The Current State of CI/CD

Keivan Matinzadeh
Gustav Dowling

April 2020

1 Introduction

CI is an acronym for continuous integration, a software development practice where team members integrate their code frequently. Automated builds are used to verify each integration in order to quickly detect errors [3]. In practice, this means continuously merging the code of different developers to the mainline. Automated builds and tests are ran to verify that the changes do not break the program or produce the wrong output. This allows the developers to continuously, even several times a day, add new features to the project. As the new code is continuously integrated, errors and bugs are also detected more quickly.

CD stands for both continuous delivery and continuous deployment. In the case of continuous deployment, every change that passes the automated tests is deployed to production. When it comes to continuous delivery however; it means making sure that throughout the software's entire life cycle, it is always production ready. This implies that in a matter of minutes any build can be released to users through a fully automated process. The developers are only done with a feature when there is no more testing or deployment phases and the feature is working and in production [4].

2 CI Methods

A key part of being able to utilise continuous integration is automating the build process. Without doing this, a lot of time would be wasted by every developer figuring out all the compilation commands needed for all the different files with their different libraries. One simple example in the Linux environment is make files, allowing a project to be built with one command. For large projects, building may take a very long time. Good CI tooling may therefore allow to only recompile the parts that have been changed, significantly reducing the time needed. Automatic testing should usually be a part of the automated build process, as this makes it more likely that the new code is good.

3 CD Methods

Almost all CD methods rely on the Internet. For simplicity's sake, let's sort software into two broad categories; applications and servers. For applications, new deployments can be made simply by deleting an old binary and replacing it with a new one. For larger applications it is much preferable to use patching, where small changes can be made without the need to download the binary again (assuming that the code is closed source and you can not just recompile it yourself). An update in an application like this may be triggered by opening the application if the application has access to the internet or may be done manually by the user or automatic patches for applications on some operating systems. iOS for example deploys updates every two weeks from a staging server where developers can deploy to as often as they please [1]. For web applications, the main issue that needs to be solved for continuous deployment is handling new requests on the new deployment while also handling the old requests. "Pipelines" is a common term when discussing CD and it simply refers to sending either the code or artefacts through different tool chains.

4 Discussion about Continuous Deployment

There are almost no drawbacks to continuous integration, CI, except the initial time investment as every project benefits from an automated building process. CD, continuous deployment, on the other hand has a few potential drawbacks. For one, it may cause buggier software if a single developer makes a commit, the commit passes the automated testing and the commit is immediately put into production. Without CD the code is more likely to be reviewed by other developers and the application may be tested by a quality assurance team before it goes into production. A solution to this problem that reduces the risk for businesses is feature toggles, dark launches, blue green development, etc. These methods all have the same basic goal, releasing the newer versions of the software only to a subsection of the user base. The idea is that if there are any critical bugs then not all users are affected and the risk is minimised. This solution may increases the complexity of the project, but CI/CD tools such as the ones listed above may make this a fairly simple process to automate. Another issue is that this may cause confusion for users. Users teach each other how to use software, through tutorials or just asking each other. If for example a menu item is moved in an update, but it shows in the old location for some users, the amount of confusion and frustration may be immense as the only thing the person trying to help can reasonably say is "works on my device".

What is good about continuous deployment compared to continuous delivery is that it saves developer time. For large projects, it is common with multiple deployments every day. Automating the deployment procedure not only saves developer time, but also reduces the chances of human error. The steps from the staging server to production only needs to be engineered once, using continuous deployment tools. As an analogy, solving a problem one time in a calm manner

is preferable to having it solved by multiple people many times under time pressure.

5 CI/CD Tools

In this section we will introduce the tools TeamCity, Jenkins, Buddy, and Github Actions. We will describe what they offer and their standout features. We will then compare them to each other to see if that leads to any interesting conclusions.

5.1 TeamCity

TeamCity is a continuous integration server created by JetBrains in 2006. It used by big companies such as Twitter, Ebay and Wikipedia [6]. Features that stand out are gated commits and build grid. A "Gated commit" lets developers automatically merge a branch and run building + testing and then automatically commit the code if it passed, thereby turning a few manual steps into one automatic. Build grid is TeamCity's solution to parallelising the building and testing process, the grid consists of several build agents that each can run one build each.

5.2 Jenkins

Jenkins, originally named Hudson, is a popular, free, open source continuous integration server created by Kohsuke Kawaguchi in 2004. Jenkins gives the developer a more robust and faster way to integrate the build, test and deployment stages. Also, by allowing integration with a multitude of testing and deployment technologies, developers can continuously deliver their software [5]. In Jenkins, continuous integration is achieved by using plugins for the various stages, e.g. Maven for the build stage.

Each commit a developer makes to the source code will automatically trigger a build in Jenkins. The build will test the code and if the build fails the developer will know which commit was responsible. Jenkins is scaleable as it is self hosted, it can be set up to run a near infinite number of builds in parallel given processing power. Another thing to note is that Jenkins is published under a MIT license, and thereby classifies as free software

5.3 Buddy

Buddy is a CI/CD software with docker based tools first launched in 2015. Git platform for web and software developers with Docker-based tools for Continuous Integration and Deployment. It is focused on good performance which it achieves by for example caching in the docker layer.

5.4 Github Actions

Github Actions is a CI solution accessible inside of Github launched in November 2019. It lets the developer setup CI/CD pipelines inside of the repository of the project. The workflow is defined in a YAML file [2]. There are several features available, such as running tests against integration services, and secrets; a feature which lets the user inject variables defined outside of the workflow into the workflow.

5.5 Comparison of the Tools

These tools all offer an automated building, testing and deployment. How they go about this differs. TeamCity for example uses one "build agent" process for each build that is done in parallel, where free users may only run 3 build agents and enterprise users may run more. Jenkins is by far the most used, some reasons for may be the scaleability, the ease of setup, or the vast amount of plugins available. According to their respective websites, Jenkins has 1400 available plugins while TeamCity offers 390 different plugins, this discrepancy is likely due to Jenkins having more users. Buddy does not offer plugin support, but has around 100 "actions" which offer some of the functionality that TeamCity and Jenkins offer with plugins. Buddy is easy to set up if you are also using docker, and also has good integration with git which is the most popular VCS software. Jenkins also has the advantage when it comes to the "network effect", where more users indirectly makes the software better. This may be in the form of maintaining or auditing the source code of Jenkins, making and sharing plugins, creating tutorials or answering questions on forums.

Github Actions is newer and thus its harder to gauge the popularity relative to other tools. One big difference compared to the other tools brought up though, is that Github Actions removes the need for third party applications. This can be beneficial for small teams or lone developers who may want to have everything organised in the same place.

All the discussed tools so far have something in common, they are self hosted and not cloud based, unlike something like AWS CodePipeline.

6 Conclusion

Almost all software projects have the same steps that are needed for something to go from idea into production. It starts with implementing the idea into code, this is the main work of a developer. To test that the code compiles and works as intended the code then goes through a building and testing phase, and if everything looks good developers then share their code with each other using some version control software such as git. It is then sometimes compiled again and sent to a staging server where it awaits deployment to the users. A perfect CI/CD pipeline allows the developer to focus only on turning ideas into code, all the other steps can be done automatically. Continuous integration automates the building and testing phase. CI is fairly easy to implement for most projects

and makes the developers life a lot easier. Continuous delivery only makes sense if there is some form of staging server. It may be worth using it for a project that is pushing to the staging server often. The automation for continuous delivery must also be maintained. Continuous deployment has a similar trade off where it is worth to do if it saves more time in deployments than it loses in implementing and maintaining the automation. Continuous deployment also has a lot of nuances that are worth considering, see section 4. This essay also gives an overview and comparison of a few CI/CD tools.

References

[1] Continuous Delivery at Facebook and Oanda, 2016.

[2] GitHub Actions Documentation, 2019.

[3] Martin Fowler and Matthew Foemmel. Continuous integration, 2006.

[4] Jez Humble. Continuous Delivery vs Continuous Deployment, 2010.

[5] Saurabh. What is Jenkins? — Jenkins For Continuous Integration, 2019.

[6] UpGuard. TeamCity vs Jenkins for Continuous Integration, 2020.