A review on Continuous Integration, Delivery and Deployment using Jenkins

Arpita S.K[1*], Amrathesh[2], Dr. Govinda Raju M[3]

[1,2]Student, Department of Electronics and Communication, R V College of Engineering, Bengaluru, Karnataka, India
[3]Assistant Professor, Department of Electronics and Communication, R V College of Engineering, Bengaluru, Karnataka, India

[1]asknk99@gmail.com,[2]amrathesh.ec17@rvce.edu.in,[3]govindarajum@rvce.edu.in

Abstract: *Continuous Integration (CI) is the technique of integrating small changes made to the code more often rather than waiting till the end of the development cycle for integration. The software practice where in the software deployment can be done anytime to the market is called Continuous Delivery (CD). With continuous integration and continuous delivery, the problem of taking time to find and resolve the bug can be reduced to a large extent. As the time to find the bugs and fix it gets reduced, many releases adhering to the given timeline can be made by an organization. Various software tools have been developed for the continuous integration process which include Jenkins, Bitbucket, TeamCity. In this paper, a review on the standard practices, approaches, challenges faced while using the continuous integration/delivery in the software development, methods of solving them and using Jenkins for the implantation of continuous integration/delivery is done.*

Keywords: Continuous Delivery, Continuous Integration, Jenkins, master/slave configuration

1. INTRODUCTION

In the software development, Continuous Integration (CI) is a practice, where in the integration of the changes that are made by the developers contributing to the source code is made when the code is committed. The integration is verified with automated building and testing. If the test passes, the build is tested in deployment. If the build succeeds in deployment, it is move onto the production. As this process of integrating the changes, building, testing, deploying is done continuously every time, it is called Continuous Integration/ Deployment (CI/CD). It is summarized in the Figure 1. With the integration made regularly, the patch set causing any build failure can be easily identified. Even though continuous integration doesn't remove the bugs from the source code, it helps in easy identification of the bugs causing for the failure. As a result, the issue fixes can be corrected at a faster rate and does not cause much hindrance for the future work. With this process incorporated, the integration blunder which is usually kept at the end can be avoided. Earlier in Nokia, a procedure called nightly build was implemented wherein the building of the software takes place at every night. With such a large time gap, it was difficult to identify and isolate the errors in the huge codebase. Later the continuous integration was incorporated which helped in faster development process.

Some of the noticeable features of CI are enabling the maintenance of single source repository for the entire project among all the developers, testing the clone in the CI production environment. One of the important feature is availability of current build to all the developers contributing towards the source code. Few reasons to employ CI include, avoiding last minute confusion at release, allowing software developers to work independently on features in parallel and helping with repeated testing. Some of the best practices of using CI systems include committing early and more often, fixing the build failure as early as possible.

The most commonly used tools for CI/CD are Jenkins, Bamboo, and Team City. Jenkins is an open-source CI software written using the programming language – Java. Bamboo is a CI build server which performs automatic building, testing, and releases in a single place. It works along with Bitbucket and JIRA software. It supports many languages and technologies such as Git, AWS etc. Team City is a CI server which supports many powerful features. It maintains a CI server in a healthy and stable state even when none of the builds are running. It helps in providing better code quality for the project.

The rest of this paper examines the Jenkins tool, as well as approaches, obstacles, standard practices, and improvisations utilized in the implementation of CI/CD utilizing Jenkins rather than other CI/CD tools.

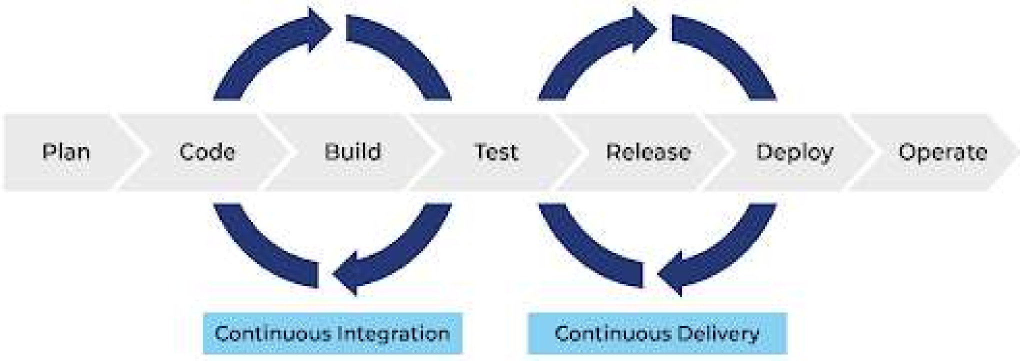

Figure 1 Steps in CI/CD

The most commonly used tools for CI/CD are Jenkins, Bamboo, and Team City. Jenkins is an open-source CI software written using the programming language – Java. Bamboo is a CI build server which performs automatic building, testing, and releases in a single place. It works along with Bitbucket and JIRA software. It supports many languages and technologies such as Git, AWS etc. Team City is a CI server which supports many powerful features. It maintains a CI server in a healthy and stable state even when none of the builds are running. It helps in providing better code quality for the project.

The rest of this paper examines the Jenkins tool, as well as approaches, obstacles, standard practices, and improvisations utilized in the implementation of CI/CD utilizing Jenkins rather than other CI/CD tools.

2. RESEARCH FINDINGS

In [1], 30 approaches and tools have been identified after extensive reviews from about 69 papers. These approaches facilitate the implementation of CI practices that reduce build and test time, detect violations, faults and flaws, support (semi-) automated continuous testing, address scalability and security issues in deployment pipeline and improve reliability and dependability of the deployment process. The list of critical factors, such as good design principles, highly motivated and skilled team, team awareness and transparency, testing (effort and time), customer, application domain, and appropriate infrastructure that must be carefully considered when introducing CI practices in a given company is also determined. Among the reviewed papers 34.7% were validation research types and 36.2% were evaluation research types.

Sriniketan Mysari in [2] has focused on using Jenkins with pipeline methodology for building and integration. The deployment is done using the Ansible tool. As Jenkins is an open source software tool and also is supported with many plugins, it is the preferred tool for the CI when compared to other existing tools such as Bamboo, Team City. Using automation in integration and Ansible in deployment helps to save the time which is one of the major criteria to be followed by the companies. When the Ansible is used for deployment, an easy access to the ssh can be made and also be run on Jenkins node.

In [3], the open source software tool Jenkins is discussed. The design, functionality, installation and usage of Jenkins is spoken about. The Jenkins supports various Source Control Management (SCM) tools such as Rational Team Concert (RTC), Subversion, Mercurial, Perforce and Clear case.

The Jenkins Integration Development Environment (IDE) is highlighted, as well as an examination and comparison of five distinct software integration tools in order to establish their efficacy and usefulness is presented. From the analysis it is suggested that with the help of Jenkins, the critical bugs can be solved quickly and easily when compared to the rest of the tools.

Jenkins has progressed from being a pure Continuous Integration Platform to a Continuous Delivery one. It has adopted the new design trend wherein the release and the process of delivery are automated along with the build [4]. The various challenges that need to be solved for strengthening Jenkins tracking capability is introduced. When transitioning from CI to CD, a few flaws must be addressed, such as versioning of artefacts that must be shippable on a continuous basis and tracing the environment in which artefacts are created. Despite the fact that many effort have been made to solve these issues and various solutions have been proposed, a more thorough investigation is required.

The Jenkins implementation for the integration of software patch and release to the clients is discussed in [5]. With a real-life scenario considered, the advantage of using Jenkins tool for software development in corporate ventures to save the work hours of developers by automating the entire process is provided. Master/slave architecture is used in the Jenkins implementation as shown in Figure 2 where Jenkins server is the master node and Jenkins clients are the slaves. The importance and use of various plug-ins that are available which allow in development are discussed in detail. The automation scripts developed have future scope of expansion.

The work on automatic build repair in a CI system, which includes both the build script and source code, is described in [6]. An empirical analysis of software build failures and build fix patterns is conducted as a first step. Based on the findings of the empirical investigation, a method for automatically correcting build defects using build scripts has been devised. A proposal is being developed to expand this repair methodology to include both source code and build script. It is proposed that a user research be conducted to quantify the automatic fixes, as well as a comparison between the fixes created by the proposed technique and actual repairs.

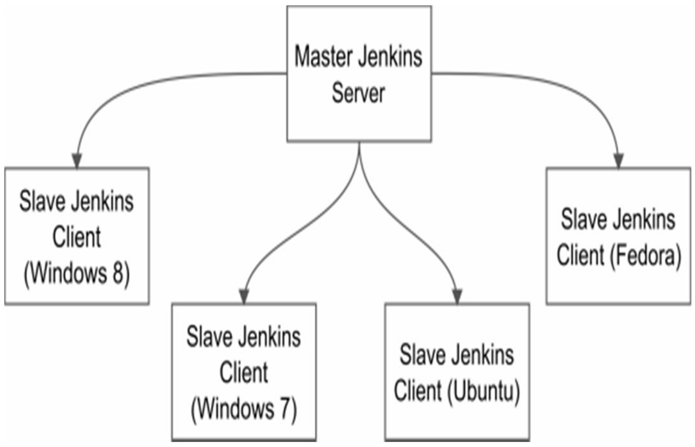
Master Jenkins Server
Slave Jenkins Client (Windows 8)
Slave Jenkins Client (Fedora)
Slave Jenkins Client (Windows 7)
Slave Jenkins Client (Ubuntu)

Figure 2 Jenkins Master / Slave configuration

Manual testing takes time and effort to complete. Manual testing has the potential to miss some types of faults. [7] Proposes an excellent approach for automated testing to address this issue. The suggested framework aids in the automation of test case execution, distribution, and analysis. Tables are used to graphically represent the workflow of test environments and tests. Software development and testing methods can be automated and simplified with the help of this framework. A CI system can be built using the framework by incorporating automated build tools or CI servers. The best practices on automated CI solutions utilizing the suggested framework are presented to provide testers and/or developers with a better understanding of progress and code quality throughout the project lifecycle so that they may focus their time and expertise on more critical, tough topics.

When an integration is done, the code is built and tested. Some of the bugs detected by the testing may be hard for the developers to figure it. Also few times, the time taken to test large code blocks may be very long. To reduce this, the developers will resort to integrating large changes in code which will eventually slow down the integration process. A new testing approach is proposed in [8], in which the testing is done to the small piece of code that is changed rather than testing the entire code i.e. the test is targeted towards a particular part of entire code only. Micro-pipeline is a notion that is introduced. A micro–pipeline is a collection of test blocks, each of which is dedicated to a particular piece of code and performs a certain type of test. When a particular block passes the test, the next test in the pipeline is triggered i.e. sequential execution of the tests is done. As a result when the integration occurs, the user will be given an option to test only the changed part of the entire code block. A scenario of development of a simple calculator (addition, subtraction, multiplication and division) is considered. Each unit is provided with a micro-pipeline which in turn has a number of test suites. The division unit is considered and it is configured with three test blocks normal, abnormal and performance as shown in Figure 3. When the normal test passes, it triggers abnormal and next performance tests. When the code related to the division unit changes, the developer specifies the test suite for testing the changes. The test suite specified in this case is the normal. If the regular test suite passes, the abnormal is tested, and if the abnormal passes, the performance test suite is tested. If the performance test suite fails, the developer is notified, and if the error is fixed, the developer can resume testing from the performance test suite rather than starting from the original test suite, which is customary.

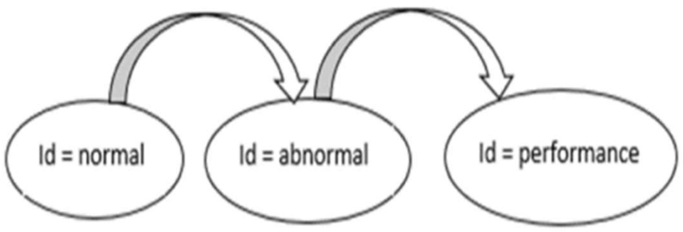

Figure 3 Micro – Pipeline for division unit

3. CONCLUSION

Employing CI/CD in software development is very important as it saves time in finding bugs from a large source code and fixing them. In this paper, the challenges while implementing CI/CD using Jenkins, such as not able to find and correct the bug easily, the preference given to Jenkins tool in the development over others, advantages of using Jenkins, methods to improve the existing method such as using micro – pipeline to reduce the testing time, having techniques to repair the build failure have been discussed. The plug in's that are available with Jenkins, and the faults that has to be addressed while moving from CI to CD is also discussed.

REFERENCES

[1] M. Shahin, M. Ali Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," in IEEE Access, vol. 5, pp. 3909-3943, 2017, doi: 10.1109/ACCESS.2017.2685629.

[2] S. Mysari and V. Bejgam, "Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1-4, doi: 10.1109/ic-ETITE47903.2020.239.

[3] P. Rai, Madhurima, S. Dhir, Madhulika and A. Garg, "A prologue of JENKINS with comparative scrutiny of various software integration tools," *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2015, pp. 201-205.

[4] V. Armenise, "Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery," 2015 IEEE/ACM 3rd International Workshop on Release Engineering, 2015, pp. 24-27, doi: 10.1109/RELENG.2015.19.

[5] N. Seth and R. Khare, "ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development," 2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS), 2015, pp. 1-6, doi: 10.1109/RAECS.2015.7453279.

[6] F. Hassan, "Tackling Build Failures in Continuous Integration," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1242-1245, doi: 10.1109/ASE.2019.00150.

[7] E. H. Kim, J. C. Na and S. M. Ryoo, "Test Automation Framework for Implementing Continuous Integration," 2009 Sixth International Conference on Information Technology: New Generations, 2009, pp. 784-789, doi: 10.1109/ITNG.2009.260.

[8] M. K. A. Abbass, R. I. E. Osman, A. M. H. Mohammed and M. W. A. Alshaikh, "Adopting Continuous Integration and Continuous Delivery for Small Teams," 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), 2019, pp. 1-4, doi: 10.1109/ICCCEEE46830.2019.9070849.