Simple CI/CD Pipeline for Web Application using Azure

Ms.N.Selvalakshmi, Student, Department of Computer Science, Rathinam College of Arts and Science, Coimbatore

Dr. M. USHA DEVI M.Sc., M.Phil., Ph. D., NET.SET. Assistant Professor Department of Computer science ,Rathinam College of Arts and Science,Coimbatore.

ABSTRACT

This project aims to design and implement an automated Continuous Integration and Continuous Deployment (CI/CD) pipeline for a web application using Microsoft Azure DevOps. CI/CD is a modern DevOps practice that enables developers to integrate code changes frequently, automatically build applications, and deploy them efficiently with minimal manual intervention. The system integrates a source code repository using GitHub and automates the process of building, testing, and deploying the application through Azure Pipelines. Whenever code changes are pushed to the repository, the pipeline is triggered, ensuring faster feedback and early detection of errors. This reduces development time, minimizes deployment risks, and improves overall software quality. The project also focuses on automating the deployment of the web application to cloud platforms such as Azure Virtual Machines or Azure App Services.

Keywords: Continuous Integration, Continuous Deployment, Azure DevOps, GitHub, CI/CD Pipeline, DevOps, Cloud Deployment, Automation, Web Application, Azure App Service, Version Control, Build Automation, Release Management, Software Delivery, Pipeline Automation

1.INTRODUCTION

In modern software development, delivering applications quickly and reliably is a major challenge. Traditional deployment methods rely heavily on manual processes such as code integration, testing, and deployment, which are time-consuming, error-prone, and inefficient. These limitations often lead to delays, inconsistent deployments, and reduced software quality. Existing deployment systems lack automation and do not provide real-time integration and continuous delivery mechanisms. There is also limited monitoring and tracking

of application updates, making it difficult to maintain consistency across development, testing, and production environments. This project introduces a Simple CI/CD Pipeline for Web Application using Azure DevOps

The main contributions of this work are:

Development of an automated CI/CD pipeline using Azure DevOps.

Integration of GitHub for version control and automatic pipeline triggering.

Implementation of build, test, and deployment automation using Azure Pipelines.

Deployment of a working web application on Microsoft Azure cloud

2. LITERATURE SURVEY

Previous research in DevOps and cloud computing shows that **CI/CD pipelines** play a major role in automating software development and deployment. Many studies highlight how tools like Azure DevOps, Jenkins, and GitHub Actions improve efficiency by enabling continuous integration and continuous delivery.

However, many traditional systems still lack **fully automated pipelines integrated with cloud deployment and real-time monitoring dashboards**. Some approaches focus only on build automation but do not include complete end-to-end deployment and monitoring. These studies support the development of a **fully automated CI/CD pipeline system**, which integrates version control, build automation, deployment, and monitoring into a single platform.

3. DATASET COLLECTION

In this project, there is no traditional dataset used as in machine learning systems. Instead, the system works with application source code and pipeline-related data. The primary inputs include HTML files for the frontend, Node.js files for the backend, and configuration files such as the Azure pipeline YAML file. Additionally, data is generated through GitHub repositories in the form of commits, branches, and version history, which act as inputs to trigger the pipeline process.

During pipeline execution, various types of data are automatically generated, including build status, deployment status, execution logs, timestamps, and error messages. In this project, there is no traditional dataset used as in machine learning systems. Instead, the system works with application source code and pipeline-related data. The primary inputs include HTML files for the frontend, Node.js files for the backend, and configuration files such as the Azure pipeline YAML file. Additionally, data is generated through GitHub repositories in the

form of commits, branches, and version history, which act as inputs to trigger the pipeline process. During pipeline execution, various types of data are automatically generated, including build status, deployment status, execution logs, timestamps, and error messages.

4. PROPOSED WORK

The proposed work focuses on developing a Simple CI/CD Pipeline for Web Application using Azure DevOps, which aims to automate the entire software development and deployment lifecycle. The system is designed to eliminate manual processes and provide a reliable and efficient solution for continuous application delivery. It integrates GitHub for source code management and Azure DevOps for pipeline automation, ensuring that all development activities are streamlined.

The working of the system begins when a developer pushes code to the GitHub repository. This action automatically triggers the CI/CD pipeline configured in Azure DevOps. During the Continuous Integration phase, the system installs required dependencies, builds the application, and verifies whether the code is functioning correctly. Once the build process is successful, the Continuous Deployment phase begins, where the application is automatically deployed to cloud platforms such as Azure Virtual Machine or Azure App Service. This ensures that deployment is fast, consistent, and free from manual errors. The system also provides real-time monitoring through the Azure DevOps dashboard, where developers can track build status, deployment progress, and execution logs.

The overall workflow of the proposed system includes:

- Code Push (GitHub)
- Pipeline Trigger (Azure DevOps)
- Build & Integration
- Testing & Validation
- Deployment to Azure
- Monitoring via Dashboard

5. SYSTEM ARCHITECTURE

The system architecture defines the overall structure and interaction between components in the CI/CD pipeline-based web application system. The proposed system follows a cloud-based client-server architecture, where developers interact with the system through a version control platform, and the backend pipeline automates the processes of building, testing, and deploying the application. The architecture integrates multiple components such as GitHub,

Azure DevOps, the build environment, deployment services, and monitoring dashboards to ensure smooth and automated software delivery. The architecture consists of key components including the Source Code Repository, CI/CD Pipeline Engine, Build and Execution Environment, Cloud Deployment Platform, and Monitoring Dashboard. The Source Code Repository, implemented using GitHub, acts as the central storage for application code. Developers push changes to this repository, which serves as the primary input for the system. It also maintains version control, tracks changes, and enables collaboration. The CI/CD Pipeline Engine, implemented using Azure DevOps, is the core component that manages the automation process. It continuously monitors the GitHub repository and automatically triggers the pipeline whenever code changes are detected. The pipeline is defined using a YAML configuration file, which specifies the sequence of steps required for building, testing, and deploying the application. This ensures consistency and repeatability in the workflow.

The Build and Execution Environment is responsible for preparing the application for deployment. It installs necessary dependencies, executes application code, and validates the functionality during the Continuous Integration phase. This environment ensures that the application is error-free and ready for deployment. The execution process is automated and runs on cloud-based virtual machines provided by Azure DevOps.

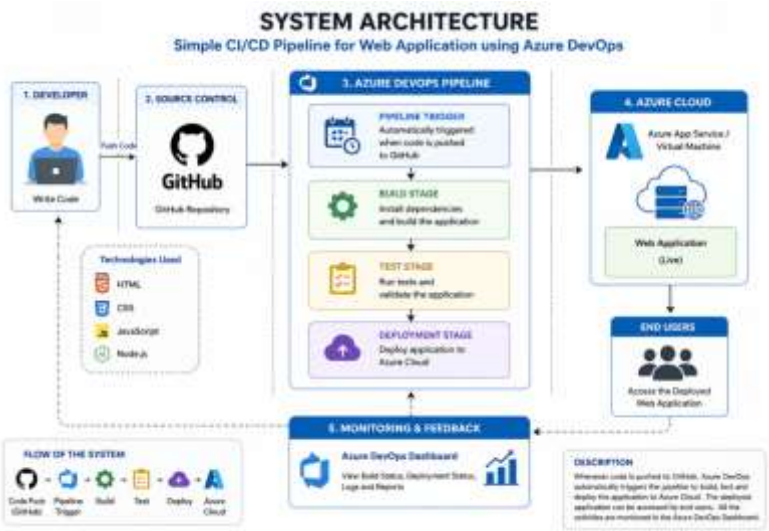

Fig 5.1 System Architecture of Cyber Incident Portal

6. METHODOLOGY

The methodology of the proposed system explains the step-by-step process used for automating application development and deployment using a CI/CD pipeline. The system begins with Code Integration, where developers write application code and push it to the GitHub repository. Next, Pipeline Triggering is performed automatically by Azure DevOps whenever changes are detected in the repository.

The code is then processed in the Continuous Integration stage, where the pipeline installs dependencies, builds the application, and checks for errors to ensure proper functionality. Based on this process, the system verifies whether the build is successful or failed and displays the result to the developer through logs and status updates. After successful integration, the system proceeds to the Continuous Deployment stage, where the application is automatically deployed to the Azure cloud platform such as Azure Virtual Machine or Azure App Service. Once deployment is completed, the application becomes accessible to users.

The methodology follows the sequence: **Code Integration → Pipeline Triggering → Continuous Integration → Build Validation → Deployment → Monitoring Dashboard.**

7.RESULTS

The proposed system successfully automates the process of application development, integration, and deployment using a CI/CD pipeline. The results show that whenever code is pushed to the GitHub repository, the Azure DevOps pipeline is automatically triggered and performs build and deployment operations without manual intervention. The system correctly installs dependencies, executes the application, and verifies the build status as Success or Failure based on the code functionality.

The system also provides real-time feedback through logs and status updates, helping developers quickly identify and fix errors. After successful build validation, the application is deployed automatically to the Azure cloud platform, making it accessible to users without delays. All pipeline activities, including build status, deployment results, and execution logs, are displayed through the Azure DevOps dashboard for monitoring and management.

The overall results demonstrate that the proposed system effectively performs automated integration, continuous deployment, and centralized monitoring, providing a reliable and efficient solution for modern software delivery.

8. CONCLUSION

The proposed CI/CD Pipeline for Web Application using Azure DevOps provides an efficient and automated solution for software development and deployment. The system successfully integrates GitHub for version control and Azure DevOps for pipeline automation, enabling continuous integration and continuous deployment of applications. By automating build, testing, and deployment processes, the system reduces manual effort, minimizes errors, and improves the speed and reliability of software delivery. The use of cloud platforms ensures scalability and consistent deployment across different environments. Additionally, the monitoring dashboard provides better visibility and control over the entire pipeline process.

Overall, the project demonstrates how DevOps practices and cloud technologies can be used to improve application development efficiency, ensure faster delivery, and maintain high software quality in real-world scenarios.

9.REFERENCES

Here is a sample **References** section for your project (paper format style):

[1] Humble, J. and Farley, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley, 2010.

[2] Kim, G., Humble, J., Debois, P., and Willis, J. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations.* IT Revolution Press, 2016.

[3] Microsoft. “Azure DevOps Documentation.” Microsoft Learn. Available: https://learn.microsoft.com/azure/devops

[4] Microsoft. “Azure Pipelines Overview.” Microsoft Learn. Available: https://learn.microsoft.com/azure/devops/pipelines

[5] GitHub. “GitHub Documentation.” Available: https://docs.github.com

[6] Fowler, M. “Continuous Integration.” Martin Fowler Blog. Available: https://martinfowler.com/articles/continuousIntegration.html

[7] Amazon Web Services. “What is CI/CD?” AWS Documentation. Available: https://aws.amazon.com/devops/what-is-ci-cd

[8] Red Hat. “What is CI/CD?” Red Hat Developer. Available: https://www.redhat.com/en/topics/devops/what-is-ci-cd

[9] Node.js Foundation. “Node.js Documentation.” Available: https://nodejs.org/en/docs

[10] Docker Inc. “Docker Documentation.” Available: https://docs.docker.com

\