Achieving Better Deployment Efficiency with Automation via Open Source Software

Rudhuwan Abu Bakar
Software Engineering Lab
MIMOS Berhad, Kuala Lumpur, Malaysia

Wan Mohd Firdaus Wan Ramli
Software Engineering Lab
MIMOS Berhad, Kuala Lumpur, Malaysia

Abstract- In Software Development Lifecycle (SDLC), one of the most challenging phases is deployment and implementing consistent deployments across environment should be a goal in all development projects. IT organizations are finding it difficult to make quicker subsequent releases without adding more resources and in turn, incurring additional risk. Deployment becomes even more challenging if it involves complex system integration with multiple modules, involves larger development teams and targeted for various deployment environments. Due to the system complexity and various factors, the process of deploying application needs to introduce better methodologies to increase the application deployment efficiency and to speed up process of updating or upgrading the existing application. Among commonly encountered deployment issues are missing dependencies, wrongly compiled codes being deployed, incorrect version of deployed packages and so on. Finding the root cause of those issues themselves requires lots of resources and time. However, most of those issues can be easily prevented if the deployment process is automated. A proper control required to maximize time and resources to eliminate impending those issues. Automated deployment will definitely improves deployment efficiency and reduce time and resources in troubleshooting. It also minimize human interaction where human are error prone and one of the most common root cause in deployment activities. This paper will discuss the limitations of manual deployments, common failure patterns, the benefits of automation, and recommended features an automated deployment should have. 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.

Keywords – deployment, automation

I. Introduction

Deployment is one the most important phases of Software Development Life Cycle (SDLC). One of the areas that deployment has to deal with is having a good and solid process in place. This will enable better deployment efficiency and smooth the process of upgrading existing software. Even with development of better tool to help the deployment process, still manual deployment which is human driven deployment remains the main trait in most of the development projects today. Perhaps one of the hold-back is the tool maturity. But the tools are getting sophisticated. Among other tools that have been actively developed are PUPPET[1], CHEF[2] and ANSIBLE[3], just to name a few. However, the paper will only discuss on Jenkins[4] and its deployment features since it is the main build server in the organization.

Most of the time, deployment process revolves around *entirely manual*, *mostly manual*, or *mostly scripted*. These processes would fit in nicely in a small scale deployment. Once it grew larger in size and complexity, these processes do not scale up. Due to that, they required substantial process reengineering to implement. It may eventually involve time and money. Apart from that, these processes provide little visibility into the "who, what, when, where, why, and how" of deployments.

Moreover, there is a concern on mostly human driven deployment. Humans by nature are prone to mistakes even the most particular ones. In deployment world, a slightest of mistakes will cost lots of resources and time especially in finding the root causes. Sometimes, these process even will consume majority of resources and time where most of those will simply cam be eliminated if there is a degree of automation in place.

In terms of deployment tool availability, most commercial products provide feature to support automated deployment but implementing such products is deemed as high investment. On the other hand, there is always an

open source solution that can achieve the same objective but with lower Total Cost of Ownerships (TCO). Of course, there are disadvantages when deploying these alternative solutions..

This paper will be proposing different approaches taken into making automated deployment and showcases some best practices applied in current running project. Lastly, the paper will present a recommendation on the best approach that may benefit others that are considering the same initiative. In the end, it can be demonstrated that even without spending a fortune; an enterprise level solution can be achieved with open source software technologies.

II. PROBLEM STATEMENT

The current way of deploying software is mostly manual and most of the times are totally human driven. The common pitfalls faced can be broken down as follows:

- *Slow turnaround time to deploy*

  One main feature in manual deployments is communication medium. Email is often used as a medium to inform others such as testing team or deployment team that they should deploy "particular released artifact" from SCM. However, 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. The actual deployment will only start after an additional delay.

- *Inconsistent deployment*

  Developers often consider manually deploying to a private test environment burdensome, so they will tend to create short cuts or scripts to deploy. The same thing will be done by deployment and testing teams, as both will create scripts to fit their needs. Between testing team desire to avoid false bugs and test the deployment process, they also want to mirror the controlled production deployments. Unfortunately, test teams often lack 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.

  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. When each team is deploying differently, a pattern emerges; when minor adjustments are made to the deployment process. The tweaks on particular settings or configuration are communicated to other teams only in response to a breakage. 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 notifies deployment team of the change and the deployment fails in production. All these created unnecessary wasted time and resources to resolve broken deployment.

- *Prone to error and failure*

  Managing manual deployment into various environments such as development, testing, staging and production at the same time, is a disaster waiting to happen. Operators need to log into various machines and execute installation prepared scripts orderly. Even most attentive person will likely to make mistakes and break the deployment. Team members have to shift through logs on numerous machines to track down the error. These are not just manual but also laborious. Probability of certain changes done to the process may not properly handed over to other teams is very high.

- *Lacking in visibility and traceability*

  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 deployment failure occurred. In current deployment process, traceability is limited. The core audit trail is the series of emails scattered between many inboxes, making complete visibility virtually impossible. 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. The team members 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. There is no possible way to verify the integrity of the files.

- *Deployment Steps Contained in Large Document*

  Depending on the complexity of the application, a more complicated application usually requires longer instruction steps which may create a large deployment document. This is directly proportional to time taken

to complete the deployment process. Executing those steps correctly and exactly in the right order is difficult and time consuming. The process inevitably becomes slow and prone to failure.

Clearly the current process is not the ideal way of deploying software. It is still too dependent on manual task executions which are often prone to human error and failures. It is impossible for human to maintain high level of consistency and discipline over time even for the most attentive person. It is always better to minimize human intervention and better to delegate task that requires consistency and discipline to a so-called "machine". Existing commercial solution would be able to meet the requirement as set by the management. However, deploying such solution is not an option as there is no budget allocated for the implementation. Therefore, there is a need to devise an implementation to support those requirements within the current infrastructure with a minimal cost. Furthermore, commercial solution comes with lingering user license cost and that is not the alternative which management agreed to proceed

III. APPROACHES AND IMPLEMENTATIONS

Due to those pitfalls, the organization top management has proposed new requirement for deployment process and system. The system has to have the following features in order to properly support day-to-day software deployment activities:

- It should provide superior audit trail spanning across all environments
- It should provide consistent and fast deployment which provide latest build as early as possible
- It should provide mechanism for ease of deployment
- It must deal with the risk of artifacts being tampered with and changed between build
- It should provide robust security and separation of duties - controlling who can do what, when, and where

The solution not only should encompass all above but also it should once and for all eliminate common assumption that software projects will always finish late, when delivered, it will underperform, and will not provide good ROI (Return on Investment). Therefore, there is a need for a new way to develop and deploy software that not only supported constant, rapid change, but scaled fast

a. Solutions to Slow and Inconsistent Deployment

To fulfill a greater need for simultaneous deployment across different platforms, having a standardized tool is important. Build server is significantly perfect for auto deployment. It can be configured to only allow successful build only to be deployed. The deployment can be control as to where or which environment this particular build is for. We have enough issues to manage developers, testers and deployment team mindsets, so by defining a standard deployment tool.

Having a dedicated build server is important as to manage the deployment activities such as deploying a development release to system test. This would eliminate notion of "it works on my machine" mantra. Occasionally, when a build break the developer will shrug off the failure by stating that the build work fine on his machine. Most of the time, however it appeared that the failure is due to missing dependencies where the library is sitting on the developer's machine only. Therefore, there is a need for proper collaboration between development and deployment teams as to avoid these types of issues. By deploying a dedicated build server, the team would only deploy from one single source and this would definitely increase the consistency of the deployment process.

Apart from build server, the needs to have a single SCM[5] tool for project artifacts such as codes are also important. By having the repository, the artifacts will not have to be distributed through email as an attachment any longer. Moreover, the integrity of the artifacts is suspect where the codes for example may have been tampered with or changed before it released for test system build. To build directly from SCM would definitely ensure the same codes are used by developers for their build.

In terms of producing software, unit test is one of the important aspects of good quality coding. Therefore, to have built-in unit test is an added advantage. By producing unit test code, developers would straight away gauge whether or not their codes cover the entire requirement. One of the frequent software quality metrics monitored are code coverage and code compliances. The coverage data is only possible if there is unit test code; therefore, making writing unit test code mandatory is definitely needed to produce better software. Apart from that, build server has incorporated automated unit testing where it will be executed during build. This is an important feature that a build server should have.

b. Solutions to Lack of Traceability and Visibility

Lack of traceability and visibility throughout software development process certainly plays its part in the quality of the software deployed. It is always a disaster waiting to happen when development, testing and deployment teams are working in silo. All of them will only start to communicate once deployment to production starts to have issues. There is certainly a need to have a better collaboration among the team. One way is to have a common tool to have traceability and visibility to the project. So that everyone would know the current status of the project for example the quality side of it. Being able to monitor the quality of the codes from time to time is important. The whole team should have access to see the statuses of their code quality are. Build server would be able to provide required dashboard for that purpose as it seamlessly can be integrated with SonarQube[5] and Cobertura[6]. It provides a dashboard with information like code coverage and code compliance for software quality reporting. Certainly, to have a system with dashboard that displays the data periodically will keep the teams aware of the quality of software that they are producing.

As the deployment environment varies, there should be a proper way to build and deploy to respective environment fast and efficient. Instead of having deployed packages transferred to various environments, with build server, the tasks can be segregated in terms of build jobs. Each job will pull codes from specific repository and upon successful build; it will automatically deployed the build to the respective environments. With this, the deployment would be better controlled and visible to all team members. The questions of who, what and where would be answered through this implementation.

It is always difficult to control deployment process. There are times where different people are executing the deployment. Sometimes, there is no way to have traceability to see who did what and where it was deployed. This create uncontrollable scenario where anybody can deployed anything. There is a need to limit the deployment role where only certain people have the permission to build and deploy. Build server is tightly integrated with user roles and therefore can limit who does what. Each build can be assigned to specific personnel and therefore prevent unauthorized build and deployment. This create traceability and answer who, what and where questions. In any case of manual deployment, usually only specific personnel such as developer will know the current status of the build deployed. Nobody else knows what has been deployed, and where it is deployed and who actually deployed. As project complexity grows, these will create worst case scenario when it came to troubleshooting the root cause. Build server will be able to tightly control who and what can be done for build and deployment. It equips with system log also which will helpful during troubleshooting.

Besides that, notification or alert for failed build is important as the team will move quickly to work on fixes. Without proper alert system, things do not move as fast as the information from one to another maybe delay. There are cases where the exact failure cause is not communicated. This will create further delay. Notification or alert creates project visibility to the project members where the notification to all. Build server would be able to provide such feature it will notify developers on any failed build via email. With this, developers would be able to react more quickly and fixes would be delivered much faster.

c. Implemented Solution

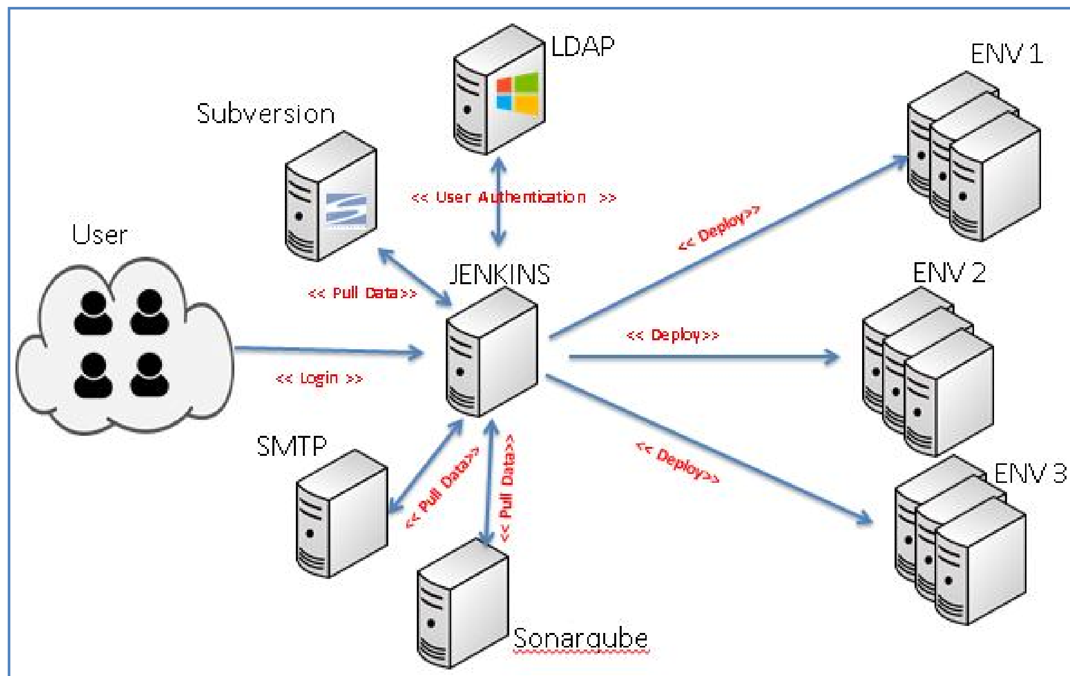

Figure 1: Automated Deployment with Jenkins

As shown is the implemented solution for auto deployment. The heart of it all is a build server called Jenkins.It is proven as reliable software and easily configurable and runs on multiplatform and best of all is free. It will act as a dedicated build server where users and roles are pre-configured. This is to enable control on all builds and deployments to various environments ENV1, ENV2 and ENV3. All artifacts will be inside a Subversion repository which will be pulled by specific Jenkins job for specific environment. To add traceability and visibility to the project, related software quality metrics will either be displayed in Jenkins own dashboard or in Sonarqube[7] dashboard. All these will be depending on what type of metrics being generated. Last but not least, SMTP[8] is primarily for notification or alert dissemination. All these should be the minimum requirement for auto deployment initiative.

As emphasized, there is a need to control the integrity of codes and build for deployment. Jenkins works seamlessly with Subversion[9] which is a very dependable SCM[10] tool. Having a single repository to manage your codes will enable the control element. Each time the build is executed, it will pull the latest changes from the repository from specific releases or version. With Jenkins also a build can be scheduled for example nightly build. And equipped with automated notification on failure will make fixes can be delivered the next day. This will speed up the development process. Furthermore, while keeping the traceability and visibility aspects in check also, with automated dashboard.

To speed up releases to various environments (ENV1, ENV2 and ENV3), Jenkins would be able to support simultaneous build and deployment. A specific Jenkins job can be created for a specific environment for example development sandboxes, testing and so forth, and all these build can be triggered simultaneously. Upon successful build, those build packages will be automatically deployed to each specified environment. With this, control is imposed on which version is going to which environment. The new build can be deployed faster and released to specific environment consistently without fail like manual deployment usually does. Once automated, the process will not change accidentally ever again. The common plugin used for Tomcat-based deployment is Deploy Plugin[11] while scp[12] and rsync[13] usually used for Apache-based deployment. Moreover, it definitely will eliminate “it works on my machine” mantra completely.

IV. CONCLUSION AND RECOMMENDATIONS

As seen from the implementation, the efficiency in deploying releases and fixes increases tremendously because a machine is executing the plan. It will consistently follow precisely with no lag time in between. Thus, this basically

removes unnecessary issues usually caused by human errors. An automation solution should contain at least the following key elements:

- The same basic process for deployment must be used through all the development, test and production environments.
- The codes being built and deployed should come from a controlled artifact repository such as Subversion. The repositories should be integrated into, or natively accessible from, the deployment automation system.
- Security, traceability and visibility should underpin the entire system Access control should integrated with LDAP[14], role-based security, read-only, assigned build capability 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. Also, SMTP server should be integrated into automated deployment system for notification on build failure.
- Build server should be the only source used for deployments, and deployment errors should be corrected by fixing the automation script and re-run the deployment.
- The build for deployment should target specific environments and automatically adapt itself to a target environment which servers does each component of the application get deployed.
- The entire deployment should translate a large document containing the deployment steps into an automated system.

All in all, having a dedicated build server would provide better efficiency and control over deployment process, in turn minimizes human interactions. Furthermore, using free and mature open source software added more advantages the whole deployment process.

REFERENCES

[1] https://docs.puppet.com/guides/introduction.html last accessed 29 Jul 2016.
[2] https://www.chef.io/chef/ last accessed 29 Jul 2016
[3] https://www.ansible.com/how-ansible-works last accessed 29 Jul 2016
[4] https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins last accessed 3 Aug 2016
[5] https://en.wikipedia.org/wiki/Software_configuration_management, last accessed 3 Aug 2016.
[6] https://en.wikipedia.org/wiki/SonarQube, last accessed 15 June 2016.
[7] http://cobertura.github.io/cobertura/, last accessed 2 Aug 2016
[8] https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol last accessed 20 Jul 2016
[9] https://en.wikipedia.org/wiki/Apache_Subversion, last accessed 25 Jul 2016
[10] https://wiki.jenkins-ci.org/display/JENKINS/Deploy+Plugin last accessed 3 Aug 2016
[11] https://wiki.jenkins-ci.org/display/JENKINS/Publish+Over+SSH+Plugin last accessed 3 Aug 2016
[12] https://codeship.com/documentation/continuous-deployment/deployment-with-ftp-sftp-scp/#continuous-deployment-with-rsync last accessed 3 Aug 2016
[13] https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol last accessed 3 Aug 2016