Optimize Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment using different techniques and practices

Dileepkumar S R¹, Juby Mathew²
¹ Research Scholar, Lincoln University College (LUC), Malaysia
dileedil@gmail.com
² Faculty of Computer Science and Multimedia, Lincoln University College, Malaysia
jubymg@gmail.com

Abstract. Delivering a software is not a simple process, even if it is a completely new software created from the scratch, the enhancement of an existing software or fixing the bugs of an existing application. As per the new industrial standard most of the companies follow the DevOps methodologies to deliver the software products, which leverage the flexibilities of the software delivery with expected quality on time with minimal errors. One of the most important factors of the DevOps methodology is Continuous Integration and Continuous Deployment. As the entire software industry is moving towards Cloud Computing one of the most powerful, cost effective and easily maintainable environment to host the software applications. There are various Cloud computing providers available in the industry now. Most of them have its own advantage and dis-advantages. There are many researches going on various topics on these areas and the importance of these topics are getting more relevant in the industry now. There are lot of gaps on the Continuous Integration and Continuous Deployment concepts and more improvement options have to be identified and certified throughout the various researches. To understand the importance of this platform, we decided to concentrate the research on the Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment using different techniques and practices. Here our intention is to identify the various factors which helps the industry to do a better Continuous Integration and Continuous Deployment in the Azure DevOps for a controlled Microsoft .NET environment using different techniques and practices. Among the “n” number of factors identified on this subject, only couple of factors are considered as part of this research.

1. Introduction

.NET Framework is a platform developed by Microsoft, which helps to develop and run any type of applications/software. It is a programming model that provides run-time execution environment that manages various applications that target .NET Framework. .NET Framework provides memory management and other system services through Common Language Runtime (CLR). It also helps programmers enable to take advantage of robust, reliable code for all major areas of application development with the support of extensive class library. The .NET Framework also supports multiple programming languages. While creating service based applications that may be a soap based webservices, restful API, microservices or gRPC services in .NET or .NET core framework, the hosting
of these applications could be fully controlled by Microsoft environment which is related to Web Application Servers (IIS, Kestrel, etc) and also as a service in Windows Service. Applications can be hosted independently outside of the .NET environment and fully integrated and operated in .NET framework. It can also be in cloud or on-premise environment. The area being concentrated is more on Azure cloud environment.[1]

DevOps is a set of practice that integrate software development and IT operation teams by working together across the entire software development life cycle. DevOps contains various tools that increases an organization’s ability to deliver applications and services at a faster pace with the support of shorten development life cycle and continuous integration and deployment. It follows Agile methodology. Microsoft introduced Azure DevOps, Software as a service (SaaS) platform that follows the Agile methodology. Azure DevOps helps organizations to manage end-to-end software development and deployment process effectively and easily with the help of leading tools on the market.[2] Azure DevOps is good option for composing the DevOps tool chain.

A CI/CD (Continuous Integration Continuous Deployment) Pipeline is the new culture or practice which industry follows to eliminate the gap between development and operations teams by implementing automating the build process, quality control/testing and finally deployment process of the applications. It provides the smaller code changes options.CI/CD pipeline implementation is considered as the backbone of the DevOps environment in the modern industry, based on the set of principles and culture it follows. The collection of practices in CI/CD pipeline leverages the Development teams to deliver code changes more frequently and reliably. In software engineering, CI/CD is enforcing the organizations to introduce automation in building, testing and deployment of applications and bridges the gaps between operation team and development team and also their activities during the SDLC life cycle. The basic workflow of the CI/CD pipeline is represented in Fig.1.

Fig 1. CI/CD basic workflow representation

2.Methodology

The review work of this paper accommodates the following phases for a systematic review of the literature. Fig.2 presents a specific set of steps followed in the management of this review paper.
2.1 Problem Statement

Identifies the major impact of the above-mentioned drawbacks, corns, issues, pitfalls, difficulties and challenges in the respective Cost Based, Administration and Configuration, Architectural, Development and Technical and Quality Assurance, Manual Testing, Testing Automation and Testing Coverage areas while implementing the Continuous Integration and Continuous Development in Azure DevOps for a controlled Microsoft .NET environment.

2.2 Hypotheses

Suggesting the best techniques, policies and tools which will help to resolve the couple of above-mentioned drawbacks, corns, issues, pitfalls, difficulties and challenges in the couple of respective Cost Based, Administration and Configuration, Architectural, Development and Technical and Quality Assurance, Manual Testing and Testing Automation and Testing Coverage areas while implementing the Continuous Integration and Continuous Development in Azure DevOps for a controlled Microsoft .NET environment. The main research questions are based on the below mentioned respective areas. Cost Based, Administration and Configuration, Architectural, Development and Technical and Quality Assurance, Manual Testing and Testing Automation and Testing Coverage.

2.3 Research questions and Research String

The main purpose of this research is to comprehend the options to be considered to optimize Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment and how they are implemented and evaluated and in what ways or aspects they could be experimented with. To this end, we defined the following research questions:

Q1 What are the most relevant studies about CI/CD in Azure DevOps?
Q2 Which are the possible techniques to be used in CI/CD in Azure DevOps?
Q3 What are the problems and challenges faced in CI/CD in Azure DevOps?
Q4 How CI/CD in Azure DevOps are evaluated?
Q5 Which are the different types of applications to be considered?
Q6 Which directions are most promising for future research?

Based on the above mentioned questions, our research focus points are based on the below mentioned policies.
1) Find out the best tools and infrastructure (based on the cost, performance and easiness of use) to implement the CI/CD pipeline

2) How to improve the coding standard and avoid poor coding related issues.

3) How to make sure all the environments (Development, QA, UAT, Performance and Production) are completely independent and loosely coupled on CI/CD pipeline.

4) How implement and configure a well-connected alert system for the Application exceptions and CI/CD exceptions.

5) How to implement a creative and user-friendly dashboard with respective data matrix.

Continuous integration (CI) and continuous delivery (CD) is a culture, collection of operating principles, and set of practices that support application development teams to deliver code changes recurrently and constantly. In an organization the software engineering team members can integrate their work with cumulative frequency. Continuous delivery (CD) is mainly handling the packaging and deployment that CI covers on the build and test. The overall CI/CD workflow is represented in Fig.3.

![Fig 3. overall CI/CD workflow](image)

The representation of an existing Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment can be found in Fig.4.

![Fig 4. CI/CD in Azure DevOps for a controlled .NET environment](image)

Methods to follow as part of the Research

The need to analyse the existing CI/CD pipelines if any to find out the optimize solutions are mentioned as following. Analyse the existing Infrastructure used for CI/CD pipeline and it’s cost
involved. Analyse the existing coding standard approaches and its drawbacks based on the existing code review comments repositories. Analyse the various environment established and how the CI/CD pipeline configured and how it coupled each other’s. Extract the current matrix and dashboard mechanism and find out the improvement gaps. Analyse the alert mechanism and find out the drawbacks and gaps. Analyse alert data, alert source and prioritise alerts by multiple factors. Analyse test results obtained from unit test, integration test and UAT results in CI system by applying test analytics. The Recommendation Weightage of the options are represented in Fig.5.

![Recommendation Weightage](image)

**Fig 5. Recommendation Weightage**

Most of the IT projects in the industry are developed or developing using any of the Microsoft products. .NET framework and the Azure Cloud platform are wildly using in the industry now a days. .NET is the development platform which helps to create any type of applications such as user interface, services, APIs and microservices, etc. Azure platform is mainly using for hosting the applications, backends and the respective deployments. The percentage of expecting optimization benefit chart is represented in Fig.6.

![Expecting Optimization Benefit Level](image)

**Fig 6. Expecting Optimization Benefits**

*Sampling, Data Collection Methods and Data Analysis Methods*

Data sampling is a technique for the statistical analysis, which is used to select, manipulate and analyse a representative subset of data points to verify and identify trends and patterns in the larger data set being examined. The Data collections is planning to gather from the various application development team who use the Azure Cloud and Microsoft .NET platform to develop the software applications. Tools
like Azure Application Insight, Rivery and Azure CI/CD dashboard are used to analyse the huge data to extract the statistical analysis data, used for the research.

3. Literature review

As this Continuous Integration and Continuous Deployment is new technique and introduced recently in the software industries the researches on these subjects are on going processes. As per the information collected from the various sectors the scope of the improvements in this area is wildly open. There are lot of existing drawbacks, corns, issues, pitfalls, difficulties and challenges that are facing in the current Continuous Integration and Continuous Deployment pipeline approaches.[3] There are various options that can be delivered as part of the various researches with problem solving statements.

As part of the research study, Optimize Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment using different techniques and practices, the research is focused on Azure DevOps for a controlled Microsoft .NET environment. The outcomes may can apply to any of the Cloud DevOps methodologies and other non .NET Microsoft environment and also a completely non-Microsoft environment with slight changes in the entire approaches and outcomes. In the cost based approaches, as per the studies of Forrester Research, Inc, 38 percent of infrastructure decision-makers of the organizations that implemented DevOps and their continuous deployment and release automation efforts in 2018, saw revenue growth of 10 percent or more from the prior year.[4] But the same time 25 percent of those that had not adopted DevOps reported only limited comparable growth. The output of the Continuous integration and continuous delivery implementations are the agility that can give them the competitive advantage to win, with higher confidence and faster feedback.[5]

The Administration and Configurations practices are mainly focusing on the tools which can be used to leverage the CI/CD implementations. CI/CD tools can be mainly categorized as Repository and Version Control tools, Build tools, Automation tools, Test Automation tools, and Monitoring tools. Managing the changes and Version Management are defined in the Repository and Version Control Tools. Git has become more popular and widely used version controlling system with CI/CD pipeline due to the open source and distributed architecture support [6], [7], [8]. It also supports TFS too. The outstanding available tools for the CI/CD are Harness Jenkins, TeamCity, Bamboo, Buddy, GitLab CI, CircleCI and TravisCI, etc.

As per the research paper [9], an application’s Architecture can be a key barrier. To better understand the characteristics that make an application amenable to CD, they compared the architectural characteristics of the applications that were easily moved to CD with those that were difficult to move to CD [10], [11]. Characteristics, in the form of Architecturally Significant Requirements (ASRs) [12], was identified [10]. Of those, Deploy ability and Modify ability often shape architectural decisions.

Development and Technical and Quality Assurance techniques is another another practice can be followed. In the early nineties, Booch initially advocated [13] and introduced Continuous Integration (CI) which is a software development practice as part of the extreme programming practices [14], foreseeing frequent builds integrations per day, then incorporated with an automated build machinery often hosted on a separate server. CI servers favors an early discover of integration errors without leaving the burden to developers, by the automated verification process. Over the years, researchers have investigated the CI adoption[11], and studied specific build phases, namely compilation [12] and testing. Besides the usefulness of automated compilation and testing, software projects can greatly benefit the execution of Automated Static Code Analysis Tools (ASCATs) within CI. For example, build breakages due to ASCATs checks can be used to force developers to follow coding standards/guidelines or to make sure source code is kept conform with the intended architecture as the project evolves. Automation testing
tools also helps the testing team to run the regression testing faster and accurate. Sonarqube is leading tools for the .NET framework.

The most least considerable option is the Manual Testing. The CI/CD pipeline always pushes for continuous development, testing, and deployment, it is often inundated with manual testing. Most important issues with manual testing are the delay per iteration, which starts accruing and pushing the deployment back. QA team has to spend more time to do the application testing introducing painfully slow releases that defeat the main purpose of CI/CD and the manual testing is unable to keep up with the dynamic requirements. It is always necessary to run multiple tests depending on the objective of test cases and plans. Someone must identify test cases to conduct these tests, so the manual testing makes process slow. Concussively, the test cycles call for separate test environments that teams will have to build manually, upgrade, and tear down. The effort that goes into mimicking the end-user environment may require multiple permutations and combinations that the team will have to identify, build, and update.

Manual testing is a failing battle in CI/CD so need to elaborate clear advantages of CI/CD pipeline automation testing. Automation testing and Testing Coverage techniques is faster and more reactive feedback loops that continuously test the codes and share feedback within a duration of a few minutes, so it witnesses rapid acceleration. The detection of test procedures, room for parallel testing capabilities and automated cross-browser concurrent testing are useful features which will improve test coverage and reduce the testing time. The testing teams can set up test environments by using the automatic provisioning in just a few clicks! Testing team not even worried about the latest versions of operating systems and browsers because automation tool takes care of those. So, the teams valuable time for recreating the various environments can be easily reduced. Automation testing offers high scalability than manual testing, so development team can always meet the quality and security requirements. While manual testing can be reserved for specific testing types and modules, such as exploratory testing, it is best to automate testing for seamless integration of QA in the CI/CD pipeline. The table below (Table 1) describes the various techniques or areas used to optimize Continuous Integration and Continuous Development in Azure DevOps for a controlled Microsoft .NET environment.

**Table 1. Comparison of Techniques/Area**

| Techniques /Area | Involvement Criteria | Advantages | Disadvantages |
|-----------------|----------------------|------------|---------------|
| Cost Based      | Analysis of existing Infrastructure used for CI/CD pipeline. | Reliable Cost Effective Options. | Resistance forms current custodians. Complete changes in the tools involved. More Training Requirements. |
| Administration and Configuration | Analysis of existing Administration and Configuration Options. | Easy maintenance and Least configuration efforts | Resistance forms current custodians. More Training Requirements. Less optimization expectation. |
| Architectural  | Overall system and implementation architectural. | The most effective optimization options. | Resistance forms current custodians. Technical challenges. Resolution Duration. Cost Involvement. |
### Techniques /Area

| Techniques /Area                  | Involvement Criteria          | Advantages                                                                                           | Disadvantages                                                                 |
|----------------------------------|------------------------------|------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| Development and Technical and    | SDLC Life cycle              | The most effective optimization options. Effortless review process. Easy to incorporate with any type of | Re iterating SDLC life cycle. Complete changes in the tools involved. Initial  |
| Quality Assurance                |                              | source controls.                                                                                     | Cost Involvement.                                                             |
| Manual Testing                   | Quality Assurance Team       | Highly robust functional error free application                                                      | More Resource involvement. More Training Requirements. Time consuming process. |
| Testing Automation and Testing    | Quality Assurance Team and   | Easy turn around for regression testing. Effective for frequent changes.                             | Time consuming process for the initial stages. Initial Cost Involvement.       |
| Coverage                         | Development Team             |                                                                                                       |                                                                               |

4. Discussion

4.1 Techniques and Challenges

CI/CD pipeline will provide users with personalized and targeted suggestions [38]. Therefore, they address the problem of the regular implementations constrains in the digital era. Different tools have recently been developed, which can incorporate the process to find optimized CI/CD pipeline in a controlled .NET environment.

4.2 Evaluation matrices

The most prominent evaluation metrics in the research literature measure the accuracy of the system's predictions. Accuracy is measured based on the various techniques and options which are considered during the research.

4.3 Expected Contribution

Understand patterns identified drawbacks, corns, issues, pitfalls, difficulties and challenges from the respective area and improve the efficiency, accuracy, easiness and cost-effective implementation of the Continuous Integration and Continuous Development in Azure DevOps for a controlled .NET environment. Provide the helpful matrix to make the effective and cost effective industrial standard recommendations in the early stage of the implementation. How to design an effective alert or runtime communication mechanism with dashboard data representation. Best recommendation for the code analysis and code review process and the techniques and practices which help to do the proper test coverage for the manual and automated test cases.
5. Conclusion

Came up with various suggestions and techniques together with best practices and policies needed to follow to setup an Optimize Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment using different techniques and practices. In this paper, we presented possibilities of optimizing the Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment using different techniques and practices. We presented different options, methodologies, techniques and practice appropriate for this study. Looking through articles, available tools, CI/CD practices, and available methods, we can prove that lot of possibilities can be considered in optimizing the Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment using different techniques and practices. As part of this study our proposed solutions are mainly focusing on Cost Based, Administration and Configuration, Architectural, Development and Technical and Quality Assurance, Manual Testing and Testing Automation and Testing Coverage. Each option has it’s own capabilities to leverages the CI/CD pipeline implementation in a Azure DevOps for a controlled Microsoft .NET environment, which will help to reduce the cost, effort, duration, process and manual interventions through out the SDLC life cycle of an application development and it’s maintenance. The benefits will vary based on the options, techniques, practices and tools which are considering during the process. In order to succeed in the Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment using different techniques and practices, it is important to focus on the changing of the existing business, operational culture and to get the buy-in support from the executive team. Identifying the right approaches, which include the techniques, practice and tools that fit for the business needs. Leveraging our research, future work will be focused on any one/more of mentioned specific techniques, practice or tools to optimize the Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment effectively.

References:
[1] Mahesh Kuruba, Prashant Shenava and Jenny James 2018 Real-time DevOps Analytics in Practice Preprint QuASoQ Workshop
[2] Krusche S and Alperowitz L 2014 In Proc. of ICSE’14. IEEE on Introduction of Continuous Delivery in Multi-Customer Project Courses
[3] Continuous delivery vs. continuous deployment, “Practicing Continuous Integration and Continuous Delivery on Aure,” Available online: (accessed on 15 June 2020).
[4] Rahman A, Helms E, Williams L, and Parnin C 2015 In Proc. Agile Conf. (AGILE) on Synthesizing continuous deployment practices used in software development pp 1-10
[5] Shahin M, Ali Babar M and Zhu L 2017 Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices IEEE Access vol 5 pp 3909-3943
[6] “Adopting Continuous Delivery” in Continuous Delivery Whitepaper. [Online] Available: Levi9 IT Services, 2016 [Accessed: Sept, 2017].
[7] Feitelson D G, Frachtenburg E, and Beck K L 2013 July – August Development and Deployment at Facebook in IEEE Internet Computing vol. 17 pp 8-17
[8] Chen L 2015 Towards Architecting for Continuous Delivery in Software Architecture (WICSA) 12th Working IEEE/IFIP Conf. pp 131-134
[9] Chen L 2018 Microservices: Architecting for Continuous Delivery and DevOps in IEEE International Conf. on Software Architecture (Seattle: USA, IEEE).
[10] Chen L, Ali Babar M, and Nuseibeh B 2013 Characterizing Architecturally Significant
Requirements Software IEEE vol. 30 pp 38-45

[11] Ayewah N and Pugh W 2010 The google findbugs fixit in Proceedings of the 19th international Symp. on Software testing and analysis ACM pp 241–252

[12] Beller M, Gousios G, and Zaidman A 2016 Oops, my tests broke the build: An analysis of travis CI builds with GitHub (PeerJPrePrints vol. 4 [Online] Available: http://dx.doi.org/10.7287/peerj.preprints.1984v1)

[13] Hilton M, Tunnell T, Huang K, Marinov D, and Dig D 2016 Usage, costs, and benefits of continuous integration in open-source projects in Proceedings of the 31st IEEE/ACM International Conf. on Automated Software Engineering (ASE) pp 426–437

[14] Kim S, Park S, Yun J, and Lee Y 2008 Automated continuous integration of component-based software: An industrial experience in Proc. Int. Conf. Automated Software Engineering (ASE) IEEE pp 423–426.

[15] Laukkanen E I, Paasivaara M, and Arvonen T 2015 Stakeholder perceptions of the adoption of continuous integration - A case study in AgileConf. (AGILE) pp 11–20