An Open Source Approach to Paying Off Infrastructure Technical Debt Blocking the Path to the Cloud

best practices and lessons learned

Presentation to the Linux Foundation Open FinTech Forum

Evan Bauer | OPSTACK

Technical debt is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer. Technical debt can be compared to monetary debt.

- *Wikipedia*

Technical Debt is a wonderful metaphor developed by Ward Cunningham to help us think about this problem. In this metaphor, doing things the quick and dirty way sets us up with a technical debt, which is similar to a financial debt.

– Martin Fowler

Operational technical debt is the implied cost of the approaches, processes, and technologies we own that don’t take into account the manual effort and lost opportunities — as well as the implied security and regulatory compliance costs that they generate.

As in software technical debt, the cost of the debt will increase on its own until paid off.

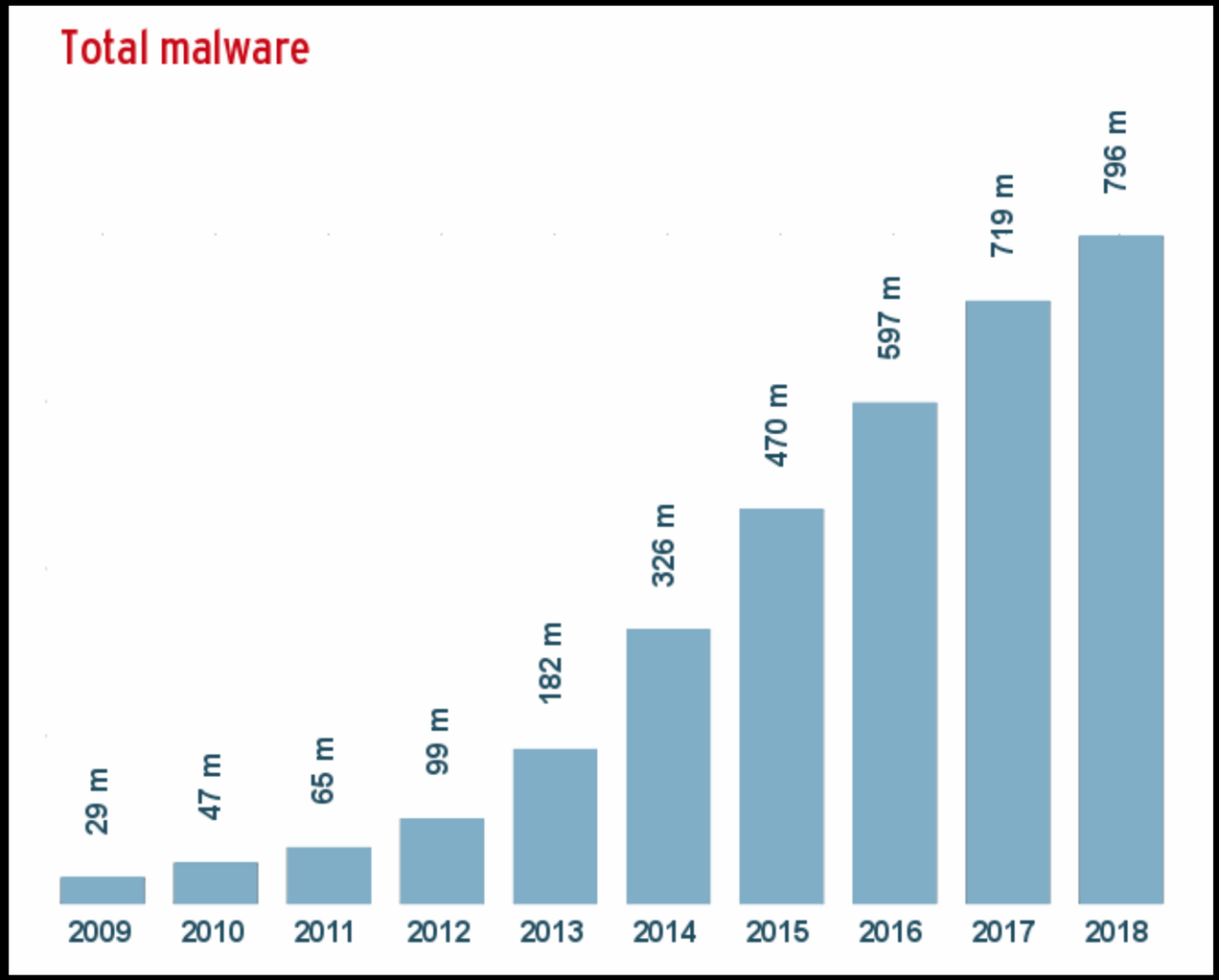
What happens when the ops team is too busy?

- Unacceptable provisioning time
- Manual, error-prone releases/changes
- Dev teams move to public clouds — but without compliance and violating segregation of duties
- Systems fail after patching attempts or for lack of hygiene (disk space, memory, expired certificates…)
- Maintenance is deferred, leaving systems vulnerable and fragile

Security team is distracted by all of the following, and wants them addressed first

- The number and duration of security policy exceptions are growing
- Not all systems are at current, fully-patched, version levels
- Scans show no decrease in the number of critical unpatched vulnerabilities
- Newly deployed servers and newly released applications fail security scans
- Unable to easily produce audit evidence of system compliance
- No tested and ready mechanism to immediately and effectively eliminate, mitigate, or control a zero-day critical vulnerability or a breach in progress

…and it is hard to blame them



Reported infections by year. Source: AV-Test Institute

…while business-as-usual increases the debt

- **M&A** adds diversity and complexity
- **Public cloud** adds sprawl without lifecycle support
- **DevOps** speeds deployments — but security and compliance is limited by the skills of each DevOps team
- **Increasingly complex architectures and deployments** cause application owners to resist patches and upgrades
- Increasing **numbers of systems** and **volume of data** overwhelm the capacity in place to maintain and protect them

The interest on technical debt compounds with complexity, volume, and the continual stream of new vulnerabilities and needed remediations.

No wonder ops can’t find the time to make DevOps and Cloud Native a reality for the whole institution

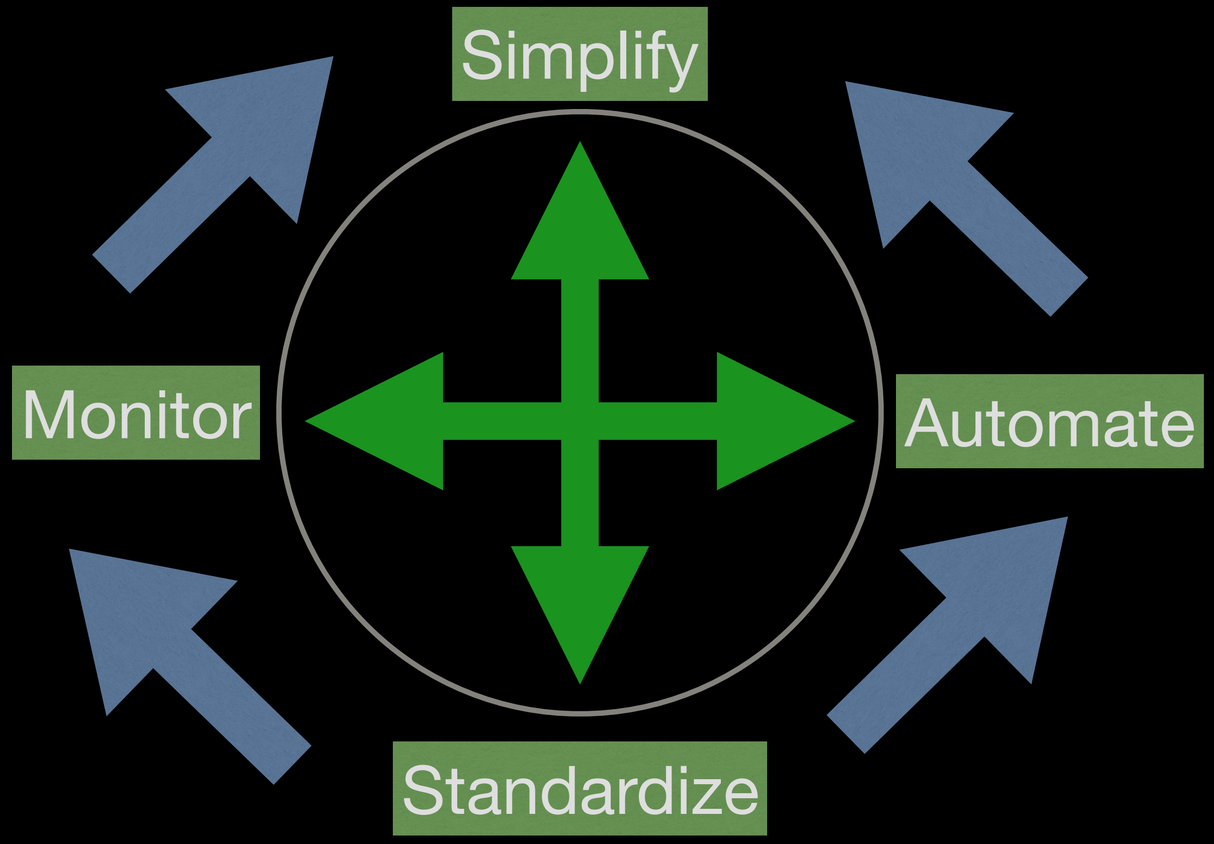
Manual processes and divided focus can’t keep up.

The only thing that works is to change the game.

Make all manual processes obsolete through automation

So what does this look like?

The four tenets of successful operations



Automate all builds and deployments

- Stop creating any new technical debt
- Create all new non-prod and production instances with an automated process — cloud, virtual, and physical
- Every server or container starts fully loaded and fully patched with all security, monitoring, and admin packages installed and tested — and is in the CMDB
- Access controls, group privileges, auditing, firewall configurations are all set as part of the build
- Make every server image easily replaceable with a clean build at any time
- By making it self-self-service, secure is easier than insecure

Patch Frequently and in Real-Time

- Patch (or rebuild) every system, at least once each month
- Respect the patch window — neither skip it, nor extend it
- Have a scheduled cadence for DEV, QA, UAT, PROD, and DR
- Monitor patching in real-time and fix whatever breaks in the window
- Keep patching atomic, any failures are corrected or rolled back in the window
- Make patching non-disruptive for applications served by clusters and farms

While SWAT teams tackle legacy applications

- Application teams too often treat servers as pets rather than cattle, this is especially true for complex applications with long lives
- Inject ops and security team members into dev and support teams to reverse engineer the code into the secure CI/CD process and create clean server builds to host them
- Simplify and expedite sign-offs for the applications that fully comply — reward good behavior

This produces a Win-Win for application owners (compliance with lower maintenance costs) and security — now it is possible to make the jump to cloud and cloud native

The solution is known, but where and how do you start?

- Skilled people are required to assess the technical estate and establish or verify standards
- New processes are needed to implement those standards — stopping the accrual of technical debt and paying down the outstanding balance
- When the operations team is tied down doing manual support (and occasional build) tasks, they don’t have time to implement an automation platform
- With all the pieces available (and no one “be and end all” product) there is a learning curve and a long lead-time to automated platforms and processes

What processes and technologies need to be in scope?

- System baselines
- Security policies
- Lifecycle policy
- Configuration management
- Deployment standards
- Deployment process
- Change control policy
- Maintenance schedule
- Scripting and orchestration
- Monitoring and Reporting
- Testing
- Repositories for source, apps, and external packages
- An operations control plane
- Interface to an ITSM

Why an Operations Stack?

There is no one product “silver bullet” — if there was you’d already have it.

It’s almost never a green field, the new stack should supplement and extend existing or legacy components.

These should then be tied together by an automation architecture and data models.

Open and Best-of-Breed

Scripting and Orchestration	Python | Salt | Ansible
Monitoring, Reporting, and Testing	ELK | Nageos | Selenium
Source, App, and Package Repos	Git | Artifactory | Uyuni
Operations Control Plane	Django | PostgreSQL
…and for Windows add	PowerShell | WSUS

People, Process, and Technology

An ITIL principle is that success isn't possible without all three. Too often we start by spending time and money finding and purchasing products — but a skilled team and a set of standard processes to automate has to come first.

- Injection of a skilled and experienced automation team that delivers the processes and platform along with full technology transfer to the operations team
- A clean set of ITIL-derived processes tailored to your needs
- A solid set of proven, open, integrated tools

Thoughts and Questions?

Evan Bauer

evan@opstack.us