CHAPTER

8

Practical DevOps

Leveraging Existing Reference Sources, Roles, and Practices

In enterprise computing, DevOps will never operate in a vacuum. Over many years businesses have invested, adopted, and adapted many other methodologies and practices. For DevOps to be successful, this means many practices and existing roles (beyond development and operations) should be carefully reviewed and, if necessary, adjusted to drive improvements across the DevOps-enabled software factory.

DevOps and Enterprise Architecture

For many years, organizations have understood the dangers of technical debt. That is, the additional overhead arising when badly designed, poorly tested, and defect-ridden software is accepted for short-term gain. Analogous to financial debt, too much technical debt and the associated interest can cripple an organization to a point where they're constantly putting right previous wrongs at the expense of delivering new innovations.

Architectural debt is similar to technical debt and equally problematic. Poor architectural decisions can severely limit an organization's ability to move toward more agile styles of delivery. The result can be lower levels of innovation, slower time to market, or more cost and effort consumed rebuilding applications. These poor decisions are the consequence of conflicting architectural

A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_8

perspectives. On the one hand, scant regard for architecture and unconstrained development leads to software that is hard to integrate, support, and enhance. On the other, overly defined architecture is difficult and complex to implement, leading to delays in software delivery.

Some DevOps practitioners feel that they don't need to be guided Enterprise Architecture (EA). Conversely, many Enterprise Architects believe that rigorous dictates, methodologies, and practices must be adopted fully by DevOps programs. In truth, both agile development and DevOps cannot succeed without EA. However, the scope and application of the discipline must change to accelerate DevOps benefits without burdening the business with additional risk.

Without Good Architecture IT Builds Software Slums

The UK high-rise tower blocks built circa 1950-1980 were heralded as architectural wonders. Large towers housing the same population as the smaller pre-war houses they replaced. With large rooms, excellent views, and surrounding open spaces, they were initially lauded as the low-cost future for urban housing. However, there were many problems.

Due to cost cutting, substandard materials, demanding deadlines, and rushed construction practices, these modern-age housing miracles quickly became the new slums of the day. And by building tower-blocks across the country, town planners unwittingly replicated bad designs everywhere. The result: undesirable housing, elevated crime levels, and urban decay. Even the surrounding open areas and playgrounds were neglected because no one had ownership over maintenance or supervised them.

In many ways, this is analogous to many architectural design calamities made by IT in the past.

Built over many decades, rigid monolithic application designs have become difficult to scale and require costly maintenance. They're also tough to police and secure, with vandalism and theft being a constant problem. Add to this integration issues (IT's own unsupervised open spaces) and we're left with systems that are inadequate to meet the needs of modern digital business. That's not to say, however, that modern design approaches are the panacea. Web and microservices introduce new architectural issues, not the least, dependency management and monitoring complexity.

Enterprise Architecture Must Adapt to the Times

DevOps, with its focus on collaboration across the entire service lifecycle, is now seen as the answer to many of these issues. This is fine in principle, but without flexible EA, the end result could be IT developing application "slums," only more of them at an increased pace.

With some justification, many DevOps practitioners argue that heavy EA practices and frameworks haven't evolved to match the pace of agile. Iterative style development essentially means an end product very different from the initial idea, which necessitates a transition from set-in-stone architectures to a more fluid mix based on continuously evolving decisions. And if architects can't adapt, autonomous agile teams with easier access to open source and cloud-based resources are increasingly empowered to make architectural calls. But this can be problematic.

Great developers don't necessarily make great architects; they're not hard-wired that way. Charging down a development path without architectural guiderails, or tactically making team-based decisions, could address short-term development requirements but compromise broader program-level objectives. For example:

- Scalability and performance issues needed to address the business strategy of tripling the customer base and increasing satisfaction scores are neglected because development only concentrates on delivering more functionality.
- Developers acquire public cloud resources to accelerate application testing, but don't consider masking personally identifiable customer details when moving test data. As a result, organizations (especially those in heavily regulated industries) risk compliance breaches and heavy financial penalties.
- Because of team-based technical bias, a development group chooses one NoSQL database over the technology already used by another team. The existing technology would have met their needs (albeit with compromises), but their technology bias has significantly increased the support burden on IT operations.

It's important not to lay the blame squarely on development. Architects should recognize that any misalignment between existing processes (particularly the rigid ones) and the vision of an increasingly software-driven business will only further isolate the practice to the eventual detriment of the business.

New Fluid Guidelines and Principles

By incorporating the experience of enterprises architects within DevOps teams, organizations can maintain the pace of software delivery without introducing chaos. What's key, however, is to limit architectural over-engineering and provide development groups a minimum set of EA so as to avoid technical and architectural debt.

Even in minimal-mode, EA can still help developers quickly identify critical software design issues. The aim being to guide developers into understanding what architectural facets make up a successful application, and more importantly, how new ways of thinking can support more sustainable software innovation.

Interestingly, these practices are analogous to those that could be used to avoid building "housing slums":

Avoid substandard or unsupported materials—Development tools will most likely comprise both commercial and open source software and utilize both on-premise and cloud-based infrastructure. Teams must ensure application supportability becomes a key consideration, especially as more modern applications comprising multiple components move into production.

Note Just as the cost of maintaining UK housing tower-blocks reached unsustainable levels, so too will support burden on the organization when open source software isn't maintained and enhanced by the broader software community or lacks commercial backing. Enterprise architects should also work closely with development to understand where constraints lead to testing compromises—the "construction shortcuts" syndrome, resulting in compliance and quality issues.

New technologies shouldn't rehouse old problems—Enterprise architects should avoid rigid and inflexible edicts on technology usage. For example, mandating that all applications must be containerized, even legacy applications. This could be extremely problematic since the runtime-independent nature of containerized applications will extend the life of "problems" and provides no incentive to clear existing technical debt.

Note As UK high-rise housing tower-blocks degraded and vandalism increased, many local authorities tried to contain the situation by housing "problem groups" in the same units. Enterprise architects should avoid the same trap as they adopt technologies such as public cloud services and containers.

Monitor and manage sub-contractors—Whatever cloud model is adopted, abstracting away the infrastructure or application stack frees up development to focus on coding. But this shouldn't mean relinquishing visibility and control over application performance and the end user experience. To this end, enterprise architects must act as cloud-brokerage advisors. For example, working to ensure that cloud service providers incorporate open APIs with their offerings so that monitoring can be seamlessly incorporated into existing tools.

Building new digital services with DevOps is only part of the role of EA. The innovations of today may become legacies of the future, while many new apps must also integrate with existing systems that have strict compliance and risk controls. It's essential therefore that architecture covers both bases, working in minimal mode to ensure fast construction of quality services, but also applying standards and governance when needs dictate and systems change.

Actions to Establish EA in DevOps Programs

Some important steps needed to ensure EA becomes a sustainable contributor to a DevOps initiative include:

- *Communication*—To impress the importance of flexible architecture practices and standards. For example, API instrumentation shouldn't be mandated with rigid rules, but by carefully outlining how the practice improves software quality and supportability. This involves building closer ties with developers to ensure the right tooling decisions are being made. Enterprise architects should always emphasize that their participation isn't to slow down one particular team, but to ensure that team decisions (especially tooling) support broader program goals and objectives.
- *Collaboration*—In dynamic agile environments, it's natural for small teams to only focus on their own project and not consider the wider business context. Enterprise architects must apply flexible governance to ensure all stakeholders are involved in decision making without the system becoming overly bureaucratic. To this end, the governance approach must outline the need for big-picture strategy at an overall application portfolio level (complete with funding, commitment and risk management), together with support at a project level so to drive better outcomes in a business context. This support should start at the requirements phase and extend across the software development lifecycle.

DevOps and Information Security

It's a common misnomer that DevOps only involves closer collaboration between development and IT operations teams. In actuality, DevOps programs must also involve other disciplines that have traditionally been engaged late in the software development lifecycle. This is especially important with regard to information security.

At first glance it appears that the goals of DevOps and security are at odds. Whereas DevOps calls for increasing the delivery of high-quality software, security and compliance seeks careful and deliberate oversight to ensure the business isn't opening itself up to vulnerabilities. And with a mountain of rules and regulations to support, it's not surprising that security could easily become being regarded as another bottleneck in release and deployment processes.

All teams must accept that security is a key facet of "high-quality" software, which again can be established without slowing down development. There are four essential practices to consider:

Make everyone accountable for security—DevOps impresses the need shared responsibility and accountability. Therefore, security professionals should seek to build relationships with dev and ops teams and engage them as active stakeholders and participants in driving security improvements. As with enterprise architecture, this doesn't mean continually enforcing rigid and inflexible policies, but actually working collaboratively to assign security responsibilities to the team's best positioned to act on them. For example, during every application security incident, developers responsible for the actual code implicated should really be the first group called to help address the problem. These teams will be much more familiar with the software workings, plus the lessons they learn will help harden application security.

Demonstrate how DevOps improves security and vice versa—As organizations increasingly embrace DevOps, there'll be many new automated tools and practices introduced. As with everything new, these elements could introduce new threats and risk. Rather than see this as a problem, highly collaborative teams should work proactively to identify where additional guidance and controls are needed and can be applied without causing friction.

Take the development of a new mobile application for example. Here security experts can provide critical guidance on new threat surfaces, API governance requirements, and vulnerability testing. It's also important to consider that many new tools introduced (especially in areas like configuration management and release automation) also provide an opportunity for teams to build and improve security within the continuous delivery pipeline. To this end, it becomes less about making DevOps more secure and more about using DevOps (and especially automation) to improve security. This could include:

- Invoking techniques such as static code analysis during every application build, or providing development teams with comprehensive and fully automated security testing services that can be used repeatedly.
- Automatically creating the minimum set test cases with maximum security test coverage, right from the earliest stages of software development: the requirements phase.

- Reducing security test cycle preparation time by requesting and reserving accurate and compliant data from a test data repository.
- Generating realistic synthetic test data and incorporating directly into virtual or emulated services so as to improve testing quality while avoiding compliance exposures.

Shift security "left"—As with the traditional development to operations code handballing, the tendency has been to engage security very late in the development process. Too often, security teams are seen as the bottleneck police, holding up deployment with snap code audits and lengthy compliance checks. DevOps practices, however, enable security to be established during parallel development and testing. As code is developed, automated tests can be automatically invoked to continuously check and demonstrate compliance controls. This could include separation-of-duties and privileged user access controls, or masking personally identifiable customer information during cloud-based testing to demonstrate compliance.

Note By shifting security controls left into development and continuous delivery, it becomes easier to demonstrate compliance against a broad range of regulations (e.g., Federal Security Information Management Act—FISMA and General Data Protection Regulation—GDPR). High costs and delays resulting from auditors coming late into the process and finding the system isn't compliant may also be avoided.

As applications become increasingly complex and threats more pervasive, highly skilled security specialists will become highly prized and critical element to the success of any DevOps initiative. Organizations shouldn't make the mistake of assuming that developers themselves with a smattering of web application security experience can take on a full time security role (or will even want to), or that security staff (more used to maintaining security in legacy applications that infrequently change) can suddenly think like an agile developer. Over time, these skills will need to be developed by leveraging DevOps style collaboration. This could include agile teams inviting security to participate in user story development, stand up meetings, and retrospectives. For security professionals, it also means gaining credibility with a more detailed understanding of modern coding practices, providing faster feedback, and becoming an active voice in all security related discussions.

Rethinking Security Practices for DevOps

For many organizations, embedding security professionals into DevOps teams isn't practical. There just aren't enough of them and security operations may have problems scaling to handle a sudden influx of software changes. Addressing this requires a radical rethink on how to best apply security practices. This can involve:

- *Using security as a guiderail*—Security must take a lead in developing solutions and policies that all development teams can adopt. However, if teams gain management approval to bypass a policy because it slows them down, or the business unit accepts the trade-off is well worth the risk, security shouldn't stand in their way. Rather, security should measure their security capabilities and continue to inform teams about the risks of their actions.
- *Building closer collaboration with suppliers*—With applications moving to the cloud, organizations must work closely with software and cloud service providers to instruct what additional security controls and methods are needed in order to develop, test, and store information, without carrying additional risk. Businesses operating in different industry verticals will have specific compliance and data protection mandates, meaning providers must be willing to act in a more enterprise-friendly manner. This involves service providers including customers in development roadmaps and a willingness to support enterprise specific security requirements.
- *Making security a whole-of-business issue*—The role of security in DevOps should be to make security everybody's issue, not just the responsibility of the highly specialized security team. One effective way to do this is to develop a hierarchical security scoring system. If for example a deficient security practice is detected during the provisioning of a test environment, then the team responsible should be rated accordingly. That score should also bubble up to a group or divisional level. In this way, everyone (including senior management across business and IT) becomes more accountable.
- *Proactively involving security*—While it might not be practical to embed security specialization into every team, the security group can establish small teams charged with continuously testing security across the software

development pipeline. At regular times this team will focus on particular services (even groups of people) and use their expertise to hunt out vulnerabilities and log via established ticketing mechanisms with the appropriate classification and priority. During these exercises, no one should be immune from investigation or the activity limited to static systems. Even if teams are in the middle of a large important release, any severe problems detected must be addressed immediately.

Essential Characteristics of Security-Minded DevOps

As illustrated in Table 8-1, a security-minded DevOps program transcends beyond reacting and fixing security problems to protecting the business as applications are designed, developed, and tested.

Table 8-1. DevOps and Security: Organizational Mindset

Customers first	Mindset that enables security to be continually tailored according to customer needs and business outcomes.
Team alignment	Flexible organizational structures that enable security expertise to be embedded within development, testing, and operational functions.
Proactive engagement	Constantly assessing the security readiness across the software development lifecycle by introducing unplanned security events and threats.
Continuous investigation	Thorough analysis of external attempts to attack a business so teams can remediate security issues quickly and effectively.

DevOps and IT Service Management

While DevOps as a movement is relatively new and many organizations are in the early stages of adoption, most have heavily invested in more established methodologies and practices—especially ITIL®[1]. Since its inception in 1994, ITIL has been positioned as the most complete approach to IT management, with the exception of project management and enterprise architecture. It's not surprising then that upwards of two million people had some form of ITIL training (from foundational to expert) and that most enterprises have adopted many of the processes as detailed across the five ITIL volumes (service strategy, service design, service transition, service operations, and continuous improvement). For some, this starts and ends with service operations

[1]ITIL® is a (registered) trademark of AXELOS Limited. All rights reserved.

processes (especially the service desk function together with incident, problem, and change management problems), while others have embraced a fuller lifecycle-based approach to adoption.

DevOps and ITIL

Despite common misconceptions, ITIL is not specifically opposed to agile and DevOps thinking. For example, the service strategy volume promotes the notion of continuous improvement via feedback across the service lifecycle, while service design mentions agile and iterative design. However, despite the synergies, the general philosophy behind ITIL is one of rigorous sequential planning and control via process; opposite of the fast iterative design approach of agile development. ITIL also suggests that silos will continue to exist (albeit aligned around 26 processes), whereas the idea of smaller cross-functional style product teams, fast feedback, managing work in process, and small batch sizes is not well supported.

This appears to suggest that ITIL is becoming increasingly irrelevant with the practices under increased enterprise scrutiny. While this is perhaps true, it's important to appreciate where coexistence is practical and the immense value established ITIL processes can still deliver a DevOps program—especially core ITIL processes (e.g., incident and problem management).

Take problem management for example. This can provide considerable insight into the behavior and performance of a particular application, which can inform developers of needed non-functional improvements. This can also help teams avoid getting information (often conflicting) from a variety of sources. Additionally, a standard incident/problem ticketing method across development and operations may help improve teamwork and collaboration.

Overcoming Resistance

The biggest problem with respect to DevOps adoption is the resistance to change by established ITSM practitioners within the organization. Very often many roles are aligned around ITIL processes (change managers, problem managers, etc.), so it's natural people may resist DevOps if they feel their careers are threatened. To this end, DevOps and ITSM leaders must actively work to better understand how existing roles and practices can be enabler of successful DevOps capabilities and where refinements are needed. This could include:

Developing a better understanding of change—The adoption of ITIL has led many organizations to create the often-maligned Change Advisory Boards (CABs). Meeting infrequently, these groups have the unenvious task of approving production changes—essentially becoming the control point for delivery.

DevOps' approach to change is radically different. With DevOps, all change is encouraged unless it introduces greater risk and the increased probability of adverse customer and business impact. To this end, change isn't policed at the end of the cycle, but managed at the start of development. This is supported by many automated methods, including:

- Automatically establishing tests early, even when requirements are being established
- Maintaining consistent environmental configurations across development, test, and production
- Automated construction and invocation of testing during development, application builds, promotion, etc.
- Visibility and automation of complete release workflows

In dynamic agile environment, the rigor associated with CAB style change management may be too inflexible. It's important that ITIL roles readjust processes to accommodate the more continuous introduction of change. This could involve CAB leadership determining which DevOps-related changes do not require the normal process rigor and may bypass the traditional controls.

Integrating existing ITIL processes with DevOps—DevOps practitioners should collaborate with ITIL process owners to determine where integration with existing systems can drive improvements. Even if changes have to go through an approval process (e.g., for systems subject to compliance controls), this can be streamlined by integrating release automation with the change request process (normally maintained in a help desk system). Rather than rely on manually entering a change request separately, this activity would be incorporated in the actual release automation workflow itself, together with all necessary approval requests and escalations. In another example, security managers could work closely with DevOps practitioners, establishing technologies such as privileged access management to better accommodate the needs of developers without compromising strict compliance controls.

Participating in DevOps discussions—With many years of experience, ITIL practitioners can provide DevOps practitioners essential knowledge needed to drive improvements. An experienced IT operations manager could, for example, demonstrate how application performance management tools can be used in pre-production to identify any performance related problems undetected during earlier stages of testing. On the flip-side, ITIL process owners may benefit from newer tools introduced by agile and DevOps practitioners. Examples include configuration management and release automation solutions.

Beyond discussions and as illustrated in Table 8-2, there are many other practices that teams can adopt to reinforce the value of more collaborative behaviors.

Table 8-2. Example Behavioral Practices for DevOps and ITIL Teams

Practice	Example
Celebrate successes	Service managers present the business outcomes from a successful release together with lessons learned.
Improvement indicators	Service management team attends daily stand-up meetings and presents how data sharing has improved the supportability of a new application.
Recognition and rewards	At a joint weekly meeting, DevOps and service management teams jointly recognize individuals who have demonstrated strong collaboration and a shared commitment to driving improvements.
Reinforcing vision	At a weekly meeting, IT operations leadership reinforces the vision and goals of the DevOps program.
Satisfaction snapshots	Process owners are regularly engaged to assess the current level of engagement and support for a DevOps program.

DevOps and Lean Startup

While DevOps will need to coexist with and leverage traditional IT practices such as ITIL, there may be occasions when DevOps practices are very appropriate to help drive the adoption and success of newer business-driven methodologies. One such example is Lean Startup, a method for developing businesses and accelerating product development through a combination of hypothesis-driven experimentation, iterative product releases, and validated learning.

Originally proposed in 2008 by Eric Ries[2], the Lean Startup promise is to help businesses iteratively develop products to meet customer needs, while reducing market risks and avoiding heavy project funding and expensive product launches and failures.

DevOps practices have many synergies with Lean Startup, including:

- *Customer focus*—Lean Startup places great emphasis on validated learning, which is basically the process for understanding quickly what a customer actually needs or wants so that useful (and profitable) products can be developed. Agile and DevOps can help achieve this through the delivery of smaller units of work or iterations, after which results are validated.

[2]"The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation To Create Radically Successful Businesses," Eric Ries, September 2011

- *Fast feedback*—Lean Startup success depends heavily on continuous customer feedback during product development. This is to ensure that business don't invest unnecessary time and money developing features that customers don't want. DevOps supports this through two practices. First, by teams aligning activities to business outcomes and developing actionable metrics (discussed in Chapter 3), and secondly by employing continuous delivery processes to quickly release and test prototypes, experiments, etc.
- *Team collaboration*—With Lean Startup promoting the continuous running of experiments and validation, it is critical that all cross-functional teams operate in unison. In such fast-paced environments, any form of waste like long cycle times and delays hampers validated learning and inhibits the flow of value. It's why DevOps practitioners supporting a Lean Startup model place great emphasis on reducing all elements of waste across the software lifecycle (see Chapter 3).

Summary

Being myopically focused of DevOps to such an extent that existing bodies of knowledge and best practices are shunned or ignored is a recipe for disaster. That said, organizations must also be ready to adjust existing methods, roles, and practices. As this chapter illustrates, this includes enterprise architecture and information security.

In the next chapter, we'll describe important tangible business benefits and strategies needed to accelerate DevOps ROI.