Continuous Integration

Group D

ENSE 375

https://github.com/NolanFlegel/ENSE375-GroupD

Professor Mohamed El-Darieby

March 14, 2021

University of Regina

Jacob Chapman - 200386601
Nolan Flegel - 200250037
Krupalkumar Patel - 200385434
Renz Rivero - 200377174
Shane Toma - 20039608

Table of Contents

What is Continuous Integration?

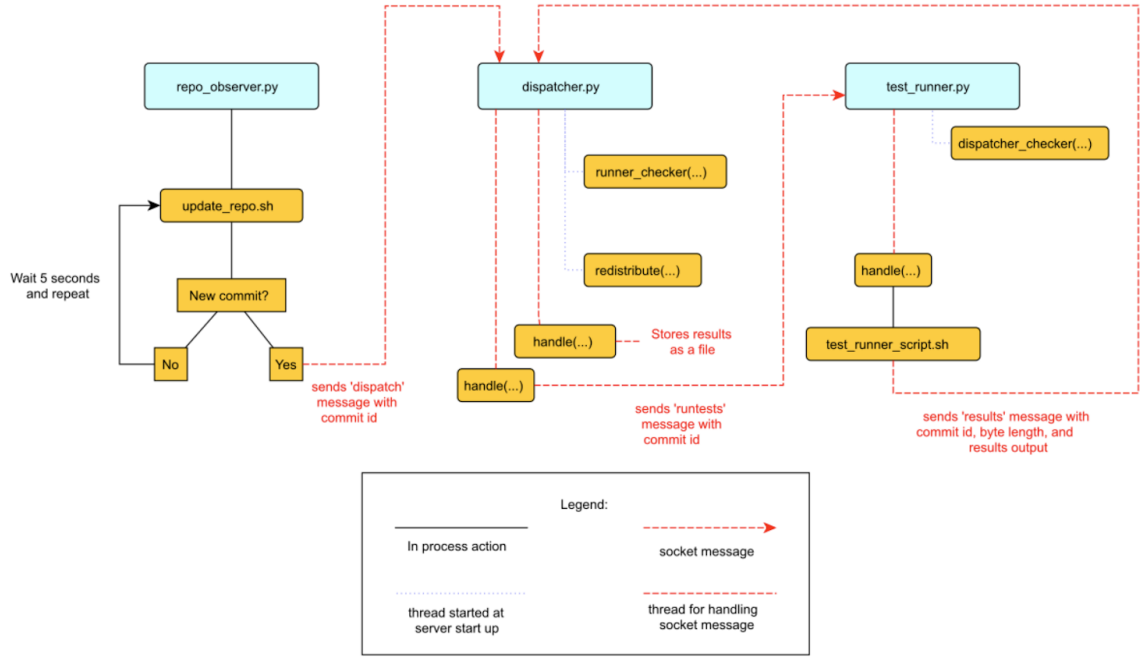
Continuous Integration is a collection of automated tools used to test and verify code changes as they are committed to a repository. The objective of Continuous integration is to assist developers in testing code on a wide variety of systems and platforms, achieving a more efficient workflow than what developers can accomplish on their own.

The minimum components for a Continuous Integration system are: an observer, a dispatcher and a runner. These three components comprise the basic structure and provide an initial architecture to build more complex Continuous Integration systems. The observer watches code repositories for changes and alerts the dispatcher when a commit has been made. The dispatcher is responsible for sending code to various test runners. The runners will test any code sent to them. The Continuous Integration system can operate multiple components and run many tests at the same time. This is a useful feature because it enables developers to make many changes simultaneously and the Continuous Integration system will scale accordingly, allowing all changes to be tested without creating a significant delay.

The Repository Observer is connected to a code repository and watches for code commits. This module is responsible for comparing commits and notifying the Dispatcher whenever tests should be performed. A typical system will trigger on every new commit and have the dispatcher run tests on every commit.

The Dispatcher exists between the Observer and the Test Runners, It listens for requests from the Observer and manages the schedule for any Test Runners, creating as many Test Runners as needed to complete any requests from the Observer. The Dispatcher is responsible for monitoring the tests and relaying the results back to the Observer.

The Test Runner performs tests against code commits sent by the Dispatcher. This module continually communicates with the Dispatcher, regularly reporting its current status. This allows the Dispatcher to manage many Test Runners and only send new tasks when the Runner is available.



What is BuildBot?

BuildBot is a continuous integration open-source tool that consists of two key components in its architecture namely “master” and “worker” which was also referred to as slave. In this system the master is involved directly with the workers to schedule requests and send the build requests.

The framework of BuildBot is quite simple in terms of understanding operation of the system as it follows waterfall methodology. The master is initiated when it detects change in the repositories and it then passes on tasks to the workers thereafter the work is complete the master sends back the built and run test and reports it back to the user. The main advantage of BuildBot is the compatibility, customizability with most of the operating systems available.

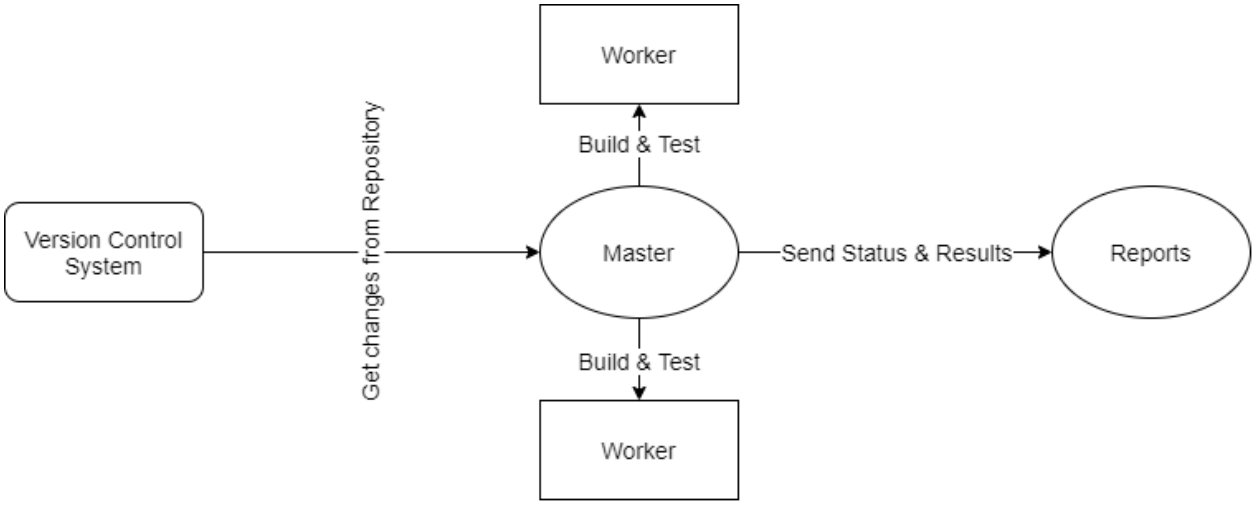


Figure 1: BuildBot Framework

Disadvantage with the architecture of BuildBot is due to centralization the Build workers cannot be attached dynamically to the master. Therefore, to provide build status and results in addition to other services they must be dependent on the master as this task is principally dedicated to its functionality. This also results in an increase of worker load on the master. On the contrary this makes the coordination between the global and local resources easy which is useful for bigger projects and their installation processes.

What is CDash?

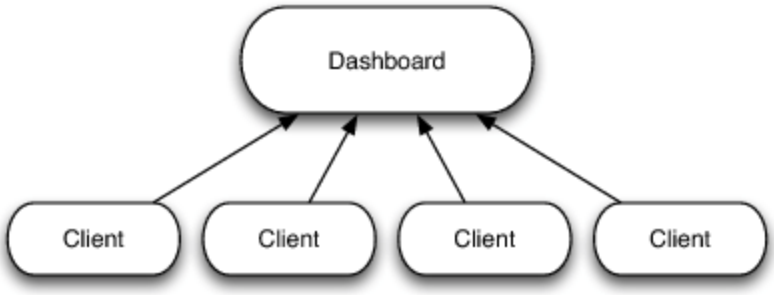


Figure 2: CDash Architecture

Note. Continuous Integration. *Figure 9.3.* From 9.2.2. Implementation Model: CDash

CDash implements a reporting server model that acts as a central repository for data on executed builds with its associated build reports as well as failures, code coverage analysis, and memory usage. The executed builds run remotely on its own schedule and submits its reports in an XML format. These builds are submitted by build clients, non-core developers, and or users that are locally running the published build process on their machines.

The reporting server model is due to the conceptual integration between CDash and elements of the Kitware build infrastructure listed below:

- CMake — build configuration system
- CTest — test runner
- CPack — packaging system

With these mechanisms, Kitware allows a high level of abstraction in a manner similar to a software capable of running under any operating system.

The CDash architecture shown in *figure 1* simplifies aspects of the client-side process of continuous integration due to its client-driven nature. To take into account client-side conditions, the decision to run a build is decided and made by build clients before starting a build. These clients are enabled to build in the central server and can appear as well as disappear as they wish.

The problem with this reporting model is that there is no centralized coordination for the resources and cannot be simply implemented in a distributed environment with questionable clients. Additionally, progress reports are not autonomous and require the server to allow incremental updating of build status plus there is also no way to request a build globally.

What is Jenkins?

Before Jenkins, developers would upload code to a shared repository with irregular commits. The repository would potentially have integration issues, which in turn would have delays in testing. The tests would then notify the developers of any bugs. After testing, there would be a release which would ultimately be delayed as well. Developers had to wait until the entire software code was built and tested to check for any errors.

Jenkins is a Continuous Integration (CI) tool that allows continuous development, test and deployment of newly created code. Jenkins is an open source automation server written in Java. It is used to automate software development processes via CI and facilitates continuous delivery. The Jenkins pipeline starts by having regular code commits, followed by building, testing, and releasing / deploying software to production.

Jenkins Architecture

The Jenkins architecture starts with developers committing changes to source code. The Jenkins server checks the repository at regular intervals and pulls any newly available code. The Build Server builds the code into an executable file. Jenkins then deploys the build application onto a testing server, and feedback is immediately available to developers. If no eros are present, the application is sent to be deployed into production. Figure 1 below illustrates this at a high level.

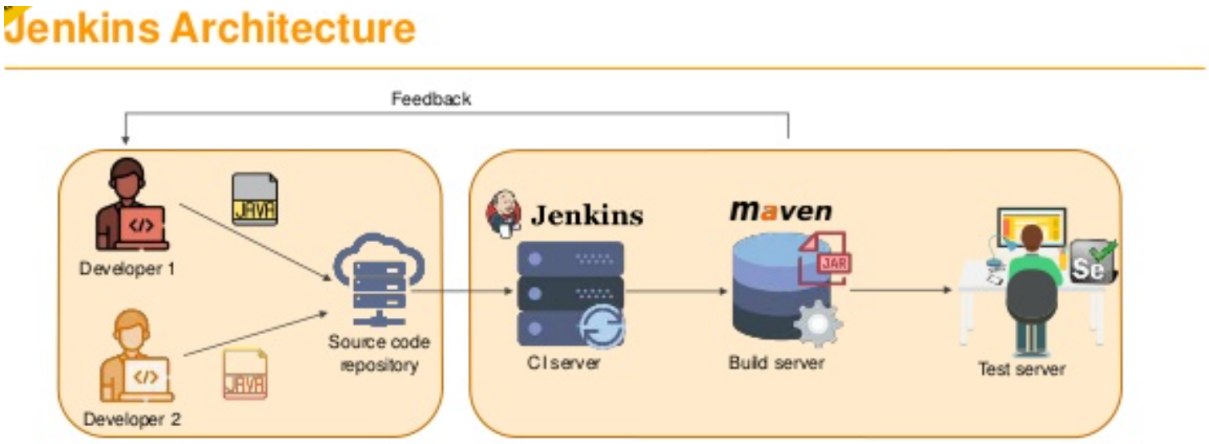


Figure 3: Jenkins Architecture

The Future

The rise of Continuous Integration (CI) systems brought a big leap in the feasibility and ease of using a Continuous Deployment methodology in software releases. Despite the value CI systems have brought, there are still steps to take to improve their performance in the future.

An obvious upgrade for CI systems is the standardization of the different commonly used CI systems. This would involve standardizing a build instruction set as well as a format for reporting. By setting universal standards CI systems would be simplified and become more accessible, a user would only need to learn a single set of commands and a single reporting format and be able to jump between different CI systems with ease. Currently every common CI system has its own unique build configuration language which makes it complicated to jump between different systems (or use more than one at a time). Standardizing a reporting format would make the information in build and test reports easier to read and more accessible to learn, it would also make it more feasible to employ multiple CI systems by making results

easier to compare. Standardization is a challenge for all new technologies, while CI systems are by no means a new technology they have yet to overcome this challenge.

Another obvious upgrade for CI systems is increasing the depth of information collected in reporting. By upgrading the available range of commands for information collection more specific and valuable information could be derived. This could be very convenient for niche applications and test cases.

Besides the obvious directions CI systems will likely move towards in the future there are other more general ways in which they will likely improve. Ease of use is likely to increase over time with the use of user interfaces and increased pre-configured options for novices, this would increase the accessibility of using a CI system. While the number of CI systems available will likely increase some will define themselves as leaders in the industry and change the popularity hierarchy, this means some newer or lesser known technologies will likely change the landscape of CI systems as time goes on. While CI systems have proven themselves to be an excellent tool for developers, the future holds bigger and better things that they will strive to become.

Conclusion

Continuous Integration (CI) systems provide automation for testing code as it is committed to a repository, this provides benefits in efficiency and reduced human error. Technologies that fall under the umbrella of CI such as Jenkins, CDash, and BuildBot provide this automation in different formats best suited to different applications. The lack of uniformity between such technologies while providing a wealth of options to the developer can also make it complex to navigate different tools and systems. This complexity makes it clear there is still room for improvement as CI systems and tools are further developed. In short; CI systems are clearly worth the learning curve to improve the quality of code committed and increase workplace efficiency.

References

Brown, T., & Canino-Koning, R. *Continuous integration*. http://aosabook.org/en/integration.html

Das, M. *A Continuous Integration System.*
http://aosabook.org/en/500L/a-continuous-integration-system.html

Docker, Jenkins
www.jenkins.io/doc/book/installing/docker/

Simplilearn (2018, September 17). *What is Jenkins? | What Is Jenkins And How It Works? | Jenkins Tutorial For Beginners | Simplilearn*. YouTube.