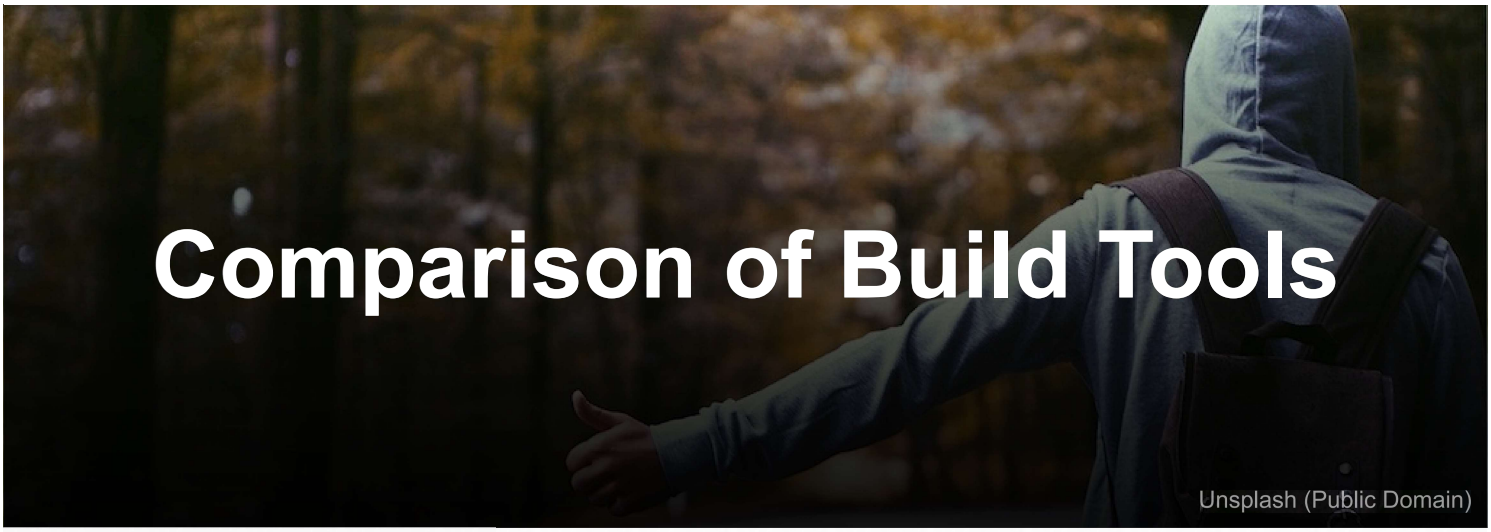

RelativeCI - In-depth bundle stats analysis and monitoring - Interview with Viorel Cojocaru

Back in the day, it was enough to concatenate scripts together. Times have changed, though, and distributing your JavaScript code can be a complicated endeavor. This problem has escalated with the rise of single-page applications (SPAs) as they tend to rely on many big libraries. For this reason, many loading strategies exist. The basic idea is to defer loading instead of loading all at once.

The popularity of Node and npm, its package manager, provide more context. Before npm became popular, it was hard to consume dependencies. There was a period when people developed frontend specific

Search

Table of Contents

package managers, but npm won in the end. Now dependency management is more comfortable than before, although there are still challenges to overcome.

> Tooling.Report provides a feature comparison of the most popular build tools.

Task runners

Historically speaking, there have been many build tools. *Make* is perhaps the best known, and it's still a viable option. Specialized *task runners*, such as Grunt and Gulp were created particularly with JavaScript developers in mind. Plugins available through npm made both task runners powerful and extendable. It's possible to use even npm `scripts` as a task runner. That's common, particularly with webpack.

Make

Make goes way back, as it was initially released in 1977. Even though it's an old tool, it has remained relevant. Make allows you to write separate tasks for various purposes. For instance, you could have different tasks for creating a production build, minifying your JavaScript or running tests. You can find the same idea in many other tools.

Even though Make is mostly used with C projects, it's not tied to C in any way. James Coglan discusses in detail how to use Make

with JavaScript. Consider the abbreviated code based on James' post below:

Makefile

```
PATH  := node_modules/.bin:$(PATH)
SHELL := /bin/bash

source_files := $(wildcard
lib/*.coffee)
build_files  :=
$(source_files:%.coffee=build/%.js)
app_bundle   := build/app.js
spec_coffee  := $(wildcard
spec/*.coffee)
spec_js      :=
$(spec_coffee:%.coffee=build/%.js)

libraries    := vendor/jquery.js

.PHONY: all clean test

all: $(app_bundle)

build/%.js: %.coffee
    coffee -co $(dir $@) $<

$(app_bundle): $(libraries)
$(build_files)
    uglifyjs -cmo $@ $^

test: $(app_bundle) $(spec_js)
    phantomjs phantom.js

clean:
    rm -rf build
```

With Make, you model your tasks using Make-specific syntax and terminal commands making it possible to integrate with webpack.

npm `scripts` as a task runner

Even though npm CLI wasn't primarily designed to be used as a task runner, it works as such thanks to `package.json` `scripts` field. Consider the example below:

package.json

```
{
  "scripts": {

    "start": "wp --mode development",
    "build": "wp --mode production",
    "build:stats": "wp --mode produc-
tion --json > stats.json"
  }
}
```

These scripts can be listed using `npm run` and then executed using `npm run <script>`. You can also namespace your scripts using a convention like `test:watch`. The problem with this approach is that it takes care to keep it cross-platform.

Instead of `rm -rf`, you likely want to use utilities such as rimraf and so on. It's possible to invoke other tasks runners here to

hide the fact that you are using one. This way you can refactor your tooling while keeping the interface as the same.

Grunt

Grunt was the first famous task runner for frontend developers. Its plugin architecture contributed towards its popularity. Plugins are often complicated by themselves. As a result, when configuration grows, it can become tricky to understand what's going on.

Here's an example from Grunt documentation. In this configuration, you define a linting and watcher tasks. When the *watch* task gets run, it triggers the *lint* task as well. This way, as you run Grunt, you get warnings in real-time in the terminal as you edit the source code.

Gruntfile.js

```
module.exports = (grunt) => {
  grunt.initConfig({
    lint: {
      files: ["Gruntfile.js",
"src/**/*.js", "test/**/*.js"],
      options: {
        globals: {
          jQuery: true,
        },
      },
    },
    watch: {
      files: ["<%= lint.files %>"],
```

```
      tasks: ["lint"],
    },
  });

  grunt.loadNpmTasks("grunt-contrib-
jshint");
  grunt.loadNpmTasks("grunt-contrib-
watch");

  grunt.registerTask("default",
["lint"]);
};
```

In practice, you would have many small tasks for specific purposes, such as building the project. An essential part of the power of Grunt is that it hides a lot of the wiring from you.

Taken too far, this can get problematic. It can become hard to understand what's going on under the hood. That's the architectural lesson to take from Grunt.

grunt-webpack plugin allows you to use webpack in a Grunt environment while you leave the heavy lifting to webpack.

Gulp

Gulp takes a different approach. Instead of relying on configuration per plugin, you deal with actual code. If you are familiar with Unix and piping, you'll like Gulp. You have *sources* to match files, *filters* to oper-

ate on these sources, and *sinks* to pipe the build results.

Here's an abbreviated sample *Gulpfile* adapted from the project's README to give you a better idea of the approach:

Gulpfile.js

```
const gulp = require("gulp");
const coffee = require("gulp-coffee");
const concat = require("gulp-concat");
const uglify = require("gulp-uglify");
const sourcemaps = require("gulp-
sourcemaps");
const del = require("del");

const paths = {
  scripts: [
    "client/js/**/*.coffee",
    "!client/external/**/*.coffee",
  ],
};

// Not all tasks need to use streams.
// A gulpfile is another node program
// and you can use all packages avail-
able on npm.
gulp.task("clean", () =>
del(["build"]));
gulp.task("scripts", ["clean"], () =>
  // Minify and copy all JavaScript
(except vendor scripts)
  // with source maps all the way
down.
  gulp
```

```
    .src(paths.scripts)
    // Pipeline within pipeline
    .pipe(sourcemaps.init())
    .pipe(coffee())
    .pipe(uglify())
    .pipe(concat("all.min.js"))
    .pipe(sourcemaps.write())
    .pipe(gulp.dest("build/js"))
);
gulp.task("watch", () =>
gulp.watch(paths.scripts,
["scripts"]));

// The default task (called when you
run `gulp` from CLI).
gulp.task("default", ["watch",
"scripts"]);
```

Given the configuration is code, you can always hack it if you run into troubles. You can wrap existing Node packages as Gulp plugins, and so on. Compared to Grunt, you have a clearer idea of what's going on. You still end up writing a lot of boilerplate for casual tasks, though. That is where newer approaches come in.

webpack-stream allows you to use webpack in a Gulp environment.

Script loaders

For a while, RequireJS, a script loader, was popular. The idea was to provide an asynchronous module definition and build on

top of that. Fortunately, the standards have caught up, and RequireJS seems more like a curiosity now.

RequireJS

RequireJS was perhaps the first script loader that became genuinely popular. It gave the first proper look at what modular JavaScript on the web could be. Its greatest attraction was AMD. It introduced a `define` wrapper:

```
define(["./MyModule.js"], function
(MyModule) {
  return function() {}; // Export at
module root
});

// or
define(["./MyModule.js"], function
(MyModule) {
  return {
    hello: function() {...}, // Export
as a module function
  };
});
```

Incidentally, it's possible to use `require` within the wrapper:

```
define(["require"], function (re-
quire) {
  var MyModule =
require("./MyModule.js");
```

```
  return function() {...};
});
```

This latter approach eliminates a part of the clutter. You still end up with code that feels redundant. ES2015 and other standards solve this.

Jamund Ferguson has written an excellent blog series on how to port from RequireJS to webpack.

JSPM

Using JSPM is entirely different than previous tools. It comes with a command-line tool of its own that is used to install new packages to the project, create a production bundle, and so on. It supports SystemJS plugins that allow you to load various formats to your project.

Bundlers

Task runners are great tools on a high level. They allow you to perform operations in a cross-platform manner. The problems begin when you need to splice various assets together and produce bundles. *bundlers*, such as Browserify, Brunch, or webpack, exist for this reason and they operate on a lower level of abstraction. Instead of operating on files, they operate on modules and assets.

Browserify

Dealing with JavaScript modules has always been a bit of a problem. The language itself didn't have the concept of modules till ES2015. Ergo, the language was stuck in the '90s when it comes to browser environments. Various solutions, including AMD, have been proposed.

Browserify is one solution to the module problem. It allows CommonJS modules to be bundled together. You can hook it up with Gulp, and you can find smaller transformation tools that allow you to move beyond the basic usage. For example, watchify provides a file watcher that creates bundles for you during development saving effort.

The Browserify ecosystem is composed of a lot of small modules. In this way, Browserify adheres to the Unix philosophy. Browserify is more comfortable to adopt than webpack, and is, in fact, a good alternative to it.

> Splittable is a Browserify wrapper that allows code splitting, supports ES2015 out of the box, tree shaking, and more. bankai is another option to consider.

Brunch

Compared to Gulp, Brunch operates on a higher level of abstraction. It uses a declarative approach similar to webpack's. To give

you an example, consider the following configuration adapted from the Brunch site:

```
module.exports = {
  files: {
    javascripts: {
      joinTo: {
        "vendor.js": /^(?!app)/,
        "app.js": /^app/,
      },
    },
    stylesheets: {
      joinTo: "app.css",
    },
  },
  plugins: {
    babel: {
      presets: ["react", "env"],
    },
    postcss: {
      processors: [require("autopre-
fixer")],
    },
  },
};
```

Brunch comes with commands like `brunch new`, `brunch watch --server`, and `brunch build --production`. It contains a lot out of the box and can be extended using plugins.

Rollup

Rollup focuses on bundling ES2015 code. *Tree shaking* is one of its selling points and

it supports code splitting as well. You can use Rollup with webpack through rollup-loader.

vite is an opinionated wrapper built on top of Rollup and it has been designed especially with Vue 3 in mind. nollup is another wrapper and it comes with features like Hot Module Replacement out of the box.

Webpack

You could say webpack takes a more unified approach than Browserify. Whereas Browserify consists of multiple small tools, webpack comes with a core that provides a lot of functionality out of the box.

Webpack core can be extended using specific *loaders* and *plugins*. It gives control over how it *resolves* the modules, making it possible to adapt your build to match specific situations and workaround packages that don't work correctly out of the box.

Compared to the other tools, webpack comes with initial complexity, but it makes up for this through its broad feature set. It's an advanced tool that requires patience. But once you understand the basic ideas behind it, webpack becomes powerful.

To make it easier to use, tools such as create-react-app, poi, and instapack have been built around it.

Vite

Vite is tool comparable to webpack. It comes with features like lazy loading, ESM, JSX, and TypeScript support out of the box. The build functionality relies on Rollup and the development server is custom code. Originally it was developed with Vue in mind but since the scope of the tool has grown to support popular frameworks like React. It's possible to extend the tool using Vite specific plugins and also Rollup plugins are supported making it a versatile solution.

Zero configuration bundlers

There's a whole category of *zero configuration* bundlers. The idea is that they work out of the box without any extra setup. Parcel is perhaps the famous of them.

FuseBox is a bundler focusing on speed. It uses a zero-configuration approach and aims to be usable out of the box.

These tools include microbundle, bili, as-bundle, and tsdx.

Other Options

You can find more alternatives as listed below:

- Rome is an entire toolchain built around the problems of linting, compiling, and bundling.
- esbuild is a performance-oriented bundler written in Go.
- AssetGraph takes an entirely different approach and builds on top of HTML semantics making it ideal for hyperlink analysis or structural analysis. webpack-assetgraph-plugin bridges webpack and AssetGraph together.
- StealJS is a dependency loader and a build tool focusing on performance and ease of use.
- Blendid is a blend of Gulp and bundlers to form an asset pipeline.
- swc is a JavaScript/TypeScript compiler focusing on performance written in Rust.
- Packem is another Rust based option for bundling JavaScript.
- Sucrase is a light JavaScript/TypeScript compiler focusing on performance and recent language features.

Conclusion

Historically there have been a lot of build tools for JavaScript. Each has tried to solve a specific problem in its way. The standards have begun to catch up, and less effort is re-

quired around basic semantics. Instead, tools can compete on a higher level and push towards better user experience. Often you can use a couple of separate solutions together.

To recap:

- **Task runners** and **bundlers** solve different problems. You can achieve similar results with both, but often it's best to use them together to complement each other.
- Older tools, such as Make or RequireJS, still have influence even if they aren't as popular in web development as they once were.
- Bundlers like Browserify or webpack solve an important problem and help you to manage complex web applications.
- Emerging technologies approach the problem from different angles. Sometimes they build on top of other tools, and at times they can be used together.

Subscribe to the blog updates

If you enjoyed this page, consider subscribing to the mailing list below or following @survivejs for

occasional updates. There is also RSS available for old beards (no pun intended).

Previous chapter
← Appendices

Next chapter
Hot Module Replacement →

Danny Alexander
5 years ago

1. How old is this?

2. And is there a reason there is not a published or last edited date somewhere obvious?

1 0 Reply

Juho Vepsäläinen Mod
Danny Alexander 5 years ago

1. It's from a year ago based on git history.

2. It's a technical issue. I haven't gotten around to associating the book chapters to the git data correctly yet.

I would say the main change that has happened is that Parcel became stronger (esp. with version 2) and Fusebox will look strong with version 4. A lot of wrappers appeared for bundlers so often developers don't even have to touch them these days.

2 0 Reply

Danny Alexander
Juho Vepsäläinen
5 years ago

Thanks, good to know.

4 0 Reply

Dmitri

8 years ago

In my opinion, and I have seen it confirmed by many people, Webpack is adding more pain than needed by replacing the clean readable Gulp JS code with unreadable chunks of configuration code that will bloat the more you use its power. Then we are back to Make :)

1 0 Reply

Juho Vepsäläinen **Mod**

Dmitri 8 years ago

Yeah, it brings problems of its own even though it can be very powerful.

This is something we want to tackle in Webpack 2. We've been talking about the topic recently at Gitter if you want to catch up. Basically it comes down to a couple of factors: ease of use (think Babel presets, but for Webpack), better warnings/errors (linting, DSLs on top of config?), neater configuration (see concord specification). I hope Webpack 2 can resolve some of the major pain points.

One of the ironies of the current situation is that we know how to package JavaScript, and we've got even a standard module definition now, but we don't have standard means to consume it. SystemJS will likely help on the browser side, but npm packaging remains a wild west for now.

Perhaps the situation will change during this year (2016), but for now it's going to be a little chaotic as people try to figure out good solutions for these common problems.

Thanks for your comments by the way. I'll get to them during this day. :)

1 0 Reply

Dmitri

Juho Vepsäläinen

8 years ago

Thanks for your answer and headup on Webpack 2, hopefully it will make

better use of Gulp. It is awesome when it works but a real nightmare when there is a slightest error. So many repos out there are broken because of it. I'll check on Gitter.

o o Reply

Juho Vepsäläinen —

Mod

Dmitri

8 years ago

Be sure to scroll back a day or two. I think I'll do a little experiment with presets for Webpack this week after I get a new release of the book out there. :)

o o Reply

Dmitri —

Juho Vepsäläinen

8 years ago

It is quite overwhelming!
I see it is focusing on the Grunt-style config, where Gulp had proven to be simpler.
Will history repeat? :)

o o Reply

Juho Vepsäläinen —

Mod

Dmitri

8 years ago

I guess this year will show that.

My immediate concerns are in developing a setup that's more maintainable than the current one. I have a good feeling about presets although that won't be a

perfect solution and at worst just obscures certain aspects further.

0 0 Reply

Dmitri

Juho Vepsäläinen

8 years ago

edited

Are there any relevant Github threads? I find them much easier to navigate and use than Gitter somehow.

0 0 Reply

Juho Vepsäläinen

Mod

Dmitri

8 years ago

Apart from the concord specification it's all very informal at the moment.

0 0 Reply

Dmitri

Juho Vepsäläinen

8 years ago

BTW, are you aware why is the config in `webpack.config.js` not shortened to `conf` as in many other packages? It caused me some headache as webpack would never complain about its config file absence.

0 0 Reply

Juho Vepsäläinen

Mod

Dmitri
8 years ago

It picks it up through convention. There's actually a separate --config parameter that can be used to pass something else to the CLI tool. Some people use that to deal with multiple configurations (file per configuration).

o o Reply

Dmitri
Juho Vepsäläinen
8 years ago

I mean if I misprint the name, it silently ignores it and only prints "Output filename not configured.", which leads me to think I forgot to declare the output, where in reality the reason was the file name. Can really drive you mad :(

o o Reply

Juho Vepsäläinen
Mod
Dmitri
8 years ago

Ah, like that.

Feel free to open an issue. Something like that shouldn't be hard to fix. Good first PR for someone. :)

o o Reply

Dmitri
Juho Vepsäläinen
8 years ago

True, but with 500+ open issues I don't want to add a trivial one, esp. given that it moves to a new version as you said.

0 0 Reply

Joe Privett

8 years ago

Could you simply just use Meteor?

1 1 Reply

Juho Vepsäläinen **Mod**

Joe Privett 8 years ago

Meteor is actually embracing React now. They have taken a very exciting direction and it will be cool to see how the project progresses this year.

2 0 Reply

Harry Moreno

6 years ago

Can you add https://github.com/choojs/b... to the list. I'm curious how it compares.

0 0 Reply

Juho Vepsäläinen **Mod**

Harry Moreno 6 years ago

Thanks for the tip! I added it to my list and the tool will make it to the book eventually (likely soon as I'm active with the book atm).

0 0 Reply

englishextra

7 years ago edited

Most of the tools mentioned above were released firstly for the purpose of self-advertising, and maybe for the devs to ease their workflow. That's why we now have such a hodge-podge of cool compilers/bundlers/blablabla, and these tools eat computers resources very impudently.

In my case I use babel/gulp/CLI, and avoid loaders and

never include 3rd-parties in the bundle. I get them when needed via fetch/xhr/script

0 0 Reply

MiltonDValler

7 years ago

Yarn? Where does that fit in now? T.You.

0 0 Reply

Juho Vepsäläinen Mod

MiltonDValler 7 years ago

Hi,

Yarn can be seen as a more performant replacement for npm. It fixes certain long standing weaknesses (working lockfile, offline usage). I recall Yehuda Katz (one of the main authors) had proposed these to npm a few years ago, but as nothing happened Yarn was born.

I hope these changes make back to npm and it goes like with iojs and Node.js in the past. That would be the ideal situation and avoid fragmentation.

My understanding is that Yarn may still have some weaknesses. I know it has had issues with private repositories. At least a while ago proper support was missing.

0 0 Reply

Vladimir Maxymenko

7 years ago

Grate basic knowledge to get by comparison. Wouldn't it be fair to include Webpack 2 in this list?

0 0 Reply

Juho Vepsäläinen Mod

Vladimir Maxymenko 7 years ago

The book has been updated to webpack 2. A lot of other goodies too.

1 0 Reply

Vladimir Maxymenko

Juho Vepsäläinen

7 years ago

Thanks, Juho!

0 0 Reply

Juho Vepsäläinen Mod

Vladimir Maxymenko 7 years ago

Yeah, once we get to final release of webpack 2, I'll get the comparison up to date. Biggest change - proper support for ES6 modules. See here for the rest.

0 0 Reply

S

SaintScott

8 years ago

It would be better provide a link for each tool to their repos or official website

0 0 Reply

Juho Vepsäläinen Mod

SaintScott 8 years ago

I can definitely add links to the projects. Thanks.

1 0 Reply

Warren

8 years ago

Should be said about webpack that the authors have up until now refused to document what they're doing to the level we're used to seeing with most other open source projects. I'm not sure if it's because they aren't proud of their work and don't want to talk about it, or if there's a language barrier, or what.... but the reality is:

- There are no CHANGELOG files anywhere
- They do not use Github's milestones or other public release tracking systems
- There is no blog where the authors communicate their intent or motivations for making changes

Almost everything I'm hearing about webpack 2 is from outside the team, which is really weird and makes me uncomfortable with upgrading my hard-won webpack 1 builds. As it is, I'm already having to troll through the commit history of every loader and library just to figure out what's changed from one point release to the next.....

0 0 Reply

Juho Vepsäläinen Mod

Warren 8 years ago

Hi,

The author of webpack maintains a what's new document for webpack 2. There are various webpack 2 issues at the official tracker. I hope that helps.

I agree the project communication could be clearer. The migration path isn't easy due to the reasons you mentioned.

Given it's a one person project, with occasional external contributor, there's only so much that one person can do. And I'm fairly sure this isn't Tobias' main job. That said, the situation isn't ideal from the user point of view (frustrating for book authors too).

2 0 Reply

Juho Vepsäläinen Mod

Warren 7 years ago

Hi,

A couple of interesting things have changed since the comment. There's an official blog now. Changes between releases are documented now. And the project has a new site.

I think we've taken a couple of steps towards a good direction. :)

0 0 Reply

This comment was deleted.

Juho Vepsäläinen Mod

Guest 8 years ago edited

Fly seems to be doing well. I'll link to it. Thanks. :)

1 0 Reply

Jebbie

8 years ago

Short; i'm loving your articles, expertise and books - great resource of excellent knowledge :)

But the discussion/decisions in the company i'm working on gives me actually headaches.. So the team decided for angular2, the strict use of system.js and gulp.

Now i got the task to rework our current angular2 seed project to get rid of webpack (my heart is crying but, ok...).

The spirit is like this:

"Webpack/Browserify/JSPM - all of them will be obselete in a near future because ng2 has set system.js as their default and somewhere in 2 years we can drop system.js because browser will understand typescript natively then"

see more

0 0 Reply

Juho Vepsäläinen **Mod**

Jebbie 8 years ago

I think we will see stronger adoption of SystemJS in the future. Webpack 2 implements the semantics and it will surely help with the situation once it gets out.

We are somewhere between two different worlds right now. Techniques that make sense for HTTP/1 are anti-patterns for HTTP/2. In HTTP/1 we want to minimize roundtrips (hence bundling), but in HTTP/2 it doesn't matter. We just set up a pipe a let it stream assets.

The question is how soon this transition from HTTP/1 to HTTP/2 will actually take. The browsers will have to catch up and the technology has to mature.

Speaking of Webpack in particular, I think we'll

see more

0 0 Reply

Avata This comment was deleted.

Juho Vepsäläinen **Mod**

Guest 8 years ago

That can be a valid approach. Gulp is better especially for higher level tasks. There's good integration as linked above.

Personally I use npm `scripts` and Webpack in tandem and so far that has worked well.

The advantage of running other assets beyond js through Webpack is that you can benefit from automatic file name hashing etc.

0 0 Reply

Sergey

Juho Vepsäläinen

7 years ago

Hi Juho, thanks for putting things together.

>>Personally I use npm scripts and Webpack in tandem and so far that has worked well.
I'm leaning to this solution, but I wonder if your opinion changed since? If it did, why?

Thank you.

0 0 Reply

Juho Vepsäläinen **Mod**

Sergey

7 years ago

Hi Serge,

npm scripts is definitely the way to go for me. I support just Unix myself (cross-platform is possible too) and I don't do anything super complex to warrant a heavier solution.

0 0 Reply

This comment was deleted.

Juho Vepsäläinen

Mod

Guest

8 years ago

It was in a tip. Here we go, gulp-webpack.

3 0 Reply

This comment was deleted.

Juho Vepsäläinen

Mod Guest

8 years ago

No worries. :)

0 0 Reply

Lars Jeppesen

8 years ago edited

I love webpack and the bundling.
However I still need tools like Gulp to manage all the other things.. copying files, making deployments etc etc.. so I don't think Webpack only is gonna cut it for a real setup.. but definitely bundling is the way to go...

0 0 Reply

Juho Vepsäläinen **Mod**

Lars Jeppesen 8 years ago

Yeah, task runners still have their place. Having only Webpack would be asking for trouble in more complex scenarios. I'll clarify this point in the next edition.

Thanks for the comment!

From the Blog

- RelativeCI - In-depth bundle stats analysis and monitoring - Interview with Viorel Cojocaru
- Future Frontend - A new conference to reimagine the future of the frontend - Interview with Juho Vepsäläinen
- Console Cat - Privacy-friendly CLI telemetry in less than five minutes - Interview with Matt Evenson
- Nhost - Backend-as-a-Service with GraphQL for modern app development - Interview with Johan Eliasson
- How to tame the devDependencies of your project?
- Medusa - Own your ecommerce stack - Interview with Sebastian Rindom
- iHateReading - Where knowledge is shared - Interview with Shrey Vijayvargiya

Need help?