International Journal of Experiential Learning & Case Studies
3 : 2 (December 2018) pp. 236-245. dx.doi.org/10.22555/ijelcs.v3i2.2451

Achieving Software Release Management and Continuous Integration using Maven, Jenkins and Artifactory

Shahbaz Ali Syed, Tariq Rahim Soomro***

Abstract

It is often required that the developers working on a single module or software project, while working from remote locations. Scenarios like that increase complexity and challenge to manage releases and to merge their daily work in a single artifact. For component based development, developers are responsible for the merging their own work with others and to make sure the integrations went smooth with no error and problems. Sometimes it's hard to do this job because of the large size of team members, big project size or geographically separated teams. Through Continuous integration this whole process becomes lot easier because all the validations and verifications will be done by an automated job. This job will check for the changes in the project code and whenever a change happens it will run the build process automatically, which detects errors and problems with the code and also notifies the concerned team member who is responsible for this error. This study introduces an approach to setup a Continuous Integrations techniques in a project using open source tools like Maven, Jenkins and Artifactory.

Keywords: *Continuous Integration, Maven, Jenkins, Artifactory*

INTRODUCTION

It is often required in agile methodology to deliver a product by speeding up the development time without compromising the product quality and reliability. To implement these kinds of scenarios, project managers and development teams often think of implementing Continuous Integration (CI) as a part of their software release management. Release management is a process of managing software releases during development phases till the release of the final product. This process sometimes difficult to manage and often cumbersome because of the size of the project, large number of developers having their own copy of codebase, several software change requests, risks, and finally the hidden software defects. Here CI comes into picture to simplify the process from development to the successful bug free delivery. CI will allow the developers to integrate software code into a shared repository several times a day, each check-in by developers to code repository is verified by the automated build process and allow the developers to know the problem, its location earlier. Software development is a complex

Correspondence:
** Research Scholar at Department of Computer Science, SZABIST Dubai, United Arab Emirates. shebi.ali@gmail.com*
*** Professor of Computer Science at College of Computer Science & Information Systems, Institute of Business Management, tariq.soomro@iobm.edu.pk*

process where business often demands enhancements or changes in existing software or to build new applications. Such kind of changes represents new business enhancements and cause more business revenues (Henson, 2009; Eliav, 2018).

Software Release Management

A release is a piece of software with a set of features comes out of the result of different steps of software development life cycle as below (Henson, 2009):

- Development: The software is being developed and tested by the team who developed
- Alpha Version: The first version tested by the team outside the development team. Most of the business related functionalities are present but this version also has many bugs
- Beta Version: The first appearance of the release makes public for testing. It may still have bugs but less than the Alpha version
- Release candidate: The final release makes available for public

For large software projects, release management is an essential process and without any doubt the proper and efficient management of software release can help the organization to produce the high quality of software products and the satisfaction of the customers to the high extent. Release management is also plays important role with regards to the quality management, since its main goal is to deliver the software without compromising the quality and user expectations (Henson, 2009; Eliav, 2018). It is quite normal in most of the IT companies that when the application development team is done writing with code then IT Operations take care of putting these changes into production. This is not an easy job as it looks. Development team while focusing on the change development, customer satisfaction and business requirement they think it is easy to change the production environment. Whereas the IT Operation team usually keep the production responsibility and are reluctant to introduce new changes and enhancements that could potentially cause instability to the production environment (Wright & Perry, 2012). Both the views discussed above are valid but the organization still wants to reduce the costs and time associated with moving the changes into live environments. They want to enhance the user experience and business extensibility while reducing the impact of failures and reducing the risks to end-users also they looking for ways to bridge the gap between Development and IT operations. Such kind of organizations needs Release Management.

Challenges and Issues in Software Release Management

Some of the issues normally software teams come across during development (Wright & Perry, 2012):

- Code validation: Developers committing their work every day, sometimes they ignore the validation and commit the code leaving the mainline in unstable state
- Integration: Sometimes developers are geographically distributed and they have some dependencies on other team's work. In order to make this process smooth, someone from the project team should manually confirm for the quality and completeness of the last build committed to the mainline
- Time consuming: Since all the above process takes considerable amount of time, and thus will add delays to the final release

In a typical agile software development methodology, release phase comes later after iterations of analysis, design, coding and testing phases. The release phase then can further be divided into two sub phases, Pre-Release phase where minor functionalities or changes has been introduced and make available for beta testing. That puts in additional acceptance testing before the release actually moved to production. In addition, the first iteration should include minor changes, and major changes should be kept aside for other iterations. Once the user acceptance test is passed, then other production phase starts, where the final release is make available for the end users (Kaur, Choudhary, & Mehta, 2012).

Continuous Integration

Continuous Integration is a software development practice, where the code of different developers of a team integrates into a single shared code repository several times a day, means each of the developer has to integrate his daily work at least daily. For every integration, there will be a process of compiling and validating the merged release by an automated build process which also includes running of unit test cases if available. This way of code integration and automated build will save time to merge the daily work of developers and help the development team to detect errors and potentials problems within the code before it's too late (Fowler, 2006; Vlaanderen, Jansen, Brinkkemper, & Jaspers, 2011).). Figure 1 below shows a typical environmental setup for CI process.

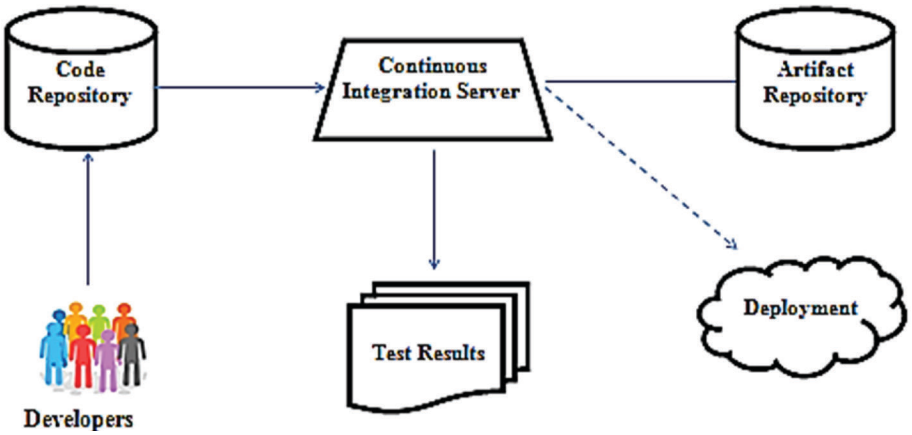

Figure 1: Continuous Integration Environment

To control the challenges, following are the key practices for Continuous Integration (CI) that is usually be followed as best practices (Fowler, 2006):

- Single Code Repository
- Automate the Build
- Make Your Build Self-Testing
- Everyone Commits to the Mainline Every Day
- For each developer commit should rebuild the entire Mainline
- Fix Broken Builds Immediately
- Keep the Build Fast
- Test in a Clone of the Production Environment

- Every developer should have access to latest build
- Everyone can see what's happening
- Automated Deployment

Apache Maven

Apache Maven is software project management and build tool, built for Java based projects. It is based on the concept of a single configuration file called Project Object Model or POM in short. Project default configurations and build can easily be done through Maven tool (Apache Software Project, 2018). The primary objective of Maven is to help java developers to build and manage any Java based projects and make their daily work easier and understandable. Apart from these main objectives, Maven also helps the development team to (Apache Software Project, 2018):

- Build the whole process easier
- Uniform build system
- Plug-in Support
- Get the quality project information
- Follow the development best practices
- Allow transparent upgrades to new versions and features

Through Maven, development team can easily fetch useful information like:

- Dependencies
- Change log document
- Cross referenced sources
- Mailing lists
- Unit test results

Maven also helps projects for release management and issue tracking. It also promotes a specific layout for the Java project so that once the developer is familiarizing with the layout, he can easily understand any other project uses Maven. Maven has two important roles in release management and CI (Apache Software Project, 2018; Oracle, 2014; Scholte, 2018):

- Distribution Management: Every project which is going to support continuous integration should have configured distribute management settings in its POM file. These settings will tell Maven that after the build process where the binary should be stored, which could be any remote or local repository
- Snapshot Repository Settings: There are some more configurations that Maven uses to communicate with the repositories. One is Update Policy, which will specify how frequently Maven should check for the latest version of the dependencies. Since API are meant to be change so frequently; therefore, this setting must be configured accordingly. Other setting important is Server Credentials, which keeps the credentials for Maven to access the remote or local repository. All the Maven repositories will authenticate the request before any new artifact or binaries put into the repository. Password can be encrypted based on the Maven's password encryption guide

Artifactory

Artifactory is an enterprise maven based repository, offers advanced proxying, caching mechanism and security procedures to provide a robust, reproducible and independent build environment. Artifactory as a repository manager serves two main purposes in release management and CI (Jfrog Artifactory, 2018):

- Providing single deployment destination for binary releases of an organization: It stores all the generated artifacts and binary releases for the organization. These binaries may call dependencies for some projects of the same organization. Once the project is build maven check in the local repository for the required dependency, if not found then it will check the remote repository and download it into the local repository for future use.
- Providing highly configurable proxies between the organizations and the public repositories: There are number of benefits in proxying the Artifactory repository. Proxying will speed up the process of build within the organization by registering a local repository, which works as a cache for all binaries downloaded from the remote repository. If development team of the organization requires certain binaries to compile their project and if they are using Maven with Artifactory the required dependencies or the dependency's dependencies will require downloading once from the remote repository to the local repository. This strategy of proxying the remote repository in local cache is saving the unnecessary load on remote repository by downloading the same artifact again and will save huge amount of build time if the binaries are in hundreds of megabytes.

Jenkins

It is a leading open source Continuous Integration (CI) server available in the market. It is written in Java. Jenkins as a continuous build tool, enables the development team to identify the failures and fix them in the source code rapidly and to automate the whole building process including testing. Jenkins is easy to use and user friendly and support variety of environments. Below are some of the main features of Jenkins (Kawaguchi, 2018; Jyothi & Rao, 2011; Gaston, et al, 2014):

- It can generate test reports
- It can be integrated with many source control systems
- Integration with various artifact repositories is possible
- Direct deployment to production or test environment
- Keep Development / Management team always updated about the progress or status of the builds.
- It can be extended by additional plugins, like building and testing android based mobile applications.

Figure 2 below shows where the Jenkins CI server would take place in the Continuous Integration environment.

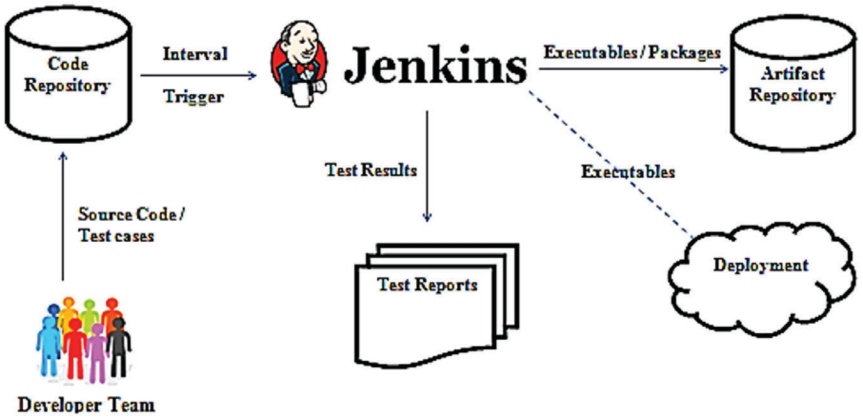

Figure 2: Jenkins Environment

MATERIALS AND METHODS

The scope of this study is to discuss various aspects and issues of Software Release management and to propose a framework to promote Continuous Integration and other advanced tools in order to have better control over software releases. Therefore, the methodology of this study Qualitative, also this study carries out a Hybrid Case study of utilizing Maven, Jenkins and Artifactory to present our hybrid approach in order to achieve Continuous Integration (Forselius & Kakola, 2009; Holck & Jørgensen, 2003; Van Der Storm, 2005). The following section presents an approach to setup the Continuous Integration environment utilizing Maven, Artifactory and Jenkins, as shown in the figure 3 below:

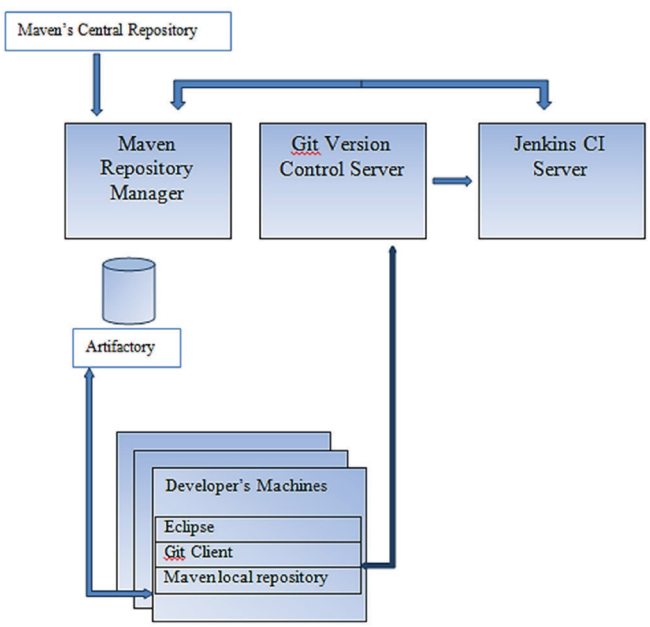

Figure 3: Hybrid Approach for Continuous Integration

Table 1 below shows the phases to achieve continuous integration in a Maven based project. Since this study focuses on the continuous integration and automated build process, therefore the section below only highlights the configuration part of these tools in Jenkins.

Table 1: Phases to setup CI environment

No.	Phases
1	*Installing and configuration of Artifactory as Maven Repository*
2	*Installing and configuration of Jenkins for continuous integration*
3	*Configuration of Artifactory in Jenkins*
4	*Configuration of a Jenkins Job*
5	*Update project POM File with Maven settings*
6	*Adding "Git" post-commit Hook to invoke the build process in Jenkins*

RESULTS AND FINDINGS

To implement all the six phases, two possible scenarios were tested with valid test case and with invalid test case.

Scenario 1: When the project build successfully

Normally build stability check is carried out by running functional test cases in the project. For this scenario, one sample test case (valid test case) in the project has been added as shown in the Figure 4 below, which results in success upon execution.

```
@Test
public void testAddBook(){

    Boolean actual = true;
    Boolean expected = true;

    Assert.assertEquals(expected, actual);
}
```

Figure 4: Valid test case

When the project is saved and upon committing the code into the "Git" repository, it has been noted that Jenkins job is automatically get started to build as shown in the Figure 5 below:

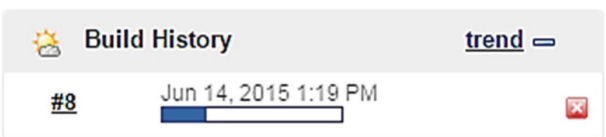

Figure 5: Automated build started

Upon completion of the build process, an Artifactory icon can be seen in right corner of the build number as shown in the figure 6 below, this proves that the build is successfully completed and Jenkins has published it to the Artifactory for future integrations. Here case 1 found successful.

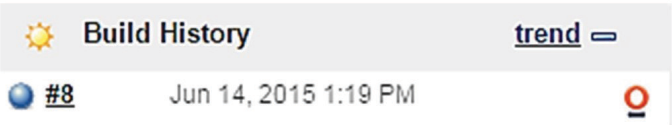

Figure 6: Automated build succeeded

Scenario 2: When something goes wrong in the build

To make the build process failed, the value of actual changed to false (invalid test case) in the same test case used in scenario 1, as shown in the Figure 7 below:

```
@Test
public void testAddBook(){

    Boolean actual = false;
    Boolean expected = true;

    Assert.assertEquals(expected, actual);
}
```

Figure 7: Invalid test case

When the project gets saved and upon committing the code into the "Git" repository, Jenkins job started as expected but this time the result is not same like scenario 1, since the test case is failed stopped the build process, and case 2 found unsuccessful, as shown in the figure 8 below:

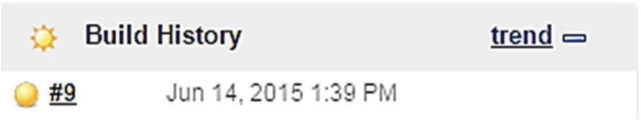

Figure 8: Automated build with failed status

There is no Artifactory icon since the build is unstable and Jenkins has notified the developers via email notifications. Hence it can be easily seen that, in both the scenarios how Jenkins worked by invoking the build process automatically without any extra workload on any team member. The Figure 9 below actually depicts the whole flow in 6 phases:

Figure 9: Continuous Integration flow

DISCUSSION & FUTURE WORK

In this study, an approach has been given to setup continuous integration environment using Jenkins, Maven, and Artifactory. This approach has used configuration as minimum as possible to achieve the final goal. One sample test case is also used to validate the results. There are some other additional features Jenkins is providing with regards to release management and continuous integration, but are out of the scope of this study. It appears that how Continuous Integration incorporates number of software development techniques that will considerably speeds up the release management and delivery process by avoiding unnecessary workload for development, project managers, and other team members as mentioned earlier. Automated build is not software but it is a process to build better software products. It is a matter of one-time setup that enables many benefits to the organization, in terms of better productivity and stable software product release.

The work presented here in this study showed how continuous integration can be setup using open source tools. As a future work, more complexity to this process can be added like if different teams consider different versions of the software build, and the developers are desirable to add third-party components integrations, for example, open source applications. One more interesting concept would be adding variability to the continuous integration environment that can be configured in varieties of ways depends upon the production environment. This study can be a good starting point to elaborate all of these cases.

REFERENCES

Apache Software Project. (2018). Maven – Introduction. Retrieved from https://maven.apache.org/what-is-maven.html

Eliav, R. (2018). The Release Management Process Explained: Optimizing IT. Retrieved from https://www.panaya.com/blog/modern-alm/release-management-process/

Forselius, P., & Kakola, T. (2009, January). An information systems design product theory for software project estimation and measurement systems. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on* (pp. 1-10). IEEE.

Fowler, M. (2006). Continuous Integration. Retrieved from http://martinfowler.com/articles/continuousIntegration.html

Gaston, D., Peterson, J., Permann, C., Andrs, D., Slaughter, A., & Miller, J. (2014). Continuous integration for concurrent computational framework and application development. *Journal of Open Research Software, 2*(1).

Henson, E. (2009). News, Tips, and Advice for Technology Professionals - TechRepublic. Retrieved from https://www.techrepublic.com/blog/tech-decision-maker/release-management-unnecessary-evil-or-holy-grail/

Holck, J., & Jørgensen, N. (2003). Continuous integration and quality assurance: A case study of two open source projects. *Australasian Journal of Information Systems, 11*(1).

Jfrog Artifactory. (2018). Artifactory - Universal Artifact Repository Manager - JFrog. Retrieved from https://jfrog.com/artifactory/

Jyothi, V. E., & Rao, K. N. (2011). Effective implementation of agile practices. *International Journal of Advanced Computer Science and Applications, 2*(3).

Kaur, M. R., Choudhary, M., & Mehta, M. R. (2012). Agile Process: An Enhancement to The Process Of Software Development. *International journal of computer science and network security (IJCSNS), 12*(7), 101-105.

Kawaguchi, K. (2018). Meet Jenkins - Jenkins - Jenkins Wiki. Retrieved from https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

Oracle. (2014). Developing Applications Using Continuous Integration. Retrieved from https://docs.oracle.com/middleware/1212/core/MAVEN.pdf

Scholte, O. (2018). Maven – Password Encryption. Retrieved from https://maven.apache.org/guides/mini/guide-encryption.html

Van Der Storm, T. (2005, September). Continuous release and upgrade of component-based software. In *Proceedings of the 12th international workshop on Software configuration management* (pp. 43-57). ACM.

Vlaanderen, K., Jansen, S., Brinkkemper, S., & Jaspers, E. (2011). The agile requirements refinery: Applying SCRUM principles to software product management. *Information and software technology, 53*(1), 58-70.

Wright, H. K., & Perry, D. E. (2012, June). Release engineering practices and pitfalls. In *Proceedings of the 34th International Conference on Software Engineering* (pp. 1281-1284). IEEE Press.