BIG DATA FEATURE

Why the creators of the Julia programming language just launched a startup

JORDAN NOVET @JORDANNOVET MAY 18, 2015 3:20 PM

Above: From left, Julia Computing cofounders Keno Fischer, Viral Shah, Stefan Karpinski, Alan Edelman, and Jeff Bezanson.
Image Credit: Julia Computing

Earlier this month, the creators of the programming language Julia launched their startup, Julia Computing LLC, to provide training, commercial support, and consulting for those who want to use the language.

Finally, all of the creators — Viral Shah, Jeff Bezanson, Stefan Karpinski, and MIT applied mathematics professor Alan Edelman — are able to work full-time on the 3-year-old language. They've attracted some people to build the business, which first started in December 2013.

And Julia is a big deal — it's a free alternative to proprietary tools for doing data science, like MathWorks' MATLAB and Wolfram's Mathematica, and it's more contemporary than open-source languages R and Python. More companies are hiring data scientists to make more data-driven decisions, and open-source tools often come in handy.

Even so, the launch did not receive very much press attention, with the exception of an article in India's Economic Times. I reached out to Shah to ask some questions about the startup and

the Julia language in general. Like a true open-source team player, Shah came back to me with answers he and his cofounders collaborated on.

My takeaway: The creators of Julia are determined to improve the language, and do it all in the open, to keep the open-source community strong. It might not be the enterprise data science tool of choice, but there is enterprise demand, and in the next few years, so long as the creators remain dedicated, it should become more popular.

Here's an edited transcript of the interview.

VentureBeat: Do you have any venture backing? Have you been approached about funding? Will you hire more? Who is on the team now other than you, Stefan, Jeff, Alan, Deepak Vinchhi, and Keno Fischer? Are there any physical offices?

Julia Computing: We have certainly been approached by investors but haven't taken any investments.

We are self-funded and have been profitable from day one — although we reinvest most of our profits into growing the language and the business. We are focused on creating the best open-source technical computing environment we can, and fostering a healthy dialogue between the open-source world, technical academia, and businesses. These communities have been far too isolated in the past, but they have much to offer each in terms of new ideas and best practices.

We have an office in Boston, and we are in the process of setting up an office in Bangalore; Stefan lives in New York, so we may have an office there in near future. The team includes the six cofounders that you mention, and the total headcount is about twice that and growing.

VentureBeat: Do you have paying customers yet?

Julia Computing: Yes. At the moment we have nine paying customers and a significant number of strong leads. Many enterprises are happy to hear about the existence of Julia Computing, and have actively sought us out for support, training and expertise in the language. In our experience, for every Julia enthusiast in an organization, there are 20 others who are interested, but not sure they want to dabble in an open-source project without a company behind it. With Julia Computing providing support, these users feel like there is someone out there watching their back.

VentureBeat: Do you have a sense of the current Julia user base other than what's shown on GitHub?

Julia Computing: It's hard to get accurate estimates for open-source software, but here are some numbers that we do have.

- Since the launch of julialang.org in 2012, the site has been seen by 1.3 million unique visitors.
- On GitHub, we get more than 500 clones and more than 9,000 unique visitors every two weeks.
- There are about 10,000 JuliaBox users, even though we've largely kept it quiet.
- As you mentioned, the repo has over 5,000 stars, and 350 contributors.
- The juliausers mailing list has about 3,000 subscribers.
- The Julia package ecosystem counts almost 600 packages today.
- If we had to hazard a guess, there are probably some 50,000 Julia users out there.

We feel really fortunate that Julia has become such a healthy open-source project – at this point, it is clearly here for the long haul. Some people have expressed concern that we might be tempted to undermine that by handicapping the open version and selling a closed version with better functionality. This would not only be bad for the project, but also terrible for our business. No one has made a good business off this kind of move: it ends up sabotaging the project, which in turn ultimately kills the business. We're in this for the duration — our goal is to create a vibrant and fruitful collaborative ecosystem, that includes academic researchers, developers who contribute for personal enjoyment, and companies using Julia for business.

VentureBeat: Any plans to do partnerships or integrations?

Julia Computing: Primarily, we focus on end customers directly, and building a great Julia brand where everyone first thinks about Julia when they have a computational problem. Many of these customers have existing data stacks they would like us to integrate with. For exactly that reason, we are exploring various partnerships — Hadoop vendors, database vendors, cloud solution providers, strategic partners, domain specialists, and so on. We will announce things

VentureBeat: Talk about adoption at web companies and growth in the Julia community in general.

Julia Computing: Adoption has been relatively slow in traditional tech companies. Big tech companies tend to have their own programming languages — Go at Google, Hack at Facebook, Swift at Apple, Java at Oracle, C# at Microsoft, or Rust at Mozilla. On the other hand, we're seeing rapid adoption in finance and financial modeling services, as well as business logistics — think operations research — and companies interested in high-performance computing (HPC).

If you think about it, this makes sense: software is the core competency of traditional tech companies — they can afford to have their legions of professional programmers use "hard" languages like C++ and Java, which are great for performance and deployment, but less good for exploration and prototyping. Financial modeling requires bringing together domain experts — quants and traders — with programmers, and often there are people who wear many hats and fall somewhere on the spectrum between these roles. Julia works well for this because it's a broad spectrum language: quants can be perfectly happy using it as a fast, free Matlab replacement, while hardcore systems programmers are happy to use it as an easier, more productive C++ replacement to build fast, reliable production systems in.

VentureBeat: Could you respond to criticisms about the immaturity of the language, such as Dan Luu's essay "Julia is Awesome, But..." from last year?

Julia Computing: Many of the criticisms Dan presented were quite legitimate. We've improved our testing of both packages and the language itself massively since then — test coverage just passed 78 percent last week, and we're shooting for 100 percent. Package management is a hard problem that well-established languages like Python and Ruby still struggle with. There isn't really any package management for languages like C and C++, so clearly this isn't a show stopper for production use of a language. However, we are actively improving the package manager, and already, installing Julia packages on Linux, OS X and Windows mostly "just works." That is no mean feat for numerical software, which is notoriously hard to build. Julia's internals are not as well documented as they should be — but, of course, that doesn't affect users of the language, just people who want to hack on the compiler.

On the whole, we think that Julia is remarkably reliable and stable for such a young language. Our customers have had very few problems in production code — We would estimate on a par with C++ — largely because it is easy to write correct, safe code without sacrificing