Deloitte.

IBM Platinum Business Partner

Mainframe agility

A Deloitte DevOps experience

Contents

Foreword

It's no secret that mainframe release processes tend to be very complex, manual, and cost inefficient ordeals, taking hours and even days to complete. In most cases development and deployment processes have not changed for decades. As organizations with mainframe assets and infrastructure move towards becoming truly agile, they need to tackle the challenge of manual software development and deployment, as well as rising MIPS (Millions of Instructions per Second) costs associated with automation testing on target hardware.

The challenge? Mainframe applications lack modularity, which makes the development process painfully slow. Even small code changes can be risky when a mainframe infrastructure is running live.

Now a new stack of modernized IBM® z Systems® mainframe DevOps tools offer increased speed of continuous integration (CI) and continuous deployment (CD) on mainframe assets. The tools improve software code quality through shift-left testing. An automated delivery pipeline accommodates more frequent automated testing, which allows the code to go through more change iterations.

DevOps can also lower costs because of fewer quality problems and software rollbacks. Ensuring new feature development can happen as smoothly and quickly on the backend system as on the frontend, giving organizations the advantage they need to be responsive to their respective markets.

Introduction

The IBM relationship with mainframes started in 1952 with the IBM 701 designed specifically for the Department of Defense. Today, mainframes remain a central figure in the modern business world. Despite developers continuously migrating towards cloud-based architecture, mainframes are still leveraged by 92 of the top 100 banks worldwide, 23 of the top 25 US retailers, all 10 of the world's 10 largest insurers, and 23 of the world's 25 largest airlines [1]. Companies in every industry segment are deploying new mobile apps that have mainframes as the middleware and the backend. Why?

Mainframes are built specifically for mission-critical applications, such as processing banking transactions, where both reliability and security are paramount. The systems are designed with redundancies and backups built in. If a component fails, it can be swapped out without affecting the system. This innovative model has garnered impressive measures: IBM's mainframes have a mean time between failures measured in decades.

Mobile and web 2.0 applications are redefining how value is delivered to customers. They've transformed the industry ecosystem. And, they are constantly evolving at a faster clip than IT has ever experienced before. These applications require a fast and nimble deployment environment to respond to fast-paced business demands. Since many of these web/mobile applications integrate with existing CICS® and IMS™ backend programs, there is a need to step out of the waterfall school of thought of time-consuming releases and go with a more agile approach, seamlessly integrating mainframe applications with newly developed software code in the mobile/web and cloud domain.

"There are 6,900 tweets, 30,000 Facebook likes and 60,000 Google searches per second. The mainframe CICS runs 1.1m transactions per second, which equates to 10bn per day."

Rob Lambs
IBM Hursley Laboratory Director [2]

Agile innovation loop

In a traditional waterfall model, teams of architects and developers analyze, design, code, test, and deploy in a sequential manner via releases. It takes too long. In contrast, agile development (as shown in Figure 1) is performed through a series of adaptive iterations by analyzing, designing, developing, and testing feature sets almost in real-time. It happens at a more granular level, module by module, feature by feature, within each cycle. During an iteration, it is imperative that both internal and external stakeholders provide feedback and ensure the features meet their needs. The major advantages of the agile innovation loop are:

- Faster feedback cycle
- Early problem identification
- Flexible prioritization
- High potential for customer satisfaction

With an agile innovation loop for mainframe software and solutions development, just imagine the outcomes: collaborative cross-functional teams, early delivery, frequent releases, rapid and flexible response to change, and continuous improvement. To achieve such high performance will likely require a transformation that isn't free of risks or challenges. The first step to minimizing potential issues is identification.

Accelerating delivery

Figure 1: Agile innovation loop

Today's mainframe development challenges

As with any transformation, the rewards must outweigh the risks for the initiative to make sense. Understanding the obstacles leads to better decisions and, ultimately, to better outcomes. Here are a few challenges with today's mainframe development.

Friction between development layers

As organizations try to rapidly build and expand new software solutions that tie back-office (system of records apps) to front-office (system of engagement apps) they naturally incur friction. Different approaches, processes, and even technologies exist in these two different layers of software development and the frequency of releases across these tiered systems is introducing new challenges. The rapid development and deployment of top layer web and mobile apps doesn't always align with the slower moving development of bottom layer mainframe applications. Mainframe app changes simply aren't keeping up with the fast pace of web and mobile app development change.

Attracting talent

The original mainframe generation is slowly retiring. With universities rarely teaching mainframes and a steep learning curve to contend with (three to five years to learn basics of mainframe development), finding the right people with the right experience can be difficult. There is an increasing need in mainframe development environments to shift towards the use of more modern tools that are attractive to a new generation of developers if organizations are going to sustain or increase current rates of development.

Lost code

Lack of proper source code management during workforce transitions can result in mission critical binaries in production. Without an accompanying codebase, making changes is nearly impossible.

Lack of modularity

By nature, mainframe applications lack modularity, which makes the development process slow and small code changes risky. Mainframe release processes are also very complex, manual, and cost inefficient, often taking hours or days to complete.

Continuous editing

Due to the long shelf life of these systems, code has been refactored and migrated numerous times, leading to poor quality and even dead code.

Lack of automation

Poor documentation and lack of modularity in existing codebase makes testing and quality assurance an uphill task.

MIPS costs

Mainframe performance is measured in MIPS, an older measure of computing power and performance. Organizations with mainframe assets and infrastructure need to tackle the challenge of rising MIPS costs associated with automation testing on target hardware if they want to become more agile. This is especially important considering that mainframes continue to be the tool of choice for executing high volume application transactions in a robust and reliable fashion.

Mainframe DevOps

Despite these challenges, mainframe software applications shouldn't be thought of differently than frontend development. The same methodologies that are applied to speed up facilitated software development and deployment on the cloud can be applied to mainframe software development and deployment. What are the best practices developers consistently define?

- Automate everything
- Track and plan everything
- Version control everything
- Test everything
- Dashboard and monitor everything

As a software delivery methodology and practice, an agile innovation loop using DevOps tools for mainframes can help bring frontend and backend IT operation teams together along with testing and QA teams, project management, and even customers in some cases. It enables smoother development and faster work while maintaining quality and integrity. As a start, a basic minimum viable DevOps pipeline, like the one shown in Figure 2, can drastically improve collaboration. More advanced and mature DevOps pipelines can also enable continuous monitoring and incorporate constant feedback from all stakeholders including customers.

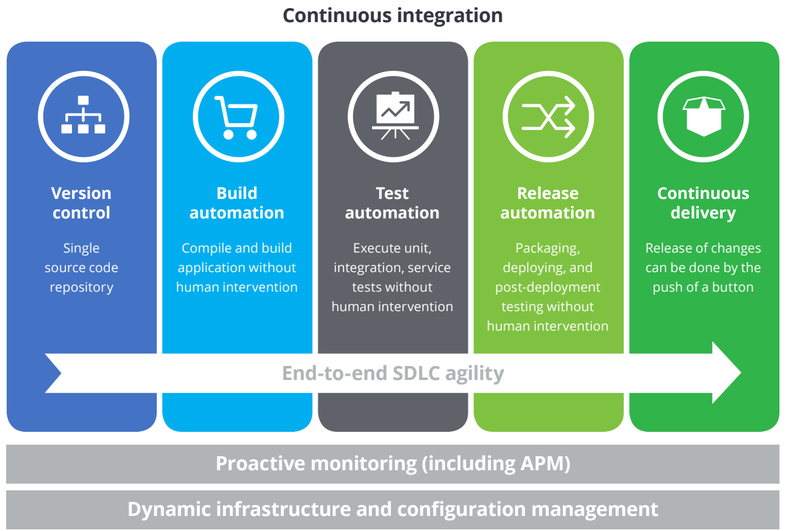

Figure 2: DevOps pipeline for end-to-end software development pipeline

DevOps for z Systems mainframes

Organizations have the potential to achieve a better and more agile way of mainframe software development and deployment—one that increases collaboration and automation by using a DevOps pipeline and tools. To test this point-of-view, Deloitte evaluated IBM's DevOps capabilities for z/OS® Mainframe Systems against today's best DevOps practices.

A stack of modernized DevOps pipeline and tools products by IBM for z Systems mainframes were installed and configured in a sandbox environment. First, a brief assessment of the following IBM software suite of products was performed:

- IDz (IBM Developer for z Systems), an IDE based on Eclipse that supports z/OS systems software development.
- IBM's RTC (Rational Team Concert™), essentially a team orchestration and collaboration tool with advanced features for workflow assignment and tracking as well as debugging and building automation.
- zUnit Test, a suite of capabilities within RTC that enables automated unit testing.
- UCD (Urban Code Deploy), a package management tool that enables fast and reliable code and solution deployment.

Table 1 shows a mapping of these tools against common best practice DevOps capabilities:

IBM DevOps offerings for z Systems Mainframes vs. core DevOps capabilities	IDz	RTC	zUnit	UCD
Code development	X		.	
Collaboration	X	X		
Build automation	X	X		
Continuous integration/development	X	X		
Continuous testing/test automation		X	X	
Continuous deployment/delivery				X
Continuous monitoring				?

IBM IDE and RTC work together in these cases as RTC module is installed on top of IDz

Table 1: IBM DevOps offerings for z Systems mainframes vs. core DevOps capabilities

After studying and analyzing the capabilities of the mentioned tools and their potentials, an attempt was made to integrate each silo component into a basic DevOps pipeline (see Figure 3) to evaluate the overall benefits of such pipeline for a z/OS mainframe ecosystem. Figure 10 in the appendix section shows a high level architectural overview of the DevOps tools integrated with a z/OS system and its peripherals such as DB2®, CICS® application code, COBOL code, ISPF, and UCD agents for continuous code integration and code deployment. Here's what was assessed.

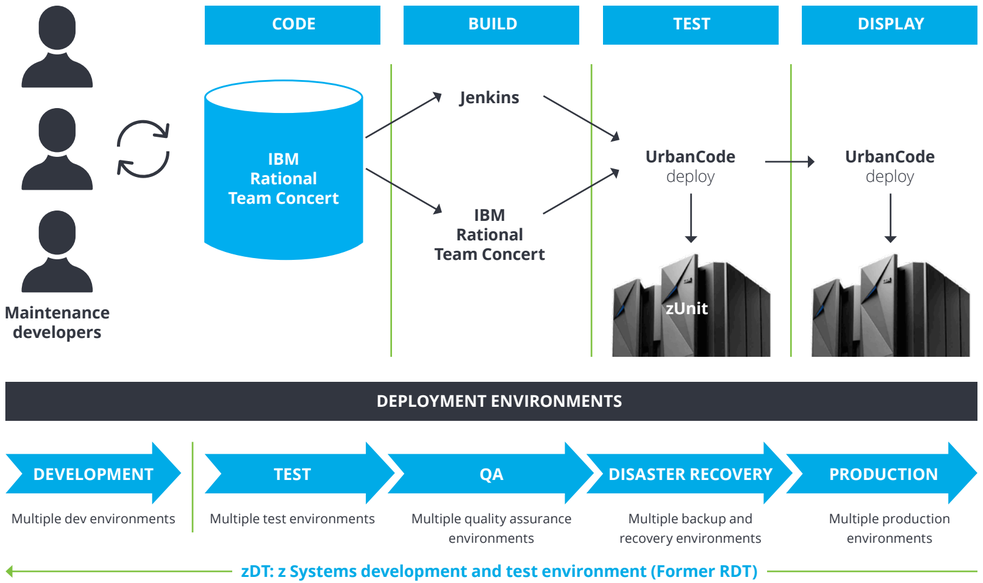

Figure 3: Proposed DevOps pipeline for software solution delivery on z Systems mainframes

Plan

For decades, mainframe application developers have mostly worked in waterfall environments with blue/green screen access terminals. Mainframes are core systems that handle major business transactions for organizations. Embrace them! Investment in mainframes, be it the people, process, or technology, is key to sustainable growth. Make the shift from a waterfall environment to an agile innovation loop environment to speed up and invigorate the process.

To enable such a cultural change, leaders can start by restructuring mainframe teams to include select mainframe developers and pair them with agile, experienced developers from other platforms. Doing so will not only help promote an agile working mindset with the mainframe development team, but also enable other developers to understand the aging mainframe code base. With organizations currently dependent only on a few developers for their continuous operations, this transfer of knowledge can hold the key to future high performance as mainframe developers begin to retire.

As the mainframe teams move toward releasing software and code in more frequent iterations, it will be important to understand their current development and release process and challenge routines that may slow them down. Each organization should establish a standardized application delivery process for both modern distributed teams and legacy mainframe teams.

Code

Sometimes experienced developers get comfortable using ISPF "green screen" editors and resist change, ignoring the productivity and quality gains of modern IDEs in finding defects earlier and debugging code once defects are found. In addition, organizations willing to invest in customer-facing application improvements often resist using resources to modernize legacy applications. The challenge is that traditional mainframe application development lifecycle tools were not designed for small, fast, automated deployments. When considering the current market conditions and industry disruption, keeping the status quo becomes unsustainable.

Modern IDEs offer faster feedback and debugging tools where older applications such as ISPF/PDF (Interactive System Productivity Facility/Program Development Facility), present significant barriers to getting the next generation of developers to work on mainframe application code. Familiar tools with integrated debuggers, syntax-directed editors, and interfaces make picking up COBOL or PL/1 relatively easy.

Applying modular coding and architectural practices and refactoring mainframe applications into smaller independent components with managed APIs helps organizations improve the speed at which they deliver. Keep in mind, not every deployment would need all the modules every single time.

Next, the tools and capabilities were examined including:

- IDE capabilities
- Team orchestration and build automation capabilities
- RTC-build vs JCL-build capabilities
- RTC-build automation
- RTC's Jenkins plugin

IDE capabilities

IBM Developer for z Systems (IDz) is a common Eclipse-based IDE that provides standardized IDE features for all types of development, including the z/OS software development. IBM has add-ons that allow product managers, developers, and testers to work on COBOL code and JCL scripts, while allowing users to enjoy a variety of features that increase efficiency and productivity. These include:

- Smart code coverage, code governance
- Unit test support
- Visual control flow analysis for both data flow and program flow analysis
- Code review and work flow metrics
- ADFz Fault Analyzer, reports, and IDz integration
- IBM Application Performance Analyzer

Team orchestration and build automation capabilities

Rational Team Concert (RTC) is a team orchestration tool that is loaded in IDz, enabling project management workflow, teamwork tracking or orchestration, and supports version control or source code management.

Embedded features within RTC also allow code review in the work item flow. In addition, it provides tools for facilitated code debugging and helps direct build requests either as a personal or project build—opposed to the more traditional way of building COBOL code through JCL scripts loaded on the z/OS mainframe along with the COBOL source code.

RTC-build vs. JCL-build capabilities

Traditionally a Job Control Language (JCL) is a scripting language used on IBM mainframe operating systems (z/OS) to instruct the system on how to run a batch job or start a subsystem. One such type of batch jobs is instructing the z/OS to create load and binary files after compiling COBOL application code. However, RTC has made it easier to make build requests directly through its feature set by creating and configuring JCL instructions under the hood without the need for the developers to worry about the build process. And, if a developer or architect chooses to create their own JCL instruction set, it is possible to submit customized JCL jobs.

RTC-build automation (scheduling)

RTC-build capabilities are not limited to personal locally invoked build requests. They also include scheduling mechanisms that allow stakeholders to plan and organize a series of automated build requests at pre-planned times. They do this by changing the requirements of a build definition inside a build configuration file on a z/OS mainframe system. Other open source auxiliary orchestration and automation tools such as Jenkins can further enhance RTC-build automation.

RTC's Jenkins plugin

IBM also provides an RTC plugin that allows the integration of the popular open source CI (Continuous Integration) and CD (Continuous Deployment) tool, Jenkins. RTC already provides build automation through one-click build requests: personal build for developers on local machines through coordination with z/OS mainframe, as well as project build requests for the entire build stream. RTC build allows automation by scheduling the builds in the build definition file. The feature allows developers to configure their own builds using a build engine of their choice. Jenkins can also be connected to test automation and orchestration as well as automated deployments through communication with Urban Code Deploy. See Figure 4.

What are some benefits of Jenkins?

- Over 100 Jenkins plugins are available
- Integration with over 100 DevOps tools
- Orchestration of the DevOps toolchain
- End-to-end CD-pipeline management

The main advantage of incorporating an automation and orchestration engine like Jenkins is that it opens up the gates of a DevOps pipeline to an additional arsenal of modern DevOps tools. See Figure 5.

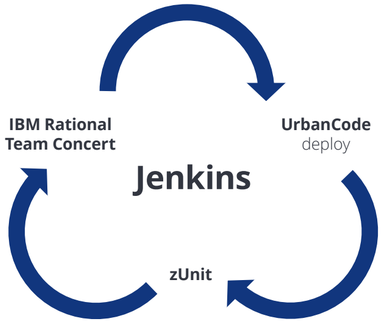

Figure 4: Jenkins used as an orchestration engine can further enhance IBM RTC, zUnit, and UCD capabilities.

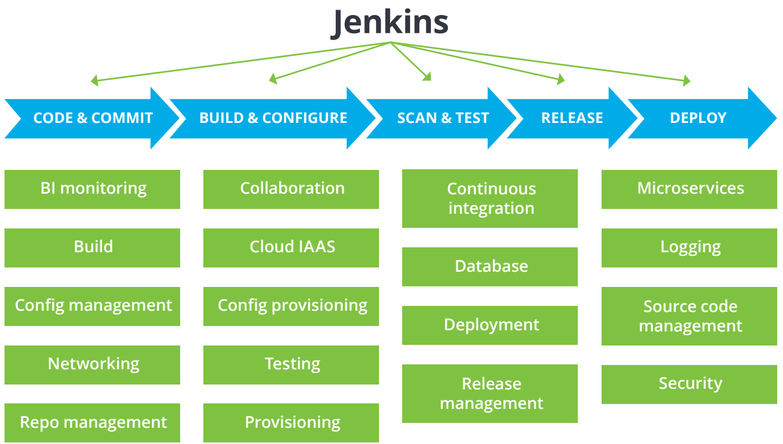

Figure 5: Jenkins can open the doors of possibilities to include other DevOps tools.

Test

In general, there are few z/OS test environments that are shared across application development teams, so there has been very little automated testing of applications and development projects. Automation plays a key role in shortening the development lifecycle and ensuring a better quality product. It also identifies validation tests earlier in the development lifecycle, helping developers to better filter out and fix the bugs during the development process. Based on new findings, implementing granular test modules for each new feature or feature modification early in the process can save on cost overruns. Take a look at Figure 6.

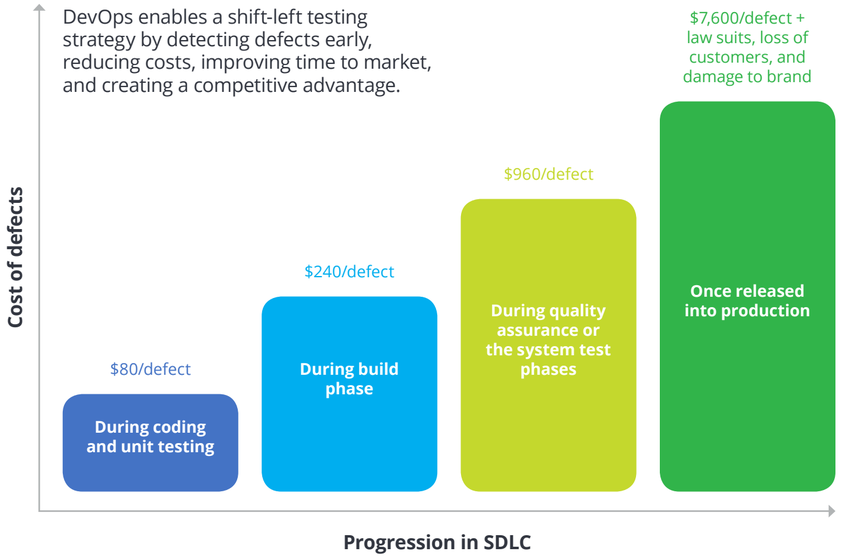

Figure 6: National Institute of Standards & Technology (Source: GBS Industry standard study)

Unit testing and automated testing capabilities

zUnit test automation is another piece of the IBM DevOps tool stack. It provides unit testing capabilities for different modules of a mainframe application. However, the tests themselves and supporting code should be written in COBOL code using a black box test methodology.

Developers can create a test function for each feature that adds in the COBOL application, comparing the outcome of the unit with the desired or pre-measured outcome, hence the name. The zUnit component provides all the necessary tools to do black box testing right along with other layers of testing shown in Figure 7.

Keep in mind, the test automation and unit testing can be added as a separate job in Jenkins build automation. Now, let's move on to deployment.

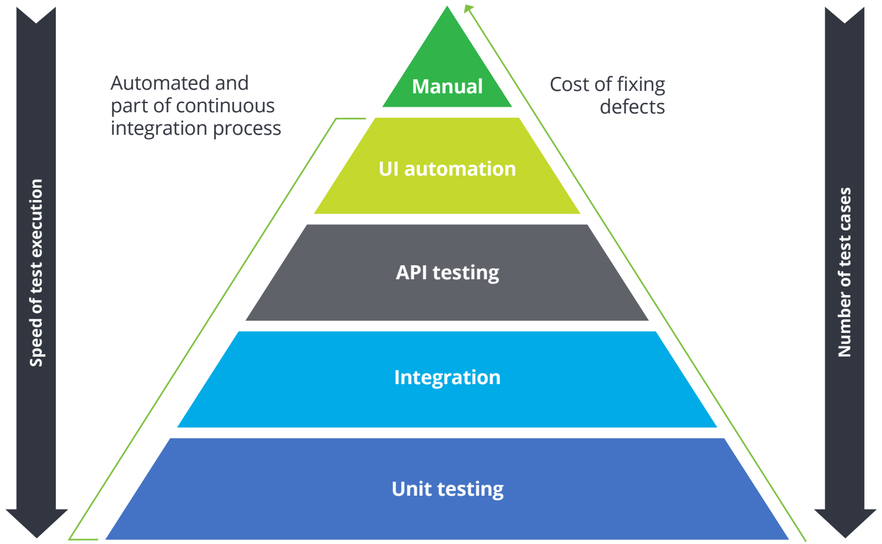

Figure 7: Agile test automation pyramid

Deploy

As continuous integration produces builds at regular intervals, the artifacts can be deployed to a variety of environments, such as SIT, UAT, and Stage, and pass through appropriate quality gates all the way to production. Continuous delivery facilitates this by automating the delivery pipeline. Highly mature organizations have automated deployment capabilities, enabling them to push changes as they come in as opposed to bundling up changes due to strenuous manual deployment processes. Environment and security configurations are standardized to reduce risks and vulnerabilities in the deployment process.

In the intensely shared nature of the z/OS environment, deployment processes are highly structured, requiring all changes to follow a common process. With a deployment capability locked in a system that does not coordinate well with other environments, a comprehensive and automated deployment solution can be difficult to achieve.

UrbanCode Deploy is a multi-tier deployment automation tool for z Systems that offers out-of-the-box integrations with Rational Team Concert and pushes artifacts out upon building. An application comprised of UCD components work together to provide a business function. Applications include application processes, which deploy the components. Applications also contain environments, which represent the target systems that the application components are deployed on.

A standardized deployment process consists of a package detailing the group of deployable artifacts as defined by the build engine's ship list. It is relayed through UCD Agents running on an all-target machine where the package is to be deployed. A deployment process set consists of component processes deploying UCD components, application processes deploying collections of UCD components, and generic processes running automated tasks on target systems. See Figure 8.

Figure 8: Deployment after packaging on test, stage, and production environments

Additional possibilities and hybrid models

Implementing DevOps on mainframes provides a platform to catch up with the fast-moving distributed applications hosted on the cloud. However, mainframes with their tightly coupled monolithic architecture and dependencies across programs still pose a significant challenge for a fast risk-free and efficient release process. On any given day, a developer making a single change in any given program can have multiple impacts across the application. This intertwined nature of code slows down the development process and makes deployment to production a very risky process.

Creating modular applications that can be modified, built, and released independently becomes increasingly important. Interdependent COBOL programs need to be refactored into smaller individual components, essentially representing themselves as a micro service wrapped around APIs to communicate with other programs in the application.

As refactoring large complex mainframe applications can be a challenging process, organizations should target the applications that require constant changes, have large dependencies, or cause issues when deployed to production.

Additional possibilities are available to organizations with mainframe assets and can help them better manage technical debt and work in progress. Take for example an even containerization of monolithic mainframe applications with the use of modern containerization solutions, such as Docker. Another possibility? Combine COBOL z/OS application development using more modern procedural environment suites. Do it using RESTFUL APIs or Java APIs, which can help bring additional talent into the landscape of mainframe application modernization and development.

Along with the focus on software production modernization for mainframes, it may make sense to encompass the use of two or more additional solutions: 1) containerization to further enhance and maximize more rapid scale up and 2) continuous mainframe solution development and deployment. Containerization of COBOL applications can provide the ability to rapidly scale up the mainframe infrastructure. (Shown in Figure 9.)

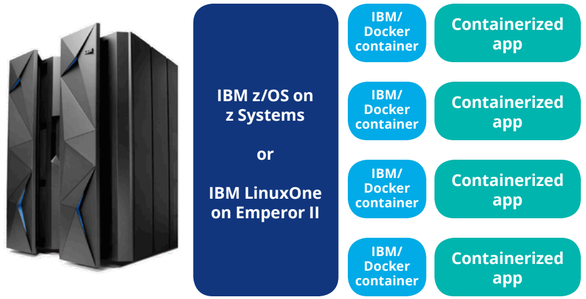

Figure 9: Containerization example on IBM Mainframes (z/OS & LinuxOne) with Docker containers

Areas of concern

Given the current challenges with mainframe development and its widespread use in various industries, DevOps isn't just an improvement, but a necessity for organizations if they are going to maintain a competitive advantage. However, it's important to consider possible areas of concern as well, such as:

- **Cultural shift.** Moving towards continuous delivery can be daunting for mainframe shops that have focused on application stability and hyper-rigorous change management, rather than on faster iterative delivery and rapid response. Coaching teams in agile methodologies and concepts can help them make the shift.
- **Archaic builds.** Due to the large size of the application, developers generally build only the changes. It is common to find parts of an application, which have not been rebuilt in decades, and that represents a challenge. Modularization of the applications can significantly mitigate this problem.
- **Point of diminishing returns.** As more and more complexity is added to a DevOps pipeline, DevOps teams should consider the costs. Only adopt additional agile and DevOps processes and technologies to the point that they don't result in diminishing returns on investment of time and resources.

Only after carefully examining and considering such areas of risks on a case-by-case basis, can an organization decide on a DevOps pipeline that includes processes and procedures to address above calculated areas of concern.

Mainframes aren't going away any time soon. A new stack of modernized IBM z Systems mainframe DevOps tools offer increased speed and an agile innovation loop of continuous integration and deployment on mainframe assets. The tools improve software code quality through both shift-left and automated testing. Finally, there's a swift, modern way of development for legacy, backend systems.

References

1. R. Radcliffe, "Mobile to Mainframe DevOps for Dummies," IBM, 2015, pp. 09-10.
2. "Digital Transformation Needs Mainframe DevOps," Forrester Research, June 20, 2016.

Appendix

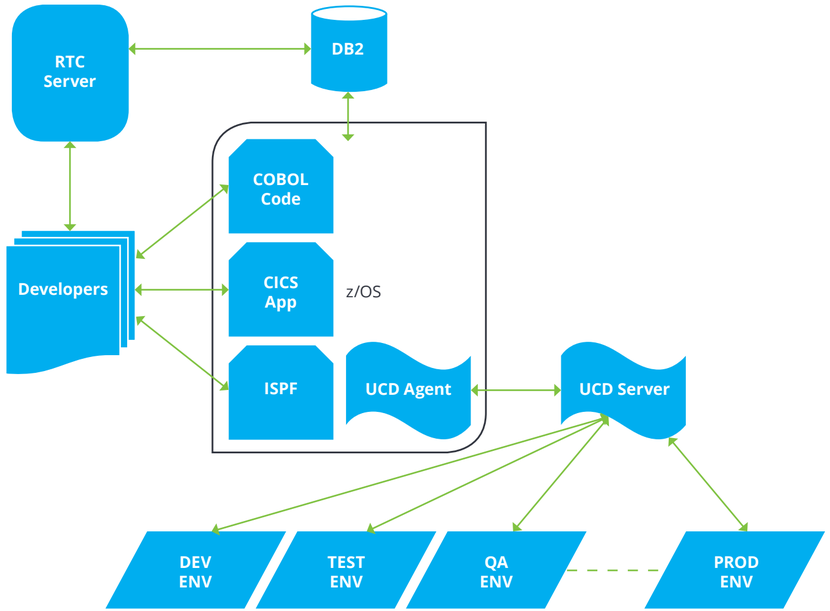

Figure 10: Internal high-level architectural view of integration of z/OS systems DevOps tools with a z/OS mainframe system and its peripherals

Notes

Authors

Arthur Papian Gorji
Digital Integration/DevOps Senior Consultant
Deloitte Inc.
apapiangorji@deloitte.ca

Prem Suresh Kumar
Digital Integration/DevOps Consultant
Deloitte Inc.
premxkumar@deloitte.ca

Contributors

Robert Miller
IBM Alliance Solution Architect
DC Specialist Leader, Deloitte Consulting —Technology Alliances
robmiller@deloitte.com

Deloitte.