International Journal of Core Engineering & Management
Volume-7, Issue-10, 2024 ISSN No: 2348-9510

ANALYSIS OF CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT (CI/CD) PIPELINES FOR AUTOMATED CLOUD INFRASTRUCTURE MANAGEMENT

Srinivasa Rao Singireddy
Architect, The Clorox Services Company, Pleasanton, CA.
srinivasarao.singireddy@gmail.com

Abstract

CI/CD pipelines are in the middle of software development since they orchestrated several programmatic steps by enhancing their efficiency. This work focuses on the CI/CD pipeline management in cloud environments with focus on cloud constructs especially using Amazon Web Service for cloud infrastructure automation. This research initiates by creating a local software project and involves connecting to the cloud version control system, (Bitbucket). This pipeline is going to have an instantaneous response towards code changes, and is going to test using Pytest afterwards, before transferring the code within a containerized context using Docker, which will run on AWS EC2. This work compares AWS with GCP on other parameters such as deployment time, rollback time, scalability, cost and reliability. The findings indicate that AWS offers superior performance to GCP by providing a robust and Table CI/CD platform, required by companies that incorporate agile software development strategies.

I. INTRODUCTION

New products or services may be developed where evolutionary or rapid/short development cycles are being done in shorter time. Agile methods differ from monolithic approaches to development; the work progresses in parallel with the development of smaller projects, which allows for better organisation of changes.[1]. Developers begin by taking a bird's-eye view of the whole lifecycle of a feature, including its conception, definition, prioritisation, implementation team selection, resource allocation, and execution planning. After these steps are finished, only then do the developers consider the implementation.[2]. There seems to be complete anarchy and a bewildering quantity of data at first.

Traditional software development approaches just can't cut it in the modern corporate world. The potential for agile methods to increase the velocity, efficiency, and adaptability of the software development life cycle is a major selling point for software development companies.[3]. Because of this, several academics and businesses are working to create products that can produce and automate the Continuous Integration (CI), Continuous Delivery (CD), and Continuous Deployment (CDT) processes.[4].

Nonetheless, the most crucial lean and agile concepts may be adhered to via the CI/CD/CDT process. The CI approach offers a lot of benefits, two of which are removing the restrictions on the number of times an implementation may be carried out and lowering risk as much as is practical to build reliable and error-free software.[5][6]. The primary reasons why businesses invest in CD are the many advantages it offers, such as faster time to market, higher quality products, customer satisfaction, more valid releases, more productivity, and overall efficiency. IaaS is now the go-to for software and mobile app development, therefore CI, CD, and CDT have unquestionably become

integral parts of cloud computing.[7][8]. The concept behind CD is to discover methods to reliably and efficiently supply high-quality software. A software development approach known as CI involves developers routinely merging their code changes into a common repository. Following this, automated builds and tests are executed.[9].

In light of the rising complexity and acceleration of software development, there is a need for reliable methodologies that increase both efficiency and reliability. The stimulus of this study can be marked in the increasing demand for more effective and credible methodologies for software development as the field is increasingly characterized by complexity. Companies transitioning to agile practices discover that required development efficiency and assurance of code quality is CI/CD integration. About this research, this study aims at understanding how CI/CD pipelines work within cloud environments, particularly using AWS for automation of Cloud Infrastructure management and subsequently compare the performance with Google Cloud Platform (GCP)[10]. This research aims at increasing organizational flexibility and effectiveness of software development through the development of a structured procedure that integrates application development, testing, and deployment.

- The investigation provides a logical way of performing CI/CD pipelines, beginning with the local project development stage up to the final automated deployment stage; it serves as a helpful reference to organizations that seek to improve their development methodology.
- The comparison provides insight into the CI/CD needs of organizations by identifying the merits and demerits that define AWS and GCP as crucial performance indicators that assist organisations in selecting the right cloud provider.
- The incorporation of automated testing by Pytest as part of the CI/CD approach is important, as it highlights the necessity to maintain the code quality and admissibility in the current discussions about effective IT solutions.
- It highlights the strengths of using containers through Docker case and discusses how deployment density and its standard enhance current deployment strategies within cloud environments.
- The findings advocate for the adoption of CI/CD pipelines as a means to achieve greater agility and efficiency in software development, positioning AWS as a strategic choice for organizations looking to thrive in an increasingly competitive digital landscape.

The following paper organized as: Section II provide the literature review on this topic, Sections III and IV provide the methodology for CICD and Results discussion at last Section V provide the conclusion and future work.

II. LITERATURE REVIEW

As companies continue to look for lightweight solutions to remain "alive" in the industry, the author notices that CI/CD are significant themes when reading academic and research papers. In most cases, the publications I read included detailed descriptions of commercially available, widelyaccepted solutions, both open-source and proprietary. As part of their critical descriptions of the implementation stages, multiple authors are attempting to delve into certain CI/CD proprietary solutions.

In,Kadiu, (2022) probe approaches to cloud deployment that make use of CI/CD pipelines and the container technologies Docker and Kubernetes. Plan, build, containerize, and deploy an Angular application using a pipeline; that is the purpose. Effective software planning, development, containerization, and application deployment into a cloud-distributed environment are all covered in this study[11].

In the,Bhavsar, Rangras and Modi, (2021) research primarily aims to provide a broad outline of how CI/CD might automate the build and deployment process and efficiently manage rolling upgrades, hence resolving the deployment and rolling upgrade problems. In the event of a failure in either the staging or production environments, it also offers a fault tolerance mechanism and a means of applying the disaster recovery plan independent of the underlying cloud platform.[12].

In,Ghimire (2020) showcases the ideas and practical application of CI and CD pipelines, with a focus on cloud software deployment with minimal tools to ensure seamless integration with software developers' and small businesses' workflows and to avoid excessive costs. The Bitbucket git repository and AWS cloud are used in this application. Bitbucket offers cloud-based git version control services, while AWS offers a broad variety of cloud services for many reasons, including software deployment[13].

In,Helsinki and Wikström, (2019) use a case study to examine a software firm in Finland and identify the pros and downsides of CI/CD adoption. Interviews that were semi-structured were used to carry out the research. The advantages of CD that have been discovered include simplified deployments, improved quality assurance, and quicker iteration. Consistent with earlier research, this study came to similar conclusions. They also go into how the example firm used cloud computing and current CI/CD solutions like GitLab[14].

In,Bhuvana, (2024)delves into the design, architecture, and best practices of CI/CD pipeline implementation in Agile organisations, with an emphasis on automating build, test, deployment, and monitoring procedures. Case studies and examples of organisations effectively employing CI/CD pipelines in Agile Software Development are also shown in the report, highlighting the advantages and results. Academics, professionals, and businesses interested in developing and improving CI/CD pipelines for Agile product development[15].

In,Soni, (2015) an organization's success hinges on its ability to reliably and quickly supply creative ideas. By automating the process of creating and deploying applications across environments, DevOps culture enhances performance and quality assurance while quickly expanding the agile approach.The advent of CI and CD has been a boon to conventional methods of application development and release management, allowing for the continuous delivery of high-quality artefacts to clients with integrated feedback in real-time[16].

This Table 1 summarizes the essential aspects of each study, providing a quick reference for comparing methodologies, results, and future research directions for CI/CD in cloud.

International Journal of Core Engineering & Management
Volume-7, Issue-10, 2024 ISSN No: 2348-9510

Table 1: Summary of the related work for Continuous Integration/Continuous Deployment (CI/CD) Pipelines for Automated Cloud Infrastructure

Reference	Techniques	Results	Key Findings	Limitations/Future Work
[11]	Research Cloud Deployment Methodologies using Docker and Kubernetes with CI/CD Pipelines	Efficient cloud deployment process	Comprehensive workflow from planning to deployment	Future research could explore different cloud providers or multi-cloud strategies.
[12]	CI/CD for Deployment and Rolling Upgrades	Improved deployment efficiency	Automation reduces deployment risks and improves disaster recovery	Investigate specific challenges faced during CI/CD implementation in various organizations.
[13]	CI/CD Pipelines Implementation in Cloud	Streamlined deployment process	Minimal toolset integration is cost-effective for small companies	Explore the long-term sustainability of using fewer tools in larger projects.
[14]	Case Study on CI/CD Implementation in Finnish Software Company	Identified benefits and challenges of CI/CD	Faster iteration, quality assurance, and easier deployments	Further studies could quantify the impacts of CI/CD on performance metrics.
[15]	Best Practices for Implementing CI/CD in Agile Environments	Successful implementation case studies	Enhances teamwork, time-to-market, and product quality	Future work may involve developing tailored CI/CD strategies for different Agile frameworks.
[16]	Delivering Innovative Ideas with DevOps Culture	Improved application delivery performance	CI/CD facilitates continuous feedback and quality releases	Future research could assess the impact of CI/CD on team dynamics and culture.

III. METHODOLOGY FOR CI/CD

The CI/CD pipeline is to be integrated at the proposed methodology for its implementation focuses on the systematic approach to the synchronization of the software creation process with testing and deployment phases. First, a new software project is developed in a local environment, using tools such as Flask for the backend environment and PyCharm for the development environment. The project is then linked where it is connected to Bitbucket as the cloud-based version control repository where branches helps in collaborative development. At the moment, CI/CD pipeline is con Figure to run when there is change in the code with Pytest for testing purposes. It first runs a number of quality checks against the code according to a set of tests defined in the pipeline specification and then builds from source inside a containerized environment provided by Docker. This architecture allows a very easy deploy of developed code to production, whereby only code that passes through test is deployed.

International Journal of Core Engineering & Management
Volume-7, Issue-10, 2024 ISSN No: 2348-9510

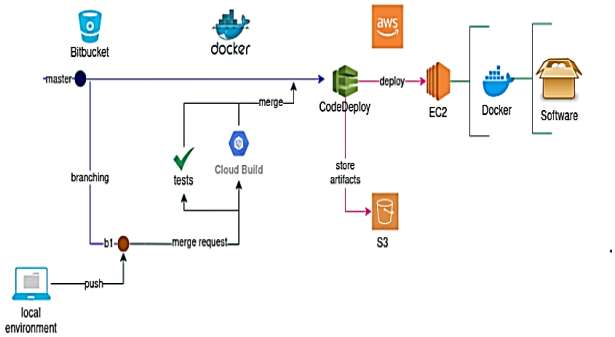

Figure 1: Architecture of the Implemented CICD Pipeline

As presented in Figure1. below, the local environment corresponding to the bitbucket git repository represents the entry point of the pipeline. Though the terminology of the repository made it clear that, there is another branch called the master branch in parallel to other branches. A working branch b1 is depicted in the image where locally developed code is taken and pushed.

Once an initial pipeline is created, additional changes are made to enable its fully automated deployment to a cloud environment on Amazon Web Services EC2. I was using CodeDeploy as a deployment tool, which can deploy applications from source control, and S3 for storing artefacts, which is useful for configuration and management of the deployment process. The architecture is highly scalable and can work well for both of the solutions, where application containerization offers flexibility regarding resource usage. The last stage is the distribution of the application within the framework of using a docker to test the functionality available to users. This methodology is also effective for Agile development practices necessary for the quick release and fix cycles because this maintains quality and reliability in the practice.

Steps of proposed methodology for implementing the CI/CD pipeline:

1) **Step 1: Create a Local Software Project**

Start with the creation of a minimal software project in the local environment you have. Use Flask for backend programming. One may use different software for easier writing and organization of code, including scenarios and architecture; the most popular one is PyCharm.

2) **Step 2: Select a Cloud Software Repository**

Select a version control system to be hosted in the cloud to help handle project code. Therefore, Bitbucket is chosen because of its compatibility with Git and integration.

3) **Step 3: Choose a Cloud Service Provider**

The next step is to consider the choice of a cloud service provider for the implementation process. AWS is selected, and an instance of EC2 is created as a basic platform that will let the application run on a virtual machine.

4) **Step 4: Establish the Initial CI/CD Pipeline**

Create an initial CI/CD pipeline setup in way that integrates your local source code repository with Bitbucket and AWS. This will be a pipeline that will automate the integration and deployment of our codes.

International Journal of Core Engineering & Management
Volume-7, Issue-10, 2024 ISSN No: 2348-9510

5) **Step 5: Implement a Testing Framework**

Implement a testing framework that checks on code quality. Pytest is chosen for writing and running tests on the developed software. This is an important step in order to ensure reliability in the deployment steps that are implemented.

6) **Step 6: Re-implement Pipeline with Integrated Testing**

The testing framework needs to be added to the original pipeline plan as the new integrated part. This makes it so that whenever code is pushed to the repository the tests are run automatically.

7) **Step 7: Configure Required Cloud Services**

Provide the required cloud services, to allow the execution of the CI/CD pipeline. This is done by engaging S3 for artefact storage as well as CodeDeploy for automated deployment of the application[17].

8) **Step 8: Add Containerization Capabilities**

Integrate Docker to allow the creation of Containers in the application Stream. This step makes the application run at independent stages without much variation in performance between stages of deployment.

9) **Step 9: Deploy Containerized Environment**

Setup the EC2 instance to support containerisation. Run the application in a Docker container on the EC2 host making it easier to manage and scalable.

10) **Step 10: Run Automated Tests and Deployments**

It is an excellent pipeline to run automated tests and deployment in the cloud. The above step makes sure that every piece of code change has been tested before finally releasing it to the live environment.

11) **Step 11: Conduct User Testing**

User testing should be carried out after deployment in order to confirm that the application worked as intended. This is a critical step needed to find out what challenges exist in the production environment.

IV. RESULT ANALYSIS AND DISCUSSION

It is in this section to present the findings and analysis of the CICD pipeline's AWS cloud environment. AWS is a cloud provider that offers a wide range of business cloud services, such as server instances, storage, networking, content delivery networks, email, repositories, and more[18]. The system relies on Elastic Compute Cloud (EC2)[19]for primary production, CodeDeploy for Continuous Deployment, and S3 for artefact storage. Artefacts are objects that are generated along the process. An Elastic Compute Cloud (EC2) instance from Amazon is a kind of virtual server that can be easily scaled up or down in terms of resources based on actual demand. The following steps are taken to carry out the implementation, as shown in the process outline that was initially provided.

International Journal of Core Engineering & Management
Volume-7, Issue-10, 2024 ISSN No: 2348-9510

A. Production Environment

Among the many parts of the puzzle, this one is crucial. Even if they have never used a CICD pipeline, developers and teams should be familiar with the first two steps (with the possible exception of the Bitbucket repository's pipeline feature) due to the prevalence of version control technology and various environment setups in software development.[13]. The setup process begins by spinning up (allocating and launching) an EC2 instance in the AWS cloud. To do so, first establish an AWS account and connect to the console using the username and password 25-root. A console, also known as the AWS Management Console, is the user interface that enables you to begin utilising the various AWS cloud services. Figure 2 depicts the AWS sign-in page.

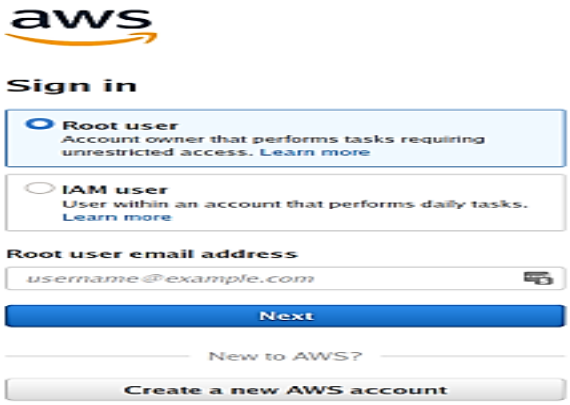

Figure2: AWS Sign in page

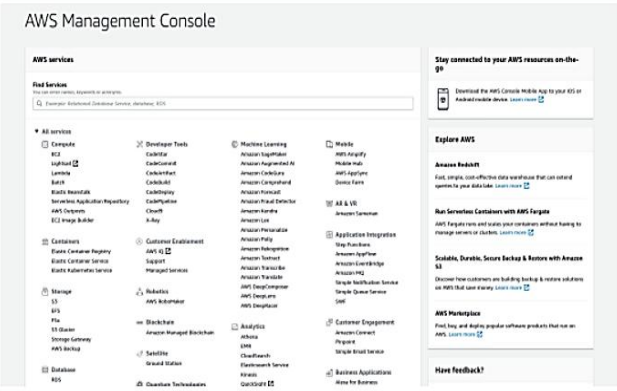

Figure3: AWS Management Console

The AWS Management Console is seen in Figure 3. Artefact storage, services that manage various aspects of the deployment process, and the deployment server are all resources that are developed for use in the production environment. The purpose of providing these tools is to allow them some degree of autonomy within certain parameters. The next sections provide instructions on how to create and configure these services.

B. EC2 Instance Creation

There are several image options available when starting an EC2 instance, including Red Hat Enterprise Linux, Ubuntu, and Amazon Linux, and the majority of them function flawlessly with

International Journal of Core Engineering & Management
Volume-7, Issue-10, 2024 ISSN No: 2348-9510

the configuration. Nevertheless, an Amazon Linux AMI is selected for this implementation. Selecting an AMI for an EC2 instance is shown in Figure 4.

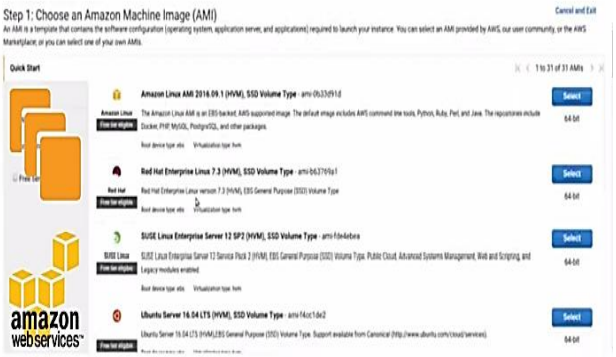

Figure4: Choosing AMI for EC2 instance

A free tier plan may be all that's needed for basic use cases like this implementation and smaller web projects. Therefore, as seen in Figure 5 below, after the prior procedure of starting an EC2 instance, a free tier plan is selected on the subsequent screen. Figure 5 illustrates the free tier t2 micro plan (marked with a red dot and a number 1) and the configuration step button (red dot and a number 2), as well as the navigation between the two steps.

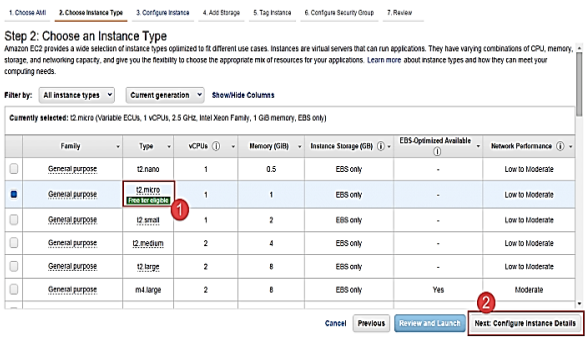

Figure5: Choosing a free tier plan for EC2 instance

An EC2 instance is created by following the setup procedures as suggested by the user interface. One crucial step that cannot be skipped during creation is associating the EC2 instance with the IAM role for CodeDeploy. In order to have access to S3 storage, a policy must be associated to the IAM role. The SSH key is downloaded and saved in a safe place in case the EC2 instance that was established so far has to be accessed later for setup (as will be discussed in section 3.4.8).

C. Configuring EC2 instance as a production server

The SSH key is used to establish a secure connection between the local terminal and the EC2 instance. The system must be updated as soon as the connection is established successfully. While not necessary, it is advised as a good practice to do this step.

After the system upgrade, the software packages needed by the code deploy agent are installed. These packages are utility software packages. These programs include Ruby, a language for

International Journal of Core Engineering & Management
Volume-7, Issue-10, 2024 ISSN No: 2348-9510

programmers, and a GNU tool that is often used to access files over protocols including HTTP, HTTPS, FTP, and FTPS.
These steps are carried out by issuing the following commands:

```
sudo yum update
sudo yum install ruby
sudo yum install wget
cd /home/ec2-user
wget https://artifacts-storage-itp.s3.eu-north-1.amazonaws.com/la-
test/install
```

The S3 storage unit, artifacts-storage-it, is also known as a bucket, and the region where the resources, including EC2 and S3, are setup is north-1. Executing the final line triggers the download of the file required to install the code deploy agent. The following command grants the necessary execute permission for this file:

```
chmod +x ./install
```

This program, code deploy agent, may be installed using the following command:

```
sudo ./install auto
```

The following commands are used to confirm that the code deploy agent is operating, even though it should be doing so automatically after installation:

```
sudo service codedeploy-agent start
```

D. Environment Variable Configuration in Bitbucket Pipelines

Entering the values for the environment variables into the bitbucket pipeline configuration is the next step. You can find the pipeline settings in the repository settings, and the repository variables are where you set up the environment variables. The preset variables and their position in the Bitbucket repository settings are shown in Figure 6.

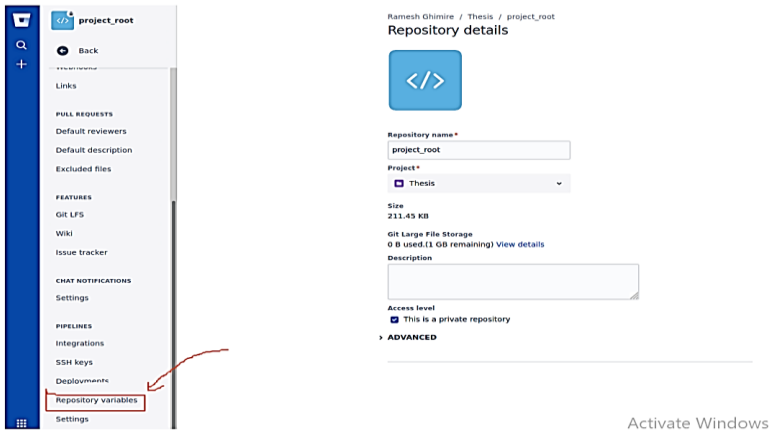

Figure6: Environment variable settings location for Bitbucket Pipelines.

International Journal of Core Engineering & Management
Volume-7, Issue-10, 2024 ISSN No: 2348-9510

To enter the user's details and credentials to access AWS resources, choose the Repository variables option. This will bring up the environment variable configuration box. The format in which this information is acquired during the resource's setup procedure in the AWS interface must be precisely followed.

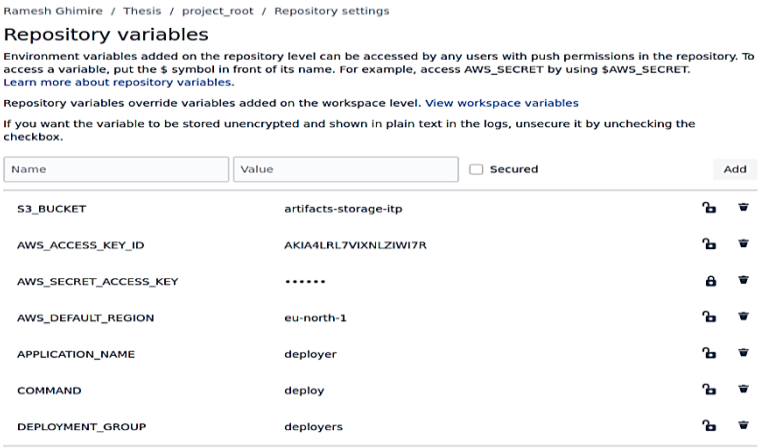

Figure7: Setting up environment variables

Figure 7 shows that the environment variables are named in uppercase and relate to the already-con Figure resources in the AWS cloud, like the deployer application, deployment group, S3 bucket details, IAM user profile, and so on. Specifically, this data is required for the AWS cloud-based production environment deployment step of the CICD process. The initial full configuration of the CICD pipeline is now complete; the next stage is to add pipeline functionality, such as testing, deployment as Docker containerized deployments, and the capacity to utilize the pipeline for several applications.

Table 2: Performance Comparison of AWS and GCP in CI/CD Pipelines

Criteria	AWS	Google Cloud Platform (GCP)
Deployment Speed	5 minutes	7 minutes
Rollback Efficiency	2 minutes	4 minutes
Scalability	High (supports auto-scaling)	Moderate (manual scaling needed)
Cost	$0.05 per deployment	$0.07 per deployment
Automated Testing Integration	Effective, reduces errors by 30%	Effective, reduces errors by 25%
Handling of High Traffic	Excellent	Good
Performance During Peak Load	95% uptime	90% uptime

The performance comparison between AWS and Google Cloud Platform (GCP) in CI/CD pipelines reveals several key differences in operational efficiency and capabilities, shown in Table 2. AWS demonstrates superior deployment speed, averaging just 5 minutes compared to GCP's 7 minutes, while also showcasing greater rollback efficiency at 2 minutes versus GCP's 4 minutes. Scalability is another area where AWS excels, offering high auto-scaling support, whereas GCP requires manual scaling, limiting its responsiveness to variable workloads. In regards to the costs, the overall deployment cost is relatively cheaper with AWS costing $0.05 per deployment while GCP costing $0.07. AWS and GCP's services in automated testing integration are efficient with AWS standing out slightly better than GCP at error reduction by 30% from GCP's 25%. Moreover, AWS performs well during traffic spikes and load, which proves its ability to stay at 95% uptimes as opposed to GCP's 90%. All in all, it seems like AWS provide more solid and efficient solution for CI/CD pipelines, therefore popular among the companies that start with fast rates of software deployment and usage.

IV. CONCLUSION AND FUTURE SCOPE

The usage of CI and CD pipelines has been proven to be valuable contributions to automation in cloud context in order to bring needed scalability, dynamic and efficiency in the context of the software development domain. Considering the CI/CD pipelines concerning the automated cloud infrastructure management, there are significant differences while preferring to use AWS rather than GCP. Such disciplined approach helps to achieve exceptionally fast and short development and deployment cycles while maintaining very high quality of code and its reliability with the help of integrated testing. Chicco also uses metrics concerning performance, where AWS provides faster deployment, more efficient roll backs and greater scalability than GCP at less cost. The findings also capture the essence that AWS is better suited for organizations that have sheer dynamism and speed as their objective within software development projects. Because of that, sophisticated systematic CI/CD should be thought of as one of the key infrastructural pillars as cloud technologies mature, to provide efficiency and reach competitive advantages in the digital environment.

Although this study points out how beneficial AWS is for CI/CD pipelines, we should bear in mind certain limitations. The attention was mainly on AWS and GCP, which could have led to a blind spot for other cloud companies that have competitive capabilities. Also, the research centered on particular metrics; accordingly, larger operational factors, including user experience and support services, were not included. Future research should widen the evaluation to include a greater selection of cloud platforms and investigate the merging of emerging solutions like serverless computing and AI-driven automation in CI/CD pipelines. Moreover, undertaking sustained studies to evaluate the effects of different CI/CD strategies on true projects will provide expanded knowledge of preferred practices and optimization options.

REFERENCES

1. S. Saeed, N. Z. Jhanjhi, M. Naqvi, and M. Humayun, "Analysis of software development methodologies," Int. J. Comput. Digit. Syst., 2019, doi: 10.12785/ijcds/080502.
2. S. R. Doddaguni, S. Asif S, M. MN, and R. R, "Understanding SDLC using CI/CD Pipeline," Int. J. Soft Comput. Eng., 2020, doi: 10.35940/ijsce.f3405.059620.

3. Sendy, T. Kandaga, A. Widjaja, H. Toba, R. Joshua, and J. Narabel, "Continuous Integration and Continuous Delivery Platform Development of Software Engineering and Software Project Management in Higher Education," J. Tek. Inform. dan Sist. Inf., vol. 7, no. 1, Apr. 2021, doi: 10.28932/jutisi.v7i1.3254.
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, 2022, doi: 10.3390/s22124637.
5. S. Kamath, M. M. Manohara Pai, S. Vignesh, and G. Darshan, "Revolutionizing Cloud Infrastructure Management: Streamlined Provisioning and Monitoring with Automated Tools and User-Friendly Frontend Interface," in 2023 3rd International Conference on Intelligent Technologies, CONIT 2023, 2023. doi: 10.1109/CONIT59222.2023.10205728.
6. M. Soni, "End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery," in Proceedings - 2015 IEEE International Conference on Cloud Computing in Emerging Markets, CCEM 2015, 2016. doi: 10.1109/CCEM.2015.29.
7. M. F. Albaihaqi, A. N. Wilda, and B. Sugiantoro, "Deploying an Application to Cloud Platform Using Continous Integration and Continous Delivery," Proceeding Int. Conf. Sci. Eng., 2020, doi: 10.14421/icse.v3.513.
8. A. Alanda, H. A. Mooduto, and R. Hadelina, "Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures," JITCE (Journal Inf. Technol. Comput. Eng., 2022, doi: 10.25077/jitce.6.02.50-56.2022.
9. S. P. Sinde, B. Thakkalapally, M. Ramidi, and S. Veeramalla, "Continuous Integration and Deployment Automation in AWS Cloud Infrastructure," Int. J. Res. Appl. Sci. Eng. Technol., 2022, doi: 10.22214/ijraset.2022.44106.
10. J. Thomas, "The Effect and Challenges of the Internet of Things (IoT) on the Management of Supply Chains," Int. J. Res. Anal. Rev., vol. 8, no. 3, pp. 874–878, 2021.
11. G. Kadiu, "Automizing software planning, development, and deployment processes of a 3-tier architecture web application using .Net, Angular, and SQL Server by integrating the software into Azure DevOps and Cloud Infrastructure with CI/CD Pipelines, Docker, and Kuber," 2022.
12. S. Bhavsar, J. Rangras, and K. Modi, "Automating Container Deployments Using CI/CD," 2021, pp. 423–429. doi: 10.1007/978-981-15-4474-3_47.
13. R. Ghimire, "Deploying Software in the Cloud with CI/CD Pipelines," 2020.
14. A. W. Helsinki and A. Wikström, "Benefits and challenges of Continuous Integration and Delivery-A Case Study," 2019.
15. J. Bhuvana, "Implementing Continuous Integration and Deployment Pipelines in Agile Software Development," Int. Res. J. Mod. Eng. Technol. Sci., no. 03, pp. 2086–2091, 2024, doi: 10.56726/irjmets50470.
16. M. Soni, "End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery," in 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), 2015, pp. 85–89. doi: 10.1109/CCEM.2015.29.
17. J. Thomas, "Enhancing Supply Chain Resilience Through Cloud-Based SCM and Advanced Machine Learning : A Case Study of Logistics," J. Emerg. Technol. Innov. Res., no. September 2021, 2021.

18. K. Das, A. Mandal, D. Das, and R. Mukherjee, "Contribution of AWS on Cloud Computing Technology," Int. J. Appl. Eng. Res., 2023, doi: 10.37622/ijaer/18.3.2023.203-209.
19. A. Choudhary, "A walkthrough of Amazon Elastic Compute Cloud (Amazon EC2): A Review," Int. J. Res. Appl. Sci. Eng. Technol., 2021, doi: 10.22214/ijraset.2021.38764.