Build Optimization Using Jenkins

M. N. Rakshith(✉) and N. Shivaprasad

Department of E&C, Sri Jayachamarajendra College of Engineering, JSS Science and Technology University, Mysuru, India
rmn2907@gmail.com, nagshivu@gmail.com

Abstract. With the advances in technology there are various software models and tools which have been developed as a platform for the validation and testing the framework. With increase in these platforms the developers face huge challenges in the process of developing a new software for their specific product thus Continuous Integration (CI) comes into the picture. CI is a practice which improves the efficiency and lessens the work complexity by integrating their work in a baseline frequently. One such tool that is widely being used for such practices is Jenkins. As a client server model, is used to trigger the build whenever a user check-in into the repository. Jenkins allows to perform this implementation with the use of numerous plug-ins. This work mainly aims at building the gap between the systematic literature survey and proposes a method which acts as an optimal way of solution for the reduced stop and wait time involved in the CI, with the view of optimization of build time. This method optimizes the serving time, code quality and code coverage capability for a developer.

Keywords: Continuous integration · Build optimization · Build time · Jenkins

1 Introduction

With an accelerated growth towards the automated world, the demand for a high-quality software from the organization is also beaming high. This leads to increased competition among the software industries to develop features accomplishing efficient resource utilization and deliver projects with reduced build time. One among the tool that allow an industry to achieve is jenkins which is an open source CI tool [5]. Cloud computing with this view has gained a wider interest in the research area for optimal performance of resource allocation. In conventional software development the resources used by the developers are very expensive which in turn increases the gross resource utilized by the R&D (Research and Development) developers. To provide a measuring platform with rapid visualization for maintaining the monitoring system, CI techniques evolved.

2 Concepts on Continuous Integration and Jenkins

2.1 Continuous Integration

With the increase in the software development requirements, the need for developers to develop the codes satisfying all the scenarios plays a vital role. In this process the code needs to be enhanced and updated periodically [3]. Achieving this flexibility becomes a tedious task when it has to be performed manually, thus CI came into the picture. The ideology behind the CI is to provide developers a platform that can integrate the code into a shared repository such as git hub. The robustness of this ideology are monitored by providing check ins as and when the code is updated, which is verified by an automatic build process. Using this developer can identify the errors and debug them quickly [9].

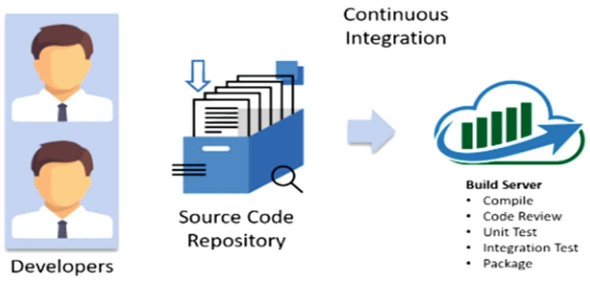


Fig. 1. Continuous integration

Advantages of this CI are

- Providing enhanced communication by increasing the visibility.
- Locate and debug issues in the test bud itself.
- Providing a solid platform.
- Reduces the integration problems which allows developer to deliver software rapidly.

The CI server performs the unit tests and integration tests on the system built and releases deployable artifacts for testing along with assigning a label to each of the updated version as represented in Fig. 1.

2.2 Jenkins

Traceability is the main aspect of developers developing a software which provides flexibility of backtracking to the root cause. Jenkins is a CI tool that allows developers to implement a realistic and complete case study to create an agile environment. This further supports different orchestrate strategies and visualization tools. Plugins are used to perform the CI which acts as a boon for this software. Development life cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis will be integrated using jenkins.

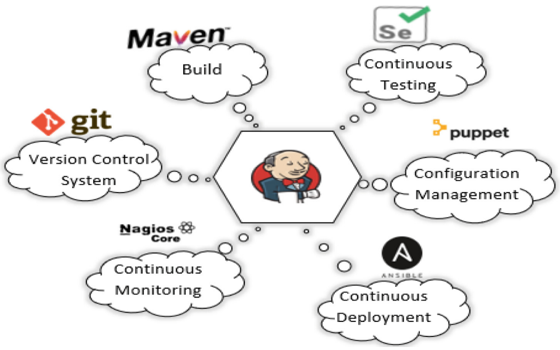


Fig. 2. Jenkins integrating various DevOps stages

Jenkins is mainly chosen for this work because, apart from other competitive tools [11], it supports a number of plugins, integrated to any development and operations (DevOps) stages as shown in Fig. 2, where jenkins is integrated with git, maven, selenium, etc. Some of the plugins that are widely used for this work are

Build Status. This plugin gives an information regarding how many builds are being done at a regular interval of time, be it daily or weekly along with the information about queue and wait time for each build. This enables user to know about the serving time required for each build [13].

Multiple SCM. This provides the user check in and out multiple source control tools that might be one or more repository which overcomes the drawback of SCM that provides only one source control tool at a time [13].

Parameterized Trigger. This plugin provides users to have their input as a variable and use it in run time. This is the most used plugin in dynamic environments where users have lots of options and user-defined values to be used in the build which may keep changing [13].

Perforce. This provides a rapid and seamless path to extract projects into pipeline stages within Jenkins from a perforce plugin. In this context perforce behaves like a repository. This plugin helps to manage the user's CI processes improvising the quality and traceability [13].

2.3 Literature Survey

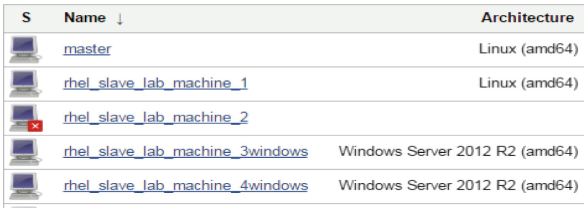
CI has become a DevOps practice used by different communities as CI allows all the team to deploy rapidly changing hardware and software resources [16], related works on our research has been previously done using CI jenkins, build optimization as an important part of CI. In [3], regarding the build, the projects that use CI are more effective at merging requests, they researched their work on several GitHub projects and provided answers for CI usage, costs and benefits. Performance analysis of CI builds for GitHub projects were carried out specifically focusing on JAVA programming language [14]. In [7], the usage of CI has been researched and have acknowledged that, build information from different sources can help developers. In [8, 14], the CI usage in software quality information and scalable test execution using CI has been discussed. A primitive part of CI is built system, either way researchers tried to improve the performance of builds [6] and enhancing the dependency retrieval [10]. So, researchers have proposed several optimizing methods for builds [4] and also discussed efficient way of running test cases [1].

3 Methodology

This section briefs out about the formulation and proposed design for the optimization of build time. From our research build time can be optimized using full build, parallel build and incremental builds. We discuss the need for CI and how to enhance the build quality and optimize build time using open source tool Jenkins and by making efficient use of available Jenkins plugins. The design formulated in the Fig. 4 gives an overview of the methodology involved in performing the automated continuous integration for the slave node.

3.1 Jenkins Architecture

Jenkins supports master-slave architecture, which is also called Jenkins Distributed Builds. This provides user flexibility of running various environments like LINUX, Windows, Ubuntu, etc., also these distributable builds are capable of running same test cases in parallel on numerous environments. Such a build helps to improve the distributed approach and achieve the desired results instantly. The master node would be responsible for scheduling the jobs, monitoring the slaves, dispatching builds to the slaves for execution [12]. List of slaves configured are shown in Fig. 3.



S	Name ↓	Architecture
	master	Linux (amd64)
	rhel_slave_lab_machine_1	Linux (amd64)
	rhel_slave_lab_machine_2	
	rhel_slave_lab_machine_3windows	Windows Server 2012 R2 (amd64)
	rhel_slave_lab_machine_4windows	Windows Server 2012 R2 (amd64)

Fig. 3. Supervising slave nodes

The slave node monitored by master node will be responsible for the execution of parallel builds. The user can configure the master to run a particular build on any slave or can schedule a build for a particular time. Each slave as a number of executors where one slave can run multiple jobs in it.

3.2 Implementation of Jenkins Jobs

Before jumping into the main approach, let us see the initial steps required for setting up a jenkins job, jenkins server will be installed on master node with the use of manage node, different OS slaves will be created based on platforms, one slave agent can run multiple tasks with the help of multiple executors plugin, when all the slave agents are up and running, master will choose a slave agent where a job can run in an OS definitive platform. Our approach is using maven project, which deals with POM (Project Object Model) file which drastically reduces the configurations. To manage source codes jenkins supports Git, Perforce, Mercurial, Multiple SCM's (source code management) etc. Jenkins allows user to pass parameter to build using parametrized build plugin.

Build steps are included in the scripts, since it is a maven build mentioning the POM file, goals like clean and install is the key part for build. Based on the obtained results from the executed jobs, concerned teams are notified and if the build fails particular person who broke the build will be notified. Above procedure are successfully executed with the help of our scripts, configuration files and with the efficient use of available jenkin plugins. Advancing to the optimization part, this process can be completely automated, but our main focus is to approach a way to reduce the build time.

```
Algorithm 1 Build Script for optimization
Input: Repository URL, POM File
Output: Build Result
 1: if (Code check-in happens) then
 2:     Jenkins server selects a slave platform
 3:     Notify changes from checked in repository
 4: end if
 5: if (Repo server not running on slave) then
 6:     Install required modules on slave machine
 7: else
 8:     Workflow Start
 9: end if
10: Build and Test the source code
11: if (Test cases Fail) then
12:     Fix Bug and Build again
13: else
14:     Generate report
15: end if
16: Email notifications to concerned teams for failed tests.
17: Build will be terminated
```

As shown in Fig. 4, whenever build is triggered to run the test cases, we need an environment to test these things. From the POM file the required files and software's are copied to the slave agent and run the test cases, this operation is full build, where every time a new job is created full build happens, so whenever a developer check-in build will be triggered. Let us take a use case whenever a build is triggered a clean install happens in other words full build, whenever there is full build it may take more than 3 h to build. To make it efficient the approach is to use incremental build. So, whenever there is a check-in in the same module, we can keep a check, so we can build only that module in other words incrementally building jenkins tasks, the algorithm 1 for build optimization is shown below.

Detailed flow of running jenkins tasks is shown and steps to it are listed below.

1. Initially the process begins with code check-in, if there is no code check-in, tasks will be in the hold state.
2. After successful code check in, Jenkins server selects the appropriate slave platform for which the workflow could be started along with the note on efficient utilization of resource.

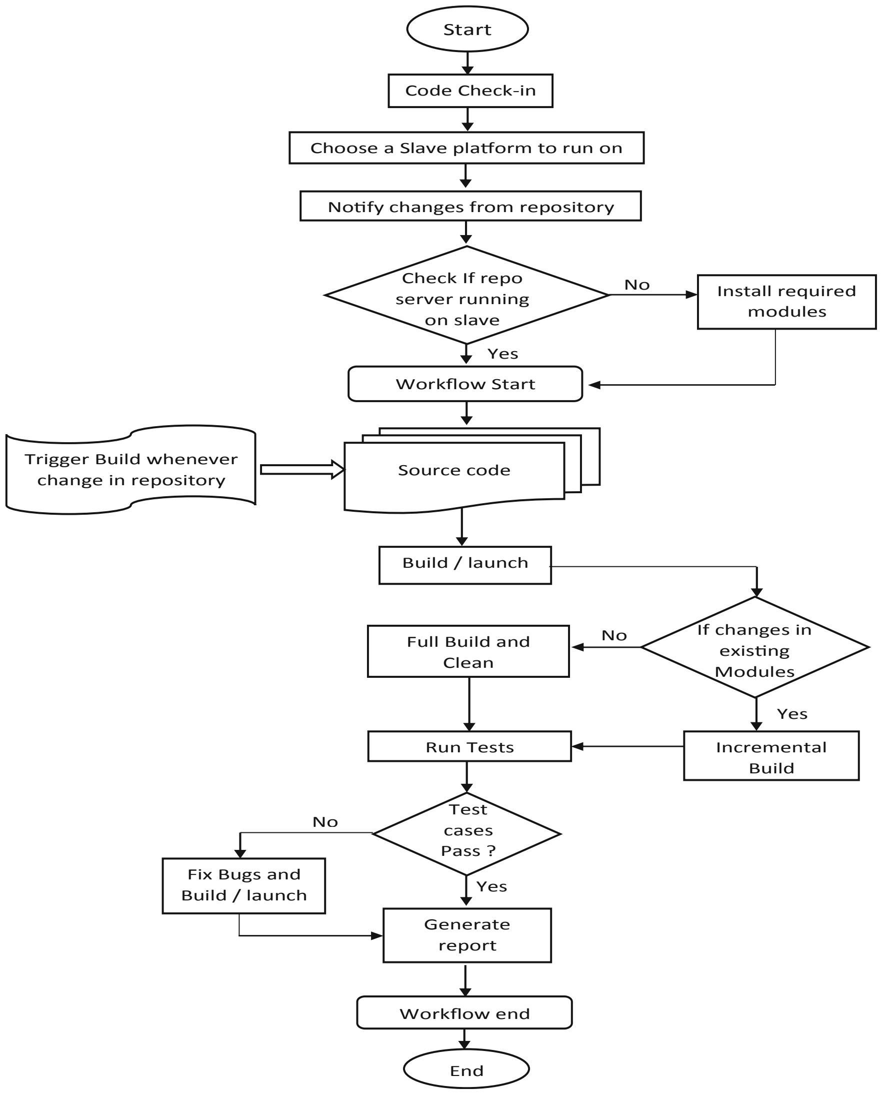

Start
Code Check-in
Choose a Slave platform to run on
Notify changes from repository
Check If repo server running on slave
No
Install required modules
Yes
Workflow Start
Trigger Build whenever change in repository
Source code
Build / launch
If changes in existing Modules
No
Full Build and Clean
Yes
Incremental Build
Run Tests
Test cases Pass ?
No
Fix Bugs and Build / launch
Yes
Generate report
Workflow end
End

Fig. 4. Proposed methodology

3. It is very important to update the changes in the repository regarding the selected slave and the code check-in status.
4. Check whether the repository sever is running on slave platform, if installed execute workflow or else install the repo server and required modules and then proceed to the workflow.
5. Build/Launch and run the Test source code on the slave machine.
6. If the test cases fail, fix the bugs and again reinitiate the build, if the build is successful generate a report.
7. The report consists of all the information about failed and passed test cases, who initiated the build, who broke the build.
8. So, if there is a code check-in in the same module, first the system checks whether there are any required modules to be installed in the system, if not incremental build will be initiated, it is at this stage the build is being optimized with the help our scripts.
9. Step 5, 6 &7 will be repeated again.
10. Terminate the workflow and the build on successful completion of the pipeline.

4 Results and Discussions

The outcome of build optimization and the time expended for list of jobs are shown in the graph, in Fig. 5. From the graph we can get to know about the time duration of number of builds, total build time and different states of builds, yellow coloured part represents the unstable build and the red coloured part depicts the failed scenarios whereas the black coloured line displays total collective build time on each time range. Logs will be available in the build console and a detailed description of all the builds or a particular build will be available. From Fig. 5, in build No. 30, Full build takes around 2 h to build, progressively the build time is getting reduced, these shows our methodology is fully functional. For example, in Fig. 6, build No. 37 took approximately 2 h to build, while build No. 42 which took approximately 1 h, this is due to incremental build of certain modules. So, whenever there is a check-in, perform full build and if there are only few changes in the builded module incremental build is a paramount option.

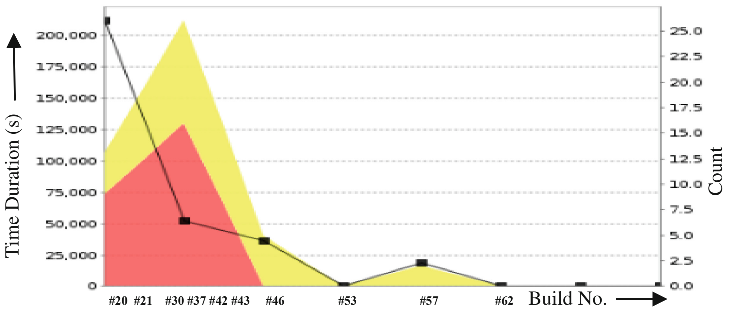
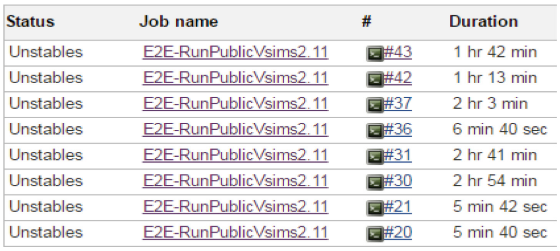


Fig. 5. Build time trend

Also, parallelly building jobs in slaves will save time. There are number of CI tools in the market, since jenkins is an open source tool and availability of efficient plugins makes it as a cost-efficient solution to all CI related problems.



Status	Job name	#	Duration
Unstables	E2E-RunPublicVsims2.11	#43	1 hr 42 min
Unstables	E2E-RunPublicVsims2.11	#42	1 hr 13 min
Unstables	E2E-RunPublicVsims2.11	#37	2 hr 3 min
Unstables	E2E-RunPublicVsims2.11	#36	6 min 40 sec
Unstables	E2E-RunPublicVsims2.11	#31	2 hr 41 min
Unstables	E2E-RunPublicVsims2.11	#30	2 hr 54 min
Unstables	E2E-RunPublicVsims2.11	#21	5 min 42 sec
Unstables	E2E-RunPublicVsims2.11	#20	5 min 40 sec

Fig. 6. Build duration

5 Conclusion and Future Scope

CI is the imperative part for any Software industry. Integration of tasks on a regular basis will optimize developers time, CI helps developers to focus more on key issues and better software quality. Jenkins afford a superior result for these tasks and it's a chosen one for CI automation. These paper gives an idea about how build time is optimized using Jenkins. The detailed steps and procedures for implementing are discussed and an overview of plugins used for Jenkins are discussed in detail in this paper. Finally, these tasks can be optimized much more in the future, allowing to build/rebuild using any repository and test whenever there is a check-in, the entire process can be automated in future.

References

1. Askarunisa, K., Punitha, A.J., Abirami, A.M.: Black box test case prioritization techniques for semantic based composite web services using OWL-S. In: Proceedings of the IEEE-International Conference on Recent Trends in Information Technology, ICRTIT (2011)
2. Fitzgerald, B., Stol, K.-J.: Continuous software engineering: a roadmap and agenda. J. Syst. Softw. **123**, 176–189 (2017)
3. Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., Filkov, V.: Quality and productivity outcomes relating to continuous integration in GitHub. In: FSE (2015)
4. Celik, A., Knaust, A., Milicevic, A., Gligoric, M.: Build system with lazy retrieval for Java projects. In: FSE (2016)
5. Smart, J.F.: Jenkins: The Definitive Guide. O'Really Media, Sebastopol (2011)
6. Laukkanen, E., Paasivaara, M., Arvonen, T.: Stakeholder perceptions of the adoption of continuous integration: a case study. In: AGILE (2015)
7. Beller, M., Gousios, G., Zaidman, A.: Oops, my tests broke the build: an analysis of travis ci builds with github. Technical report, PeerJ Preprints (2016)
8. Brandtner, M., Giger, E., Gall, H.C.: Supporting continuous integration by mashing-up software quality information. In: CSMR-WCRE (2014)

9. Leppanen, M., Makinen, S., Pagels, M., Eloranta, V.-P., Itkonen, J., Mantyla, M.V., Mannisto, T.: The highways and country roads to continuous deployment. IEEE Softw. **32** (2), 64–72 (2015)
10. Miller, A.: A hundred days of continuous integration. In: AGILE (2008)
11. Seth, N., Khare, R.: ACI (automated continuous integration) using jenkins: key for successful embedded software development. In: 2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS), pp. 1–6 (2015)
12. Online Resource. https://jenkins.io/doc/
13. Online Resource. https://wiki.jenkins.io/display/JENKINS/Plugins
14. Gopularam, P., Yogeesha, C.B., Periasamy, P.: Highly scalable model for tests execution in cloud environments. In: 18th Annual International Conference on Advanced Computing and Communications (ADCOM) (2012)
15. Erdweg, S., Lichter, M., Weiel, M.: A sound and optimal incremental build system with dynamic dependencies. In: OOPSLA (2015)
16. Sampedro, Z., Holt, A., Hauser, T.: Continuous integration and delivery for HPC. In: PEARC 2018, 22–26 July 2018, Pittsburgh, PA, USA (2018)