DevOps for the Mainframe

Rosalind Radcliffe
IBM Distinguished Engineer, Enterprise Modernization Solution Architect
rradclif@us.ibm.com

Smarter software for a smarter planet

Enabling Product and Service Innovation | Rational
Please note

IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.
Challenges delaying delivery of functionality to the business

Customers
Desire for fast and continuous innovation

Line of Business
Requirements

Dev & Test Teams
Code & Tests

Operations Team
Business Services

1st Gap
Addressed by...

2nd Gap
Addressed by...

Agile Dev

Dev Ops
Software Delivery Challenges: what we hear from customers

Needs:
- Reduce cycle time and delays
- Improving software delivery efficiencies with standardization and automation
- Improving Quality of Delivery and reducing roll-backs

Takes Weeks/Months to deliver a change

Quality Challenges
- Difficulties in reproducing production defects
- Long time to fix defects
- Poor Test coverage
- Lack of automated testing

Release Challenges
- Differences in Dev/Ops environments
- Siloed / Limited automation
- Long set up time

Process and Cultural challenges
- Point-Point, adhoc and Fragile integration of tools
- Poor visibility, stability and extensibility
- Cultural barriers limiting collaboration
- Heavy-handed control of dev environments

Simplified view of a single-release pipeline. In general, there are multiple projects, releases, and technologies at play.
Current Customer approaches addressing these challenges:

- Selective & Siloed automation of the delivery process with limited benefits
- Poor visibility and control impacting cycle time
- Coordinated automated and manual processes

**Build Automation**

**Promotion controls**

**Manual Testing**

**Deployment Automation**

- Middleware Deployment Automation
- Env Config Automation
- Infrastructure Deployment Automation
Need for a Simple approach to bringing agility across the lifecycle
Continuous, orchestrated, and automated delivery of changes leveraging Cloud

- Long Cycle Time
- Poor Visibility and Control

- Reduced cycle time
- Control, Agility and scalability
Time is now for DevOps

*Trends accelerating the need for Continuous Delivery*
DevOps: Principles & Values

- Collaborate across disciplines
- Develop and test against a production-like system
- Deploy frequently
- Continuously validate operational quality characteristics

Results in:

- Rapid evolution of deployed business services
- Reduced risk
- Decreased cost
- Improved quality across the portfolio
How do we make this happen?

- Automate *everything*
- Version *everything*
- Test *everything*
- Track and Plan *everything*
- Instrument and Audit *everything*
- Dashboard *everything*
Continuous Delivery Reference Architecture

Built on open standards allowing plug-in components from IBM products, open source, and third party

Continuous Delivery Services

Open Services Lifecycle for Collaboration
- Delivery Dashboard
- Delivery Pipeline Services
- Instrumentation & Auditing
- Automation Engine
- Library Services
- Reporting & Analytics

Integration Fabric using OSLC
- Test Automation
- Build Automation
- Deployment Automation
- Change Management
- Cloud-based Provisioning

Foundation Services

Rules  Logging  Licensing  Security  Storage
Mainframe DevOps Adoption Strategy

1. Adopt a test-everything strategy using continuous integration build approach with (automated) self-validating builds [Automate and Test]

2. Unify mainframe application asset change management to ease orchestration [Version, Track, and Plan]

3. Document standardized environments and drive cloud provisioning of isolated mainframe images and applications based on standards [Automate, Instrument, and Audit]

4. Optimize provisioning orchestration to include dependency virtualization and test data conditioning
Test Everything:
Continuous, automatic testing across the lifecycle

Did the app compile?
Did unit tests pass?
Did the app get packaged OK?

Did deployment succeed?
Did BVT/Smoke test succeed?

Did system tests succeed?
Are performance tests succeeding?
Are there any new errors in the logs?
Are transactions succeeding?
Solution: Continuous Integration
Reduced delivery time, end-to-end visibility of test activities, safer and faster upgrades (V2V)

- Fast, dependable, automatic feedback speeds time to market
- Lower cost of application testing using off-mainframe z/OS test environment
- Enables confidence by automatically tracking and promoting code health
Detailed Continuous Integration for System z Scenario

1. Check-in code
2. Build code and zUnit tests
3. Deploy build results to Test Environment
4. Execute zUnit Tests
5. Kick-off Automated Test Plan
6. Run automated interface tests against Test Environment
7. Mark execution records Pass/Fail in Test Execution Manager
8. Report test results in dashboard/build results/defect records in CI server.
Detailed Continuous Integration for System z Scenario

1. Check-in code
2. Build code and zUnit tests
3. Deploy build results to Test Environment
4. Execute zUnit Tests
5. Kick-off Automated Test Plan
6. Run automated interface tests against Test Environment
7. Mark execution records Pass/Fail in Test Execution Manager
8. Report test results in dashboard/build results/defect records in CI server.
Building and Evolving automated test execution

- Look at what is possible based on current codebase
- Start small and quick, build to more complicated test setups
- Validate core functionality first
- Progress to edge cases and error conditions
- Refactor codebase to improve testability

<table>
<thead>
<tr>
<th>Test phase</th>
<th>Test Type</th>
<th>Focus</th>
<th>Rational Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unit</td>
<td>Module</td>
<td>zUnit (RDz)</td>
</tr>
<tr>
<td>2</td>
<td>Interface</td>
<td>APIs, Service interfaces, External calls</td>
<td>RTW or RIT</td>
</tr>
<tr>
<td>3</td>
<td>Functional</td>
<td>User interface</td>
<td>RTW or RFT</td>
</tr>
<tr>
<td>4</td>
<td>Performance</td>
<td>Response time from system</td>
<td>RTW or RPT</td>
</tr>
<tr>
<td>5</td>
<td>Exploratory</td>
<td>Manual testing (“poking around”)</td>
<td>RQM</td>
</tr>
<tr>
<td>6</td>
<td>Acceptance</td>
<td>User validation / pre-production</td>
<td>RQM</td>
</tr>
</tbody>
</table>
DevOps: Version everything, package everything, define a pipeline

Enterprise Applications have thousands of disparate parts
• Currently maintained in separate systems
• Limited linkage between systems for application dependencies
• Missing assets and information is rampant

DevOps practices force linkages, automation, and standardized packaging…
Moving toward the future
Completing the DevOps solution…

Smarter software for a smarter planet

Enabling Product and Service Innovation | Rational
Standardize z/OS and distributed images to ease deployment

- Standard topologies exist today (production LPARs)
- Standardized images few and far between
- Standardized/Automated deployment of COMPLETE system is spotty

Standardize and automate provisioning of everything …
Deploying z/OS … cloud-style
Focusing testing via dependency virtualization (using Green Hat)

Integration test complex systems

Increased Fidelity

Agile testing

Production
Pipeline testing with dependency virtualization

Controlled large system testing by isolating components under test

• Easier problem determination
• Lower test environment capacity requirements
• Improved component quality

Phase 1

Test Case

WAS

IMS Data Access

3rd Party Call

CICS Commarea Call

Virtual Services

Phase 2

Test Case

WAS

IMS Data Access

3rd Party Call

CICS Commarea Call

Virtual Services

Phase 3

Test Case

WAS

IMS Data Access

3rd Party Call

CICS Commarea Call

Virtual Services
Test Data Management scales down testing to the essentials

- Create targeted, “right-sized” subsets faster and more efficiently than cloning
- Compare to pinpoint and resolve application defects faster
- Improve development efficiencies

Creating right-sized targeted test environments saves storage costs & speeds testing

<table>
<thead>
<tr>
<th>Test Environment</th>
<th>Total Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production</td>
<td>200GB</td>
</tr>
<tr>
<td>Training</td>
<td>25GB</td>
</tr>
<tr>
<td>Unit Test</td>
<td>25GB</td>
</tr>
<tr>
<td>System Test</td>
<td>200GB</td>
</tr>
<tr>
<td>UAT</td>
<td>25GB</td>
</tr>
<tr>
<td>Integration</td>
<td>25GB</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>500GB</strong></td>
</tr>
</tbody>
</table>

Infrastructure reduced by 83%
Optim Test Data Management Solution

Create “right-size” production-like environments for application testing

Requirements

- Create referentially intact, “right-sized” test databases
- Automate test result comparisons to identify hidden errors
- Shorten iterative testing cycles and accelerate time to market

Benefits

- Deploy new functionality more quickly and with improved quality
- Easily refresh & maintain test environments
- Reduce storage and operational costs
Where should I start?

1. Identify a well contained project for adoption
2. Define infrastructure code for platforms and the project application
3. Define automated tests for the infrastructure and application, including conditioned data for testing
4. Adopt a single-stage continuous delivery process to support continuous build, deploy, and test for dev and test virtual environments
5. Inject monitoring as part of the standard pattern and use the data in the delivery process to improve feedback
6. Adopt a multi-stage delivery process that supports promotion of changes from one stage to the next (e.g., Dev to QAT).
7. Adopt a delivery process with promotion to production
8. Track and manage incidents in production linked to work/tasks in development
QUESTIONS

www.ibm.com/software/rational