2025 IEEE International Conference on Emerging Technologies and Applications (MPSec ICETA) | 979-8-3315-2131-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/MPSecICETA64837.2025.11118842

Testing and Optimizing Web Applications with Continuous Integration/Continuous Deployment in Cloud Environments

Vashudhar Sai Thokala
Independent Researcher
vasudharsai7@gmail.com

Sumit Pillai
Independent Researcher
sumitpillai@yahoo.com

Sandeep Gupta
SATI
Vidisha, M.P., India
Sandeepguptabashu@gmail.com

***Abstract*—CI/CD pipeline solutions are receiving widespread adoption in order to speed up the delivery of web applications while at the same time being able to scale, secure, and optimize them. This study delves into the optimization of CI/CD pipelines in cloud systems, with a particular emphasis on AWS and Azure provided by Microsoft. The study, titled Testing and Optimizing Web Applications with Continuous Integration/Continuous Deployment in Cloud Environments, evaluates the performance of CI/CD pipelines using AWS Code Pipeline and Azure Pipelines, along with Terraform and Kubernetes for infrastructure management. By conducting a series of tests across three scenarios, including CI-only and CI+CD pipelines, the research compares deployment speeds, resource utilisation, and scalability under varying conditions. Selenium for browser testing, Jenkins for automation, and GitLab CI/CD for pipeline management are used for evaluating effectiveness, and effectiveness of the pipelines. The study can prove useful in establishing the features that differentiate between AWS and Azure specifically regarding the speeds of deploying their resources and scalability.**

Keywords—Continuous Integration (CI), Continuous Deployment (CD), Cloud Platforms, CI/CD Optimization, Deployment Speed, AWS Code Pipeline, Azure Pipelines.

I. INTRODUCTION

The CI/CD methodology "has emerged as a pivotal strategy" in the dynamic software development industry [1][2]. The software development, testing, and deployment processes are being fundamentally changed by this method. The ability of CI/CD to solve important problems in software development, such as the requirement for greater communication between "development and operations teams," faster release cycles, and higher software quality, has led to its broad acceptance [3][4][5]. Aiming to automate and streamline the software delivery process, CI/CD is basically a collection of tactics and concepts. It includes the ability to have a single place where a code base lives, to divide work into more digestible tasks or epic's and to integrate at more frequent intervals [6]. One definition of Continuous Integration (CI) is a set of best practices for facilitating the ongoing monitoring of code repositories [7][8], which in turn allows development teams to incorporate changes to code and associated check-ins more efficiently [9][10]. An automatic method for making these modifications is provided by CD, which binds the stored packaging specifications to each delivery [11].

The use of web apps has grown in importance for many businesses and nonprofits. Web applications [12][13] allow businesses to reach out to users in an easy and convenient way, which in turn increases user engagement [14]. A further benefit is the ability to centralize data storage, which facilitates its use for various reasons [15]. Cloud storage has several benefits, including improved security, simpler customization and scalability, and less hassle with installation and maintenance. To achieve their goals, users often engage primarily via web apps, which allow them to manage their data and the way it flows through the system. Cloud service providers may maximize their profit via effective resource utilisation and satisfy quality of service criteria with the aid of load balancing [16].

A. Research Motivation, Novelity and Contribution of Work

The rationale of this work is to respond to the increasing demand for the proper implementation of CI/CD pipelines in contemporary DevOps strategies moving to the cloud environment. As more organizations use Kubernetes and other cloud-native tools, they need accurate and high-speed deployment systems. This research compares AWS and Azure, two main cloud providers, to determine which provider is better suited to provide faster deployment time, optimal resource usage, and flexibility. These results show guidelines that can be used to enhance the CI/CD processes in various organizations, minimize the costs of operations and enhance the cloud deployment efficiency. This research introduces a novel comparative analysis of AWS and Azure cloud platforms in the context of Continuous Integration (CI) and Continuous Deployment (CD) pipelines, focusing on deployment speed, resource utilisation, and scalability. The study is unique in its comprehensive approach, using 150 pipeline runs across various node configurations to assess deployment speed, latency, and consistency, alongside 50 resource utilisation tests to measure CPU and memory consumption during scaling operations. The study also brings into light further insights into the effectiveness of both platforms, to a broader understanding that Azure wins in deployment time and utilisation of resources even during scalability operations. This research provides useful information to organizations and cloud stakeholders interested in improving CI/CD processes and cost efficiency in cloud settings while adopting Kubernetes infrastructures. This study makes several contributions to testing and optimizing web application development and deployment through CI and CD pipelines in cloud environments.

- Comparative Analysis of CI/CD Pipelines: Comparisons provide an understanding of how each platform approaches the CI/CD process, with Azure proving faster and more scalable deployment when using Kubernetes.
- Optimized CI/CD Pipeline Architecture: Enables synchronization across different tools and applications to enhance the flow of continuous integration and continuous deployment solutions.

- Evaluation of Deployment Speed: Evaluation reveals the performance trends, providing insights into the time it takes for pipelines to execute, as well as identifying bottlenecks in CI and downstream processes.
- Resource Utilization Analysis: assess the efficiency of each platform when handling resource-intensive tasks, providing insights into optimal resource allocation for large-scale deployments.
- Scalability Assessment Under Varying Conditions: provide practical recommendations on selecting cloud platforms based on specific deployment needs, including scaling capabilities, resource efficiency, and deployment speed.

B. *Organization of the paper*

The remainder of the paper is organized in this manner. In Section II, present a literature review. Section III presents methods and methodology, and Section IV results analysis and discussion. Conclusion and future work of this study present in section V.

II. Literature Review

In this section, discuss previous research on Continuous Integration/Continuous Deployment in Cloud Environments and also optimize and test their use of web applications.

Zampetti et al. (2021) findings from research that evaluated the measures taken to restructure the CI/CD pipeline using both qualitative and quantitative methods. Maintaining such pipelines effectively to adapt to the system and its underlying technological progress and to restrict undesirable practices may determine the efficacy of CI/CD [17].

Chatterjee and Mittal (2024) addresses the general implication of these findings, such as the need for organizations, at the current Earthly and digital age, to adopt state-of-the-art and flexible deployment models for competitiveness and efficiency. This work advances the knowledge of how CI/CD and DevOps-based automated deployment techniques can transform software development approaches [18].

Klooster et al. (2023) explore ways to improve the efficiency of CI/CD pipelines via the use of fuzz testing, particularly for security-related issues, and look for ways to prioritize contributions that don't need fuzzing. They saw that three out of the nine libraries they examined may have their fuzzing effort lowered by 63% (or 55% on average) due to their experimental investigation [19].

Sushma et al. (2023) examination also considers any vulnerabilities in the pipeline. Several attack paths that might jeopardize the software's availability, confidentiality, and integrity are discussed, including malicious code injection, unauthorized access, and exploitation of misconfigurations. Very few security measures have been included in the state-of-the-art CI/CD pipeline to address these vulnerabilities. The primary tools used in this investigation are Snyk, CodeQL, and SLSA[20].

Arachchi and Perera (2018) offers you the framework you need to create and integrate using Jenkins and pipeline methods, then deploy using the Ansible tool. Jenkins makes integration a breeze with its open source nature and many plug-in possibilities. Jenkins and Ansible automate integration and deployment, which saves a tonne of time especially when making frequent changes to the project [21].

Kataru et al. (2023) looks at four well-liked AWS deployment strategies: Elastic Beanstalk, EC2, CI/CD, and Docker and Kubernetes. Companies and developers may utilize the information gleaned from this comprehensive comparison to make informed decisions when installing applications on AWS. The need to deliver applications with least possible delay plays a vital role in the power filled digital world to give the maximum amount of satisfaction to the user [22].

Continuous Integration/Continuous Deployment in the cloud literature has been more focused on extensibility and flexibility of the pipelines with reference to technological developments. Research discusses increasing pipeline efficiency by using fuzz testing, reviewing susceptibilities, including code injection, and increasing security by utilizing tools such as Snyk and CodeQL. Among the procedures that have been mentioned in relation to integration and deployment are Jenkins and Ansible. Furthermore, observations of deployment strategies in cloud platforms like AWS EC2 vs Kubernetes comparisons accentuate the need for latency control to make users happy. Taken together these studies all center on the goals of achieving fast, secure and dynamic deployment in cloud contexts.

III. Methodology

Web development optimization is evaluated through CI/CD pipelines on AWS and Azure. The process in both systems involves tools like AWS Code Pipeline and Azure Pipelines that allow for CI/CD solutions paired with Terraform solutions for infrastructure and Kubernetes solutions for applications. CI takes over the execution of tests and integration of codes and on the other hand CD is responsible for deploying and provision of infrastructure. Testing tools include Selenium, for cross-browser testing, Jenkins for CI automation and GitLab CI/CD for pipeline. Deployment speed and resource utilisation are analyzed across scenarios, with 150 pipeline runs for CD and CI+CD comparisons and 50 runs for resource tests, focusing on CPU, memory usage, and scalability under varying conditions. These tests reveal pipeline performance, scalability, and resource efficiency across multi-cloud environments.

A. *Cloud Platforms Environment*

The rapid software delivery process has also been made possible by the presence of CI/CD pipeline tools in AWS and Azure cloud services. AWS provides the Code Pipeline service, which helps in the automation of build, test and deploy workflow by using other services, including Code Build, to create the builds and Code Deploy to carry out deployments [23]. For Code Pipelines on MS Azure, Azure Pipelines is all inclusive for builds and deployments across different platforms seamlessly integrating with the rest of Azure DevOps services. The two platforms allow the use of containers, container orchestration services, and the use of multiple clouds allowing for effective and efficient management of CI/CD processes.

B. *CI/CD Pipeline Architecture*

An essential aspect of DevOps, CI and CD automate the transfer of code modifications from development to production. Automated testing of code modifications is a part of continuous integration, and the deployment process is part

of continuous delivery/deployment. Automated testing and deployment of changes made by development teams using CI/CD pipelines allows for quicker delivery without sacrificing quality [24].

1) Continuous Integration (CI)

A term "CI" refers to the process by which developers consistently push or merge their changes into a shared source code repository's master branch. In order to catch mistakes and security risks early on in the development process, automated testing and validation methods are implemented alongside this integration. Code conflicts are minimized and bugs may be fixed quickly with the aid of CI, which automates testing and merges changes often. Version control systems enable common CI activities like static code analysis, automatic packaging, and compilation by successfully tracking code changes [25].

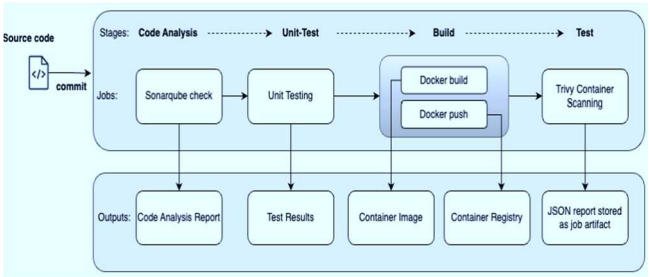

Fig. 1. CI pipeline Architecture

The CI Pipeline is shown in Figure 1 in great detail, with all of its phases, tasks, and outputs highlighted. The four steps of the pipeline configuration are described in the.gitlab-ci.yml file located at the repository root.

2) Continuous Delivery/ Deployment (CD)

In addition to continuous integration, continuous delivery streamlines the process of releasing applications and providing infrastructure. The code is prepared for deployment to any environment after it has been developed and tested in the CI process. This includes providing the required infrastructure and deploying the application to a test or production environment. Expanding on the concept of continuous integration, continuous deployment streamlines the process of releasing software to end users in real time. You may think of continuous deployment as a push strategy and continuous delivery as a pull strategy.

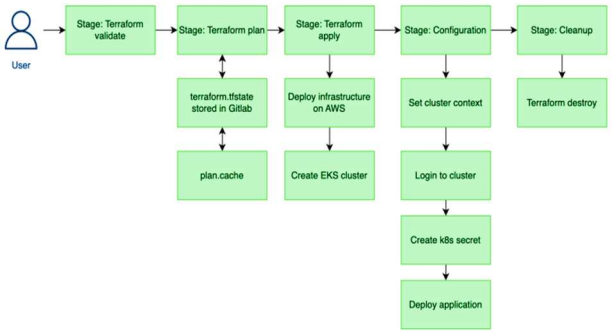

Fig. 2. CD pipeline architecture for AWS cloud environment

Key duties and the phases of the CD Pipeline in AWS are shown in Figure 2. First, the Terraform script is validated as part of the workflow. The next phase is "plan," and it's where Terraform draws up the installation plan for the infrastructure and stores it in a state file. After the application image is retrieved from the GitLab registry, it can be deployed in the EKS cluster using the Kubernetes deployment manifest that is located in the project root directory. In this step, you may also specify how many copies you want. Kubernetes application deployment is done with the following command: kubectl apply-f deployment.yaml -namespace=. The deployment.yaml configuration is executed by this command to build resources in the specified Kubernetes cluster namespace.

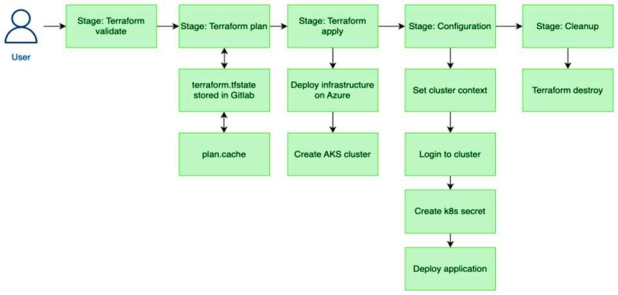

Fig. 3. CD pipeline architecture for Azure cloud environment

Key duties and the phases of the CD Pipeline in Azure are shown in Figure 3. No need to manage the master node in an AKS cluster; management of the worker nodes is sufficient. This is due to the cluster's controlled control plane. Instance type Standard_B4ms, situated in the East US region, and on-demand capacity type for worker nodes are both selected. The balanced computing, memory, and networking capabilities of the B-series burstable virtual machines make them perfect for general-purpose applications.

3) CI+CD Pipeline

The AWS and Azure cloud platforms both provide complete pipeline runs, which include application creation and deployment. These runs are made possible by CI and CD pipelines. It entails establishing a downstream pipeline that can handle many projects. Everything begins in the CI repository, which houses the CI pipeline steps shown in Figure 4.

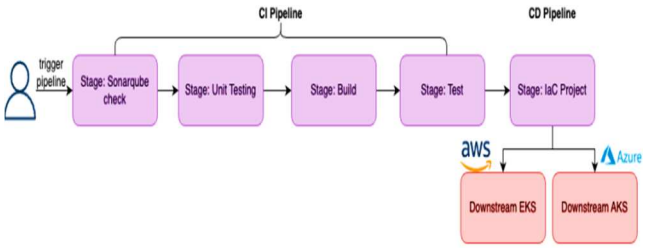

Fig. 4. CI/CD pipeline architecture with AWS and Azure

Figure 4 shows that when executed, these pipelines take care of their own tasks using their own projects and runners.

C. Testing Strategies

There are main test strategies are shown below: The "CI/CD pipeline" now includes automated testing in addition to unit and integration tests.

- **Selenium:** Web browser testing automation made easy with this open-source program.
- **Jenkins:** Code construction, testing, and reporting are all part of the CI process, which this automation server orchestrates.
- **GitLab CI/CD:** Integrated CI/CD platform that comes with GitLab.
- **AWS Code Pipeline:** A fully managed CI/CD service.

- **Azure DevOps:** A suite of Microsoft-provided development tools, including "CI/CD"-specific Azure Pipelines.

D. Testing and Evaluation Methods

1) Deployment Speed Comparison

To compare deployment speeds on AWS and Azure, three scenarios were evaluated. In **Scenario 1**, CD pipelines executed using Terraform scripts were tested across 150 runs with incremental node scaling (2, 5, 7 nodes) to measure stage-specific and total runtime, focusing on latency and deployment speed trends. **Scenario 2** extended the analysis to CI + CD pipelines in a multi-project downstream setup, also with 150 runs, capturing stage-wise and total runtime to identify CI bottlenecks and evaluate differences in handling downstream pipelines. **Scenario 3** analyzed CI + CD pipelines on pre-provisioned infrastructure through 50 test runs (alternating updates and rollbacks), assessing the time to apply Infrastructure as Code (IaC) changes and comparing IaC efficiency between the platforms. Together, these scenarios provide insights into pipeline performance, node scaling impact, and runtime consistency across AWS and Azure.

- **Stage-level Runtime:** Validate, plan, apply, configure.
- **Total Runtime:** End-to-end execution for CI/CD.
- Pipeline Completion Consistency: Variance across runs.

2) Resource Utilization Comparison

Resource utilisation on AWS and Azure was analyzed across two scenarios. In **Scenario 1**, 50 pipeline runs were conducted to evaluate CPU and memory utilisation during application upgrades and downgrades, alternating between commits (upgrades) and reverts (downgrades). The analysis focused on visualizing resource spikes and comparing overhead for both operations on each cloud. **Scenario 2** examined replica set scale operations, with 50 runs alternating between scaling up to 30 replicas and scaling down to 3 replicas. Metrics were collected to assess how scaling affected resource efficiency and to evaluate performance under constrained compute conditions (t3. large on AWS and Standard_B2ms on Azure). Together, these scenarios provide insights into resource consumption patterns and scalability on the two platforms.

- CPU Utilization: Average and peak usage during operations.
- Memory Utilization: Sustained usage vs. availability.
- Scaling Efficiency: Impact of replica adjustments on resources.

IV. Results And Analysis

In this section, research discuss about the testing and optimization for CI/CD to compare deployment speeds and Resource Utilization on AWS and Azure cloud environment for their web application.

A. Comparison of CD Pipeline Runtime

They use automated workflows to find out which cloud provider can build Kubernetes infrastructure the quickest. This would be useful for businesses in determining which cloud platform can install a Kubernetes cluster the quickest. According to the findings, Azure outperforms AWS in deploying. Deployment time grows in direct proportion to the number of nodes, according to an intriguing trend seen in AWS. Regardless of the configuration of the nodes, AWS deployments always take longer. As an example, compared to Azure, AWS has a much longer pipeline runtime for a 2-node setup. Azure continues to beat AWS even when the number of nodes climbs to 5 or 7, according to this pattern.

TABLE I. Average Runtime for CD Pipeline Across Different Node Configurations

Scaling Node	AWS (mm: ss)	Azure (mm: ss)
2 nodes	11:52	06:36
5 nodes	12:08	06:54
7 nodes	12:22	06:48

Table I, presents the average runtime for the Continuous Deployment (CD) pipeline on AWS and Azure cloud platforms across different node configurations. The scaling node count is incremented from 2 to 7 nodes, and the runtime (in minutes and seconds) for both AWS and Azure is recorded. This chart depicts time taken by completion CD pipelines and the actual deployment speeds varies from scale of the number of nodes as being determined from two platforms.

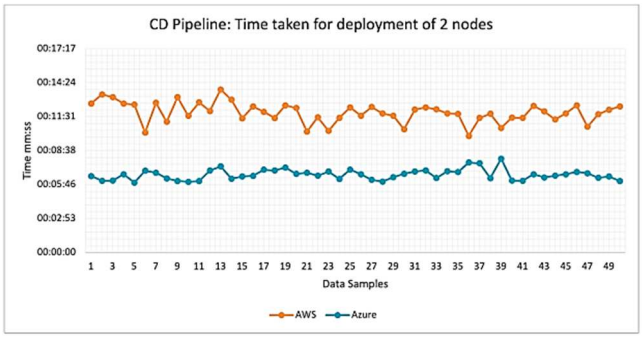

Fig. 5. CD Pipeline runtime: 2 nodes

Figure 5 shows the results of comparing the two cloud platforms' deployment timeframes for a CD pipeline with two nodes. The orange color denotes AWS as the percentage of the time spent on deploying, which shows fluctuating and large deployment time, whereas the blue line represents Azure, which always gives less and more stable deployment time. As this showcases that Azure is faster and much more reliable when it comes to deployment in this particular structure.

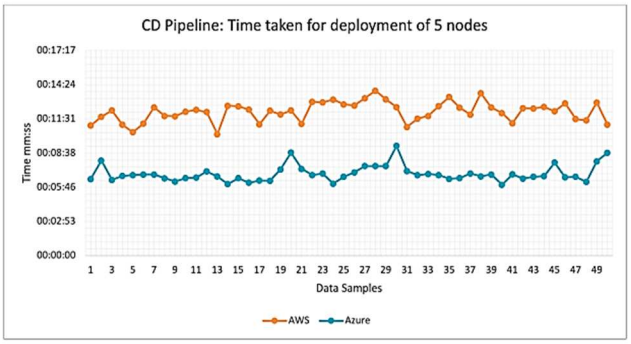

Fig. 6. CD Pipeline runtime: 5 nodes

Figure 6 displays the deployment times of a CD pipeline on AWS and Azure with 5 nodes. The orange line represents AWS, which fluctuates between 11:00 and 14:00 minutes, while blue line represents Azure, consistently showing deployment times between 5:30 and 9:00 minutes. This demonstrates that Azure provides faster and more stable deployment times compared to AWS, even as node count increases.

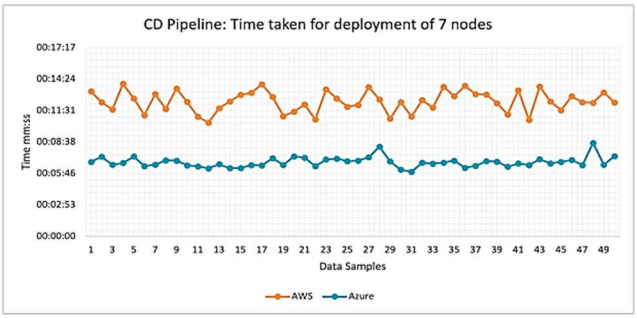

Fig. 7. CD Pipeline runtime: 7 nodes

Figure 7 illustrates the deployment times for the CD pipeline on AWS and Azure with 7 nodes. The orange line indicates AWS, with deployment times ranging between 11:30 and 15:00 minutes, while the blue line represents Azure, which consistently takes between 5:30 and 9:00 minutes. Azure demonstrates faster and more consistent deployment times compared to AWS, even as node count increases to 7.

B. Comparison of CI+CD Pipeline Runtime

They create the Kubernetes cluster from the ground up using automated Gitlab pipelines and compare the runtime of CI+CD processes in AWS and Azure to find out which cloud platform delivers apps quicker. Azure outperforms AWS by a margin of 35.53%. This pattern is true regardless of arrangement of nodes. Compared to AWS, Azure completes the CI+CD pipeline more quickly for a two-node deployment.

TABLE II. AVERAGE RUNTIME FOR CI+CD PIPELINE ACROSS DIFFERENT NODE CONFIGURATIONS

Scaling Nodes	AWS (mm: ss)	Azure (mm: ss)
2 nodes	15:54	10:22
5 nodes	16:29	10:38
7 nodes	16:09	10:17

Table II presents the average runtime for the CI+CD pipeline across different node configurations on AWS and Azure. The scaling nodes are categorized as 2, 5, and 7 nodes, with the runtime measured in minutes and seconds. For 2 nodes, AWS has an average runtime of 15:54, while Azure achieves 10:22. As the node count increases to 5, AWS takes 16:29, whereas Azure maintains a faster runtime of 10:38. At 7 nodes, AWS records 16:09, compared to Azure's 10:17. These outcomes prove Azure superior at deployment speed irrespective of the type of node.

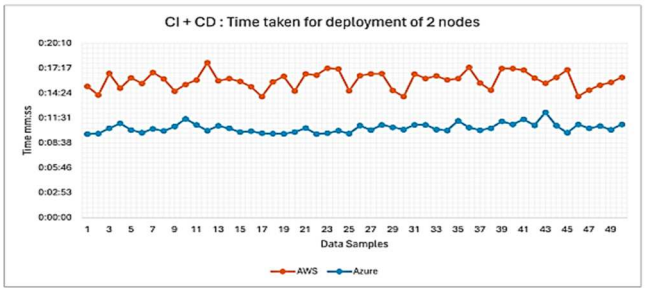

Fig. 8. CI+CD Pipeline runtime: 2 nodes

Figure 8 shows the CI+CD pipeline runtime for deploying 2 nodes across 50 samples, comparing AWS and Azure. AWS (orange line) consistently has a higher runtime, fluctuating between 14 and 18 minutes, while Azure (blue line) maintains a lower runtime, ranging from 9 to 11 minutes. This highlights Azure's better efficiency in handling deployments for smaller node configurations.

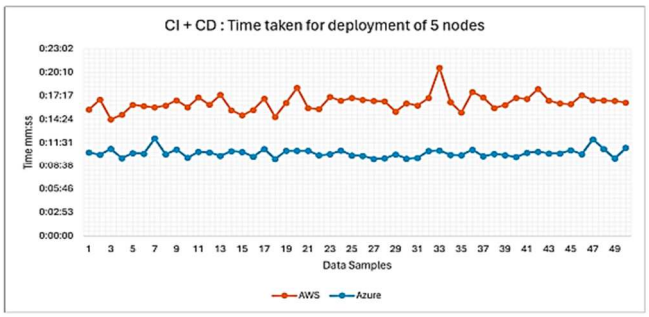

Fig. 9. CI+CD Pipeline runtime: 5 nodes

Figure 9 compares CI+CD pipeline runtimes for deploying 5 nodes across 50 samples on AWS and Azure. It can be observed that the orange line (AWS) has a relatively stable Runtimes between 15 to 18 minutes, although there are instances that slight increases up to 20 minutes. As for the blue one (Azure), the runtimes are much more constant, and range from 9 to 12 minutes on average. Azure has a better overall deployment setup and has improved efficiency and reliability for mid-sized nodes than AWS.

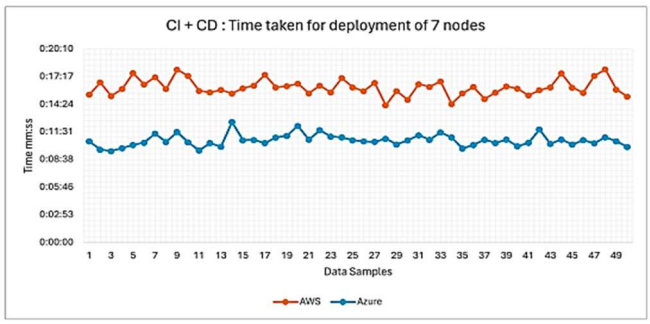

Fig. 10. CI+CD Pipeline runtime: 7 nodes

Figure 10 illustrates the CI+CD pipeline runtime for deploying 7 nodes across 50 samples on AWS and Azure. The orange line denoting AWS exhibits R-values varying between 15 and 18 min, slight variation from the threshold skewed to the higher range with a trace nearing 19 min sometimes. The blue color (Azure) denotes faster and least variance run time most of the time between 9-11 minutes. Azure is more advantageous than AWS in terms of deployment at scale for large node deployments proves Azure as best for high performance CI+CD.

Node scaling has different effects on both AWS and Azure because they employ unique approaches to the overall structure of their services. While Azure has optimized its virtual machine provisioning and Kubernetes orchestration is faster in scaling the node configuration, AWS may get slower due to complicated layered infrastructure and dependency on different instance types. Variability in the times of AWS deployment is attributed to server load, which affects performance, resource availability, and network congestion. Comparing with the industry average and other cloud vendors, Azure is considered to be faster in deploying, especially Kubernetes, than AWS. This efficiency further translate into better utilisation of resources thus Azure has less CPU and memory utilisation during scaling. Azure's CI+CD faster throughput can decrease time-to-market and scale quickly, it's possible to deploy more frequently and quickly, with smaller lead time, and is faster in responding to market call, while handling larger workload efficiently.

V. Conclusion And Future Scope

The practices of Continuous Integration/Continuous Deployment (CI/CD) pipelines which are important in improving web applications development and deployment in cloud environments. Comparing both AWS and Azure platforms used in the research shows that Azure outperforms AWS and other platforms in terms of deployment speed, runtime, and scalability, especially in Kubernetes multi-node environments. However, Azure is cheaper, has lower latency, stable performance and utilizes resources much better than Amazon, which is why developers want to deploy their applications faster and more efficiently. However, the study has some limitations that suggest that further related studies should consider results of broader evaluations involving various types of applications, regions, and realistic conditions.

Future Projects for to get a more thorough insight of CI/CD performance, this study might be expanded to include other cloud platforms as IBM Cloud and Google Cloud Platform (GCP). Future research should concentrate on incorporating cutting-edge technologies like ML and AI to improve CI/CD pipelines, especially for optimization, anomaly detection, and predictive analytics. However, there is potential to study the methods to manage the security risks, compliance, and issues in the hybrid and multi-cloud environment.

References

[1] R. Goyal, "A Review of Continuous Integration and Continuous Delivery (CI/CD) Practices in Modern Software Development," *Int. J. Core Eng. Manag.*, vol. 7, no. 06, pp. 49–59, 2023.

[2] R. Bishukarma, "Optimising Cloud Security in Multi-Cloud Environments : A Study of Best Practices," *TIJER – Int. Res. J.*, vol. 11, no. 11, pp. 590–598, 2024.

[3] S. Arora and A. Tewari, "Fortifying Critical Infrastructures: Secure Data Management with Edge Computing," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 3, no. 2, pp. 946–955, Aug. 2023, doi: 10.48175/IJARSCT-12743E.

[4] I.-C. Donca, O. P. Stan, M. Misaros, D. Gota, and L. Miclea, "Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects," *Sensors*, vol. 22, no. 12, p. 4637, Jun. 2022, doi: 10.3390/s22124637.

[5] R. Goyal, "The Role of Business Analysts in Information Management Projects," *Int. J. Core Eng. Manag.*, vol. 6, no. 9, pp. 76–86, 2020.

[6] B. Naveen, J. K. Grandhi, K. Lasya, E. M. Reddy, N. Srinivasu, and S. Bulla, "Efficient Automation of Web Application Development and Deployment Using Jenkins: A Comprehensive CI/CD Pipeline for Enhanced Productivity and Quality," in *International Conference on Self Sustainable Artificial Intelligence Systems, ICSSAS 2023 - Proceedings*, 2023. doi: 10.1109/ICSSAS57918.2023.10331631.

[7] P. Khare and S. Shrivastava, "Transforming KYC with AI: A Comprehensive Review of Artificial Intelligence-Based Identity Verification," *J. Emerg. Technol. Innov. Res.*, vol. 10, no. 12, pp. 525–531, 2023.

[8] P. Khare and S. Srivastava, "AI-Powered Fraud Prevention: A Comprehensive Analysis of Machine Learning Applications in Online Transactions," *J. Emerg. Technol. Innov. Res.*, vol. 10, no. 12, 2023.

[9] J. Thomas, K. V. Vedi, and S. Gupta, "A Survey of E-Commerce Integration in Supply Chain Management for Retail and Consumer Goods in Emerging Markets," *J. Emerg. Technol. Innov. Res.*, vol. 10, no. 12, 2023.

[10] R. Bishukarma, "Scalable Zero-Trust Architectures for Enhancing Security in Multi-Cloud SaaS Platforms," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 3, no. 3, pp. 1308–1319, Jul. 2023, doi: 10.48175/IJARSCT-14000S.

[11] A. Muñoz, A. Farao, J. R. C. Correia, and C. Xenakis, "P2ISE: Preserving Project Integrity in CI/CD Based on Secure Elements," *Information*, vol. 12, no. 9, p. 357, Aug. 2021, doi: 10.3390/info12090357.

[12] R. Arora, S. Kumar, N. Jain, and M. T. Nafis, "Revolutionizing Healthcare with Cloud Computing: Superior Patient Care and Enhanced Service Efficiency," *SSRN*. 2024. doi: 10.2139/ssrn.4957197.

[13] P. K. Vishwakarma and M. R. Suyambu, "The Impact of Incorporation Renewable Energy on Resilience and Stability of Power Grid System," *Int. J. Innov. Sci. Res. Technol.*, vol. 9, no. 10, pp. 1519–1526, Nov. 2024, doi: 10.38124/ijisrt/IJISRT24OCT1529.

[14] B. W. Wirtz, O. Schilke, and S. Ullrich, "Strategic Development of Business Models," *Long Range Plann.*, vol. 43, no. 2–3, pp. 272–290, Apr. 2010, doi: 10.1016/j.lrp.2010.01.005.

[15] V. V Kumar, M. Tripathi, S. K. Tyagi, S. K. Shukla, and M. K. Tiwari, "An Integrated Real Time Optimization Approach (IRTO) for Physical Programming Based Redundancy Allocation Problem," *3rd Int. Conf. Reliab. Saf. Eng.*, no. August, 2007.

[16] B. Priya and T. Gnanasekaran, "State of the Art - Optimization Techniques in Cloud Environment," *Int. J. Eng. Technol.*, vol. 7, no. 3.3, p. 56, Jun. 2018, doi: 10.14419/ijet.v7i2.33.13854.

[17] F. Zampetti, S. Geremia, G. Bavota, and M. Di Penta, "CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, Sep. 2021, pp. 471–482. doi: 10.1109/ICSME52107.2021.00048.

[18] P. S. Chatterjee and H. K. Mittal, "Enhancing Operational Efficiency through the Integration of CI/CD and DevOps in Software Deployment," in *2024 Sixth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, IEEE, Apr. 2024, pp. 173–182. doi: 10.1109/CCICT62777.2024.00038.

[19] T. Klooster, F. Turkmen, G. Broenink, R. Ten Hove, and M. Bohme, "Continuous Fuzzing: A Study of the Effectiveness and Scalability of Fuzzing in CI/CD Pipelines," in *Proceedings - 2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing, SBFT 2023*, 2023. doi: 10.1109/SBFT59156.2023.00015.

[20] D. Sushma, M. K. Nalini, R. Ashok Kumar, and M. K. Nidugala, "To Detect and Mitigate the Risk in Continuous Integration and Continues Deployments (CI/CD) Pipelines in Supply Chain Using Snyk tool," in *7th IEEE International Conference on Computational Systems and Information Technology for Sustainable Solutions, CSITSS 2023 - Proceedings*, 2023. doi: 10.1109/CSITSS60515.2023.10334136.

[21] S. A. I. B. S. Arachchi and I. Perera, "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management," in *2018 Moratuwa Engineering Research Conference (MERCon)*, IEEE, May 2018, pp. 156–161. doi: 10.1109/MERCon.2018.8421965.

[22] S. S. Kataru, R. Gude, S. Shaik, L. V. N. S. S. S. Kota, S. Srithar, and R. M. Balajee, "Cost Optimizing Cloud based Docker Application Deployment with Cloudfront and Global Accelerator in AWS Cloud," in *International Conference on Sustainable Communication Networks and Application, ICSCNA 2023 - Proceedings*, 2023. doi: 10.1109/ICSCNA58489.2023.10370161.

[23] V. S. Thokala, "Efficient Data Modeling and Storage Solutions with SQL and NoSQL Databases in Web Applications," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 2, no. 1, pp. 470–482, Apr. 2022, doi: 10.48175/IJARSCT-3861B.

[24] H. Mandale, P. Dhobale, P. Ganorkar, and D. Daware, "Jenkins CI/CD Pipeline with Github Integration," *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 5, no. 7, Jul. 2023, doi: 10.56726/IRJMETS42991.

[25] A. Singh, "A Comparison on Continuous Integration and Continuous Deployment (CI/CD) on Cloud Based on Various Deployment and Testing Strategies," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 9, no. VI, pp. 4968–4977, Jun. 2021, doi: 10.22214/ijraset.2021.36038.