Mainframe agility
A Deloitte DevOps experience
Mainframe agility

Introduction

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.

Foreword
Mainframe agility | Introduction

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 continuing to migrate towards cloud-based architectures, mainframes are still leveraged by many 100 banks worldwide, 23 of the top 25 US retailers, all 10 of the world’s 10 largest insurers, 23 of the world’s 25 largest airlines, and 20 of the top 25 US retailers. They’ve transformed the industry ecosystem. And, they are constantly evolving at a faster clip than IT has ever experienced before. These mainframes are built specifically for mission-critical applications with newly developed software code in the mobile/web and cloud domain.

Mainframes are built specifically for mission-critical applications, such as processing banking transactions, ensuring both reliability and security are paramount. The systems are designed with redundancies and failover capabilities, so they are continuously available 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:

Figure 1) is performed through a series of adaptive iterations by analyzing, designing, developing, and testing features almost simultaneously, breaking down the features into small, actionable pieces. Teams continuously optimized for both development and production. Operational and security aspects are systematically induced to resiliency. Regular ad-hoc testing is performed to ensure continuous improvement. Development is frequent and automated. Deployments are performed to reflect client feedback.

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 changing business demands. Agile and production failures are rejected if sanity fails. Teams have access to production-like data for testing and can condition the test data within the pipeline. Operational and security aspects are systematically induced to resilience.

Agile innovation loop

In a traditionally waterfall-based model, teams of developers and architects analyze, design, code, test, and deploy in a sequential manner. Development 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 features almost simultaneously, breaking down the features into small, actionable pieces. Teams continuously optimized for both development and production. Operational and security aspects are systematically induced to resiliency.

“Tigers in the Cove” is a view of high performance and ultimate scalability. Tiger 1 is the production environment. Tiger 2 is the staging environment. Tiger 3 is the development environment. Tiger 1 reinitializes the data from Tiger 2 to provide for high performance and ultimate scalability. Teams have access to production-like data for testing and can condition the test data within the pipeline. Application features are quickly released and deployed.

Rob Lambs
IBM Hursley Laboratory Director [2]

Accelerating delivery

<table>
<thead>
<tr>
<th>Plan</th>
<th>Develop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teams are small, multi-functional, focusing on end-to-end delivery</td>
<td>Application features are sized to small batches (less than a week) to allow for minimum viable product (MVP) and rapid development.</td>
</tr>
<tr>
<td>Product backlog is frequently updated</td>
<td>Following micro-based practices by frequent code commits for better stability and higher throughput.</td>
</tr>
<tr>
<td>Teams continuously optimized for both teams and individual productivity</td>
<td>Developers get daily feedback from acceptance, performance, and integration tests.</td>
</tr>
</tbody>
</table>

Figure 1: Agile innovation loop

Deploy

- Code commits trigger automated delivery and unit tests to test the validity of the branch (code commit is rejected if sanity fails). Teams have access to production-like data for testing and can condition the test data within the pipeline. Regular ad-hoc testing is performed to ensure continuous improvement.

Monitor

- Dashboard statistics are readily available to the team to enable continuous monitoring.
- Automated daily security checks are run to ensure overall system health.
- Continuous logging is enabled and available to developers for debugging.
- Periodic snapshots and backups are done to ensure advanced rollback capability.
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 front-office (system of engagement apps) to back-office (system of records apps) to expand new software solutions that tie front-office (system of engagement apps) to back-office (system of records apps), they naturally incur friction. Different approaches, processes, and even technologies exist in these two different approaches, processes, and even technologies exist in these two different layers of software development and the frequency of releases across these tiered layers of software development is introducing new challenges. The friction between development layers leads to better decisions and, ultimately, to better outcomes. Here are a few challenges with today's mainframe development.

- **Lack of modularity**
  - Making changes is nearly impossible.
  - Without an accompanying codebase, making changes is nearly impossible.
- **Lack of proper source code management**
  - Lost code
  - Lack of modularity
- **MIPS costs**
  - Mainframe performance is measured in MIPS, an older measure of computing power and performance. Organizations need to tackle the challenge of rising MIPS costs associated with automation testing and deployment on the cloud.

Despite these challenges, mainframe software applications shouldn't be thought of differently than frontend development. The same methodologies that are applied to software development and deployment on the cloud can be applied to mainframe software development and deployment.

- **Continuous integration**
  - DevOps tools for mainframes can help. As a software delivery methodology and practice, an agile innovation loop using continuous integration, service testing, and test automation can drastically improve collaboration. More and more teams 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 DevOps pipeline, like the one shown in Figure 2, can drastically improve collaboration. More advanced and mature DevOps pipelines can drive more continuous monitoring and incorporate constant feedback from all stakeholders including customers.

- **Continuous delivery**
  - Release of changes can be done by the push of a button.
  - Continuous delivery allows for more reliable fashion.
  - The same methodologies that are applied to software development and deployment on the cloud can be applied to mainframe software development and deployment.

- **Continuous monitoring**
  - Proactive monitoring (including APM) can drastically improve collaboration. More and more teams 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 DevOps pipeline, like the one shown in Figure 2, can drastically improve collaboration. More advanced and mature DevOps pipelines can drive more continuous monitoring and incorporate constant feedback from all stakeholders including customers.

- **Continuous deployment**
  - DevOps tools for mainframes can help. As a software delivery methodology and practice, an agile innovation loop using continuous integration, service testing, and test automation can drastically improve collaboration. More and more teams 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 DevOps pipeline, like the one shown in Figure 2, can drastically improve collaboration. More advanced and mature DevOps pipelines can drive more continuous monitoring and incorporate constant feedback from all stakeholders including customers.

- **Dynamic infrastructure and configuration management**
  - Dynamic infrastructure and configuration management is a crucial aspect of mainframe software development and deployment.

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.

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®, code, SRF, and UCD agents for continuous code integration and code deployment. Here’s what was assessed.

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

<table>
<thead>
<tr>
<th>IBM DevOps offerings for z Systems Mainframes vs. core DevOps capabilities</th>
<th>IDz</th>
<th>RTC</th>
<th>zUnit</th>
<th>UCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code development</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collaboration</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build automation</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continuous integration/deployment</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continuous testing/test automation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continuous deployment/delivery</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Continuous monitoring</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

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 SPS “green screen” editors and resist changing 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 making picking up COBOL or PL/I 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.

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 added the ability to 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 workflows, teams to build or orchestrate, 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 job is instructing the z/OS to create load and binary files after compiling COBOL application code. However, JCL 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 custom JCL jobs.

RTC-build automation (scheduling)

RTC-build capabilities are not limited to personally local 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. These tools 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 build in the build definition file. The feature allows developers to configure their own build in a build request. 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 lifecycle
- End-to-end CI/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.

Jenkins can open the doors of possibilities to include other DevOps tools.
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 by 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.

DevOps enables a shift-left testing strategy by detecting defects early, reducing costs, improving time to market, and creating a competitive advantage.

Unit testing and automated testing capabilities

Unit 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.

Figure 7: Agile test automation pyramid

Cost of defects

$7600/defect + law suits, loss of customers, and damage to brand

$960/defect

$240/defect

$80/defect

Progression in SDLC

During coding and unit testing

During build phase

During quality assurance or the system test phases

Once released into production

Number of test cases

Speed of test execution

Cost of fixing defects

Manual

UI automation

API testing

Integration

Unit testing

Automated and part of continuous integration process

Figure 6: National Institute of Standards & Technology (Source: GBS Industry standard study)
**Mainframe agility | DevOps for z Systems mainframes**

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

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

**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 a ripple effect across the application.

This intertwined nature of code slows down the development process and makes deployment to production a very risky process. This intertwined nature of code slows down the development process and makes deployment to production a very risky process. This intertwined nature of code slows down the development process and makes deployment to production a very risky process. This intertwined nature of code slows down the development process and makes deployment to production a very risky process. This intertwined nature of code slows down the development process and makes deployment to production a very risky process. This intertwined nature of code slows down the development process and makes deployment to production a very risky process. 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 microservice 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, implementing modernization for mainframes. It may make sense to encompass the use of two or more modernization solutions, such as Docker. Another possibility? Combine COBOL, Java, or Java APIs with Docker containers.

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).
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 add 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**


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