modules* section!

To enable it, install the **`webpack-dashboard`** package:

```
npm install webpack-dashboard --save-dev
```

And add the plugin into the **`plugins`** section of the config:

```
// webpack.config.js
const DashboardPlugin = require('webpack-dashboard/plugin');

module.exports = {
  plugins: [
    new DashboardPlugin(),
  ],
};
```

or using **`compiler.apply()`** if you're using an Express-based dev server:

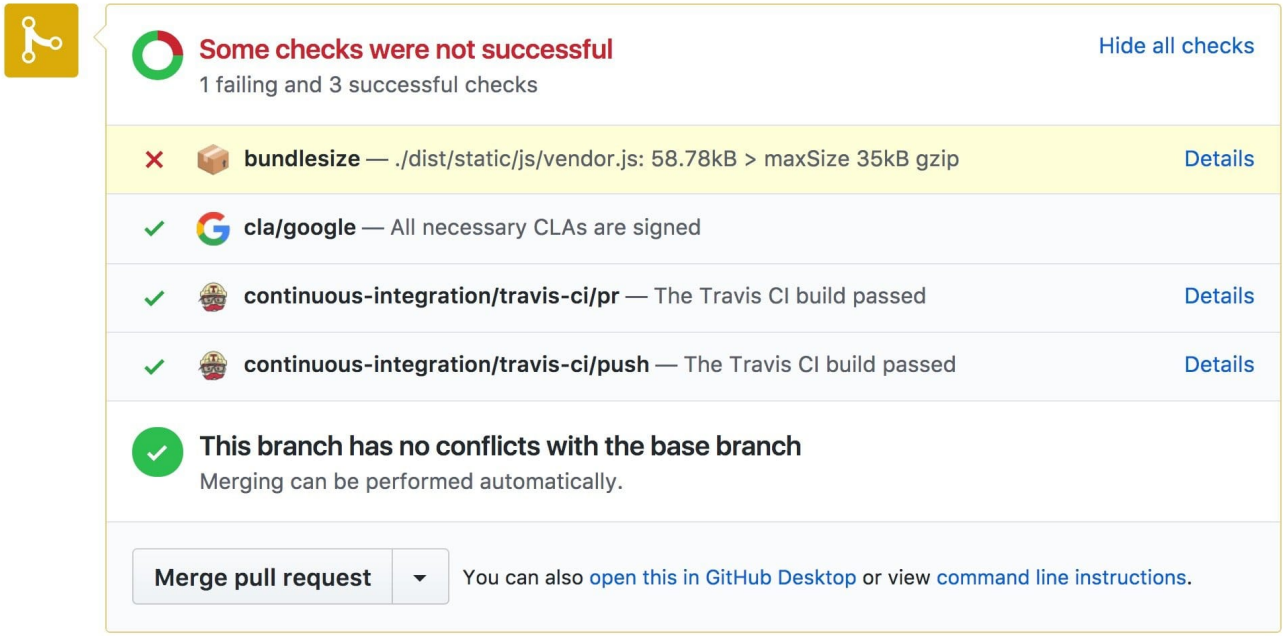
```
compiler.apply(new DashboardPlugin());
```

Feel free to play with the dashboard to find the probable places for improvement! For example, scroll through the *Modules* section to find what libraries are too large and could be

replaced with smaller alternatives.

bundlesize

bundlesize verifies that webpack assets don't exceed the specified sizes. Integrate it with a CI to get notified when the app becomes too large:



To configure it:

Find out the maximum sizes

1. Optimize the app to make it as small as possible. Run the production build.
2. Add the **`bundlesize`** section into **`package.json`** with the following content:

```
// package.json
{
  "bundlesize": [
    {
      "path": "./dist/*"
    }
  ]
}
```

3. Execute **`bundlesize`** with npx:

```
npx bundlesize
```

This will print the gzipped size of each file:

```
PASS  ./dist/icon256.6168aaac8461862eab7a.png: 10.89KB
PASS  ./dist/icon512.c3e073a4100bd0c28a86.png: 13.1KB
PASS  ./dist/main.0c8b617dfc40c2827ae3.js: 16.28KB
PASS  ./dist/vendor.ff9f7ea865884e6a84c8.js: 31.49KB
```

4. Add 10-20% to each size, and you'll get the maximum sizes. This 10-20% margin would let you develop the app as usual while warning you when its size grows too much.

Enable `bundlesize`

5. Install the **`bundlesize`** package as a development dependency:

```
npm install bundlesize --save-dev
```

6. In the **`bundlesize`** section in the **`package.json`**, specify the concrete maximum sizes. For some files (e.g., images), you might want to specify the maximum size per file type, not per each file:

```
// package.json
{
  "bundlesize": [
    {
      "path": "./dist/*.png",
      "maxSize": "16 kB",
    },
    {
      "path": "./dist/main.*.js",
      "maxSize": "20 kB",
    },
    {
      "path": "./dist/vendor.*.js",
      "maxSize": "35 kB",
    }
  ]
}
```

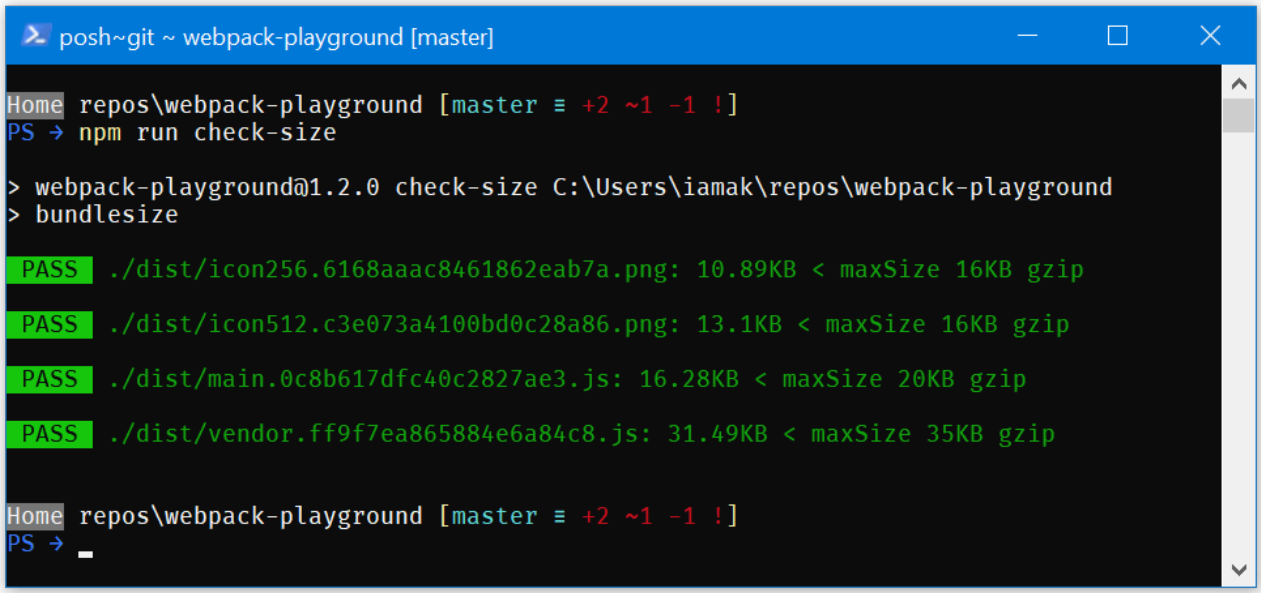
7. Add an npm script to run the check:

```
// package.json
{
  "scripts": {
    "check-size": "bundlesize"
```

```
    }
  }
```

8. Configure the CI to execute `npm run check-size` on each push. (And integrate `bundlesize` with GitHub if you're developing the project on it.)

That's it! Now, if you run `npm run check-size` or push the code, you'll see if the output files are small enough:



Or, in case of failures:

```
posh~git ~ webpack-playground [master]

Home repos\webpack-playground [master ≡ +2 ~1 -1 !]
PS → npm run check-size

> webpack-playground@1.2.0 check-size C:\Users\iamak\repos\webpack-playground
> bundlesize

 PASS  ./dist/icon256.6168aaac8461862eab7a.png: 10.89KB < maxSize 16KB gzip

 PASS  ./dist/icon512.c3e073a4100bd0c28a86.png: 13.1KB < maxSize 16KB gzip

 PASS  ./dist/main.0c8b617dfc40c2827ae3.js: 16.28KB < maxSize 20KB gzip

 FAIL  ./dist/vendor.ff9f7ea865884e6a84c8.js: 37.17KB > maxSize 35KB gzip

npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! webpack-playground@1.2.0 check-size: `bundlesize`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the webpack-playground@1.2.0 check-size script.
npm ERR! This is probably not a problem with npm. There is likely additional logging
 output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     C:\Users\iamak\AppData\Roaming\npm-cache\_logs\2017-11-29T21_20_02_646Z
-debug.log

Home repos\webpack-playground [master ≡ +2 ~1 -1 !]
PS → _
```

Further reading

- Alex Russell about the real-world loading time we should target

Analyze why the bundle is so large

You might want to dig deeper into the bundle to see what modules take space in it. Meet webpack-bundle-analyzer:

(Screen recording from github.com/webpack-contrib/webpack-bundle-analyzer)

webpack-bundle-analyzer scans the bundle and builds a visualization of what's inside it. Use this visualization to find large or unnecessary dependencies.

To use the analyzer, install the `webpack-bundle-analyzer` package:

```
npm install webpack-bundle-analyzer --save-dev
```

add a plugin to the webpack config:

```
// webpack.config.js
const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlu

module.exports = {
  plugins: [
    new BundleAnalyzerPlugin(),
  ],
};
```

and run the production build. The plugin will open the stats page in a browser.

By default, the stats page shows the size of parsed files (i.e., of files as they appear in the bundle). You'll likely want to compare gzip sizes since that's closer to what real users experience; use the sidebar on the left to switch the sizes.

Note: If you use the ModuleConcatenationPlugin, it might merge a part of modules in the webpack-bundle-analyzer output, making the report less detailed. If you use this plugin, disable it during the analysis.

Here's what to look for in the report:

- **Large dependencies.** Why are they so large? Are there smaller alternatives (e.g., Preact instead of React)? Do you use all the code it includes (e.g., Moment.js includes a lot of locales that are often not used and could be dropped)?
- **Duplicated dependencies.** Do you see the same library repeating in multiple files? (Use, e.g., the `optimization.splitChunks.chunks` option – in webpack 4 – or the `CommonsChunkPlugin` – in webpack 3 – to move it into a common file.) Or does the bundle have multiple versions of the same library?
- **Similar dependencies.** Are there similar libraries that do approximately the same job? (E.g. `moment` and `date-fns`, or `lodash` and `lodash-es`.) Try sticking with a single tool.

Also, check out Sean Larkin's great analysis of webpack bundles.

Summing up

- Use `webpack-dashboard` and `bundlesize` to stay tuned of how large your app is
- Dig into what builds up the size with `webpack-bundle-analyzer`

Previous
← Make Use of Long-term Caching

Next
Conclusion →

Last updated July 2, 2018.