Continuous Integration, Continuous Delivery and Continuous Deployment.

Nicole Carter - ncarter@kth.se

May 2019

1 Introduction

The area of DevOps has grown significantly in recent years. The term derives from the combination of software development (Dev) and the operations surrounding information technology (Ops). DevOps has become the new buzzword within the software engineering industry but what is DevOps? DevOps is a collection of software related practices that aims to improve the overall outcome and delivery of a software product (Nolet, T 2018).

There are many practices within the area of DevOps but in this paper, I will take focus on only a few of them. Continuous Integration (CI), Continuous Delivery (CD) and Continuous Deployment (CD) will be the focus of this paper as the terms are closely related and are at times, often confused with one another.

I am someone who has heard of the terms that will be discussed throughout this paper but have never knowingly been exposed to these practices in the workplace. Therefore, I chose to explore this topic by researching and writing this paper about it to inevitably understand what it means to practise these methods within the development of software.

In the following write up, I will explain what each of the three terms mentioned above mean, how they relate to each other and tools that help in the implementation of these practices.

2 Continuous Integration

I will first begin with Continuous Integration, or simply referred to as CI in the software engineering industry. CI is a workflow strategy practiced by software development teams to ensure everyone's code can smoothly integrate together (GitHub Training & Guides, 2017). This is usually done over

platforms such as GitHub and Bitbucket where separate developers work on separate branches in a repository. CI is implemented in this context by continually merging into the master branch as often as possible.

2.1 CI Server

Before new code can be merged into the master branch, it must be tested to ensure no bugs or errors are being introduced onto the master (Pittet, S). This testing is done automatically, usually with the help of a CI server. This server will test the code and return a paper with the test's results indicating whether or not the code passed the tests. If all tests are passed, the new code will be automatically integrated onto the master branch (ThoughtWorks Products, 2017). If these tests return incorrect values or simply fail, the new code will not be merged and the developer (or someone on the development team) will be notified and the code can be promptly corrected.

2.2 Improvement of Code Quality

As developers are continually pushing their code onto the master branch, there is no final release day madness of everyone pushing and merging all of their branches (Pittet, S). Along with this, as the most up to date and working codebase is always on that master branch and it has already all been tested and validated. The product could be released at any time. CI also makes way for smaller software releases meaning that if an issue or bug is discovered, debugging and finding the cause of the error is relatively simple and fast (Pittet, S).

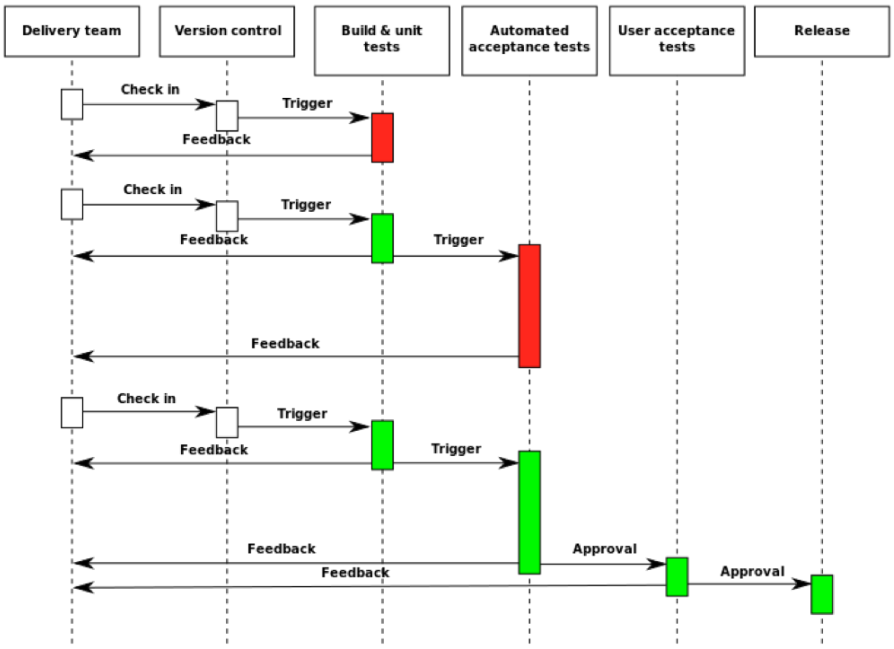
3 Continuous Delivery

The acronym CD covers two different meanings in DevOps; Continuous Delivery and Continuous Deployment. These both have Continuous Integration as a prerequisite and are each explained in detail now, beginning with Continuous Delivery.

Continuous Delivery follows directly on from CI as it goes a step further. Similar to how CI allows for the automation of integration, CD allows for the automation of actually releasing a working, bug free build of the product. It also allows for immediate deployment into the production environment at any time at the click of a button. However, the clicking of that button is not automated yet (Pittet, S). That's next.

3.1 The Pipeline

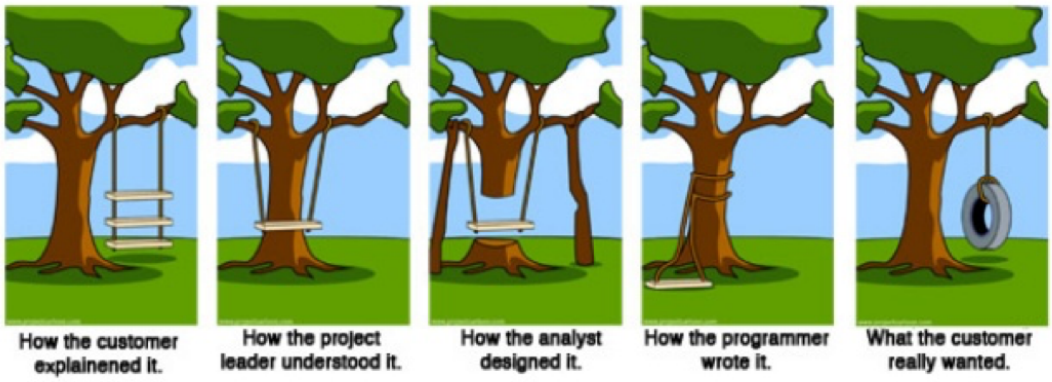
An important aspect of CD is the Pipeline. A Continuous Delivery Pipeline is what all newly developed code passes through before it can be ready for deployment (Pittet, S)(ThoughtWorks Productions, 2017). The Pipeline itself is where all automation, builds and tests are performed (Mukherjee, J). An example of such pipeline is shown in the diagram below.



Code can pass through this timeline at any time, but it is usually at the request of the business side of the product rather than the development or technical side. Having said this, an interval can be set on the codebase to ensure the code goes through this pipeline at any given interval (Pittet, S).

3.2 The Feedback Loop

An excellent benefit of introducing the practice of CD into a software development lifecycle is making quick and reliable releases to the customer. This has many advantages on its own such as giving the customer a working version of the product they paid for as soon as possible (Pittet, S). This allows for a fast feedback loop which will benefit both the development team creating the product as well as the customers who are buying the product. A great diagram that I was first introduced to during my first year of university is this one:



There are many variations of this diagram that introduce more people throughout a project's lifecycle but this one seemed the most appropriate and relevant to our topic. In short, the customer does not know what exactly they want or need from a group of developers. This is where a fast feedback loop can be of assistance. While this diagram may be true for the first iterations of the delivery of a product, the diagram does not have to be true for a final deployment of said product.

3.3 Catching Bugs

While the code does go through extensive automated testing procedures, no code is perfect. If an error or bug was missed during the CI or CD processes that was not present in the previous software release, the continuous delivery of the product ensures that each release is small and therefore if errors were found, they would be resolved clean and quickly (Pittet, S). This is also true for Continuous Integration.

3.4 Changing the Development Style

A final note about introducing CD into a project is that the development style may need to change. As a feature of CD is to release the most recent, working product to the customer, features that are still in development or are just simply not ready to be shown to the customer must be hidden or not accessible from the user side (Pittet, S)(ThoughtWorks Productions, 2017). This is important as the customer wants it to feel like a polished product and showing these incomplete work items will break down that illusion.

4 Continuous Deployment

Once again, Continuous Deployment continues on from the previous automation process of continually delivering. That "button" that is mentioned in the previous section is automatically pressed in this definition of CD. Once new code is committed and delivered, it is almost guaranteed that the code is bug and error free. This means there will be little, if any, downtime of said product (GitHub Training & Guides, 2017).

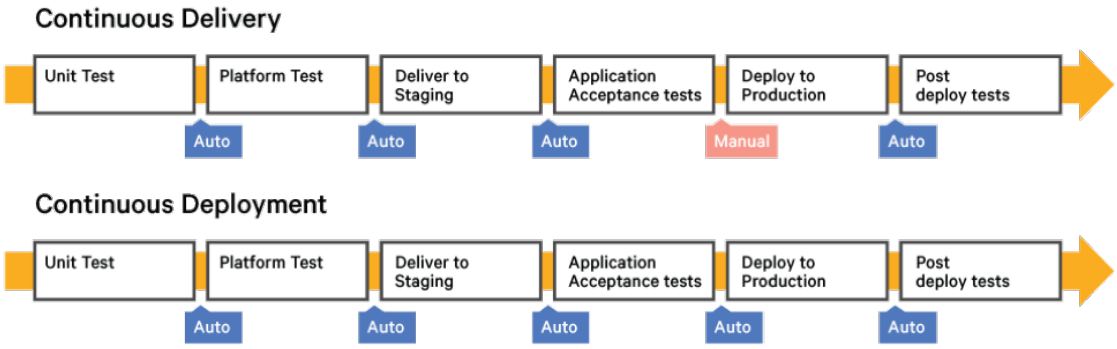
4.1 Further Feedback

Similar with the previous definition of CD we talked about, Continuous Deployment will automatically deploy the software if all tests pass. Zero human interaction is seen from code commit to seeing the product go live (ThoughtWorks Products, 2017). This is great for the development team as they can see their work in a live environment minutes after they have committed that code. It is also another way to further accelerate the feedback loop between development and customers (Pittet, S).

As mentioned previously, not all code is perfect. If an error does slip through, it will be caught and fixed as soon as possible with little to no downtime (Pittet, S). Using Continuous Deployment on a project means automatically creating the by-product of having the actual user as a tester. This user testing can be vital to a product succeeding as even though all the tests passed within the pipeline, it might be awkward for the user to actually use it. The user may also be able to see holes in the product where new features or requirements need to be added (Stackify, 2017).

4.2 Common Confusion

If all of the discussion surrounding Continuous Delivery and Continuous Deployment sounded similar, you're not wrong. It is all very similar which is a main reason why people confuse the two terms so often. This following diagram visualises the exact different between the two practices. You can clearly see why people confuse them. It confused me writing and researching this paper, but this diagram really cleared things up.



5 Tools

Now you know the theory behind these three important aspects of the DevOps culture, but before we get into the specifics of how they relate to each other we will first discuss tooling. We talked about how these approaches rely on automatic processes to test and build the code, and, after all, these approaches do all begin with the word "continuous" so automation is implied as an important aspect of it.

5.1 Cloud-Based

There are many tools on the market that assist in the global lifecycle of CI and CD. There are two types of the tools: cloud (also referred to as SaaS) and on-premise (or self-hosted). Cloud-based tools have the huge benefit of the end user or business not having to think about the maintenance of any hardware or software that the service is required to run on. All of this is handled by the cloud provider. These tools also are easily set up, limiting the potential downtime of changing to a new tool and most have simple integration with version control systems, further adding to the functionality of such tools. As great as these advantages are, nothing is perfect. A cloud solution brings with it scaling issues, specifically from a cost point of view. If teams expand, the cost to host the services will go up as there is more activity in the cloud. Almost all cloud provided tools on the market today have a pay-as-you-go cost model, which is great, but it can also get to the point where it would be a better investment to use a self-hosted tool. It's like renting a house compared to buying a house (Burton, M 2019).

5.2 Self-Hosted

The other type of tool is one that is hosted on the premises in which it is used. These tools all serve the same purpose as the cloud hosted tools, but these are local to where they are being used. A major benefit of a self-hosted tool is its extensibility. Cloud-based tools usually have reasonable extensibility functionality to them, but self-hosted products have a lot fewer limitations on what these extensions could be. One example of this is Jenkins, with over 1,400 plugins available. Along with this, there is often more support available for these self-hosted tools. Support for different platforms and different languages are examples of something that is usually seen more of in self-hosted tools rather than cloud-based tools (Burton, M 2019).

There are reasons why a company may choose to select a self-hosted tool over a cloud-based tool. But there are of course, potential problems that come with a self-hosted service. Most notably, anything that is self-hosted, is self-managed. Any patches, software updates, hardware upgrades etc. must be all managed and maintained by the business or owner themselves. Many of these self-hosted systems also do not offer the same easy set-up procedures and integration with other platforms when compared with the cloud-based tools. This can go even further as to requiring the self-managing of all authorization and authentication around the system (Burton, M 2019).

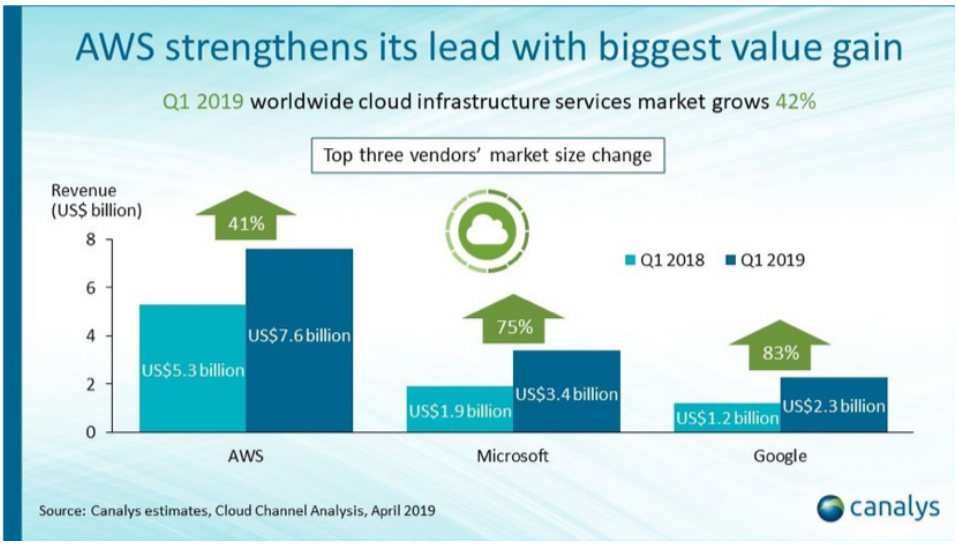
5.3 What is Best?

Both approaches have advantages and disadvantages. As for which one to choose, this choice comes down to what the specific requirements are of the business and what resources can be spared to implement a CI/CD tool. There is much debate over which approach is better due to each business having different needs, so there isn't really a "best" approach (Smith, R 2014).

In the following section, cloud-based tools will be the focus. These are popular within the industry and at a base-case level, the benefits of using cloud-based tools out weigh the disadvantages when compared to self-hosted tools.

5.4 Cloud Market for CI/CD

Within the cloud market, there are three major players: AWS, Azure (formally Visual Studio Team Services) and Google Cloud. While all three of these companies have other services within their cloud infrastructure that are not related to CI/CD, the statistics below still give a good overall picture of the market competition.



As you can see from the image, AWS is way out ahead with US$7.6 billion in revenue in the first quarter of 2019. While the other two are still trailing far behind, they both have made huge increases to their revenue over the past year (Stalcup, K 2019).

Jeff Bezos has said himself that due to the 7-year head start that AWS has over the other market players, the services AWS offers are far more advanced and functionally-rich, and therefore have the majority of the market (Stalcup, K 2019).

To further prove that AWS is the leader, and will probably continue to be for some time, another report conducted by Goldman Sachs stated that the AWS market share came in at 47% with Azure at 22% and Google Cloud at 7% (Stalcup, K 2019).

5.5 AWS CodePipeline

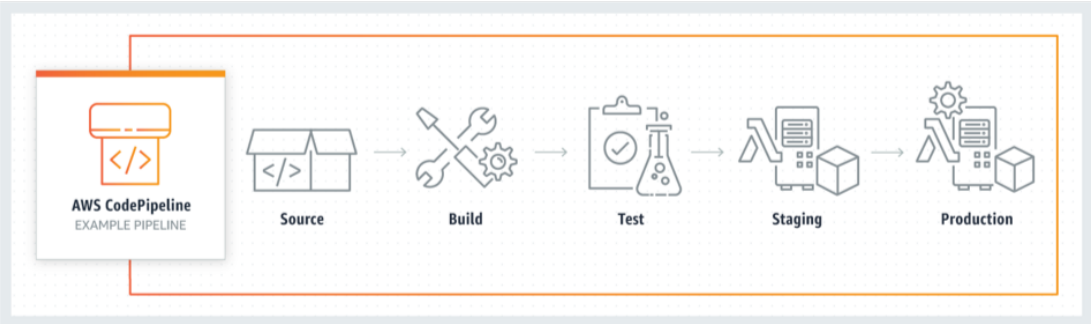
So, after all this discussion of who makes CI/CD tools and what types of tools there are out there, let us now discuss what these tools actually do. Two tools will be used as examples that are both provided by AWS, and are known as CodePipeline and CodeDeploy.

Before tools such as AWS CodePipeline were available, delivering software updates had to be done manually by the developer which resulted in infrequent and often error prone updates. Now, thanks to one of the many AWS services, this is no more. By implementing AWS CodePipeline, delivering the latest software updates has never been quicker and easier. By using the pipeline from this tool the user can visualise the workflow from the release process. This workflow is broken up into different stages including build, test and prepare for deployment. Each of steps can be defined by the user to meet the specific requirements in line with business or project needs. This can help

define exactly where, when and how the code is built, tested and deployed. When this pipeline is activated by the user (ideally activation would happen after all the requirements within the pipeline are defined), every code change that is pushed to the defined repository is automatically pushed through the pipeline and its defined set of instructions and tests as set out by the user. This ensures every change to your code base goes through the exact same procedure every time (Amazon Web Services, 2016)(Amazon Web Services, 2016).

There will be, of course, the occasional error or bug that is pushed to the repository. When this is the case, the pipeline will stop automatically. When and why a pipeline stops can be set by the user and allows for the setting of manual points of revision throughout the pipeline. This entire tool (as with every other CI/CD tool) works automatically and is designed to have the most minimal human interaction as possible. The great thing about this tool is that you can have manual interaction where defined because some things are best left to the human eyes to review rather than a predefined script (Amazon Web Services, 2016)(Amazon Web Services, 2016).

To add to the amazing features that CodePipeline offers, it is also incredibly extensible. A user can slot in their tools of choice for building, testing and deploying from both AWS as well as third party tools. There are also a large number of plugins that can be added to CodePipeline along with the ability to create your own personal plugins specific to your needs. The different parts of CodePipeline are shown in the diagram below. Each of these roles can be performed by any tool, AWS or third-party (Amazon Web Services, 2016)(Amazon Web Services, 2016).



(Amazon Web Services, 2016)

As this is a cloud-based service and AWS's goal in the end is to make a profit, there are costs associated with using this tool. Their pricing scheme is pay-as-you-go based on how many active pipelines one has per month. There are, however, no setup costs or any other monthly fees.

5.6 AWS CodeDeploy

We said in the previous section that a feature of AWS CodePipeline is that a user can simply slot in another AWS tool into the pipeline. AWS CodeDeploy is one such tool that can be used in conjunction with CodePipeline (Liber, A 2018).

Traditional software development would involve manually deploying a product to a large number of servers which is incredibly time consuming as well as the procedure often resulting in errors and therefore downtime of a product. Performing a task like this manually is just tedious and complicated. This is where AWS CodeDeploy can help. CodeDeploy coordinates an application's deployment and updates automatically across many different types of server instances. Compatible instances include other AWS tools such as EC2, Fargate and Lambda. Applications can also be deployed to instances that run on on-site servers. All instances are kept track of and have the ability to send push notifications to a developer or user to receive live updates of any and all active instances. Along with push notifications, with the use of the AWS Management Console or via the AWS CLI, all deployments can be tracked in real-time quickly and easily. If errors are found, deployments can be easily stopped at any time and even have the ability to roll back changes that cause such errors (Amazon Web Services, 2014)(Amazon Web Services, 2014).

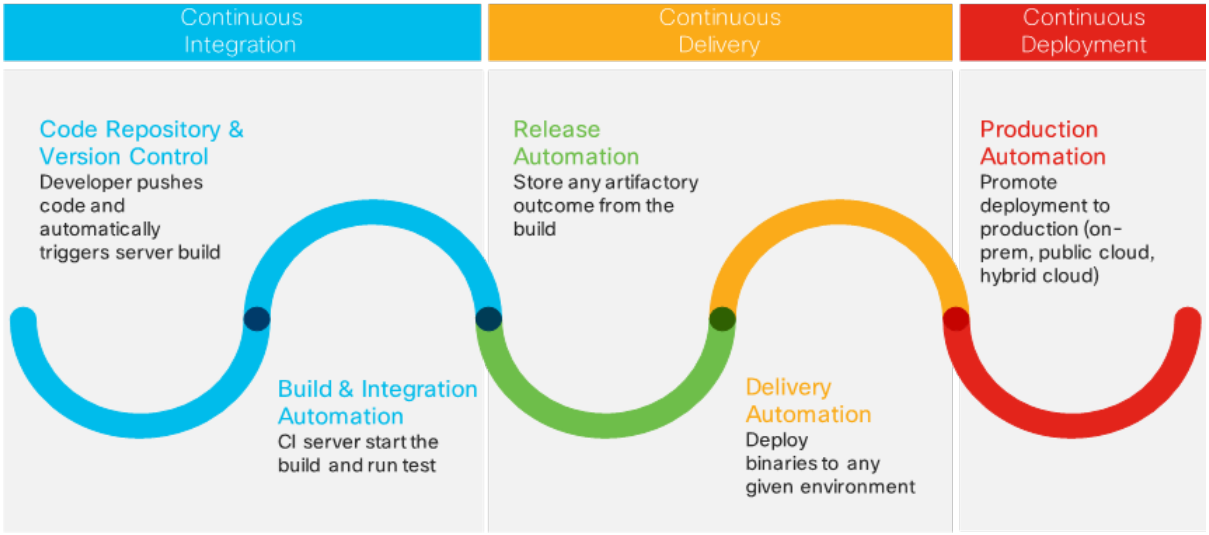
Any application files and deployment scripts work in conjunction with this tool making the adoption of CodeDeploy even easier and stress free for its users. It also seamlessly integrates with any existing CD processes or procedures such as CodePipeline. Integration possibilities are also available for many third-party services such as GitHub and Jenkins to further aid in the seamless process of transferring deployment to this AWS tool (Amazon Web Services, 2014)(Amazon Web Services, 2014).

AWS CodeDeploy keeps the application highly available at all times by introducing all changes incrementally to any avoid possible downtime and frustrated customers (Amazon Web Services, 2014)(Amazon Web Services, 2014).

Finally, there is no additional charge for using AWS CodeDeploy. A user simply pays for the resources that are required to store and run the application. For example, one may need to pay for any EC2 instances being used or active pipelines(Amazon Web Services, 2014)(Amazon Web Services, 2014).

6 CI/CD Relationship

When using CI/CD you can't really have one without the other (unless it's Continuous Integration). To have Continuous Delivery you need Continuous Integration and to have Continuous Deployment you need Continuous Delivery. Due to this, you cannot simply just choose one and be done with it. The following diagram is designed to help visualise the process that has been discussed throughout this report.



(Gioia, S 2018)

As is visualised within this diagram, each step follows on from the previous step. The entire process is triggered at each commit of the code to a version control system and ends with the new code being deployed to a production environment immediately (Gioia, S 2018).

So, after all of this discussion, we now know what all these terms refer to and what they bring to a development lifecycle. But, which one do you choose?

An analysis must be carried out to determine to what extent a project should employ these strategies. Two of the main factors that go into deciding what a project needs are: what the project is itself and what the business needs are. As a general rule of thumb, Continuous Delivery is best in almost all cases and then the decision as to whether Continuous Deployment is required is discussed within the development and business teams (IBM Cloud, 2017).

As an example, the introduction of Continuous Deployment is great for products that the development team has complete (or a large amount of) ownership over. This could be a web application

or some other product that is web-based. The inverse of this would be practicing Continuous Deployment on a product that is required to be installed by an end user. This is just not efficient on the user's end as every update that is deployed would need to be reinstalled by the end user again (ThoughtWorks Products, 2017). Which basically defeats the purpose of Continuous Deployment as end users may not always be using the most up to date version of the product.

7 Conclusion

Throughout this paper a few of the most well-known practices within DevOps were explored and researched. We began by explaining what DevOps meant as a whole and laid down the foundations of why this topic was chosen as the focus for the paper.

This is where the different types of tools are spoken about and what companies are fighting for this market share. Following this, two examples of tools that can be used to achieve a CI/CD workflow were researched and discussed. To end the Tools section, suggestions were offered to the reader in order to help advise them on when and where a business should use certain practices and when they should not use them.

The body of the paper began with discussing the inner practices of DevOps. Explanations were also given as to what each practice is and why they are used. After all of the three practices were laid out on the table, it was time to get to the main event. While CI and CD had been compared to a small extent when each approach was introduced, the main comparison and subsequent suggestions came in at the end.

As stated in the introduction to this paper, I chose to write about this to better educate myself on these practices having heard of them but not actually knowing what they were. Thanks to the research I conducted I was able to articulate and explain in a way that spoke to me. I now believe I have a great foundational understanding of all three practices discussed throughout this paper, and that was my goal from the start.

As a final note, especially regarding a paper that talks about technical procedures, there is only a limited amount of information about a procedure that one can learn from simply reading and watching other people talk about CI and CD. I learn by example and I think I speak for many when I say the most valuable lesson one can learn is actually seeing a practice being, well, practiced.

8 References

Nolet, T 2018, Exploring new frontiers in CI/CD and DevOps, Hackernoon, viewed 4 April 2019, https://hackernoon.com/exploring-new-frontiers-in-ci-cd-and-devops-420b0f9bde53

GitHub Training & Guides, Professional Guides: Continuous Integration Continuous Delivery, video, YouTube, 6 April 2017, viewed 4 April 2019, https://www.youtube.com/watch?v=xSv_m3KhUO8

Pittet, S, Continuous integration vs. continuous delivery vs. continuous deployment, Atlassian, viewed 4 April 2019, https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

ThoughtWorks Products, Continuous Delivery 101 (Part 1), video, YouTube, 24 March 2017, viewed 4 April 2019, https://www.youtube.com/watch?v=HnWuIjUw_Q8

Mukherjee, J, Continuous delivery pipeline 101, Atlassian, viewed 6 April 2019, https://www.atlassian.com/continuous-delivery/pipeline

2019, Continuous Delivery, Wikipedia, viewed 6 April 2019, https://en.wikipedia.org/wiki/Continuous_delivery

IBM Cloud, Continuous Delivery vs. Continuous Deployment, video, YouTube, 1 August 2017, viewed 6 April 2019, https://www.youtube.com/watch?v=hQ0recUXk9o

Bodenheimer, P, Conrad, B 2014, 'Software Development to Help You End Up with the Product You Really Want', The Idea Village, Slideshare, 25 April 2019, https://www.slideshare.net/IdeaVillage/software-development-33087796

2017, Dev Leaders Compare Continuous Delivery vs. Continuous Deployment vs. Continuous Integration, Stackify, viewed 27 April 2019, https://stackify.com/continuous-delivery-vs-continuous-deployment-vs-continuous-integration/

Caum, C 2013, Continuous Delivery Vs. Continuous Deployment: What's the Diff?, Puppet, viewed 27 April 2019,
https://puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff

Burton, M 2019, How to Decide Between a CI/CD SaaS Tool vs. Self Hosted, ParkMyCloud, viewed 29 May 2019,
https://www.parkmycloud.com/blog/ci-cd-saas-tool/

Smith, R 2014, Continuous Integration Servers: Should you self-host or rely on SaaS?, Rainforest, viewed 29 May 2019,
https://www.rainforestqa.com/blog/2014-08-08-hosted-vs-self-hosted-ci/

Stalcup, K 2019, AWS vs Azure vs Google Cloud Market Share 2019: What the Latest Data Shows, ParkMyCloud, viewed 29 May 2019,
https://www.parkmycloud.com/blog/aws-vs-azure-vs-google-cloud-market-share/

Gioia, S 2018, Have You Ever Considered CI/CD as a Service?, Cisco, viewed 29 May 2019,
https://blogs.cisco.com/cloud/have-you-ever-considered-ci-cd-as-a-service

Amazon Web Services, Introduction to AWS CodeDeploy - Automated Software Deployment with Amazon Web Services, video, YouTube, 12 November 2014, viewed 29 May 2019,
https://www.youtube.com/watch?v=Wx-ain8UryM

2014, AWS CodeDeploy, Amazon Web Services, viewed 29 May 2019,
https://aws.amazon.com/codedeploy/

Amazon Web Services, Introduction to AWS CodePipeline - Continuous Delivery on Amazon Web Services, video, YouTube, 25 August 2016, viewed 29 May 2019,
https://www.youtube.com/watch?v=YxcIj_SLflw

2016, AWS CodePipeline, Amazon Web Services, viewed 29 May 2019,
https://aws.amazon.com/codepipeline/

2019, Canalys: battle for enterprise cloud customers intensifies as spending grows 42% in Q1 2019,

Canalys, viewed 29 May 2019,
https://www.canalys.com/newsroom/canalys-battle-for-enterprise-cloud-customers-intensifies-as-spending-grows-42-in-q1-2019

Liber, A 2018, Continuous Deployment with AWS CodePipeline, CodeDeploy and GitHub on EC2, Medium, viewed 29 May 2019,
https://medium.com/@ariklevliber/continuous-deployment-with-aws-codepipeline-codedeploy-and-github-on-ec2-36eeccfb7a5e