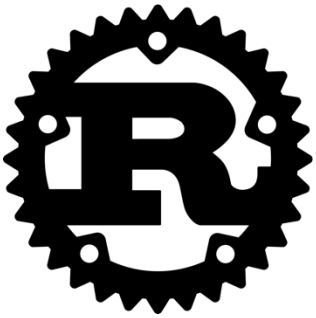
Jan. 2018



Rust Case Study:

How Rust is Tilde's Competitive Advantage

The analytics startup innovates safely with the help of Rust

graphics by Aldric Rodríguez and Chameleon Design from the Noun Project

Jan. 2018

Rust at Tilde

Tilde, Inc is a startup based in Portland, OR disrupting the application performance monitoring space. Their product, Skylight, turns performance data from Ruby on Rails applications into actionable insights so that developers can keep their applications fast. With a team of 6 developers of various skill levels, Tilde is gaining ground in a crowded market. One of their competitive advantages? **Rust.**

Low Customer Tolerance for Resource Usage and Crashing

"Because our product helps people identify why their apps are slow, it is very important that we ourselves do not make their app slow," says Yehuda Katz, CTO of Tilde. The Skylight agent runs within a customer's production Rails application to monitor real performance metrics. The tolerance customers have for memory and CPU overhead used by this agent is extremely low, and their tolerance for crashes lower still.

Attempted Solutions: Ruby and C++

In 2013, Tilde shipped the first version of the agent, written in Ruby. While the Ruby implementation enabled Skylight to get started, the agent was limited in the data it could collect. Any feature added to the agent would make it use an unacceptable amount of memory. If Skylight wasn't able to add differentiating features, it wouldn't be able to compete against the entrenched players in the analytics space.

Katz and his team spent time squeezing every drop of performance out of Ruby including removing uses of higher-level features of the language that required more resources. This had only a marginal impact on performance and

made the code much less maintainable. Members of the team would have to become experts in Ruby and maintain constant vigilance lest a common yet inefficient Ruby feature got added to the agent. This would take time away from implementing features and severely constrain who Tilde could hire.

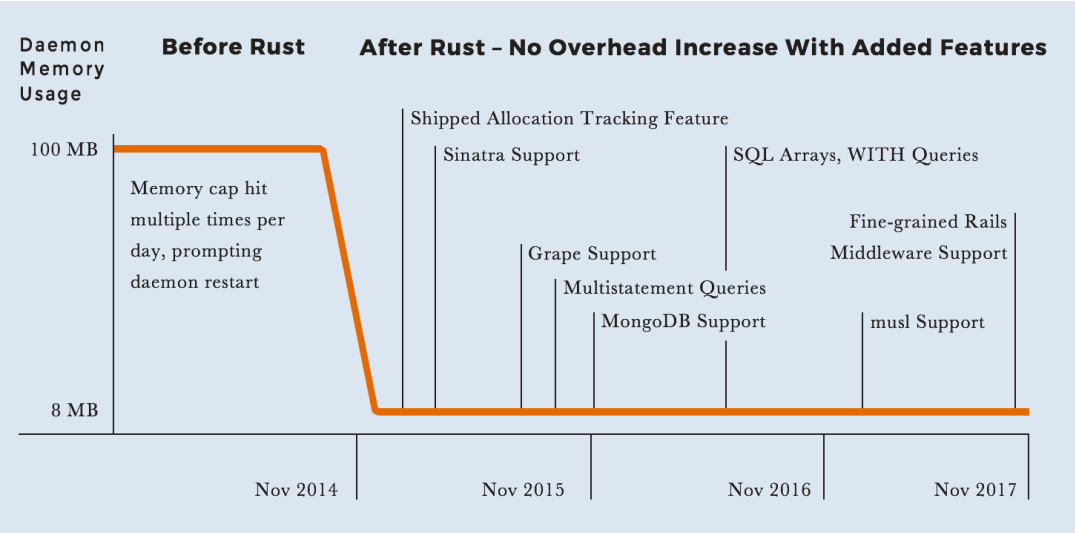
Next, they prototyped the agent in C++. Although the C++ implementation used fewer resources, it also introduced an unacceptable risk of crashes. Training existing Ruby developers to maintain safe C++ code would take too much time. Hiring a C++ expert specifically for maintaining the agent wouldn't be ideal in a startup where everyone wears multiple hats and everyone can help with many parts of the product. C++ wasn't ideal either.

Enter: Rust

In December 2013, Rust started making waves. Even though Rust had yet to release 1.0 with its associated stability guarantees, Katz was intrigued by the safety guarantees Rust promised without requiring a garbage collector. After only replacing the Ruby data structures in the agent with Rust data structures, they saw a larger impact in reducing the amount of overhead used by the agent than they saw from all the earlier Ruby performance tuning.

By 2014, the Skylight agent with some Rust code was ready to ship to production. The reduction in overhead enabled by the use of Rust made it possible for Skylight to add a killer feature of tracking the memory allocations made by Rails requests. Even better, there were no reports of segfaults from Skylight's customers! This early success encouraged the team to rewrite the entire agent in Rust.

Most of Skylight's customers hosted their applications on Heroku, where they were subject to a 256 or 512 MB memory limit. To avoid putting customers over this limit, the Skylight agent came with a daemon that would restart the agent if its memory usage exceeded 100 MB. Before the Rust rewrite, the daemon was restarting Skylight agents constantly and customers still often reported they received errors from Heroku that their applications were over the memory limit.



After rewriting of the agent in Rust, the agent consistently used 8 MB: 92% smaller! Once this version was shipped to production, there were no more customer reports of being over the Heroku memory limit. In terms of raw performance, Rust was a clear win, but that wasn't the only benefit the Tilde team saw.

Making Room for New Features

Because of the strict memory requirements in the environment in which the Skylight daemon runs where the total application memory had to fit under the Heroku limits, memory usage was an important design consideration. Rust allowed the Tilde engineers to craft differentiating features that collect more data in Skylight without having to worry about unintentional resource bloat caused by garbage collection and a language runtime. In the time between the switch to Rust in Nov 2014 and the end of 2017, Tilde was able to add to the agent support for applications like Sinatra and Grape, databases like MongoDB, and understanding of more complex types of queries, all while keeping the daemon's memory usage around 8 MB.

Effectively Zero Crashes

Tilde reports that the Skylight project has been reliable across a range of customer operating systems and contexts, including macOS and various Linux

flavors. Katz also observes that Rust's safety features have maintained the reliability of the agent, unlike other programming languages they could have chosen. "We've never made a programming error in Rust code that has caused a segfault that our users have reported," he is pleased to say.

More Maintainable

Rust's dependability meant that updates were "fire and forget," allowing Tilde to regularly push out updates to a variety of clients without fear that software issues would cause downtime for their clients. Over the last 3 and half years, there have been 63 releases of the Skylight agent; each of which were quickly and easily adopted by their customers.

Prevented Data Races at Compile Time

Rust's prevention of data races at compile time allowed Tilde to discover potential issues long before a user could find it in production. In one such case, engineers new to Rust changed Skylight's logging ability to use a new data structure to help prevent duplicates in log messages. The Rust compiler gave an error and showed that this new data structure would not play well with the concurrency inherent in the system. Once fixed, the application worked as expected. Katz notes that a bug like this would have gone undetected in other systems languages and could have caused crashes, data corruption, or even data loss in production, while also being difficult to discover and fix correctly. Rust enabled the team to catch and fix this problem early in the development process.

Teachable to a Broad Team

Katz points out that there are three aspects of Rust that make it eminently teachable to a team with a broad range of experience levels:

- **Interest in learning Rust.** Engineers have a sense that Rust genuinely improves their marketable skills in an area that is very different from what they're already good at. As Vaidehi Joshi, Software Engineer at Tilde, puts it, "I think that Rust does a good job of combining the theory and

practice of Computer Science concepts, without being *too* scary for someone who is new to systems programming."

- **Rust's safety.** Rust is much easier to teach than C or C++, largely because of the depth of knowledge an engineer needs to have before modifying C or C++ code correctly. "The compiler is easily my favorite part of working on Rust projects at Tilde," says Software Engineer Lee Baillie. "It often feels almost like Test Driven Development in that I can be certain the code is at least technically correct if it will compile. When it doesn't compile, I get a usually-helpful error message that helps me figure out what went wrong. This is nice compared to writing Ruby or JavaScript because I often don't know that something will break until runtime. I would much rather know about these kinds of small mistakes before they become a problem!"

- **Cargo and the Rust library ecosystem.** Because these tools are similar to modern package managers in other languages, the team was comfortable jumping in to Rust. Cargo reduces the need to learn a less-ergonomic environment than that of other systems programming languages just to get basic builds working. This allows them to focus on learning the language, rather than spending time getting tools and build systems set up and working properly.

The Future of Rust at Tilde

The Tilde engineers, encouraged by these improvements in the Skylight agent, have begun rewriting the data collection and processing server application from Java to Rust. Rust's smaller memory footprint was immediately noticeable; while the Java server could use up to 5GB of RAM, the comparable Rust server only used 50MB.

While Rust won't ever be the only tool in Tilde's toolbox, the success the company has seen in their ability to add features without overhead has made the team confident that Rust is a solid choice.