Deployment Automation Basics

Implementing Consistent Deployments Across Environments

IBM

Contents

2 Key Findings

2 Introduction

3 Patterns of Failure in Manual Deployments

4 If They Are So Bad, Why Are Manual Deployments Common?

5 Elements of an Automated Solution

7 Maximizing Automation Investments

7 Simple Deployment Checklist

IT organizations are finding it difficult to release more quickly without adding more people and incurring additional risk. Here we discuss the limitations of manual deployments, common failure patterns, the benefits of automation, and a base set of features an automated system should provide.

Key Findings

- Manual deployments are inherently slow and error-prone.
- Deployment automation used only in development or only in operations may help one silo, but leads to a hand-off where changes to the process may be insufficiently communicated.
- Automated deployments provide superior audit trails.
- An automated deployment infrastructure provides a framework to build upon. Additional activities such as automated functional testing can leverage the deployment infrastructure.
- Deployments standardized across environments must still take into account environmental differences. The environment configuration is a key concern.

Introduction

In a continuing poll sponsored by UrbanCode prior to its acquisition by IBM, over 75 percent of respondents identified their deployment process as either "Entirely Manual," "Mostly Manual," or "Mostly Scripted." Relatively few indicated that they had push-button deployment capabilities. This result is not surprising: Until recently, deployment-automation options have been sparse, with vendors offering "solutions" that do not scale to the enterprise; require considerable process reengineering to implement; and provide little visibility into the "who, what, when, where, why, and how" of deployments.

Companies have turned to internal solutions and more scripting to help their team members achieve more consistent deployments and keep pace with the increasing demands of competitive markets. However, as with many "homegrown" solutions the expense ratio measured in both time and dollars often proves high: most organizations are not in the business of providing "process automation," and have to borrow resources in order to design, create, implement, and maintain an automation initiative. Often, even after initial success, the solution does not stand the test of time—changes in practices, such as the introduction of Agile, require a complete rewrite of the initiative.

Considering the options, companies attempt to improve human-driven deployments. It is difficult to invest in an automation initiative when perceived gains do not outweigh risks, when the options seem to provide only marginal returns. However, these incremental improvements do not sufficiently address the underlying problems of manual deployments. They still rely on a human to log into a machine, consistently run scripts in a precise order, and then notify interested parties.

This situation seems to lead to a single question, "Is automation really worth the time and effort?" If you ask teams with automated deployments, the short answer will most likely be similar to, "Yes, but it depends on what you mean by automated." As it turns out, a scripted process is not really an automated process (as we will discuss below). For those in the business of providing automation, a much higher threshold is set: If the "automation" does not address the patterns of deployment failure, then the process is not automated.

Patterns of Failure in Manual Deployments

Using the rough and ready definition of manual deployments, it is easy to see where they fail. It is safe to say that any deployment is manual if it is:

- Characterized by operators logging into various machines and following written scripts
- Slow and inconsistent
- Prone to error and failure
- Lacking in visibility and traceability

The first three are rather straightforward. However, the final criterion requires a better understanding of what happens to a deployment once the process is "complete." This is because a deployment process is not a separate and independent event in time; once the deployment is run, team members still need to know what was deployed where, why it was deployed, and who performed the deployment—especially if a failure occurred. For example, if a push-button deployment exists but the system does not provide visibility into the process, team members have to sift through logs on numerous machines to track down the error—which is not just manual but also laborious.

Keeping in mind this extended definition of a deployment may help as we identify a pattern of practices, described below, that show why manual deployments are slow, error prone, and "black boxes." In turn, these "bad" practices will provide insight into what is needed from a system that will operate quickly, be highly successful, and provide high traceability (i.e., an automated system).

Handoffs through Email or Drop-boxes

Because the person executing the deployment varies from environment to environment, email is often used to inform another team that they should deploy "something." It is not uncommon for the files that are to be deployed to be attached to the email or placed in a designated location on the network (i.e., a drop box).

Lag, inherent to this process, is a main reason why manual deployments are slow. A considerable amount of time can pass between sending a request for deployment and the recipient actually reading the email. In a perfect world, the deployment will take place as soon as the email is read; however, in the real world the actual deployment will take place after an additional delay.

Because different people request and execute the deployments in each environment, inconsistency is built-in, raising the risk of failures in production deployments. Individuals will respond to requests and interpret installation instructions differently.

Traceability is limited. The core audit trail is the series of emails scattered between many inboxes, making complete visibility virtually impossible. Further, because file shares (or attachments) are used to deliver the artifacts for deployment, the organization must deal with the risk that those files are tampered with, changed between build and various deployments, or just don't make it into the correct deployment.

Development, Test, and Operations Deploy Differently

Developers see manually deploying to a private test environment slowly and carefully as inefficient and burdensome, so they will often create short cuts or scripts to deploy. If the operations team has scripts, they tend to be created by operations for operations. Test teams have the desire to move quickly like developers. Between their desire to avoid false bugs and test the deployment process, they also want to mirror the deliberate, controlled production deployments. Unfortunately, test teams have little pull on developers or production support and often invent a third way of deploying.

Because inconsistency across environments is built into the process, repeated failures, on top of the duplication of work, are inevitable.

Adjustments to Deployment Process Communicated on "As-broken" Basis

When each team is deploying differently, another nefarious pattern emerges; when minor adjustments are made to the deployment process, those tweaks are communicated to other teams only in response to a breakage. It is common for a test team to report that the application is not working, or to file a bug. Time is wasted due to a broken deployment. At some point, the development team eventually realizes that they changed a setting, or added a step to the deployment process. This is eventually communicated to test teams that hurriedly apply the change and try to catch up on their testing. The same pattern repeats itself when neither development nor test team remembers to tell operations of the change and the deployment fails in production.

Deployment Steps Contained in Large Document

Complicated, manual deployments require careful instructions—often in the form of large deployment documents. The documents take many forms; from a Microsoft Word manual, to a spreadsheet, to a wiki. Often, there will be dozens, hundreds, or even thousands of steps that must be assigned to someone on the team and then precisely executed.

It is very difficult to correctly document and record hundreds of steps. Likewise, executing those steps correctly and exactly in the right order is difficult and time consuming. The process inevitably becomes slow and prone to failure when the deployment instructions become lengthy.

Developers Attain Production Passwords

Many of these patterns have lead to production deployment failures, which leads to another failure. When there is trouble in production, the highest priority of the deployment team is to repair production. Developers who know the system best are often brought in to explore it, figure out which of the steps in the large document were skipped, or what tweak was not communicated. More often than expected, the developers are simply given the production passwords so they can work more quickly. After all, fixing production is of the highest importance.

When we interview a diverse set of people from the same organization and ask if developers have passwords for production, usually almost everyone will say "No, that would be absurd and violate our audit requirements." And yet, one developer will say, "Yes, I have passwords to fix broken deployments." Because the passwords are handed over in a stressful time, and usually late at night, the deployment team often forgets they gave out the password. The result is a breakdown in control, audit capability, and traceability.

If They Are so Bad, Why Are Manual Deployments Common?

Manual deployments are slow, painful, and fail in production; yet, they are still extremely common in the IT industry. Why? A simple answer may be that until recently, powerful deployment automation tooling was not available.

However, even today, there is a great deal of resistance to automating deployments. Deployment teams remain comfortable with their existing practices. Because they are at the keyboard executing the manual steps, they feel in control.

They immediately see all the output and can respond to problems appropriately. Operators' feeling of control, along with organizational inertia, prolongs the use of manual deployments even when those not performing the deployment see serious problems.

Also common is the notion that "discipline" and "consistency" can remove the patterns of failure associated with manual deployments—but this is not the case. The problems that we see with manual deployments are often exposed by human error. If discipline and consistency were the answers, then there would be no need to communicate on an "as broken" basis if every change process change is properly documented. However, even the most disciplined organizations rely on this method—humans are not the best at performing rote activities. Likewise, executing dozens of steps correctly every time would not be a problem—if proper attention were paid. The reality is that no matter how attentive a person is, they will eventually error. Finally, the notion that a manual deployment process would not be as slow if everyone involved executed each step immediately when the previous one completed is unrealistic. How many people sit around waiting for something to do? More often than not, when that deployment request comes in, the recipient is busy doing something else.

Unfortunately, and as many studies have shown, people struggle to be consistent over time. Instead, humans are better at utilizing their creativity than following a script. To demand that people perform their tasks mechanically and absolutely consistently is to demand something against their nature, and an invitation to failure.

Unlike humans, machines have yet to perfect the art of creative thinking; however, they are excellent at acting, well, mechanically. They have proven their ability to perform repetitive tasks, to let us know when something has gone wrong, and have the "discipline" and" "consistency" to perform those tasks that hamper human performance.

So, if a task requires consistency and discipline, why not delegate it to a machine? Why not free up people to do what they are good at, and gain positive returns in saved time and improved performance?

Elements of an Automated Solution

Manual deployments are broken and cannot be saved by more disciplined deployment engineers. The rote work of executing a deployment should be delegated to an automated system that can execute a process consistently. However, not all automated solutions are alike. Whether you are looking to buy or build a deployment automation system, there are some "must have" goals to keep in mind. A mature deployment system should be characterized by:

- Reliable, highly successful deployments—especially in production
- Easy deployments—encouraging teams to take new builds faster
- Fast deployments—allowing early test environments to be on the newest build as possible
- Complete audit trail—spanning across all environments
- Robust security and separation of duties—controlling who can do what, when, and where

To achieve these goals, an automation solution should contain at least a few key elements:

- The entire deployment should be automated
- The deployment should target specific environments and automatically adapt itself to a target environment
- The files being deployed should come from a controlled artifact repository
- Security and visibility should underpin the entire system

One Complete Definition of Deployment Process—Automated

The first element of a successful solution is to translate the large document containing the deployment steps into an automated system. The same basic process for deployment must be used through all the test environments and production. Likewise, only the automated system should be used for deployments, and deployment errors should be corrected by fixing the automation and performing the deployment again.

Using the same process in all environments, and only fixing failures through redeployments ensures that when a change to the process is needed in an early test environment, the automation is updated. Tweaks are no longer communicated on an "as broken" basis, but are instead immediately incorporated. The deployment plan is always up to date. Because a (consistent) machine is executing that plan, it is always followed precisely and with no lag time between steps.

Formal Environment Definitions

To use the same process for all environments requires factoring out all environment specific information. This element is critical and a formal definition of an environment can capture key information including:

- To which servers does each component of the application get deployed
- Preconditions for deployment such as approval requirements
- Parameter variances, like passwords, should be handled securely while others are more visible

Combining a generic deployment plan with the environment to deploy to should yield the full deployment configuration.

Repository for Deployable Artifacts

Traceability is important and deploying "whatever happens to be on the file share" lacks traceability and increases risk. One needs to know that not only has the deployment process worked, but it has worked with the files that are about to be deployed. One (or more) authoritative repositories should be employed to provide a definitive source of uniquely addressable file versions.

These repositories should be integrated into, or natively accessible from, the deployment automation system. Example repositories include better build servers, maven repositories, asset management systems, or even the more robust deployment tools.

Inventory and Reporting

When the deployment automation framework is the sole deployer of an application, it knows what versions of each application are in each environment. This information can be leveraged to provide clear information about which build is where, as well as optimize system deployments, skipping deployments of components already present in the environment.

Access Control

In manual deployments, deployment access control has been equivalent to server access control. With an automated system in place, that is no longer true. Access to the automation provides deployment rights. The automation systems must themselves be secure and provide role-based security.

This approach provides new capabilities. An environment owner, who may only have approved manual deployments, but lacked skill or permissions to execute them, can now be provided self-service deployment capabilities. Likewise, it may be appropriate to revoke permissions from some users who were granted broad permissions to servers in order to perform deployments, but were instructed to not otherwise modify the servers.

Audit Trail

In many manual deployments, the audit trail consists only of what people think they did, or recorded. In some, extra logging is added in production. Others require those who execute deployments to take screenshots along the way. An automated system should provide full logging in all environments, automatically, every time. It should be clear what steps were executed as well as who triggered each deployment.

Maximizing Automation Investments

An automated deployment infrastructure is something the team can build upon. Smoke tests can be integrated so that rollbacks occur automatically on test failure. Likewise, application deployment can be coupled with automated updates to application and application server configuration. Deployments to test environments can be automatic based on standing schedules or events.

Once deployment automation is in place, these sort of additional practices can be layered on top. The deployment infrastructure provides the mechanics and traceability. People provide the creative insight to understand what would most benefit their organizations.

Simple Automated Deployment Solution Checklist

- ☐ Capable of automating your deployment processes
- ☐ Same process can be used in all environments
- ☐ Coordinates deployments of multi-tiered applications
- ☐ Models full application version across versioned components
- ☐ Tracks which servers are in which environment
- ☐ Tracks server roles in environment (e.g., Database Server vs. Web Server)
- ☐ Per environment parameters (e.g. Database password in Dev vs. Prod)
- ☐ Secure management and filtering of passwords/secrets
- ☐ Integrated Artifact Repository
- ☐ Integrations with Third Party Artifact Repositories
- ☐ Integrates with many or all of your deployment targets
- ☐ Extensible with the ability to create custom steps
- ☐ Integration with authentication technologies (e.g., LDAP)
- ☐ Role based security
- ☐ Maintains logs of all commands executed in deployments
- ☐ Inventory tracks which version is where
- ☐ Tracks who ran a deployment

For more information

To learn more about IBM UrbanCode Deploy, please contact your IBM representative or IBM Business Partner, or visit the following website: **ibm.com**/software/products/us/en/ucdep

Additionally, IBM Global Financing can help you acquire the software capabilities that your business needs in the most cost-effective and strategic way possible. We'll partner with credit-qualified clients to customize a financing solution to suit your business and development goals, enable effective cash management, and improve your total cost of ownership. Fund your critical IT investment and propel your business forward with IBM Global Financing. For more information, visit: **ibm.com**/financing

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America
October 2013

Please Recycle

RAW14335-USEN-00