Article

Introduction to the Node.js reference architecture, Part 6: Choosing web frameworks

December 3, 2021

Kubernetes, Node.js , OpenShift

Bethany Griggs

Senior Software Engineer

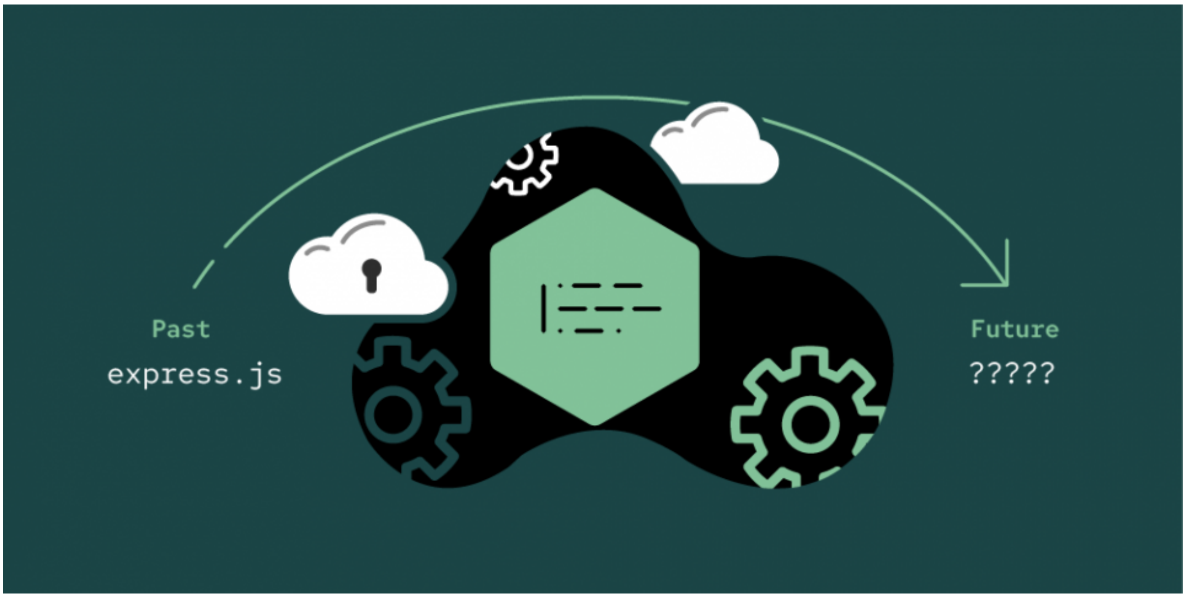


Table of contents:

One of the key choices you make when building an enterprise Node.js application is the web framework that will serve as its foundation. As part of our Node.js reference architecture effort, we've pulled together many internal Red Hat and IBM teams to discuss the web frameworks they've had success with. From our meetings, we've learned that most of the developers we spoke to are still happy with Express.js. This web framework has long been considered the default for Node.js, and it holds that place in our reference architecture as well.

However, Express.js is considered to be in maintenance mode. Thus, as part of the process of developing the reference architecture, we analyzed some data on web framework usage to try to get an idea of what might come next. In this article, you'll learn why Express.js is still a good fit for many Node.js developers and what the future could hold.

As with all our Node.js reference architecture recommendations, we focus on defining a set of good and reliable default choices. Some teams will deviate from some of these recommendations based on their assessment of what best fits their use case.

> **Read the series so far**:
>
> - Part 1: Overview of the Node.js reference architecture
> - Part 2: Logging in Node.js
> - Part 3: Code consistency in Node.js
> - Part 4: GraphQL in Node.js
> - Part 5: Building good containers
> - **Part 6**: Choosing web frameworks

Why Express.js?

We consider Express.js a good default choice for a number of reasons:

- It's used widely, which means that there's a lot of shared knowledge about it both externally and within our organization.
- New users can find a significant amount of resources to help them get started.
- It has a relatively shallow dependency tree, with many dependencies maintained by the Express.js organization.
- It's stable—it doesn't introduce breaking changes too frequently, but still addresses security vulnerabilities as necessary.
- It's compatible across Node.js versions.
- It's been used widely and successfully across IBM and Red Hat, including in the IBM Cloud user interface.

From our in-depth discussions as to which web framework we should recommend as our default choice, we also learned about and documented some other recommendations when using Express.js. Here are two key tips:

- Register a liveness and readiness endpoint even if you're deploying initially to Kubernetes. These endpoints are useful in environments other than Kubernetes for problem determination and monitoring.

- Use Helmet to set HTTP headers for a basic level of protection from some common attacks.

Learn more by reading the full details of our web framework recommendations.

Beyond Express.js: The next generation of web frameworks

While Express.js is considered a good choice of web framework today, discussions and sentiment from our meetings indicate that might not continue to be the case in the future. Express.js is considered to be in maintenance mode, not in active development, and has not seen a new major release in more than five years. Because of this slow release cadence, there is concern that the framework might not keep up with the evolution of the Node.js runtime.

We came away from our discussion process suspecting that in the future our default web framework recommendation will change. As a result, we spent some time digging into various metrics to see what our recommended web framework might be five years from now.

Before we began this investigation, we needed to define its scope. We considered web frameworks that are likely to be used to handle requests and build APIs. We intentionally kept the initial pool of potential candidates as broad as possible, and tried to focus on use cases rather than looking for like-for-like frameworks.

For example, in the past, combining Node.js, Express.js, and a templating engine was a popular choice for building a web application. However, today you can solve the same problem using a dedicated static site framework. There are a lot more options in today's ecosystem, and where years ago for a given use case Express.js might have been the default choice, a more specialized framework might now exist for your use case.

We compiled an initial list of candidates from our reference architecture group discussions, as well as from lists of top Node.js frameworks compiled by outlets like Simform and Hackr.io.

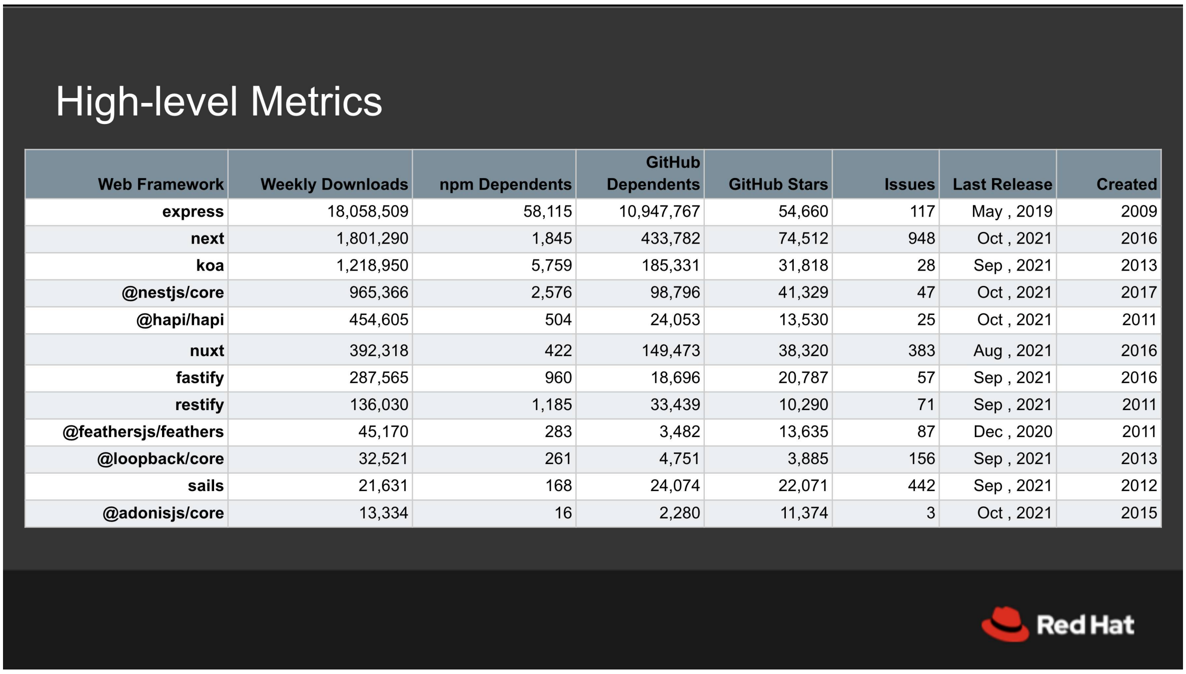
Key metrics

Once we had defined the candidates, we collated some key metrics for each of the frameworks, including:

- Weekly downloads
- npm dependents (that is, how many packages on npm depend on this module)

- GitHub dependents
- GitHub stars
- Issues
- Last release
- Creation date

Generally, the metrics across the board illustrated what we expected, as you can see in Figure 1. Express.js topped the download statistics, and saw the most dependents on both GitHub and npm.



High-level Metrics

Web Framework	Weekly Downloads	npm Dependents	GitHub Dependents	GitHub Stars	Issues	Last Release	Created
express	18,058,509	58,115	10,947,767	54,660	117	May , 2019	2009
next	1,801,290	1,845	433,782	74,512	948	Oct , 2021	2016
koa	1,218,950	5,759	185,331	31,818	28	Sep , 2021	2013
@nestjs/core	965,366	2,576	98,796	41,329	47	Oct , 2021	2017
@hapi/hapi	454,605	504	24,053	13,530	25	Oct , 2021	2011
nuxt	392,318	422	149,473	38,320	383	Aug , 2021	2016
fastify	287,565	960	18,696	20,787	57	Sep , 2021	2016
restify	136,030	1,185	33,439	10,290	71	Sep , 2021	2011
@feathersjs/feathers	45,170	283	3,482	13,635	87	Dec , 2020	2011
@loopback/core	32,521	261	4,751	3,885	156	Sep , 2021	2013
sails	21,631	168	24,074	22,071	442	Sep , 2021	2012
@adonisjs/core	13,334	16	2,280	11,374	3	Oct , 2021	2015

Figure 1. High-level metrics for each web framework.

One other key takeaway is Next.js's relatively high position on the list, even though it's much newer than some of the other frameworks.

Downloads

Download metrics are not particularly useful for determining popularity, as the numbers can be heavily skewed by automation (from continuous integration builds, for example), and also do not include organizations that use internal npm registries or caches.

However, these metrics can help make the relative positions of the frameworks clear. The graph in Figure 2, based on data collected on October 14, 2021, shows weekly npm downloads by web framework. Express.js dominates as expected, and Next.js is also in a strong position.

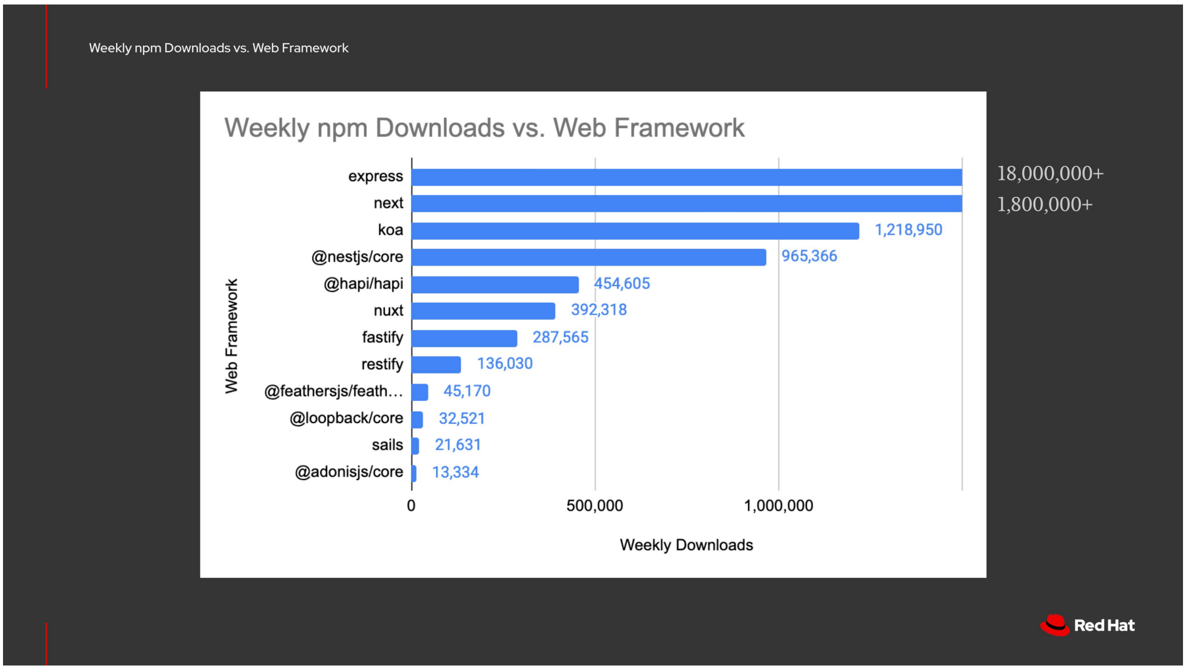


Figure 2. Weekly npm downloads by web framework.

Another way of looking at npm download data is to examine the percentage share of registry downloads by module, as shown in Figure 3. This helps to account for the fact that overall registry downloads are increasing year over year. This information can be calculated using the npm API; for example, to get the total number of downloads for 2020, you can use the endpoint https://api.npmjs.org/downloads/point/2020-01-01:2020-12-31.

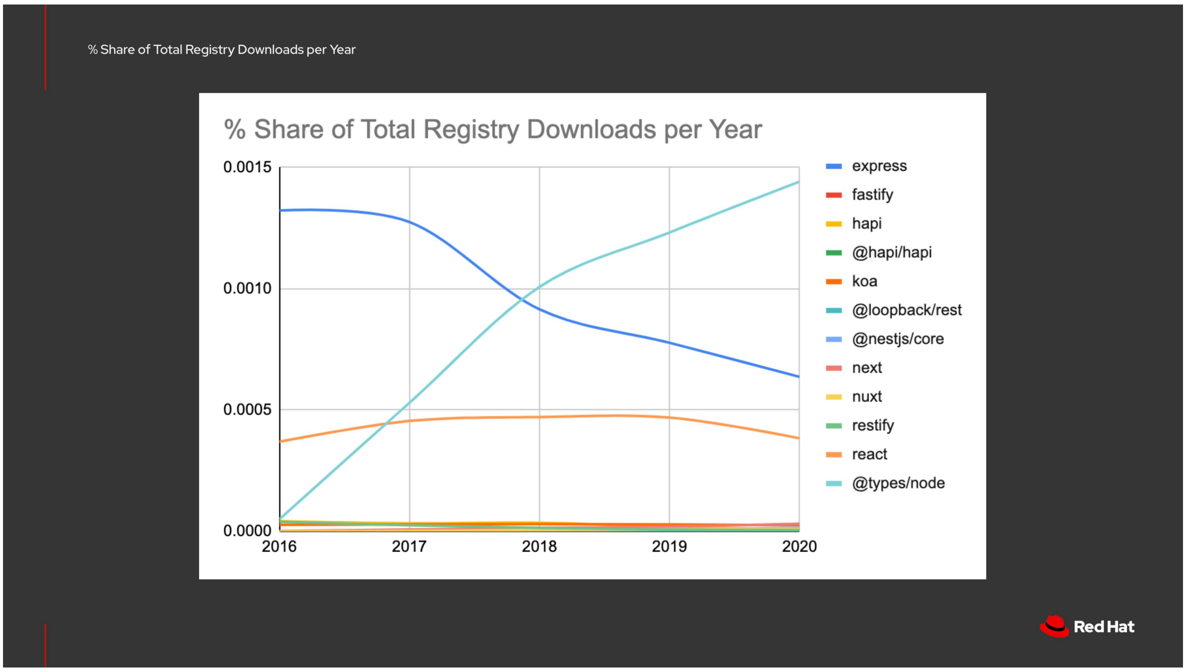


Figure 3. Percentage share of npm registry downloads by web framework.

Figure 3 shows that, over time, the number of downloads to the registry for Express.js as a proportion of the whole is declining. This doesn't necessarily indicate that Express.js usage is declining; the registry downloads might simply be becoming more spread out. We added React to our analysis as a comparative measure, and found that it's seeing a similar trend to Express.js.

Note that hapi is listed on the graph twice—the scoped and unscoped versions are treated separately.

In Figure 3, you can see that a number of less frequently downloaded frameworks are clumped together at the bottom of the chart. The trends here are interesting, so Figure 4 zooms in on them.

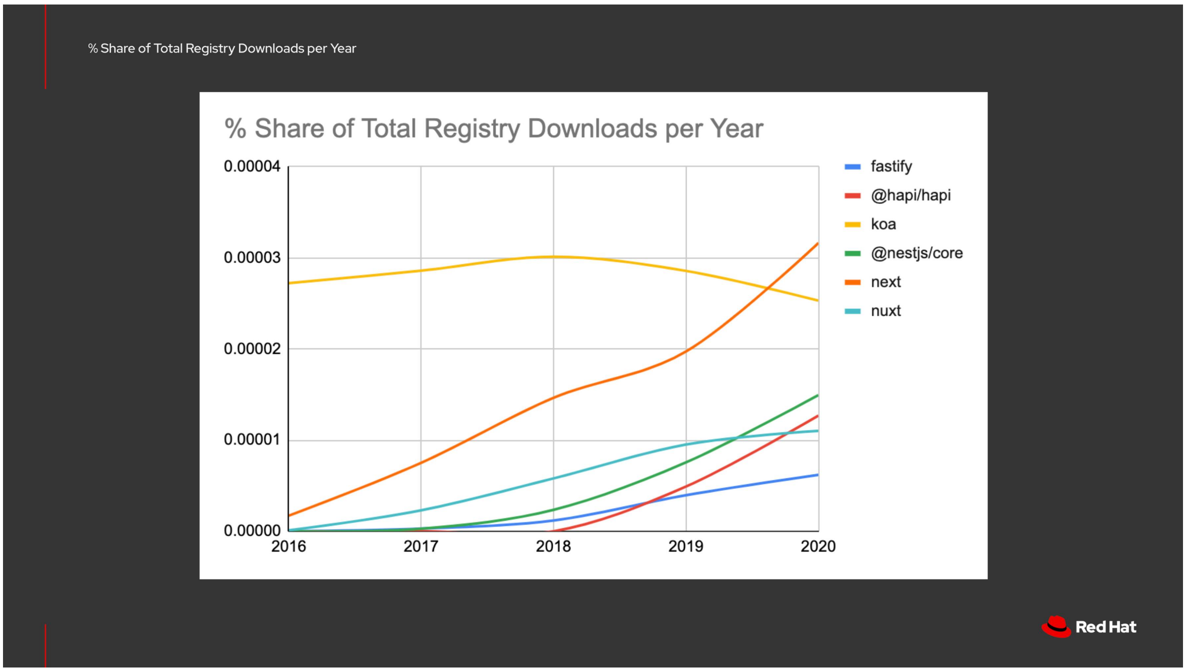
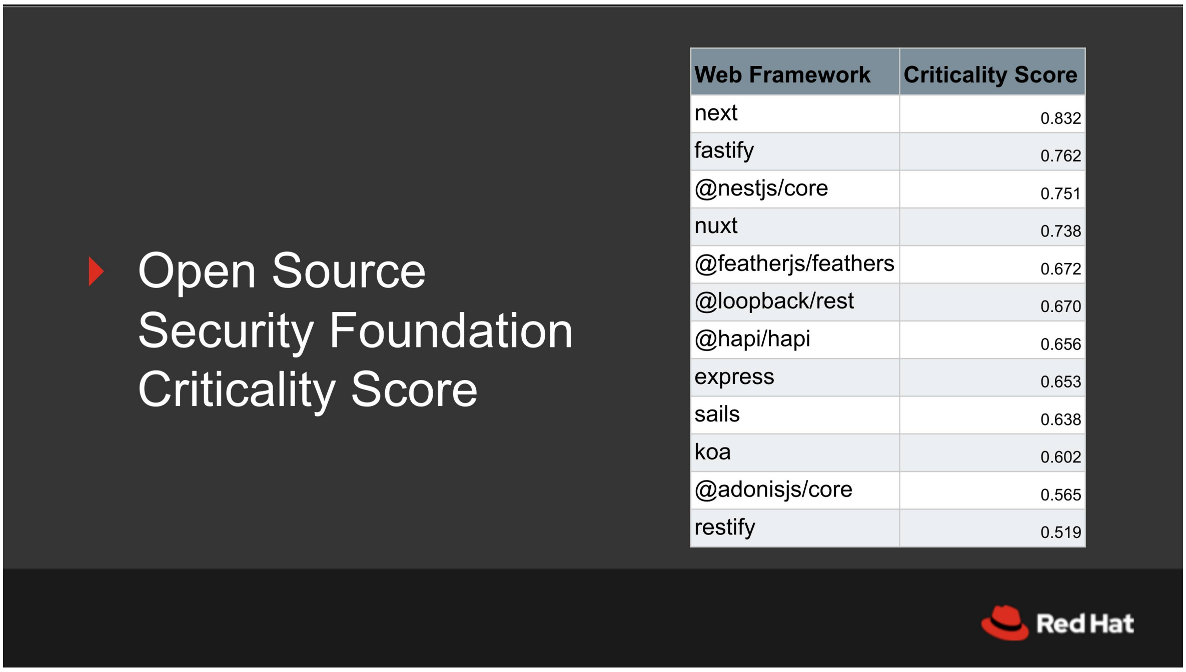


Figure 4. Table of high-level metrics for each web framework (zoomed).

Koa is going through a decline that's similar to what we saw with React and Express.js. Interestingly, we're seeing @hapi/hapi, Fastify, Nest.js, Next.js, and Nuxt.js all increasing, likely indicating that they're gaining popularity. However, the @hapi/hapi increase might be affected by the migration from hapi, the unscoped version of the module.

Open Source Security Foundation criticality scores

The Open Source Security Foundation (OpenSSF) has devised a criticality score that can be used to assess how critical a project is to the open source ecosystem as a whole. We generated criticality scores for all of our web framework candidates, with the results shown in Figure 5.



Open Source Security Foundation Criticality Score

Web Framework	Criticality Score
next	0.832
fastify	0.762
@nestjs/core	0.751
nuxt	0.738
@featherjs/feathers	0.672
@loopback/rest	0.670
@hapi/hapi	0.656
express	0.653
sails	0.638
koa	0.602
@adonisjs/core	0.565
restify	0.519

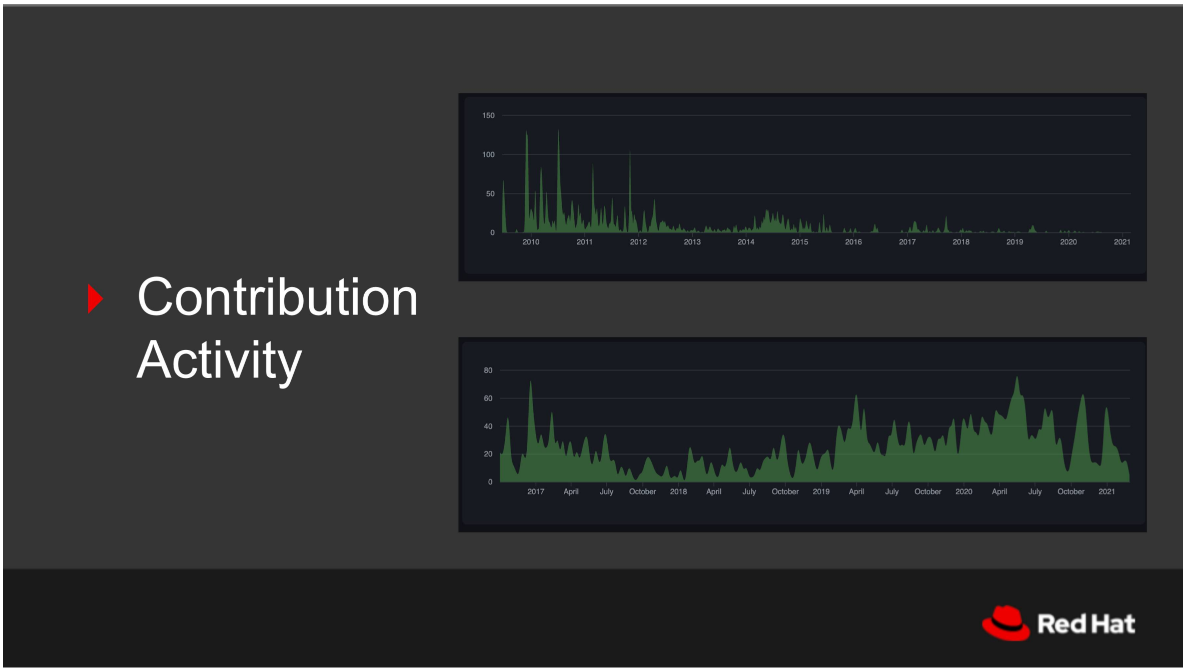
Figure 5. Open Source Security Foundation criticality scores.

Next.js and Fastify generated the highest criticality scores, with Express.js further down the list. You can download the tool to generate these scores, along with documentation explaining more about how they're generated, from the project's GitHub repository.

Contributions

Contributions and contribution activity are also useful metrics for assessing frameworks. We began by looking at the contribution graphs generated by GitHub for each of the web frameworks we considered. (You can find this data by going to the Insights view on a project's GitHub repository.)

The top graph in Figure 6 illustrates a common scenario for many older web frameworks: An early peak in contribution activity, followed by a tailing off in more recent years. This could indicate that the project is in a maintenance phase rather than in active development. A few of the more recent frameworks, such as Next.js, demonstrated a more consistent pattern of activity, as illustrated in the lower graph in Figure 6.

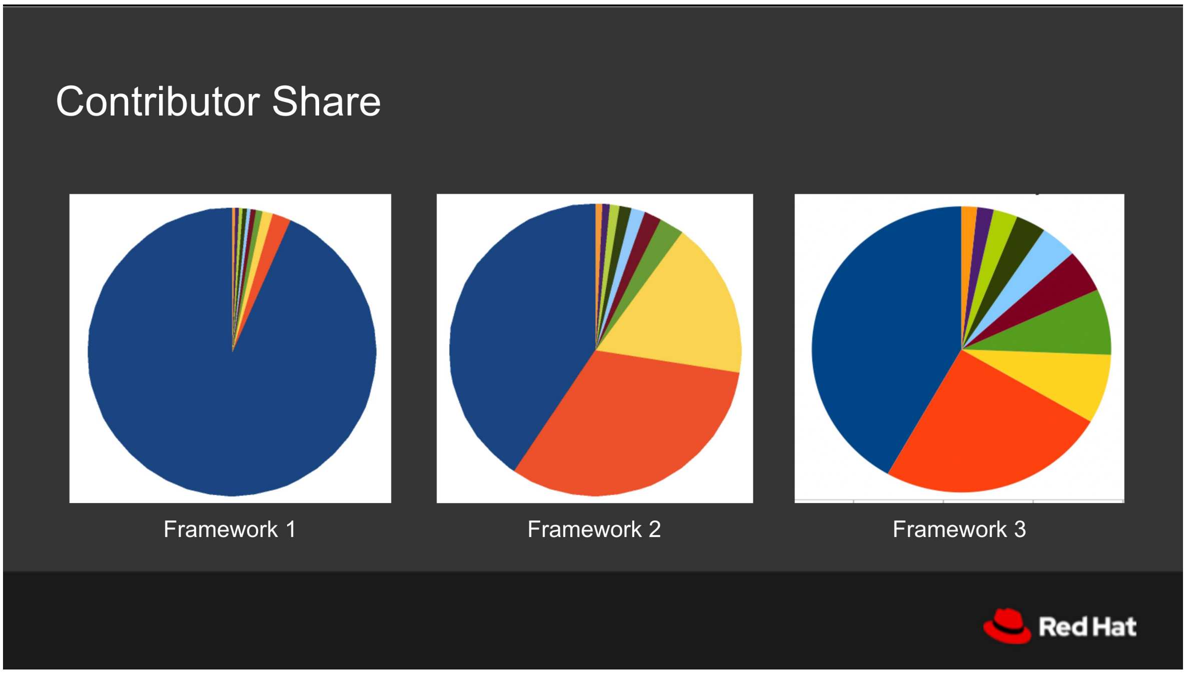


Contribution Activity

Figure 6. GitHub contribution activity graphs.

Another metric that we considered was the distribution of contributions, so we looked at the share of commits per user (and their associated organizations). From this data, we could infer whether a project was mostly led by a specific individual, company, or community. Anton Whalley, Technology Partner Architect at IBM, created an application to generate these metrics using the GitHub API. You can access the application here or take a look at the source code.

As the charts in Figure 7 illustrate, some of the web frameworks we examined, like Framework 1, are mostly dominated by a single contributor. Others, like Framework 3, have a more spread out distribution of contributions.



Contributor Share

Framework 1

Framework 2

Framework 3

Figure 7. Share of commits per GitHub user.

The priorities of truly community-led frameworks are not controlled by a single entity, and new users are likely to be able to contribute on an equal level.

Conclusion

Our Node.js reference architecture recommendations are based on what teams across IBM and Red Hat have used successfully. As such, we expect that over time our recommendations will continue to evolve. We still feel that Express.js is a good default choice for a web framework today, though we're keeping in mind some of the known concerns about it.

At the same time, we're continuously evaluating other frameworks by looking at their metrics and considering the qualities that would favor a different candidate in the future. Ideally, a web framework should:

- Be adequately maintained.
- See regular releases.
- Keep up-to-date with Node.js core features and changes.
- Be community-led with open governance.

We'll continue to review internal and client usage and success stories. There are a few promising up-and-coming web framework candidates, but no single framework has outpaced the others enough to make it our new default recommendation—yet.

We hope you find these recommendations useful. While you wait for the next installment in the Node.js reference architecture series, you can check out the GitHub project to explore sections that might be covered in future articles.

If you want to learn more about what Red Hat is up to on the Node.js front, you can also explore our Node.js topic page.

Recent Articles

Node.js at Red Hat: 2021 year in review

The GDB developer’s GNU Debugger tutorial, Part 2: All about debuginfo

Debug .NET applications running on Kubernetes with VS Code

Why you should migrate your Java workloads to OpenShift

Simplify Java persistence using Quarkus and Hibernate Reactive

Related Content

- Node.js serverless functions on Red Hat OpenShift, Part 1: Logging
- Introduction to the Node.js reference architecture, Part 1: Overview
- Introduction to the Node.js reference architecture, Part 3: Code consistency
- Introduction to the Node.js reference architecture, Part 4: GraphQL in Node.js
- Introduction to the Node.js reference architecture, Part 5: Building good containers

Comments

Overall, how satisfied are you with this article?

21 Responses

😍 Very Satisfied 👍 Slightly Satisfied 😯 Neutral 😥 Slightly Unsatisfied 😤 Very Unsatisfied

Red Hat Developer Comment Policy

Please keep your comments relevant and polite. Opinions shared in comments are not official Red Hat news or policy.
Please read our Comment Policy before commenting.

0 Comments **Red Hat Developer** 🔒 **Disqus' Privacy Policy** 1 **Login**

♡ **Favorite** Tweet Share **Sort by Best**

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name