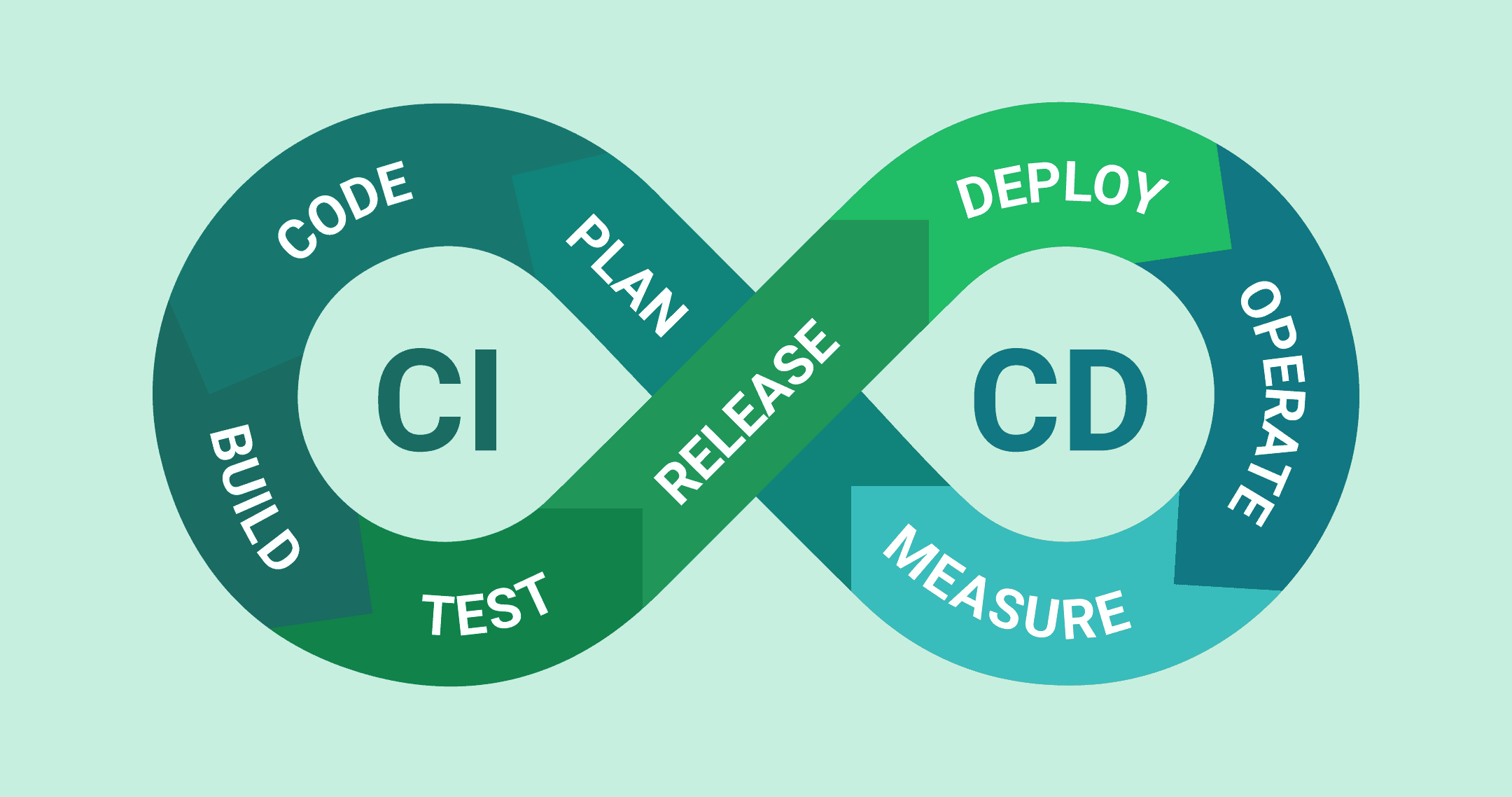
CI/CD Research



Jeroen Koeter

S3-DB01

Table of contents

[What is Continuous Integration (CI)? 3](#_Toc68080782)

[What is Continuous Delivery (CD)? 4](#_Toc68080783)

[CI/CD in Github 5](#_Toc68080784)

[CI/CD in Azure 6](#_Toc68080785)

[List of sources 7](#_Toc68080786)

# What is Continuous Integration (CI)?

Continuous Integration is basically the practice of constantly checking different branches for errors and if they pass all the tests, then you merge them with the main branch. The reason this is important is that you can see that if you add an piece of code someone else wrote that it won’t crash your application. It’s an safety precaution to make sure that everything you make won’t break another piece of code unexpectedly.

Some of the key benefits of using continuous integration will be visible if you work with a bigger team. The biggest of benefits is that you won’t have any so called “Merge hell” moments while making an application. Merge hell is when you have been working with multiple people on the same code for weeks and you all only merge your code with the main branch after these weeks. The problem that will arise at this location is that you’ll get a lot of merge conflicts which will need to be handled individually, if you use continuous integration you wont have these issues. This will be the case because everybody will merge their code each day or multiple times a day, so that if any merge conflict emerge there won’t be a hundred merge conflicts at once.

Another great benefit is that you’ll always have an testable build, for if you want to do some more demanding test than your average unit tests. This build will be ready with the latest code and it should be working right because you test the build each time another merge request comes in.

So in short, continuous integration makes it so that you always have a testable build ready and that you won’t have a “Merge hell” at the end of a work week, but only a couple at the end of the day.

# What is Continuous Delivery (CD)?

Continuous delivery is the next step after the integration process, the continuous delivery makes sure that you code is always in an deployable state after building it. This makes the deployment a lot easier, you can deploy with a click of a button. If you want to have the full benefits of continuous delivery you should also deploy as often as possible, so that you can trouble shoot problems that may arise when you implement small new features. So if you use CI and CD you’ll always have safe tested code that is ready for deployment with the click of a button.

The biggest benefits of continuous delivery are that deployment process is made a lot easier. This will take away a lot of the time that usually goes into deployment. Another benefit is that because the deployment is so much easier you can deploy a lot more often and this means that you can get faster feedback on new features. It’s also a smaller commitment to deploy a small change so your users can experience even the smallest changes and give feedback on these small changes, instead of only being able to give feedback to the big updates.

Another benefit of using both CI and CD is that you’ll have more time to preform higher level tests like, usability and performance for example. This is the case because you won’t have to spent as much time on the smaller tests, because these are now done automatically.

With continuous deliver you can also use things like A/B tester more easily, again this is the case because you can easily make small edits to your software once you get the result of you’re A/B tests. A/B test will in turn also help you make better software for your users.

So in short, continuous delivery makes it so that you can always change your application based on the smallest of feedback from you end users. It makes the deployment process a lot easier so you’ll have time to focus on other aspects of software development.

# CI/CD in GitHub

GitHub supports both CI and CD. There are numerous tutorial on how to use CI/CD in GitHub. GitHub has an separate optional function called [GitHub Actions](https://docs.github.com/en/actions)  which can help you set up and CI/CD pipeline. They offer guides for setting up pipelines with all different kinds of coding languages, from .Net to Node.js to Python. As you can see on the GitHub Actions page they also have guides on how to deploy to different services like how to deploy to Azure app services.

Some of the biggest pros of using GitHub Actions are that it’s free, this is a big pro if you are a student on a tight budget. Another pro is that there is a lot of documentation on how to use GitHub Actions. Another real benefit is that it is fully compatible with GitHub Repo’s, this is kind of a given but it still is a great benefit, this means that you do not need SSH keys for integration in GitHub repo’s.

So in short if you already use GitHub for version control it’s easy to choose for GitHub Actions. The most important reasons for this are that there is a lot of documentation on how to use it and it’s easily integrated with your GitHub repo’s.

# CI/CD in Azure

[Azure DevOps](https://docs.microsoft.com/nl-nl/azure/devops-project/azure-devops-project-aspnet-core) provides an option to set up an CI/CD pipeline on the Azure services, the great thing is that you can integrate you GitHub repo’s into Azure. This might be a little bit of an hassle compared to integrating with GitHub Actions. Azure also has a lot of documentation on how to use it. It has documentation on how to create your CI pipeline and it has different documentation on how to create an CD pipeline. These guides on how to set up the pipelines are there for many different coding languages such as .NET, Node.js, Java just to name a few.

Even though Azure DevOps has a lot of documentation it isn’t ordered in a clear way so it can be quite hard to find what you are looking for form time to time. So if you use Azure repo’s for your version control it’s a smart choice to use Azure DevOps for your CI/CD pipeline because is fully integrated, but if you use GitHub as a version control system you are better off using GitHub actions.

# List of sources

* <https://www.atlassian.com/continuous-delivery/continuous-integration>
* <https://www.youtube.com/watch?v=1er2cjUq1UI>
* <https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html>
* <https://devon.nl/wat-is-continuous-delivery/>
* <https://docs.github.com/en/actions/guides/about-continuous-integration>
* <https://stackshare.io/github-actions>
* <https://knapsackpro.com/ci_servers/github-actions>
* <https://docs.microsoft.com/nl-nl/azure/architecture/example-scenario/apps/devops-dotnet-webapp>
* <https://www.azuredevopslabs.com/labs/azuredevops/github-integration/>
* <https://docs.microsoft.com/en-us/azure/devops/boards/github/connect-to-github?view=azure-devops>