Feb 25, 2019

Rust Case Study:

Community makes Rust an easy choice for npm

The npm Registry uses Rust for its CPU-bound bottlenecks

Rust at npm

npm, Inc is the company behind npmjs.com and the npm Registry. The largest software registry in the world, it sees upwards of 1.3 billion package downloads per day. In addition to maintaining the open source Registry, npm also offers an enterprise registry product and maintains a command-line tool to interact with registries.

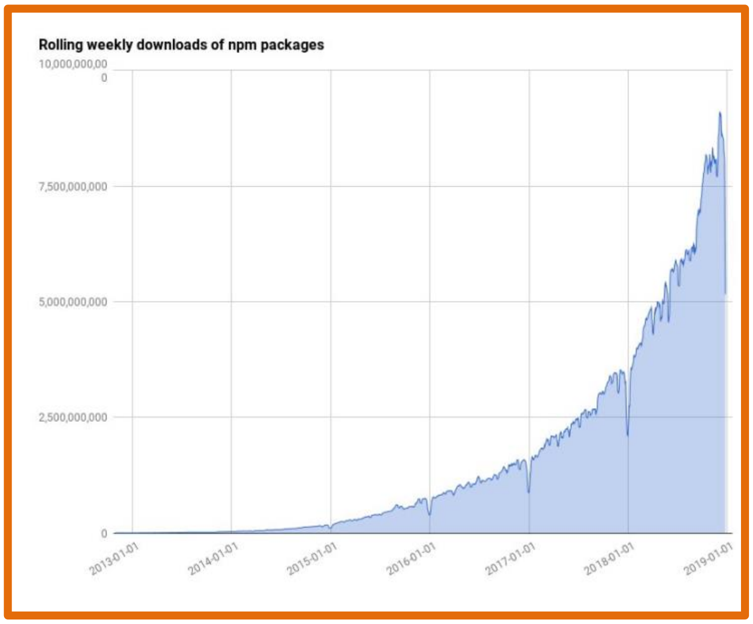
The challenges that npm faces demand efficient and scalable solutions. When a service can be deploy-and-forget, that saves valuable operations time and lets them focus on other issues. npm employees also value having a helpful community around any technology they use. Rust fits all these criteria and is currently in use as part of npm's stack.

Facts and Figures

- Over 836,000 JavaScript packages are available
- Each month, npm serves over 30 billion packages
- These downloads come from over 10 million users

Problem: Scaling a CPU-bound Service

The npm engineering team is always on the lookout for areas that may potentially become performance problems as npm's exponential growth continues. Most of the operations npm performs are network-bound and JavaScript is able to underpin an implementation that meets the performance goals. However, looking at the authorization service that determines whether a user is allowed to, say, publish a particular package, they saw a CPU-bound task that was projected to become a performance bottleneck.



The legacy JavaScript implementation of this service in Node.js was in need of a rewrite in any case, so the npm team took the opportunity to consider other implementations to both modernize the code and improve performance before service degraded.

Solutions Considered

When considering alternate technologies, the team quickly rejected using C, C++, and Java, and took a close look at Go and Rust.

A C or C++ solution is no longer a reasonable choice in the minds of the npm engineering team. “I wouldn't trust myself to write a C++ HTTP application and expose it to the web,” explains Chris Dickinson, an engineer at npm. These languages require expertise in memory management to avoid making mistakes that cause catastrophic problems. Security problems, crashes, and memory leaks were not problems that npm was willing to tolerate in order to get improved performance.

Java was excluded from consideration because of the requirement of deploying the JVM and associated libraries along with any program to their production servers. This was an amount of operational complexity and resource overhead that was as undesirable as the unsafety of C or C++.

Given the criteria that the programming language chosen should be:

- Memory safe
- Compile to a standalone and easily deployable binary
- Consistently outperform JavaScript

the languages that remained under consideration were Go and Rust.

Evaluation

To evaluate candidate solutions, the team rewrote the authorization service in Node.js, Go, and Rust.

The Node.js rewrite served as a baseline against which to evaluate the change in technology to either Go or Rust. npm is full of JavaScript experts, as you would expect. The rewrite of the authorization service using Node.js took about an hour. Performance was similar to that of the legacy implementation.

The Go rewrite took two days. During the rewrite process, the team was disappointed in the lack of a dependency management solution. npm is a company dedicated to making the management of JavaScript dependencies predictable and effortless, and they expect other ecosystems to have similar world-class dependency management tooling available. The prospect of installing dependencies globally and sharing versions across any Go project (the standard in Go at the time they performed this evaluation) was unappealing.

They found a stark contrast in the area of dependency management when they began the Rust implementation. "Rust has absolutely stunning dependency management," one engineer enthused, noting that Rust's strategy took inspiration from npm's. The Cargo command-line tool shipped with Rust is similar to that of the npm command-line tool: Cargo coordinates the versions of each dependency independently for each project so that the environment in which a project is built doesn't affect the final executable. The developer experience in Rust was friendly and matched the team's JavaScript-inspired expectations.

> “
> **Rust has absolutely stunning dependency management.**

The rewrite of the service in Rust did take longer than both the JavaScript version and the Go version: about a week to get up to speed in the language and implement the program. Rust felt like a more difficult programming language to grapple with. The design of the language front-loads decisions about memory usage to ensure memory safety in a different way than other common programming languages. “You will write a correct program, but you will have to think about all the angles of that correct program,” described Dickinson. However, when the engineers encountered problems, the Rust community was helpful and friendly answering questions. This enabled the team to reimplement the service and deploy the Rust version to production.

Results

npm’s first Rust program hasn't caused any alerts in its year and a half in production. "My biggest compliment to Rust is that it's boring," offered Dickinson, "and this is an amazing compliment." The process of deploying the new Rust service was straight-forward, and soon they were able to forget about the Rust service because it caused so few operational issues. At npm, the usual experience of deploying a JavaScript service to production was that the service would need extensive monitoring for errors and excessive resource usage necessitating debugging and restarts.

> “
> **My biggest compliment to Rust is that it’s boring.**

Community Matters

npm called out the Rust community as a positive factor in the decision-making process. Particular aspects they find valuable are the Rust community's inclusivity, friendliness, and solid processes for making difficult technical decisions. These aspects made learning Rust and developing the Rust solution easier, and assured them that the language will continue to improve in a healthy, sustainable fashion.

Downsides: Maintaining Multiple Stacks

Every technical decision comes with trade-offs, and adding Rust to npm's production services is no different. The biggest downside of introducing Rust at npm is the maintenance burden of having separate solutions for monitoring, logging, and alerting for the existing JavaScript stack as well as the new Rust stack. As a young language, Rust doesn't yet have industry-standard libraries and best practices for these purposes, but hopefully will in the future.

Conclusion

Rust is a solution that scales and is straightforward to deploy. It keeps resource usage low without the possibility of compromising memory safety. Dependency management through Cargo brings modern tools to the systems programming domain. While there are still best practices and tooling that could be improved, the community is set up for long-term success. For these reasons, npm chose Rust to handle CPU-bound bottlenecks.