# An Introduction to Continuous Integration, Continuous Delivery and Continuous Deployment in the context of AWS

**Authors:** Nicholas Steele: nsteele@kth.se, Helal Uddin: mhud@kth.se

April 27, 2020

# Contents

# 1 Introduction

Software has become an underlying foundation within modern society, shown by use cases within a vast array of industries and through current times of pandemic, by keeping millions of people connected throughout times of disconnect. To deliver software effectively it has become important that businesses implement appropriate processes and practices that enable them to keep up with the relentless pace of ever-changing technology-driven markets.

Modern businesses are faced with large amounts of competition, it has become increasingly difficult to stay ahead of competitors while maintaining a high degree of product quality. To get ahead in modern business, it requires the ability to deliver products at high velocity or as Mark Zuckerberg puts it *"Move fast and break things"*, while maintaining overall product quality. This can be achieved by utilising the correct tools, processes and philosophies.

DevOps is a methodology that allows software to be released into production environments at high speed while maintaining product verification standards. This essay first uncovers Continuous Integration, Continuous Delivery and Continuous Deployment pipelines and essential pipeline practices such as Continuous Testing and Continuous Improvement. After introducing CI/CD the essay moves into practical implementation utilising Amazon Web Services, the worlds largest cloud provider. This is done because choosing relevant technology for implementation can be the difference between project success and failure within a DevOps environment. DevOps practices are constantly evolving with the introduction of new technologies, but the fundamentals remain the same, which is the focus of this paper.

# 2 Background of DevOps

A DevOps team is the product of merging the Software Developers and Information Technology Operations teams. DevOps combines the work these two teams would traditionally carry out separately in methodologies such as Waterfall. The merger of these two teams creates a highly collaborative, fast-paced and highly efficient workflow. The DevOps methodology is highly flexible, not only can Developers and IT Operations combine to create a highly collaborative team, it is also possible to include quality assurance and security experts if required. The fundamental life-cycle that the DevOps team follows is shown below.

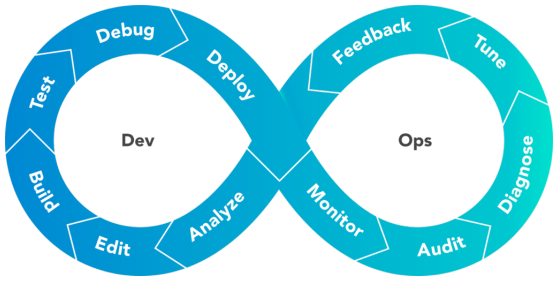


Figure 1: Fundamental DevOps Life-Cycle. *https://www.compuware.com/lifecycle-overview/*

## 2.1 The CI/CD Pipeline

To achieve the life-cycle shown above, DevOps utilises a CI/CD pipeline. A CI/CD pipeline is made up of multiple components that pass code from its initial development stage to the production environment, where it is used by consumers of the application. As code travels further down the pipeline a greater level of confidence that the product will be suitable to run in a production environment is achieved. Automated testing at each major component within the pipeline serves as the gatekeeper for code to progress further down the pipeline. If the built artifact reaches the delivery stage of the pipeline it means that the software has not been rejected by any tests and is likely of a high enough quality for the production environment. If the software does not pass the test present at a stage within the pipeline, it will be rejected, this means it cannot continue to travel through the pipeline. When this happens the relevant development and operations teams will be notified of test failures so errors can be amended and sent through the entire pipeline again. With so many stages of a CI/CD pipeline, it is vital that relevant services and tools are used in order to help achieve high levels of process automation and extensive measures of automated testing, this is discussed later in section 2.5.

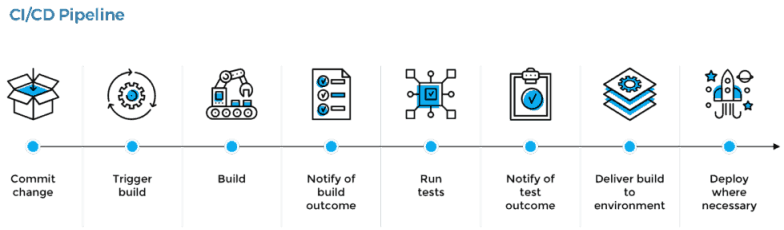


Figure 2: Overview of the entire CI/CD pipeline workflow", *https://lh3.googleusercontent.com/D5grjflbdwdBLjTjEDAtzMQNnhMaq0amkxfJ3ft8sbe5PcMNDaACJTE9yUjamfaqIpJw=s170]*

## 2.2 Continous Integration

Continuous Integration *CI* is the initial stage of the pipeline. A CI server is primarily responsible for combining the code pushed by developers into a Source Code Management System *SCMS*. If code integration is successful then a build stage will compile the code and generate an executable software artifact *e.g* a .jar or .exe file, which will be released to the Continous Delivery section of the pipeline. In order to achieve the maximum benefit from CI, it is ideal that code developed by each team member is pushed multiple times a day. The aim of this constant integration is to reduce the need for medium to large scale code integrations which can be costly as they generally take longer to fix. When small integrations are made daily it is easier to recognise and rectify smaller faults, as opposed to dealing with larger problems capable of halting the workflow for long periods of time.

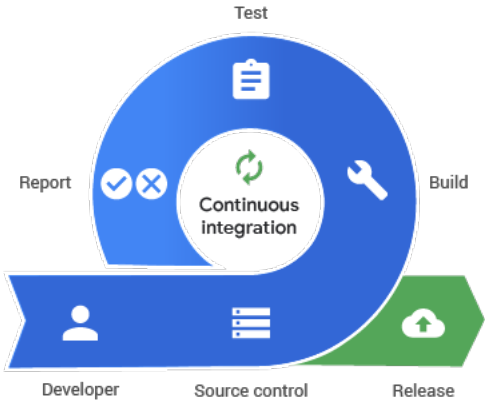


Figure 3: Continous Integration Cycle. *https://www.pagerduty.com/wp-content/uploads/2020/01/image1-1.png*

## 2.3 Continuous Delivery

Continuous Delivery *CD* is designed to build upon the work achieved by the CI phase. The process of CD is graphically depicted in Figure 4, notice that that CD is initiated when it is parsed the executable software artifact. The CD stages typically run on a Deploy, Test and Release cycle to achieve the goal of release into production. Main responsibilities of CD include the provisioning of physical infrastructure so that the application can be deployed onto physical hardware. It is important to recognise that the Continous Delivery practice requires *"humans in the loop"* to determine if the work achieved by the pipeline is a suitable quality for the production environment. If CD has been configured correctly there will always be a new software release waiting to be pushed onto production servers. An efficient CD process is a true exhibition of the DevOps goals, as successful CD can directly impact the speed in which the product can be released to customers, through automating tasks that traditional software development methodologies have required manual configuration in the past. This ultimately lowers risks in regards to deployment failure, as reaching the delivery stage requires passing extensive levels testing implemented within CI process.

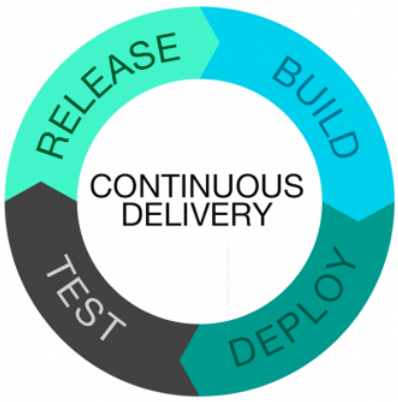


Figure 4: Graphical depiction of Continous Delivery *https://dzone.com/storage/temp/2861664-1.png*

## 2.4 Continous Deployment

Continuous deployment takes Continuous Delivery to the next level by removing the need for manual human intervention to approve the software releases into production, *Graphically shown in Figure 5.* This is achieved through automating the entire pipeline up until its release into production. To achieve Continous Deployment a high level of confidence in the capabilities of the CI/CD pipeline is required. Pipeline confidence is achieved through cutting-edge test automation principals *(discussed below in section 2.5)* and Continuous Improvement strategies *(discussed in 2.9)* . If Continous Deployment is integrated and configured

appropriately, the pipeline will exhibit less downtime than Continuous Delivery methods because Continuous Deployment contains a higher level of pipeline automation. Companies with mature Continous Deployment implementations such as Netflix are capable of deploying changes made by developers into the production environment on the same day. This example is a true exhibition of high-velocity software deployment capabilities through the implementation of Continous Deployment.

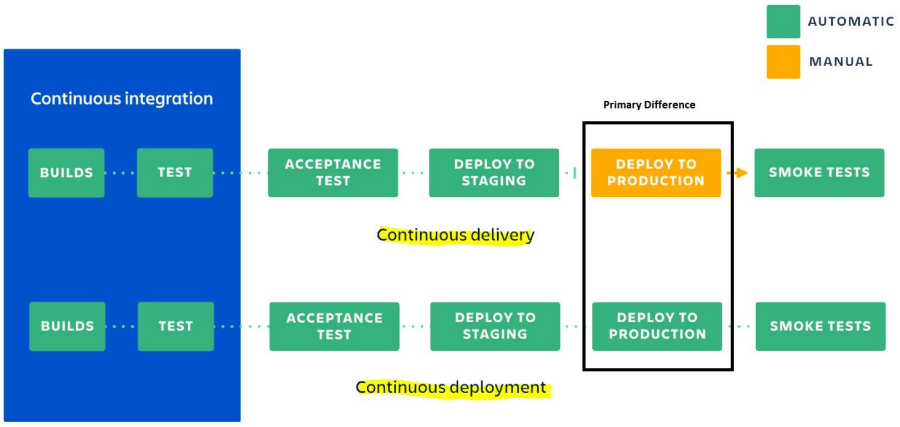


Figure 5: Continous Delivery vs Continous Deployment, *https://wac-cdn.atlassian.com/dam/jcr:7083182c-613d-4645-853e-2f2939556fb1/cicd-CIvsCIvsCD.png?cdnVersion=kw*

## 2.5 Continous Testing

It has been made clear that test automation is a major contributor to a successful CI/CD pipeline. Continuous testing essentially helps the team understand the business risk associated with releasing into the production environment. Automated testing is essentially the protection mechanism used in fast-paced development environments. Continous testing aims to *"Test early and test often"*. For testing to be effective, the right set of tests must run at the correct time within the pipeline. Testing can be broken into two main categories: Shift-Left Testing and Shift-Right Testing methodologies.

## 2.6 Shift-Left Testing

As the name suggests Shift-Left tests exist to the left-hand side of the CI/CD pipeline, this testing aims to find bugs and fix them as quickly as possible so that the developers can identify and rectify faults made within code. Shift-Left Testing practices usually include System Testing, System Integration Testing and Unit Testing methodologies.

## 2.7 Shift-Right Testing

Right-Shift Testing aims to gather feedback after the development is done. Shift-Right testing is mainly concerned with Continous Experimentation methodologies such as Canary Releasing, Shadow Releasing, Fault Injection Testing, and A/B testing.

Because a CI/CD pipeline is constantly evolving test suites must remain relevant and keep up with pipeline evolution. Test suites must be constantly reviewed and maintained to ensure test cases are relevant to a constantly maturing pipeline. Like traditional software testing methodologies, it is important that the implementation of test cases possess minimal redundancy and achieve high code coverage. Strong Continuous Testing suites will improve the overall code quality and maturity of the entire CI/CD pipeline. Like all other aspects of a CI/CD pipeline, the necessary tools and services must be utilised to support testing and test automation.

## 2.8 Future of Testing within DevOps

In the future, current Continous Testing methodologies may not be able to keep up with ever-increasing delivery time frames and maybe the primary burden for achieving shorter software deployments. Current research focuses on the implementation of Artificial Intelligence within Continuous Testing. Primary methods under investigation include analysing how customers are using deployments and the use of intelligent AI agents that are capable of predicting and imitating end-user usage before they have even done so. This data would then be used in order to generate create test cases for scenarios that have not manifested into a reality.

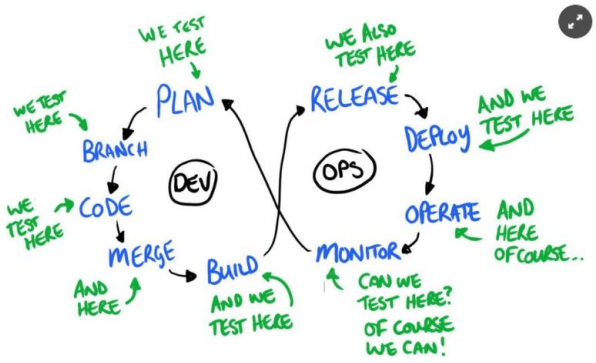


Figure 6: Summary of Continous Testing in DevOps *https://www.blazemeter.com/blog/what-is-shift-right-testing*

### 2.9 Continuous Improvement

A major advantage of the DevOps practice is its ability to improve with each life-cycle iteration. This process is referred to as Continous Improvement and is made possible by using the data and insights uncovered by previous pipeline iterations. To achieve improvement, it is important to understand where the pipeline is at in terms of its capability to deploy software and the overall product quality deployed. Through monitoring and recording operations at each stage of the pipeline, it is possible to recognise pipeline components that are most efficient and contrarily most inefficient, so that both areas can be improved with future life-cycle iterations. To initiate a Continous Improvement process, consider collecting data points such as Overall CI/CD cycle time, deployment frequency, average time spent at each stage of the pipeline, total delivery time and total time taken to recognize faults that occur both within development and production environments.

## 3 Amazon Web Services and CI/CD

Based on CI/CD covered in this essay, a question regarding how is it possible to get practical with CI/CD may arise. This section provides a starting point, to begin with the technical implementation of CI/CD pipelines using AWS technology. AWS services provide highly desirable non-functional quality attributes, such as high levels of service interoperability, modifiability, testability, performance, security, scalability, monitorability. Additionally, AWS provides comprehensive documentation for each service. As established many different components make up a full CI/CD pipeline and AWS has a solution for each pipeline requirement. In almost all cases it would be out of project scope to re-invent the wheel by coding services around CI/CD pipelines, this is mainly done by leading companies with mature CI/CD pipeline such as Amazon, Net-

flix, and Google. Amazon has played a vital role in developing and forming DevOps as a whole, this is why using AWS is a great place to start with CI/CD pipelines.

**A CI/CD pipeline can be configured using the following AWS Services.**

- AWS CodePipeline.
- AWS CodeCommit.
- AWS CodeBuild.
- AWS CodeDeploy
- Cloud Watch/ AWS lambda are honourable mentions.

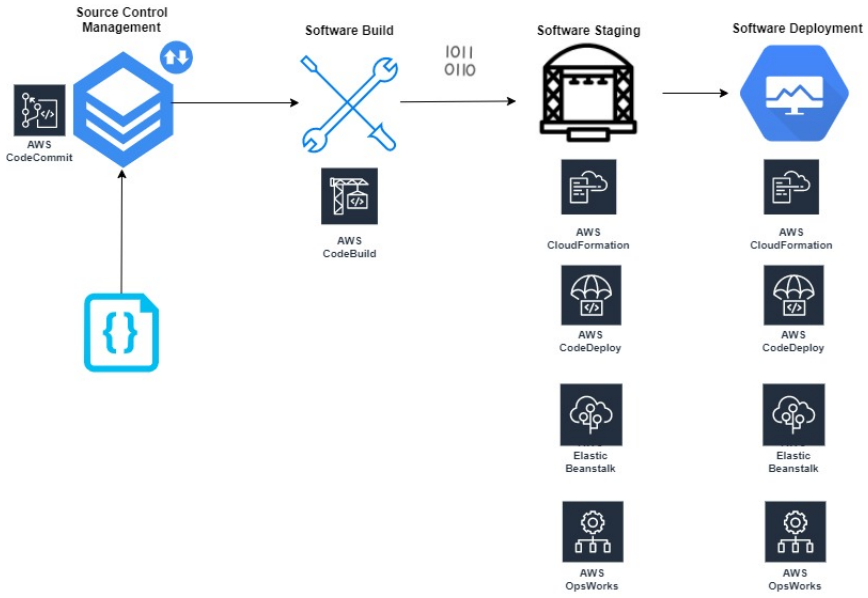


Figure 7: CI/CD lifecycle with relevant AWS Service mappings.

## 3.1 AWS Code Pipeline

Code Pipeline is essentially the wrapper and the enabler of rapid delivery within the entire CI/CD pipeline. With a chain-like structure, its main responsibility is to carry data between the different stages and components of the pipeline. Starting from the SCMS to the delivery of the end-product on the production servers. The Code Pipeline is highly flexible and configurable. For example, it is possible to configure the pipeline to instantiate data transfers every time that there is a state change within the CI/CD pipeline, this is a highly desirable attribute that a CI/CD pipeline must address, as it contributes greatly to workflow automation. Components within the Code Pipeline can be added, removed, interchanged or edited as project requirements change over time.

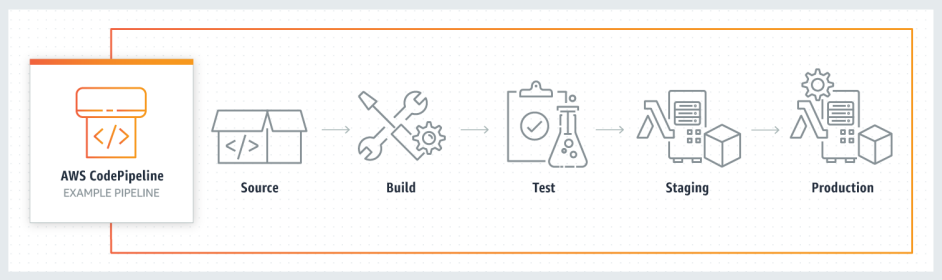


Figure 8: Common pipeline flow between different stages of the CI/CD pipeline *https://aws.amazon.com/codepipeline/*

## 3.2 Code Commit

Code Commit is a leading SCMS service capable of managing the code repositories that make up a DevOps driven project. Section 2.2 *CI* has stressed the importance of implementing highly functional SCMS to provide a place for the developers to commit their code so that the Continous Integration phase can begin its work. Code Commit is an advanced tool that comes with desirable quality attributes such as Secure Encryption of stored files, Access Control, Fault Tolerance, High Availability. Code Commit can be integrated with the other AWS DevOps oriented services discussed below. Code Commit is usually the first stage of the AWS CI/CD pipeline, below is a depiction of Code Commit configured as the the first tool in the chain of the CI/CD pipeline, graphically by Figure 9.

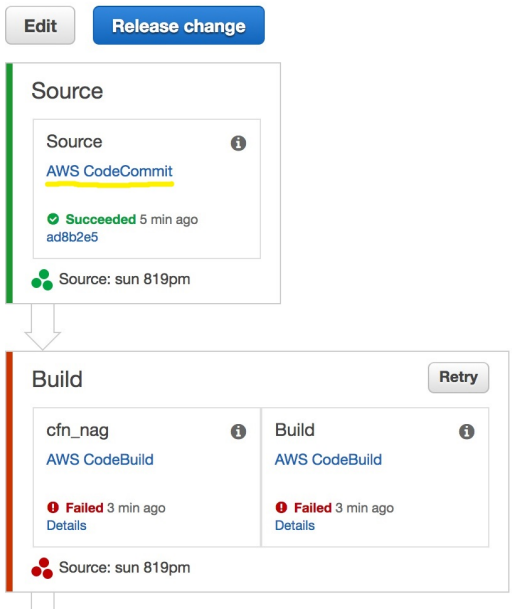


Figure 9: Code Commit configured within the Code Pipeline User Interface *https://stelligent.com/wp-content/uploads/2017/10/codepipeline-failure-1.jpg*

## 3.3 AWS Code Build

Code Build is a fully managed integration service that generates built artifacts. Code build can be configured through a .YAML file, in which it is possible to define configurations related to software installing, build commands and run commands. Additionally, it is possible to define when to build software artifacts. Code Build is capable of conducting branch checks, so that pull requests can be made by other members of the team, resulting in high collaboration. Within a mature CI/CD pipeline, it is possible to speed up the time taken to run builds by using techniques such as parallel software building, where builds are run concurrently and not procedurally. This is all built into the AWS Code Build service and is capable of dramatically decreasing software build times. Additionally Code Build provides a rich analysis service so that relevant feedback regarding software builds can be viewed on-demand.

## 3.4 AWS Code Deploy

AWS Code Deploy is a service that enables an application to deploy to services such as AWS EC2. AWS Fargate and S3 which are common services used for hosting applications in production. Code Deploy is a highly flexible service as it can work with any application file type and development scripts. Code Deploy makes a valuable addition to any CI/CD pipeline as it supports the ability to configure automated deployments into the production environment, because of its automation capabilities Code Deploy contributes to minimizing downtime. Additionally Code Deploy presents a centralized User Interface where deployment instructions are given. Code Deploy is easily integrated into your existing code pipeline.

## 3.5 AWS DevOps Honourable Mentions

As CI/CD pipeline matures and runs a large number of services working in conjunction together, it is common that custom scripts and algorithms are required to carry out tasks unique to a project. This can be achieved using AWS Lambda. Several use-cases of AWS Lamda include performing Security Updates, Automated Backups and File Conversions that may be required as information passes from one component to another.

Cloud Watch is another useful service that enables monitoring and observability capabilities within the CI/CD pipeline. Cloud Watch is a data analytics service that is capable of providing visually appealing data dashboards regarding metrics relevant to your project, some of these metrics include the health of the pipeline. Additionally, it is possible to detect unusual behavior regarding your pipeline, this is built into Cloud Watch and utilizes a powerful Machine Learning engine to make this possible. Another vital feature is utilizing Cloud Watch alarms. Alarms can be configured for events such as a sudden spike of users within the user base, which would require the provisioning of more physical

infrastructure and resultingly come at a cost.

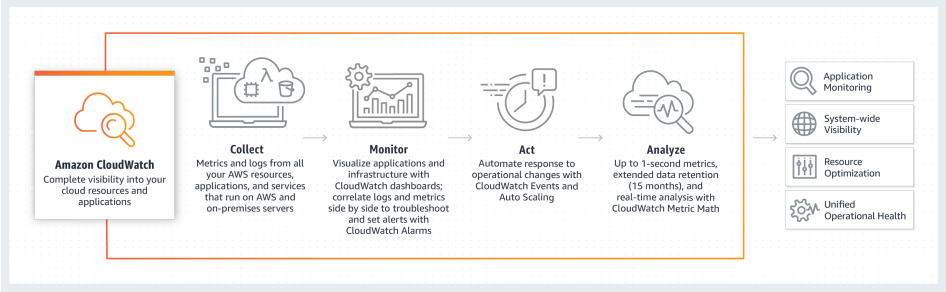


Figure 10: Cloud Watch monitoring the entire AWS Code Pipeline, *https://aws.amazon.com/cloudwatch/*

# 4 Conclusion

This paper has uncovered crucial DevOps practices by analyzing CI/CD pipelines in the context of highly advanced AWS technology. It has been made clear that CI/CD pipelines have provided the technical means for businesses to deploy at high velocity while maintaining quality standards. By introducing the CI/CD pipeline in the context of AWS technology, you are now able to get ahead of competitors by delivering with velocity and confidence through creating a CI/CD pipeline or maturing an existing pipeline. A final note; when getting started with CI/CD, it is important to iteratively increase the maturity of the pipeline. Remember to start small and build up. The process will be slow at the start, but as automation processes begin and the available Amazon Web Services are used to their full potential, the maturity of the pipeline will increase dramatically with time.

# References


[1] Atlassian - What Is Continuous Deployment?
https://www.atlassian.com/continuous-delivery/continuous-deployment

[2] Atlassian - What Is Continuous Integration?
https://www.atlassian.com/continuous-delivery/continuous-integration

[3] AWS Cloud Watch - Application and Infrastructure Monitoring,
https://searchsoftwarequality.techtarget.com/definition/shift-right-testing

[4] AWS Code Build - Fully Managed Build Service 2020,
https://aws.amazon.com/codebuild/

[5] AWS Code Commit - Managed Source Control Service 2020,
https://aws.amazon.com/codecommit/

[6] AWS CodeDeploy - Automated Software Deployment 2020,
https://aws.amazon.com/codedeploy/

[7] AWS Code Pipeline - Continuous Integration and Continuous Delivery 2020,
https://aws.amazon.com/codecommit/

[8] AWS Lambda - Serverless Compute 2020,
https://aws.amazon.com/lambda/

[9] AWS - Practicing Continuous Integration and Continuous Delivery on AWS 2017,
https://d0.awsstatic.com/whitepapers/DevOps/practicing-continuous-integration-continuous-delivery-on-AWS.pdf

[10] Circle CI - A Brief History of DevOps, Part IV: Continuous Delivery and Continuous Deployment,
https://circleci.com/blog/a-brief-history-of-devops-part-iv-continuous-delivery-and-continuous-deployment/

[11] KaiNexus - 6 Principles of the Continuous Improvement Model
https://blog.kainexus.com/continuous-improvement/6-principles-of-the-continuous-improvement-model

[12] Puppet -State of DevOps Report 2019,
https://puppet.com/resources/report/state-of-devops-report/

[13] Search Software Quality - What is shift-right Testing?,
https://searchsoftwarequality.techtarget.com/definition/shift-right-testing

[14] Smartbear - What is the shift left in testing means,
https://smartbear.com/learn/automated-testing/shifting-left-in-testing/

[15] Smartbear - What is the shift left in testing means,
https://smartbear.com/learn/automated-testing/shifting-left-in-testing/

[16] Software Testing Help - Shift Left Testing: A Secret Mantra for Software Success,
https://smartbear.com/learn/automated-testing/shifting-left-in-testing/

[17] Tricentis - What is Continuous Testing?,
https://www.tricentis.com/products/what-is-continuous-testing/

[18] Thought Works – Continuous Integration,
https://www.thoughtworks.com/continuous-integration