Compliments of IBM

2nd IBM Limited Edition

DevOps

FOR DUMMIES®

A Wiley Brand

Learn:

- The business need and value of DevOps
- DevOps capabilities and adoption paths
- How cloud accelerates DevOps
- Ten DevOps myths

Sanjeev Sharma
Bernie Coyne

DevOps

FOR DUMMIES®

A Wiley Brand

2nd IBM Limited Edition

by Sanjeev Sharma and Bernie Coyne

FOR DUMMIES®

A Wiley Brand

DevOps For Dummies®, 2nd IBM Limited Edition
Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN: 978-1-119-04705-6 (pbk); ISBN: 978-1-119-04729-2 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Editor: Carrie A. Johnson
Acquisitions Editor: Katie Mohr
Editorial Manager: Rev Mengle
Business Development Representative: Sue Blessing
Production Coordinator: Melissa Cossell

Table of Contents

Introduction 1

About This Book 1
Icons Used in This Book 2
Beyond the Book 2

Chapter 1: What Is DevOps? 3

Understanding the Business Need for DevOps 3
Recognizing the Business Value of DevOps 4
Enhanced customer experience 5
Increased capacity to innovate 5
Faster time to value 6
Seeing How DevOps Works 6
Develop and test against production-like systems 6
Deploy with repeatable, reliable processes 7
Monitor and validate operational quality 8
Amplify feedback loops 8

Chapter 2: Looking at DevOps Capabilities 9

Paths to DevOps Adoption 9
Steer 10
Develop/Test 11
Collaborative development 12
Continuous testing 13
Deploy 13
Operate 14
Continuous monitoring 14
Continuous customer feedback and optimization 14

Chapter 3: Adopting DevOps 15

Knowing Where to Begin 15
Identifying business objectives 16
Identifying bottlenecks in the delivery pipeline 16
People in DevOps 17
DevOps culture 17
DevOps team 19
Process in DevOps 19
DevOps as a business process 19
Change management process 20
DevOps techniques 21

Technology in DevOps 24
Infrastructure as code 25
Delivery pipeline 26
Deployment automation and release management 28

Chapter 4: Looking at How Cloud Accelerates DevOps . . . 31

Using Cloud as an Enabler for DevOps 32
Full-Stack Deployments 34
Choosing a Cloud Service Model for DevOps 35
IaaS 35
PaaS 37
Understanding What a Hybrid Cloud Is 38

Chapter 5: Using DevOps to Solve New Challenges . . . 41

Mobile Applications 42
ALM Processes 43
Scaling Agile 43
Multiple-Tier Applications 44
DevOps in the Enterprise 45
Supply Chains 46
The Internet of Things 46

Chapter 6: Making DevOps Work: IBM's Story 49

Taking a Look at the Executive's Role 50
Putting Together the Team 51
Setting DevOps Goals 51
Learning from the DevOps Transformation 52
Expanding agile practices 52
Leveraging test automation 53
Building a delivery pipeline 54
Experimenting rapidly 56
Continuously improving 57
Looking at the DevOps Results 58

Chapter 7: Ten DevOps Myths 59

DevOps Is Only for "Born on the Web" Shops 59
DevOps Is Operations Learning How to Code 60
DevOps Is Just for Development and Operations 60
DevOps Isn't for ITIL Shops 60
DevOps Isn't for Regulated Industries 61
DevOps Isn't for Outsourced Development 61
No Cloud Means No DevOps 61
DevOps Isn't for Large, Complex Systems 62
DevOps Is Only about Communication 62
DevOps Means Continuous Change Deployment 62

Introduction

DevOps (short for development and operations), like most new approaches, is only a buzzword for many people. Everyone talks about it, but not everyone knows what it is. In broad terms, DevOps is an approach based on lean and agile principles in which business owners and the development, operations, and quality assurance departments collaborate to deliver software in a continuous manner that enables the business to more quickly seize market opportunities and reduce the time to include customer feedback. Indeed, enterprise applications are so diverse and composed of multiple technologies, databases, end-user devices, and so on, that only a DevOps approach will be successful when dealing with these complexities. Opinions differ on how to use it, however.

Some people say that DevOps is for practitioners only; others say that it revolves around the cloud. IBM takes a broad and holistic view and sees DevOps as a business-driven software delivery approach — an approach that takes a new or enhanced business capability from an idea all the way to production, providing business value to customers in an efficient manner and capturing feedback as customers engage with the capability. To do this, you need participation from stakeholders beyond just the development and operations teams. A true DevOps approach includes lines of business, practitioners, executives, partners, suppliers, and so on.

About This Book

This book takes a business-centric approach to DevOps. Today's fast-moving world makes DevOps essential to all enterprises that must be agile and lean enough to respond rapidly to changes such as customer demands, market conditions, competitive pressures, or regulatory requirements.

If you're reading this book, we assume that you've heard about DevOps but want to understand what it means and how your company can gain business benefits from it. This book is

geared to executives, decision makers, and practitioners who are new to the field of DevOps, who seek more information about the approach, and who want to cut through the hype surrounding the concept to get to the meat of it.

Icons Used in This Book

You'll find several icons in the margins of this book. Here's what they mean.

The Tip icon points out helpful information on various aspects of DevOps.

Anything that has a Remember icon is something that you want to keep in mind.

The Warning icon alerts you to critical information.

Technical Stuff material goes beyond the basics of DevOps. It isn't essential reading, however.

Beyond the Book

You can find additional information about DevOps and IBM's approach and services available by visiting the following web pages:

- **IBM DevOps Solution:** `ibm.com/devops`
- **DevOps — the IBM approach (white paper):** `ibm.biz/BdEnBz`
- **The Software Edge (study):** `ibm.co/156KdoO`
- **Adopting the IBM DevOps Approach (article):** `ibm.biz/adoptingdevops`
- **DevOps Services for Bluemix (service):** `bluemix.net`

Chapter 1

What Is DevOps?

In This Chapter

- Seeing a business need for DevOps
- Finding business value in DevOps
- Understanding DevOps principles

Making any change in "business as usual" is always hard and usually requires an investment. So whenever an organization adopts any new technology, methodology, or approach, that adoption has to be driven by a business need. To develop a business case for adopting DevOps, you must understand the business need for it, including the challenges that it addresses. In this chapter, we give you the foundation you need to start building your case.

Understanding the Business Need for DevOps

Organizations want to create innovative applications or services to solve business problems. They may want to address internal business problems (such as creating a better customer relationship management system) or to help their customers or end-users (such as by providing a new mobile app).

Many organizations aren't successful with software projects, however, and their failures are often related to challenges in software development and delivery. Although most enterprises feel that software development and delivery are critical, a recent IBM survey of the industry found that only 25 percent believe that their teams are effective. This execution gap leads to missed business opportunities.

This problem is further amplified by a major shift in the types of applications that businesses are required to deliver, from systems of record to systems of engagement:

- **Systems of record:** Traditional software applications are large systems that function as systems of record, which contain massive amounts of data and/or transactions and are designed to be highly reliable and stable. Because these applications don't need to change often, organizations can satisfy their customers and their own business needs by delivering only one or two large new releases a year.
- **Systems of engagement:** With the advent of mobile communications and the maturation of web applications, systems of record are being supplemented by systems of engagement, which customers can access directly and use to interact with the business. Such applications must be easy to use, high performing, and capable of rapid change to address customers' changing needs and evolving market forces.

Because systems of engagement are used directly by customers, they require intense focus on user experience, speed of delivery, and agility — in other words, a DevOps approach.

Systems of engagement aren't isolated islands and are often tied to systems of record, so rapid changes to systems of engagement result in changes to systems of record. Indeed any kind of system that needs rapid delivery of innovation requires DevOps. Such innovation is driven primarily by emerging technology trends such as cloud computing, mobile applications, Big Data, and social media, which may affect all types of systems. We discuss these emerging technologies in light of DevOps in Chapters 4 and 5.

Recognizing the Business Value of DevOps

DevOps applies agile and lean principles across the entire software supply chain. It enables a business to maximize the speed of its delivery of a product or service, from initial idea to production release to customer feedback to enhancements based on that feedback.

Because DevOps improves the way that a business delivers value to its customers, suppliers, and partners, it's an essential business process, not just an IT capability.

DevOps provides significant return on investment in three areas:

- Enhanced customer experience
- Increased capacity to innovate
- Faster time to value

We discuss all three areas in the following sections.

Enhanced customer experience

Delivering an enhanced (that is, differentiated and engaging) customer experience builds customer loyalty and increases market share. To deliver this experience, a business must continuously obtain and respond to customer feedback, which requires mechanisms to get fast feedback from all the stakeholders in the software application that's being delivered: customers, lines of business, users, suppliers, partners, and so on.

In today's world of systems of engagement (see "Understanding the Business Need for DevOps," earlier in this chapter), this ability to react and adapt in an agile manner leads to enhanced customer experience and loyalty.

Increased capacity to innovate

Modern organizations use lean thinking approaches to increase their capacity to innovate. Their goals are to reduce waste and rework and to shift resources to higher-value activities.

An example of a common practice in lean thinking is *A-B testing,* in which organizations ask a small group of users to test and rate two or more sets of software that have different capabilities. Then the better-capability set is rolled out to all users, and the unsuccessful version is rolled back. Such A-B testing is realistic only with efficient and automated mechanisms such as those that DevOps facilitates.

Faster time to value

Speeding time to value involves developing a culture, practices, and automation that allow for fast, efficient, and reliable software delivery through to production. DevOps, when adopted as a business capability, provides the tools and culture required to facilitate efficient release planning, predictability, and success.

The definition of *value* varies from organization to organization and even from project to project, but the goal of DevOps is to deliver this value faster and more efficiently.

Seeing How DevOps Works

The DevOps movement has produced several principles that have evolved over time and are still evolving. Several solution providers, including IBM, have developed their own variants. All these principles, however, take a holistic approach to DevOps, and organizations of all sizes can adopt them. These principles are

- Develop and test against production-like systems
- Deploy with repeatable, reliable processes
- Monitor and validate operational quality
- Amplify feedback loops

We describe the principles in more detail in the following sections.

Develop and test against production-like systems

This principle stems from the DevOps concept *shift left,* in which operations concerns move earlier in the software delivery life cycle, toward development (see Figure 1-1).

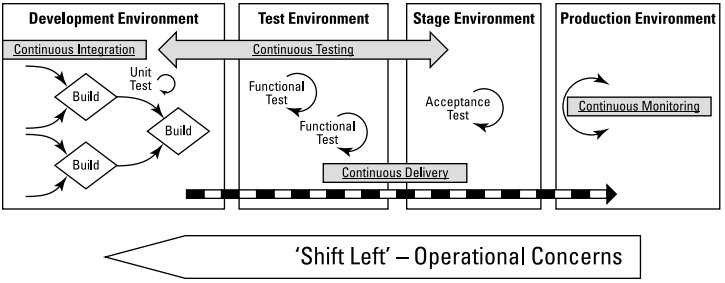


Figure 1-1: The shift-left concept moves operations earlier in the development life cycle.

The goal is to allow development and quality assurance (QA) teams to develop and test against systems that behave like the production system, so that they can see how the application behaves and performs well before it's ready for deployment.

The first exposure of the application to a production-like system should be as early in the life cycle as possible to address two major potential challenges. First, it allows the application to be tested in an environment that's close to the actual production environment the application will be delivered to; and second, it allows for the application delivery processes themselves to be tested and validated upfront.

From an operations perspective, too, this principle has tremendous value. It enables the operations team to see early in the cycle how their environment will behave when it supports the application, thereby allowing them to create a fine-tuned, application-aware environment.

Deploy with repeatable, reliable processes

As the name suggests, this principle allows development and operations to support an agile (or at least iterative) software development process all the way through to production. Automation is essential to create processes that are iterative, frequent, repeatable, and reliable, so the organization must create a delivery pipeline that allows for continuous,

automated deployment and testing. We talk more about delivery pipelines in Chapter 3.

Frequent deployments also allow teams to test the deployment processes themselves, thereby lowering the risk of deployment failures at release time.

Monitor and validate operational quality

Organizations typically are good at monitoring applications and systems in production because they have tools that capture production systems' metrics in real time. But they monitor in a siloed and disconnected manner. This principle moves monitoring earlier in the life cycle by requiring that automated testing be done early and often in the life cycle to monitor functional and non-functional characteristics of the application. Whenever an application is deployed and tested, quality metrics should be captured and analyzed. Frequent monitoring provides early warning about operational and quality issues that may occur in production.

These metrics should be captured in a format that all business stakeholders can understand and use.

Amplify feedback loops

One goal of DevOps is to enable organizations to react and make changes more rapidly. In software delivery, this goal requires an organization to get quick feedback and then learn rapidly from every action it takes. This principle calls for organizations to create communication channels that allow all stakeholders to access and act on feedback.

- Development may act by adjusting its project plans or priorities.
- Production may act by enhancing the production environments.
- Business may act by modifying its release plans.

Chapter 2

Looking at DevOps Capabilities

In This Chapter

- Understanding the reference architecture of DevOps
- Considering four paths to DevOps adoption

The capabilities that make up DevOps are a broad set that span the software delivery life cycle. Where an organization starts with DevOps depends on its business objectives and goals — what challenges it's trying to address and what gaps in its software delivery capabilities need to be filled.

In this chapter, you look at a DevOps reference architecture and the various ways that it enables a business to use DevOps.

Paths to DevOps Adoption

A *reference architecture* provides a template of a proven solution by using a set of preferred methods and capabilities. The DevOps reference architectures discussed in this chapter help practitioners access and use the guidelines, directives, and other material that they need to architect or design a DevOps platform that accommodates people, processes, and technology (see Chapter 3).

A reference architecture provides capabilities through its various components. These capabilities in turn may be provided by a single component or a group of components working together. Therefore, you can view the DevOps reference

architecture, shown in Figure 2-1, from the perspective of the core capabilities that it's intended to provide. As the abstract architecture evolves to concrete form, these capabilities are provided by a set of effectively enabled people, defined practices, and automation tools.

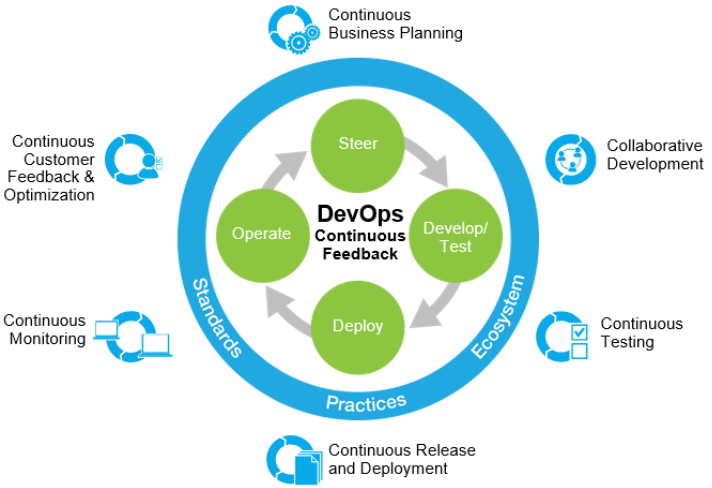


Figure 2-1: The DevOps reference architecture.

The DevOps reference architecture shown in Figure 2-1 proposes the following four sets of adoption paths:

- Steer
- Develop/Test
- Deploy
- Operate

In the remaining sections of this chapter, you take a detailed look at these adoption paths.

Steer

This adoption path consists of one practice that focuses on establishing business goals and adjusting them based on customer feedback: *continuous business planning*.

Businesses today need to be agile and able to react quickly to customer feedback. Achieving this goal centers on an organization's ability to do things right. Unfortunately, traditional approaches to product delivery are too slow for today's speed of doing business, partially because these approaches depend on custom development and manual processes and because teams are operating in silos. Information required to plan and re-plan quickly, while maximizing the ability to deliver value, is fragmented and inconsistent. Often the right feedback isn't received early enough to achieve the right level of quality to truly deliver value.

Teams also struggle to incorporate feedback that should inform the prioritization of investments and then to collaborate as an organization to drive execution in a continuous delivery model. For some teams, planning is viewed as governance overhead that's intrusive and slows them down instead of an activity that enables them to deliver value with speed.

Faster delivery provides greater business agility, but you must also manage speed with the trust and confidence that what you've delivered is the right thing. You can't deliver software at speed if you don't trust the accuracy of your business goals, your measurements, and your platforms.

DevOps helps to reconcile these competing perspectives, helping teams collaboratively establish business goals and continuously change them based on customer feedback thereby improving both agility and business outcomes. At the same time, businesses need to manage costs. By identifying and eliminating waste in the development process, the team becomes more efficient but also addresses cost. This approach helps teams strike an optimal balance between all these considerations, across all phases of the DevOps life cycle in moving to a continuous delivery model.

Develop/Test

This adoption path involves two practices: collaborative development and continuous testing. As such, it forms the core of development and quality assurance (QA) capabilities.

Collaborative development

Software delivery efforts in an enterprise involve large numbers of cross-functional teams, including lines-of-business owners, business analysts, enterprise and software architects, developers, QA practitioners, operations personnel, security specialists, suppliers, and partners. Practitioners from these teams work on multiple platforms and may be spread across multiple locations. *Collaborative development* enables these practitioners to work together by providing a common set of practices and a common platform they can use to create and deliver software.

One core capability included within collaborative development is *continuous integration* (see Figure 2-2), a practice in which software developers continuously or frequently integrate their work with that of other members of the development team.

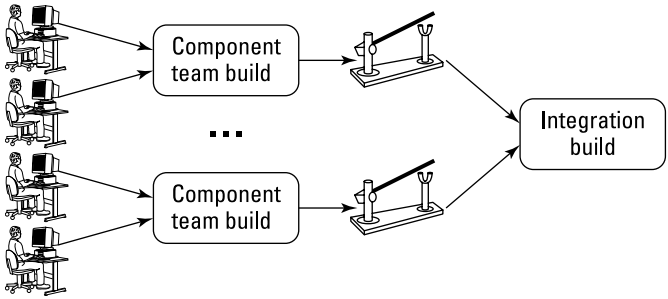


Figure 2-2: Collaboration via continuous integration.

Continuous integration was made popular by the agile movement. The idea is for developers to regularly integrate their work with that of the rest of the developers on their team and then test the integrated work. In the case of complex systems made up of multiple systems or services, developers also regularly integrate their work with other systems and services. Regular integration of results leads to early discovery and exposure of integration risks. In complex systems, it also exposes known and unknown risks — both technical and schedule-related.

Continuous testing

Continuous integration (see the preceding section) has several goals:

- Enable ongoing testing and verification of code
- Validate that the code produced and integrated with that of other developers and other components of the application functions and performs as designed
- Continuously test the application being developed

Continuous testing means testing earlier and continuously across the life cycle, which results in reduced costs, shortened testing cycles, and achieved continuous feedback on quality. This process is also known as *shift-left testing*, which stresses integrating development and testing activities to ensure quality is built-in as early in the life cycle as possible and not something left to later. This is facilitated by adopting capabilities like automated testing and service virtualization. Service virtualization is the new capability for simulation of production-like environments and makes continuous testing feasible.

Deploy

The Deploy adoption path is where most of the root capabilities of DevOps originated. Continuous release and deployment take the concept of continuous integration to the next step. The practice that enables release and deploy also enables the creation of a delivery pipeline (see Chapter 3). This pipeline facilitates continuous deployment of software to QA and then to production in an efficient, automated manner. The goal of continuous release and deployment is to release new features to customers and users as soon as possible.

Most of the tooling and processes that make up the core of DevOps technology exist to facilitate continuous integration, continuous release, and continuous deployment. I discuss these topics in more detail in later chapters.

Operate

The Operate adoption path includes two practices that allow businesses to monitor how released applications are performing in production and to receive feedback from customers. This data allows the businesses to react in an agile manner and change their business plans as necessary.

Continuous monitoring

Continuous monitoring provides data and metrics to operations, QA, development, lines-of-business personnel, and other stakeholders about applications at different stages of the delivery cycle.

These metrics aren't limited to production. Such metrics allow stakeholders to react by enhancing or changing the features being delivered and/or the business plans required to deliver them.

Continuous customer feedback and optimization

The two most important types of information that a software delivery team can get are data about how customers use the application and feedback that those customers provide upon using the application. New technologies allow businesses to capture customer behavior and customer pain points right as they use the application. This feedback allows different stakeholders to take appropriate actions to improve the applications and enhance customer experience. Lines of business may adjust their business plans, development may adjust the capabilities it delivers, and operations may enhance the environment in which the application is deployed. This continuous feedback loop is an essential component of DevOps, allowing businesses to be more agile and responsive to customer needs.

Chapter 3

Adopting DevOps

In This Chapter

- Making people more efficient
- Streamlining processes
- Choosing the right tools

Adopting any new capability typically requires a plan that spans people, process, and technology. You can't succeed in adopting new capabilities — especially in an enterprise that has multiple, potentially distributed stakeholders — without taking into consideration all three aspects of the capabilities being adopted.

In this chapter, we discuss the people, process, and technology aspects of DevOps.

Although the name *DevOps* suggests development-and-operations-based capabilities, DevOps is an enterprise capability that spans all stakeholders in an organization, including business owners, architecture, design, development, quality assurance (QA), operations, security, partners, and suppliers. Excluding any stakeholder — internal or external — leads to an incomplete implementation of DevOps.

Knowing Where to Begin

This section provides guidance on how to get started with DevOps, including creating the right culture, identifying business challenges, and finding those bottlenecks to eliminate.

Identifying business objectives

The first task in creating a culture is getting everyone headed in the same direction and working toward the same goal, which means identifying common business objectives for the team and the organization as a whole. It's important to incent the entire team based on business outcomes versus conflicting team incentives. When people know what common goal they're working toward and how their progress toward that goal is going to be measured, fewer challenges exist from teams or practitioners that have their own priorities.

DevOps isn't the goal. It helps you reach your goals.

Chapters 4 and 5 highlight several new business challenges that DevOps addresses. Your organization can use those challenges as a starting point to identify goals that it wants to achieve; then it can develop a common set of milestones toward those goals for different teams of stakeholders to use.

Identifying bottlenecks in the delivery pipeline

The biggest sources of inefficiencies in the delivery pipeline have been categorized as the following:

- Unnecessary overhead (having to repeatedly communicate the same information and knowledge)
- Unnecessary rework (defects being uncovered in testing or production forcing assignments back to the development team)
- Over-production (functionality developed that wasn't required)

One of the biggest bottlenecks in the delivery pipeline is deploying infrastructure. The adoption of a DevOps approach increases the velocity of application delivery and puts pressure on the infrastructure to respond more quickly. That is where software-defined environments enable you to capture infrastructure as a kind of programmable and repeatable pattern, thereby accelerating deployments. Check out the

section "Infrastructure as Code" later in this chapter for more information.

Drilling down further, you may want to optimize the pipeline with an even flow from end to end. The throughput of each process must be equal in order to avoid backlogs. To help achieve this balance, you need to instrument or measure the delivery pipeline at key points so you can minimize the wait time in backlog queues, optimize the work in progress, and adjust capacity and flow.

People in DevOps

This section addresses the people aspect of adopting DevOps, including creating the necessary culture.

DevOps culture

At its root, DevOps is a cultural movement; it's all about people. An organization may adopt the most efficient processes or automated tools possible, but they're useless without the people who eventually must execute those processes and use those tools. Building a DevOps culture, therefore, is at the core of DevOps adoption.

A DevOps culture is characterized by a high degree of collaboration across roles, focus on business instead of departmental objectives, trust, and high value placed on learning through experimentation.

Building a culture isn't like adopting a process or a tool. It requires (for lack of a better term) social engineering of teams of people, each with unique predispositions, experiences, and biases. This diversity can make culture-building challenging and difficult.

Lean and agile transformation practices such as Scaled Agile Framework (SAFe), Disciplined Agile Delivery (DAD), and Scrum are at the core of DevOps, and if your organization has already applied these, they can be leveraged to help adopt a DevOps culture.

Measuring culture

Measuring culture is extremely difficult. How do you accurately measure improved collaboration or improved morale? You could take a direct measure of attitudes and team morale by taking surveys, but surveys can have a high statistical error rate, as teams usually are small.

Conversely, you can take an indirect measure by tracking how often a development team member reaches out to a member of an operations or QA team to collaborate on resolving an issue without going through official channels or multiple layers of management.

Collaboration and communication across stakeholders — that's the culture of DevOps.

Building a DevOps culture requires the leaders of the organization to work with their teams to create an environment and culture of collaboration and sharing. Leaders must remove any self-imposed barriers to cooperation. Typical measurements reward operations teams for uptime and stability, and reward developers for new features delivered, but they pit these groups against each other. Operations knows that the best protection for production is to accept no changes, for example, and Development has little incentive to focus on quality. Replace these measurements with shared responsibility for delivering new capabilities quickly and safely.

The leaders of the organization should further encourage collaboration by improving visibility. Establishing a common set of collaboration tools is essential, especially when teams are geographically distributed and can't work together in person. Giving all stakeholders visibility into a project's goals and status is crucial for building a DevOps culture based on trust and collaboration.

Sometimes, building a DevOps culture requires people to change. Those who are unwilling to change — that is, to adopt the DevOps culture — may need to be reassigned.

DevOps team

The arguments for and against having a separate DevOps team are as old as the concept itself. Some organizations, such as Netflix, don't have separate development and operations teams; instead, a single "NoOps" team owns both sets of responsibilities. Other organizations have succeeded with DevOps liaison teams, which resolve any conflicts and promote collaboration. Such a team may be an existing tools group or process group, or it may be a new team staffed by representatives of all teams that have a stake in the application being delivered.

If you choose to have a DevOps team, your most important goal is to ensure that it functions as a center of excellence that facilitates collaboration without adding a new layer of bureaucracy or becoming the team that owns addressing all DevOps related problems — a development that would defeat the purpose of adopting a DevOps culture.

Process in DevOps

In the preceding section, we discussed the role of people and culture in adopting DevOps. Processes define what those people do. Your organization can have a great culture of collaboration, but if people are doing the wrong things or doing the right things in the wrong way, failure is still likely.

A vast number of processes are identified with DevOps — too many to cover in this book. This section discusses some of the key processes in light of their adoption across an enterprise.

DevOps as a business process

DevOps as a capability affects a whole business. It makes the business more agile and improves its delivery of capabilities to customers. You can extend DevOps further by looking at it as a *business process:* a collection of activities or tasks that produces a specific result (service or product) for customers.

In the reference architecture introduced in Chapter 2, the DevOps business process involves taking capabilities from the idea (typically identified with business owners) through development and testing to production.

Although this business process isn't mature enough to be captured in a set of simple process flows, you should capture the process flows that your organization already uses to deliver capabilities. Then you can identify areas of improvement, both by improving the processes themselves and by introducing automation (see the "Technology in DevOps" section, later in this chapter).

Change management process

Change management is a set of activities designed to control, manage, and track change by identifying the work products that are likely to change and the processes used to implement that change. The change management process that an organization uses is an inherent part of the broader DevOps process flow. Change management drives the way the DevOps processes absorb and react to change requests and customer feedback.

Organizations that have adopted application life cycle management (ALM) already have well-defined and (probably) automated change management processes in place.

Change management should include processes that enable the following capabilities:

- Work-item management
- Configurable work-item workflows
- Project configuration management
- Planning (agile and iterative)
- Role-based artifact access control

Traditional change management approaches tend to be limited to change request or defect management, with limited capability to trace the events between the change requests or defects and the associated code or requirements. These approaches don't provide integrated work-item management

across the life cycle or built-in capability to trace all types of assets. DevOps, however, requires all stakeholders to be able to view and collaborate on all changes across the software development life cycle.

DevOps- or ALM-centric change management includes processes that provide work-item management for all projects, tasks, and associated assets — not just those affected by change requests or defects. It also includes processes that enable the enterprise to link work items to all artifacts, project assets, and other work items that are created, modified, referenced, or deleted by any practitioner who works on them. These processes give team members role-based access to all change-related information and also support iterative and agile project development efforts.

DevOps techniques

Following are a few specific techniques that you need to include when you adopt DevOps:

- Continuous improvement
- Release planning
- Continuous integration
- Continuous delivery
- Continuous testing
- Continuous monitoring and feedback

The following sections examine these techniques in detail.

Continuous improvement

In true lean-thinking fashion, process adoption isn't a one-time action; it's an ongoing process. An organization should have built-in processes that identify areas for improvement as the organization matures and learns from the processes it has adopted. Many businesses have process improvement teams that work on improving processes based on observations and lessons learned; others allow the teams that adopt the processes to self-assess and determine their own process-improvement paths. Regardless of the method used, the goal is to enable continuous improvement.

Release planning

Release planning is a critical business function, driven by business needs to offer capabilities to customers and the timelines of these needs. Therefore, businesses require well-defined release planning and management processes that drive release roadmaps, project plans, and delivery schedules, as well as end-to-end traceability across these processes.

Most companies today accomplish this task by using spreadsheets and holding meetings (often, long ones) with all stakeholders across the business to track all business needs applications under development, their development status, and release plans. Well-defined processes and automation, however, eliminate the need for those spreadsheets and meetings, and enable streamlined and — more importantly — predictable releases. Leveraging lean and agile practices also results in smaller, more frequent releases, permitting enhanced focus on quality.

Continuous integration

Continuous integration (described in Chapter 2) adds tremendous value in DevOps by allowing large teams of developers, working on cross-technology components in multiple locations, to deliver software in an agile manner. It also ensures that each team's work is continuously integrated with that of other development teams and then validated. Continuous integration thereby reduces risk and identifies issues earlier in the software development life cycle.

Continuous delivery

Continuous integration naturally leads to the practice of continuous delivery: the process of automating the deployment of the software to the testing, system testing, staging, and production environments. Although some organizations stop short of production, those that adopt DevOps generally use the same automated process in all environments to improve efficiency and reduce the risk introduced by inconsistent processes.

In test environments, automating configuration, refreshing test data, and then deploying the software to the test environment followed by the execution of automated tests speeds feedback cycles of test results back to development.

Adopting continuous delivery typically is the most critical part of adopting DevOps. To many DevOps practitioners, DevOps is limited to continuous delivery, so most tools promoted as DevOps tools address only this process. As you see throughout this book, however, DevOps is much broader in scope. Continuous delivery is an essential component of DevOps but not the only component.

Based on your organization's business needs and pressing challenges, you may choose to start adoption with another of the processes or adoption paths described in Chapter 2.

Continuous testing

We introduced continuous testing in Chapter 2. From a process perspective, you need to adopt processes in three areas to enable continuous testing:

- Test environment provisioning and configuration
- Test data management
- Test integration, function, performance, and security

In an organization, QA teams need to determine what processes to adopt for each area. The processes that they adopt may vary from project to project, based on individual testing needs and on the requirements of service level agreements. Customer-facing applications may need more security testing than internal applications do, for example. Test environment provisioning and test data management are more important challenges for projects that use agile methodologies and practice continuous integration than they are for projects that use waterfall methodology and test only once every few months. Likewise, function and performance test requirements for complex applications with components that have different delivery cycles are different from those for simple, monolithic web apps.

Continuous monitoring and feedback

Customer feedback comes in different forms, such as tickets opened by customers, formal change requests, informal complaints, and ratings in app stores. Especially due to the popularity of social media and app stores (see Chapter 5), businesses need well-defined processes to absorb the feedback from myriad sources and incorporate them into software delivery plans. These processes also need to be agile enough to adapt to market and regulatory changes.

Measuring process adoption

You can measure the success of process adoption by seeing whether a set of efficiency and quality metrics is improving over time. This type of measurement has two prerequisites:

- You must identify the right set of efficiency and quality metrics. These metrics should really matter to the business.
- You need to establish a baseline against which to measure improvement.

You can use any of several well-defined frameworks to measure process maturity. For DevOps-specific processes, models such as the new IBM DevOps Maturity Model can assess maturity. More information about the IBM maturity model is available at `ibm.biz/adoptingdevops`.

Feedback also comes from monitoring data. This data comes from the servers running the application; from Development, QA, and Production; or from metrics tools embedded in the application that capture user actions.

Data overload is possible, so businesses need data-capture and data-use processes that enhance their applications and the environments they run in.

Technology in DevOps

Technology enables people to focus on high-value creative work while delegating routine tasks to automation. Technology also allows teams of practitioners to leverage and scale their time and abilities.

If an organization is building or maintaining multiple applications, everything it does has to be repeatable, in a reliable manner, to ensure quality across all applications. It can't start from scratch with each new release or bug fix for every application. The organization has to reuse assets, code, and practices to be cost-effective and efficient.

Standardizing automation also makes people more effective (see "People in DevOps," earlier in this chapter). Organizations may experience turnover in employees, contractors, or resource providers; people may move from project to project. But a common set of tools allows practitioners to work anywhere, and new team members need to learn only one set of tools — a process that's efficient, cost-effective, repeatable, and scalable.

Infrastructure as code

Infrastructure as code is a core capability of DevOps that allows organizations to manage the scale and the speed with which environments need to be provisioned and configured to enable continuous delivery.

Evolving around the notion of infrastructure as code is the notion of *software-defined environments.* Whereas infrastructure as code deals with capturing node definitions and configurations as code, software-defined environments use technologies that define entire systems made up of multiple nodes — not just their configurations, but also their definitions, topologies, roles, relationships, workloads and workload policies, and behavior.

Three kinds of automation tools are available for managing infrastructure as code:

- **Application- or middleware-centric tools:** These tools usually are capable of managing as code both application servers and the applications that run on them. Such tools are specialized, bundled with libraries of typical automation tasks for the technologies that they support. They can't perform low-level tasks such as configuring an operating-system (OS) setting, but they can fully automate server and application-level tasks.
- **Environment and deployment tools:** These tools are a new class of tools that have the capability to deploy both the infrastructure configurations and application code.
- **Generic tools:** These tools aren't specialized for any technology and can be scripted to perform several kinds of tasks, all the way from configuring an OS on a virtual or physical node to configuring firewall ports. They require much more work up front than application- or middleware-centric tools do, but they can handle a greater range of tasks.

By using an environment management and deployment tool like IBM UrbanCode Deploy with Patterns, organizations can design, deploy, and reuse environments quickly and help accelerate the delivery pipeline.

Delivery pipeline

A *delivery pipeline* consists of the stages an application goes through from development through to production. Figure 3-1 shows a typical set of stages. These stages may vary from one organization to another, however, and may also vary from one application to another based on the organization's needs, software delivery process, and maturity. The level of automation may also vary. Some organizations fully automate their delivery pipelines; others put their software through manual checks and gates due to regulatory or company requirements. You don't have to address all stages at once. Start by focusing on the critical parts of organization — not everything all at once — and then gradually broaden to include all stages.

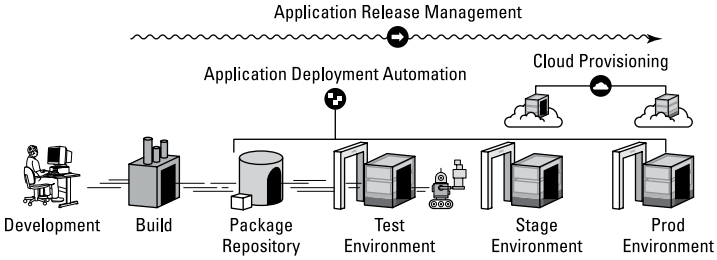


Figure 3-1: Stages of a typical DevOps delivery pipeline.

A typical delivery pipeline has the stages described in the following sections.

Development environment

An application's development effort takes place in a *development environment,* which provides multiple tools that enable the developers to write and test code. Beyond the integrated development environment (IDE) tools that developers use to write code, this stage includes tools that enable collaborative development, such as tools for source control management, work-item management, collaboration, unit testing, and project

planning. Tools in this stage typically are cross-platform and cross-technology, based on the type of development being undertaken.

Build stage

The *build stage* is where the code is compiled to create and unit test the binaries to be deployed. Multiple build tools may be used in this stage, based on cross-platform and cross-technology needs. Development organizations typically use build servers to facilitate the large number of builds required on an ongoing basis to enable continuous integration.

Package repository

A *package repository* (also referred to as an *asset repository* or *artifact repository*) is a common storage mechanism for the binaries created during the build stage. These repositories also need to store the assets associated with the binaries to facilitate their deployment, such as configuration files, infrastructure-as-code files, and deployment scripts.

Test environment

A *test environment* is where the QA, user acceptance, and development/testing teams do the actual testing. Many flavors of tools are used in this stage, based on QA needs. Here are a few examples:

- **Test environment management:** These tools facilitate provisioning and configuring the test environments. They include infrastructure-as-code technologies and (if the environment is in the cloud) cloud provisioning and management tools.
- **Test data management:** For any organization that wants to enable continuous testing, managing test data is an essential function. The number of tests that can be run and the frequency with which they're run are limited by the amount of data that's available for testing and the speed at which that data can be refreshed.
- **Test integration, function, performance, and security:** Automated tools are available for each of these types of tests. These tools should be integrated with a common test asset management tool or repository where all test scenarios, test scripts, and associated results can be stored and traceability established back to code, requirements, and defects.

- **Service virtualization:** Modern applications aren't simple, monolithic applications. They're complex systems that are dependent on other applications, application servers, databases, and even third-party applications and data sources. Unfortunately, at test time, these components may be unavailable or costly. Service virtualization solutions simulate the behavior — functionality and performance — of select components within an application to enable end-to-end testing of the application as a whole. These tools create stubs (virtual components) of the applications and services that are required for the tests to run. The behavior and performance of the application can be tested as it interacts with these stubs. IBM's Rational Test Virtualization Server provides such test virtualization capabilities.

Stage and production environments

Applications are deployed in the staging and production environments. Tools used in these stages include environment management and provisioning tools. Tools for infrastructure as code also play a critical role in these stages, due to the large scale at which the environment in these stages exist. With the advent of virtualization and cloud technologies, stage and production environments can today be made up of hundreds or even thousands of servers. Monitoring tools allow organizations to monitor the deployed applications in production.

Deployment automation and release management

Managing the automation of application deployment from one stage to the next requires specialized tools, some of which we discuss in the following sections.

Deployment automation

Deployment automation tools are the core tools in the DevOps space. Such tools perform orchestrated deployments and track which version is deployed on which node at any stage of the build and delivery pipeline. They can also manage the configurations of the environments of all the stages to which the application components must be deployed.

Measuring technology adoption

Measuring return on investment tools and technology is fairly straightforward. Typically, you can measure the efficiencies created by automation. Also, automated tools allow you to enhance the scalability and reliability of tasks — something that isn't always possible with manual tools. Finally, using an integrated set of automated tools facilitates collaboration, traceability, and improved quality.

Deployment automation tools manage the software components that get deployed, the middleware components and middleware configurations that need to be updated, the database components that need to be changed, and the configuration changes to the environments to which these components are to be deployed. These tools also capture and automate the processes to carry out these deployments and configuration changes. IBM UrbanCode Deploy is such a deployment automation tool.

Release management

Orchestrating the release plans and deployments associated with each release requires coordination across the business, development, QA, and operations teams. Release management tools allow organizations to plan and execute releases, provide a single collaboration portal for all stakeholders in a release, and provide traceability for a release and its components across all stages of the build and delivery pipeline. IBM UrbanCode Release provides such release management capabilities.

Chapter 4

Looking at How Cloud Accelerates DevOps

In This Chapter

- Using cloud as an enabler for DevOps
- Understanding full-stack deployments
- Looking at different cloud service models
- Uncovering the hybrid cloud

DevOps and cloud are both catalysts and enablers for each other. As organizations adopt cloud, the value proposition of leveraging cloud for hosting a DevOps workload becomes self-evident. The flexibility, resilience, agility, and the services a cloud platform brings allow for streamlining an application delivery pipeline hosted on the cloud. Environments from development through testing and all the way to production can be provisioned and configured as needed and when needed. This process minimizes the environment-related bottlenecks in the delivery process. Organizations are also looking to leverage cloud platforms for either reducing the cost of development and test environments or to provide a modern streamlined developer experience for their practitioners. These make for an extremely compelling business case for cloud adoption with and for DevOps.

This chapter explores different models of cloud for DevOps and examines the value proposition of DevOps as a workload on cloud.

Using Cloud as an Enabler for DevOps

The main goal of DevOps is to minimize bottlenecks in the delivery pipeline, making it more efficient and lean. One of the biggest bottlenecks that organizations experience is for environment availability and configuration. It isn't uncommon for practitioners, especially developers and testers, to requisition an environment through a formal ticketing process, and this process request can take days if not weeks to fulfill.

One of the tenets of DevOps is to develop and test on a production-like environment. Adding to the bottleneck of environment availability is the challenge of the available environment not matching the production environment. This mismatch may be just as simple as differences in configuration of the environment — at the operating system (OS) or middleware level — or as drastic as a completely different OS or middleware type on the development environments from what is used in production.

The lack of availability of environments results in potentially significant wait times for practitioners. The mismatch between development and production environments can introduce significant quality issues because the developers can't verify how the application being developed will behave in the production environment, or if it can even be deployed to production through the processes used to deploy to test environments.

Cloud addresses these problems in the following ways:

- The speed of environment provisioning on cloud platforms can provide practitioner self-service to the practitioners with on-demand environment availability and access.
- The ability to dynamically provision and de-provision these environments as needed allows for better environment management and cost reduction by reducing the need for permanent, static test environments.

- The ability to leverage "pattern" technologies that allow organizations to define and version environments as software allows for the availability of provisioning environments that match the practitioners' needs — and more importantly are production-like environments.
- From an automation perspective, the availability of application deployment automation technologies such as IBM UrbanCode Deploy can with one tool provision the cloud environment and deploy the right versions of applications to these environments as needed and when needed. They can also rapidly configure the environment and the application to match the needs of the practitioners.
- The availability of service virtualization technology, such as IBM Rational Test Virtualization Server, operating in conjunction with cloud environments, allows for the simulation of services that are needed for testing without having to provision real instances of the services.

Figure 4-1 shows how cloud environments work in conjunction with deployment automation and service virtualization technologies to provide end-to-end Develop/Test environments.

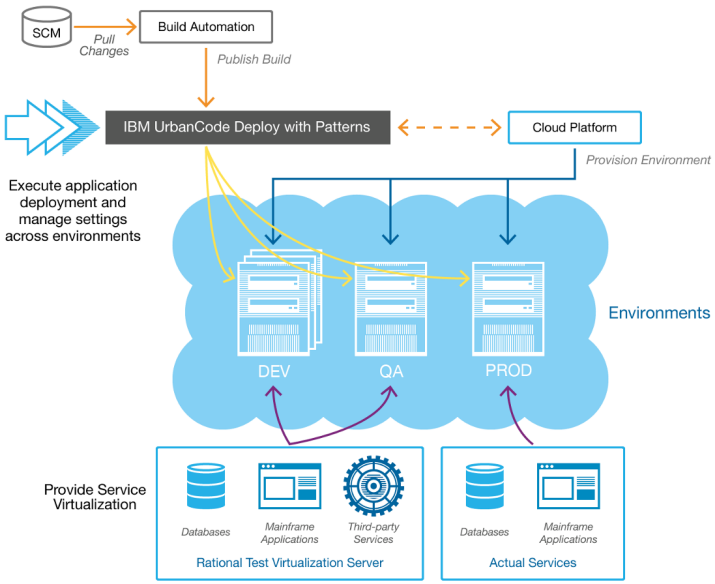


Figure 4-1: End-to-end Develop/Test in the cloud.

Cloud without DevOps means not leveraging all the benefits of cloud. Adopting DevOps with environments hosted in the cloud enable the capabilities that provide the full benefit of cloud to organizations delivering software applications.

Full-Stack Deployments

Deploying a cloud application consists of deploying the application and configuring the cloud environment on which it runs. These two tasks can be performed separately, but when they're combined, this is known as a *full-stack deployment.* We discuss these two approaches in more detail in this section.

The first approach is to separate the cloud environment provisioning from application deployment. In this scenario, there is no single point of orchestration of cloud environments and the applications that are deployed on them. The application deployment automation tool simply sees cloud environments as static environments. This scenario doesn't maximize the benefits of deploying to cloud.

The second approach is to leverage the deployment automation tool as the single orchestration tool for cloud environment provisioning and application deployment to the environments provisioned. You can achieve this by creating "blueprints" that capture the cloud environment definition and topology and then map the application components and configurations to the nodes defined in the cloud environment.

Multiple pattern technologies such as the IBM Virtual System Patterns and OpenStack HOT templates can be used to define the cloud environments as templates. Deployment automation tools such as IBM UrbanCode Deploy with Patterns can deliver full-stack provisioning using these blueprints. This includes provisioning the cloud environment defined in the blueprint and deploying the application to the provisioned environment. After the environment is provisioned, further application, configuration, and content changes can be continuously deployed to the cloud environment as updates.

Alternatively, organizations can choose to always have full stack deployment where environments and the associated applications are always provisioned together as one

deployable asset. In this case, no updates are made to the existing environment.

Choosing a Cloud Service Model for DevOps

When adopting cloud, you first want to decide on the scope of responsibility that you plan to hand over to the cloud platform and what responsibility you want to take on yourself. There are two primary service models for cloud: Infrastructure as a Service (IaaS) and Platform as a Service (PaaS).

IaaS

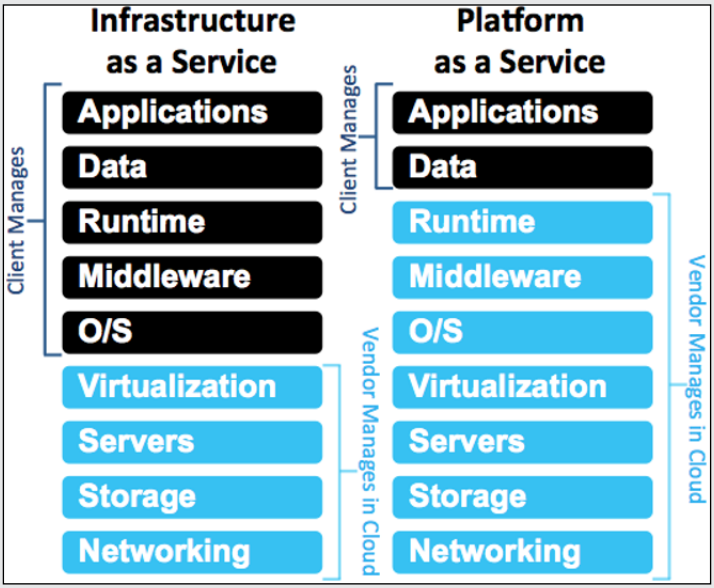
When adopting cloud under an IaaS service model, the cloud platform manages the underlying infrastructure and provides you with capabilities and services that allow you to manage all the virtualized infrastructure. The installation, patching, and management of the OS, middleware, data, and application remain the responsibility of the user.

In the context of adopting DevOps as a workload on cloud, the decision of which cloud service model to use determines how DevOps is adopted. For an IaaS service model, the user organization is responsible for managing the entire delivery pipeline. All the tools and integrations of the delivery pipeline become the responsibility of the user organization, including acquiring the right toolset and ensuring they're integrated to form the delivery pipeline. In addition, they need to ensure that the collaboration between the development and the operations teams follows a DevOps culture. Just because a cloud platform is utilized doesn't change the need to remove the silos of responsibilities between the developers delivering the code and the operations teams delivering the infrastructure, now as a cloud-based service.

While the cloud adds tremendous value in terms of providing IaaS to application delivery teams, they still need to have all the right DevOps capabilities in place to deliver the desired value that DevOps brings.

Separation of duties

One of the key questions when leveraging DevOps on IaaS cloud is to define the separation of duties between the cloud platform and the application deployment tool. Which tool is responsible for what? An easy way to look at it is from a perspective of slow and fast moving assets in the cloud stack. The sidebar figure shows the different layers of an application stack from the OS, storage, and network layer all the way up to the application.



The application, data, and middleware configuration layers are fast moving in nature. These change often because the application, its data, and its usage iterate. This velocity of change can be very high for an application still under development. The lower layers under this include the middleware (application server, database, and so on), the OS, and storage, and they don't change as often. Because updating and re-provisioning all the layers for a simple change that just impacts the application, its content, or configuration isn't efficient, it makes sense to separate the duties of these fast versus slow moving layers between an application deployment and cloud management tool. The fast moving layers are managed and automated by the application deployment tool and the slow moving layers by the cloud management software provided by the cloud platform.

PaaS

When adopting a PaaS cloud model, your only responsibility as the user becomes the application and data. All other capabilities are provided by the cloud platform as services. The result is a significantly enhanced practitioner experience for the application delivery teams. The application development and testing tools are now available as services on the platform that can be accessed by the practitioners. The application delivery organization is no longer responsible for managing the delivery pipeline. Instead, it's embedded in the PaaS and allows for practitioners to focus exclusively on rapidly delivering applications. The development and test tools and the infrastructure provisioning are all abstracted from the practitioners as services, which allows the practitioners to focus on their core duties of delivering applications.

IBM Bluemix is a PaaS. IBM and its partners manage the platform and the services provided on it. The platform embeds IBM DevOps Services — a set of services providing all the capabilities for teams to adopt DevOps and more specifically an application delivery pipeline as a set of services. Application delivery teams can use the services without any concern about how the services are hosted and delivered to them. The DevOps services include the following:

- Web-based Integrated Development Environment (IDE) as a service
- Build as a service
- Planning and task management as a service
- Security scanning as a service
- Deploy as a service
- Monitoring and analytics as a service

The platform also provides scalable runtime environments for applications running in different environments in the delivery life cycle — from development, testing, and staging all the way to production.

Understanding What a Hybrid Cloud Is

Hybrid cloud has become an extremely common term in the cloud space. It's probably overused to describe multiple cloud scenarios where either multiple cloud technologies coexist or where cloud and physical infrastructure coexists. A simple way to define hybrid cloud is to start by looking at these myriad cloud scenarios:

- **Cloud and physical infrastructure:** This is an extremely common hybrid cloud scenario. Unless an organization is born on the cloud, this is actually the default scenario. All given organizations have workloads and applications that are currently running on their existing physical infrastructure. In many cases, some of these applications continue to run on physical infrastructure. Typical examples include mainframe applications and data heavy system of record applications that aren't going to be migrated to the cloud, due to technology or cost constraints. Even if an organization is migrating all its workloads to the cloud, the migration can't take place overnight and will have a potentially extensive period where the physical and the cloud infrastructures will coexist.
- **On-premise and off-premise cloud:** In this scenario, an organization may adopt both an off-premise cloud (public or virtual-private) for some applications and workloads and an on-premise (private) cloud for others. An example would be an organization that's leveraging low-cost off-premise cloud for development environments and an on-premise self-managed cloud in its own data center for all production workloads.
- **IaaS and PaaS:** This scenario includes customers that have adopted a PaaS cloud service model for some workloads — new innovative systems of engagement type applications, for example — and IaaS for more traditional system of record workloads.

For DevOps adoption, the existence of hybrid cloud introduces new challenges because it results in application delivery pipelines that span across complex hybrid cloud and

physical environments. Examples of these hybrid cloud environments include the following:

- An organization may choose to use a public cloud for the development, testing, and other non-production environments, while using an on-premise cloud or even physical infrastructure for production.
- An organization may have some system of engagement applications deployed to a cloud environment, while the systems of record applications that provide back-end services for the core business applications may still reside on physical infrastructure, such as a mainframe.
- Organizations may leverage a public PaaS for experimentation with innovative applications and want to bring them to a private cloud, once an experiment succeeds.
- Organizations may want to have portability of application workloads across multiple cloud platforms in order to ensure vendor lock-in doesn't exist or to provide the ability to deploy critical workloads across multiple cloud vendors.

The core requirement for adopting DevOps with a hybrid cloud approach is the need for application deployment across these multiple cloud and physical environments. Applications like IBM's UrbanCode Deploy with Patterns utilize application blueprints to map applications and configurations to multiple environments, physical and cloud, allowing for automated application deployment across complex, hybrid cloud environments.

Chapter 5

Using DevOps to Solve New Challenges

In This Chapter

- Enabling mobile applications
- Dealing with ALM processes
- Scaling agile
- Managing multiple-tier applications
- Looking at DevOps in the enterprise
- Working with supply chains
- Navigating the Internet of Things

DevOps originated in so-called *born on the web* companies (companies that originated on the Internet) such as Etsy, Flickr, and Netflix. These companies, while solving complex technology challenges at a very large scale, had fairly simple architectures — unlike large enterprises that grew around legacy systems and/or through acquisitions and mergers, with complex multi-technology systems that had to work together. These challenges are further aggravated by the demands being put on modern enterprises by new technologies like mobile and application delivery models such as software supply chains.

This chapter explores some of these challenges that enterprises face and that DevOps can help solve.

Mobile Applications

In an enterprise, mobile apps are typically not stand-alone apps. They have very little business logic on the mobile device itself and serve more as front-ends to a multiple enterprise applications already in use by the enterprise. These back-end enterprise applications may range from transaction processing systems to employee portals to customer acquisition systems. Mobile development and delivery is complex and requires a set of dependent services to be delivered in a coordinated fashion in a reliable and efficient manner.

For enterprise mobile apps, release cycles and new feature releases need to be coordinated with those of the enterprise applications and services that the mobile apps interact with. Therefore, DevOps adoption should include mobile-app teams as first-class citizens and participants along with the rest of the enterprise software development teams.

DevOps and app stores

One unique aspect of mobile apps is the need for deployment to app stores. Most mobile apps can't be deployed directly to mobile devices; they have to go through a vendor-managed app store. Apple introduced this distribution format with its App Store (and locked its devices to prevent direct installation of apps by app developers or vendors). Device manufacturers such as Research In Motion, Google, and Microsoft, which once allowed direct app installation, now follow the Apple model.

This situation adds an asynchronous step to the deployment process. Developers can no longer deploy updates to an app on demand. Even for critical bug fixes, new app versions have to go through an app store's submission and review processes. Continuous delivery becomes submitting and waiting. Continuous deployment to development and testing remains available, however, with the test environment being simulators for the devices on which the application will be deployed or banks of physical devices.

Eighty percent of the world's corporate data originates on the mainframe, and 70 percent of all transactions touch a mainframe. Unlocking a mobile path to these mainframe capabilities can transform the way you conduct business and engage with customers, but getting there can be challenging. You may be confronted with skill gaps, organizational silos, and multiple platforms that result in long release cycles, unnecessary delays, and wasted resources. To provide mobile access to enterprise applications, businesses are embracing DevOps, a software delivery approach that focuses on speed and efficiency without sacrificing stability and quality.

No specific DevOps concepts or principles apply solely to mobile apps. Mobile apps, however, add to the need for DevOps due to their inherent short development life cycles and rapid change.

ALM Processes

Application life cycle management (ALM) is a set of processes employed to manage the life of an application as it evolves from an idea (a business need) to an application that's deployed and eventually under maintenance. Hence, looking at DevOps as an end-to-end business capability makes ALM the fundamental concept underlying the DevOps process. DevOps broadens the scope of ALM to include business owners, customers, and operations as part of the process.

The DevOps Develop/Test adoption path (see Chapter 2 for more info) most closely aligns with the traditional ALM capabilities of requirements management, change management, version control, traceability, and test management. However, other ALM capabilities such as tracking and planning occur as part of the Steer adoption path, and dashboards and reporting are included in the Operate adoption path.

Scaling Agile

Lean and agile development are the underpinnings of the DevOps approach — waste reduction from more efficient teams is one of the results. Efficiency and repetition of best practices lead to shorter development cycles, allowing teams to be more

innovative and responsive, thereby increasing customer value. Scaling lean and agile principles beyond the development team to a team of teams and across the entire product and software delivery life cycle is core to the DevOps approach.

Many teams have already adopted agile and want to scale their current processes as part of their DevOps adoption. Many popular frameworks are available to help scale agile. These frameworks include the Scaled Agile Framework (SAFe) and Disciplined Agile Delivery (DAD). Some organizations have also been effective at scaling the Scrum process to very large teams. The purpose of these frameworks is to provide a methodology for adopting agile at the enterprise level. That means taking into consideration not just the development of code but also including architecture, project funding, and governance of the processes and roles required by management, applying the very same lean and agile principles that have worked well at the team level. No matter the framework used to scale agile, you take those basic agile principles and apply best practices to leverage them to drive efficiency and effectiveness across the enterprise.

Multiple-Tier Applications

In a typical large IT shop, it's not uncommon to find multiple-tier applications that span many platforms, each with its own unique development process, tools, and skill requirements. These multi-tier systems often integrate applications on the web, desktop, and mobile applications on the front-end and back-end systems such as packaged applications, data warehouse systems, applications running on mainframes, and midrange systems. Managing and coordinating the releases of the parts of multiple-tier systems, many of which may be on different platforms, can be overwhelming even for the most disciplined IT organization.

A sensible approach is to follow automated, consistent build, configure, and deployment processes through all stages of development. This approach ensures that you're building all the parts you need — and only the parts you need. It also ensures that the application remains whole as changes come in and the project moves through the cycle of testing, QA, and production. IBM UrbanCode Deploy has an application model that helps automate the complex deployment of multiple-tier applications.

Maintaining separate tools for different teams based on platform is a reality in today's multi-platform, multi-vendor world. This is where open platforms such as IBM Jazz can integrate disparate tools to provide a unified solution. Consistent deployment practices can help ensure that teams are using reliable, repeatable deployment across platforms to provide true business value.

DevOps in the Enterprise

Today's enterprise depends on the speed with which IT can deliver software. These businesses typically operate systems of record applications (home grown or packaged apps) deployed on mainframe and midrange systems. They face many challenges:

- Regulatory hurdles
- Process complexity
- Skills gaps
- Organizational silos
- Platforms and tools that result in long release cycles, unnecessary delays, and wasted resources

DevOps at the enterprise level enables planning, development, testing, and operations stakeholders to continuously deliver software within their organizations. Enterprises today deploy applications that are truly cross-platform — from mobile to mainframe. The DevOps approach to development uses lean principles to create an efficient and effective delivery pipeline that allows applications to be developed, tested, and delivered as it helps raise the quality, increase the speed, and reduce the costs of development.

Given the true multi-platform nature of today's enterprises, with the presence of mobile, cloud, distributed, and mainframe applications — all of which need to be created, integrated, deployed, and operated — the need for the efficiencies, streamlining, and collaboration that DevOps provides is becoming a key competitive differentiator.

Supply Chains

With the increasing use of outsourcing and strategic partnerships to supply skills and capabilities to an enterprise, software supply chains are becoming the norm. A *supply chain* is a system of organizations, people, technology, activities, information, and resources involved in moving a product or service from supplier to customer. The various suppliers in the chain may be internal or external to the enterprise.

In an organization that has adopted a supply-chain model for delivering software, adopting DevOps can be a challenge, because the relationships among suppliers are managed more by contracts and service level agreements than by collaboration and communication. Such an organization can still adopt DevOps, however. The core project teams retain ownership of the planning and measurement capabilities, with other capabilities being shared among the other suppliers. In the delivery pipeline, different suppliers may own different stages of the pipeline. Using common tool sets and a common asset repository is therefore essential. A work-item management tool, for example, provides reporting on all items being worked on by all suppliers, as well as transfer of ownership of work items across suppliers. Using a common asset repository provides a mechanism for passing assets through the pipeline, enabling continuous delivery.

The Internet of Things

The next big step for DevOps is its evolution into the systems or embedded-devices space where it's often referred to as *continuous engineering.* When the Internet started, most of the data shared on it was human-generated. Today, innumerable Internet-connected devices (such as sensors and actuators) generate much more data than humans do. This network of inter-connected devices on the Internet is commonly referred to as *the Internet of Things.*

In this space, DevOps is potentially even more essential, due to the co-dependence of the hardware and the embedded software that runs on it. DevOps principles are reflected in continuous engineering to ensure that the embedded software

delivered to the devices is high-quality software with the right engineering specifications.

"Operations" in continuous engineering is replaced by hardware or systems engineers who design and build custom hardware for the devices. Collaboration between the development and testing teams and the systems engineers is crucial to ensure that hardware and software are developed and delivered in a coordinated manner despite hardware and software development needing to follow different delivery cycles. The development and testing needs for continuous delivery and testing remain the same. Simulators are used to test software and hardware during development.

Anti-patterns

In the real world, there are always limitations to adoption of DevOps principles. Some of these limitations are functions of the industries and environments in which a business exists, such as regulatory compliance, complex hardware systems, or immature software delivery capabilities. In such cases, DevOps needs to be adopted in light of *anti-patterns* (ineffective or counterproductive patterns) that may not be acceptable for an organization, based on its business needs.

Water-SCRUM-fall

Forrester (`www.forrester.com`), a global research and advisory company, coined the term *Water-SCRUM-fall* to describe the current state of adoption of agile software development methodologies. From a DevOps perspective, this means that whereas development teams may have adopted agile practices, the teams around them may still have manual, waterfall-style processes that don't allow for continuous delivery. In several enterprises, this situation results from the corporate culture. A company that adopts DevOps must embed manual processes in broader DevOps practices.

NoOps

In a NoOps organization, Operations is eliminated as a separate department, and its responsibilities are merged into those of Development. Netflix, an Internet television provider, is a proponent of this method. NoOps may work well for some organizations, but some waiting is involved to see if this organizational model will have wider practical appeal.

Chapter 6

Making DevOps Work: IBM's Story

In This Chapter

- Understanding the best practices for executives
- Organizing your team
- Identifying DevOps goals
- Taking note of the DevOps transformation
- Learning from the DevOps results

DevOps is being adopted company-wide at IBM and continues to regularly evolve. This adoption is a result of the success of using a DevOps approach pioneered at IBM Software Group (SWG) Rational and now being used at Watson, Tivoli, Global Business Services, and other divisions. This chapter provides a case study of IBM SWG's own adoption of DevOps capabilities by the IBM Rational Collaborative Lifecycle Management product team.

This software delivery effort is unique in that it's developed in the *open* — the software delivery team delivers all its development artifacts and on-going work, including all detailed work-items, on `jazz.net`. This website is open to the public, and any registered user can look at the work planned, work ongoing, and the history of all the development work done for the software products.

Taking a Look at the Executive's Role

Culture is a hidden thread in an organization. It is based on values and behaviors that evolve from both management and employees. Many times you don't actually understand the culture of the organization until you embark on a significant change. There will be those skeptics who take a wait and see approach to determine if this is the passing fad of the month. Leaders will emerge. It is essential to establish an approach to understand these dynamics and to know who is who so you can address the real inhibitors.

To address the cultural dynamic, the IBM SWG executive used a number of approaches:

- **Select the right leader:** The leader's role is to pull together the differing viewpoints to start to bring the team to a common set of objectives, inhibitors, process changes, and decisions on where to start first.
- **Involve stakeholders:** Support for these changes has to come from the leadership, management, and individual contributors across different development disciplines. There must be business stakeholders, architects, developers, testers, and operations involved and named leaders from these areas who are champions for change.
- **Measure improvements and outcomes:** It's critical to have a set of key metrics that incorporate both the needed efficiencies and the business outcomes. These goals and measurements should set a high bar and hold people accountable, but they shouldn't cause disengagement.
- **Build momentum with early successes:** Understanding these inefficiencies and measuring the improvements in each area builds momentum for change.
- **Communicate and listen:** As a leader, it's important to understand the real dynamics of how the change is taking hold in the team. Spending time having one-on-one conversations and regular face-to-face interactions with the technical teams, management, and business leaders helps to gauge the buy-in of the team, their perspective on the inhibitors, and, equally important, an opportunity for management to share perspectives on priorities and progress.

If you're an executive, you should support the teams and make yourself available to understand and remove obstacles. Operating as a whole team with clear business goals is necessary to bringing everyone together on the same path.

Putting Together the Team

The IBM SWG Rational Collaborative Lifecycle Management product team is part of a larger group that develops a set of over 80 software development tools in the categories of software delivery planning, software development, application deployment, software quality management, and application monitoring and analytics.

This IBM SWG product team is a large, global organization with four core product teams working at more than 25 locations in 10 countries. Before adopting a DevOps approach, the group worked on a yearly release schedule including an additional three to six months of lead time to actually determine what went into that yearly release.

Setting DevOps Goals

The IBM SWG team felt they took too long to respond to shifts in the market as well as the shifts in demand from customers. The team decided to shorten the delivery cycle, not only in the development and test phases but also for the collaboration and interactions with the business stakeholders and customers. The goal was set to move from a yearly release schedule to once every quarter.

In addition to the need to accelerate its development to deliver new capabilities more frequently, the team had to move more quickly to support cloud delivery models, mobile development, mobile testing, and other capabilities to address technology shifts. The team chose to embrace DevOps principles and practices to transform the way the group develops software to deliver value to its customers earlier and more frequently.

Making a modification of this scope required a cultural change within the organization, so four workgroups were established that were made up of members of the management team and

technical leaders. These workgroups examined the software delivery processes from beginning to end and took responsibility for changing the ways of working. A specific set of measurements and action plans were established to address the key points in the development process. A continuous delivery champion team was created, including an evangelist who educated teams and shared best practices across the organization.

The IBM SWG team started its journey of adopting DevOps by identifying these goals:

- Streamline the process and introduce new methodologies
- Leverage tools for consistency, for scalability to other teams, and for traceability and metrics
- Evolve the culture to continuously improving

Learning from the DevOps Transformation

This section describes the steps taken by the IBM SWG team to facilitate their DevOps transformation.

Expanding agile practices

Existing agile practices were expanded beyond development and test to include clients, business stakeholders, and operations in order to break down silos and improve results. This broader agile model allows teams to work together to produce consistent, high-quality software that delivers value for the business by using a set of processes that is integrated at every step.

A "one team" approach was taken that combined product management, design, and development. The development team included the traditional roles of development managers and team leads but also brought in operations management and architects to support an end-to-end life cycle strategy.

Dedicated resources were provided to coach and mentor teams in agile and continuous delivery across the organization. A focus on capabilities versus product components

helped to break down traditional silo boundaries and allow for ship readiness with every sprint and automation. These feature teams were also empowered by the assignment of dedicated development managers. Regular Scrum meetings were held at all levels of the development organization to identify and solve blocking issues, track key metrics, utilize live dashboard data, and communicate critical information.

To improve the timeliness of market changes with development priorities, a strategic product committee was formed and consisted of product management, development directors, architects, and business owners. Their responsibilities included the following:

- Allocating and ensuring funding for program execution success
- Driving, assisting, and supporting program execution
- Establishing long-term vision and direction for the business
- Prioritizing epics and user stories for annual releases that align with the long-term vision

Leveraging test automation

To eliminate the traditional long back-end test cycles and improve the quality of releases, an agile continuous testing approach was adopted using automation and virtualization. A rhythm was established with four-week iterations ending with a demo and four-week milestones ending with a customer-useable release. Retrospectives after each milestone and understanding technical debt helped eliminate waste in future iterations. The IBM SWG team motto was "test early and test often."

The team adopted the following best practices for test automation:

- Automate repetitive and labor-intensive tests.
- Automate in areas where bugs are frequently found.
- Run automation on every build; run early and often.

- Create automation that's resistant to user interface (UI) changes — use a framework that separates the UI from the tests.
- Make it easy to create, deliver, and maintain the automation establishing strong feature team ownership.
- Plan automation development work into your estimates and ensure developers have time to work on it.
- Develop metrics so you can evaluate whether your automation is useful (you can't improve what you can't measure).
- Constantly re-evaluate if your automation is finding bugs and refactor it if it's not.

To support test automation, the team deployed IBM Rational Test Workbench for functional and performance testing, and to enable more frequent testing, automating the deployment of builds was critical. By using IBM UrbanCode Deploy, the team saw test deployment costs reduced by 90 percent through automated build deployment, which included automating any necessary application and database server configuration settings.

Building a delivery pipeline

The IBM SWG team decided to build a delivery pipeline that leveraged "tools-as-a-service" and enabled developers to commit code, test, and deploy to a production environment in about 60 minutes compared to one to two days prior. This process reduced the need for rework and maximized productivity.

In the team's deployment, it recognized that a continuous delivery pipeline needed to embrace the following best practices:

- Shift-left testing and automate as much as possible
- Use the same deployment mechanisms everywhere
- Strive to maintain a constant state of ship-readiness
- Treat infrastructure as code

In Figure 6-1, you see the products and the functions provided as part of the continuous delivery pipeline adopted by the IBM SWG team.

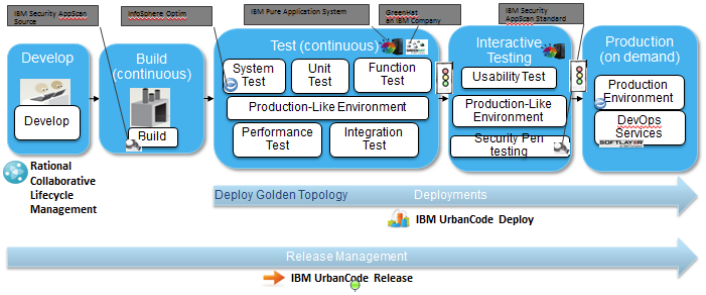


Figure 6-1: The continuous delivery pipeline.

A key best practice essential in implementing a continuous delivery pipeline, is to "treat infrastructure as code." What this means is that the developer can write scripts to configure the required infrastructure for their application as part of their application code. In the past, this was typically done by a system administrator or an operations person, but now the control and the efficiencies it provides can be accomplished by the developer directly. Puppet, Chef, and IBM UrbanCode Deploy with Patterns are examples of the new category of infrastructure automation tools that make infrastructure as code a practical reality.

The IBM SWG team now treats infrastructure as code and follows these best practices:

- Treat pattern definitions, script packages, and services as code.
- Version everything.
- Automate deployment of topology patterns to the cloud.
- Manage versions of patterns across multiple cloud environments.
- Automate the testing of patterns.
- Cleanup catalog resources to avoid sprawl.

Experimenting rapidly

The concept of continuous delivery includes not only software development activities such as continuous integration and continuous deployment but also the more fundamental activity of learning, which is best achieved through frequent experimentation and measuring the results.

When features and functions are added to an application, you never know for certain if the customer will receive the expected or intended benefits. So that's why it's important to IBM to experiment early and often, solicit feedback from customers as to what actually works for them, and discard those features that have little benefit or perhaps are even a hindrance. This strategy is depicted in the sketch in Figure 6-2.

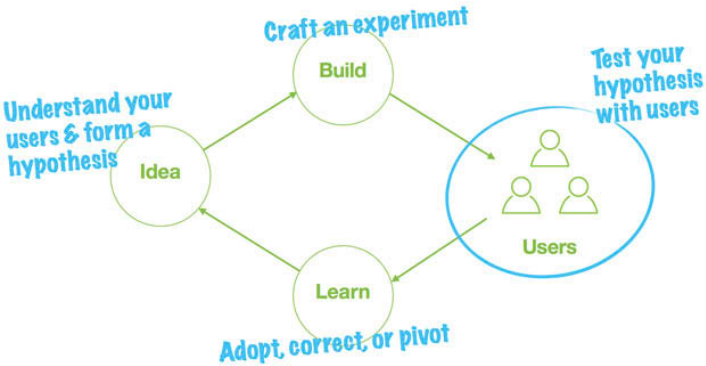


Figure 6-2: A look at hypothesis-driven development.

The IBM SWG team learned a great deal about frequent experimentation and developed the following best practices:

- Establish metrics and success/failure criteria.
- Figure out what works by running experiments — tiny tests for a small subset of users to help determine the usefulness of a feature.
- Run multiple experiments continuously.
- Make fact-based decisions quickly.
- Deliver faster and you can experiment faster.

- Establish a mechanism to enable system-wide experimenting (Google Analytics, IBM Digital Analytics, and so on).
- Consider different models of experimenting (classical A/B testing, multi-armed bandit, and so on).
- Follow two paths simultaneously for related projects: experiment on a cloud-based project and use the data from the experiments to not only drive the direction of that project but also related on-premise projects.

Continuously improving

The IBM SWG team wanted to create a culture of continuous improvement and leverage measures of effectiveness and efficiency to ensure they were actually improving. The teams manage their continuous improvement efforts like an agile project. They support continuous improvement by tracking maturity goals, pain points, and associated improvements actions to address the issues. They track continuous improvement work like other development work to ensure the investment is widely understood. Maturity goals (for example, capabilities) may take one or more quarters to actually develop and adopt. Large pain points may take many months to reduce or eliminate. But in any case, specific improvement actions should all be sized to deliver within a month.

The IBM SWG team uses retrospectives to institutionalize continuous improvement. A *retrospective* is a regular review of what went well, what didn't go so well, and what actions need to be taken to improve. If you aren't doing retrospectives, it implies a level of perfection in software development that has yet to be achieved. In a large team, you can have a hierarchy of retrospectives. For the IBM SWG team, each component team does a retrospective, and these are used as input into application-level retrospectives that are then used as input in a higher solution-level retrospective. Actions from the retrospectives are documented as pain points with corresponding improvement actions to take in order to reduce or alleviate the pain.

And to ensure teams are getting better, the IBM SWG team established both business metrics and operational metrics to measure the effectiveness of the DevOps transformation. The business metrics consist of measured improvements in

- Faster time to delivery
- Improved client satisfaction
- Reduced maintenance spending while increasing innovation investment
- Increased client adoption

Operational metrics impact team's efficiency over time and measure the following:

- Time to initiate a new project
- Build time
- Iteration test time

Looking at the DevOps Results

A DevOps approach has helped the IBM SWG team realize gains in improved customer satisfaction, increased customer adoption, and a double-digit revenue growth. Shorter timeframes have energized delivery teams within IBM, resulting in rapid delivery of upgraded on-premise solutions and new cloud services such as Bluemix, DevOp Services for Bluemix, and Collaborative Lifecycle Management as a Managed Service (CLM aaMS).

As a specific example of the success of a DevOps approach at IBM, Figure 6-3 shows the measured results achieved by the IBM SWG Rational Collaborative Lifecycle Management product team.

Lifecycle Measurements	2008	2010	2012 – 2014	Total Improvement
Project Initiation	30 days	10 days	2 days	**28 days**
Groomed Backlog	90 days	45 days	On-going	**89 days**
Overall Time To Development	120 days	55 days	3 days	**117 days**
Composite Build Time	36 hours	12 hours	5 hours	**700 %**
BVT Availability	N / A	18 hours	< 1hour	**17 hours**
Iteration Test Time	5 days	2 days	14 hours	**4 days**
Total Deployment Time	2 days	8 hours	4 hours -> 20 minutes	**2 days**
Overall Time To Production	9 days	3 days	2 days	**7 days**
Time Between Releases	12 Months	12 Months	3 Months	**9 Months**

Figure 6-3: IBM SWG team measured improvements.

Chapter 7

Ten DevOps Myths

In This Chapter

- Understanding what DevOps is for
- Knowing what DevOps isn't for

The DevOps movement is young and still emerging, especially among enterprises. Like any new movement or trend, it has attracted myths and fallacies. Some of these myths may have originated in companies or projects that tried and failed to adopt DevOps. What's true in one situation, however, may not necessarily be true in others. Here are some common myths about DevOps — and the facts.

DevOps Is Only for "Born on the Web" Shops

What is generally referred to as DevOps originated in "born on the web" companies (companies that originated on the Internet) such as Etsy, Netflix, and Flickr. Large enterprises have been using DevOps-aligned principles and practices to deliver software for decades, however. Furthermore, current DevOps principles, as described in this book, have a level of maturity that makes them applicable to large enterprises that have multiple-platform technologies and distributed teams.

DevOps Is Operations Learning How to Code

Operations teams have always written scripts to manage environments and repetitive tasks, but with the evolution of infrastructure as code, operations teams saw a need to manage these large amounts of code with software engineering practices such as versioning code, check-in, check-out, branching, and merging. Today, a new version of an environment is created by an operations team member by creating a new version of the code that defines it. This doesn't mean, however, that operations teams need to learn how to code in Java or C#. Most infrastructure-as-code technologies use languages like Ruby, which are relatively easy to pick up, especially for people who have scripting experience.

DevOps Is Just for Development and Operations

Although the name suggests a development-plus-operations origin, DevOps is for the whole team. All stakeholders in the delivery of software — lines of business, practitioners, executives, partners, suppliers, and so on — also have a stake in DevOps.

DevOps Isn't for ITIL Shops

Some people fear that DevOps capabilities such as continuous delivery are incompatible with the checks and processes prescribed by the Information Technology Infrastructure Library (ITIL), a set of documented best practices for IT service management. In reality, ITIL's life-cycle model is compatible with DevOps. Most of the principles defined by ITIL align very well with DevOps principles. ITIL has, however, received a bad reputation in some organizations due to being implemented predominately with slow, waterfall processes that didn't allow for rapid changes and improvement. Aligning such practices between development and operations is the essence of DevOps.

DevOps Isn't for Regulated Industries

Regulated industries have an overarching need for checks and balances and for approvals from stakeholders who ensure compliance and auditability. Adopting DevOps actually improves compliance, if it's done properly. Automating process flows and using tools that have built-in capability to capture audit trails can help.

Organizations in regulated industries will always have manual checkpoints or gates, but these elements aren't incompatible with DevOps.

DevOps Isn't for Outsourced Development

Outsourced teams should be viewed as suppliers or capability providers in the DevOps delivery pipeline. Organizations should ensure, however, that the practices and processes of the supplier teams are compatible with those of their internal project teams.

Using common release planning, work-item management, and asset repository tools significantly improves communication and collaboration between lines of business and supplier and project teams, enabling DevOps practices. Using application release management tools can greatly improve an organization's ability to define and coordinate the entire release process across all participants.

No Cloud Means No DevOps

When you think of DevOps, you often think of cloud because of its ability to dynamically provision infrastructure resources for developers and testers to rapidly obtain test environments without waiting days/weeks for a manual request to be fulfilled. However, cloud isn't necessary to adopt DevOps

practices as long as an organization has efficient processes for obtaining resources for deploying and testing application changes.

Virtualization itself is optional. Continuous delivery to physical servers is possible if the servers can be configured and deployed to at the necessary speed.

DevOps Isn't for Large, Complex Systems

Complex systems require the discipline and collaboration that DevOps provides. Such systems typically have multiple software and/or hardware components, each of which has its own delivery cycles and timelines. DevOps facilitates coordination of these delivery cycles and system-level release planning.

DevOps Is Only about Communication

Some members of the DevOps community have coined humorous terms such as *ChatOps* (teams carry out all communications through communication tools like Internet Relay Chat) and *HugOps* (DevOps focuses only on collaboration and communication). These terms stem from the misconception that communication and collaboration solve all problems.

DevOps depends on communication, but better communication coupled with wasteful processes doesn't lead to better deployments.

DevOps Means Continuous Change Deployment

This misconception comes from organizations that deploy only web applications. Some of these companies proudly state on their websites that they deploy to production daily. Daily

deployment, however, is not only impractical in large organizations that deploy complex applications, but may also be impossible due to regulatory or company restrictions. DevOps isn't just about deployment, and it's certainly not just about deploying continuously to production. Adopting DevOps allows organizations to release to production when they want to and not based on a particular date marked on a calendar.

Notes

Notes

Notes

Notes

The world runs on software

Today's fast-moving world makes DevOps the essential approach for any business aspiring to be lean and agile. In order to respond rapidly to changing customer and marketplace demands, DevOps helps you achieve continuous delivery of software-driven innovation. This book helps you understand DevOps and how your organization can gain real business benefits from it. You also discover how a holistic view of DevOps that encompasses the entire software delivery life cycle — from ideation and the conception of new business capabilities to implementation in production — can bring competitive advantage in a continuous delivery world.

- ***Exceed customer expectations* — *deliver higher-quality experiences***
- ***Meet the needs of the business* — *respond to dramatic increases in software delivery frequency and volume***
- ***Leverage new technology trends* — *use mobile, cloud, big data, and social***
- ***Establish better collaboration* — *engage stakeholders across the software delivery life cycle***

Open the book and find:

- **DevOps capabilities**
- **How to work with mobile, cloud, and other leading technologies**
- **How DevOps aligns with application life cycle management (ALM) processes**
- **How to manage multi-tier applications**
- **IBM's story of using DevOps**

Go to Dummies.com®
for videos, step-by-step examples, how-to articles, or to shop!

FOR DUMMIES®
A Wiley Brand

ISBN 978-1-119-04705-6
Part #: RAM14026-USEN-00
Not for resale

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.