DAVID HEINEMEIER HANSSON

January 12, 2023

They're rebuilding the Death Star of complexity

I started my career in programming during heydays of Java Enterprise Edition (J2EE). This was late 90s/early 00s, and there was a rich ecosystem of enterprise vendors hawking application servers, monitoring tools, and boxes upon boxes of other fancy solutions. These tools were difficult to learn, expensive to license, and required an army of contractors, consultants, and sales people to purchase, configure, and run. All to do essentially what a Perl CGI script or PHP page could also do: Pull data from a database, and render it as HTML to a browser!

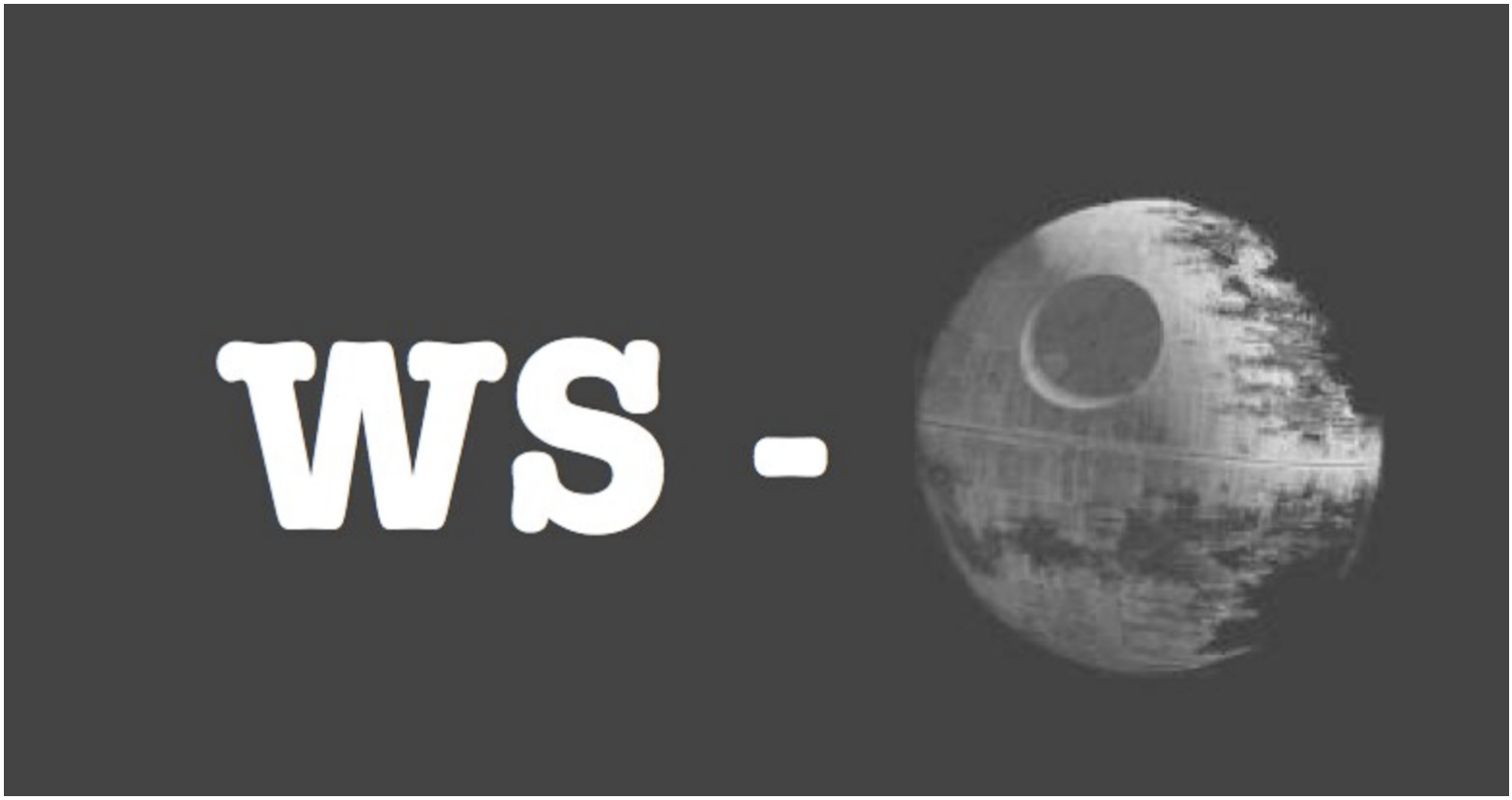
It's not that there weren't good ideas hiding underneath the cruft of complexity required to vendorize these solutions. But that the ratio between Good Idea and Bullshit was completely out of whack.

That out-of-balance ratio seeded the original mission for Rails: To liberate the good ideas from the bullshit. To flush out all the nonsense, and to compress the remaining intrinsic complexity, so mere mortals without an advanced degree in enterprisey gibberish could employ the hidden gems of ingenuity that really were present.

It worked! Rails introduced a deep catalogue of solid patterns from J2EE and elsewhere to a group of programmers that never would have had the patience to use them in their original wrapping. And it gave those already versed in the patterns a simpler environment to exploit them.

It also drained away much of the enterprisey swamp in the process. There just wasn't room for the same breadth and depth of commercial vendors selling their convoluted wares to an ecosystem of companies and developers running commodity hardware with commodity databases, web servers, and simple open-source programming frameworks.

But the merchants of complexity don't close up shop just because a single market dries up. They migrate to the next. And shortly after Rails came to the scene, the next big complexity gold rush was to be Web Services, also known as WS-* (or, as I referred to it back in the day, WS-DeathStar).



Like with the original J2EE spec, which sought to complicate the basic mechanics of connecting databases via HTML to the internet, this new avalanche of specifications under the WS-* umbrella sought to complicate the basic mechanics of making applications talk to each other over the internet. With such riveting names as WS-SecurityPolicy, WS-Trust, WS-Federation, WS-SecureConversation, and on and on ad nauseam, this monstrosity of complexity mushroomed into a cloud of impenetrable specifications in no time. All seemingly written by and for the same holders of those advanced degrees in enterprisey gibberish.

Thankfully, just like J2EE itself, this XML-infested mess quickly imploded under the pressure of the plain alternative: Just sending JSON over standard HTTPS to servers running whatever on the backend. While practitioners argued for years about what level of purity was required to really be called RESTful, it didn't really matter. The internet voted with its web requests, and simple JSON over HTTPS once again beat back the merchants of complexity to save the human race of programmers from a regrouping empire.

Today you'll still find fragments of the old world around the internet. A reference to SOAP here, some bank still running an original J2EE stack there. But the wide world of the internet belongs to much simpler tooling and ideas in forms such as Rails, JSON, REST, and commodity databases and web servers. (JavaScript is an interesting hybrid case that needs its own exploration, but I'll align it more with the rebels than the empire for the purposes of this discussion).

But as with WS-* rising from the same incentives that drove J2EE, you should always expect the merchants of complexity to start rebuilding their Death Star in another unsuspecting galaxy, once one is destroyed in your own. The struggle between profitless simplicity and profitable complexity is

eternal in the world of software. When things become too easy, too approachable, there just isn't room for all the enterprisey services, sales, and solutions to prosper.

That finally brings us to the latest frontier: Containerized running of software in the cloud! The initial advancement and popularization of containers rank as one of my favorite milestones of progress in our industry. But damn if the merchants of complexity didn't see such simplicity as a bug to quickly correct, and use their impressive tractor beam of bullshit to pull us back towards the ever-reconstructing Death Star once more.

Again, I'm certain that there are good ideas hidden inside the advancement of Kubernetes, and the mushrooming of tools, services, and standards that are following in its wake, but the ratio of bullshit is also already completely out of whack.

What are we trying to do here? We're trying to run an application on some hardware in a reasonably efficient manner, where we can easily redeploy it when it changes. Yeah, sure, fair. But try contrasting what's required to get going with Kubernetes, both within your application, and to operate control planes and what not, with the breakthrough simplicity of the original Docker advancement, and it's damn depressing.

But also predictable. The complexity that's been conjured up is immensely profitable. The renewed demand for advanced degrees in enterprisey gibberish is propelling the same expansion of jargon, specifications, and techniques as we saw with J2EE and WS-DeathStar. This is what this industry does!

In hindsight, it usually always looks silly. Try reading some of those early 2000s manuals for WebSphere or whatever and you'll marvel at the incantations that were used to be required to simply get to Hello World. Or I dare you to digest some of those WS-* specifications, and try to make sense of it all. It's technical masochism.

But when you're in the thick of it, it usually all seems far more, if not reasonable, then at least excusable. Yes, this is all horrendously complicated, *but* look at what we can do! And really, here are 17 reasons for why the simple approach is just naive or unsophisticated. If you can't see the splendor of the king's fine threads, surely it is you, little boy, who is mistaken.

Beware carnies dressed as storm troopers. The circus empire is always coming to town. Keep a safe distance, a skeptical mind, and one hand on your wallet.

ABOUT DAVID HEINEMEIER HANSSON

Creator of Ruby on Rails, co-owner & CTO of 37signals (Basecamp & HEY), best-selling author (REWORK, It Doesn't Have to Be Crazy at Work, REMOTE), Le Mans class-winning racing driver, antitrust advocate, investor in Danish startups, frequent podcast guest, and family man.

Subscribe to get future posts via email (or grab the RSS feed)

Type your email… Subscribe