

CI-CD Deployment

Milan Ples - 20088120

01-01-2023



Contents

Plagiarism Statement	2
Acknowledgements	2
Abstract	2
Introduction	2
Purpose, Intended Use and Audience	3
Rationale for the Project	3
Risk Assessment	3
Methodology	4
Traditional Waterfall vs Continuous Integration and Continuous Deployment	4
Introduction:	4
Cost Savings:	4
Speed:	4
Workload:	5
Conclusion:	5
Development cycle	5
Planning	5
An Agile Approach with Kanban	5
Test-Driven-Development	5
Continuous Integration/Development	5
Technologies Used	5
Git	5
Github	6
Github Actions	6
Jenkins	6
Docker	6
AWS	6
Kubernetes	7
Terraform	7
Results	7
Discussion	7
Limitations	7
Conclusion	7
Bibliography	7

Plagiarism Statement

I confirm that the work contained in this report is my own, except where due reference is made in the text to the work of others. I have acknowledged all material, data or ideas from other sources, whether quoted verbatim or paraphrased, by the use of suitable references and footnotes. I have also acknowledged any assistance received in preparing this report.

Acknowledgements

I would like to thank my supervisor, Patrick Felicia for his guidance and support throughout this project. My team at Unum who helped me immensely during my internship in executing the project and providing endless support and help.

Abstract

The purpose of this project is to streamline the software development process through the use of a CI-CD workflow that automates the steps of building, testing, and deploying code changes. This is achieved through the utilization of git for version control, Github for code repository management, Jenkins for continuous integration, Docker for packaging code into containers, and AWS for deployment.

By implementing this workflow, developers can make code changes and have them automatically built, tested, and deployed, reducing the time and effort required to bring new features and updates to production. This enables teams to deliver software updates faster and with fewer errors, improving the overall efficiency and quality of the software development process.

Introduction

Continuous Integration-Continuous Deployment (CI-CD) is a software development approach that aims to automate the process of building, testing, and deploying code changes. This approach is designed to improve the efficiency and quality of the software development process by allowing teams to deliver updates and new features faster, with fewer errors.

One way to implement a CI-CD workflow is through the use of tools such as git for version control, Github for code repository management, Jenkins for continuous integration, Docker for packaging code into containers, and AWS for deployment. In this project, these tools are used together to create a streamlined process for building, testing, and deploying code changes. This enables developers to focus on writing code and adding new features, while the CI-CD pipeline handles the rest.

Purpose, Intended Use and Audience

The purpose of this CI-CD project is to automate the process of building, testing, and deploying code changes, in order to improve the efficiency and quality of the software development process. It is intended to be used by software development teams who want to streamline their workflow and deliver updates and new features to their users faster, with fewer errors.

The intended audience for this project is software developers and devops professionals who are responsible for building, testing, and deploying code changes. By implementing a CI-CD workflow, these individuals can focus on writing code and adding new features, while the pipeline handles the rest of the process. This allows them to work more efficiently and effectively, and helps to ensure that code changes are thoroughly tested and deployed in a timely manner.

Overall, the goal of this CI-CD project is to make the software development process more efficient, reliable, and scalable, enabling teams to deliver high-quality software updates to their users more quickly and with fewer errors.

Rationale for the Project

As a software developer and Dev-Ops enthusiast, I am well aware of the benefits of automating processes in order to optimize the development process and decrease expenses. When I was first introduced to continuous integration and continuous deployment (CI-CD) during my internship, I was immediately drawn to the potential this methodology had to revolutionize how code modifications are created, tested, and deployed.

Utilizing tools such as git, Github, Jenkins, Docker, and AWS, a CI-CD pipeline can automate the steps of building, testing, and deploying code changes, decreasing the time and effort needed to bring new features and updates to production. This not only increases the efficiency of the development process, but also helps to guarantee the reliability and quality of the final product.

I have therefore made it a priority to learn about and understand the implementation of a CI-CD pipeline. I am confident that this approach has the potential to greatly benefit any organization and its users, and I am eager to see the positive impact it can have on the software development process. By embracing automation and adopting a CI-CD workflow, I believe it is possible to significantly enhance the speed and reliability of the development process, while also reducing costs and minimizing errors.

Risk Assessment

The risks associated with this project are as follows:

As I embark on this project, I am aware that there is a risk that I may not be able to deliver it as planned. This risk is especially relevant due to my lack experience with the technologies and tools being used, as I may encounter difficulties or delays in implementing the desired functionality. Additionally, if the project is overly complex or involves a large number of features, it may be difficult for me to manage and could result in delays or cutting out features.

I also need to consider the possibility that external factors such as third-party APIs or vendor support may not be reliable or may change unexpectedly, which could impact the project's timeline and success. To increase the chances of delivering the project successfully, I will need to carefully assess and manage these risks. This may involve seeking additional resources or expertise, breaking the project down into smaller tasks, or implementing contingency plans. By proactively addressing these risks, I can increase the likelihood of delivering the project as planned.

Methodology

Waterfall is a traditional linear method of software development that involves completing each phase of the development process before moving on to the next, while CICD involves continuously integrating and deploying code changes.

Traditional Waterfall vs Continuous Integration and Continuous Deployment

Introduction:

Waterfall and CICD (Continuous Integration and Continuous Deployment) are two different development methodologies that are used to build software. While Waterfall is a traditional method that follows a linear, sequential process, CICD is a more modern approach that emphasizes rapid iteration and continuous delivery of software updates. In this article, we will compare the two approaches in terms of cost savings, speed, and workload.

Cost Savings:

One potential advantage of the Waterfall model is that it may be less expensive to implement, as it does not require the same level of infrastructure and automation as CICD. Waterfall also typically involves fewer resources, as teams do not need to constantly monitor and deploy code changes. However, the Waterfall model can be inflexible and may not be well-suited to projects with rapidly changing requirements or a high degree of uncertainty. As a result, teams may need to go back and redo work that has already been completed, leading to additional costs and delays.

On the other hand, CICD can offer significant cost savings by reducing the need for manual testing and deployment processes. By continuously integrating and deploying code changes, teams can quickly and easily make updates to their software and respond to changing business needs. This can also lead to reduced maintenance costs, as teams can more easily fix defects and make updates to the software as needed. However, CICD does require a significant investment in infrastructure and automation, as well as a well-defined process for testing and deploying code changes.

Speed:

The Waterfall model is typically slower than CICD, as each phase of the software development life cycle (SDLC) must be completed before moving on to the next phase. This can lead to delays and missed deadlines, particularly

if there are issues or changes that require going back and redoing work that has already been completed.

In contrast, CICD allows for rapid iteration and continuous delivery of software updates, allowing teams to quickly respond to changing business needs or customer feedback. This can lead to faster time-to-market and the ability to deliver new features and functionality more quickly. However, it is important to ensure that code changes are thoroughly tested and do not break existing functionality, as this can lead to delays and additional work.

Workload:

The Waterfall model may require less workload upfront, as teams do not need to continuously integrate and deploy code changes. However, the inflexibility of the Waterfall model can lead to additional workload if changes or defects need to be addressed once a phase has been completed.

CICD, on the other hand, requires a more continuous workload, as teams must continuously integrate and deploy code changes. This can lead to a more complex and challenging workload, as teams must ensure that code changes do not break existing functionality and that the software is stable and reliable. However, the ability to rapidly iterate and respond to changing business needs can also lead to a more dynamic and engaging work environment.

Conclusion:

Both Waterfall and CICD have their own benefits and limitations, and the appropriate approach will depend on the specific needs and constraints of a project. Waterfall may be more suitable for projects with well-defined requirements and a low level of uncertainty, while CICD may be more appropriate for projects with rapidly changing requirements or a need for rapid iteration and innovation. Ultimately, the choice between the two approaches will depend on the specific cost, speed, and workload considerations of the project, as well as the resources and capabilities of the development team.

Development cycle

Planning

An Agile Approach with Kanban

Test-Driven-Development

Continuous Integration/Development

Technologies Used

Git

Git: Git is a version control system that allows developers to track changes to their codebase and collaborate with other team members. It allows developers to save different versions of their code and switch between

them as needed, making it easier to track and manage changes. Git also allows developers to work on the same codebase at the same time, without having to worry about overwriting each other's work. This enables teams to work more efficiently and effectively, and helps to ensure that code changes are thoroughly tested and deployed in a timely manner.

Github

GitHub: GitHub is a web-based platform that allows developers to host and review code, manage projects, and build software. It is built on top of Git and allows developers to easily collaborate on code and track changes.

Github Actions

GitHub Actions: GitHub Actions is a tool provided by GitHub that allows developers to automate their workflow by setting up a series of tasks, known as "actions," that can be triggered based on certain events or conditions. These actions can be used to build, test, and deploy code changes, as well as perform other tasks such as releasing software and publishing documentation. GitHub Actions can be used in conjunction with other tools, such as Jenkins or Kubernetes, to create a full CI-CD pipeline.

Jenkins

Jenkins: Jenkins is an open-source automation server that helps developers automate parts of the development process, such as building, testing, and deploying code changes. It allows developers to set up a series of tasks, known as "jobs," that can be automatically triggered based on certain events or conditions.

Docker

Docker: Docker is a tool designed to make it easier to create, deploy, and run applications in a containerized environment. A container is a lightweight, standalone, and executable package that includes everything an application needs to run, including code, libraries, dependencies, and runtime. Using Docker allows developers to easily package and deploy their applications in a consistent manner across different environments.

AWS

AWS (Amazon Web Services): AWS is a cloud computing platform that provides a wide range of services, including computing, storage, and database management. It allows developers to build, test, and deploy applications at scale, without having to worry about the underlying infrastructure. AWS also offers a number of tools and services specifically designed for continuous integration and continuous deployment (CI-CD), such as CodePipeline and CodeBuild.

Kubernetes

Kubernetes: Kubernetes is an open-source container orchestration platform that allows developers to manage and deploy containerized applications at scale. It provides features such as automatic scheduling, self-healing, and horizontal scaling, which make it easier for developers to deploy and manage applications in a production environment.

Terraform

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. It is an open-source tool that allows users to define infrastructure as code (IaC) and manage it using a command-line interface. With Terraform, users can create, update, and version infrastructure resources such as virtual machines, networking components, and storage solutions across multiple cloud providers (such as AWS, GCP, and Azure) and on-premises data centers.

Using Terraform, users can define infrastructure resources in a configuration file written in the HashiCorp Configuration Language (HCL). This configuration file is used to create and manage resources in a consistent, repeatable way. Users can use Terraform to create, update, and delete resources in a predictable manner, making it easier to manage infrastructure changes and roll back updates if necessary.

Terraform also includes features such as dependency management, which allows users to specify the order in which resources are created, and resource tracking, which allows users to keep track of resources that have been created and modified. Overall, Terraform is a powerful tool for managing infrastructure in a consistent and repeatable way, making it easier for organizations to build and maintain infrastructure on a variety of cloud and on-premises platforms.

Results

Discussion

Limitations

Conclusion

Bibliography

example: - Oakland, J. (1987). The Economics of Non-excludability. The Journal of Economic Perspectives, 1(1), 19-32. doi:10.1257/jep.1.1.19