Distribution Decisions

Cory House

@housecor | reactjsconsulting.com

The Plan

Key decisions

1. Closed, inner, or open source
2. Package hosting
3. Import approach
4. How to specify package files
5. Output formats
6. Documentation hosting

Decision:
Closed source, inner source,
or open source?

Consider open-sourcing your components

Open Source

Help the community

Recruiting

Public review = Find issues quickly

Free development

Closed Source

Privacy

Freedom

Include company-specific features

Consider “Inner Sourcing”

Walmart

Components are *internally* “open source”

No team owns a component

Everyone uses and contributes

Centralized

Single group “on the hook”

Avoid duplication

Inner sourced

No bottleneck

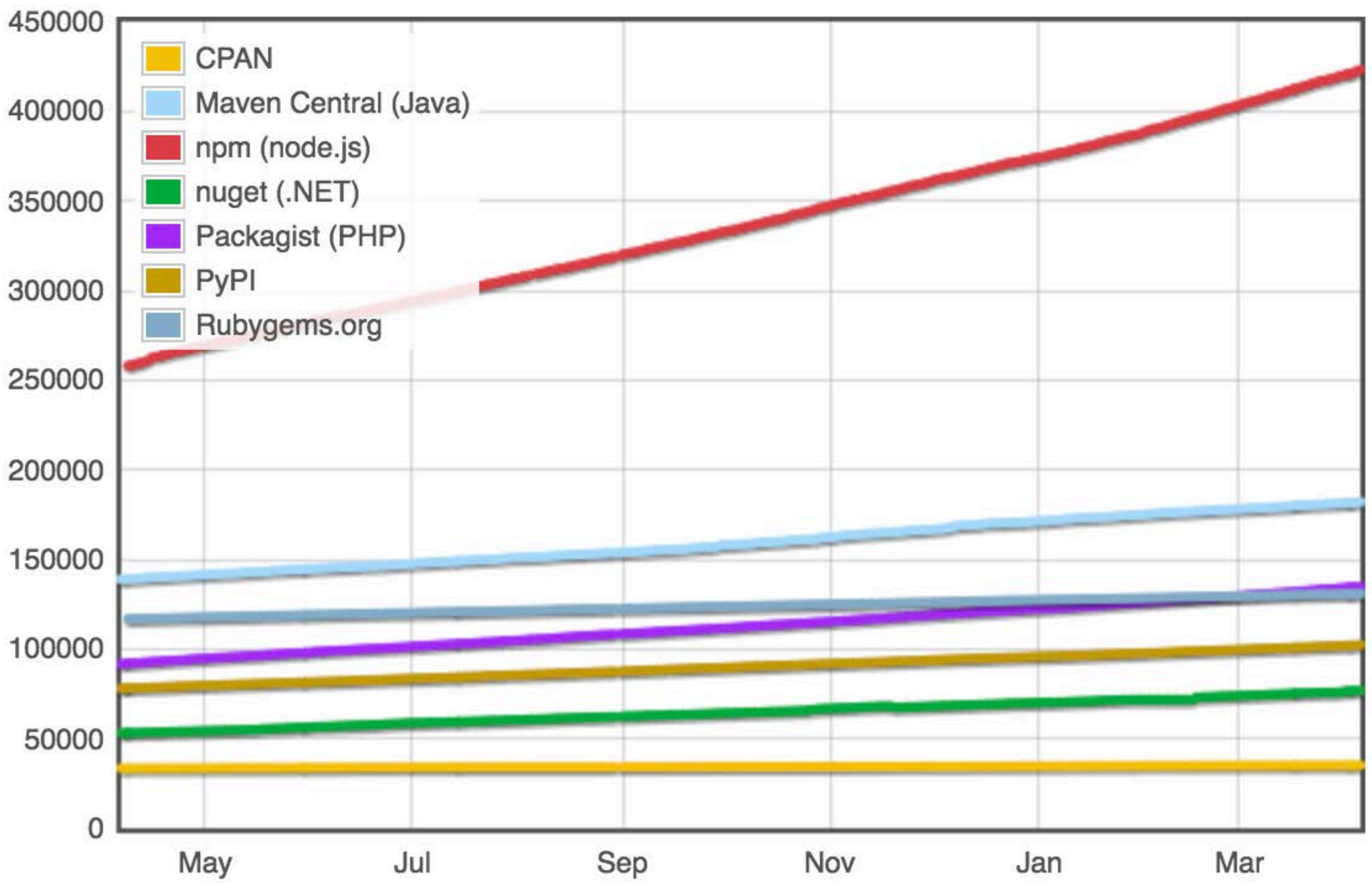
More involvement

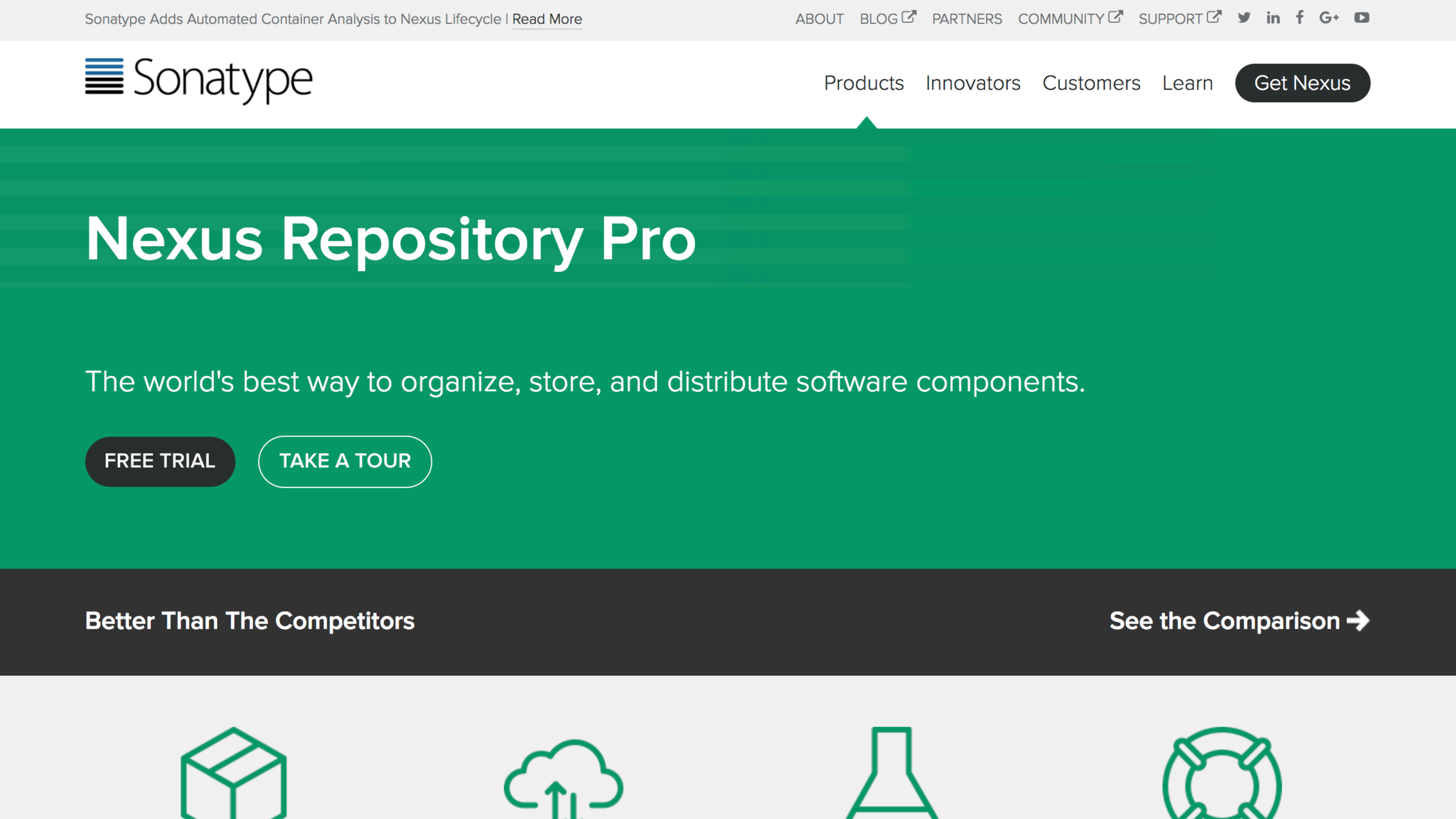
More investment

Decision:
Package hosting

npm

Module Counts

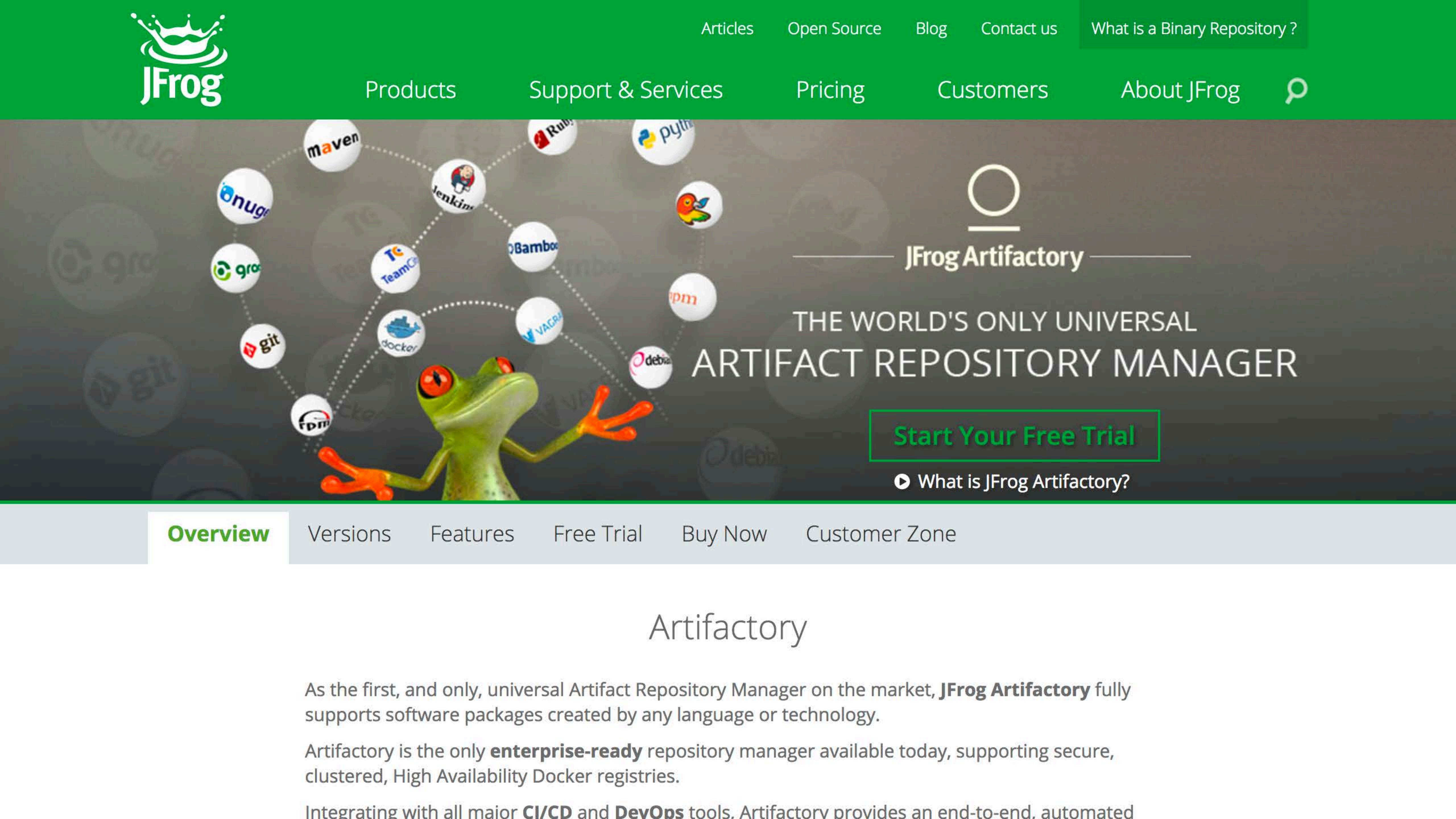




Nexus Repository Pro

The world's best way to organize, store, and distribute software components.

Better Than The Competitors

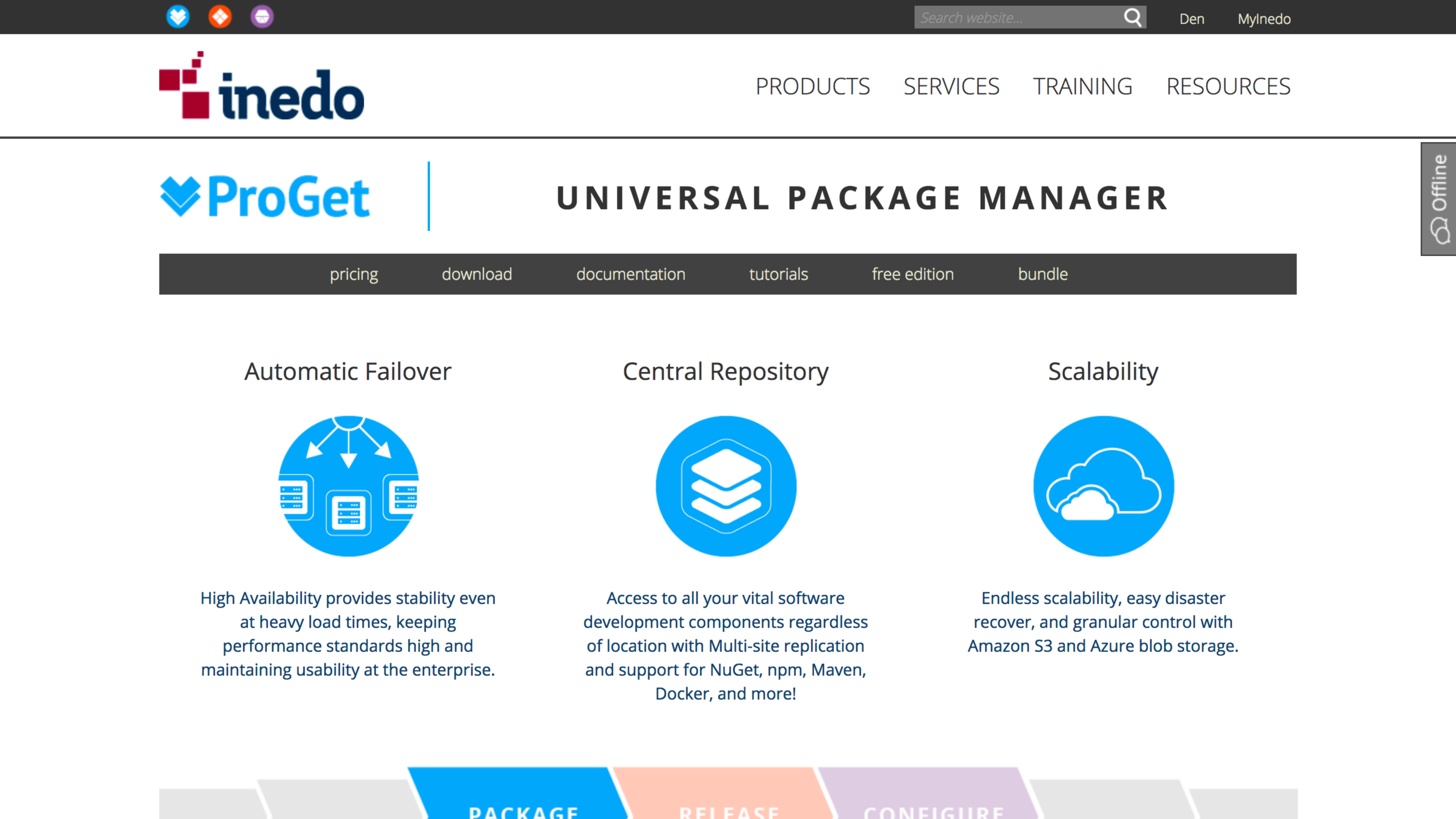


Artifactory

As the first, and only, universal Artifact Repository Manager on the market, **JFrog Artifactory** fully supports software packages created by any language or technology.

Artifactory is the only **enterprise-ready** repository manager available today, supporting secure, clustered, High Availability Docker registries.

Integrating with all major **CI/CD** and **DevOps** tools, Artifactory provides an end-to-end, automated



ProGet | UNIVERSAL PACKAGE MANAGER

Automatic Failover

High Availability provides stability even at heavy load times, keeping performance standards high and maintaining usability at the enterprise.

Central Repository

Access to all your vital software development components regardless of location with Multi-site replication and support for NuGet, npm, Maven, Docker, and more!

Scalability

Endless scalability, easy disaster recover, and granular control with Amazon S3 and Azure blob storage.

Universal Package Managers

Advantages

Complete control

Centralized asset hosting

Easier on-boarding / management

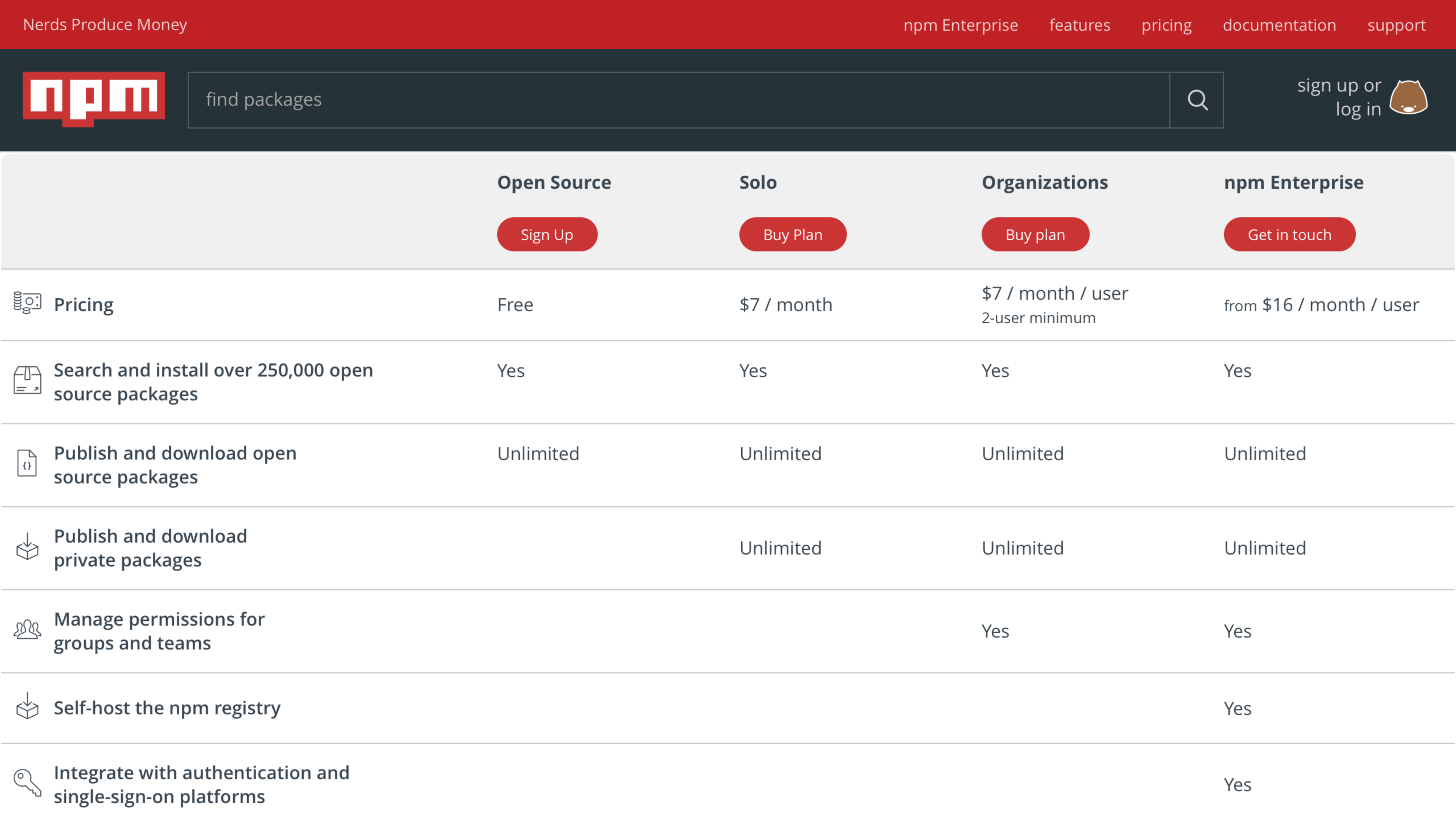
Avoid separate fees for each type

Disadvantages

Undiscoverable

Must grant access

Must configure registry setting



	Open Source	Solo	Organizations	npm Enterprise
	Sign Up	Buy Plan	Buy plan	Get in touch
Pricing	Free	$7 / month	$7 / month / user 2-user minimum	from $16 / month / user
Search and install over 250,000 open source packages	Yes	Yes	Yes	Yes
Publish and download open source packages	Unlimited	Unlimited	Unlimited	Unlimited
Publish and download private packages		Unlimited	Unlimited	Unlimited
Manage permissions for groups and teams			Yes	Yes
Self-host the npm registry				Yes
Integrate with authentication and single-sign-on platforms				Yes

Private namespace!

```
npm install @your-org/package
```

We’re going to publish our package publically to npm.

Decision:
Import Approach

Import Approaches

1. Named import

```
import {Label} from 'ps-react';
```

2. Import from /lib

```
import Label from 'ps-react/lib/Label';
```

3. Import from package root

```
import Label from 'ps-react/Label';
```

Approach 1: Named Import

```
import {Label} from 'ps-react';
```

1. Imports *everything* in ps-react
2. References the Label component as “Label”

- ✓ Most concise
- ✗ Imports entire library
- ✗ Bloats bundle ☹
- ✗ Need index.js at root

Approach 2: Import from /lib

```
import Label from 'ps-react/lib/Label';
```

- ✓ Imports single component
- ✓ No index.js at root needed
- ✓ Most common
- ✓ Simple build setup
- ✗ More typing

Approach 3: Import from Package Root

```
import Label from 'ps-react/Label';
```

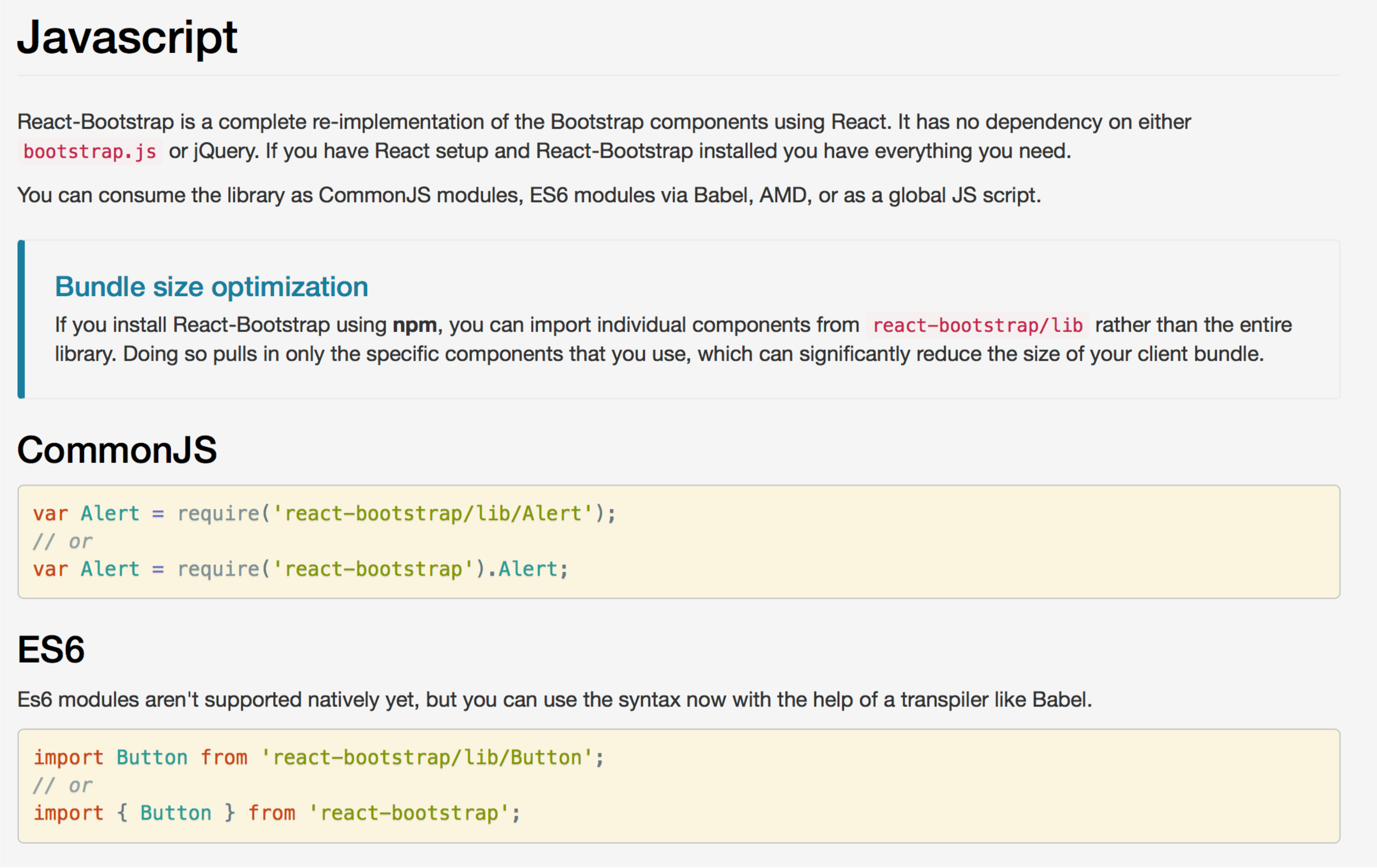
- ✓ Imports single component
- ✓ Concise - No /lib in path
- ✓ No index.js at root needed
- ✗ Custom build
  - Generate package.json
  - Copy assets
  - No .npmignore/files array
  - Run ”npm publish” from /lib

```
// Ant Design
import { Breadcrumb } from 'antd';


// Blueprint
import { Spinner } from "@blueprintjs/core";


// React Toolbox
import AppBar from 'react-toolbox/lib/app_bar';


// Material-UI:
import AppBar from 'material-ui/AppBar';
```



Javascript

React-Bootstrap is a complete re-implementation of the Bootstrap components using React. It has no dependency on either `bootstrap.js` or jQuery. If you have React setup and React-Bootstrap installed you have everything you need.

You can consume the library as CommonJS modules, ES6 modules via Babel, AMD, or as a global JS script.

> ### Bundle size optimization
>
> If you install React-Bootstrap using **npm**, you can import individual components from `react-bootstrap/lib` rather than the entire library. Doing so pulls in only the specific components that you use, which can significantly reduce the size of your client bundle.

CommonJS

```
var Alert = require('react-bootstrap/lib/Alert');
// or
var Alert = require('react-bootstrap').Alert;
```

ES6

Es6 modules aren't supported natively yet, but you can use the syntax now with the help of a transpiler like Babel.

```
import Button from 'react-bootstrap/lib/Button';
// or
import { Button } from 'react-bootstrap';
```

Import Approaches

1. Named import

```
import {Label} from 'ps-react';
```

2. Import from /lib

```
import Label from 'ps-react/lib/Label';
```

3. Import from package root

```
import Label from 'ps-react/Label';
```

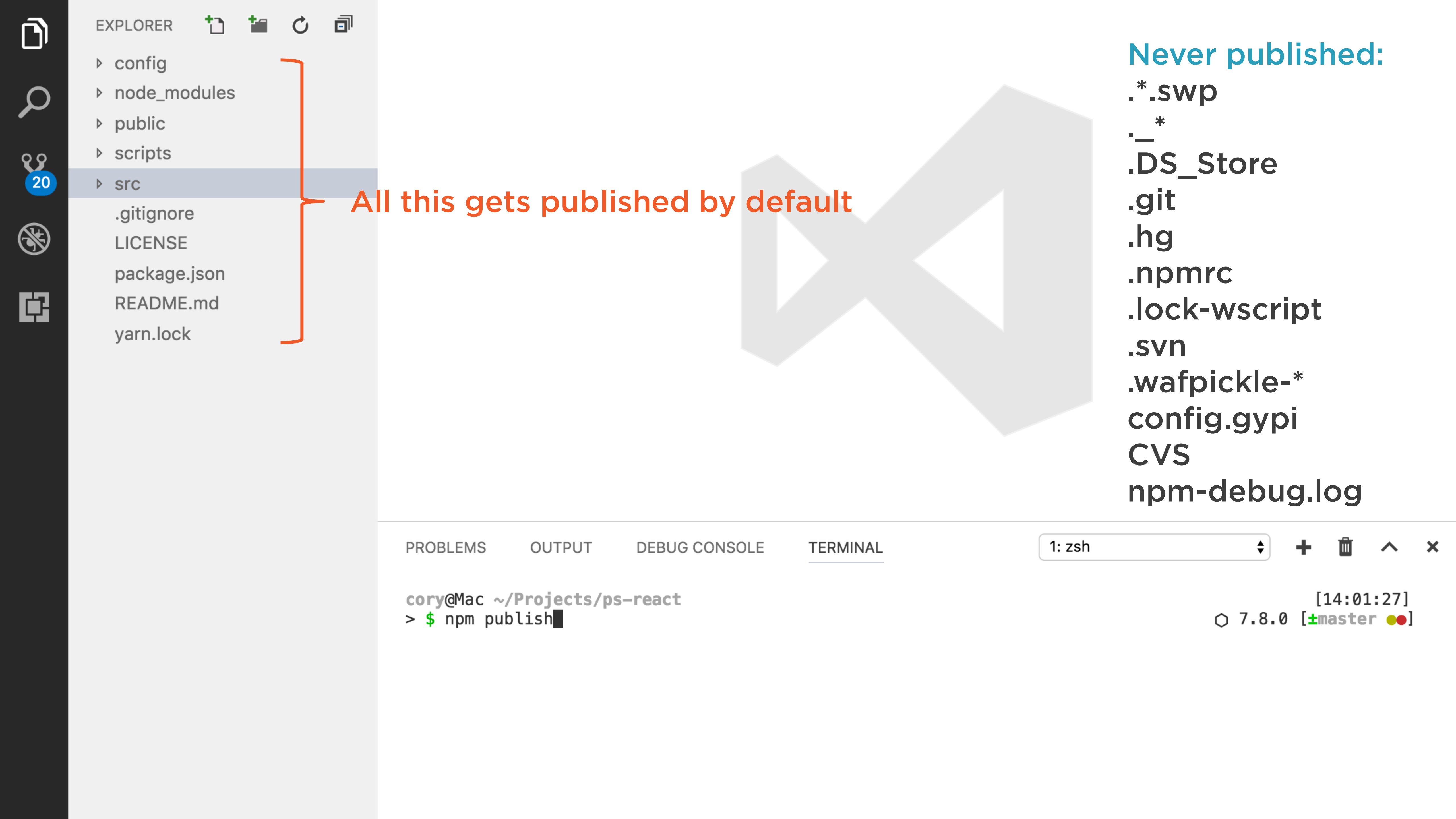
I’ll show how to support each approach

Decision:

How do I tell npm which files to publish?

“npm publish”

Publishes ^Nearly everything it finds in the directory where it’s run.



All this gets published by default

Never published:

.*.swp
._*
.DS_Store
.git
.hg
.npmrc
.lock-wscript
.svn
.wafpickle-*
config.gypi
CVS
npm-debug.log

```
cory@Mac ~/Projects/ps-react                                   [14:01:27]
> $ npm publish                                    ⬡ 7.8.0 [±master ●●]
```

Decision: Specifying npm Package Files

.npmignore

List files to ignore

files array in package.json

List files to publish

Dedicated folder

Copy files to publish

.npmignore

List files you DON’T want published

Place in project root

Comments start with #

```
#React Storybook Stories
Stories

#Uncompiled src code
src

#Create react app config and scripts
config
scripts

#The documentation build
docs

#Public dir for docs
public

#Editorconfig
.editorconfig
```

files array in package.json

```
"files": [
  "lib"
],
```

Why use files array?

Easy to understand
No extra file
No accidental publish
Less maintenance

Automatically included:

package.json
README (and its variants)
CHANGELOG (and its variants)
LICENSE / LICENCE

1. Write build to /lib
2. Write package.json to /lib
3. Copy relevant files to /lib
4. Run ”npm publish” from /lib

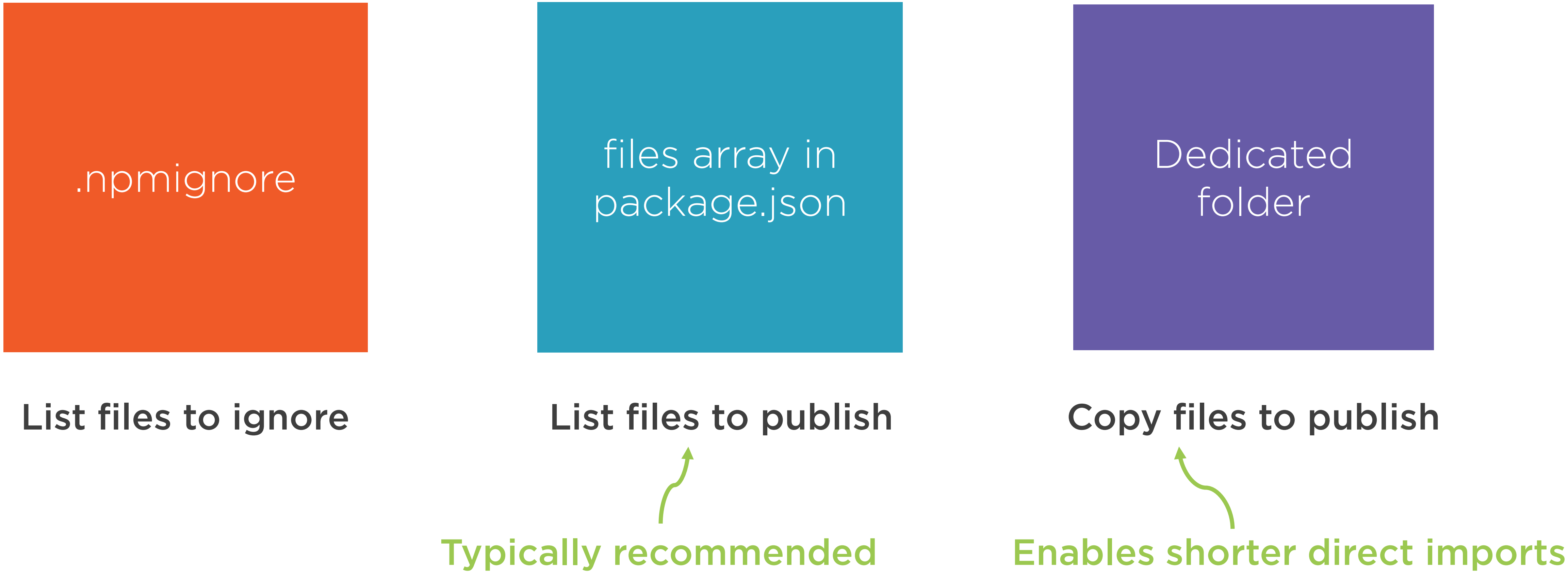
Dedicated folder

Why bother?

Writes components to package root
Enables short, direct imports:

```
import Label from 'ps-react/Label';
```

Decision: Specifying npm Package Files



.npmignore

List files to ignore

files array in package.json

List files to publish

Typically recommended

Dedicated folder

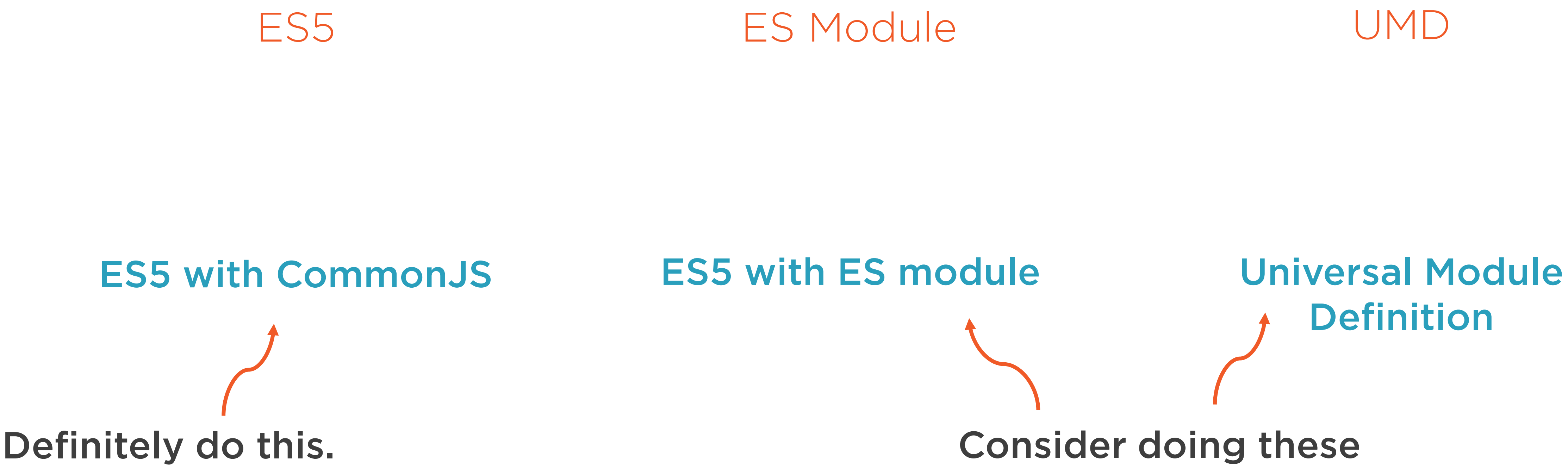
Copy files to publish

Enables shorter direct imports

In the final module, I’ll demonstrate the files array approach and the dedicated folder approach.

Decision:
Output formats

Potential Build Output Formats



ES5

ES Module

UMD

ES5 with CommonJS

ES5 with ES module

Universal Module Definition

Definitely do this.

Consider doing these

ES Module Build

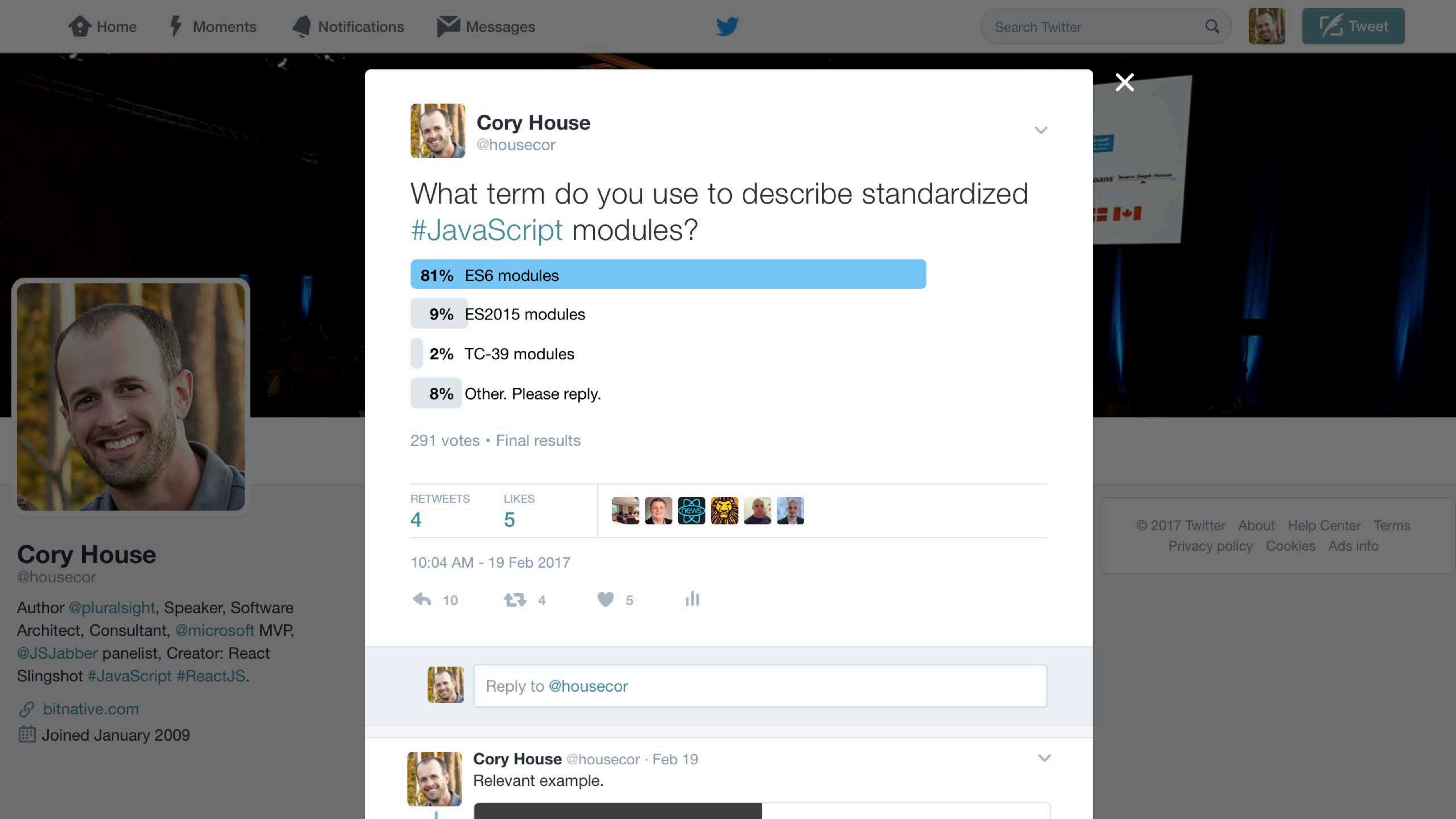
```
import react from 'react’

// code here

export default MyComponent;
```

People call this…

- ES6 Module
- ES2015 module
- TC-39 module
- JavaScript module
- EcmaScript module
- ES Module
- ESM
- Module



Cory House
@housecor

What term do you use to describe standardized #JavaScript modules?

81%	ES6 modules
9%	ES2015 modules
2%	TC-39 modules
8%	Other. Please reply.

291 votes • Final results

RETWEETS	LIKES
4	5

10:04 AM - 19 Feb 2017

Reply to @housecor

Cory House @housecor · Feb 19
Relevant example.

Cory House
@housecor

Author @pluralsight, Speaker, Software Architect, Consultant, @microsoft MVP, @JSJabber panelist, Creator: React Slingshot #JavaScript #ReactJS.

bitnative.com

Joined January 2009

```
import react from 'react’

// code here

export default MyComponent;
```

People call this…

- ES6 Module
- ES2015 module
- TC-39 module
- JavaScript module
- EcmaScript module
- ES Module
- ESM
- Module

What’s an ES Module Build? ¯_(ツ)_/¯

Transpile everything *except* modules

```
var jquery = require(‘jquery’)
// Code here
module.exports = myObj;
```

◀ CommonJS

...into this

```
import jQuery from ‘jquery’
// Code here
export default myObj;
```

◀ ES module

Our current build transpiles this...

Why Provide an ES Module Build?

Modules are statically analyzable

- Reliable autocompletion
- Tree shaking / dead code elimination
- Smaller bundle size

Step 1: Call Babel Preset

```
"presets": [
  "./config/pluralsight-babel-preset",
  "stage-1",
  "react"
]
```

Step 2: Disable ES Module Transpile in ES Build

```
const env = require('babel-preset-env').buildPreset;

module.exports = {
  "presets": [
    [”env", {
      "es2015": {
        "modules": process.env.BABEL_ENV === 'es' ? false : 'commonjs'
      }
    }]
  ]
};
```

Step 3: Add npm Script

```
"build:es": "cross-env BABEL_ENV=es babel ./src/components
--out-file ./lib/index.es.js --ignore spec.js",
```

Step 4: Add Module Entry to package.json

```
"module": "./lib/index.es.js",
```

Is this worth doing?

¯_(ツ)_/¯

Importing specific components provides most of the benefits without this extra complexity.

Only 2 of the 5 most popular React component libraries offer an ES module build.

UMD Build

UMD

Universal Module Definition

Expose your component as a global variable

JavaScript Module Styles

- IIFE
- AMD
- CommonJS
- ES modules (aka ES6)
- UMD

If you publish in UMD, people using all these module styles can use your component

Why UMD?

- Just slap a script tag on the page
- Exposes code on global variable
- No build needed – just add a script tag
- Friendly to experimentation in JSFiddle, JS Bin, etc.
- Useful for public components

UMD

```
(
  function (global, factory) {
    typeof exports === 'object' && typeof module !== 'undefined' ? factory() :
      typeof define === 'function' && define.amd ? define(factory) :
        (factory());
  }(this, function () {
    // Your JavaScript code here
  })
);
```

Q: What should I name the global var?

A: PascalCased component name.

Only 1 of the top 5 React component libraries offers a UMD build.

We’re going to set up a traditional npm package build:

We’ll transpile our code to ES5 with CommonJS using Babel.

Decision:
Documentation Hosting

Cloud Hosting

amazon web services
Microsoft Azure
HEROKU
Firebase
Google Cloud Platform

Static files only

Wrap Up

Key decisions

1. Closed, inner, or open source?
2. Package hosting
3. Import approach
4. How to specify package files
5. Output formats
6. Documentation hosting

Final module: Let’s publish!