iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** VI **Month of publication:** June 2021

DOI: https://doi.org/10.22214/ijraset.2021.36038

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue VI June 2021- Available at www.ijraset.com

A Comparison on Continuous Integration and Continuous Deployment (CI/CD) on Cloud Based on Various Deployment and Testing Strategies

Avishek Singh[1], Vibhakar Mansotra[2]
Department of Computer Science and IT, University of Jammu.

Abstract: ***Continuous Integration/Continuous Delivery/Deployment (CI/CD) emphasizes the rapid release of small, incremental changes and the use of automation throughout the development process. CI/CD technique is central to DevOps and key to its success. Consumers expect to have continuous interaction with DevOps team so that they can provide their continuous feedback. DevOps is blending of two terms: development and operations which aims to provide conjoin approach to industry's software development and operation team job in software development lifecycle. Continuous practices, i.e., continuous integration, delivery, and deployment, are the software development industry practices that enable organizations to frequently and reliably release new features and products. Shows the comparison on deployments. With the increasing interest in literature on continuous practices, it is important to systematically review and synthesize the approaches, tools, challenges, and practices reported for adopting and implementing continuous practices.***
Keywords: Continuous Delivery and Continuous Deployment (CI/CD), CICD Pipeline, Deployment and Testing Strategies, Devops, Zero-Downtime Deployment, Kubernetes.

I. INTRODUCTION

Continuous deployment is a software engineering process that focuses on rapid delivery of software changes to end-users, and in this software engineering process incremental software changes are automatically tested, and frequently deployed to production environments. Facebook, GitHub, Netflix, and Rally Software are some of the many software companies who are using continuous deployment to deploy their product. Software companies who are using continuous deployment have reported several benefits of this software process, such as improved customer satisfaction, improved software quality, and savings in development effort.
Continuous delivery and continuous deployment are two software engineering processes that focus on delivering software changes quickly to end-users. Humble and Farley [2] defined continuous deployment as a software process that releases software changes automatically to end-users after they pass the required automated tests. According to Martin Fowler [3], continuous delivery is the software engineering process that builds software in such a way that it is releasable at any time; and continuous deployment is the software process that actually releases software to production as soon as they are ready, resulting in many deployments to production every day [4]. Humble and Farley and Fowler's definitions of continuous deployment are similar, though Fowler does not stipulate the use of automated tests. [2]. On the other hand, DevOps has emerged as a methodology that is involved in the entire product lifecycle through marketing, planning, human resources, and sales along with development and operations.[5]

II. CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT (CI/CD)

The creation of a continuous system or application in the information technology life cycle includes several elements, namely planning, design, development, testing, error correction, continued testing, and implementation in a production environment. In these stages there are three divisions involved, between developers, testers and operational units. For developers to concentrate on the speed of making the program code and the functional amount implemented into the program, the examiner will see how many errors and bugs are detected, for the operational unit will see in terms of system stability and the small risk of a system failure. With such workflows often leading to conflicts of interest, on the developer side trying to write program code quickly and sending it to testers, testers examine and test program code as long as necessary to reveal all bugs, and third parties operating departments are hesitant to make major changes in the program code, because this change has the potential to open access to all IT infrastructure to serious risks. Therefore, it is necessary to have a mechanism that harmonizes the functions of the three elements mentioned above. This thesis will analyze the application of the CI / CD concept (Continuous Integration), using cloud computing services that are implemented in financial technology companies using Amazon Web Services (AWS) as a VPS (Virtual Private Server), using

GitLab as a CI / CD application [6].

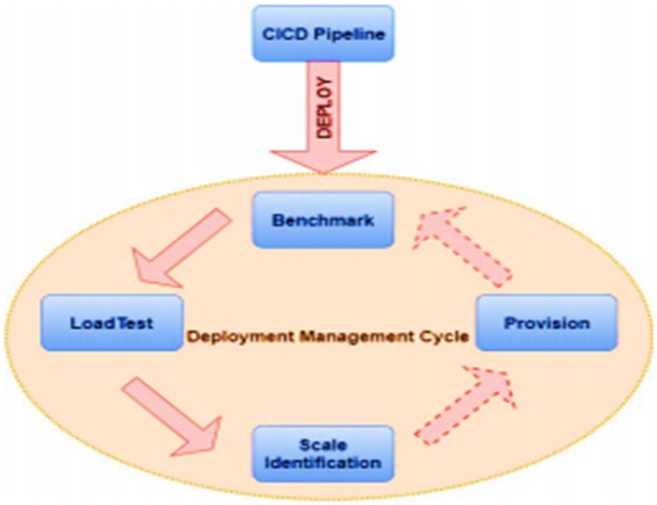


Fig.1 The deployment management cycle with CICD

A. Continuous Integration

CI **[7]** is a software development practice where team members integrate their work regularly and automate build, test and validate. It helps to find and fix bugs quickly and improve the quality of software. The source repository, version control system and CI server are the main components of the CI. As per AWS, more frequent commits to common codebase, maintaining single source repository, automating builds and automating testing are mentioned as challenges when following CI practices. Build automation, code stability, Analytics, CD enablement, faster releases & cost saving, improved productivity & code quality are the benefits when adopting CI practices.

B. CI/CD Pipeline

When an **[7]** organization tries to adopt CICD pipeline, they may not be able to adopt it at once. First, they have to practice CI in order to adopt CD. When moving from CI to CD and then Continuous Delivery to Continuous Deployment, this pipeline has reduced the manual process execution and finally full process becomes automated. The main difference between Continuous Delivery and the Continuous deployment is automation at production deployment.

C. CI/CD Tools

CI/CD tools **[7]** can be classified as Repository and Version Control tools, Build tools, Automation tools, Test Automation tools, and Monitoring tools.

1) *Repository and Version Control Tools:* Version Management is defined as managing the changes. Due to the support of distributed architecture and being opensource software, Git has become more popular and widely use version controlling system with CICD approaches
2) *Build Tools:* Ant, Maven and Gradle are famous builds tools available under open source. CI servers provide centralized control over build tools and as per Shahin, Bamboo, Jenkins, Cruise Control, Hudson, Sysphus, Hydra, TeamCity are few available CI servers. But in CI/CD approaches used Jenkins as their main CI server, due to its features such as customizable, scalable, cross platform support and security.
3) *Automation Tools:* Configuration Management (CM) is process of ensuring consistency in infrastructure, which is the key objective of automation with benefits such as cost reduction, reduce the complexity of release process, proper host management, and easy configuration, reliability, and efficiency and optimize resource utilization. CF Engine was the first modern open source CM tool released in 1993.
4) *Test Automation:* The idea of "test automation pyramid" in agile process was introduced by Mike Cohn. This pyr: - mid consists of three different levels as Unit, Service and UI. AWS has introduced another layer called Performance/ Compliance on top of service test layer.

D. CI/CD Benefits [8]

There are some advantages of CI/CD [8]:

1) *Faster Identification and Resolution of defects:* CI/CD allows way to establish the appropriate quality gates in the development and test. A fast feedback loop to the developers ensures that are addressed early in the development cycle.
2) *Reduced assumptions:* CI/CD replaces testing assumptions with knowledge. So, it eliminates all cross-platform errors at the development stage.
3) *Better quality assurance:* CI/CD enables QA teams to release deployable software without it the projects are prone to delayed releases because of unforeseen issues which arise at any point in the traditional development and test process.
4) *Software health measurability:* By establishing continuous testing into automated integration process, software health attributes such as complex it can be tracked overtime.
5) *Reduced overhead cost:* Finding a bug at the development stage is the cheapest possible way. If the same bug was to be fixed in any other environment, it would cost more.CI/CD requires some upfront overhead cost.
6) *Faster time to market:* Faster test and QA cycles enable organizations to get quality products and services to market faster and more efficiently.
7) *Varied Application Development Stacks:* Need to build a platform that creates a CD pipeline foreach application. No single vendor provides a comprehensively suite of tools.

III. APPLICATION DEPLOYMENT AND TESTING STRATEGIES

A. Deployment Strategies

When you deploy a service, it's not always exposed immediately to users. Sometimes, it's only after the service is released that users see changes in the application. However, when a service is released in-place, deployment and release occur simultaneously. In this case, when you deploy the new version, it starts accepting production traffic. Alternatively, there are deployment strategies for provisioning multiple service versions in parallel which are given below as follows.

1) *Recreate Deployment Pattern:* With a recreate deployment, you fully scale down the existing application version before you scale up the new application version. The advantage of the recreate approach is its simplicity. You don't have to manage more than one application version in parallel, and therefore you avoid backward compatibility challenges for your data and applications. The recreate method involves downtime during the update process. Downtime is not an issue for applications that can handle maintenance windows or outages. However, if you have mission-critical applications with high service level agreements (SLAs) and availability requirements, you might choose a different deployment strategy. A deployment defined with a strategy of type Recreate will terminate all the running instances then recreate them with the newer version.
2) *Rolling Update Deployment Pattern:* In a rolling update deployment, you update a subset of running application instances instead of simultaneously updating every application instance. In this deployment approach, the number of instances that you update simultaneously is called the window size. In the preceding diagram, the rolling update has a window size of 1. One application instance is updated at a time. If you have a large cluster, you might increase the window size. A ramped deployment updates pods in a rolling update fashion, a secondary Replica Set is created with the new version of the application, then the number of replicas of the old version is decreased and the new version is increased until the correct number of replicas is reached.
3) *Blue/Green Deployment Pattern:* In a blue/green deployment (also known as a red/black deployment), we perform two identical deployments of your application. A blue/green deployment differs from a ramped deployment because the "green" version of the application is deployed alongside the "blue" version. After testing that the new version meets the requirements, we update the Kubernetes Service object that plays the role of load balancer to send traffic to the new version by replacing the version label in the selector field.

B. Testing Strategies

The testing patterns discussed in this section are typically used to validate a service's reliability and stability over a reasonable period under a realistic level of concurrency and load.

1) *Canary Test Pattern:* In canary testing, you partially roll out a change and then evaluate its performance against a baseline deployment, as the following diagram shows. In this test pattern, you deploy a new version of your application alongside the production version. You then split and route a percentage of traffic from the production version to the canary version and

evaluate the canary's performance. You select the key metrics for the evaluation when you configure the canary. We recommend that you compare the canary against an equivalent baseline and not the live production environment.

2) *A/B Test Pattern:* A/B testing is really a technique for making business decisions based on statistics, rather than a deployment strategy. However, it is related and can be implemented using a canary deployment so we will briefly discuss it here. With A/B testing, you test a hypothesis by using variant implementations. A/B testing is used to make business decisions (not only predictions) based on the results derived from data.
3) *Shadow Test Pattern:* Sequential experiment techniques like canary testing can potentially expose customers to an inferior application version during the early stages of the test. This risk can be managed by using offline techniques like simulation. However, offline techniques do not validate the application's improvements because there is no user interaction with the new versions [9, 10].

IV. ZERO-DOWNTIME DEPLOYMENT

Zero-downtime deployment [11] is when a new version of the application can be introduced into the production environment without making the user aware of any downtime. This allows for deployments of new features and bug fixes to take place more often without the users noticing. Common techniques for implementing zero-downtime deployment are Blue-Green Deployment and Canary Releasing. Zero-downtime deployment techniques are commonly used when implementing rapid deployment approaches like continuous deployment.

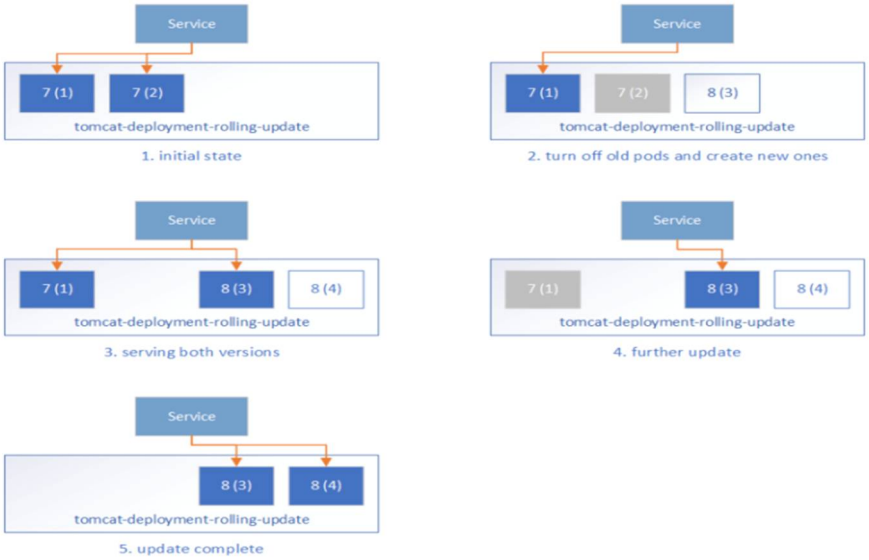


Fig. 2. Zero-Downtime Deployment in Kubernetes

Zero-downtime deployments ensure updates can be made available to users, without degrading their experience while the update happens. This in turn enables smaller, more frequent updates. The cost of implementing zero-downtime deployments, however, is quite high due to the need for a load balancer and multiple servers.

V. DEVELOPMENT AND OPERATIONS (DevOps)

Due to [12] increasing competition in software industry, organizations play a major in assigning required resources to develop and deliver trustworthy and high-quality products to consumers. Consumers expect to have continuous interaction with DevOps team so that they can provide their continuous feedback. DevOps is blending of two terms development and operations which aims to provide conjoin approach to industry's software development and operation team job in software development lifecycle. It provides a good communication between these two teams. DevOps describes the conformation of automation and programmable software development and infrastructure deployment and maintenance. Continuous integration, continuous deployment and continuous delivery are the important factors in software industry that helps organizations to constantly release new attributes and products that

are trustworthy. Continuous integration focuses on integrating each developers work multiple times per day so that debugging of error is easy. Continuous delivery focuses on demoting discordance in deployment or release process and automating the build step so that code can be released securely at any time. CI/CD pipeline provides following benefits in software delivery lifecycle: obtaining rapid feedback from customers, rapid and steady release leads to have customer satisfaction and quality assured product, CD helps to automate tasks which was carried out manually.

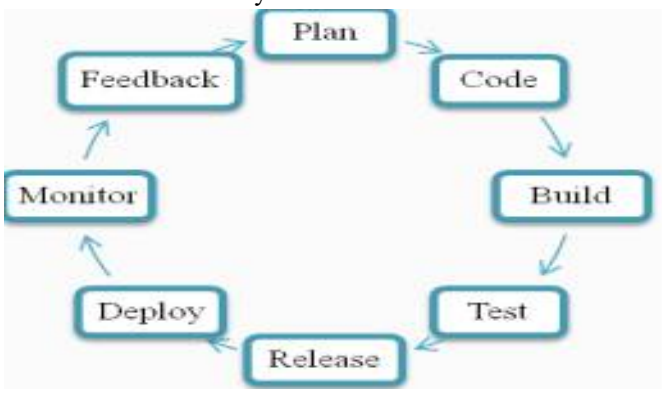


Fig. 3. DevOps lifecycle

A. DevOps Tools

The various tools of DevOps:

1) *Terraform:* Terraform helps to build, change and version infrastructure safely and efficiently. It is designed to support and manage the lifecycle of a vast resources, physical servers and SaaS products. It focuses on the automation of the infrastructure itself. It is also used to avoid code duplication.
2) *Git:* Git is a version control tool used to push code into remote repository i.e., Github.com during software development lifecycle. It is also used to monitor changes in file sets. Developers push their code to repository created in Github.com using git commands. Initially install git in the server using sudo apt-get install git command.
3) *Maven:* Maven is project management and comprehension tool which provides complete build lifecycle framework for developers. Maven is based on Project Object Tool (POM) file. POM is used for project builds, dependency and documentation. POM is an XML file that is present in the base directory of project as pom.xml. POM file contains all the necessary information and configuration details of the project.
4) *Jenkins:* Continuous integration (CI) process is carried out using Jenkins tool. Jenkins is an open source automation server helps to automate manual work of software development lifecycle
5) *Docker:* Docker is a containerization platform that is used to create a package containing an application and all its dependencies altogether in the form of a docker container to make sure that the application works perfectly in all environments. Docker Container is a standardized unit which is created on the fly to deploy a specific application or environment.
6) *Kubernetes:* Kubernetes is a platform for deploying and managing containers. It is production-grade, open-source infrastructure for the deployment, scaling, management, and composition of application containers across clusters of hosts. It is primarily targeted at applications composed of multiple containers. It is therefore a group of containers using pods and labels into tightly coupled and loosely coupled formations for easy management and discovery.

VI. OVERVIEW OF KUBERNETES

Kubernetes is an open-source software for automating management of computerized services, such as containers. Kubernetes relieves application developers from the complexity of implementing their application's resiliency. Therefore, it has become a popular platform for deploying microservice based applications [13, 14].

Kubernetes is a platform for automating the deployment and scaling of containerized applications across a cluster. The Kubernetes cluster has a master-slave architecture. The nodes in a Kubernetes cluster can be either virtual or physical machines. The master node hosts a collection of processes to maintain the desired state of the cluster. The slave nodes that we will refer to them simply as nodes have the necessary processes to run the containers and be managed by the master. The most important process running on every node is called Kubelet. Kubelet runs the containers assigned to its node via Docker, periodically performs health checks on

them, and reports to the master their statuses as well as the status of the node. The smallest and simplest unit that Kubernetes deploys and manages is called a pod. A pod is a collection of one or more containers and provides shared storage and network for its containers. Containers in a pod share its IP address and port space. A pod also has the specifications of how to run its containers. Customized labels can be assigned to pods to group and query them in the cluster. All this information is described in the pod template. In practice, microservices are containerized and deployed on a Kubernetes cluster as pods [15].

Kubernetes has become the leading platform for powering modern cloud-native micro-services in recent years. Its popularity is driven by the many benefits it provides, including [16]:

Cloud-native design: Kubernetes encourages a modular, distributed architecture which increases the agility, availability, and scalability of the application.

Portability: Kubernetes works exactly the same way, using the same images and configuration, no matter which cloud provider or data-center environment is being used.

Open-source: Kubernetes is an open-source platform that developers can use without concerns of lock-in and is the most widely validated in the market today.

Once you select this open-source container orchestration platform, the next step is to select how best to deploy Kubernetes. Shown below is an overview of the role Kubernetes plays in placing, scaling, replicating, and monitoring containers on nodes.

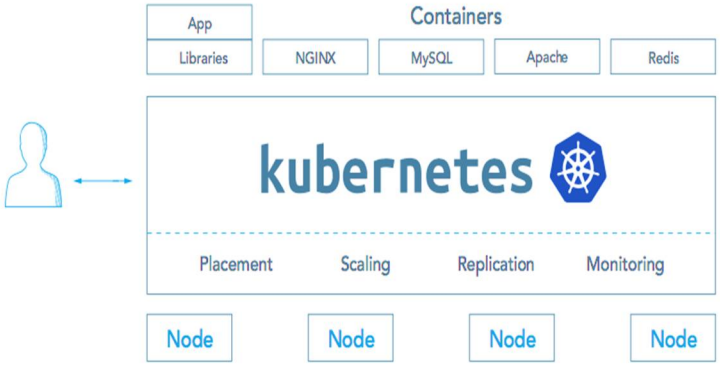


Fig.4. Overview of Kubernetes

A. Kubernetes Deployment Strategies

In Kubernetes there are a few different ways to release an application, it is necessary to choose the right strategy to make your infrastructure reliable during an application update. Choosing the right deployment procedure depends on the needs, we listed below some of the possible strategies to adopt [17]:

1) *Recreate:* terminate the old version and release the new one. Destroy the old version and up the new version.
2) *Ramped***:** release a new version on a rolling update fashion, one after the other. The version **b** is slowly rolled out replacing version **a**.
3) *Blue/green:* release a new version alongside the old version then switch traffic. New version is shipped alongside to old version and the traffic its switch off.
4) *Canary***:** release a new version to a subset of users, then proceed to a full rollout. New version it's deployed to a subset of users and gradually increment for all users.
5) *A/B testing:* release a new version to a subset of users in a precise way (HTTP headers, cookie, weight, etc.). A /B testing is really a technique for making business decisions based on statistics but we will briefly describe the process. This doesn't come out of the box with Kubernetes, it implies extra work to setup a more advanced infrastructure.

VII. COMPARISON

The canary release is a limited deployment to a subset of production nodes with sticky sessions activated. That way, if you release a bad issue, you can control and reduce the number of users/customers that are impacted. They have two mirrored production environments ("blue" and "green"), and you push changes to all the node of either blue or green at the same time, then utilize

networking magic to control which version users are routed to through DNS. The distinction is whether or not the new 'green' version is put to real-world testing. Regular Blue/Green with smart routing of particular users to the latest version is a popular approach to deploy Canary. The Recreate strategy supports lifecycle hooks for inserting code into the deployment process and provides basic rollout behavior. There are more release techniques than rolling and blue-green deployments. Another option is to use canary releases. In a canary release, only a subset of all production environments receives a software upgrade at first. However, rather than continuing to roll out to the entire system, this partial release will be kept in place for testing purposes. A load balancer or a feature flag then directs a subset of users to the updated software. When you wish to collect statistics and comments from a specific group of users regarding updated software, canary releasing makes sense. Canary releases work well with broader rolling deployments because you may gradually push out the updated software to larger and larger portions of your user base until all production servers were updated.

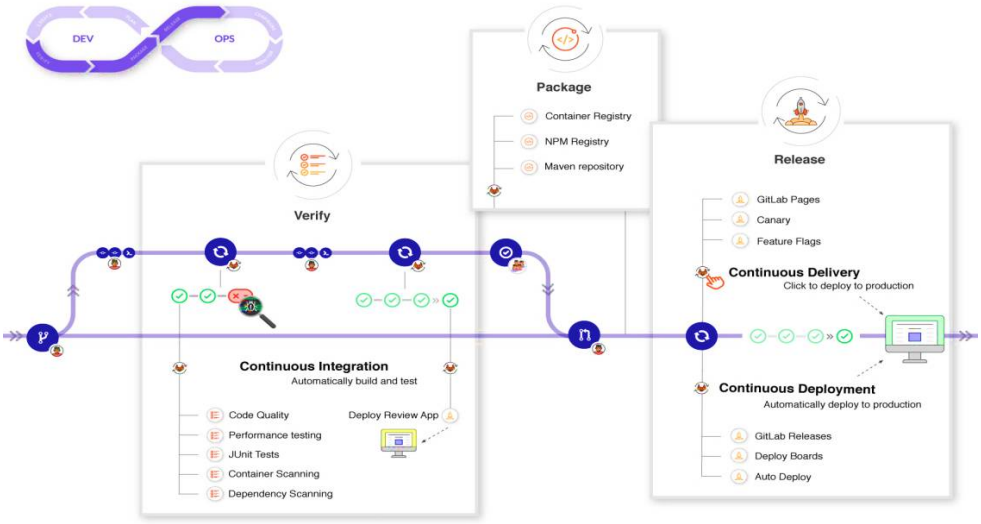


Figure 5: Deeper look into the CI/CD workflow

For new application releases, rolling, blue/green, and canary deployments are all common choices. Some use cases are more adapted to each approach than others.

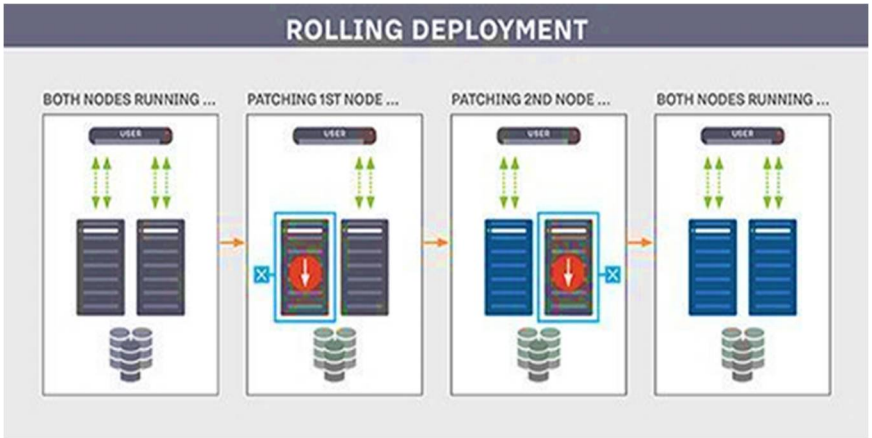


Figure 6: Rolling Strategy

As long as there are no issues with the new version, it is deployed across the hosting environment until all of the servers are running the updated application. However, if problems arise, the operations team may divert all traffic to the servers that still host the known-good version until the new release's fault is resolved.

The blue/green deployment strategy requires teams to manage two separate application hosting infrastructures, one blue and the other green. One of these infrastructure configurations is serving the production version of the programmed at any given time, while the other is kept in reserve.

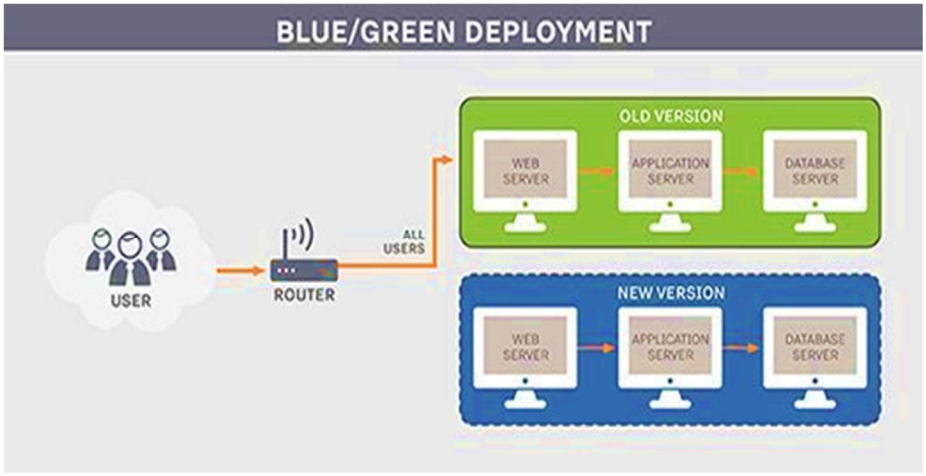


Figure 7: Blue/Green Strategy

The IT team makes the new version available to certain users before others in a canary deployment pattern, which is comparable to a rolling deployment. However, rather than target certain servers, the canary method targets specific users to gain access to the latest programmed version.

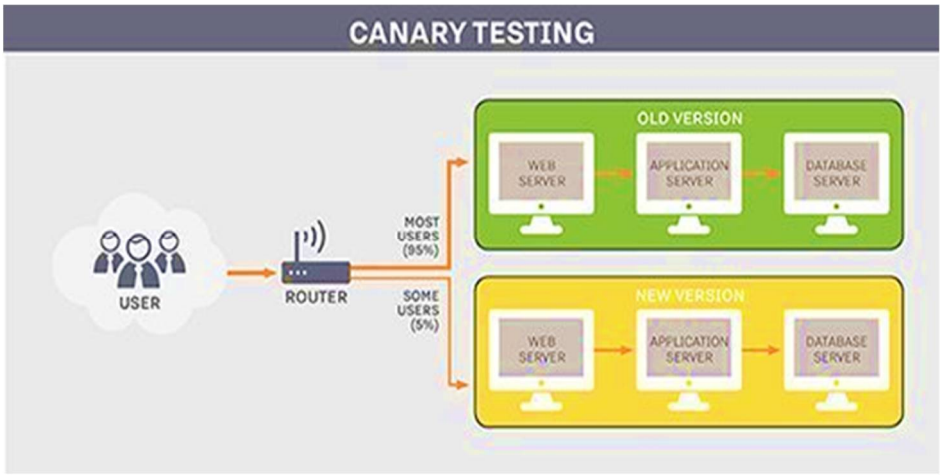


Figure 8: Canary Strategy

Canary deployments are superior to blue/green and even rolling deployments for IT companies because they don't require any extra hosting infrastructure. As a result, it's a useful approach for startups and other organizations with limited budgets.

VIII. LITERATURE REVIEW

M. De Jong and A. Van Duersen (2015) In this paper, they studied the problem of continuous deployment in the presence of database schema evolution in more detail. They identify a number of shortcomings to existing solutions and tools, mostly related to avoidable downtime and support for foreign keys. They propose a novel approach to address these problems, and provide an open-source implementation. Initial evaluation suggests the approach is effective and sufficiently efficient [18].

O. Günalp (2015) In this paper they present Rondo, a tool suite that enables continuous deployment for dynamic, service-oriented applications. At the center of these tools, they propose a deterministic and idem potent deployment process. They provide with Rondo a deployment manager that implements this process and capable of conducting deployments and continuously adapting applications according to the changes in the current target platform. The tool suite also includes a domain-specific language for describing deployment requests. They validate our approach in multiple projects, for provisioning the platform as well as for installing applications and continuous reconfigurations [19].

M. Soni et al. (2015) In past few years, the transition to cloud platforms has given benefits such as agility, scalability, and lower capital costs but the application lifecycle management practices are slow with this disruptive change. The objective of the paper is to

create a proof of concept for designing an effective framework for continuous integration, continuous testing, and continuous delivery to automate the source code compilation, code analysis, test execution, packaging, infrastructure provisioning, deployment, and notifications using build pipeline concept [20].

K. Gallaba et al. (2019) in this thesis, they set out to study and improve the robustness and efficiency of CI/CD which included (1) conceptual contributions in the form of empirical studies of large samples of adopters of CI/CD tools to discover best practices and common limitations, as well as (2) technical contributions in the form of tools that support stakeholders to avoid common limitations (e.g., data misinterpretation issues, CI configuration mistakes) [21].

Q. Liao et al. (2020) this study attempts to design a model that can not only capture the abstraction of the pipeline but also provides a testing environment for the impersonal influencers of CI/CD performance. The current study, therefore, aims to contribute (1) a pipeline model based on the logic of the queueing system and enabled by agent-based simulation, and (2) an experimental environment which allows the testing of different settings and operation scenarios [22].

A. Cepuc (2020) This paper presents an entire automated pipeline, starting with detecting changes in the Java-based web application source code, creating new resources in the Kubernetes cluster to host this new version and finally deploying the containerized application in AWS. The solution follows DevOps best practices and relies on Jenkins for the Continuous Integration stage. The novelty herein is that they used Ansible for Continuous Deployment thus increasing the scalability and overall ease of use. The solution ensures zero downtime and proves fast, even though it combines six different technologies and requires very few computational resources [23].

J. Mahboob and J. Coffman (2021) This paper describes a complete CI/CD pipeline running on Kubernetes that addresses four gaps in existing implementations. First, the pipeline supports strong separation-of-duties, partitioning development, security, and operations (i.e., DevSecOps) roles. Second, automation reduces the need for a human interface. Third, resources are scoped to a Kubernetes cluster for portability across environments (e.g., public cloud providers). Fourth, deployment artifacts are secured with Asylo, a development framework for trusted execution environments (TEEs) [24].

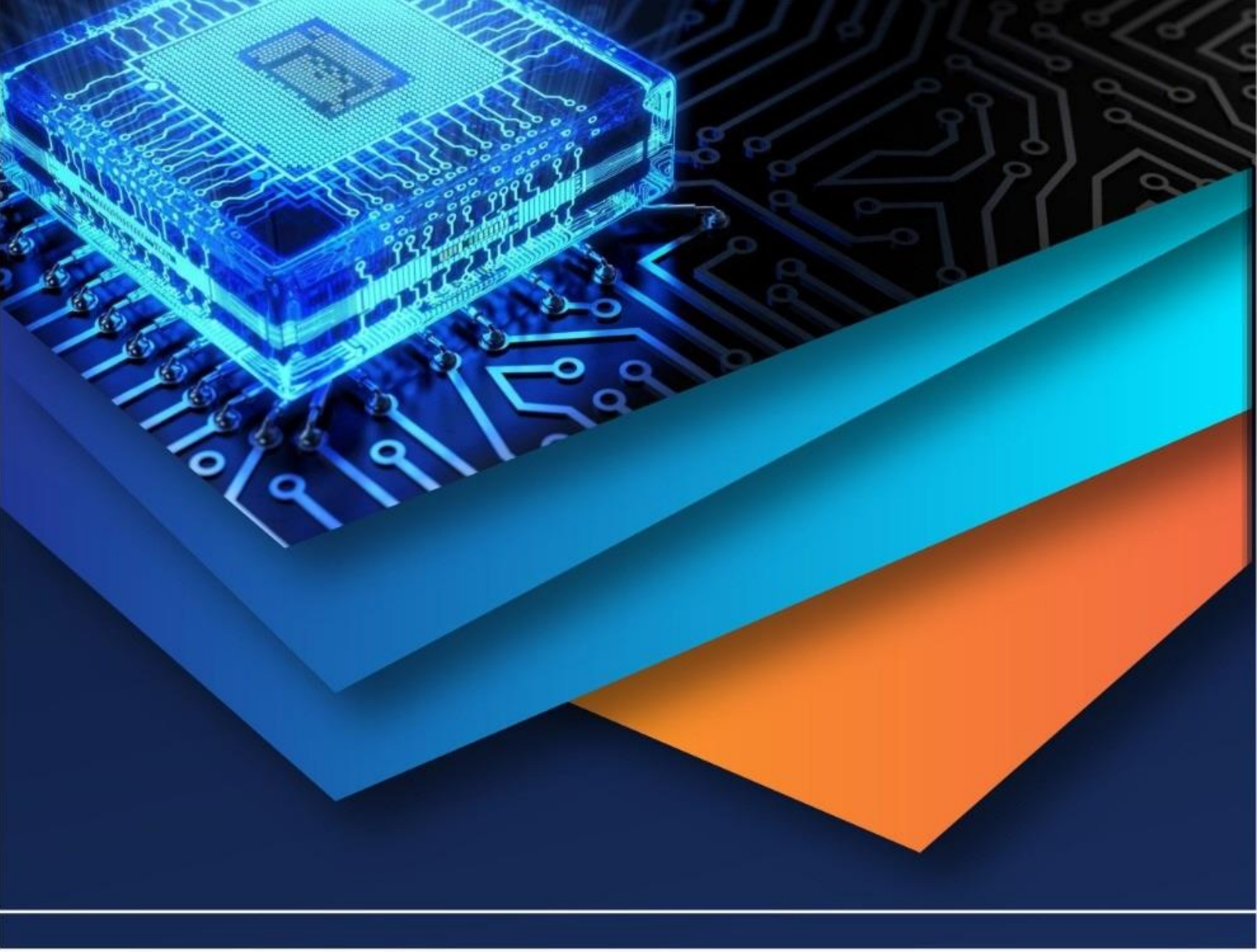
IX. CONCLUSION

CI/CD is closely related to DevOps as a concept. CI/CD aims at rapid delivery of software changes to end-users via automated build, test and deployment. On the other hand, DevOps has emerged as a methodology that is involved in the entire product lifecycle through marketing, planning, human resources, and sales along with development and operations. Due to increasing competition in software industry, organizations play a major in assigning required resources to develop and deliver trustworthy and high-quality products to consumers. Consumers expect to have continuous interaction with DevOps team so that they can provide their continuous feedback. Continuous practices have become an important area of software engineering research and practice. Whilst the reported approaches, tools, and practices are addressing a wide range of challenges, there are several challenges and gaps which require future research work for: improving the capturing and reporting of contextual information in the studies reporting different aspects of continuous practices; gaining a deep understanding of how software-intensive systems should be (re-) architected to support continuous practices; addressing the lack of knowledge and tools for engineering processes of designing and running secure deployment pipelines.

REFERENCES

[1] Rahman, A. A. U., Helms, E., Williams, L., & Parnin, C. (2015). *Synthesizing Continuous Deployment Practices Used in Software Development. 2015 Agile Conference.* doi:10.1109/agile.2015.12
[2] J. Humble, and D. Farley, "Continuous Delivery," 1st Ed. Addison Wesley, 2011
[3] M. Fowler (2013, May 30), Continuous Delivery [Online]. Available: http://martinfowler.com/bliki/ContinuousDelivery.html
[4] P. Swartout, "Continuous Delivery and DevOps: A QuickStart Guide," 1st Ed. Packet Publishing, 2012
[5] Shahin, M., Babar, M. A., Zahedi, M., & Zhu, L. (2017). *Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges. 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* doi:10.1109/esem.2017.18
[6] Asfin Achdian and M Akbar Marwan, "Analysis of CI/CD Application Based on Cloud Computing Services on Fintech Company," International Research Journal of Advanced Engineering and Science, Volume 4, Issue 3, pp. 112-114, 2019.
[7] S.A.I.B.S. Arachchi, Indika Perera," Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management", 978-1-5386-4417-1/18/$31.00 ©2018 IEEE
[8] NIYA N J, JASMINE JOSE," A Literature Survey on Development and Operation", 2020 IJCRT | Volume 8, Issue 4 April 2020 | ISSN: 2320-2882
[9] https://blog.container-solutions.com/kubernetes-deployment-strategies
[10] https://cloud.google.com/architecture/application-deployment-and-testing-strategies
[11] https://avikdas.com/2020/06/30/scalability-concepts-zero-downtime-deployments.html

[12] Sushmitha M S," Setting Up CICD Pipeline for Web Development Project in Cloud", International Journal for Research in Engineering Application & Management (IJREAM) ISSN: 2454-9150 Vol-05, Issue-02, May 2019
[13] "Kubernetes," Kubernetes. [Online]. Available: https://kubernetes.io/. [Accessed: 24-Jan-2018].
[14] Md. Shazibul Islam Shamim," XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices", arXiv:2006.15275v1 [cs.CR] 27 Jun 2020
[15] Leila Abdollahi Vaughan," Kubernetes as an Availability Manager for Microservice Applications" https://arxiv.org/ftp/arxiv/papers/1901/1901.04946.pdf
[16] https://platform9.com/docs/deploy-kubernetes-the-ultimate-guide/
[17] https://blog.container-solutions.com/kubernetes-deployment-strategies
[18] M. De Jong and A. Van Deursen, "Continuous Deployment and Schema Evolution in SQL Databases," 2015 IEEE/ACM 3rd International Workshop on Release Engineering, 2015, pp. 16-19, doi: 10.1109/RELENG.2015.14.
[19] O. Günalp, C. Escoffier and P. Lalanda, "Rondo: A Tool Suite for Continuous Deployment in Dynamic Environments," 2015 IEEE International Conference on Services Computing, 2015, pp. 720-727, doi: 10.1109/SCC.2015.102.
[20] 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," 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), 2015, pp. 85-89, doi: 10.1109/CCEM.2015.29.
[21] K. Gallaba, "Improving the Robustness and Efficiency of Continuous Integration and Deployment," 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 619-623, doi: 10.1109/ICSME.2019.00099.
[22] Q. Liao, "Modelling CI/CD Pipeline Through Agent-Based Simulation," 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2020, pp. 155-156, doi: 10.1109/ISSREW51248.2020.00059.
[23] A. Cepuc, R. Botez, O. Craciun, I. -A. Ivanciu and V. Dobrota, "Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible and Kubernetes," 2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet), 2020, pp. 1-6, doi: 10.1109/RoEduNet51892.2020.9324857.
[24] J. Mahboob and J. Coffman, "A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework," 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), 2021, pp. 0529-0535, doi: 10.1109/CCWC51732.2021.9376148.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY