CI-CD Deployment

Milan Ples - 20088120

01-01-2023

Contents

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.

Declaration

I confirm that this report has not been submitted for any other course and that I have not previously submitted work of a similar nature for this or any other course.

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.

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 if I 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 an increased risk of project failure.

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. Finally, if the project is underfunded or I do not have the necessary resources to complete it, there is a risk of delays or a reduction in scope.

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

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

AWS

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