Compliments of IBM

2nd IBM Limited Edition

Application Release & Deployment

FOR DUMMIES®

A Wiley Brand

Learn to:

- Assess and improve your current software delivery methods
- Adopt deployment standards for rapid delivery of quality applications
- Successfully implement application release and deployment solutions

Eric Minick
Allan Wagner
Claudia Ring

Application Release & Deployment

FOR DUMMIES®

A Wiley Brand

2nd IBM Limited Edition

by Eric Minick, Allan Wagner, Claudia Ring

FOR DUMMIES®

A Wiley Brand

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

ISBN: 978-1-119-27073-7 (pbk); ISBN: 978-1-119-27074-4 (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
Editorial Manager: Rev Mengle
Acquisitions Editor: Steve Hayes
Business Development Representative: Sue Blessing
Production Editor: Antony Sami

Table of Contents

Introduction

Software applications are large drivers of business revenue, and the timely release and deployment of those applications has become a critical part of the business life cycle. After all, what good is it to create an innovative application if you can't deploy it to test environments efficiently or deliver it to users on schedule?

Traditionally, *deployment automation* is defined as applying a set of application changes to an environment. *Release management* encompasses the governance and coordination of delivering these changes through environments seeking to ensure safe delivery into production. Release management and deployment automation differ but have the same objective: to continuously deliver incremental change as quality applications to the end-user.

Because release management and deployment automation have similar goals, the terms are often used interchangeably. For purposes of this book, we refer to release management and deployment automation as defined in the preceding paragraph and distinguish them as required throughout the remainder of this book.

Throughout the book, we give you insight into the differences between release management and deployment automation. We also show you how to leverage release management and deployment automation solutions to help your organization speed time to market, drive down cost, and reduce risk.

About This Book

We wrote this book to serve as a relatively simple introduction to what can be a very complex topic. Use it as a reference, not as a manual. If you're interested in certain topics but not in others, feel free to read only certain chapters. Also feel free to skip around. You don't have to read the chapters in order.

Icons Used in This Book

You'll find the following icons in the margins of this book:

The Tip icon points out helpful information.

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

You don't have to read Technical Stuff material unless you want deeper understanding of a topic.

Be sure to read anything marked with a Warning icon, which alerts you to risk.

Beyond the Book

Throughout this book, we talk about the benefits of application deployment automation, and release management solutions. We include a few industry success stories as well as best practices. As you begin your journey to grow deployment automation and release management skills, we suggest that you take advantage of the many assets available on IBM developerWorks in the UrbanCode Developer Center website: `http://developer.ibm.com/urbancode`. Watch some of the many videos or read the tech tips, product documentation, and how-to guides to grow skills quickly.

You can also find more information on application release and deployment by visiting the following web pages:

- **IBM DevOps:** `http://ibm.com/devops`
- **IBM UrbanCode Deploy product page:** `http://ibm.co/UCDeploy`
- **IBM UrbanCode Release product page:** `http://ibm.co/UCRelease`

- **7 Proven Practices to Strengthen Release Management:** `http://ibm.co/7ProvenPractices`
- **The ABCs of Continuous Release and Deploy in a DevOps Approach:** `http://ibm.co/ABCsRaD`

Finally, you can read other *For Dummies* titles from the IBM Limited Edition eBook Series. Visit `http://ibm.biz/devopsdummiesbooks` for titles such as

- *DevOps For Dummies,* IBM Limited Edition
- *Agile For Dummies,* IBM Limited Edition
- *Mobile to Mainframe DevOps For Dummies,* IBM Limited Edition
- *Service Virtualization For Dummies,* IBM Limited Edition
- *Hybrid Cloud For Dummies,* IBM Limited Edition

Chapter 1

What Drives Effective Release and Deployment?

In This Chapter

- Looking at the DevOps movement
- Understanding the business benefits of effective software delivery
- Understanding the technical practices that accelerate application delivery
- Discovering the cultural implications of automating software deployments

In today's competitive world, being first to market can be the difference between success and failure. For companies focused on developing software to achieve their business objectives, many are exploring the DevOps movement as a way to achieve more reliable and repeatable deployment automation and release management capabilities as they work to become more agile and eliminate waste. Being able to deliver software faster and at a lower cost is a competitive advantage as companies look for ways to increase revenue, disrupt the market or deliver that next innovative software solution.

Continuous Delivery and Continuous Feedback

What does Continuous Delivery and Continuous Feedback mean to the business? Being able to quickly and efficiently release increments of software innovation to the market shortens the time in understanding how end-users view the solution, use the solution, and what enhancements they

would like next. What a company does with this new information and how it analyzes the feedback is incredibility important. Delivering the minimal viable product and capturing feedback for a release will expose whether you are on the right track or perhaps hit the "bull's eye." This feedback also indicates if there's room for improvement. The faster you understand the sentiment of end-users, the faster you can react and deliver the functionality they crave.

Unfortunately, the effort to deliver applications is often plagued with numerous bottlenecks that manifest themselves as delays in the release cycle due to a variety of reasons:

- Manual deployments and hand-offs
- Lack of environments for testing
- Inconsistency between environments
- Inefficient processes
- Reliance on the "Super-Hero" to save the day

Automating the deployment of applications and improving how teams execute against the release plan can reduce the delays associated with manual handoffs. Treating infrastructure as code, including leveraging cloud technologies to stand up (or tear down) environments on demand, provides organizations the flexibility, scalability, and reliability they need to succeed. Deploying with consistency across the entire delivery pipeline can help organizations improve processes and reduce relying on the "Super-Hero" to solve production issues.

Adopting a solid automation deployment solution coupled with continuous testing practices enables the delivery team to not only measure the quality of the software to be released, but also validate the means by which that software is being deployed.

Continuous Delivery is a term used to describe an approach to software development where the delivery team incrementally produces new software functionality that can be released at any time. This includes building, deploying, and testing the application to be released with greater efficiency, accuracy, and consistency.

In a DevOps world, Continuous Delivery is complimented by the related principal of Continuous Feedback. *Continuous Feedback* closes the learning loop by quickly getting information about

customer experience back into the hands of business stakeholders and the delivery team. This principle helps to improve decision making, leading to predictability with greater accuracy. It is hard to achieve business objectives when you don't consider how the end-user reacts to your offering. Success is not only measured in developing software better, cheaper, and faster, but also it must delight the end-users who use it.

Continuous feedback isn't limited to an exchange between the business and the end-user. In a DevOps world of experimentation and learning, continuous feedback across the delivery pipeline is critical for more efficient execution during each phase of the software development life cycle. The continuous delivery and feedback cycles are shown in Figure 1-1.

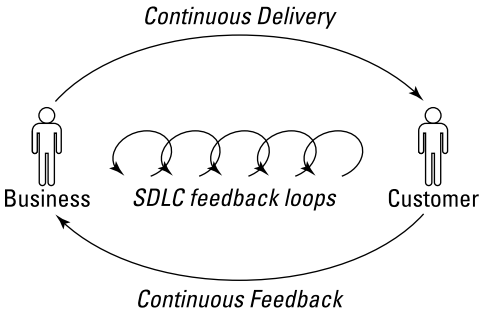


Figure 1-1: The Continuous Delivery and Continuous Feedback loop.

Continuous Integration and the Continuous Delivery Pipeline

The adoption of Agile and the need to become leaner, more efficient, in the way that software is developed drove a need for continuous integration. A practice where a software change commits to the trunk, creating a new version of that software, triggers the execution of a build, the execution of unit tests (we hope) and the creation of a deployable build package. This process improvement through automation, while helping the programmers become more efficient, exposed new bottlenecks further out in the delivery pipeline. The first of which was delays in deploying the build package for the purposes of testing.

Continuous Integration is facilitated by the use of a solution that automatically builds newly integrated changes when they're checked in. Often, automatic verification testing accompanies the automatic build so that defects can be found and addressed faster. This feature is especially powerful when combined with Continuous Delivery, discussed in the next section.

Deploying what you have, virtualizing what you don't

Eliminating the testing bottleneck can greatly improve one's chances of shortening the delivery cycle and help organizations get their innovative software products into the hands of the end-user much faster than they could in the past.

Unfortunately, for many organizations, Testers could spend up to 50 percent (or more) of their time standing up test environments which may or may not be an accurate representation of the production environment — the environment where this software will eventually reside.

Organizations need to be able to start testing their software end to end much sooner to validate quality and get feedback to the programming team so that the delivery team, programmers, and testers are working in sync.

A deployment automation solution can go a long way to eliminating this bottleneck and enabling teams to begin testing earlier or shifting testing to the left. Using infrastructure as code and blueprinting to stand up (or tear down) environments on demand can quickly provide a production like test environment. The time sink testers faced is eliminated. External systems, or parts of the system that aren't yet ready can be modeled using a service virtualization solution.

Using a deployment automation solution which offers integration with test data management functionality, service virtualization technologies and test automation can shorten the test cycle giving feedback on the quality of the latest build at speeds never seen before.

Imagine a world where a code commit from an application developer automatically triggers a build in your automated build solution. The automated build is run including the

execution of unit tests. The build passes and is shared to a deployment automation solution

Combined with Agile development practices, Continuous Integration is another step toward high-frequency, high-quality releases. It allows development teams to produce testable builds at a very high rate. It also puts more pressure on operations teams to deploy more frequently to later environments.

A deployment automation solution integrated with the other technologies delivers what modern test teams need:

- The ability to reset the test databases as a step in the deployment automation process
- The ability to activate virtual services and simulate missing software and services for end-to-end testing
- The ability to launch a set of automated tests validating the functionality of an application end to end

Now stop imagining, because this capability is available today in the IBM UrbanCode deployment automation solution.

With the right application release and deployment methods, both upstart and established companies can quickly and consistently deploy their application on demand. As effective software delivery is a key to becoming leaner and business success, equipping your release and deployment teams with the right people, process, and tools can make the difference between long-term success and failure.

Decreasing expensive failures

The cost of a deployment failure depends on the environment in which the failure occurred. For the most part, failures in early environments, such as DEV and SIT, are far less expensive than failures in production. Finding an application error in these environments allows you to make a correction that may prevent a critical break in production — or worse, a critical breakdown for users. Forbes.com noted that in the summer of 2013, Amazon experienced a 30-minute outage that resulted in a projected loss of $66,240 per minute, or almost $2 million in total lost revenue.

Estimating the cost of failure

A failure not only can cause a loss of revenue but also can cost your organization more money to fix the failure. You can begin estimating the cost of failure of an application release by calculating an hour's pay for an engineer who is responsible for correcting the problem. A rough calculation of additional costs is 20 percent to 40 percent of salary. Here's the formula:

Estimated annual salary / 52 weeks per year / 40 hours per week * 1.3

If you have an engineer who makes $80,000, for example, a reasonable estimate of hourly total cost is

80,000 / 52 / 40 * 1.3 = $50

Now use this formula for every person on the recovery team, and multiply the result by the average number of hours spent taking corrective action. If a six-person cleanup team spends ten hours fixing a broken release, for example, the cost is $3,000 for that single release. That cost is merely productivity cost, however; it doesn't account for *opportunity cost:* the loss in revenue or the intangible potential loss of credibility due to the failure.

The cost of failure certainly drives the need for improved application release and deployment practices. In later chapters, we look at ways to prevent these failures from occurring.

Scaling complex releases and deployments

Organizations now tend to have more deployment targets than ever before on a variety of devices: local, cloud-based, and physical and virtual machines. Scaling a highly manual process up to an efficient, enterprise-wide system of interdependent application releases while maintaining security, traceability, and visibility is a difficult task.

The number of manual steps that go into a single application's release may make up a laborious process, but the application inevitably ends up in production. With larger, more complex releases, risk grows with the number of interdependent applications being released. Manual processes combined with primitive release tracking solutions such as spreadsheets invite human error and, inevitably, costly failure.

As applications scale to enterprise complexity, release teams often scale to dozens or hundreds of people, adding another layer of complexity. When IT teams scale to manage application complexity, job responsibilities and specialties become more siloed, and communication becomes less frequent as teams perform their respective processes. Silos occur due to job specialization as well as differences in time zones, cities, and countries. Distributed teams using different solutions and processes often discover that these differences double or triple their efforts and make them unable to deliver on time.

The coordination of each deployment, each process, and each associated set of manual steps requires herculean efforts and expensive planning and spreadsheet creation. Using a specialized tool that promotes visibility can help you plan more quickly, and execute more fluidly and accurately.

If you have ten steps to deploy an item to a server, and you go from one to ten servers, you go from having ten manual steps to having 100. Then, if you go from having one deployment item to ten items, you have another factor of ten, producing 1,000 manual steps. Manual processes won't scale as these three levels of complexity (number of items deployed, number of servers, and number of steps per deployment) come together to push exponential growth.

Organizations that can establish an efficient release and deployment process can keep up with the pace of development and can release smaller batches of applications, or even single changes or versions, at a time. Efficiency reduces risk and allows the organization to focus on timely satisfaction of business needs instead of process.

Eliminating the Release Weekend and Improving Team Morale

Working weekends to deploy the latest software release isn't something on anyone's "top ten list" of favorite things they like to do as a member of the release team. Very little could be less rewarding than spending the entire weekend at the office only to have to roll-back a deployment due to quality issues. However, this is the reality for many.

As many of today's software deployments are done manually, an organization typically needs to have all hands on deck to execute the many manual steps involved in deploying software and testing the quality of the deployment. Often a spreadsheet is circulated amongst the team where each person completes specific (or collection of) tasks and signs off on that task before handing off to the next person on the list. Delays in this manual handoff process are the norm. It is because of this manual effort and manual handoffs a release will often take an entire weekend.

If one step is missed or incorrectly executed, the entire team jumps into action to try and figure out where the process broke down. Revisiting steps and confirming each step was done correctly is counterproductive and unrewarding because the entire team would prefer to be spending quality time with their families.

Increasing efficiency, accuracy, and consistency by automating deployments can essentially eliminate the dreaded weekend. The Release Team is able to reduce what used to take the entire weekend down to a matter of minutes. As components are delivered to their target servers faster and testing can commence quicker than before, software quality can be measured delivering faster feedback on whether the deployment is stable enough to share with the external user community, requires a software patch to resolve an issue, or needs to be rolled back.

This goes a long way to improving morale, which in turn can improve employee retention for a company as teams move from being reactive to executing on more proactive tasks while learning, experimenting, and getting better at deploying the software the business depends on to meet its objectives.

Chapter 2

Best Practices for Deployment Automation and Release Management

In This Chapter

- Recognizing the four pillars of a gold-standard deployment process
- Realizing the benefits of a gold-standard deployment process

In this chapter, we focus on addressing both business and technology drivers by applying the gold-standard deployment process. At the end of the chapter, we show how applying these gold-standards can increase the success rate of releases.

The Four Pillars of Gold-Standard Deployment

To ensure a successful application release, teams should start by adopting the four pillars of the gold-standard deployment process:

- Use the same process.
- Automate, automate, automate.
- Deliver incremental changes.
- Release what you test.

We discuss these pillars in detail in the following sections.

Use the same process

One of the main problems with scaling application deployments to more complex scenarios and higher volumes is that siloed teams and specialists often create custom processes to fit their own responsibilities. Developers are often tasked with writing deployment instructions for operations to use in production but deploy to their own environments using simplified scripts. The different needs often manifest in the following pattern:

- Developers want to deploy and test quickly to implement more application changes. They often create shortcuts in the deployment process by writing scripts or by skipping burdensome steps that may be required for application performance in production.
- Testing teams strive to mimic the pace of development teams, but they understand the need to take their time in testing the applications. These teams often create their own deployment processes to get to testing faster and avoid delays.
- Operations teams can tolerate a slower pace as they strive for stability. Because they're responsible for keeping business-critical systems running, they can't endure the risks associated with rapid deployments. Their deployment process is designed to keep the production environment running smoothly.

When deployment processes differ between teams and their target environments, the chance of errors increases. Undocumented steps, environmental differences, and lack of input from all teams increase the chances of deployment failure.

To reduce the risk that comes from using different deployment processes, the best practice is to be consistent using the same deployment process when promoting from environment to environment throughout the continuous delivery pipeline to production.

Consistency in how deployments are performed results in increased confidence as you are now testing both the application and the deployment process.

Automate, automate, automate

Manual or half-scripted steps in a deployment process increase the risk of deployment failure. Even skilled operators make mistakes, and error is more probable when humans must run through a long list of manual steps. A deployment process (automated or manual) should include predefined quality gates or approvals that a team must perform or meet for the deployment to enter the next stage. By automating the deployment process (or at least what you can of it), you may add compliance and auditability through increased visibility into

- What components are included in the deployment
- Who deployed what application
- Where was the application deployed
- What version of the application was deployed
- When the application was deployed

There's a difference between automating and writing a few scripts that an expert still has to kick off. True automation removes risks of changing parts and allows authorized stakeholders to kick off a deployment with the click of a button. In addition, true automation provides self-service deployment capabilities, visibility, and traceability.

Deliver incremental changes

The third pillar of a gold-standard deployment process is making incremental application changes. Deploying only what changed means being able to deploy more often and on demand while lowering risk. This reduction in risk comes from deploying only what has changed, implementing quality gates to ensure a certain level of quality is met prior to deploying into the next environment, and automating an approval process. These smaller deployments are usually faster as well.

If you're using the same process for every deployment, making small application changes is both possible and desirable. Smaller application changes should be easy to pass through testing environments and into production environments.

When you're performing incremental changes, you may not be using the complete deployment process. To make incremental changes, set up the gold-standard deployment process that runs only the steps required to deploy what has changed.

Release what you test

In most software shops, it's assumed that you will test a change before releasing it into production. However, release managers often learn late in the testing cycle that one component of a larger system will not be ready to ship. Other teams may say that they can still release without the updates from the component that won't ship. Often they will be correct. Other times issues will surface in production.

For example, the Inventory Tracker component of an e-commerce application might be held back while updates to the Shopping Cart component are cleared for release. This pillar demands that the release manager ensure that Shopping Cart component be tested against the prior version of the inventory tracker before releasing.

This may be non-obvious. In an integrated system many of the tests are implicitly validating the Shopping Cart and Inventory Tracker work together. The testing up to this point may have only validated that the version of the Inventory Tracker that won't ship works with the Shopping Cart. Nobody may have actually tested the interaction between old Tracker and new Cart. To release without the additional testing would mean releasing something untested and expose the business to additional risk.

The Positive Effects of Gold-Standard Deployment

Employing the four pillars of a gold-standard deployment process (see the preceding section) can help you begin to align your organization's business needs with its technical needs, which can have positive effects on the application release. This section discusses some of those effects.

Avoiding the Monday Morning Effect

The Monday Morning Effect occurs the Monday following a major release weekend where the release team may face numerous problems, such as a break in production or defects in the application. The release team is required to take heroic, immediate corrective action. Restarting the release, correcting the mistake or mistakes, and ruling out other possibilities without incurring a substantial system outage are nearly impossible. To avoid the Monday Morning Effect, deployments must be automated and consistent, changes should be introduced incrementally, and release activity must be visible to everyone involved with the release.

Automating and using the same deployment process

When you use the same automated deployment process across multiple application life cycles, you greatly affect the overall release in a positive way. The release is less error-prone and takes much less time to perform.

Performing incremental releases

Performing a large deployment can be risky by itself, but deploying a large release of dependent applications is even more dangerous. By deploying small batches of change teams can minimizes risk while making the deployments easier to manage.

To release and deploy on demand, you must have a streamlined process in place. For this reason, continuously improving your standard deployment process and automating error-prone tasks are the keys to successful incremental deployments and smooth releases.

Managing defects and decreasing risk

There are three contributors to defects: code, configuration, and complexity. Complexity here refers to the number of interdependencies, or relationships, in the release package.

Whereas code and configuration defects increase risks linearly as batch sizes increase, complexity defects increase risks quadratically.

For organizations that use manual deployment processes or slow releases to gain better control of quality, the risk of failure has grown exponentially. Larger release batches laden with interdependent applications carry the risk of multiple failures. Using the same deployment process and automating manual tasks increases efficiency while lowering the potential for error, delivering incremental changes at a higher frequency also contributes to reducing risk to the business.

Release what you test

We strongly suggest readers follow this simple rule when it comes to testing a release: The Granularity of Release Equals the Granularity of Test. In simpler terms, adopting this rule means that if it needs to be tested together, it needs to be released together. This implies that integration testing is now a necessity to ensure that all the parts of a composite application are proven to work together — the changed and unchanged pieces — the entire composite business application can only be deployed with confidence as one unit.

If teams are deploying multiple dependent applications as part of a release, they then must rely on system integration testing to ensure these individual applications also work together as a cohesive unit. Managing this type of release becomes even more challenging as the application dependencies not only increase the size of the release but also the amount of risk associated to the release.

One doesn't have to look that far back in time to remember some story on the Internet sharing details of a software release that went poorly and the impact it had on the business. However, the level of risk can be reduced if this larger release unit is a known, versioned set of interrelated applications which have been tested and proven to work together delivering a sufficient level of quality to minimize potential impact to the business.

Chapter 3

The Next Step in Application Deployment Automation and Release Management Adoption

In This Chapter

- Understanding and preparing for changes
- Choosing the deployment automation solution that's right for you
- Selecting the right Release Management solution for your organization

Managing your application, or multiple applications, through the delivery pipeline with automation, while orchestrating and coordinating the overall release process requires a change in process, evolving roles, and specialized tools. While application deployment automation and release management solutions are designed to get applications to customers as quickly as possible, your delivery process itself must be evaluated, including one's role in that process, to use those tools effectively.

In this chapter, we discuss how to prepare your organization for the changes that deployment automation and release management solutions will introduce. Then we present basic criteria for evaluating these solutions.

Understanding and Preparing for Changes

Before you begin to look for a solution, remember the goals of deployment automation and release management solution from the business perspective: Speed time to market with less risk and cost — increasing the frequency of software delivery with increased end-user satisfaction, reduced time to feedback, reduced manual labor, and fewer errors with higher quality. Also, recall the goal of deployment automation and release management solutions from the technical perspective: Achieve repeatable, reliable, auditable processes — defining a consistent process to be used by multiple people or teams, executed repeatedly with very few (or no) errors, implementing well defined quality and user controls. Achieving both business and technical goals requires acknowledging the changes required for individual team members, interactions, processes, and solutions. Acclimating your people to the proposed cultural changes may be the longest and most difficult step in the implementation process, but it's required for streamlined, fully automated deployments and well-managed releases.

Defining roles and identifying changes in responsibilities

Some of the most important changes you're going to be making involve individual team member's tasks and the interactions among team members. Taking a note from the Agile Manifesto, you and your team members should be ready to adjust methods of interaction, the frequency of those interactions, and who they interact with. They must also be ready to accept automation solutions and the new roles that they will play within the organization.

Discuss changing team members' roles and responsibilities with management to get their buy in. This is critical in helping the employees accept that the automation solutions will help them perform their day-to-day job responsibilities. It is important that both management and the delivery team members understand that the solution isn't meant to replace human

talent; instead, it's used to help facilitate the outcome the organization wants to achieve by reducing the amount of risky repetitive tasks and enabling team members to focus on the tasks that require more creative thinking.

Try to find an internal (ideally, executive) champion who sees the benefits associated with the change. He or she can present the business objectives and share how the entire team contributes to achieving those objectives often provides focus, support, and perspective on the long-term effects of the solution. An executive champion who is on board with the new solution gives management a chance to hear the benefits from someone who can provide not only financial support but also air cover during the transition when there are inevitable bumps in the road.

You should introduce this cultural shift of evolving roles as early as possible. Chapter 4 gives you a rough idea of how long various changes may take.

Modifying process to increase effectiveness and efficiency

In a DevOps approach to software development, continuous experimentation and learning is one of the underlying principles. From these experiments and learning exercises, individuals and interactions may evolve to achieve business needs or goals. This may also mean that day-to-day business functions change. When introducing the solution you want to use, be aware that you'll face some resistance. Keep your team motivated while executing in a no-blame culture. While encouraging this cultural shift of continuous experimentation and continuous learning, organizations must be prepared and plan for periods of troubleshooting. Getting through the first major project using the solution and continuously testing the process to achieve greater efficiency will force everyone to challenge the new solution and new process. However, lessons learned along the way will drive out ways teams can become more effective and efficient in how they deliver software. Teams must also remember to constantly review their process looking for opportunities to optimize.

Which Deployment Automation Solution Is Right for You?

Application deployment automation solutions manage application components (and their versions), track which version is deployed to which environment, and log who deployed what to where. Due to the complexity of today's applications, these solutions are essential in achieving continuous delivery.

Supporting current and future technologies

When you're looking for a deployment automation solution, depending on your business goals, you should seek out a solution, which not only supports deployment to your current environments but also has the following capabilities:

- Reuses deployment processes across environments
- Coordinates application deployments across multiple tiers
- Integrates with existing technologies
- Easily extendable to support future technologies
- Provides role-based security and approval gates
- Maintains logs of all commands executed in deployments
- Tracks who deployed which version of a certain deployable artifact to which target
- Deploys infrastructure, middleware, database, networking and storage configurations (Infrastructure as Code)

Prioritize the features that your organization needs most to achieve the desired business goal. Your organization may want to ensure auditability and governance, for example, or to minimize the effort required to pass audits. In such a case, the solution's capability to provide role-based security and maintain deployment logs is more important than its other capabilities.

Successful deployment with IBM UrbanCode Deploy

A financial organization was developing a new trading platform to serve as the lifeblood of the organization. The development team used Agile development practices to produce results faster, but the process of deploying applications across hundreds of servers consisted of a mostly manual operation with customization required for each application. Introducing Agile development practices actually caused changes to build up for the operations team, whose deployment process wasn't equipped to handle frequent changes. This problem eventually brought development to a halt.

The financial institution still wanted to achieve the benefits of Agile development methodologies. After carefully evaluating numerous solutions, the organization replaced its manual deployment practices with IBM UrbanCode Deploy, which provided several benefits:

- Deployment times went from three days to two hours.
- The organization saved more than $2 million in the first year alone by eliminating the cost of manual deployments.
- The organization achieved compliance and gave teams a self-service option for deploying applications.

Provisioning environments on demand

With Cloud rolling in and becoming critical in an organization's ability to deliver software innovation with speed, the days of waiting for physical infrastructure to be ordered, shipped, installed, wired in, configured, and made available for use are quickly disappearing. Deployment automation solutions now need to offer the capability of provision these environments with minimal effort and at a lower cost eliminating one of today's biggest delivery bottlenecks. Hello "Infrastructure as Code."

Infrastructure as Code is a concept where environment patterns or "blueprints" are constructed pre-defining what an environment should contain or provide. These blueprints

are then used to provision on demand. From a click of a button, teams can now stand up complete and realistic environments leveraging cloud technologies and virtual computing resources. This capability goes a long way to on-boarding new resources, reducing the typical dev/test environment availability bottlenecks, and eliminating scalability and stability issues in production.

Selecting the Right Release Management Solution for You

The management of a release isn't a trivial matter. Understanding the dependencies between applications and making sure that the correct version of each is released into each environment in a timely manner while offering visibility throughout the deployment process — end to end — can be the difference between delighting customers with new business initiatives or losing money.

We don't need no stinkin' spreadsheets

Planning, managing, and executing a major application release is typically done with the help of spreadsheets or release documents that dictate the tasks, their owners, and the sequence required for a successful release. The problem with executing a release in this manner is that there's no way of tracking who did what, what code was deployed where, and when the release happened.

Release management solutions are designed to help in the planning and execution of a release by providing collaborative release planning that encompasses application and infrastructure changes. Ideally, a solution provides full visibility into the release process as well as end-to-end planning and execution capabilities.

When evaluating a release coordination solution, you should search for a solution that can do the following:

- Streamline the release of multiple applications in a release
- Update the release plan on the fly
- Display the progress of the release in real time
- Track changes in applications and infrastructure
- Notify stakeholders of release escalation
- Sequence and coordinate automated activities and manual activities in the release workflow, all relationships among activities, and all related communications to people and automated systems
- Automatically promote applications that meet the entrance criteria of lower environments

Although doing so isn't necessary, it's advisable to use a deployment automation solution with your release management solution. Just as application release and deployment solutions often work together toward the same goal, release management and deployment automation solutions often work together to help build an automated delivery pipeline. Release management solutions integrate with deployment automation solutions to kick off automated deployments of multiple applications during release time.

Understanding the impact of change

When multiple applications are included in a release, it's extremely important to know if all the changes in each of the applications has been successfully completed. Manually tracking development work items for each of the applications and quickly gaining an understanding of the situation to make that go/no-go decision can be very time consuming. Be sure to investigate and select a Release Management solution that offers integration to your existing change management system. In short order, you should and need to be able to see the status of all work items for all applications that make of a release.

Making the auditors happy

Capturing execution details for each step of the release plan, across all applications included in a release, goes a long way to meeting the needs of the audit team. Providing them with easy access to the information describing who did or who approved what along the way can ease their pain. Having an understanding of what may have gone wrong along the way and sharing that you are aware and actively "working the problem" so it doesn't repeat itself should decrease time to being compliant.

One client shared (without solicitation) that the level of information captured in IBM UrbanCode Release during a pilot project resulted in the Auditors mandating the use of the IBM release management solution across the entire organization.

Successful releases with IBM UrbanCode Release

A not-for-profit organization was spending far too much time planning and coordinating releases. Team members were using spreadsheets to track release activities and meeting multiple times to review the plans and changes in plans. The organization wanted to achieve better visibility into release efforts and to cut down on meeting time during the release process. It eventually selected IBM UrbanCode Release, the first solution designed specifically for complex application releases. IBM UrbanCode Release enabled the organization to reduce release meetings by 50 percent, reduce the length of each meeting by 50 percent, and add visibility to the entire release. By the end of their second release with the solution, team members had a good release template in place. By the end of the third run, the team leader who implemented the solution felt so comfortable with the solution that he went on vacation during the release.

Chapter 4

Rolling Out Solutions

In This Chapter

- Putting an application deployment automation solution to work
- Introducing a release management solution

After your organization selects the best application release and deployment solutions for its needs (see Chapter 3), you face the task of implementing those solutions. This chapter gives you some pointers.

Putting the Continuous Delivery Puzzle Together

Deployment automation and release management tooling fits into a broader DevOps tool chain focused on continuous delivery. The deployment automation solution in particular is dependent on actually having something to deploy. Deployment automation and release management tooling fits into a broader DevOps tool chain focused on continuous delivery. See Figure 4-1 for an example of continuous delivery pipeline.

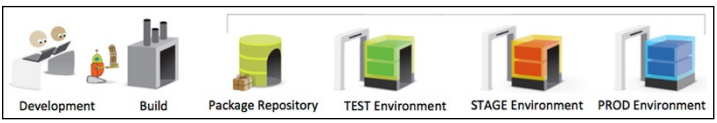


Figure 4-1: An example continuous delivery pipeline.

As we mention in Chapter 2, the goals of automation include traceability and visibility via a solution that tracks changing software components from build to through production

release. Knowing what those things are sure helps. So you need to have the things to deploy and know what is in them.

Changes will generally start with developers who deliver changes into source control. That source will then be built into working software by a build automation tool known as a Continuous Integration (CI) server.

The output of that process, the build, will need to be stored somewhere safe and managed so it isn't changed when you aren't looking. While well-structured areas on a file share are often used, it's best to use a dedicated repository to hold deployable software. These are broadly known as Artifact Repositories. There are numerous repositories available, including open source options. IBM UrbanCode Deploy also includes a repository out of the box.

Your CI server should provide traceability between changes in source control and the versions registered in your artifact repository. The application deployment automation solution will then be responsible for providing visibility between what is in the repository and what has been deployed to the various environments — on the servers. This simple tool-chain delivers auditable visibility from code change to product release.

Developers work on code that's delivered to source control. A continuous integration server transforms the code into a deployable artifact or "build." The CI server ensures that the build is registered in an artifact repository. The deployment automation solution is either informed of the new version or discovers the version. It then performs the deployments through the remaining environments. If those deployments need to be coordinated with other applications, a release management solution provides that greater level of orchestration.

To implement a deployment automation solution successfully, you must have a consistent, reliable build process in place. If your builds are inconsistent or unreliable, deployment automation may find the flaws of that process sooner, but the intrinsic quality issue remains.

Implementing an Application Deployment Automation Solution

We have seen organizations be successful rolling out a deployment automation solution in a number of ways. The most consistently successful approach is to find the right project, at the right time, with the healthy infrastructure. In this section, you also look at some pitfalls and how to avoid them.

Choose the ideal time for deployment

When you're moving forward with the implementation, your two major challenges are overcoming resistance from your team and finding the project or application that has the qualities you need for automation. These two challenges are intertwined. Choosing the wrong application for automation creates resistance, and automation that's poorly planned or poorly documented is likely to fail.

The best practice, therefore, is to choose the right time to deploy the application or project in question.

One ideal time is during a *greenfield project,* when a new application has a deployment plan that requires automation. An ideal project would be one that is important, under heavy development, and in need of help.

You have to take a deep look to tell whether your project or application is ready for automation. Ideal applications have a well-documented, repeatable deployment process. If you're looking at a greenfield project, it may use an application that's similar to a previously released application, or its existing deployment plan can simply be moved to the automation solution. The same rules apply to an established project. If you have a manual deployment process in place and are feeling pressure to deliver faster, you can move the process to an automation solution.

Figure 4-2 shows the tasks that you should be thinking about to get an idea of how to plan and roll out your solutions.

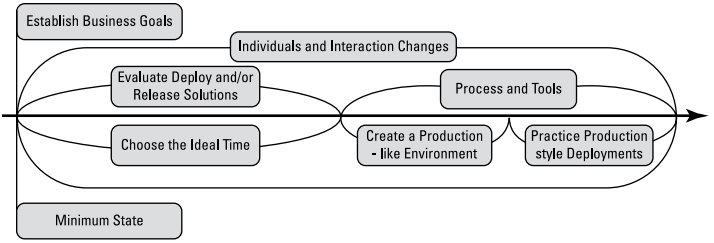


Figure 4-2: An example of an implementation and rollout timeline.

Ensure a production-like environment

As soon as you have your project in mind, you should ensure a production-like environment as a starting point for development. If test environments are too different, the validity of testing is questionable. Developers are famously chastised for "It works on my machine" excuses, and you don't want to be asking, "It worked in Test; why is it broken in Production?"

This environment will likely be smaller but should use the same operating systems, middleware, and configurations as the production environment. Production resources that are unavailable to test environments should be simulated through service virtualization, if possible.

For more information on service virtualization, see *Service Virtualization For Dummies,* IBM Limited Edition, at `http://ibm.biz/devopsdummiesbooks`.

A production-like environment improves the accuracy of your testing for both the application and deployment processes. You can simplify your environments as you work your way back to earlier environments and remove unnecessary components.

The DevOps team members from the Operations group should be brought into the deployment process discussion very early on. Their understanding of corporate standards around technology and the configuration of that technology are key when building production-like environments. The best practice is to first construct the production-like environment and work backwards. It would be silly not to tap into the knowledge and

expertise of what that production-like environment looks like or how it is configured — knowledge and expertise offered by the Operations team members.

This shift in thought and process is the essential goal of DevOps. It forces Development to collaborate with Operations, with both groups taking each other's concerns into consideration throughout the SDLC instead of just at deployment time. Development then has access to production environment expertise, which enables them to develop and test against a more realistic system, like the one that users will use. Operations teams also benefit by getting a preview of how their environment will react to the application and where support enhancements can be made.

Design for production first

If you want to fail, automate how developers deploy to their test environments, and then try to scale that toward production. The developer style deployment will often take critical shortcuts that don't matter in development but would be catastrophic in production. The classic example is dropping the database tables and rebuilding them from scratch. In production, your data tends to be a bit more valuable.

Instead, you want to create production style deployment processes, and execute them in the (simpler) development environment first. It's better to consider how you handle a load balancer and then skip those steps in early environments, then to not consider production concerns until you get there.

Working in these early environments is a good place to flex your DevOps muscles. Operations teams will be key to understanding the production deployment, while developers and testers are the key "customers" looking for speed and self-service.

Designing the process for production first provides several benefits:

- It allows teams to test their deployment process before the critical final deployment (see the preceding section).
- It allows teams to refine and further streamline the process. Automation furthers this goal by stabilizing

previously manual steps in a complex process and reduces unnecessary effort in future deployments.

- It drives alignment between development and operations and provides an opportunity for them to work together without the stress of a major production deployment event.

Developers focus on testing their individual contributions and the representative piece of the entire application that applies to their function in the project. For this reason, developers often don't take into consideration or don't know what's needed to keep production environments stable and functional. Due to the magnitude and scale of what needs to be tested and functional in production, the deployment process shouldn't be designed by teams that aren't familiar with the production environment.

To fully align your teams and facilitate collaboration from the start of a project, you must recognize the importance of starting with the most complicated processes and removing steps to simplify them. It's easier to remove steps than to add during a crisis or when a deployment has failed in production.

Implementing a Release Management Solution

Release management is ideal for organizations that release multiple applications at the same time or that desire more visibility into a complex release process. As we mention in Chapter 3, you can use a release management solution with or without deployment automation solutions.

The first decision teams need to make is whether they will implement release management on its own or with a move to deployment automation. While concerns around tracking changes on their way to production will be similar, migrating manual steps from spreadsheets to release management tool only to immediately migrate them to automation can be wasted effort.

Identify a realistic release model

To achieve success with a release management solution, you need to be familiar with your current process, the applications being released, and the complexity of those applications. You should start with a small release and work your way up to the release of many applications. Even in a small or sample release, you should still know the application(s) and feature(s) being released and who is involved in the typical process.

The key to this concept is scale. You want to begin with the smallest logical model of a typical release process. You should use a release that, though small, will be useful in scaling up to future releases by representing your ideal process from the beginning, which means including the people who will be part of the most complex releases.

Don't bog yourself down with "what-if" scenarios, because you won't be able to predict everything until you actually begin using the solution and discover areas for improvement. You and/or your team should find the way to implement the release management solution that provides the greatest value.

Choose an implementation path

After you choose a release to serve as a baseline for future releases, you should identify a logical path toward implementation. Three main paths provide a smooth transition to the use of a release management solution:

- **Use the new solution in parallel with the existing process.** The first path is to use the solution in parallel with an existing release process. This path enables the release team to map an existing process to the process in the release solution while they're both running. In practice, this means conducting a release in the legacy fashion while running the release in the new solution to see how tasks differ using the tool. This practice not only helps the team acclimate to the idea of the new process, but also provides a no-risk dry run for first-time use, thereby reducing anxiety about the next release, demonstrating the parallel tasks in each method of release, and providing a before-and-after picture of the process.

- **Conduct a post-release run.** The second path is to conduct a post-release run by using the solution to model a recently completed release — that is, releasing an application by using the current-standard process and then using the same process to release an application with a release management solution. Much like running the solution parallel to a release in progress, this practice gives the team a before-and-after picture of the process and a safe way to use the solution for the first time.

  The first two paths are exercises that provide realistic use cases for the solution and give team members a chance to realign their expectations and interactions.

- **Dive into the deep end.** The third path, which is less conservative than the first two paths, is to dive into the deep end and simply use the new solution for a live release. You should set up your teams for success by choosing a small release that you know they can execute with the solution. Take the selected release down to its minimum representative state of complexity and give it to your team as a low-risk live first use of the solution.

Chapter 5

Ten Myths about Application Release and Deployment

In This Chapter

- Seeing why automation can help, not hinder
- Understanding what a specific solution can and can't do

Sometimes, the best way to understand what's true about a concept is to understand what's false about it. In that spirit, here are ten myths about application deployment and release management.

Automating Deployment Means Writing Scripts

Although kicking off individual deployment scripts may work for small deployments to a few servers, that technique won't hold up under the scale and complexity of deploying interdependent enterprise applications.

Relying on a subject-matter expert (SME) to kick off deployment scripts puts a strain on the whole deployment team and especially on the person who owns that responsibility. If that SME is unavailable to kick off the script (or is no

longer employed by the organization when a change is ready to advance), the team can miss deadlines or miss the step entirely. In addition, if the SME is responsible for running the scripts for multiple deployments, the chance of errors increases as the likelihood of bugs increases with batch size. Then one must consider the human constraints such as fatigue, anxiety, and nerves that affect performance when a person is performing a long series of repetitive and high-pressure tasks.

The best solution is to let an application deployment automation solution execute the deployments. If your chosen solution (for example, IBM UrbanCode Deploy) can effectively integrate with your existing tools, you should be able to create a deployment process using a simple process designer that does not rely on writing scripts. Effective integration with other technologies, through plugins, and defining process steps in a drag-and-drop process designer can remove most or all scripting from your deployment process.

Who Needs An Ops Team?

Developers know how an application should work but may not know how the application topology in a production environment works, because they don't need to. Assigning the development team the job of designing a release process is likely to result in a plan that neglects key operational concerns or production-specific configurations such as clustering, load balancers, and integrations with operational systems such as monitoring and backup.

Starting to plan the deployment process in collaboration with the Operations team for a production release allows the development team to test the application against a realistic production-like environment throughout the delivery pipeline. It also allows the development and operations teams to test the deployment process early and often. The best deployment processes are the result of collaboration among development, operations, and release engineering.

Complex Releases Can Be Easily Managed without Specialized Solutions

Sure, you can manage simple application releases with various methods, such as e-mails, run-book documents, spreadsheets, and deployment automation solutions. The first three methods, however, don't provide traceability or collaborative release planning, and none of these methods can be scaled to accommodate the complexity of an interdependent enterprise release.

Although it's possible to use a deployment automation solution to manage a release, it's better to use a release management solution. Release management solutions are designed to assist in planning and executing a release of multiple dependent applications by providing collaborative release planning that encompasses both application and infrastructure changes.

Continuous Delivery Means Constant Production Releases

Continuous Delivery (CD) isn't continuous release. Instead, it focuses on speeding a new version through the delivery pipeline as fast as possible and then waiting for the business to decide when to release. The same technology is used for the production deployments, but the decision is a human one.

For most organizations, their automated tests are insufficient for validating that a new version is ready for production. For them, Continuous Delivery minimizes the friction caused by moving versions through environments and maximizes the productivity of testing teams.

Automation Compromises Controls like Separation of Duties

Automation boosts quality because it leverages computing capabilities to consistently run repetitive tasks that can be botched when performed manually.

Controls are also improved. The deployment button is behind role-based security, and approval and quality-gate rules can be enforced automatically. When you have a full audit trail showing who configured a process and who ran it, you know exactly what happened at all times.

A Spreadsheet Is a Good Release-Management Tool

Many release teams manage to function by using spreadsheets, but this method of coordination almost always results in human error and delay. Spreadsheets do enable management of the release process at the highest level, but they do not automate the tasks, which increases the chance of error, and they also require constant maintenance to ensure that all members of the release team are on track. Finally, as team locations vary, teams grow, and applications become more complex and inter-reliant, spreadsheets become more error-prone and less useful because they don't scale with release complexity.

Unlike spreadsheets, release management solutions capture interdependencies and determine the most effective deployment strategy. They also alert members of the release time when certain milestones in the process have been met or when certain team members' skills are required.

Release management solutions are highly advised for organizations that have a complex release process or distributed teams, or that have tried and failed to manage their release process with spreadsheets. This will also make your auditors happy.

Release management solutions can be used without deployment automation solutions, but it's advisable to use both to reduce the risks associated with repetitive manual tasks and to speed time to market.

One Large Release Is Less Risky Than Several Small Ones

In fact, large releases carry increasingly higher risk with the number of interdependencies included than small, frequent releases do. If you streamline the release and deployment process by using automation solutions, you free your people to improve the process itself and enable the organization to get feedback from the end-user faster instead of experiencing the delays associated to having to perform highly repetitive manual tasks.

Smaller releases have fewer pieces and fewer independencies. Errors caused by misunderstanding or poorly accounting for interdependency are dramatically reduced in a smaller release.

Incremental changes through true CD are the ultimate goals of any organization that wants to deliver value by delivering new features at a faster rate than its competitors do. Small, frequent changes delivered through automation solutions are the first step in reducing the cost and risk associated with traditional releases.

Release Automation Is Separate from the Build Process

Unfortunately, until you reach the minimum requirements of a working artifact repository, dependency management, and high-quality output from your build process, you're not ready to look into deployment automation. Deployment items must be versioned or ready to be versioned to allow for tracking of what's being deployed and where the deployment item is deployed. To complete successful releases that provide

visibility across the entire delivery pipeline, you need to make sure that dependencies are manageable, visible, and evident as part of your build process.

You usually can achieve the required minimum state with the help of a Continuous Integration (CI) server that also provides versioning and self-service capabilities for deploying to testing environments. Manual forms of CI can give you a leg up on deployment automation but lack the full visibility that we recommend achieving. The primary goal of a streamlined build process is to create working builds that are versioned and ready to be deployed through a number of testing stages onto production.

Some application deployment automation solutions provide a versioning capability with a proprietary artifact repository. This capability allows you to skip versioning your own builds and use the application deployment automation solution to incorporate that step. One such solution is IBM UrbanCode Deploy.

A Backlog of Deployables Doesn't Indicate a DevOps Problem

If you claim to be practicing CD, but your operations team is facing a huge backlog from development, you have a DevOps problem, and you aren't really practicing CD. DevOps practices and solutions help you seize market opportunities and reduce time to customer feedback by enabling true CD. When you decide to implement release and deployment solutions, you should prepare to work in a DevOps culture, which involves changes for individuals, interactions, processes, and solutions (see Chapter 1 for more on DevOps).

When you've minimized differences between development and operations environments, standardized your release and deployment process, and automated most or all of your manual tasks, you have a DevOps solution in place. This DevOps solution accelerates software delivery; reduces time to customer feedback; improves governance; and balances quality, cost, and speed.

Release Coordination Solutions Fix All Problems

The move from a completely or mostly manual release and deployment process to an automated one is difficult, and it requires several considerations. One of the most important long-term changes is in the culture of your organization. Your people and interactions must be aligned with the new method of work and potentially shifting responsibilities. You should prepare your teams for the shift, expect resistance, empower them to make decisions, and remain aware of what works and what doesn't work for your organization.

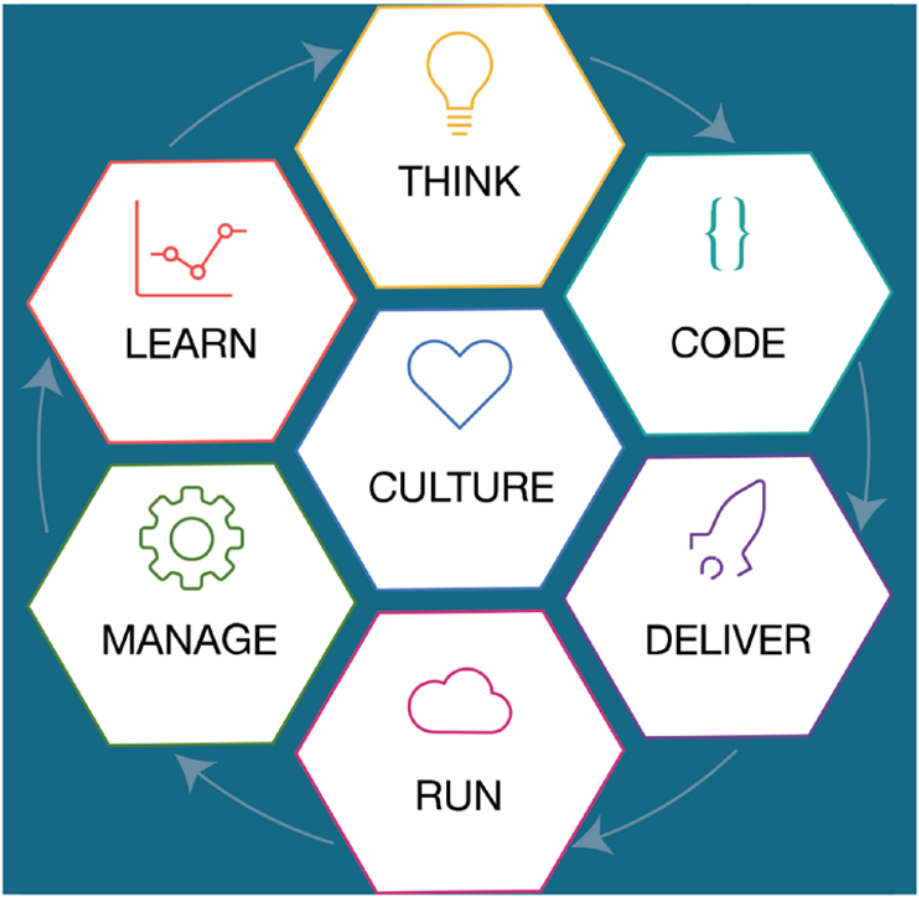
When you've selected the right project to implement application deployment automation and release management capabilities, your teams will have no choice but to collaborate. Preparing them as much as possible for the change, however, will make the transition easier and will help you reach your business objectives. Just as addressing build and CI issues reduce bottlenecks in deployment and CD, you may begin to see other opportunities for improvement in your SDLC after you introduce deployment and release solutions.

Notes

Notes

Notes

Regardless of where you are in your IT transformation journey, continuous learning is another important component in a DevOps approach to software development. To learn more on the topics of continuous delivery, deployment automation, and release management, along with other DevOps practices, we encourage you to check out the IBM Bluemix Garage Method: **ibm.com/devops/method.**



The DevOps Release and Deploy practice enables more frequent application delivery

A business' relevance is driven by its ability to release and deploy applications on pace with user's needs, while preserving the critical systems that support them. This book defines the basics of application release and deployment and provides best practices for implementation with resources for a deeper dive.

- ***Speed time to market* — Increase frequency of software delivery through automated, repeatable deployment processes across development, test and production**
- ***Drive down cost* — Reduce the amount of manual labor, resource wait-time, and rework by eliminating errors and providing self-service deployments**
- ***Reduce risk* — Deliver higher quality application releases with increased compliance through end-to-end transparency, auditability and reduced time to feedback**

Open the book and find:

- **The business and technical drivers behind automated application release and deployment**
- **DevOps practices for implementing application release and deployment solutions**
- **Evaluation guides for application release and deployment solutions**
- **Success stories from organizations utilizing application release and deployment solutions**

Making Everything Easier!™

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

FOR DUMMIES®
A Wiley Brand

ISBN: 978-1-119-27073-7
Not for resale

Part #: RAM14012USEN-01

WILEY END USER LICENSE AGREEMENT

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