Enhancing the efficiency of continuous integration environment in DevOps

Jetty Benjamin¹ and Juby Mathew²

¹Research Scholar, APJ Abdul Kalam Technological University, Kerala, India
²Associate Professor, Amal Jyothi College of Engineering, Kanjirappally, Kerala, India

E-mail: jettybenjamin@amaljyothi.ac.in

Abstract. Faster release of features to the production environment is the need of the hour of every IT organizations. This faster and continuous software delivery is made possible through DevOps. DevOps is a set of practices that bridges the gap between development team and operations team. The different aspects of DevOps came from agile methodology. Continuous integration is the paramount of DevOps life cycle. In a continuous integration development practice, the development code and the applications code are continuously integrated several times a day. Developers follow the practice of committing changes frequently into the version control system such as git. A change in the version control system triggers the continuous integration system such as Jenkins. This exploratory study presents the ways that support the developers during the build break in continuous integration environment and thereby improve the efficiency of continuous integration process. Our findings contribute that the selected list of plugins in the Jenkins support the developers during the build break, thus developers can come up with faster recovery of build failures which in turn increase the efficiency of continuous integration environment.

1. Introduction

DevOps aims to bring together development and operations. The primary focus of the development team is to bring maximum number of changes that are pushed to production environment whereas the IT operations team aims to make the system stable and available. Adoption of DevOps practice in software companies has made a positive and significant impact. The usage of wide variant of tools accelerated the deployment pipeline activities.[1]. Frequent changes to the production environment take place through the two core processes continuous integration and continuous delivery.

Continuous integration is the process in which the developers continuously integrate or merge the code changes into a shared repository. Continuous delivery is the process in which new changes are released to the customers frequently. Every change that passes all stages of continuous integration/continuous delivery pipeline is released to production. Although the continuous delivery approach increases the delivery of features, there are numerous challenges that needs to be considered during the adoption of DevOps. Categories of problems including build design, integration, testing, release when adopting continuous delivery is addressed and the number of cases per testing problem is found to be greater [2].

When a complete transformation occurs from the traditional software development to continuous development a change will be reflected in every environment such as development, testing, deployment,
operation and maintenance. A set of strategies to overcome the challenges while adopting the continuous delivery is described in [3]. The strategies include convincing the stakeholders about the advantage and benefits of implementing continuous delivery system, establishing a team with experts in different areas and taking a gradual shift to continuous delivery by adding small features, parallelly obtaining the feedback from the team.

Integrating the features continuously will shorten the cycle of release and improves the quality of the software. However, the entire process will come to a halt when there is break in the automated build in continuous integration. Analyzing the build failures, finding the reason for the build failures and quickly fixing the build break is a prior need of continuous integration system.

2. Background and related work
The IT organizations are trying to release small sets of features through a series of release trains instead of big features. The smaller change sets results in lesser complexity. Frequent updation without compromising the quality, results in high customer satisfaction. The collaboration between the development team and the operation team obtained through the DevOps practice helps to create a positive environment.

| Continuous Delivery Features | Stakeholders’ gain |
|------------------------------|-------------------|
| Frequent release of features. | Observe continuous improvement |
| Continuous integration of small fragment of code | Make the changes smaller, thereby less likely to break build. |
| Continuous testing through automated testing tools. | Lesser bugs at the production environment |
| Automated build and deployment. | Ensures repeatable, reliable process and saves a lot of time. |
| Visibility of build status to the entire team | Increased visibility and transparency. |
| Faster feedback loop | Improvement in the quality of application |

The usage of toolchain supporting the continuous integration and continuous deployment practice obtained from 2020 DevOps trends survey.[13]

![Figure 1. Usage of tools supporting DevOps practice](image-url)
Continuous integration involves the process of build automation and code testing every time when a team member commits the changes to the version control. A build breakage can occur due to many factors. To focus our study, we have constructed the following concepts.

- Build failures have a significant impact in the continuous integration environment.
- Build failures increase the cycle time of DevOps.
- Quick responds to the build failures increase the efficiency of continuous integration environment.

2.1 Previous related work

The adoption of DevOps practice has a significant impact on the software quality. There are set of features that contributes to the quality attributes. Automation, collaboration and fast feedback are analysed as the key features to improve software quality. [4]

The anti-patterns or poor practices that affects the performance of continuous integration system is analysed. The implemented method automatically detects anti-patterns. A metric which assess the performance of the continuous integration is established. The practices that improve the performance of continuous integration and the practices that degrades the performance of continuous integration system is identified through the metric.[5]

Build Abstraction and Recovery Tool [BART] was created to analyse the build log, extract the keywords and group the keywords into categories as testing, compilation, code quality etc. BART generates the information that is relevant or that cause the build failure from the build log. It identifies the reason for failure and searches for similar errors on developers’ forums. It provides developers with proposed solutions. Summaries generated with BART increases the understandability of build log.[6]

A method to reduce the cycle time of DevOps by optimizing the test, finding out the test that covers maximum fault detection and minimum run time execution was proposed. Although there are many tests in consideration in a continuous integration, the work is done with integration testing.[7]

Reinforcement learning method to prioritize the test cases in a continuous integration environment was proposed. Test execution results are also used for learning test case prioritization. The method analyses the important indicators for failing test cases. [8]

Test automation plays an integral part in the continuous development and continuous delivery of features. There are several factors that needs to be considered in test automation in a DevOps environment. In addition to the factors such as selection and integration of test tools, the factors including team effort and agile-oriented process helps in the improvement of test automation process.[9]

Continuous testing is the process of running any type of automated test case as quickly as possible in order to provide rapid feedback to the developers and detecting crucial issues before going to production.[10]

3. Methodology

3.1 Research goal and questions

Continuous integration tool Jenkins is selected for the experimental study. The goal of the study was to analyse the developer supporting plugins in Jenkins. These developer assistance tools help to analyse the build failures thereby facilitates the faster recovery of build failure. This process in turn improves the efficiency of continuous integration. Previous related works [5] deals with the working of BART to summarise the build log for easier understandability. This paper analyses the working of test result analyser plugin in Jenkins. Thus, we propose the following research questions:

- What are the categories of build failures?
- Does the build break that results from test failures have a significant impact on continuous integration?
- Does the automated test result analyser improve the understandability of test failures?
- Does the report obtained from the automated system helps to overcome the future issues related to continuous testing?
3.2 Proposed Model

![Diagram showing the proposed model]

Figure 2. Build history of the build failure due to testing shown by the automated test result analyzer

3.3 Implementation

To perform the experimental study, we use the Jenkins 2.249.2 version [11]. Jenkins is a continuous integration tool that facilitates continuous development, continuous testing and continuous deployment of newly created codes. The entire software development process is accelerated through Jenkins. Any code changes in the places like GitHub, automatically triggers the continuous integration process. The code is build, quality of the code is analyzed and the code is tested and it is deployed to production environment. Jenkins offers more than 1700 plugins. The software pre-requisites for Jenkins are Java 7 or above. We used Java 8.A maven project [12] with automated test cases is selected for the study. Maven is an automation tool for build that is used primarily for Java projects.

Table 2. Maven build lifecycle

| Maven build lifecycle phases | Functions |
|-----------------------------|-----------|
| validate                    | The project is validated to check whether it is correct and all information’s are available |
| compile                     | Source code of the project is compiled. |
| test                        | Compiled source code is tested using unit testing framework. |
| package                     | Package the compiled code in a distributable format. |
| verify                      | Check whether the package is valid and met the quality criteria |
| install                     | Package is installed into the local repository |
| deploy                      | Final package is copied to the remote repository |
Table 3. Software configurations for the study

| Software configurations for the study | Version |
|--------------------------------------|---------|
| Plugins installed in Jenkins for performing the study |         |
| Git                                   | 4.3.0   |
| Maven Integration Plugin              | 3.8     |
| Test Result Analyzer Plugin           | 0.3.5   |
| Software's installed in the local system |         |
| Java                                  | 1.8.0_271 |
| Apache Maven                          | 3.6.3   |
| System Configuration                  | Windows 10, 64-bit, 4 GB RAM |

We performed the required goal and options “**compile test**” in the continuous integration system for the maven project. The required post build action “Aggregate downstream test results” is also executed to obtain the test report.

**Figure 3.** Goals and options

**Figure 4.** Post build action

**Figure 5.** Build Trend

**Figure 6.** Build status
3.4 Results and Discussions
The obtained test results which are in a tabular tree format helps and assist the users to categorize the results as passed, failed and skipped. We obtain a graphical report of our test execution. The version of the Test Result Analyzer Plugin which we have used to perform the study (version 0.3.5) allow the users to specify the number of builds that are needed to be fetched by the plugin for the reporting purpose. This test results gives a better understanding about the test failures which in turn helps the stakeholders to quickly fix the failure. Continuous testing is an important phase in a continuous delivery process and test failure is one among the major cause of build failure. The cycle time of the continuous integration and delivery process is significantly increased if there is delay in fixing the build. If the developer’s understandability in test results can be improved, then we can significantly reduce the cycle time of continuous delivery process.

![Figure 7. Test Passed and Failed](image)

4. Conclusion
In the continuous integration, the process of continuous testing is followed and when we run a series of test executions, it is a difficult task for the users to move to the each and every build report and analyze the result. The automated test analyzer tool solves the issue by presenting a tabular tree format of the test results, which helps the developers and testers to identify which all tests have passed, which all tests have failed and which all tests have been skipped.

5. Future work
We have discussed the causes of build failures. From the different causes of build failures, we selected and studied the test failures which is one among the cause of build breakage. We worked with a Maven project and obtained the above results. As a future work the test results obtained from the test result analyzer can be further processed to identify the anti-patterns that affects the efficiency of continuous integration environment.

6. References
[1] Lwakatare L E, Kilamo T, Karvonen T, Sauvola T, Heikkilä V, Itkonen J, Kuvaja P, Mikkonen T, Olivo M and Lasssenius C 2019 DevOps in practice: a multiple case study of five companies *Elsevier* 114 217-230
[2] Laukkanen E, Itkonen J and Lasssenius C 2017 Problems causes and solutions when adopting continuous delivery-a systematic literature review *Elsevier* 82 55-79
[3] Chen L 2017 Continuous delivery: overcoming adoption challenges *Elsevier* 128 72-86
[4] Mishra A and Otaiwi Z 2020 DevOps and software quality: a systematic mapping *Elsevier* 38
[5] Vassallo C, Proksch S, Gall H C and Penta M Di 2019 Automated reporting of anti-patterns and decay in continuous integration *Int conf on automated software engineering* (San Diego, CA, USA) IEEE
[6] Vassallo C, Proksch S, Zemp T and Gall H C 2019 Every build you break: developer-oriented assistance for build failure resolution *Springer* 25 2218-2257
[7] Marijan D, Liaaen M and Sen S 2018 DevOps improvements for reduced cycle times with integrated test optimizations for continuous integration *Annual computer software and applications conf* (Tokyo, Japan) IEEE 01
[8] Spieker H, Gottlieb A, Marijan D and Mossige M 2017 Reinforcement learning for automatic test case prioritization and selection incontinuous integration *Proc of 26th Int Symp on*
software testing and analysis (Santa Barbara, CA, USA) ACM 12-22

[9] Wang Y, Pyh´aj´arvi M and M´antyl´a M V 2020 Test automation process improvement in a DevOps team: experience report Int conf on software testing, verification and validation workshops (Porto, Portugal) IEEE

[10] Mascheroni M A and Irraz´abal E 2018 Continuous testing and solutions for testing problems in continuous delivery: a systematic literature review ISSN 221009-1038

[11] https://www.jenkins.io/doc/book/installing/

[12] https://maven.apache.org/

[13] https://www.atlassian.com/whitepapers/devops-survey-2020