OPINION

The Death of System Administration

TODD UNDERWOOD

Todd Underwood is a site reliability director at Google. Before that, he was in charge of operations, security, and peering for Renesys, a provider of Internet intelligence services; and before that he was CTO of Oso Grande, a New Mexico ISP. He has a background in systems engineering and networking. Todd has presented work related to Internet routing dynamics and relationships at NANOG, RIPE, and various peering forums. tmu@google.com

I come to bury system administration, not to praise it.
The evil that well intentioned, clever geeks do lives after them;
The good is oft interred with their bones (and their previous jobs);
So let it be with system administration. The noble devops
Hath told you sysadmins were ambitious but insufficiently so:
If it were so, it was a grievous fault;
And grievously hath system administrators answer'd it.
Here, under leave of software engineers and the rest, —
For we are an honorable people;
So are they all, all honorable sysadmins, —
Come I to speak in sysadmins's funeral.
It was my friend, faithful and just to me and to all of us...

—with apologies to Bill Shakespeare

We are in the final days of system administration. I'm as nostalgic as many of you are. But I hope it is the same nostalgia that causes some of us old folks to fetishize terrible, unsafe, inefficient cars from the 1960s—a nostalgia borne of history but with no designs on the future. I hope we can cherish our shared history and accomplishments and then put them to rest in favor of a future where computers work for us rather than us for them. I dream of a future where we stop feeding the machines with human blood.

In the Beginning...ISPs

I grew up as a sysadmin first at a university and then at an Internet Service Provider (AS2901) during the 1990s. This is just what we did at the dawn of the Internet age. ISPs were a fantastic place to learn. We were all working massively beyond our capabilities and knowledge, gleefully skating over the edge of competence. At my ISP, we ran Sendmail (didn't you?). Everyone was still configured as an open relay, of course. One of the first things I did when I started was teach a class in locking down Sendmail. I didn't know how to lock down Sendmail. Obviously, I had to learn sendmail.cf syntax, m4, and at least a little bit about what the heck these spammers were doing with our mail systems. And I needed to teach a class on these things. In three days. Those were heady days.

Everyone did some of everything. The networking staff ran their own name servers, BIND 4 on AIX and FreeBSD. The systems staff often ran our own cable, and sometimes terminated it for ourselves, badly with plenty of near-end crosstalk. And the developers (sometimes) did their own backups. We were all learning and all building something that had never been seen before. As we went, we looked for more and better solutions. By the fourth time we had to modify our "new user creation" script, we were already pretty sick of it. We wanted centralized, automatable authentication and authorization. NIS? NIS+? RADIUS + some custom DB2 module (hat tip to Monte Mitzelfeld here)? LDAP! And that was better. Not perfect, just better. We traded the challenge of maintaining buggy, custom software for the challenge of maintaining, integrating, and automating LDAP. But things were moving on up.

It didn't help very much for the Windows hosting customers, and it didn't fix the entries in the billing database perfectly. But it was better.

At my next gigs, I tried to consistently automate away the toil, but the trivial toil (editing crontab by hand) was often replaced by larger scale toil (maintaining yum repositories, updating installer infrastructure, editing and testing Puppet configs). The toil became more interesting and the scale grew considerably, but I realized that the work itself didn't seem substantially better.

Site Reliability Engineering at Google

Google chose to solve this set of problems very differently. They invented something that's rapidly becoming an industry-standard role: Site Reliability Engineering. The history of exactly how and why that happened isn't written down and I wasn't there, which leaves me perfectly free to reimagine events. This almost certainly isn't what happened, but it is completely true.

Google was, and still is, incredibly ambitious and incredibly frugal. They had dreams of scaling beyond what anyone had seen before and dreams of doing it more affordably than anyone thought possible. This meant avoiding anything with costs that scaled linearly (or super-linearly) to users/queries/usage. This approach applied to hardware, and it most certainly applied to staff. They were determined to not have a NOC, not have an "operations" group that just administered machines. So, in the early days, software developers ran their own code in production, while building software to create the production environment.

Developers didn't like that very much, so they proceeded to automate away all of the annoying drudgery. They built automated test, build, push infrastructure. They built a cluster operating system that handled task restarting, collected logs, deployed tasks to machines, and did distributed tracing/debugging. Some of the software engineers working during these early days were much more production-oriented than others. These became the first site reliability engineers.

SRE at Google is composed of systems and software engineers who solve production problems with software. That's how it started and how it still is.

SRE != DevOps

The DevOps cultural movement that has been happening over the past few years is a huge improvement over the culture of system administration, IT, and operations that came before. It emphasizes collaboration among several siloed organizations and closer interaction between developers and the production environment they will work on. The rise of DevOps coincided with the conclusion of a many-year sysadmin obsession with configuration management systems (CFEngine, Puppet, anyone?). Sysadmins finally agreed that OS configuration management was drudgery and was best dealt with using configuration management tools.

As adoption and recognition of DevOps has grown, we have seen artificial and counterproductive barriers among organizational divisions fall. Well, not "fall," but at least "become somewhat more porous." We've also seen a bevy of software developed in the spirit of DevOps and claiming the DevOps mantle. This is mostly just marketing, where "DevOps" and "Cloud" are words that don't mean much more than "a buzzword to get this PO approved."

Still, almost everything is better now. We're not romanticizing the days when grizzled sysadmins built user accounts by useless-use-of-cat-ing individual bytes into /etc/passwd and /etc/shadow files by hand. We are slowly growing to realize that our human potential is much better suited to actual creative endeavors. More building; more thinking; less futzing.

But we're not even close to done. If there's one significant failing of the DevOps movement/community, it's that they don't hate operations nearly enough. They want to make operations better and to embed operational concerns and perspectives into software development, security, and networking. I think that's a good start along the way to killing operations entirely, for software-centric, Internet-built infrastructures.

Adrian Cockcroft of Netflix coined the term "NoOps" [1] to describe this vision of an operations-free future. I referred to a similar vision as "PostOps" recently [2]. This is the ambitious future, and near-present, that I think we should be collectively demanding. In this vision, the platform is a service and software developers code to it. The platform builds, deploys, and monitors your code. It balances load among your tasks, provisioning more resources when you need them. Your structured storage is provisioned according to your service needs, storage quantities, access patterns, and latency/availability requirements. Your unstructured storage is scalable and essentially infinite (or at least capable of tens of petabytes of capacity without significant planning). And when the infrastructure fails, it notifies you in a sensible, configurable, and correlated way. When the fault is with your code, debugging tools point to the problem. When the fault is with the infrastructure, mitigation is automatic.

We should be demanding more from our infrastructure providers than a quickly provisioned VM and some dashboards. We should be demanding a glorious future where we focus on interesting problems that really cannot be solved with software yet and rely on a platform to do the rest. When we get there, I'll be leading the charge towards the couch in the corner where we can sip bourbon and eat bon-bons. The extent to which DevOps encourages us to be satisfied with where we are is the extent to which DevOps and I part ways. I like what it's done so far, but I'm not satisfied.

Why You Probably Shouldn't Care About Operations Research

As the industry has been trying to figure out where to go, some people have suggested for the past few years that operations research is a really relevant source of academic thinking related to software operations. This is a field of study mostly focused on optimizing the output of a fixed set of constraints (traditionally a factory). The intent is to accurately model the set of constraints to understand where and why bottlenecks occur in the output. This facilitates intervention in the process design to improve output and efficiency.

Operations research is appealingly congruent to what we do in systems infrastructure. We try to build and maintain a set of infrastructure, with the intent of processing data with the minimum hardware and people, all the while maintaining service, availability, and performance within acceptable levels. Given a set of requirements ($x TiB of data at rest, $y queries per second of $z CPU cost per query on average), we should be able to provide the infrastructure specification that meets those requirements.

The problem arises as soon as you assume that the substrate below is fixed. It is not. It is software. Your network is software. Your chip's instruction set is software. Your OS is software. Your RPC marshalling/unmarshalling system is software. Your cluster file system is software. When it's convenient to think of each of these things as fixed, it is of course reasonable to do so. The constraints on the software stack, however, are massively less than the constraints on the machines in a factory. Your services infrastructure is exactly like a factory where a 3D printer can create new automation for each step at the same rate as you can specify and design it. We will have such factories, and when we do they will look much more like cloud infrastructure than cloud infrastructure looks like factories.

What's Next?

Let's back up for a second and review. Are operations and system administration already dead?

Not really. In some ways, the early part of this piece of writing is more exaggerated polemic than literal description. Our software and infrastructure are woefully incapable of actually handling a PostOps (or NoOps) world. For some time to come, the traditional sysadmin will continue to reinstall OSes, update configurations, troubleshoot network device drivers, specify new hardware, and reboot Windows boxes. But the writing is on the wall [2]. The computers are getting better and the tasks are getting more interesting. Most of these jobs will, and should, go away, to be replaced by a world with better, distributed infrastructure and more interesting jobs.

When that better, distributed infrastructure arrives, there are two things you need to know:

1. There will still be some operations to be done.
2. Operations will be done with software.

The wonderful, distributed self-regulating system of the future will not be stable and self-adjusting forever. It will encounter faults. Humans will have to intervene. When they do, operational sensibilities and approaches will be required to troubleshoot and resolve problems. So some amount of reactive, operational work endures. The scale and complexity will be much higher and the percentage of operational work will be much lower. You are definitely going to have to read, modify, and write software. If you're any kind of sysadmin, you already know how to code, but you may think you don't. You're already better than 50% of the people out there who are professionally employed as software engineers. Just get better than that.

As is usually the case, the future belongs to those who build it. So let us do that together—starting now.

References

[1] Adrian Cockcroft on NoOps: http://perfcap.blogspot.com/2012/03/ops-devops-and-noops-at-netflix.html.

[2] Todd Underwood, "PostOps: A Non-Surgical Tale of Software, Fragility, and Reliability," 27th Large Installation System Administration Conference (LISA '13): https://www.usenix.org/conference/lisa13/technical-sessions/plenary/underwood.

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Publish and Present Your Work at USENIX Conferences

The program committees of the following conferences are seeking submissions. CiteSeer ranks the USENIX Conference Proceedings among the the top ten highest-impact publication venues for computer science.

Get more details about each of these Calls for Papers and Participation at www.usenix.org/cfp.

JETS* Volume 2, Number 3: *USENIX Journal of Election and Technology and Systems
JETS is a new hybrid journal/conference, in which papers will have a journal-style reviewing process and online-only publication. Accepted papers for Volume 2, Numbers 1–3, will be presented at EVT/WOTE '14, which takes place August 18–19, 2014.

CSET '14: 7th Workshop on Cyber Security Experimentation and Test
Co-located with the USENIX Security '14
August 18, 2014, San Diego, CA
Submissions due: April 25, 2014, 11:59 p.m. EDT
CSET invites submissions on the science of cyber security evaluation, as well as experimentation, measurement, metrics, data, and simulations as those subjects relate to computer and network security and privacy.

3GSE '14: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education
Co-located with the USENIX Security '14
August 18, 2014, San Diego, CA
3GSE is designed to bring together educators and game designers working in the growing field of digital games, non-digital games, pervasive games, gamification, contests, and competitions for computer security education. The summit will attempt to represent, through invited talks, panels, and demonstrations, a variety of approaches and issues related to using games for security education.

FOCI '14: 4th USENIX Workshop on Free and Open Communications on the Internet
Co-located with the USENIX Security '14
August 18, 2014, San Diego, CA
Submissions due May 13, 2014, 11:59 p.m. PDT

HealthTech '14: 2014 USENIX Summit on Health Information Technologies
Safety, Security, Privacy, and Interoperability of Health Information Technologies
Co-located with the USENIX Security '14
August 19, 2014, San Diego, CA
Submissions due May 9, 2014, 11:59 p.m. PDT

LISA '14: 28th Large Installation System Administration Conference
November 9–14, 2014, Seattle, WA
Submissions due: April 14, 2014, 11:59 p.m. PDT
If you're an IT operations professional, site-reliability engineer, system administrator, architect, software engineer, researcher, or otherwise involved in ensuring that IT services are effectively delivered to others—this is your conference, and we'd love to have you here.

OSDI '14: 11th USENIX Symposium on Operating Systems Design and Implementation
October 6–8, 2014, Broomfield, CO
Abstract registration due: April 24, 2014, 9:00 p.m. PDT
OSDI brings together professionals from academic and industrial backgrounds in what has become a premier forum for discussing the design, implementation, and implications of systems software.