DevOps for Legacy Systems

遗留系统的DevOps

Introducing Continuous Delivery, Infrastructure as Code, and similar

practices into a legacy environment can be a heavy lift. There are

usually a lot of different technology platforms and application architectures

to deal with, and outside of Linux and maybe Windows environments, there isn’t a lot of good DevOps tooling support available yet for many legacy systems.

在遗留环境中引入连续交付、基础架构即代码和类似实践可能是一个沉重的负担。通常有很多不同的技术平台和应用架构需要处理，对于许多遗留系统来说，并没有很多好的DevOps工具可用，除了在Linux和Windows环境下之外。

IBM and other enterprise vendors are jumping in to fill in the tooling

gap, with upgraded development and automated testing tools,cross-platform release automation solutions, and virtualized cloud services for testing. Organizations like Nationwide Insurance are implementing Continuous Integration and Continuous Delivery on zSeries mainframes, and a few other success stories prove that DevOps can work in a legacy enterprise environment.

 IBM和其他企业供应商正竞相涌入来弥补工具的缺乏，提供升级的开发和自动化测试工具，跨平台的发布自动化解决方案和虚拟化云测试服务。像**美国**全国保险公司这样的组织在zSeries大型机上实施持续集成和持续交付和其他一些成功案例证明了DevOps在遗留企业环境中同样有效。

There’s no reason not to try to speed up development and testing, or

to shift security left into design and coding in any environment. It’s

just good sense to make testing and production configurations match; to automate more of the compliance steps around change management and release management; and to get developers more involved with operations in configuring, packaging, deploying, and monitoring the system, regardless of technology issues.

没有理由不尝试去加速开发和测试，或者在任何环境中将安全转换为设计和编码。让测试和生产环境的配置匹配也很有道理；围绕变更管理和发布管理，将更多的合规步骤实现自动化；无论什么技术问题，让开发人员有更多参与运营中的配置、打包、部署和监控系统的机会，

But you will reach a point of diminishing returns as you run into limits of platform tooling and testability. According to Dave Farley:

Software that was written from scratch, using the high levels of automated testing inherent in Continuous Delivery, looks different from software that was not. Software written using automated testing to drive its design is more modular, more loosely coupled, and more flexible—it has to be to make it testable. This imposes a barrier

for companies looking to transition. There are successful strategies to make this transition but it is a challenge to the development culture, both business and technical, and at the technical level in terms of “how do you migrate a legacy system to make it testable?”19

但你终将因为平台工具和可测试性的限制而遇到收益拐点。根据戴夫·法利的说法：

从零开始编写的软件，通过持续交付中的高水平自动化测试，使之与没有自动化测试的软件不同。使用自动化测试驱动设计的软件更加模块化，更松耦合，并更灵活，为了能使其可测试。这对于寻求转型的公司成了障碍。有成功的实现这一转变的战略，但这是对开发文化，包括企业文化、技术文化形成了挑战，以及在技术层面上 “如何迁移旧系统使其成为可测试的？“19

 Legacy constraints in large enterprises lead to what McKinsey calls a

“two-speed IT architecture”, where you have two types of systems:

1. Slower-changing legacy backend “systems of record,” where all

the money is kept and counted

2. More agile frontend “systems of engagement,” where money is

made or lost—and where DevOps makes the most sense

DevOps adoption won’t be equal across the enterprise—at least, not

for a long time. But DevOps doesn’t have to be implemented everywhere

to realize real benefits. As the Puppet Labs “2015 State of

DevOps Report” found:

It doesn’t matter if your apps are greenfield, brownfield or legacy—

as long as they are architected with testability and deployability in

mind, high performance is achievable… The type of system—

whether it was a system of engagement or a system of record, packaged

or custom, legacy or greenfield—is not significant. Continuous

Delivery can be applied to any system.

大型企业的遗留限制导致麦肯锡所谓的“双速IT架构”，其中有两种类型的系统：

1。缓慢变化的遗留后端“记录系统”，其中所有的钱被保存并清点了

2。更灵活的前端“参与系统”，即赚钱或者亏钱的地方，也是DevOps最为有效的地方DevOps的采纳程度在整个企业中不会是均匀的，至少，在不是很长的一段时间内。但不必为了实现真正的收益而在企业任何地方都实施DevOps。作为Puppet实验室“2015年DevOps 状态报告“发现：

不管你的应用程序是绿地、棕地还是遗留系统的-只要它们在架构时考虑了可测试性和可部署性，高性能就是可以实现的…系统类型-无论是一个参与系统还是一个记录系统，商业套件或者客制化的、遗留系统或绿地系统，都不重要。持续交付可应用于任何系统。

From Infrastructure to Code

从基础设施到代码

It’s a massive job for an enterprise running thousands of apps on thousands of servers to move its infrastructure into code. Even with ITIL and other governance frameworks, many enterprises aren’t sure how many applications they run and where they are running,never mind aware of the details of how the systems are configured.

How are they supposed to get this information into code for tools

like Chef, Puppet, and Ansible?

对于一个在数千台服务器上运行数千个应用程序的企业来说，将其基础设施转换为代码这是一项艰巨的任务。即使有ITIL和其他治理框架，许多企业并不能确定他们运行了多少应用程序以及它们运行的位置，更不要说系统配置的细节了。他们应该如何将这些信息输入像Chef、Puppet和Ansible这些工具的代码中？

This is what a tech startup called UpGuard is taking on. UpGuard’s

cloud-based service captures configuration details from running

systems (physical or virtual servers, databases, or cloud services),

and tracks changes to this information over time. You can use it as a

Tripwire-like detective change control tool, to alert on changes to

configuration and track changes over time, or to audit and visualize

configuration management and identify inconsistencies and vulnerabilities.

这正是一家名为UpGuard的科技初创公司正在采取的行动。UpGuard基于云的服务可以从运行的系统中获取其详细配置信息（物理或虚拟服务器、数据库或云服务），并随着时间的推移跟踪对这些信息的变更。你可以把它当作类似TripWire的变更控制探测工具，用于当配置变更时产生告警，以及随时跟踪变更，或审核和可视化配置管理和识别不一致和漏洞。

UpGuard takes this much further, though. You can establish policies

for different systems or types of systems, and automatically create

fine-grained tests to check that the correct version of software is

installed on a system, that specific files or directories exist, that specific

ports are open or closed, or that certain processes are running.

UpGuard can also generate manifests that can be exported into

tools like Puppet, Chef, or Ansible, or Microsoft PowerShell DSC or

Docker. This allows you to bring infrastructure configuration into

code in an efficient and controlled way, with a prebuilt test framework.

不过，UpGuard将此更推进了一步。您可以对于不同的系统或系统类型来建立策略，并自动创建细粒度测试，以检查正确版本的软件是否被安装在系统上，特定的文件或目录是否存在，特定的端口是否打开或关闭，或者某些进程是否正在运行。UpGuard还可以生成可导出到的清单，并导出到如Puppet、Chef或Ansible，或Microsoft PowerShell DSC或Docker等工具。这允许您将基础设计的配置以有效且可控的方式转换成代码，并使用预构建的测试框架。