Brought to you by

DevOps from APIs to z Systems

for dummies®
A Wiley Brand

IBM Limited Edition

Understand the mainframe's value

Use APIs to transform organizations

Digitally reinvent enterprise IT

Rosalind Radcliffe

DevOps from APIs to z Systems

IBM Limited Edition

by Rosalind Radcliffe

for dummies®
A Wiley Brand

DevOps from APIs to z Systems For Dummies®, IBM Limited Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN: 978-1-119-41023-2 (pbk); ISBN: 978-1-119-41024-9 (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. Burchfield
Editorial Manager: Rev Mengle
Acquisitions Editor: Steve Hayes
Business Development Representative: Sue Blessing
Production Editor: Magesh Elangovan

Table of Contents

INTRODUCTION 1
About This Book 2
Icons Used in This Book 2
Beyond the Book 3

CHAPTER 1: **Understanding the Value of the Mainframe** 5
Defining Mainframes and Their Importance 6
Calculating the Value of the Mainframe 7
Placing Mainframe at the Heart of the Hybrid IT World 9
Hybrid infrastructure 10
Hybrid data 10
Hybrid applications 10
Integrating Systems of Engagement with Systems of Record 11
Transforming Organizations to Service Providers with APIs 12

CHAPTER 2: **Understanding the Typical Mainframe Application Development Challenges** 15
Understanding Application Complexity 16
Measuring Risk in Changing Business Critical Applications 17
System Availability 18
Manual Test Processes 19
Fragmentation of Skills and Tools 19
Siloed Teams 20

CHAPTER 3: **DevOps and the Mainframe: Mission Possible?** 21
Defining DevOps? 21
Culture 23
Think 24
Code 24
Deliver 27
Manage 27
Learn 28
Why DevOps Is Critical for Mainframe Environments 28

The Balancing Act: People, Processes, and Tools 31
People 31
Processes 32
Tools 33
What Problems Does DevOps Solve? 34
Improving Application Development with DevOps 35

CHAPTER 4: **Looking at DevOps Best Practices for z Systems** 37
Adopting DevOps Best Practices for z Systems 37
Modern development practices and tools 38
Test environments 40
Common deployment 43
Improving Productivity and Optimizing Systems 43
Improving productivity 43
Optimizing systems 45

CHAPTER 5: **Applying Open-Source Tools to DevOps and z Systems** 47
Open-Source Pipeline 48
Git 48
Build system 50
Pipeline orchestrator 50
Deployment system 50
Build artifact repository 50
Open Source and z Development 51
SCM for z/OS source 51
Build system 52
Pipeline orchestrator 52
Deployment system 52
Built artifact repository 53

CHAPTER 6: **Building for the Digital Economy** 55
Understanding How Mainframe Is at the Heart of the Digital Economy 55
Looking at Full Application Life Cycle Management 58

CHAPTER 7: **Seeing DevOps Success Stories in the Enterprise** 61
Development Modernization 61
Large financial organization 62

Large financial organization in Europe 63
CICS development organization 64
Automated Testing and Deployment 65
Application Understanding 66
Seizing control of development 67
Unleashing progress 67
Full Practice Adoption 68

CHAPTER 8: **Making a DevOps Transition** 71
Getting Started with DevOps 71
Guiding You through a DevOps Transformation Process 72
Step 1: Pick the right set of applications 73
Step 2: Develop the vision 73
Step 3: Determine what's holding you back 74
Step 4: Transition to DevOps 74
Step 5: Measure the ROI 74
Step 6: Repeat 76

CHAPTER 9: **Understanding Where DevOps Can Take You** 77
How Cloud and DevOps Come Together in the Enterprise 77
The problem 78
APIs: A paradigm shift 79
Building the Next Generation Enterprise IT Skills 82
Fit for Purpose Platforms 83
Myth Busters for Mainframe DevOps 84
DevOps is for born-on-the-web companies 84
DevOps implies continuous deployment into production 84
DevOps and ITIL don't go together 85
DevOps and separation of duties don't mix 85
DevOps is only for small companies 85
DevOps is only about automated deployment 86
DevOps is all about improving cycle time 86
DevOps is a tool 87

Introduction

DevOps is a way of doing business; it applies to all aspects of an organization. DevOps brings Agile and Lean IT practices to the entire organization to allow for quick reaction to business demands. DevOps applies to all organizations — new, born-on-the-web companies as well as traditional small and large businesses that have been around for many years.

With the drive today toward an application programming interface (API) economy and the need to respond even more quickly to business requirements, companies must change their internal practices. This is why DevOps has become the de facto standard for high-achieving businesses. The ability to respond quickly to new ideas, experiment with different possible solutions, and deliver existing business with greater reliability and security requires this fundamental change.

In some ways the term *DevOps* is misleading, implying only Development and Operations are involved, but really it's all of the organization from the business through the running system. Everyone in the organization must be responsible for delivery. The organizations that embrace this overall change have seen the greatest value from the transformation.

In large-scale organizations this transformation can be harder, but it's more critical for just those companies. The number of silos is generally greatly increased in large organizations, and removing the silos is critical to DevOps. These large-scale organizations have existing systems that provide significant value but have been built up over years. These systems sometimes referred to as the *systems of record* have valuable business assets that need to be exposed to new business possibilities and need to be able to be updated rapidly. DevOps brings this ability to these systems and improves the ability to integrate these systems with the new mobile interfaces or systems of engagement.

DevOps is the cultural change to help achieve the value promise of the API economy, applying to all organizations and all parts of the organization. With DevOps, companies transition their policies and procedures to support this new culture as well as bring in the right tools to support this change.

About This Book

DevOps from APIs to z Systems For Dummies, IBM Limited Edition, is designed for executives, decision makers, and practitioners who are in organizations that have to deal with mainframe systems or anyone who wants to learn more how DevOps can work within large scale existing organizations.

This book also helps companies with existing development and operations organizations that are separated and siloed, including those that are outsourced. You also discover a little history behind how these silos developed and how now is the perfect time to transition to a DevOps culture.

DevOps from APIs to z Systems For Dummies, IBM Limited Edition, contains some examples of how other companies have gained business value in their transitions to DevOps. DevOps is an ongoing journey for companies, and in this updated edition, the examples show the greater value that can be achieved by continuing with the transformation and by truly building a DevOps culture.

Icons Used in This Book

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


TIP

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


REMEMBER

Anything with the Remember icon is something to keep in mind as you adopt DevOps.


WARNING

The Warning icon alerts you to possible side effects and/or implications of adopting DevOps.


TECHNICAL STUFF

Technical Stuff goes into specific details that may be more relevant to a practitioner than to an executive.

Beyond the Book

This book provides an introduction to DevOps and how it fits in organizations that have cross-platform environments including mainframes. But these pages can hold only so much, so for more information beyond what's covered here, check out the following links:

- **IBM Enterprise DevOps:** `http://ibm.biz/DevOps4z`
- **IBM DevOps for Enterprise Systems POV:** `http://ibm.biz/IBMDOESPOV`
- **Building an Enterprise API Strategy:** `https://ibm.biz/BdrjtH`
- **DevOps on z Systems eGuide:** `http://ibm.co/1QWTCDa`

IN THIS CHAPTER

- Defining mainframes and why they're important
- Seeing the mainframe's value
- Defining the mainframe's role in the hybrid IT world
- Discovering how to integrate systems of engagement with systems of record
- Using APIs to transform organizations

Chapter 1
Understanding the Value of the Mainframe

When you hear the word *mainframe,* many different thoughts may come to mind. To some, the mainframe is an old machine no longer in use. To others, mainframes are, by far, the most reliable computing solution available. There was a time when it was "common knowledge" that the mainframe was dead. To paraphrase Mark Twain, news of the mainframe's demise was greatly exaggerated.

Most consider the mainframe's birth to coincide with the April 7, 1964, announcement of the IBM System/360 line of computers. After more than 50 years of evolution and technological advances, the mainframe continues to run most of the world's businesses. According to IBM market development and insights

- The mainframe is essential to 44 out of the top 50 worldwide banks.
- 70 percent of the United States' largest retailers use z Systems.

- 10 out of 10 of the world's largest insurers use the mainframe.
- 90 percent of the world's largest airlines use z Systems.
- 1.3 million CICS transactions are processed every second, every day. In comparison, there are 68,542 Google searches every second globally.

In this chapter, I give you the background for the mainframe and how it's used today.

Defining Mainframes and Their Importance

Large scale computing started out with built-for-purpose machines, targeted to specific use cases, such as the business class machines and the scientific class machines. This division of labor even gave rise to two separate user groups, GUIDE and SHARE: GUIDE for the Business and SHARE for the Scientific. Over the years, these groups merged as well. In the United States, SHARE continues to be the premier user group for enterprise organizations. There are equivalent groups all over the world.

Some 53 years ago, on April 7, 1964, IBM introduced a new line of general-purpose computers to serve the needs of businesses, universities, governments, and, in fact, any organization that needed large-scale transaction performance. These computers and their successors became known as *mainframes.*

A number of operating systems have been developed to run on the mainframe hardware:

- For ultra-high-speed transaction processing, there's z/TPF or Transaction Processing Facility.
- z/VM is used to provide virtualization environments to allow other operating systems to share the mainframe platform.

- For more general purpose enterprise production workloads, there's z/OS.
- The most recent addition is Linux for z Systems, completing the Linux story across the server lines.

These different operating systems provide different capabilities for organizations while taking advantage of the underlying reliability of the hardware.

The current generation of the mainframe is the z Systems z13. The z13 can process 2.5 billion transactions a day — the equivalent of 100 cyber Mondays every day of the year. The z13 is the most modern mainframe hardware available, with a new processor design able to address 10TB of memory, with faster I/O and up to 141 processor units in one machine. A single footprint can run as many as 8,000 virtual servers.

Calculating the Value of the Mainframe

The mainframe was designed from the casters up to be the ideal platform for traditional online transaction processing (OLTP) workloads. The z13 was specifically enhanced to handle the explosive growth that today's mobile users are driving into the mainframe. The z13, with built-in encryption technology and the capability to do inline analytics, can be the basis for key business solutions such as real-time analytics and in-transaction fraud detection.

The z Systems machines provide this capability in the most energy-efficient machines. The average business class mainframe (z13s) uses about the same amount of power as just four kitchen coffee makers, or less electricity than a clothes dryer. While some server lines fill rooms and challenge the HVAC infrastructures that support them, with a single footprint, the z Systems z13 provides the processing power required to run large-scale applications. With built-in redundancy and efficient multi-site capabilities, a few of the machines can provide all the processing power required for large-scale organizations in full high availably, disaster-recovery configurations.

With the z Systems capability to run efficiently at 90 percent utilization (or higher), clients can put essentially all the processing power they purchased to effective productive work. With sharing algorithms, honed to a fine edge over 50 years of experience, mainframes provide the ability to consolidate many workloads into very few machines while still ensuring the most important workloads have the processing power they need. Mainframe users get all this efficiency with a nearly seamless ability to scale with their business.


REMEMBER

The LinuxONE system for running Linux workloads provides another demonstration of the flexibility of hardware that's designed and built with reliability at the core. By using LinuxONE, you can consolidate your various Linux servers onto a single system with the built-in security. In addition, the Blockchain High Security Business network on LinuxONE is providing a new foundation for the way business can run. By running on LinuxONE and taking advantage of the built-in security and the secure services container, businesses can trust the blockchain and have improved performance and scalability.


REMEMBER

In addition, z Systems are the most secure, with security built into it from the very start. z Systems have the highest commercial server security ratings in the industry. IBM z Systems are the only commercial platform with EAL5 security classification. This security classification helps guarantee that the system has been verified to meet specific security requirements. Only the z Systems servers provide an integrated encryption infrastructure — from the server, to the z/OS operating system — to protect sensitive data over the Internet, on databases, and tape.

Looking at today's mainframe, it's better to see it as a large server that has additional reliability and security built in. Thinking of it as a large server with its own quality of service (QoS) helps position it in the right way as you think about what applications or parts of applications you would run on the mainframe and what you would run on other systems. The term *fit-for-purpose platform* is generally used to describe this concept of picking the type of hardware most appropriate for the work being performed.

LOOKING AT WHO'S WHO IN THE MAINFRAME WORLD

To decipher the who's who of the mainframe world, you must first start with a little look back at history:

- A *Semi-Automated Business Research Environment*, better known as *SABRE,* is still at the heart of most airline reservations systems. SABRE has been running on z Systems since its conception.
- When Neil Armstrong and Buzz Aldrin became the first humans to walk on the moon, forever changing the view of the world, the System 360, the mainframe of the time, was used to process the data for the lunar landing and liftoff.
- Today, z Systems are used primarily in financial, insurance, healthcare, and retail environments. IBM z Systems process more than 23 billion ATM transactions per year worth more than $1.4 trillion (that's more than $3 billion a day).
- First National Bank (FNB) of South Africa is bringing mobile banking to the unbanked in Africa through the use of z Systems.
- As part of the 50th Anniversary of the mainframe celebration in 2014, a number of companies discussed their use of mainframe; companies like VisaNet have been running without an outage for over ten years, and it continues to run without outage.
- In the FIFA 2014 World Cup, 14 cameras were set up to capture goal mouth action. The data from these cameras was sent to a mainframe computer that analyzed each goal and, in turn, sent a signal to a watch worn by the referee to confirm that a goal had been scored.

Examples like these demonstrate why people use z Systems. They're built with reliability and resilience at their core, and they provide built-in encryption and the most secure system currently available.

Placing Mainframe at the Heart of the Hybrid IT World

Hybrid cloud is the standard today for large companies — the capability to run systems in whatever environment is appropriate:

public cloud, private cloud, or on-premise. Different factors contribute to the appropriate location for the systems, including security concerns as well as country laws. When governments require that personal data stay within the country boarders, companies, depending on where they're located and where the country they're working in, may need to add a new private cloud or add their own data center.

But when you think of a cloud, you may be thinking of commodity hardware and wondering what z Systems has to do with it. Critical business applications run on z Systems so they need to be part of the overall cloud solutions. And if you think about it, mainframes, in many ways, were the first cloud systems, providing a time-sharing system for multiple applications, allowing the scaling up of the application.

When thinking about hybrid cloud, different models exist. These are covered in this section.

Hybrid infrastructure

Hybrid infrastructure is about managing two or more different clouds as one big one. Consider the model where you want to leverage Linux instances from within the enterprise and in a public cloud. You want to have a single interface managing those instances regardless of where they're running. Businesses that can effectively utilize multiple cloud resources will maximize their efficiency.

Hybrid data

Hybrid data is about leveraging data from multiple sources to create information. This is important to businesses today because much of their traditional data is in relational structures like DB2 or Oracle, and much of the cloud data is in unstructured NoSQL models like Mongo or Couch. Businesses need to be able to unite both to be successful.

Hybrid applications

Hybrid applications are about splitting up applications and running them across multiple environments. This is extremely important to enterprises because this is about breaking up their monolithic applications and turning them into services or application programming interfaces (APIs). These APIs can then be combined

in new ways, with new tools and technologies. Businesses that unleash their business assets as APIs will not only create a new agile technological environment but also enable a powerful agile development model.

Integrating Systems of Engagement with Systems of Record

In the past, information was aggregated in the data center. The enterprise had applications running on the mainframe or distributed servers that took data and information and aggregated it. These applications were accessed through terminals or clients outside the data center. Even Internet applications followed the model of accessing information within a data center via devices that were accessed through a client (a browser).

The proliferation of mobile devices radically changes this model. Mobile devices represent a new way of turning data into information. In fact, mobile devices create information from data and other pieces of information. In short, these devices are the new collection point for information. These devices are called *systems of engagement* because they work with the users where they are. They also provide data about the users in addition to the users' direct input. They can provide location, contact, movement, or event data that can be provided by the device itself in addition to the input known by the users. They are meant to represent the users and their needs.

REMEMBER

These systems of engagement provide vast amounts of data and, in some cases, information itself, but they still need to take advantage of critical information available only in the data center. Therefore, they need systems of record to perform business functions that are just not suited to the systems of engagement. The *systems of record* are those systems, generally running on z/OS, that provide the actual data and transactions for processing the data. These two systems work together and provide new ways to understand, sell, and service the needs of the user community. They also provide a completely new way to design and deploy applications, which can change the way an enterprise interacts with its users forever.

Imagine this: You go to work in the morning, define a new idea for marketing your products to users (or perhaps a small demographic of your user community), and have an application supporting that idea out in the users' hands the same day. This is the promise of systems of engagement working with systems of record. Instead of carrying the burden of creating everything in one place, this model exists by taking the core capabilities in the systems of record and providing them to systems of engagement as APIs that can be easily consumed. These APIs then can be combined and recombined in different ways to create new results from the same base data. Because the tools used on the systems of engagement leverage tools like Ruby and JavaScript, they can create powerful apps quickly, and because they use systems of record APIs, they can inherit enterprise class capabilities.

Transforming Organizations to Service Providers with APIs

The model of systems of engagement talking to systems of record is another disruptive technology in hybrid cloud. Exposing existing business function via APIs allows enterprises to compete in a world that's beyond their reach and enables them to sell capability that was once only available internally. The model also helps enterprises turn their IT from a cost center to a profit center.

Representational State Transfer (REST)-based interfaces are being added to existing business logic to expose it as an API. The REST standard has helped transform the ease at which applications can interface with each other. The REST standard helps build new applications through the combination of multiple existing applications in ways never intended with the development of the original application.

REST is an architectural style to provide a simple way to interact between systems, providing a stateless interaction with simple HTTP interactions of PUT, GET, POST, and Delete. The messages are self-descriptive allowing a simple interaction. Traditionally web services have had more structure and definition, but REST makes it much easier to find and use services and is becoming the most commonly used standard.

Many of today's mainframe applications contain transactions that can easily be exposed as APIs by using the REST standard, and with capabilities such as SWAGGER used to define these APIs, it's easier for developers to find and combine a set of APIs into a new business or business function. Even if the applications today are monolithic, services can easily be created using z/OS Connect to define the API. These can then be exposed internally and externally by using an API manager. By exposing the existing mainframe applications business value through APIs, organizations can more easily access this value to drive new business opportunities.


TIP

Transforming to a service provider is a way to not only provide better value to the business, but also to change the way IT is seen within an organization. By becoming a revenue generator, the ability to invest and innovate increase.

IN THIS CHAPTER

- Understanding the application complexity
- Seeing the risk in changing business critical applications
- Knowing why system availability is key
- Looking at the manual test processes
- Discovering the implications of fragmented skills and tools
- Working with siloed teams

Chapter 2
Understanding the Typical Mainframe Application Development Challenges

Application development for the mainframe has remained the same for years (in updating this book, this is one of the chapters I was hoping to change dramatically; however, not much has changed in the last two years). In many environments, you see the same tools and processes in use that were in place 30 years ago. In this chapter, I review the challenges that have caused these processes to be put in place and how these tools and processes are now limiting the capability to change as quickly as today's business requirements demand.

Understanding Application Complexity

Applications that have evolved over many years, by their nature, will end up more complex than originally intended. Business critical applications, with their need to be highly available, drive even more complexity. Developers, unwilling to update existing code they didn't write (and may not even understand), tend to add new code that works around the existing code, making the result even harder to maintain. At the core of the original applications are usually sets of transactions performing some business task. For example, there could be a transaction to create a customer record, update a customer record, read that customer, or delete a customer record. These transactions would get some input usually through a 3270 screen, as displayed in Figure 2-1, and then perform the action on the data. The best of the applications didn't mix the display and presentation logic with the business logic but some did. These transactions are now functions hidden inside the monolithic application, but they're what provide the business value.

The applications that form the systems of record have been built and changed over the years by using tools with 3270 interfaces. Those 3270 oriented tools have survived long after the demise of the 3270 hardware thanks, in part, to the 3270 Emulator. 3270 Emulators turn modern workstations into 30-year-old user input devices. Figure 2-1 shows the standard Interactive System Productivity Facility (ISPF) primary option menu displayed on a modern workstation.

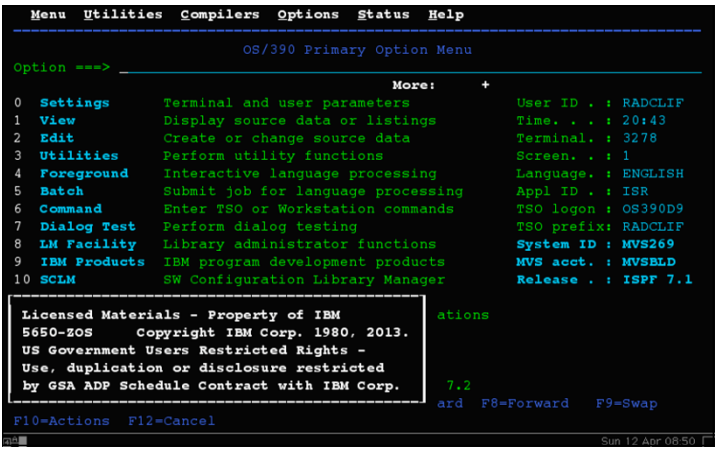

FIGURE 2-1: The standard ISPF primary option menu displayed on a modern workstation.

WHERE IS THE DOCUMENTATION?

Developers don't like to document their code, a reality that has not changed over time. With applications that started 30 years ago, the lack of documentation and clarity of design means the applications have been modified, changed, and extended over the years without clear understanding of the original design. Even when developers documented the original design, it was usually not tied directly to the code and rarely, if ever, updated as the code was changed. This left the documentation with little semblance to the current code base.

Originally designed in the 1980s and optimized for data entry, these interfaces were intended for a limited set of "power" users. Over time, end-user systems evolved, first as friendlier interfaces for Windows workstations, then to browser-based interfaces in many cases.

As part of the client server movement, application servers were added into the environment to access the back-end systems of record with new modern interfaces to hide the 3270 interfaces from the users. These application servers added another layer into the application design and additional complexity into the environment. They also provided a second channel for interaction for clients, allowing some users to use the traditional 3270 interface with others using the new front ends.

Now with mobile front ends, applications need to support additional channels for interaction. Applications continue to increase in complexity with the mobile app and browser interface, the mid-tier providing its own processing, and the system of record interacting with the data and providing the business transactions.

Measuring Risk in Changing Business Critical Applications

Any time an application change goes into production, a problem may cause an outage. Sometimes, deploying the change requires a short outage. You can plan for this. After the change is in place, unless everything was done absolutely correctly (taking all the

possible interactions into account), there could be problems. Many production problems are the result of an incorrect "deployment" of a change — not that the change itself wasn't coded correctly.

REMEMBER

Because the fact that making a change could cause an outage, many organizations reduced the number of windows where application changes could be made. Reducing the number of outage windows was seen as reducing the risk, and although more changes were applied at once, they were grouped and hopefully tested together. Reducing changes was seen as the key way to reduce risk, which would then group up changes into large releases scheduled for production in these planned outages. This grouping of changes, although seen as reducing risk, actually increased the risk of incompatible changes or required long testing cycles to ensure compatibility.

System Availability

In Chapter 1, I explain that mainframe systems have not only been in use but also have provided continuous service for years. *System availability* generally refers to the percentage of time the capability provided by the system is available. For many systems, this availability is generally measured as 99 percent, outside of planned outages. However, many clients measure z/OS availability by the number of years the system has been running without an outage. The basic assumption of the mainframe is that its hardware and software are designed to be both reliable and highly available. There is redundancy built into every part of the system. The software is written to support multiple workloads in a shared environment with failure isolation and dynamic recovery facilities that at least mask if not avoid outages as perceived by the users.

TECHNICAL STUFF

Specifically z/OS supports Parallel Sysplex or a sysplex environment, allowing multiple z/OS systems in logical hardware partitions (LPARs) to provide a seamlessly redundant environment for applications and their data. This allows updates to be made to hardware, operating systems, subsystems, and even the applications themselves without affecting the overall availability as seen by the user. The same has been true of databases with data sharing groups created to allow the data to remain available while systems elements are offline for maintenance. Pressured to deliver more reliable solutions, distributed systems are now adopting the concepts the mainframe has had in place and in production for decades.

Manual Test Processes

Manual testing is the norm for many mainframe applications. With little application understanding, determining what needed to be tested for each change was hard. For many organizations, this resulted in taking months to manually test any change that was going into production, running entire batch runs, and having the business analysts manually verify the application changes.

WARNING

The promise of automation hasn't delivered when it comes to testing z/OS applications. Building automated tests was difficult due to lack of automated testing frameworks designed for the dynamic, highly resource shared, multi-system environment that is z/OS. As testing frameworks such as JUnit were developed for distributed Java systems, no equivalent was available for z/OS until recently. With the latest updates in automated testing capabilities, now is the time to look at these tools again.

Manual tests were the norm as the testing tools were limited and the processes had separate teams doing the testing in a later phase. The concepts of Test Driven Development weren't used in the mainframe. With the advent of zUnit and with the transition to more application programming interfaces (APIs) to expose business functions, automated testing can be more easily accomplished.

Fragmentation of Skills and Tools

Looking at the application development environment available today, you see an unfortunate diversity of tools and practices. The z/OS development teams are generally still using 3270 based tools that have been around for the last 30 years. The distributed teams, however, have advanced their development environments, taking advantage of the latest and greatest tools as they build applications. While the z/OS teams stay with the same tools year after year, the distributed teams experiment and develop new capabilities to improve the development process. Starting with the individual development environment (IDE) that provides individual productivity, distributed teams have had the value of program flow, code completion, and a wide range of other productivity improvements just for the developer. Then comes modern

source code managers (SCMs) that allow for greater sharing and visibility to the code. The SCMs have evolved over the years to support parallel development, providing efficient ways for clearly understanding the state of the code for any level. Modern build tools have developed that aren't part of the SCM but allow users to easily select the right build tool and the right SCM for the work. These build tools have developed to provide complex build processes without complex scripting requirements. The list goes on for the new advancements the distributed teams have had while the z teams continued to use the same tools with very limited enhancements over the years. Some examples include xcode for iPhone, Android Studio for Android development, IBM Developer for z Systems for z/OS development, or web development environments associated with cloud-based platform-as-a-service (PaaS) environments, such as Bluemix.

With the distributed teams modernizing and taking advantage of modern tools and practices, they've been able to deliver code faster and share skills across teams. The mainframe team, by staying with its old tools, has limited its ability to share capabilities and increase speed. The distributed teams have learned through this that changing tools isn't so hard and not something to worry about. The tools can change to improve the process without breaking the application.

Siloed Teams

The fragmentation of tools and skills discussed in the preceding section has led over the years to the siloed teams you see today. These organizational silos are based on one of the platforms the solution is targeted for. Some exceptions exist to this where teams are building Java applications that might run on different platforms or some development teams specifically build code to support multiple platforms with C/C++. But this is much less common.

Silos generally exist for the mobile teams, distributed teams, and separate z/OS teams. The applications the business needs cross all these teams, but the teams continue to work separately; therefore, collaboration, if any even exists, is limited. The interfaces between these teams are where the hardest problems develop, largely because the interfaces are more afterthoughts than intended design.

IN THIS CHAPTER

- Understanding what DevOps is
- Looking at the importance of DevOps for mainframe environments
- Going through the balancing act: People, processes, and tools
- Solving problems with DevOps
- Improving application development with DevOps

Chapter 3
DevOps and the Mainframe: Mission Possible?

If you've been reading this book straight through, you've read about the challenges in the mainframe environment as well as the need for speed in addressing business needs. This requirement for change is why this chapter addresses this need, and DevOps is the way to do it. In this chapter, I define DevOps and how it applies to complex environments.

Defining DevOps?

REMEMBER

DevOps is about bringing the principles of Agile and Lean IT to the entire enterprise, from the initial business users, to development and test, through to operations. The term *DevOps* comes from the merging of Development and Operations, but it's important to understand it's more than that. DevOps is about an organizational strategy to manage the entire life cycle from requirements to a

running system, across all platforms and teams. DevOps is about the people and processes used. Tools help support the processes and people.

DevOps includes such practices as continuous planning, collaborative development, continuous integration, continuous delivery, continuous testing, and continuous feedback. DevOps is all about bringing automation, transparency, and collaboration to the entire organization.

REMEMBER

Notice that I said it's a *strategy* as well as about people and processes. DevOps is a transformation in the way people work and how they work together. It's about breaking down the silos that have built up between different groups over the years. Also, it's important to understand that while tools help, use of tools doesn't make an organization DevOps.

To understand DevOps, it's important to understand a little of the history behind how the development and operations organizations have developed. Over the years a virtual brick wall developed between the development and operations teams, though not always a virtual one. Development had its own set of practices and tools, which evolved over time. Operations had its own practices and tools, evolving in different ways. Development focused on producing more function faster, while Operations focused on controlling the environment to improve availability. A goal of DevOps is to break down the wall and instead have development and operations working together, collaborating to provide more and more timely business value. This is where Figure 3-1 comes into play. One side shows a developer with a machine, while the other side is users with servers showing operations. The brick wall needs to disappear for collaboration to exist.

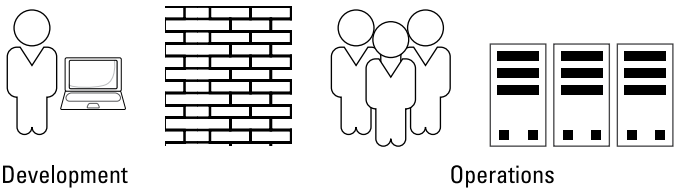

FIGURE 3-1: The brick wall between Development and Operations.

DevOps is a broad topic. To more clearly understand it, you can break it down into a set of areas:

- Culture
- Think
- Code
- Deliver
- Run
- Manage
- Learn

Take a look at Figure 3-2. It shows the continuous loop with Culture in the middle. Culture is at the heart of all DevOps.

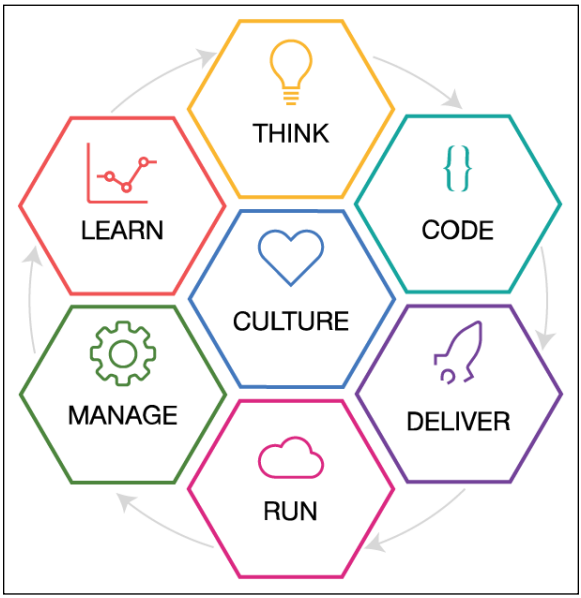

FIGURE 3-2: DevOps Continuous feedback loop.

Culture

Culture is at the heart of any organization; the culture dictates if the transformation brings value or you just change tools. Setting the right understanding across the organization and backing it up with practices and rewards is critical to gain the business value. Those companies that have transformed indicate that cultural change is what gave the organizations new life and excitement.

Think

Think is the first area of DevOps and represents the areas related to driving the overall business direction and receiving user feedback. Businesses need to be more agile, responding to the ever-changing client desires and government regulations. Achieving this goal of agility requires the ability to deliver quickly, but both high quality and appropriate feedback ensures what's being delivered is actually going to satisfy the requirements. This need to deliver fast change with immediate customer feedback as well as high quality presents a real challenge.

The current development environment consists of different teams operating in silos, making it harder to collaborate. The siloed environment forces extra processes, many of which are manual and slow down the delivery of function. The information required to plan and replan appropriately is fragmented across the teams, often in different tools. In addition, many teams see this planning and replanning as governance overhead that slows the teams down instead of an activity to help them deliver value with speed.

REMEMBER

DevOps helps reconcile the competing perspectives of delivering with speed and reducing outages while providing high-value solutions through multiple activities. It focuses on continuous feedback into the business planning cycle, identifies the key areas for development, and removes waste in unnecessary development.

Code

The code area focuses on all the practices related to development and test, including collaborative development and continuous integration.

Collaborative development

Software development practices, which have evolved over the years, have been formalized into many different disciplines, such as development, test, quality assurance, performance, and business analysis. In larger shops, separate teams often handle these disciplines. This leads to many different groups having to work together to deliver the end result. In addition to the teams organized by discipline, the advent of geographically dispersed teams and outsourcing of parts of development have significantly complicated the situation.

Collaborative development's goal is to bring all these disciplines together in cross-functional teams to more efficiently produce business value. This cross-functional team is supported by tooling to allow people, no matter their location around the globe, to truly collaborate together, sharing ideas and providing the transparency needed for the entire organization. In the best of all worlds, this cross functional team, including operations and business analysts, would sit physically together to take advantage of the natural collaboration possible through colocation. However, in today's world that's rarely possible, so tooling needs to be used to bring the team together virtually.

REMEMBER

Building cross-functional teams for collaboration doesn't necessarily imply or require an organizational reorganization, nor does it imply the removal of the separation of duties requirements. Different individuals need to be responsible for different tasks; they just work together in this cross-functional team. Changing the organizational structure from the very beginning isn't required, but over time, changing the organization will become critical to the continued expansion of DevOps across the entire organization.

A core concept of collaborative development is continuous integration, shown in Figure 3-3. Continuous integration implies individual changes are integrated early to drive out integration problems sooner. Continuous integration is supported through tooling to allow the teams to deliver and easily build, deploy, and test the solution through automation.

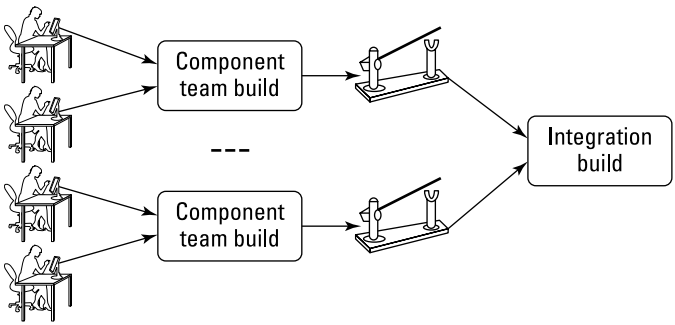

FIGURE 3-3: Continuous integration representing the bringing together of the build for each team and final integration between teams.

Continuous testing

Continuous integration is helpful only with continuous testing. Continuous testing provides the following:

- Early validation that the new changes don't break existing behavior
- Validation of new capabilities through automated unit testing
- Validation of integration between the various changes being made throughout the team
- Continuous testing of applications as changes are introduced to more quickly and easily identify the change with a problem

Continuous testing requires automated testing that's developed and maintained. Automated interface testing also helps introduce the ability to test the interfaces between parts of the application sooner and more frequently. The concept of "shift left testing" or the testing of interfaces earlier in the development cycle helps identify those harder to fix problems earlier and helps improve the interface definition so that both the caller and called programs have more clearly defined interactions.

REMEMBER

Automation needs to be available at all levels of testing from unit test, to function test, system test, and to performance and scalability testing.

The first or lowest level of testing is unit testing, which should be used to make sure all new lines of code are covered in the test. Tools such as jUnit or zUnit can be used to build up individual tests that can be run as part of new development with concepts such as Test Driven development. They can also be added to the regression bucket that's run with each build.

Function test, the next level of testing, should begin with function tests being built based on the requirements. This ensures tests are created as function is built and they can be tested together in the same iteration. Automating function tests at the API or interface level can simplify the process of creation of the tests and the maintenance of the tests. These tests can also be run with code coverage reports to provide additional coverage of the existing code.

System testing sometimes referred to as integration test, performance testing, and scalability testing are generally done in environments that are similar to the production environment.

Deliver

Deliver is the area of DevOps that focuses on the automated release and deployment process. Continuous deployment is one area that many organizations moving to DevOps focus on; however, it's important to consider that continuous deployment doesn't have to imply deployment into production. The deployment needs to include all environments within the organization.

Many of the tools that are so called DevOps tools are created for this area of continuous deployment. The goal is to develop a DevOps pipeline, shown in Figure 3-4, that allows the organization to move changes through the pipeline in an automated manner as far as the business requires.

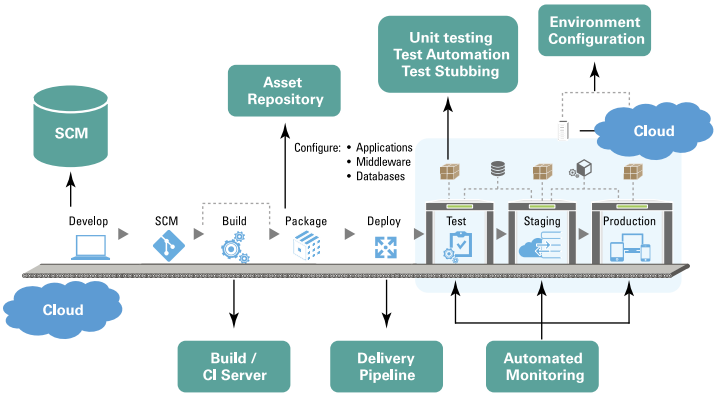

FIGURE 3-4: The DevOps pipeline.

Manage

The next area, which is manage, includes both Continuous Monitoring and Optimization, both providing feedback into the continuous planning process. Continuous monitoring provides the data and metrics to operations, quality assurance, development, and line-of-business. It's important to remember this monitoring isn't done just in production, but it's done in all environments to provide feedback as early as possible. It may not be reasonable to

run production-monitoring tools on each development machine, but the same monitoring tools that run in production should be run on the test environments.

The most important data a development organization can receive about an application is the data on how users actually use the application and any direct customer feedback. This information needs to be captured and not just used by Operations in tuning the system. It should also be feedback to the application design to improve the overall user experience.

One simple example would be noticing that on days when employees might logically be receiving paychecks, the number of requests for current balances significantly increases. This can indicate to development that providing a simple way of always showing the actual current balance through something such as the Passbook on the iPhone or PassWallet on the Android provides the function the users are looking for easier, without the same load on the back-end system.

Learn

The last area is learn; in many ways learn should just be an overlay of the entire process. One key goal of DevOps is to get feedback as quickly as possible and as often as possible. Do whatever it takes to experiment to provide the best possible solutions for your users.

Why DevOps Is Critical for Mainframe Environments

To understand why DevOps is critical for mainframe environments, you need to start with the development side. DevOps is about breaking down the silos, and nowhere are the silos more separated than between the mainframe teams and the distributed or mobile development teams. Separation develops for many reasons, but in the case of the distributed and mainframe teams, the separation was caused partially by different tools, languages, and development practices.

Over the years the tools, languages, and practices continued to evolve for the distributed teams — think OS/2, smalltalk, C/C++,

Java, Javascript, Python, and so on. Practices such as Agile and Test Driven development evolved in the distributed space, while the mainframe development teams generally continued to practice waterfall methodologies and develop in COBOL and PL/1. Because the practices and languages were not changing, neither did the tools. Developers became very proficient at building code in Interactive System Productivity Facility (ISPF).

For those of you who don't know ISPF, it's the green screen menu system available for z/OS, shown in Figure 3-5.

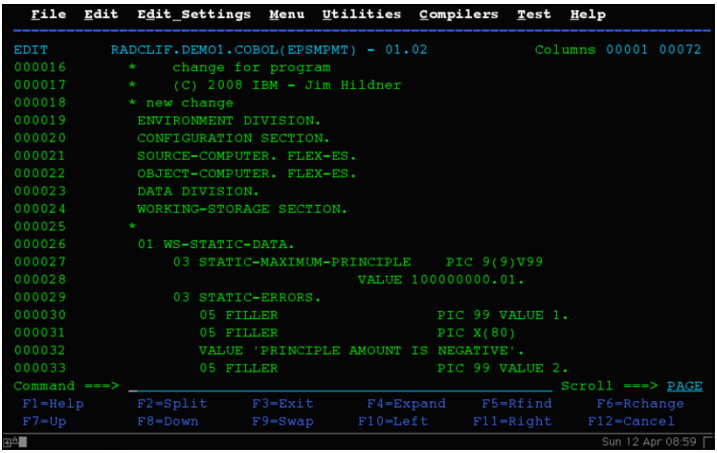

FIGURE 3-5: The ISPF.

ISPF provides the editor and the interface that allow others to build the tools for development and operations. Many vendors built library management systems and production control systems to control the source code and load modules for deployment into production. These tools were generally built for Operations to use to manage the environment and were then used by development as the tool was already available.

When I started in ISPF development 30 years ago, I used the same tools that many z/OS developers are still using today. Think about that. What else in this world is the same as 30 years ago? If you're a mainframe developer, you may be thinking, "Yes, but it works, so why change?" It may work for you, but does it work for your child? Remember these systems provide critical business value and need to be maintained and updated. The mainframe can be the system of innovation for the company, but does your child

want to use the tools and process you use, or does she want to use the modern tools she is used to? Mainframe organizations need to take advantage of the DevOps transformation in order to stay relevant.

WARNING

Current z/OS developers are comfortable with ISPF and the current practices, and, yes, it's possible for new developers to learn them as well, but to continue to grow and maintain the z/OS development, you must modernize the tools. The current products weren't built to support modern development practices, and new developers expect basic functions provided in modern tools that just aren't available with the traditional z/OS development. This is a significant change, but the value in not having to worry about lack of future workforce alone makes that investment worth it.

REMEMBER

The other important part is the move from waterfall to agile. Waterfall process increases delays and reduces quality by separating the people doing the individual activities and delaying testing among many other things. The move to agile processes improves the quality of delivery by getting feedback faster and having an integrated team working together. For the traditional mainframe applications, you may be wondering how this is possible with the monolithic code base, but in fact, this makes agile even more important. Making small changes and having a test suite to test those changes quickly reduces risk by quickly verifying the existing function.

Now look at the operations side. Again you will find separation of teams between mainframe and distributed. The mainframe operations teams have built up procedures over the years to ensure availability and reliability. In order to do this, generally they have reduced the times when changes are introduced into the system and have created release windows. During these windows, the changes that have been approved by the appropriate date will be deployed. These change windows are sometimes called *release parties*, where all participants involved are on a bridge call for the entire time, coordinating each part of the change to ensure all related changes are timed in the right order, across all platforms. Most mainframe applications today have some front-end system or system of engagement that provides the interface. Changes to these parts need to be coordinated to ensure the system correctly functions. This coordination is done through the bridge call. Because the teams have different processes and tools, people have to do the coordination.

Many times there is a third team involved in this picture as well — the Quality Assurance team. Check out Figure 3-6.

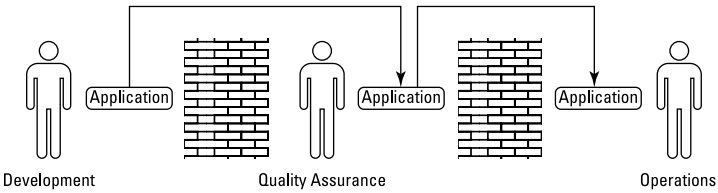

FIGURE 3-6: A look at the virtual brick walls between the Dev, QA, and Ops teams.

These teams can be named different things, but they're the groups responsible for testing the changes before they go into production. This third group has its own tools and processes. This leads to more silos and more virtual brick walls as shown in the figure: Now you have brick walls between Development, QA, and OPS and the application is figuratively thrown over the wall to the next team.

The Balancing Act: People, Processes, and Tools

The key point to a DevOps transformation is that it's truly about people and processes, where tools are used to help. Adopting the best collaboration and automation tools is useless without the cultural change associated with DevOps.

REMEMBER

Cross-functional teams focused on delivery characterize a DevOps culture. These teams focus on business objectives and place high value on early feedback, transparency, and automation.

People

Many people find change hard. DevOps is about changing the way people work to be more collaborative and cross teaming. Building a DevOps culture isn't like adopting a tool or a changed process; it's changing the focus of the organization to the end delivery and building trust across the various groups.

People have to be measured appropriately. Traditionally, development has been measured based on new function availability while operations has been measured based on availability of the systems. These can be seen as competing metrics. In DevOps, everyone should be measured on delivery of new function with availability of the environment. Everyone is therefore focused on the same end goal and can work together more efficiently.

Building a DevOps culture requires change throughout the organization, from the top leaders on down. One way to help with this change is to start with a cross-functional team on a project, and have this team be the example to learn and develop the practices. As this team gains the benefits of the DevOps transformation, it can begin the grassroots campaign to change the overall culture. Winning examples are always the best way to promote organizational change.

WARNING

Sometimes people are unwilling to change; if so, they may need to be moved to other opportunities within the organization.

Processes

In the preceding section, I discuss people and the cultural aspects of DevOps. Processes define what the people actually do. A DevOps culture is the first critical aspect, but you also need to change the processes to fit with this new culture. Numerous processes are associated with the entire DevOps model (too many for this book to cover). I've addressed continuous integration, continuous deployment, and shift-left testing. One additional process I want to discuss is the overall Change Management process.

Change Management is a key process for all businesses, and there are many aspects to change management. The primary objective of Change Management is to enable beneficial changes to be made, with minimum disruption to IT Services. Change Management addresses the entire scope of the change process. Within this there are various aspects of change that should be addressed. After reading the definition of Change Management from ITIL, it's important to understand that DevOps helps organizations satisfy this requirement through the automation and early validation to ensure changes are successful and provide business value.

Tools

DevOps is not tools, but tools do help support the transformation. Tools are required to support the automation, transparency, and collaboration required as part of DevOps. When determining what tools will be used, it's necessary to take an enterprise view. Most organizations today use multiple languages and run on multiple platforms. The tools selected need to support all those languages and platforms in use. If the tools can't support all the languages and platforms then those teams are left out of the collaboration and silos remain. Collaboration tools need to span the entire organization providing the transparency and integration with the other change management tools or help desk related tools. The deployment tooling should support all the platforms, allowing a coordinated deployment without the need of a set of people to coordinate between platforms.

REMEMBER

Specifically addressing the tools for the mainframe portions of the organization will be important to move to modern tools to support the new development practices.

DevOps has evolved over the last number of years. At the beginning, many organizations thought allowing teams to select their own tooling would provide the best solution; others looked for more standardization. Now it has become clear through many different companies' transformations that some tools need to be standardized, while others can have more flexibility. Several tools are needed for standardization:

- **The Source Code Manager (SCM):** This tool is one that's critical to standardize on. Getting to one organizational SCM should be the goal. This provides a single central repository to share organization assets, search for reuse, and build tooling around.
- **The Continuous Integration/Continuous Deployment (CI/CD) coordinator:** The CI/CD coordinator is another of the critical tools for standardization. Having a standard pipeline coordinator allows all teams to focus on their own processes and not supporting multiple tools, as well as allows the collection of central metrics.

REMEMBER

- **The work tracking system:** This single tool tracks work across teams, has a managed backlog, and coordinates collaboration — all of which helps organizations maintain a clear understanding of business work to be accomplished.

  Having a single standard tracking system across the entire organization, including operations, may not be realistic. For example, there will be a help desk system that collects problems as well. However, having the systems integrated so the help desk tickets are replicated into the problem tracking system used ensures a single managed backlog of defects and enhancements. Having different and separate tracking systems makes managing the work in progress for a team much more difficult and causes confusion on priorities.
- **The test management tool:** The test management tool provides a central repository for tracking and management of tests, which allows for reuse and tracking.
- **Deployment capability:** Application deployment for all required platforms simplifies the overall process and management. This is critical as a single managed system for the organization. This integrates with the CI/CD coordinator to provide clarity of deployment.
- **IDE and specific test automation tools:** These need the most flexibility because they're tied more to the language being developed and the system. This area is where individual preference can make a significant difference. I like to say it's not worth fighting the editor battle. People have their favorite editor; let them use it.
- **Code coverage tools:** Code coverage tools are required and should be part of the toolbox provided to the developer, but an appropriate code coverage tool for each language and platform is required. This is an example of where multiple tools are required but all for the same functionality.

These tools are covered in more detail in Chapter 4.

What Problems Does DevOps Solve?

Many organizations have understood that the number of changes should be reduced, so process and procedures have been put in place over the years to reduce the number of times that changes

are made in the production environment. This delays the deployment of business value and gives the false impression that the mainframe is slow. This causes organizations to implement function on other platforms, which then causes additional complexity in the application environment.

By adopting DevOps practices across the entire organization, changes can be delivered for the appropriate platform with the speed required for today's businesses.

Increased automated testing with proper code coverage indications can help reduce the time spent testing while decreasing the risk of those changes. Introducing fewer smaller changes into the environment more often brings business value faster.

REMEMBER

Building function in smaller pieces, by focusing on user-centric function and getting early feedback, and validating that function to ensure it satisfies the user needs early and often, allows for smaller incremental change delivering business value faster.

Improving Application Development with DevOps

DevOps with the principles of automation and incremental improvements help transition the overall development process. This helps all teams improve their ability to deliver business value faster. For the mainframe teams in particular, as comfort with the automated interface testing increases, the ability to make changes in the back-end application design can also increase. This automated interface testing allows developers to more confidently make major structural changes to back-end applications.

Many applications that have been developed over the last 30 years have become large and monolithic. With complete automated testing, developers can have the confidence to make structural changes. With application understanding tools, the developers can understand how the parts fit together, understand the dependencies between applications, and ensure the tests are covering what is required. Also with application understanding capability, applications can now be split up, redesigned, and the no-longer-used code can be removed. These improved applications become

easier to maintain and enhance. This allows developers to focus more on the delivery of business value.

Another key principle of DevOps, building the smallest increment of value and validating it with end-users, can now more easily be accomplished with the confidence of being able to regression test the existing capability. This small incremental value is many times referred to as the *Minimum Viable Product (*MVP*).* It's important not to be confused by the use of the word product. The goal is to build the smallest change that has incremental value that can be demonstrated to the end-user.

Not only is DevOps on the mainframe Mission Possible, but also it's Mission Critical in order to enable continuous software delivery to respond to business requirements and build competitive advantage. This transformation allows your business to gain the value from the resources that are seemingly locked in the mainframe.

IN THIS CHAPTER

- Looking at the DevOps best practices for z Systems
- Using adoption for greater productivity and optimization of systems

Chapter 4
Looking at DevOps Best Practices for z Systems

If you read Chapter 3 before coming here, you discovered what DevOps is and why it's important to mainframe development. Don't worry! You don't have to read that chapter before this one, but in this chapter, I focus on some best practices for this transition.

Adopting DevOps Best Practices for z Systems

When adopting DevOps, you should consider a number of areas. I cover those areas in this section with some recommended best practices.

REMEMBER

DevOps is about cultural change, breaking down the silos, and moving to Agile and Lean practices across the organization. Some of these best practices, such as building minimum viable products (MVPs), I discuss earlier in the book. The examples in this chapter are some of the best practices, but they aren't a complete list (that would require a much longer book).

Modern development practices and tools

Probably the most important step in the DevOps transformation for organizations with z Systems is to apply modern tools and development practices within the z development teams. Trying to break down the silos without this step is very difficult.


TIP

This transition to modern tools and practices can also be a great way to start the transition to the DevOps culture; in fact, your z teams may be the leaders in transformation in the organization through their significant transformation. They have the farthest to go, so starting with the modern practices and tools is a key factor:

- The first important practice is agile development. Notice that I didn't capitalize the "a" in "agile" — that's because it's not that you have to be exactly Agile; it's that adopting agile practices help improve software quality and improve speed. There have been many papers written about the death of waterfall, but unfortunately it's still practiced in many shops. There are many studies that show that waterfall hasn't helped improve software development processes, and actually it has done the opposite. Having silos between teams and trying to define everything upfront isn't possible. The requirements demanded by the business today may not be the top requirements for the business even a few weeks later. Having a managed backlog and working through items one by one as small enhancements help ensure that the correct functions are built, and time isn't spent on extras that aren't needed anyway. Agile incorporates many different practices, too many to discuss here, but the practices are the first place to start.
- The next modern tool is the Integrated Development Environment (IDE). Having modern editors and tools that help with application understanding allows z developers to work more efficiently and increases their productivity. This helps the existing team understand the code, as well as helps new members of the team more easily understand the existing monolithic systems. Providing a modern IDE also doesn't require a cultural change or any process change; it provides a set of productivity helpers and allows new developers to come up to speed on the existing code much faster.

REMEMBER

Providing a modern IDE is critical, but don't push it as a new editor. People have their own favorite editor, so let them use it. The key to the IDE is the new capabilities it should provide for application understanding and easy access to the z System. Using z/OS explorer as a base for all users of z/OS is a good start. It's provided for anyone who has a z system, so it's not an extra purchase, but it's the provided modern interface for developers and system programmers. The additional development tools should fit into z/OS explorer.

- Another area is a modern source code manager (SCM). Traditional host-based SCM solutions are generally more library managers than SCM solutions. Modern SCM solutions allow developers to more easily work with their changes and merge those changes with other changes also in progress. Supporting parallel development efficiently is key to these modern SCM solutions. The best modern SCM for your organization will be one that can support all the languages and platforms for the organization.

  On z/OS, you can work in a variety of languages, including COBOL or PL/I but also Java or possibly Swift. With this language variety, it's critical to have an SCM that easily works with all these languages in the way users expect. Having COBOL source in the same SCM as the Java code allows developers to see the interfaces used between the two and makes available to everyone the logic of the business rules. Another consideration is the file systems on z/OS — most traditional host SCMs were designed and built for partitioned data sets (PDSs). However, as the system has evolved, the hierarchal file system (HFS), which is a more natural file system for most people because it's just like any other system they work on, has developed and needs to be supported as a first-class resource. The modern SCMs support the HFS naturally.

- Collaborative development tooling is the next area of focus and should again support all languages and platforms. All teams should be able to work within the same solution so they can collaborate across the organization. I describe this capability in Chapter 3. The key from a mainframe perspective is that it's included. Providing all of this in dashboards available to the entire company helps facilitate the goal of everyone being responsible for delivery.

- For mainframe development, another required capability is application understanding. Existing mainframe applications are generally monolithic and not well documented. Having application understanding tools helps teams understand the dependencies between applications as well as understand the flow of individual programs. Clearly understanding the application makes it much easier for anyone to update the application and estimate the time required to do so.
- Modern testing tools, such as unit test capability, code coverage, and code rules enforcement, are also critical to this process improvement. With tools to help teams in these areas, the adoption of automated testing increases.
- In addition, tooling to help generate interface tests and virtual service definitions allows the teams to work independently but together at the same time.

These tools support the transition to the modern development practices, allowing teams to work together across the platform boundaries. With these tools, teams can more easily adopt practices such as Test-Driven Development, development of MVPs, and iterative development.

Test environments

Providing sufficient test environments for individual teams to be able to do their development and test, without delays, is the next key for the DevOps transformation. It doesn't help to build function in small increments or improve the productivity of the development team if there isn't an environment for them to develop and run their automated tests. Today, many organizations have shared test environments that are scheduled between the multiple teams. Figure 4-1 shows a typical testing environment today with multiple teams sharing the same environments.

Test data management

Test data management is an area often left out in the discussion of automated testing. As automated tests are created, the associated test data must also be developed to be able to run the test and validate the results. Test data should be created with all levels of tests, unit, function, performance, and scalability.

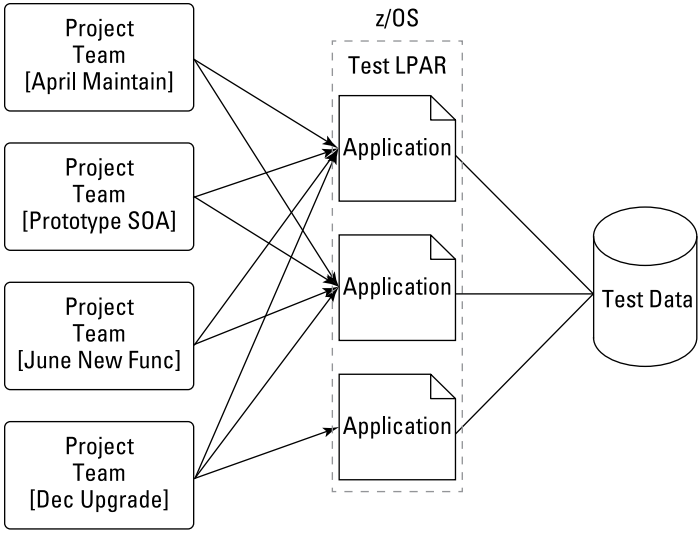

FIGURE 4-1: A typical z/OS testing environment showing sharing of resources.

There are many ways to create test data, and different creation and amounts of data are required for different levels:

- For unit and function testing, test data should be fabricated, and it should be as small as possible. As the environments come closer to production, more test data may be required.
- For performance and scalability testing, use production data that has been obfuscated or masked.

  Never just use production data; in just about every case, it will include some private information that shouldn't be accessible to the development and test individuals.

WARNING

One area generally lacking is negative testing or failure testing. This is especially true when using production data that has been scrubbed. Additional data must be added for the data that can't get into the system. Many teams build the happy path tests but fail to focus on the bad data or boundary conditions. Both for unit testing and for functional testing, having these negative tests is critical to ensure complete code coverage. The other area of negative testing is for security testing; adding extreme data and trying for buffer overflows are critical parts of testing that many times get left out of the mainframe testing. With the focus on bringing

mainframe applications to be directly exposed to mobile applications, security becomes more critical. All of this test data needs to be part of the test data management plan.

The goal of the automated testing should be to provide 100 percent code coverage. The 100 percent goal may never be achievable with today's existing systems because some percentage of it is probably never reachable. It's important to come up with the percentage you're comfortable with, and understand what isn't being tested. Realistically, companies start with current code coverage of 10 percent and add such goals as 100 percent code coverage of the new code, then slowly increase the coverage of existing or heritage code.

Interface testing and virtual services

One other practice is to use interface tests to discover integration problems sooner and to improve code coverage. Testing via application programmable interfaces (APIs) wherever possible helps improve the test coverage without building the more fragile user interface tests that may need to be updated with each change in the end-user interface. Figure 4-2 shows how virtual services can be used to test portions of an application independently.

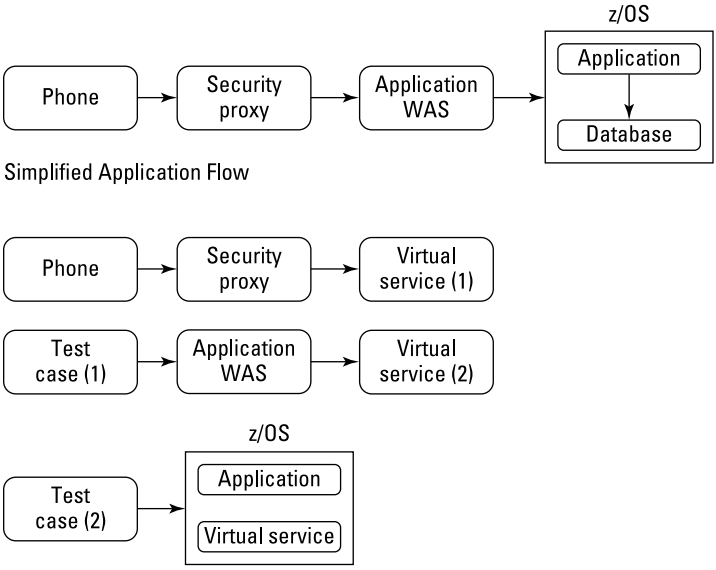

FIGURE 4-2: The use of virtual services for different parts of an application.

REMEMBER

For a Representational State Transfer (REST) interface that will be used between a mobile application and a back-end system, the teams can define the JavaScript Object Notation (JSON) schema to be used for the REST call. With this definition, a virtual service can be created to allow the mobile team to develop the front end without having access to a running back-end system. The interface can also be used to create a test case that's used by the back-end team as it develops the back-end capability. By using this same definition, the likelihood of integration problems significantly decreases.

Common deployment

Common deployment capability is another critical area to be addressed. With the complex nature of applications, making sure a single automated deployment capability is provided for all parts of the application and used in all phases of the development process significantly helps reduce deployment issues. This application deployment should also include the configuration required for any middleware component as part of the deployment. Imagine having a single tool that does the deployment of the mobile, mid-tier, and back-end system together in every phase of development. By the time the capability is deployed to production, it may have been deployed a hundred times in earlier environments, thereby discovering any deployment issues earlier. This also ensures that components actually tested together were actually deployed together.

Improving Productivity and Optimizing Systems

The first half of this chapter discusses some of the best practices in DevOps adoption. In this section, I let you know how adoption leads to greater productivity and allows the optimization of systems.

Improving productivity

One of the simplest productivity gains occurs with the adoption of automated testing. Many organizations see a significant gain in the transformation from mostly manual testing to automated testing. The first factor can be the change in skills required.

Instead of taking the time from the line of business for the basic manual verification testing, these individuals can focus on verification of the delivered value, with already tested capability. They can now focus on the value provided for verification and can spend most of their time on business activities.

With this transition to automated testing, development skills are generally required to build the tests. In the test-driven development and agile development strategies, developers themselves build more automated testing, and members of the team may cycle through building automated tests and building business capability. While the individual unit tests for function should be developed by the developer creating the capability, functional verification should be created by different individuals within the team so the tests are written based on the requirements, not the implemented code.

With automated testing being developed into the builds, developers' productivity increases by receiving almost immediate feedback on their code. In addition to the productivity gained through automated testing, modern tools bring significant individual productivity gains for developers who've been using ISPF and ISPF based tools.

Another area for productivity gains come from application understanding tools. By providing a tool that generates application flow, impact analysis, and data flow, users can more quickly understand where to make changes and how to make the changes. With impact analysis, you can also identify which programs are related to each other, allowing you to understand what areas need testing when making a change. This reduces the testing required and allows the initial testing to focus on the specific areas of impact to get feedback quickly. A visual of an application discovery browser is shown in Figure 4-3.

REMEMBER

Productivity gains aren't seen immediately. Education is required to learn to use the new tools and methods. Most individuals will decrease their productivity at first in the transition to the modern tools, but their productivity will significantly increase as their familiarity with the new tools grows. Individual experience varies, but in all studies that have been performed, the productivity increases are measurable.

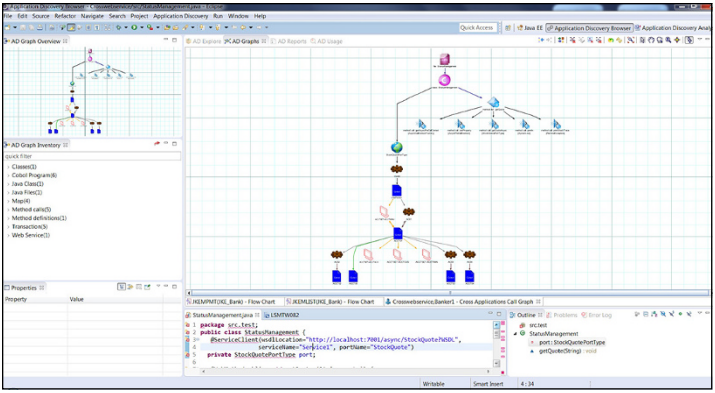

FIGURE 4-3: An application discovery browser.

TIP

Provide go-to experts on the new tools for the team. Have someone who has already learned the new tools assist others within a group to learn the capabilities.

Optimizing systems

With the new development tools, practices, and automated testing there comes a great opportunity to optimize the environment. For mainframe environments, the simplest first step is to take advantage of the optimizations by using the latest releases of the compliers. Each release of the mainframe hardware has developed new capabilities, and many of these capabilities require the use of the new compliers to gain the full advantage. Recompiling programs that haven't been compiled for 20 years is the first big step.

You may ask why these programs haven't been recompiled. Due to the desire to decrease risk, many mainframe organizations build only the changed programs. If some existing programs haven't needed to change, they haven't been recompiled. Now, with the automated testing capability, recompiling these programs introduces less risk.

REMEMBER

Even with automated testing, it may not be acceptable to recompile all the programs at once. Focus on those programs that are most often run. This is another area where development and operations need to come together, providing the data from the operational environment on how often programs are actually

being run over time and how they individually perform. This data can be used to make the best decisions on where to optimize first.

Another common focus area for optimization is identifying the duplicated solutions. Many organizations have grown through acquisition. Through the acquisitions, multiple versions of the same function exist within the organization. By identifying the various duplicated capabilities, business can decide how best to standardize on common capabilities, removing the cost and effort of maintaining the multiple solutions.

IN THIS CHAPTER

- Looking at open-source tooling for DevOps pipelines
- Understanding the pros and cons of using open-source tools in an enterprise environment

Chapter 5
Applying Open-Source Tools to DevOps and z Systems

When you see presentations or conferences about DevOps, you see many different open-source tools. Many organizations have focused on creating open-source pipelines for their distributed deployment process. Why is open source having such an effect? Open source is software with source code that anyone can read, change, and enhance. The goal of open-source software is to provide an open exchange, support collaborative participation, provide transparency, and encourage community-oriented development. Instead of a single organization being responsible for all parts, individuals with an interest can review and submit their own updates for enhancements or changes.

REMEMBER

Open-source projects have been created for many different kinds of projects. Open source for development and DevOps tools has gained significant interest due to the number of people contributing to the solutions.

In this chapter, I discuss some of the key aspects and tools that are being used, and then you look at the applicability to the z System. This chapter also covers open-source tools that have seen significant adoption in the marketplace. This study isn't scientific; it isn't intended to list all the possible open-source options, but it's based on my customer experience and the conferences I've attended.

Open-Source Pipeline

Most DevOps pipelines have a set of key capabilities (see Chapter 2 for more details). They include a Source Code Manager (SCM), a build tool, a pipeline orchestrator, a deployment capability, and a built artifact repository. There are a number of other standard tools such as a code quality manager, test managers, test automation tools, and the IDE used by the developer, but those vary much more often and aren't generally included in the standard pipeline required for use. Check out Chapter 3 for more information.

Git

Git is a popular open-source source code manager (SCM), and many different vendors provide supported versions. This SCM has become the de facto standard for the industry. It's focus on sharing code between users, and it's distributed support makes it ideal for development of open-source solutions themselves.

A distributed SCM takes a peer-to-peer approach to the solution, replicating the entire source tree, including all version information to any system in a network — unlike a centralized SCM where there's a central location that's the master with all the version information and individual users or build systems pulling the version they want to work with.

In Figure 5-1, you see the difference between the centralized and distributed SCM.

With companies trying to develop a model of internal open source, this has increased the demand. Also Git has been placed at the center of development tooling such that just about any developer already understands how it works.

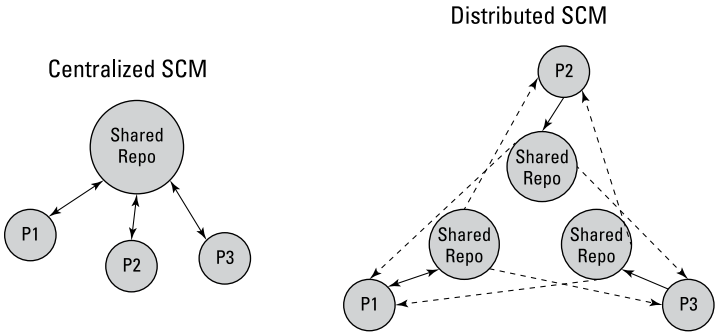

FIGURE 5-1: A comparison of centralized and distributed SCMs.

WARNING

Git does have its limitations. Because Git is a distributed SCM, replicating the entire code base to a local machine is easier, which makes it also easier for someone to walk off with the entire code base of an application or for someone to have a laptop stolen with the entire code base. For many organizations, this security risk limits their use of Git to non-critical or proprietary software. Others reduce the risk by requiring virtual desktop solutions, so the code always resides within the data center. Most organizations also require laptops to be encrypted, which reduces the risk if a laptop is lost.

Many people also have used their existing SCM to store the build output — *build output* includes the executables and many times listings generated by compiling the source code. This output is large and not needed by developers in their local copies. By the nature of Git replicating everything, this practice is no longer a good idea, and the build output must be stored in a build output repository and linked to the source code. The build output repository can be tied to the deployment system or can be a separate repository. The key is to provide appropriate links between the built output and the source code that was used to create it.

TIP

When using Git as the SCM, store your build output in an artifact repository instead of the SCM and provide the proper linkages between the systems.

Build system

The build system is responsible for compiling the source code and has been a target for a large number of open-source tools. The most common build tool has been Ant. Maven came along to improve Ant, and now Gradle provides even more flexibility and capability. Each of these tools provides the capability to define how the source code should be processed by the compilers. Tools such as Jenkins are many times used as the build coordinator as well as the pipeline orchestrator using Ant or another tool as the actual build script.

Pipeline orchestrator

The pipeline orchestrator is used to coordinate all the other activities. For the pipeline orchestrator, Jenkins is the most common open-source coordinator, but Travis CI has become more popular with smaller teams. Jenkins and Travis CI provide the capability to define a process flow and call other tools to perform those actions. They also pull the information back from those tools to provide dashboards of information about the status of the pipeline.

Deployment system

For actual deployment tools, there hasn't been a significant introduction of open-source tools. However, containers such as Docker have evolved to simplify deployment. Docker is an open-source tool that makes it easier to deploy applications by creating a container that includes the application and required libraries. By packing all the parts together, all the dependencies are deployed together so the developer and operation staff don't have to worry about the possible differences in an operating system. This container can now run on any system.

Build artifact repository

A *build artifact repository* houses all the output from the build process, such as the executables and the listings produced. This artifact repository will store all the various versions of the build output and can also be used to store the executables provided by external sources that may need to be deployed. The artifact repository can also be used as input to the build process when build output from one part of an application is required as input to another part of the application.

Artifactory and Nexus are the two most commonly discussed open-source repositories. Having an artifact repository has become a requirement for all customers using Git (see the earlier section, "Git," in this chapter for more information) to have a storage for the built applications.

Open Source and z Development

Open source plays a larger and larger role in applications and in development tools. As the ecosystem expands, z Systems has focused on leveraging open source. When thinking about z Systems, however, there are different aspects to keep in mind:

- The first is Linux on z, and when it comes to development tooling, it should be the same as the rest of your Linux development. There is nothing about Linux on z that makes the requirements different.
- The second is Java on z/OS, and for that environment, it's the same tooling as Java on any other platform.

For both of these prior systems or languages, many organizations have chosen to use primarily the same open-source tooling, if their company choice is open source.

When you're thinking about traditional z/OS development, answer these questions:

- Is open source appropriate?
- Is it available?

In this section, you see these choices in detail and how they could be used to work for your organization or company.

SCM for z/OS source

On z/OS traditional source code is in Extended Binary Coded Decimal Interchange Code (EBCDIC) code page and has a very different process for building. This code page difference is important for any SCM to understand and why open-source tooling that didn't understand this has been more difficult to use. Thanks to the efforts of Rocket Software, Git has been ported to z/OS and now

has code page support when running on the platform. It's been done in a way that the code can be in EBCDIC on z/OS but will be in American Standard Code for Information Interchange (ASCII) or Unicode Transformation Format (UTF)-8 for all other platforms, which allows z/OS code to fully participate in the open-source SCM space and to make it easier to open-source services, using traditional z/OS languages such as COBOL and PL/I.

From an SCM standpoint, now Git can be used for z/OS development if you're interested in the distributed SCM capabilities.

Build system

Jenkins has a z/OS connector that allows for a build process on z/OS that can use the Git support. Other open-source build tools haven't provided z/OS specific support. By using this support, you can submit traditional Job Control Language (JCL) to do a traditional z/OS compile or kick off other z/OS based tools for the build process.

Pipeline orchestrator

Depending on your build and deployment system, your pipeline orchestrator just has to be able to connect to them, so specific z/OS support isn't required. Companies have used Jenkins as a pipeline orchestrator for z/OS applications. Travis CI doesn't have z/OS support at this time.

Deployment system

Deployment requires specific z/OS support, and Jenkins provides the connector but no specific deployment capability for z/OS because no open-source tools exist that support the traditional z/OS languages. Docker images aren't directly supported on z/OS today; however, Docker-like function is being developed to allow for the easy creation of containers on z/OS for applications. As of the time of this writing, the first capability is coming out for CICS applications to get early feedback on the concept for traditional z/OS applications. This Docker-like container builds on the fact that z/OS is designed for running multiple applications within a single system and handling the workload appropriately.

Built artifact repository

As long as your build process can create an artifact that can be moved off z/OS or you create a secondary process that creates such an archive, any artifact repository should be able to handle z/OS artifacts. The difficultly will be in creating the archive to transport the files.

z/OS executable, known as *load modules* that reside in PDSEs, cannot simply be copied to an artifact repository; they must be extracted in a form that's transportable by using the XMIT function, and then they can be compressed together in a tar or pax file that can then be stored in an artifact repository. Having a packaging system that understands how to do this makes using an external repository much easier, but it's possible to do with any build system and custom scripting.

IN THIS CHAPTER

- Getting to the API ecosystems in the hybrid cloud
- Taking advantage of full application life cycle management

Chapter 6
Building for the Digital Economy

The digital economy is transforming the way business provides value to its end-users. Multiple interfaces are required to reach the different possible audiences. This change drives the need to expose more of the business value through application programming interfaces (APIs) to not only support internal demands, but also to support new external opportunities. In this chapter, I discuss the implications for the mainframe applications and how they can fully participate in this new environment.

Understanding How Mainframe Is at the Heart of the Digital Economy

Teams can work together in different ways. Common processes and tooling are the first aspect of having teams work together. With a common source code environment, the teams can share common artifacts such as interface definitions easily. Automated

interface testing is another aspect of how the teams can work together. By having interface definitions and building test cases and virtual services based on those interfaces, the teams can work independently of each other's schedules. This may seem at first like a counterpoint to working together, but by having this independence, teams can collaborate without having any one side hold up the progress of the other.

The common tooling, processes, and interface testing are tools and processes to help, but how do they actually work together to provide value? This is where application design and collaboration really come into play.

Essential to this discussion is understanding how the existing applications actually work. There is significant value locked in the mainframe applications, but it's sometimes hidden inside a monolithic system. You can't just go find the documentation, because it's not there or not accurate. This is one place where application-understanding tools are critical. These tools such as Application Discovery help you understand the relationships, understand the dependencies, and, most importantly, identify the areas that should be exposed as an API or service to be used by other applications.

Thinking back to when these applications were originally written (something I can almost do, but some actually were started *before* I was working in the industry), they were built as transactions in either Information Management System (IMS) or Customer Information Control System (CICS). These transactions performed some specific function, and the transactions were connected together. These core transactions many times provide the actual API now needed by a front-end application. Finding these transactions and identifying the parts that make up the capabilities are key in being able to expose them for new business purposes.

The front-end application, many times now a mobile application, provides the interaction with the end-user, but the mobile applications require data from the back-end systems. The mobile application wants to simply call a Representational State Transfer (REST) service to get the data or perform the action required for the end-user.

REMEMBER

With mobile applications, REST interfaces are the most common way of connecting. These REST services are provided in multiple ways:

- Directly with REST services from CICS
- Through the use of CICS transaction gateway
- IMS gateway
- MQ
- z/OS Connect Extended Edition

z/OS Connect provides a simple way to add a service interface to an existing application without changing that application. The key here is "without changing the application." z/OS Connect can easily be used in combination with application understanding to expose an existing set of capabilities to new business needs.

These services that are exposed via z/OS Connect or other methods are many times a more IT-centric view of the capability. Some of these may actually be business services that you want to expose to a wider audience. For this, an API manager such as API Connect can be used to expose the services, and then secure, manage, and meter them.

REMEMBER

Mobile applications will change frequently in order to satisfy user demand for constant change. Users expect updates for mobile applications, so the back-end systems have to support multiple versions of the mobile application and browser-based applications at the same time. In order to do this with the least impact, well-defined interfaces or APIs are created. By working upfront collaboratively on the design of the APIs, the APIs can evolve over time without breaking the front-end applications.

By defining and maintaining well-defined APIs, the challenge of deployment is also decreased. While it's critical to be able to deploy the full application together at any time, it's just as important to be able to deploy the individual components at their own pace.

These defined APIs provide the base for all the different interactions with the systems of record, allowing organizations to provide the various interfaces required by their customers. These APIs can also be provided as services to outside organizations to provide additional business value and enable efficient cloud environments.

LEVERAGING DEVOPS SERVICES IN THE CLOUD

Bluemix is a platform-as-a-service (PaaS) offering from IBM that provides an efficient way to build the systems of engagement connected to back-end services. Bluemix is a combination of IBM, third-party, and open-source technologies. IBM Bluemix DevOps Services provides the application development environment for these systems of engagement that's based on the same technologies that can be used for the in-house development.

Consider a well-defined and managed set of APIs from the back-end systems; now mobile developers using platforms such as Bluemix can easily experiment and develop very robust applications based on the information available from the systems of record. By providing these services or APIs to the Bluemix environment, your organization can more easily and rapidly adapt to changes in the market place. You can also use the same virtual service capability to allow the developers in test to build against those virtual services without having to have access to the real service early on.

Looking at Full Application Life Cycle Management

Application Life Cycle Management (ALM) is a well-defined process for applications from idea, development, and maintenance through retirement of the application. DevOps is a way of influencing the ALM processes to provide greater transparency, collaboration, and automation. The ALM processes need to be updated to provide the continuous feedback cycle and ensure full transparency within the organization.

As I mention in Chapter 1, DevOps is about bringing the concepts of Agile and Lean IT to the full organization. This means your ALM processes and procedures need to be updated to reflect these changes. In the current ALM practices, it's important to be measuring the overall practice. In order to remove waste, you have to

understand where that waste is. Some possible measurements to be capturing are

- Time to deploy the application or application changes in each stage, development, test, and production, along with the number of people required.
- Time to find defects from when the developer checks in the code. Is it found within a few minutes with the build, or does it take weeks before it's even tested?
- Time to complete a regression test. How much of it is automated? How many people does it take to run the tests?
- Defects found by stage and type. By stage I mean development, test, or production; by type I mean the kind of defect — was it a defect in design, misunderstanding of the requirements, coding defect, integration defect, and so on?
- Average time to get a new business requirement into production, from the initial requirement identification.
- Mean time to recovery when a problem occurs in production.
- Wait times, how long teams have to wait for test environments or systems.

Select the appropriate metrics for your organization and make sure you have some history for those metrics before the transition. These help you clearly show the return on investment in the transition. For example, if it takes five days to deploy the environment today, but you can do it in five hours after automated deployment tooling, it's easy to see that return. Not all measurements will be as significant as this, but they help demonstrate the value.

REMEMBER

One important note in the definition is retirement of the application. Retiring applications at the right time is almost as important as is building the right capability in the first place. As organizations develop multiple channels for user access, the need for clearly defined and managed ALM processes across all aspects of the capability becomes even more critical. The ALM processes and procedures must be clearly updated to be supportive of the cultural transformation.

IN THIS CHAPTER

- Starting with development modernization
- Working toward automated testing and deployment
- Practicing full adoption

Chapter 7
Seeing DevOps Success Stories in the Enterprise

Different organizations take different approaches to their DevOps transformation. In this chapter, you learn from a number of enterprise clients how they decided to take on this transformation and discover what they've gained. I have selected just a few stories for this book to provide some representative samples. More are available from the links provided in the beginning of the book.

Development Modernization

Many examples of organizations exist that start their DevOps journey with development modernization. This can be one of the least organizational change-related changes and is a start for many organizations as the first required step for the mainframe part of the organization. Development modernization includes the adoption of modern tools and practices for the mainframe developers.

Large financial organization

An example I would like to start with is a large-scale financial organization primarily located in the United States. It has both distributed and z/OS development teams. Development is located in multiple sites, including off shore; however, teams are generally located together rather than split across the sites.

As in most typical mainframe development organizations, this large-scale financial organization was using Interactive System Productivity Facility (ISPF) with an existing host source code manager (SCM) and deployment capability. This tool set had been used in basically the same way for the last 20 years. The z/OS teams are completely separate from the distributed teams. This client has some small applications with a few developers, and a very large application with a large development team. Honestly, I would say the large application is five different applications, but the client has so many shared components the client works as if it's one application. It has to be built and deployed together.

This organization needed to update its SCM and build structure due to the large size of its critical application. In order to make this transition, the company would require a significant redesign and update to the existing environment, and when it was finished, it would still be in the same tooling with the same old process.

The client assessed the effort for the change to the existing environment and the effort to move to a modern SCM and build solution. Both efforts required significant effort and time, but at the end of the modernization project, the company would have moved forward and have the ability to standardize across the distributed and z/OS development efforts on a single development environment.

To solve its problem, the company chose Rational Team Concert (RTC) along with IBM Developer for z Systems (IDz), formerly known as Rational Developer for z Systems, as the solution and planned a staged migration. The migration planned to move each application team at a time, leaving the largest application for the last. In this case, the company decided to start with the least active projects in order to learn from early deployments. With the scale of the migration and the number of users, the application and team migrations happened over a couple of years.

While the z/OS teams were migrated, distributed teams were also moved into RTC to provide the agile planning tooling required for their agile transformation. After the z/OS teams completed their migration, additional pilots were created to move to continuous integration, starting with some of the distributed teams.

As with any update, as the years have gone by, additional changes have been made. The prior example was in the first edition of this book, and now a few years later, the story has evolved. The teams have upgraded versions of RTC and are adopting new capabilities of IDz, including capabilities such as zUnit. They will also be moving to UrbanCode Deploy for the deploy activities currently done by RTC.

As the distributed side moves to UrbanCode Deploy as well, this provides a single common deployment capability. For the distributed side, as it moves to more cloud development, it has decided to implement Github Enterprise, Jenkins, and UrbanCode Deploy integrated with RTC. This will move the team entirely out of Clear Case and keep RTC as its planning, dashboarding, and audit point. RTC will be the system of record for all aspects of development, test, and deployment. Github Enterprise was selected to allow developers to replicate the entire code base with history to the cloud environment but still keep an enterprise managed central repository.

REMEMBER

This transformation is significant, so allow the teams a time in reduced productivity while they get used to the new tools and processes.

Large financial organization in Europe

Another example of transformation is a company in Europe that has a new development project on z/OS. For this project, it wanted to start out following DevOps practices and work to build a full end-to-end modern pipeline in support. In this case, the company has decided to keep the pipelines separate for distributed and z and are focused on providing the full stack for z/OS development, including the management of the middleware as part of the solution.

The organization selected Rational Team Concert (RTC) and UrbanCode Deploy (UCD) as the base for its pipeline. RTC provides the full SCM build integrated with UCD for the deployment steps.

As a key part of this activity, not only is the source code managed but also the configuration for the middleware, such as any MQ updates and Datapower configurations. UrbanCode Deploy provided the right plug-ins in support of each of the systems required.

As part of this new application development, this organization is creating service interface definitions for the capability so that it can be exposed to the front-end development activities. These service definitions are defined in CICS with the use of IDz. The process uses RTC to store these new type artifacts in components along with the COBOL application, uses RTC to build the parts, and then uses UCD to deploy them together.

This company plans to continue to expand the DevOps practices to database updates, scheduler updates, and any other system updates required as part of an application deployment. It also has a full practice around test automation so it can receive feedback quickly on development activities.

CICS development organization

The IBM development team responsible for building the CICS (Customer Information Control System) product had been doing development in the same way since the beginning of CICS. The CICS team wanted to be more iterative because it was using the same internal tools that all z/OS teams had been using for at least 20 years (the tooling is probably at least 40 years old). It was an old VM-based development environment. If you had grown up on the tooling, you loved it, but it was not helping attract new talent. It also couldn't be used for the new Java development the CICS team was working on, so a different set of tools was used for the Java development teams.

In order to facilitate its transformation, the CICS team started with RTC's work items and planning capability. This provided the collaborative development environment that all teams could use and a single SCM they could all move to. Over time the teams moved their full source out of the internal VM-based library system and fully into RTC. CICS also developed new automated testing capability that allowed individual testers to easily create new test cases to test the new function being developed.

This combined process changes and modern tooling approach has moved the CICS team to a more DevOps culture. The team realized business value in the form of

- Increased communication with a standard toolset for consistency and a platform for dialogue
- Increased speed with automation and cutting release cycles by 50 percent
- Increased responsiveness by incorporating feedback into daily development and builds
- Increased innovation through reduced waste and greater collaboration

  This area can easily be seen through the updates the CICS team releases, including application bundles and the z/OS cloud work. The CICS team has, through its transformation, become an example within IBM as to what can be done through modern practices.

Automated Testing and Deployment

A second focus area for starting a DevOps transformation is that of automated testing and deployment. Many organizations find automated deployment to be one of the major problems causing outages and delays in getting innovations to market.

Another large financial institution provided this next example. This organization has many separate units that provide different business focus areas. The group is spread throughout the country and has major locations around the world as well. A senior development manager provided the following description of its transformation:

"Our firm is experiencing a major change in how we think about our mainframe systems. Historically, these systems have had rigorous, precise but unfortunately slow upgrade cycles. New applications were installed at specific times and projects were mapped out in a detailed waterfall fashion. Agile methodologies have started to change how we think about our systems. The firm has started to think about a minimum viable product and how test automation can be leveraged to produce applications faster, test earlier and more frequently, and use automation to put those applications into production."

The firm is leveraging a large array of new technology in this change, and two technologies stand out as significant:

- **IBM Development and Test for z Systems (formerly Rational Development and Test for z/OS) D&T** allows the firm to create virtual z/OS instances on distributed hardware. The zD&T and systems serve a dual purpose. The first is they're easy to create just like any cloud system, so there's no harm in throwing it away after running a battery of tests and starting with a fresh copy for the next test run. The second purpose is as a learning system. What better way to learn how a complex system like a mainframe works than by actually using one and what better way to find out "what that button does" than by pressing it and not fearing any negative consequences because the system itself was virtual to begin with?

  The ability to create virtual z/OS images has been combined with a cloud deployment model to give fully automated API access to create the systems. The combination gives the firm a direct path from code checked into source code control, through build, unit testing, and functional testing on a z/OS system in a fully automated fashion.

- **UrbanCode Deploy:** UrbanCode Deploy allows the firm to create environments needed to test applications in a fully automated fashion. Taking the manual approach out of the environment creation allows for faster testing, but it also allows a verifiable way to create the exact same environment multiple times; that way the only thing that is changing is the application code.

  The UrbanCode Deploy environment creation and automation can be used on the virtual RD&T images and also on real mainframe systems, which gives consistency between test environments, QA, and production.

Application Understanding

As technology disrupts the insurance sector, providers are finding that traditional business models may no longer be enough to hang on to market share. Customer expectations are on the rise — and personalized products and services can be an all-important

source of competitive advantage. However, many leading insurers find their ability to innovate is slowed down by complex, inflexible systems of record.

The Senior Director of Quality Assurance for a leading UK insurance company elaborates: "We know how critical it is to evolve the application portfolio on our IBM z Systems platform to support new products and services. However, growth through mergers and acquisitions had left us with an extremely complicated application landscape, with some parts managed by outsourcing providers. We realized that over time, the quality of our application code had degraded, and our lack of control made it difficult to change."

Seizing control of development

The insurance company transformed its approach to development by defining and rolling out a set of best practices for coding, supported by the IBM Application Discovery and Delivery Intelligence solution. Initially, the firm focused on its COBOL-based policy applications, and used these as a baseline for the rest of its application portfolio.

"The IBM solution gave us the insight into our application landscape that we needed to develop coding best practices that take quality and complexity metrics into account," adds the company's spokesperson.

The firm's developers can now track compliance with the new standards early in the testing cycle. The spokesperson also added that "by monitoring how well our new code complies with our targets, we can reduce the number of defects once new applications have been released into production. Any defects we do find can be fed into a process platform for remediation, helping us fix them faster."

Unleashing progress

By increasing control and speed of development, the insurance company has gained the tools to delight customers, foster innovation, and out-maneuver competitors to win new market share.

"IBM Application Discovery and Delivery Intelligence helps us drive up the speed and quality of our development cycles," comments a spokesperson. "Earlier and more accurate testing gives

us the freedom to innovate without wasting resources. As a result, we can get new products and services out in the marketplace ahead of the competition, giving us an edge in a crowded industry."

The insurance company has also succeeded in driving down development costs: "Since deploying the IBM solution, our developers can work much more efficiently and need less help from third-party providers, allowing us to make better use of resources. Armed with IBM tools, we can look to future challenges with more confidence, safe in the knowledge that we can adapt faster than ever before."

Full Practice Adoption

In this section, I have included Nationwide Insurance Co., whose focus was on creating the full DevOps culture. It started with the goal of transitioning a full project with a cross-functional team and then expanding team by team.

Nationwide is a Top 100 Fortune Company, headquartered in Columbus, Ohio. It employs more than 30,000 associates and is a leader in several lines of business: Auto, Life, Home, Financial, Banking, Retirement Plans, Pet Insurance.

Nationwide has operationalized agile practices across IT, lines of business, and the entire delivery life cycle, as well as across technology domains that include the existing mainframe, distributed systems, and business intelligence data solutions. Using a DevOps approach, Nationwide can now perform continuous integration of its code and continuous deployment into its development environment several times a day. Teams can also perform acceptance testing of customer requirements in the same iteration with development. They can show the customer, in near-real-time, what developers are producing. This almost immediate feedback helps ensure that what's being produced is going to meet the customer's needs.

To support collaboration, Nationwide organically forms cross-functional teams that sit together physically at pods, comprising members from the business and multiple IT disciplines, including development and infrastructure. Collaboration has become an expected part of the culture and is built into the office space

and practices. The company has embraced change and knows that it won't know all the answers at the beginning of the software delivery life cycle.

The DevOps approach has transformed software development at Nationwide. Some of the specific outcomes seen are higher productivity and higher quality. It's not uncommon to develop code that, when delivered to system test, has zero defects. Fifty-eight percent of the Nationwide teams are in the top quartile of the industry based on lines of code produced. The company has improved software quality by 50 percent over the last three years, and it has reduced user downtime by 70 percent.

Outside organizations have recognized Nationwide's achievements. Nationwide is one of the first agile at scale organizations to achieve a Capability Maturity Model Integration (CMMI) Level 3 rating. Further, the company's Des Moines, Iowa, development center won a 2013 Iowa technology award for best use of technology.

Although the company continues to mature its DevOps practices, it's realizing business value from a growing adoption model.

Nationwide is building off these gains by extending agile and lean practices across the entire delivery life cycle. Its current focus is on providing the capability for integrated continuous flow and visibility. By creating this delivery highway, Nationwide provides business areas with the ability to deliver as fast as they need to, based on their own determination of cost, risk, and value. Time-consuming manual processes based on out-of-date information are replaced with automated deployment policies that use real-time information synced from systems of record. The combination of dashboard and dials gives the business the visibility and control to drive at a much faster pace than was formerly possible. Lack of trust in development outcomes has historically been an impediment to speed. The new automated approach provides confidence to move faster to serve the business without compromising the quality of deliveries.

IN THIS CHAPTER

- Getting started with DevOps
- Guiding you through the DevOps step-by-step process

Chapter 8
Making a DevOps Transition

In this chapter, I describe the steps to get started with DevOps. This chapter uses examples and experiences based on a number of customer engagements.

Getting Started with DevOps

DevOps is hard. Enterprise DevOps is harder yet. It not only represents a complete transformation of your software development life cycle (SDLC), but also requires a change in both mindset and organizational practices. Furthermore, it's not just about tooling and technology or infusing speed, but represents a more holistic way of managing software development. This transformation is no easy task, and there are many different parts of the organization that need to participate. There are also many different silos that need to be broken down between the different teams.

REMEMBER

DevOps is a journey. It's not something any large existing organization can take on all at once. The important point is to get started. Start somewhere with some part of the organization. Some organizations may already have parts of the culture with different terms. It's important to recognize what your current culture is to start with.

Today, however, very few organizations haven't started a DevOps journey in some way. Some don't use the term *DevOps,* but they're moving to a culture of small batches, automation, and continuous improvement through removing waste. Most of the organizations, however, have started with the distributed or mobile applications and have left the mainframe side out. Now they're looking at the difficulties that have increased due to the transformation of the front end, while leaving the systems of record process unchanged. Enterprises are confused on how to improve without breaking their systems of record.

As a result, many IBM clients have asked for help with "how to get started" and/or creating an enterprise roadmap. Many times this starts with a DevOps workshop, which can be the first time parts of the organization have truly talked together. These workshops need to be cross-functional and cross business areas so a clear picture of the current problems can be identified. Really this workshop helps point out the problems with the current processes and the differences between teams in those processes. So this section gives you the steps you need to make that transition into DevOps.

REMEMBER

Don't do these steps in a silo or department-only environment. Reach out to stakeholders in every area that the application touches. Invite them to the analysis, gain their input, get their opinion, and eventually their buy-in. Doing it just for your department is a recipe for disaster. Also, this exercise is meant to be iterative. Play around with all the steps. Tinker and move things around, change scope, update solutions, redefine vision, and so on until you come up with an objective that's both transformative and achievable. Remember, DevOps is a journey, which can be two steps forward and one step back.

Guiding You through a DevOps Transformation Process

The steps in this section are iterative, as the last step says Repeat. These are a set of steps that have been developed with many different client engagements.

Step 1: Pick the right set of applications

Not all applications are created equal. Some adjust themselves more easily to DevOps than others. Some require DevOps more than others. To be successful in DevOps, especially in the enterprise space, picking the right set of applications for the initial trials is key. IBM has a Digital Transformation Model for Applications to help you categorize enterprise applications and chart a roadmap for your DevOps journey. Remember, this is an organizational transformation; all applications will need to change. The first selected applications are important because you use these to learn how best to transition and provide an important example for a successful transformation.

TIP

Assuming you've already begun the transformation in the distributed space, pick a back-end application associated with a distributed application that's the farthest along in the transformation. The distributed team can help the mainframe team understand the transformation and help with the processes and procedures.

REMEMBER

Another way to start is with a mainframe team that's most interested in transforming. Never underestimate the value of existing culture and the team dynamics. A good team that's interested in transforming will help support the changes and help make your first changes the most successful. Using application-understanding tools to understand the impact between the applications can also help which to choose first. It may be easier to start with an application that's less interconnected with other systems.

Step 2: Develop the vision

Many organizations focus solely on speed of delivery. While it is a good start, frankly it's incomplete. What use is code put into production quickly if it's full of defects and takes an army to delivery it? Along with speed, think about your quality and productivity as well.

An example of a more holistic vision would be, "move from a quarterly release cycle to monthly, with only less than 5 percent defects slipping into production, and taking less than 50 percent of the current resources to deploy." In many ways, the three dimensions provide checks and balances against each other. Any vision without the other two will eventually fall apart in time.

Step 3: Determine what's holding you back

If you know where you want to go, the next step is to figure out what's holding you back. Analyze the overall cycle time (lead + processing time), quality (number of defects at various stages), and productivity (number of resources involved) metrics for the applications in question. A holistic study, across all three dimensions, gives you insights into not only where your true bottlenecks are, but also their severity and their impact on the software delivery life cycle.


TIP

Rank the bottlenecks by time delays/impact/severity to help you build the foundations of your DevOps roadmap. Perform this exercise over a variety of applications to get an organization plus an application view. Don't skimp here. Yes, it will take time, but the results uncovered will drive the rest of your DevOps strategy.

Step 4: Transition to DevOps

Resist the urge to jump to technology first (control your inner nerd; I know it's hard). Instead focus on the three guiding principles of a DevOps solution: automation, standardization, and frequency. Determine where and how you would like to apply these principles across your bottlenecks. Don't be afraid of choosing more than one. Case in point, if environment management is an issue, don't just stop at automation to help you with speed. Also use standardized platforms and process to improve quality and increase frequency of provisioning/de-provisioning platforms to impact productivity and costs.


TIP

Go deeper and pick a set of solutions for each bottleneck. Again, think in three dimensions: people, process, and technology. While picking options, make sure all the solutions can talk to each other. Think people and process first and then technology; think pilot not big bang, and for a fast start, consider partnering.

Step 5: Measure the ROI

It's amazing how many organizations, after going through the first four steps, simply skip this step. First, do a quick sanity check. If you eliminated *all* the bottlenecks, can you still meet your objectives? If not, go back to the drawing board. If yes, zero down on what bottlenecks, areas, or solutions you must solve first. Use the output from here to prioritize your roadmap.

Second, determine if this initiative is worth doing by building a return on investment (ROI) model. Look for a minimum of a two times the return over a three- to five-year horizon. Stick with speed, quality, and productivity for benefits. For costs, consider both real costs (software, services, consulting, and so on) and hidden costs (people and process change, disruptions, and more).

Enterprise DevOps solutions save money in many dimensions:

- By increasing staff capacity through productivity improvements
- By discovering defects earlier in the life cycle, thus reducing the resolution costs and improving quality
- By automating deployment across the parts of the application, thus removing deployment errors

In order to calculate the ROI for your DevOps transformation, the first aspect is to measure what you're doing today. Understanding your current activities helps in calculating the benefit to the transformation.

REMEMBER

Key areas to measure include the following:

- The time it takes to test
- Time spent waiting for resources or the next step
- Defects found by phase
- The time it takes to deliver an idea to the end-user, from requirement to delivery for small and large changes
- Time to complete deployment in each environment

All these metrics should be reasonable to calculate in an existing environment.

The next step is to monetize the value of improvements in the velocity of delivery and the increase in application quality. These metrics should be continuously measured, then as the transformation takes place, these updated metrics can be compared to the original metrics to show actual return values.

Many examples exist. For instance, in one client example, the deployment times went down from 49 to 99 days to 3 to 5 days. That ROI is easily seen.

Step 6: Repeat

DevOps is a transformation. Once you have transformed the first few applications and gained knowledge and understanding of how it works in your organization, it's time to repeat the cycle through all applications. It doesn't end once all application teams are transitioned. Now comes the continual improvement. Continue to look for additional bottlenecks and areas for improvement, and continue to improve. It's important to keep the metrics in place to see the continual improvement.

REMEMBER

All applications need to be transformed. If you have some applications that you think aren't worth the investment to transform because of limited or infrequent updates, think about a few years from now. All the rest of your development and operations teams work in a DevOps manor, and you have these other applications using a different process and set of tools. You have to maintain those other tools just for these infrequent updates, and you have to have people remembering the other processes, and you open yourselves up for additional errors. There are no applications that should just be left behind.

IN THIS CHAPTER

- Bringing Cloud and DevOps together in the enterprise
- Building the next generation enterprise IT skills
- Looking at fit for purpose platforms
- Seeing the top myth busters for mainframe DevOps

Chapter 9
Understanding Where DevOps Can Take You

In this chapter, I discuss the additional value a DevOps culture can provide an organization. This chapter provides specific examples of how cloud relates to the mainframe in this new DevOps environment. In this chapter, I also discuss a number of myths that have developed around DevOps for the enterprise and how you should determine the best fit platform for your unique development needs.

How Cloud and DevOps Come Together in the Enterprise

Now that you've started your DevOps transformation, how do you deal with the increasing demands for capabilities as well as the disruptive technology advancements? This section goes into more detail of mobile and cloud and how the enterprise can deal with the technology challenges to come out ahead.

The problem

Most enterprises are facing competition today that didn't exist five or ten years ago. Back then their competition was other enterprises. Today an enterprise also competes with small start-ups that provide only a small piece of what the enterprise provides. These start-ups begin with a DevOps culture and don't have the existing organizational structures or systems to slow them down. While none of these start-ups really competes with the enterprise in its entirety, together they represent an attack that affects the enterprise's viability.

REMEMBER

What makes these start-ups so attractive to users is that they use new technologies and methodologies to connect to the user personally, and they have the agility to change their products quickly to meet the user's changing demands. The start-up can have a new app out and on a user's phone in the time that the enterprise takes to determine which systems have to change to support the demand. The complete DevOps culture is what allows the small start-ups to quickly respond.

What has given these start-ups so much power? Mobile devices. These tools bring new power to the user in a new unprecedented way. In the past, the collection point for creating information was in the data center. Whether you had distributed servers or mainframe systems, the focus was having server infrastructures in the data center that turned data into information, and you had clients or terminals outside the data center that would communicate to them. Even when the web began to provide browser front ends to information, the real work was always done in the data center. Today, smartphones allow users to obtain information outside the data center. They take information from multiple sources and combine it with data that only exists on the phone to build new information tailored to this specific user. Start-ups don't have the myriad services that exist in the enterprise. They don't have the business assets that the enterprise has constructed over the last 50 years, but they *do* have the ability to morph and change to meet client demand immediately, and they leverage that to take business away from the enterprise.

The belief by some people in the cloud field is that cloud marks the end of the enterprise. They believe this because they feel that there's no way for the enterprise to move with the speed and agility that these start-ups move. They feel that the services that the enterprise

provides are the very inhibitors to success because these services are monolithic applications that inherently can't move quickly enough to support this new model. But what if there was a way to retain those business assets and be agile too?

If an enterprise could embrace a model that allowed those business assets to be used in an agile way that would be a disruptive technology, it would break the conventional cloud belief that the future belongs to a bunch of small components playing together. This model is, at its core, a DevOps culture.

APIs: A paradigm shift

As cloud starts to focus on providing application programming interfaces (APIs) via services instead of applications, it provides new opportunities for the enterprise. If an enterprise can express the business assets that are currently combined in an application as APIs, it can combine those APIs in a more dynamic way downstream. This breaks up the value chain that's currently locked together in an application and makes each link available to be used in new ways.

For example, imagine you have a store with a loyalty program. You have an application in the data center that alerts the cashier that the customer whose card has just been scanned at purchase time will get a 20 percent discount on any item in the store on his next purchase. This application not only informs the cashier to mention it to a client, but also it prints it out on the receipt and maybe even sends an email reminder to the customer's registered email.

Compare that to a mobile app that the customer runs as he enters the store or perhaps at home before he gets to the store. If the mobile app looked at what the customer wanted to purchase, it could ascertain that it would give loyalty point value immediately. Because it isn't only tied to the purchase process, it could be used to market other items to the customer that it knows he is interested in buying. It could actually show the customer how buying that item at the same time would be cheaper because it could take advantage of the savings at the time of purchase. The idea is to separate that loyalty program from the purchasing process and be able to tie it to other shopping activities. Now the store can use the loyalty program as a way to sell more items immediately instead of just trying to get the customer to come back to the store later.

Of course it could still be used at purchase time the way it had always been used, but now the functionality can be reused in new and different ways.

This is at the heart of what needs to be built. Smartphones and other mobile devices connect to users in a much more intimate way than even laptops. Have you ever decided against reaching for your laptop in favor of doing something on your smartphone or tablet? Over 2.6 billion smartphones are in use worldwide with the preponderance of users below the age of 55. In fact, as the demographic gets closer to 25, the more highly dependent on smartphones it becomes. These systems of engagement are table stakes for businesses today. Like websites in the 1990s, businesses need to be accessible via smartphone technology. The user community of these systems is sensitive to the usefulness and power of applications and is keenly aware of how quickly it takes advantage of new technologies.

REMEMBER

With this in mind, the focus of systems of engagement is to provide the ability for the creation of new apps that can take advantage of new trends and technologies. While the apps on the phones are often written in languages that fit their operating systems (Objective C or Swift on iPhone and Java for Android), they often rely on another tier to provide specific content. These layers are written in languages like Ruby or JavaScript and developing with Swift, which allow developers to create and deploy new application functionality quickly that connect to users and provide content and capability that are constantly changing. They facilitate rapid development and allow the programmer to control the ability to provide new content in ways that can be augmented with technological capabilities available from the devices. Businesses that can take advantage of this rapid development capability become radically more responsive to their user communities.

Of course these systems of engagement need some support. They normally live much closer to the edge of the enterprise and provide functionality that's focused on the activities that are currently of interest to the user. They don't normally have access to much of the rich business function that's at the heart of most enterprises today. To be truly successful in this space, an enterprise needs to take advantage of the information that is held in the systems that have run the business for the last 50 years. These systems of record have evolved over the course of a business's life and have focused on solving complex business problems with an eye on

sharing data securely in a high-performance platform with high availability.

Because these systems of record have evolved over the last 50 years, they often have components that still survive those early years. In fact, mainframe systems are so sophisticated, they allow programmers to use the latest technologies while still being able to run code that was developed in the 1960s. These systems have been traditionally more focused on providing support for the business processes that have provided business success over the years.


TIP

Systems of record mean more to the business than just a set of data that's securely accessed over the life of the company. They're even more than the transactional benefits that have been the hallmark of their success. The benefit of current z Systems is that they can provide a way for systems of engagement to access information in ways that have been unavailable in the past.

Notice that the discussion isn't around data but information. Information can be created without necessarily exposing any of the secure data required to create the information. For example, I could create an API that determines a patient's financial responsibility for a specific medical procedure without exposing the underlying health care data required to make that decision. Therefore, if a system of record provided an API that took as input a patient name and a particular medical procedure, it could respond with that patient's financial responsibility for the procedure (Jack is responsible for 10 percent of the cost of this ER visit) without exposing to the caller any of the patient's health care information (Jack's account information, previous medical history, other medical treatments, and so on).

Enterprises are focusing on how to take all the business assets that currently exist in their systems of record and free them to make them available on systems of engagement. As they move from the monolithic applications that have grown over the last 50 years into business and technology APIs that can be called, the traditional business models of the enterprise get a new life. After the enterprise breaks up the value chain of business capabilities on the systems of record, the links of that chain can be combined in new ways as different lines of business pick and choose the elements that enable them to reach new clients in new ways using system of engagement and their user centric capabilities.

The value of systems of engagement leveraging systems of record doesn't end with the business being able to relate to user communities better. It can actually sell this functionality as hybrid cloud business APIs to other businesses. Suddenly those niche competitors become value added resellers of the enterprise capability. The enterprise gains a new revenue stream, as IT becomes a new way to earn revenue. Traditional niche providers are able to grow by providing a more complete portfolio of business capability tailored to the community it serves. Users also benefit as services become more personalized and focused on their specific wants and needs.

This link between systems of engagement and systems of record represents a new disruptive technology in cloud. In the first wave of cloud, the focus was on how to make use of low-cost infrastructures to lower the barrier to entry of a business into a market or to lower the operating cost of an existing business or enterprise. This second wave of cloud is much more focused on providing enterprise business capability to any business. It brings a new business model to existing enterprises, allowing them to become managed service providers to other businesses. It allows smaller businesses to take advantage of enterprise capability at a low cost and provide new capabilities to their users. It allows end-users to have unprecedented access to myriad information that was previously unavailable. In the new hybrid cloud world, the mobile device becomes that personal shopper, assistant, and route to business and peers in an increasingly interconnected world.

The DevOps culture allows organizations to make these changes and evolve into providing the business services as required.

Building the Next Generation Enterprise IT Skills

If I look back to when I started in the industry, it was long before the current proliferation of IT technology. There were no cellphones, computers weren't common in the home, and the personal computer hadn't even been released. In those days when new talent was hired to work in IT, they were trained on the companies' languages and practices. When I started in IBM, no one

assumed I knew all the ins and outs of PL/X or 360 assembler. I learned them when I started. Over the years, the assumption that talent could be hired with those existing skills has morphed the development landscape.

In order to deal with the desire to hire people trained with skills in a wide range of areas, the Academic Initiative was started years ago. With the Academic Initiative, IBM and other companies have partnered with universities and colleges to provide an enhanced curriculum that includes topics appropriate for future mainframe developers.

In addition, there are the Master the Mainframe contest and additional gaming contests focusing on either building on z/OS or building front-end interfaces to z/OS back-end applications. These challenges are driving interest among younger and younger students and helping to develop the next generation of mainframe developers.

REMEMBER

Another key factor in the next-generation enterprise IT skills is to remove the separation of tools based on platform. Sometimes an optimized tool for a platform makes sense, such as the IDE or editor that understands the language or platform characteristics. But beyond that, using common modern tooling for the SCM gives new developers from anywhere the ability to find and learn about your code faster; having a modern CI/CD pipeline provides clear visibility and understanding to the new developers; and, finally, using common testing tools and practices allows individuals to come up to speed much faster.

Fit for Purpose Platforms

One additional key advantage of DevOps with its cultural transformation and the breaking down of silos is that platform selection is more easily based on the characteristics of that platform instead of which development team can produce the solution the fastest.

Fit for purpose is the process of selecting the platform based on the actual business requirements for an application or service. For some services, there aren't characteristics that strictly determine the best platform; for others there are. For example, functions

that require high availability and have high transaction throughput are generally seen as more appropriate for z/OS. Additional attributes such as closeness to data and high volume are additional characteristics that might lead to z/OS.

REMEMBER

In a DevOps culture, with everyone focused on delivery, the platform selection can be made based on the actual business requirements, not on team-perceived ability to deliver quickly.

Myth Busters for Mainframe DevOps

Many common misunderstandings exist about what DevOps is and what it implies. In this section, I debunk a few of these myths for you.

DevOps is for born-on-the-web companies

DevOps as a culture is applicable to all organizations. Breaking down the silos helps large-scale multiplatform organizations as much as it does the born-on-the-web companies. The practices of automation are well proven, and those large-scale enterprises that have been transforming have shown the business value received through the transformation.

DevOps implies continuous deployment into production

DevOps culture facilitates the deployment of capability into the end-users for validation, which doesn't imply or require production. Deployment into production is to deliver business value; delivering for the sake of delivery serves as little purpose as holding releases for monthly deployments just for the sake of holding them. The pipeline should be created such that deployments happen to provide the business value.

One great example to use to understand the concept of delivery being a measure of success is that every organization has a calendaring service or API. This service provides critical business value to all business applications. How often does the calendaring service need to change to provide business value? Keep in mind that the Gregorian calendar is still in use.

If you don't recognize the reference to the Gregorian calendar, it's really just the calendar we use today with 12 months, and it's named after Pope Gregory XIII, who introduced the calendar in October 1582. My point is that the calendar hasn't changed in hundreds of years, so there's not a lot of need to change this service.

This service needs to be updated for federal holiday changes, for time zone changes, and to provide new interfaces, such as REST. But other than that, there's no business value to updating the function, so frequent delivery isn't a good measure. Yes, this is an extreme example, but it's important to recognize that delivery for the sake of delivery isn't important; delivery for business value is what needs to be measured.

DevOps and ITIL don't go together

DevOps complements Information Technology Infrastructure Library (ITIL), a set of documented best practices for IT service management. DevOps practices of automation and visibility help support the ITIL practices, specifically the change management practice by decreasing the likelihood of problems with deployment.

DevOps and separation of duties don't mix

One example of separation of duties is the requirement to have someone who didn't work on a change to approve that change to be deployed. This principle can and should continue for production deployments. The approval process is tied into the overall DevOps pipeline. Approval could be based on passing some series of automated tests or based on additional reviews.

DevOps is only for small companies

Large organizations benefit as much as small organizations from a culture of collaboration, automation, and transparency. In fact, large-scale development organizations with many silos benefit most from the transformation. This transformation can bring new focus and life into more stagnant development organizations, returning the innovation and creativity that have been suppressed due to processes.

DevOps is only about automated deployment

It's easy to fall into this trap. After all, most of the DevOps products and services currently available in the marketplace focus on deployment — using software to automate the deployment cycle.

However, if you look at the data, there are five key impediments to realizing the full benefits of DevOps. They are (in order of importance) as follows:

- Environment availability
- Manual deployments
- Test and use case setup
- Governance
- Development practices

Environment availability and configuration cause more delays ("waste" for you lean process buffs) than any of the other four. It is no wonder that organizations that just focus on the automated deployment aspect then run into serious obstacles during implementation.


TIP

So instead of just solving for deployment, solve for the first two impediments first. The two are closely linked together and doing so will release the bottleneck at the end of the software delivery life cycle, bringing you some benefits early and in turn putting pressure to improve on activities earlier in the life cycle.

DevOps is all about improving cycle time

It is true, DevOps is all about speed. Leaders such as Facebook, Netflix, Etsy, and Amazon release new code into production multiple times a day. But what good is speed if it takes an army to deliver bad code? Inherent in DevOps value proposition isn't just speed, but also quality and productivity. In fact, the three are closely linked, and without any one of them, the other two fall apart.

Start with productivity. Most IT leaders get this one; after all, if you're using automation, you're using fewer resources and that frees up time to focus on more value added activities. But quality,

on the other hand, isn't initially apparent. If you dig deeper, you'll find the four dimensions to quality improvement:

- Making small code changes to production dramatically decreases the risk of things breaking. Inserting one new line of code is easier to test rigorously than 100 or 1,000 lines.
- Standardized, repeatable, and reliable processes executed by software bring in a level of rigor and quality improvements that's usually unmatched by manual processing.
- Many defects arise due to a mismatch between the dev/test/prod environments (how often have you heard "It worked on my box"?). DevOps does away with these platform mismatch bugs by automating the configuration so every environment looks the same.
- Due to automation you can ping pong code between dev and test teams frequently to the point where most bugs are fixed, which in turn greatly increases the confidence that the code will work in production.

So, while you may be chasing the sexiness of speed, make sure you keep an eye on the nuts and bolts. If you do it right, you should be able to enjoy all three for the price of one.

DevOps is a tool

This last myth is the worst of all the myths. Many organizations consider DevOps to simply be a tooling exercise — where getting the right tool is the key to the DevOps puzzle and that once it's implemented correctly, the organization will be well on its way for faster turnaround times. Unfortunately, the reality is a bit more complicated.

Here are a few things you may not realize about DevOps:

- DevOps requires not just one tool, but an entire chain of tools. Stitching tools from multiple vendors (open source and paid) is not only painful, but also requires skills and a mindset that's usually foreign to the enterprise.

- For the tools to work effectively, the back-end infrastructure has to change as well. In many cases, this dramatically undoes years of organizational process, protocols, and control.
- Arguably the hardest realization is changing the people and processes to enable a seamless deployment process across multiple silos that traditionally have had diametrically opposite objectives.


TIP

It has been my experience that organizations that follow a tool-led strategy ultimately end up either buying more tools, being stuck or (worse) frustrated, and throwing in the towel. Always keep in mind DevOps is a cultural transformation; tools are to support this.

Notes

Notes

Build a DevOps culture

DevOps is a cultural change to an organization that applies to companies large and small. In today's era of digital disruption, enterprises need to increase the pace of innovation while ensuring the stability of core business systems. DevOps can provide an integrated multi-platform roadmap to increased growth through the cloud and API ecosystems.

Inside...

- DevOps best practices for z Systems
- DevOps success stories
- Applying open-source tools to z Systems
- Building for the digital economy

Rosalind Radcliffe is an IBM Distinguished Engineer, Chief Architect for DevOps for Enterprise Systems. She is responsible for driving product direction, working with the internal teams, and working with clients on their DevOps journeys. In her 30 years at IBM, she has worked in the different areas of Dev and Ops.

ISBN: 978-1-119-41023-2
Print part #: ZSM03026USEN-00
ePDF part #: ZSM03027USEN-00

WILEY END USER LICENSE AGREEMENT

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