CI/CD

What Led Amazon to its Own Microservices Architecture

8 Oct 2015 2:39pm, by Scott M. Fulton III

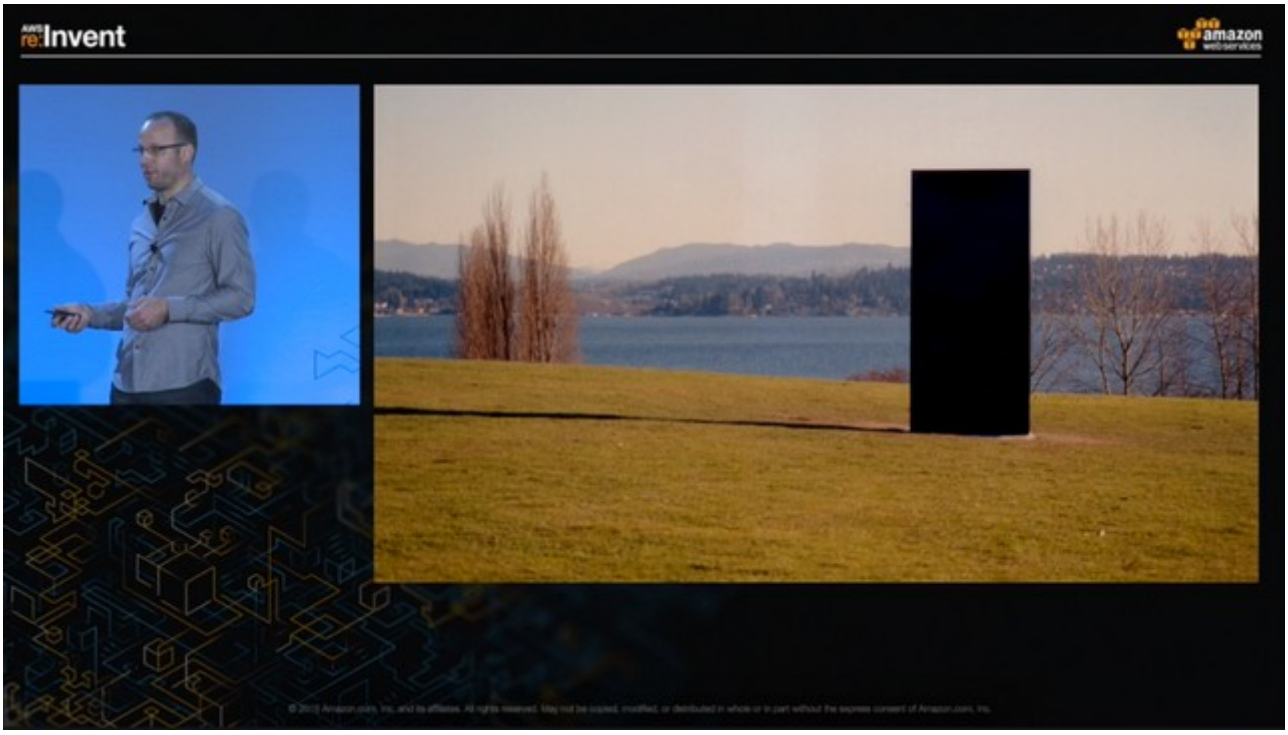
2

The real story of The New Stack, time and time again, has been about how companies with huge server demands, constrained by the inability of existing architectures to fulfill these demands, resolved their issues for themselves, and then turned around and resold their solutions to the rest of the world. How quickly we forget that the earliest example, but perhaps still among the best, is Amazon.

> "If you go back to 2001," stated Amazon AWS senior manager for product management Rob Brigham, "the Amazon.com retail website was a large architectural

Brigham was addressing a mostly well-attended main stage at Amazon's re:Invent 2015 conference in Las Vegas Wednesday. His accompanying slide showed a certain well-recognized "2001" monolith, this time towering next to a brilliant Washington mountain lake, probably someplace nearby. And he was snickering just a bit at the cleverness of either Amazon's historical timing, or his own.

Also Sprach Zarathustra



"Now, don't get me wrong. It was architected in multiple tiers, and those tiers had many components in them," Brigham continued. "But they're all very tightly coupled together, where they behaved like one big monolith. Now, a lot of startups, and even projects inside of big companies, start out this way. They take a monolith-first approach, because it's very quick, to get moving quickly. But over time, as that project matures, as you add more developers on it, as it grows and the code base gets larger and the architecture gets more complex, that monolith is going to add overhead into your process, and that software development lifecycle is going to begin to slow down."

Brigham brought up the subject of the software development lifecycle (SDLC) as the fabric of a development team — specifically, as the substance that, when dealing with a monolithic application such as Amazon.com faced in 2001, kept developers apart from each other and separated the team from its ultimate goal. Their truly revolutionary approach to resolving this issue became one of the prototypes (NASA's Nebula project being one of the others) that led to the creation of cloud computing.

What Brigham also revealed is that the very tarball-like stickiness of the original Amazon.com inspired another revolutionary concept: the decoupling of service architectures. Maybe Amazon didn't invent microservices, and maybe it wasn't the pioneer of Agile methodologies. But evolution doesn't always happen in one place first. There's plenty of evidence that Amazon did come up with these concepts on its own initiative.

Brigham told the story of Amazon's engineering group, which in 2000 had the unenviable task of reconciling the in-process changes from hundreds of developers, resolving all the conflicts between them, merging them into a single version, and producing a master version that waits on the queue to be moved into production. "Even when you have that big, large, new version," he said, "it still adds a lot of overhead on this delivery pipeline. That whole, new code base needs to be rebuilt. All of the test cases need to be re-run, to make sure there have been no rushes. And then you need to take that entire application, and deploy it all to your full production fleet."

Amazon's approach is not to get rid of the pipeline, but to simplify it. The company's continuous deployment tools — CodeDeploy, CodePipeline and CodeCommit — are built around truly cloud-native web applications where the various segments of that pipeline can be scripted and automated. Rob Brigham said both the architectural and logistical changes that Amazon made, beginning at the turn of the century, directly led to the tools it now offers to development teams.

Evoking memories of the SpringOne 2GX conference last month, where Pivotal engineer Rohit Kelapure described in exhausting detail the decomposition of monolithic architectures, Amazon's Brigham related how he and his colleagues "teased it apart," with respect to the Amazon.com monolith, into a service-oriented architecture.

"We went through the code, and pulled out functional units that served a single purpose, and we wrapped those with a web service interface," he said. For example,

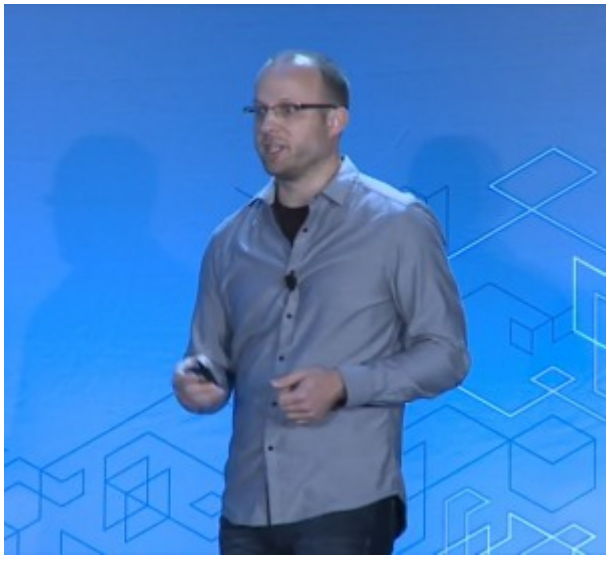
At the time of their creation, these single-purpose functions seemed simple enough to accomplish. But imagine the hundreds of development teams, some comprised of dozens of developers at that time (instead of the more comfortable "two-pizza" size of no more than eight), whose simple-enough, single-purpose functions had to be merged together week after week ... and later, month after month, as the fabric of the SDLC became bigger and bulkier.

The Decoupling Pipeline

The solution to the single-purpose function problem was the creation of a rule, to be adhered to by developers, that functions could only communicate with the rest of the world through their own web service APIs. "This enabled us to create a very highly decoupled architecture," said Brigham, "where these services could iterate independently from each other without any coordination between those services, as long as they adhered to that standard web service interface."

The decoupling of services enabled the creation of one of the first automated deployment systems, and the prototype for much of what Amazon offers customers today — appropriately named "Apollo." It helped introduce the pipeline model to the culture of Amazon, and it is probably here in the session that Brigham judiciously applied some discretionary editing, because that process could not have been simple.

But he was forthright about this: By being able to see a pipeline as a graphical thing, with size and shape, Amazon's engineers could get a firmer handle on just how much they needed to change about their processes. Sure, they could be automated, but why automate redundancy?



"We still noticed that it took a long time for a code change to go from a developer check-in, to be running in production where customers could use it," he related. "So being a data-driven company, we did a study on that. And we measured the amount of time it took a code change to make its way through that deployment lifecycle, across a number of teams. When we added up that data, and looked at the results, and saw the average time it took, we were frankly embarrassed. It was on the order of weeks."

Breaking down those actions helped engineers to realize that the sequence and arrangement of segments in this pipeline was leading to "dead time" — intervals where nothing was happening. This especially occurred in-between manual handoffs between departments — handoffs whose personalization was supposed to introduce process integrity, but which actually resulted, he said, in inefficiencies, wasted space, and long, long queues.

> "For a company like Amazon that prides itself on efficiency — for a company that uses robots inside of our fulfillment centers to move around physical goods, a company that wants to deploy packages to your doorstep using drones — you can imagine how crazy it was," he said, "that we were using humans to pass around these virtual bits in our software delivery process."

Brigham's talk led to a demo of CodePipeline at work, which included inline scripting of events that take place in the deployment pipeline, and integration with private repositories on Amazon as well as GitHub. Amazon took care at this point to show that it was avoiding the tack of locking development shops into an Amazon-branded way of doing things, contrary to what some outside its partner ecosystem have alleged.

You might think at this point that Rob Brigham was preaching to the choir, or that he was fishing for some overdue praise deserved by Amazon for helping to create this industry. The truth, embarrassing though it may be for a great many people in that audience and watching the live stream to admit, was that Amazon's story of 2001 was their story of 2015.