ROI: The Value of Deployment Automation

IBM

Contents

2 Numerous sources, different focuses

3 Automating manual work

4 Reducing script writing and maintenance

4 Error-prevention (pre-prod)

4 Error-prevention (prod)

5 Audit and bug triage: visibility into what's where and how it got there

5 Reduced evening/weekend demands on release engineers

6 Bottom line

6 Appendix A

How will we benefit from adding a software deployment automation framework? Whose work will be impacted by the changes it brings? Can we justify the investment in making the transition to automated deployments?

This guide will help you answer these questions and explain how a deployment automation framework can streamline release operations and increase business agility.

Numerous sources, different focuses

An organization considering the adoption of deployment automation will do so to meet its own business objectives. Some organizations may be focused on clear reporting for audit, while others will be focused on getting features to market more quickly.

At IBM, we work with a variety of teams evaluating our industry leading application deployment automation solution, IBM UrbanCode Deploy. While this paper is not intended to be tool specific, we have drawn extensively from our interactions with our customers to lay out the value proposition behind that type of tool and provide examples.

Most evaluation teams focus only on the value of meeting their top priority. However, when trying to understand the value of a deployment automation tool, avoid focusing exclusively on the most pressing problems. Instead, consider the range of benefits the organization can expect.

Value: productivity, error reduction, visibility, & time to market

There are three major types of value delivered by deployment automation. First, productivity improves as work that was done through slow manual processes is replaced with automation, and test labs suffer fewer outages. Second, reducing errors eliminates rework, wait times and the business impact of failures in production. Third, an automation system should record and broadcast what it does improving visibility and auditability.

Together, these core benefits enable companies to deliver to market faster, improving business agility. Common release bottlenecks are eliminated, while the cost and risk associated with production releases are dramatically reduced.

This document will examine the benefits of automation across several categories:

- Automating manual work: The simple reduction of deployment effort from automating a previously manual process.
- Reducing script writing and maintenance: Benefits from using prebuilt integrations over writing your own scripts and tools to help with a deployment.

- Error prevention: Manual processes are error-prone. Benefits in both production and pre-production are examined.
- Audit and visibility: With better tracking of what is where, bug triage times are improved and audits go more smoothly.
- Reduced evening/weekend hours for release engineers: Benefits to productivity, fewer errors and even retention.

Calculate based on desired state

A standard ROI calculation for deployment automation asks, "How many application deployments do you do in a year?" and "How many servers are updated with each deployment?". If you are trying to improve an existing process, this can be an appropriate measurement.

Use a forward-looking approach to counting deployment frequency and scale. Calculate based on the reality the deployment automation effort will support when implemented. For example, if part of the motivation is that a major application is scaling up from a handful of servers to several dozen, consider the value based on the more numerous count.

Likewise, if the organization is trying to speed time to market, estimate the impact of deployment pace to both production and pre-production environments. A twofold to fourfold increase in production releases, and as much as a tenfold increase in deployments to test labs are common.

Automating manual work

Reducing the efforts needed to execute a release is an obvious saving. Detail the team's existing process and how much time it takes to execute—effort and clock time are both important. The effort saved is a direct cost savings. Clock time will be useful when examining the savings associated with reducing environment downtime.

When calculating savings (See Appendix A), consider various types of deployments that are executed and the effort required for each. Often, a deployment to a test environment is much faster, and easier, than a production release. Since test environment deployments are more frequent, the smaller savings per deployment are still important. Calculate an expected savings for each type of deployment.

Many find that a deployment to a test lab happens about a day after it is requested and takes 30 - 180 minutes of effort. After implementing a deployment automation workflow, the manual effort is reduced. In the example in Appendix A, the deployment starts shortly after the request, requires only five minutes of effort, and reduces the outage by 50 percent.

Production deployments can follow a similar pattern. But due to their larger scale, they take more effort. Because an automation system can multi-task better than humans, the savings tend to be more dramatic as the number of servers increases. This is especially true for the sizeable minority of organizations who spend multiple days, or even weeks, on a production release.

Remember to count configuration changes

For many groups, the task of updating application configuration to match the environment is handled separately from the deployment of the application itself. When considering deployment time and errors introduced in a deployment, be careful not to forget the impact of configuration, as a good deployment automation framework will help streamline that process as well.

Shrinking outage windows

With shorter deployments come shorter outage windows. This can mean reduced impact on revenue generating systems in production. In test environments, less of the developers' and testers' time is wasted as they wait on the application to be testable.

"We went from barely completing work in the 4 hour window (and always extending the window for roll-backs) to finishing in about 20 minutes"

—Director Release Management for a mutual fund company

Reducing script writing and maintenance

Most manual processes leverage "helper scripts". These scripts must be written and maintained by experts. They must also be distributed to the appropriate servers regularly. When a script maintainer leaves the company, the scripts left behind are often complex enough that the maintainer's replacement is afraid to make modifications. Scripts (and the processes they supported) become static, and potential improvements are avoided until enough pent-up demand results in a complete and costly rewrite.

Good automation frameworks provide integrations to streamline deployments to common target platforms. Our UrbanCode Deploy customers find that almost all of the script writing work is eliminated by plugins provided with the tool. The remaining scripts are automatically distributed to the target servers and executed consistently.

To understand the savings from reduced scripting, interview deployment engineers and developers to see how much time they spend on these activities.

Internal deployment tools

Some organizations have gone beyond mere scripting and created in-house deployment automation frameworks. The costs of maintaining those systems and adapting it to new needs are often high enough to justify switching to a third-party tool.

Error prevention (pre-prod)

Deployment errors in development and test environments are expensive. At IBM, we have worked with groups that averaged one day each month where offshore test teams did no testing due to a catastrophic deployment failure. Another organization had a four week test/hardening cycle. On average, they spent two weeks sorting out deployment and configuration problems. If these challenges were eliminated they would get to market two weeks faster or have higher quality.

Interview testers and developers to see what kinds of time leaks they are facing. For a simple ROI, thinking in terms of engineer time wasted is probably easiest. However, the business may care much more about speeding time to market or quality, and that should be part of how you present your results.

Error prevention (prod)

What does a deployment error cost in production? Do we impact our users in an expensive way? Do we lose the goodwill of our customers? Do we lose money on e-commerce? How much effort is required to remediate?

Placing a dollar figure on many of these concerns (aside from e-commerce losses) can be challenging—make your best effort. Then calculate the losses per deployment error, the frequency of error, and multiply by the number of production deployments the business wants.

Rollback should also be considered. If the deployment is executed well but the application behaves badly in production, reverting to an old version may be desirable. What are the savings of being able to rollback to a previous version in a matter of minutes?

The Cost of an Hour

An input to many of the equations below is the cost of an hour of a "fully loaded" engineer. The concept of a "fully loaded" engineer is to include the company's outlay for benefits, vacations, insurance, etc. You might get a good number to work with from HR, but more likely, you'll need to estimate it yourself. For an IT worker, these additional costs can frequently be anywhere from 20 - 40 percent of salary. We'll assume 30 percent. That makes the estimated loaded cost calculation:

Estimated Annual Salary / 52 weeks/year / 40 hours/week * 1.3

So if we have an engineer making $80,000, a reasonable estimate of hourly loaded cost would be: 80,000/52/40*1.3=$50/hour.

Audit and bug triage: visibility into what's where and how it got there

The best deployment automation solutions also provide excellent visibility and automate reporting into what is where, who deployed, who approved, and how the deployment process was designed. In short, audits become much easier. This can result in two line-items in an ROI calculation.

1) Effort saved in complying with audit requirements
2) Reduced likelihood of audit failure

A better understanding of what is where can help test and release planning teams as well. Bug triage is easier when everyone understands exactly what versions of each system component were present in a given environment. Knowing exactly what tests passed is crucial when deciding what to promote to a higher environment such as production.

To support both audit and visibility, it's critical that the deployment automation framework provides precise tracking of versions, and can guarantee that what is deployed to production is bit-for-bit identical to what was certified in prior environments. Better deployment automation frameworks, like UrbanCode Deploy, even provide an integrated package repository to support these needs.

Reduced evening/weekend demands on release engineers

Deployment automation often allows us to reduce or eliminate the frequency of release engineers working long hours on evenings and weekends. With automation, one may simply schedule a deployment to run automatically when the window starts. Or, by speeding the deployment and using zero downtime techniques, the deployment window may be able to be moved into normal working hours.

Avoiding long hours late at night is important. In the day (or days) following an all-nighter, engineers involved in a large release effort are less productive and more likely to make mistakes. At the most, the engineers simply take the morning or day after a major release off to sleep. Disruptions in their normal sleep patterns can impact their ability to focus for the rest of the week as well.

Late night deployments are also more risky. When up late, engineers suffer reduced ability to make sound judgments based on risks. They will tend to take riskier paths during the most critical releases. It is important to acknowledge that the highly critical deployments we often push to the middle of the night to minimize impact, are executed by engineers whose lack of sleep represents a significant risk.

Releases at odd hours also have an impact on retention. While most engineers will absorb the occasional all-nighter pretty well, increasing release frequency can make repeated all-nighters unbearable. This can drive away some of your best people, leading to expensive hiring and training cycles.

Bottom line

Manual deployment processes are inefficient and error-prone. As our industry moves towards more frequent, more modular releases, the problems become unbearable. With automation, teams are able to deliver more frequently, with higher quality, and less painfully.

Appendix A: the math

The cost of an hour

An input to many of the equations below is the cost of an hour for a "fully loaded" engineer. The concept of a "fully loaded" engineer is to include the company's outlay for benefits, vacations, insurance, etc. You might get a good number to work with from HR, but more likely, you'll need to estimate it yourself. For an IT worker, these additional costs can frequently be anywhere from 20 - 40 percent of salary. We'll assume 30 percent. That makes the estimated loaded cost calculation:

Estimated Annual Salary / 52 weeks/year / 40 hours/week * 1.3

So if we have an engineer making $80,000, a reasonable estimate of hourly loaded cost would be: 80,000/52/40*1.3=$50/hour.

The cost of a deployment

Task	Duration – Manual (min)	Effort – Manual (min)	Duration – Automation (min)	Effort – Automation (min)
Request deploy	5	5	4	4
Lag until deploy starts	60	0	0	0
Copy files to 20 servers	15	5	10	0
Execute install steps on 20 servers	180	180	10	0
Update config files	30	30	2	0
Manual smoke test of env	15	15	15	15
Total	305 min	235 min	41 min	19 min

Knowing what a deployment costs with and without automation is central to understanding the benefits of automation.

With this kind of calculation, we can see that for each deployment to a 20-server environment, we save 206 minutes of effort—about 3.5 hours. If we determine that we are doing one of these deployments a week, and an hour of engineer time costs $50, we save:

(206 mins/deploy) * (1/60 hours/min) * (50 dollars / hour) * (52 deploy/year) = $8,926 per year.

The other way to look at this is that for the same engineering cost, you could go from 52 deployments per year to 546—about 2 deployments each business day. If moving to more regular deployments is the goal, then you are comparing an automation tool against hiring many more engineers to do the deployments:

(206 mins/deploy) * (1/60 hours/min) * (50 dollars / hour) * (546 deploy/year) = $93,730 per year.

Using "clock time" savings not just effort time.

Above, we determined how much less cost is associated with actually doing the deployment. In the example above, we also care that the deployment takes 4.5 hours less time to accomplish.

In a test lab, we may determine that our testers are wasting about half their time while their primary test environment is down. If that's currently happening once a week, we could calculate the cost of lost time as:

(305 minutes – 41 minutes) (1/60 hours/min) * 4 testers * 50 percent wasted * (40 dollars / tester hour) * 52 / year = $19,136 / year savings.

The new time to deploy is less than an hour. We now have the option of having no impact on the testers at all if deployments are scheduled to happen during lunch or standing meeting—nudging the value up to $21k. More importantly, a standing deployment schedule could push updates twice a day (overnight and during lunch) ensuring that the test team was always testing the latest code from development. A bug found sooner is cheaper for developers to triage and fix, than one found later.

* This is just for one weekly deployment process. Do this estimation for a representative sample of your processes and extrapolate a total benefit.

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 IT solutions that your business needs in the most cost-effective and strategic way possible. We'll partner with credit-qualified clients to customize an IT financing solution to suit your business goals, enable effective cash management, and improve your total cost of ownership. IBM Global Financing is your smartest choice to fund critical IT investments and propel your business forward. For more information, visit: **ibm.com**/financing

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America
January 2013

IBM UrbanCode Deploy.

The content in this document (including currency OR pricing references which exclude applicable taxes) is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Please Recycle

RAW14331-USEN-00