International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 7, Issue 5, ISSN No. 2455-2143, Pages 99-103
Published Online September 2022 in IJEAST (http://www.ijeast.com)

AUTOMATING DEPLOYMENTS OF THE LATEST APPLICATION VERSION USING CI-CD WORKFLOW

Spoorthi Jayaprakash Malgund, Dr Sowmyarani C N
Department of Computer Science and Engineering
RV College of Engineering, Bengaluru, Karnataka, India

Abstract— **The actions a developer should take to deploy a new version of a software product are essentially specified by a CI CD pipeline. The developer would still have to perform the same tasks manually, which is much less efficient if the pipeline is not automated. Thus, the steps listed below are experienced by most software releases. We give an overview of how a secure SDLC process, along with continuous integration and continuous deployment, is used to automate the deployment of a new version of an application. This allows software development teams to concentrate on meeting business requirements while ensuring code quality and software security.**

Keywords— **CI/CD, Jenkins, Spinnaker, AWS, Kubernetes, GitHub.**

I. INTRODUCTION

All businesses nowadays are moving toward automating a number of processes in an effort to save time and increase productivity. By adding automation to the various stages of app development, CI/CD is a technique for regularly delivering apps to clients. Continuous integration, continuous delivery, and continuous deployment are the three core CI/CD concepts. CI/CD is a solution to the issues that development and operations teams may encounter while integrating new code. In particular, CI/CD brings continuous monitoring and continual automation throughout the whole lifespan of an app, from the integration and testing stages to the delivery and deployment stages. The development and operations teams supporting this "CI/CD pipeline" collaborate in an agile manner using either a DevOps or Continuous Delivery approach.

II. BACKGROUND

The objective of contemporary application development is to have numerous developers working on various aspects of the same app concurrently. The effort that results, however, can be laborious, manual, and time-consuming if a company is set up to combine all branching source code in one day [5]. This is so that modifications made to an application by a developer working alone won't necessarily conflict with other changes being made to the same application at the same time. Instead of the team deciding on a single cloud-based IDE, this issue may be made worse if each developer has customized their own local integrated development environment (IDE).

Developers can more frequently—sometimes even daily—merge their code changes back into a shared branch, or "trunk," with the aid of continuous integration (CI). When a developer merges changes to an application, the changes are checked to see whether they haven't broken the app by automatically creating the application and executing various levels of automated testing, often unit and integration tests. This entails testing each component of the software, including its many modules as well as its classes and functions. If automated testing identifies a conflict between updated and legacy code, continuous integration (CI) facilitates quick and frequent issue fixes.

Continuous delivery automates the deployment of that validated code to a repository after the automation of builds and unit and integration testing in CI. Therefore, it's critical that CI be already included in your development pipeline to have a successful continuous delivery process. A codebase that is constantly prepared for deployment to a production environment is the aim of continuous delivery.

Every step in continuous delivery—from merging code changes to delivering builds fit for production—involves test automation and code release automation. The operations team can then swiftly and easily deploy an app to production after that procedure is complete.

The continuous deployment marks the end of a well-developed CI/CD pipeline. Continuous deployment is an extension of continuous delivery, which automates the release of a build that is ready for production to a code repository, and it automates the release of an app to production. Continuous deployment largely relies on carefully thought-out test automation because there is no manual gate at the pipeline level before production. Continuous deployment implies that a developer's contribution to a cloud application might go online within minutes of being written (assuming it passes automated testing). It is now much simpler to regularly gather and consider customer input. When all these related CI/CD techniques are used, the deployment of an application becomes less hazardous, making it simpler to release app modifications incrementally rather than all at once. The

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 7, Issue 5, ISSN No. 2455-2143, Pages 99-103
Published Online September 2022 in IJEAST (http://www.ijeast.com)

significant upfront investment is also required because a number of testing and release phases in the CI/CD pipeline need the creation of automated tests.

III. RECENT STUDY

Leonardo, et al. [1] explain the current survey and the difficulties with DevOps from the viewpoints of engineers, managers, and researchers in Leite. We conduct a study of the literature and create a conceptual map of DevOps, connecting the principles to the DevOps automation technologies. We next go into their real-world ramifications for researchers, managers, and engineers. Finally, we examine some of the most important DevOps difficulties mentioned in the literature critically. We have covered the DevOps principles and issues raised in the literature in this survey. By connecting these ideas to specific technologies, we helped practitioners select the best toolkit.

The methodical presentation of the most important ideas, resources, and consequences from the viewpoints of academics, managers, and engineers—including developers and operators—helps IT, professionals, as well. Based on each profile, we think our reader can now more clearly grasp how DevOps affects daily tasks. The two cornerstones of DevOps are automation and cross-departmental human cooperation.

The major goal of this project is to automate the creation of a project in Maragathavalli [2], which covers the phases of creating the code, testing it, and project deployment, all of which take more time to do manually. To offer a mechanism for tracking, resolving, and automating application issues. 2) To create a system, a continuous pipeline method of development, testing and deployment is used. - Since there will be several modifications to the organization every day, each of these changes requires a unique build.

In terms of increased productivity mentioned in the Sikender [3], DevOps allows teams to take more time to create value. Developers won't have to wait while computers are being set up or code is being installed if firms adopt automated testing and installation. By selecting a button on the self-service portal, you may complete both chores. For example, some European banks have used DevOps in their business systems, which has improved the delivery of online banking upgrades by 25%. Additionally, it lessens the effort for the IT staff, allowing them to concentrate on more challenging tasks that will benefit the firm more.

To improve performance code and hasten recovery, developers also begin work on their next project as soon as new products hit the market. Because of this, they seldom have a strong reason to anticipate or stop potential software difficulties; instead, the operations team is always in command. DevOps, on the other hand, involves engineers across the whole software life cycle, resulting in higher-quality code. A few changes are also necessary so that developers can monitor any potential code problems and fix them. Developers may be more easily held liable for faults that occur in any one product since they create less code overall [4].

IV. TOOLS USED IN CI/ CD WORKFLOW

Jenkins	One of the most popular continuous delivery and continuous integration tools available is an open-source CI automation server. Jenkins, the top open-source automation server, offers thousands of plugins to help with creating and automating any project. Once a project has been tested, Jenkins also offers CD code deployment
SonarQube	A continuous code analysis tool that is open-source that contributes to improved source code quality
Spinnaker	A multi-cloud continuous delivery technology called Spinnaker allows for the release and deployment of software updates across many cloud providers.
GitHub	Software projects are stored, tracked, and collaborated on using GitHub, an online platform for software development.
Docker Hub	For the purpose of locating and sharing container images with your team, Docker offers the service of Docker Hub
Helm Repo	Remote servers that house a collection of Kubernetes resource files are known as helm chart repositories.

V. METHODOLOGY

I. CONFIGURE THE ENVIRONMENTS

i. Set up Kubernetes cluster on Teleport

Secure, consistent access to your Kubernetes clusters may be provided through Teleport. Set up SSO for authenticating to the teleport cluster after deploying teleport in a Kubernetes cluster first.

One will install a single Teleport pod running the Auth Service and Proxy Service in your Kubernetes cluster as part of this setup, as well as a load balancer that permits traffic from the outside world to reach your Teleport cluster. The Teleport cluster that is operating inside of your Kubernetes cluster may then be used by users to access it. a website with registration. This is necessary for Teleport clients to validate the Proxy Service host and for Teleport to configure TLS using Let's Encrypt. The load balancer we deploy in this post requires a Kubernetes cluster hosted by a cloud provider.

The deployment of a single Teleport pod utilizing a persistent volume as a backend. The settings of CLUSTER NAME and

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 7, Issue 5, ISSN No. 2455-2143, Pages 99-103
Published Online September 2022 in IJEAST (http://www.ijeast.com)

EMAIL, where CLUSTER NAME is the domain name you are using for your Teleport deployment and EMAIL is an email address used for alerts, should be changed in accordance with your environment. An external load balancer is used by Teleport's Helm chart to generate a public IP address. Create two A DNS records: *.tele.example.com for web apps that use Application Access, and tele.example.com for all other traffic. Make a local user first, then SSO for Kubernetes, then last

ii. Host a Docker Registry

Establishing a Kubernetes environment and installing a TLS-enabled Private Docker Registry as a Pod. This will enable us to upload our specially created images to the registry, from whence any worker node may download them and execute them as containers in Pods.

iii. Host a Helm Repository

Helm Charts are only a collection of Kubernetes YAML manifests that can be distributed across your Kubernetes clusters as a single package. It can be laborious and time-consuming to create and maintain Kubernetes YAML manifests for all the necessary Kubernetes objects. You would require at least three YAML manifests with duplicated and hard coded variables for the most straightforward deployments. This procedure is made simpler by Helm, which generates a single package that can be marketed to your cluster.

Tiller (the helm server) was formerly required to be deployed in your cluster in order for Helm, a client/server application, to function. When you initialise the helm on your client system, this is installed. Tiller merely takes client requests and instals the requested package into your cluster. Helm is similar to Linux's RPM or DEB packages in that it gives developers an easy way to package and distribute a programme for installation to its end customers.

Helm consists of two components: the client (CLI), which is installed on your computer, and the server (Tiller), which is used to carry out operations on the Kubernetes cluster. Tiller will ensure that the state is indeed the case by adding, updating, or removing resources from the chart once the CLI pushes the resources you require. There are three ideas we need to become familiar with in order to completely understand helm:

iv. Create an Isolated Environment- Dev, Stage, Prod

Dev, Stage/ Integration, and Prod are the three environments you should create in the Kubernetes cluster. The environment on your PC is the development environment.

You will make all your code modifications here. It's where all your branches and commits, as well as those of your coworkers, are stored. The configuration of the development environment differs often from that of the user's workspace.

The production environment and the stage environment are as comparable as they may be. This time, rather of having the code on a local machine, it will be located on a server. Without affecting the production environment, it will attempt to connect to as many services as it can.

Here is where the hard-core testing takes place. Both database migrations and configuration modifications will be tested here. The stage environment assists in identifying and resolving any difficulties that may arise while doing significant version changes.

Users can view the final code in the production environment after all upgrades and testing. This habitat is the most crucial of all the others.

VI. CODE DEVELOPMENT

a. **Code Commit-** Version control is another name for a step of code commit. A commit is an action that uploads a developer's most recent modifications to the repository. Every revision of a developer's code is permanently archived. Developers create the code and commit after the software requirements, feature additions, bug fixes, or change requests are finished after discussion and evaluation of the changes with collaborators. Source Code Management is the process of managing the repository where modifications and commit changes are made (SCM tool). Code modifications are incorporated into the base code branch kept at the central repository, such as GitHub once the developer submits a code commit.

b. **Static Code Analysis using SonarQube-** When a developer produces code and publishes it to a repository, the system is immediately prompted to begin the subsequent code analysis process. Imagine a stage where the code is directly built after being committed but fails either during the build or deployment. When it comes to the use of resources, both human and mechanical, this process becomes sluggish and expensive. The code must be examined for static policies. Static application security testing, also known as SAST, is a white box testing technique that looks at the code from the inside using SAST tools like SonarQube etc. to identify weaknesses and flaws in software. The code is quickly examined for any syntactic mistakes throughout this procedure.

VII. CODE DEVELOPMENT

a. Build

Check out the code and load the Jenkins file. A webhook can be designed to automatically trigger the pipeline after a commit. A docker container must be set up as build agents for Jenkins. Then, a Jenkins Pipeline needs to be created with the necessary configurations and stages.

The GitHub directory is linked to the pipeline along with the branch. Jenkins then pulls the latest code changes and builds the pipeline. After the pipeline builds, the latest version packages will be uploaded to the AWS S3 buckets. Merging

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 7, Issue 5, ISSN No. 2455-2143, Pages 99-103
Published Online September 2022 in IJEAST (http://www.ijeast.com)

routine code contributions and continually producing artifacts is one of the objectives of continuous integration. By immediately identifying errors and determining if the newly introduced module works well with the old modules, this technique can help developers. As a result, it contributes to shortening the total time needed to test a new code update. The compilation and creation of executable files or packages is assisted by the build tools. Every day, several builds could be made, making it challenging to keep track of them all. Therefore, the build is transferred to the S3 bucket for storage as soon as it is created and confirmed.

b. Testing

There are several kinds of CI tests to run throughout the automated build as the CI process progresses from the point at which it is initiated (i.e., when code is checked in) to the point at which an artifact is released.

- **Code Quality Test:** Code quality tests may or may not be a component of the CI process, depending on when the formal CI process begins and whether a quality gate for checking in or merging code is present. Code quality tools that concentrate on static code analysis, like SonarQube and GitHub CodeQL, are careful during code modifications. Although code analysis might suggest quality, code by itself does not execute and take into consideration all of the infrastructure and environment variables that power operating software.
- **Unit Test:** Unit tests are the fundamental tests that are carried out when new features are added, developed, or updated. The DockerFile is used in the Jenkins pipeline to run the unit test. The code blocks and methods that changed are the focus of unit testing. These would cover the functionality of the application if creating a brand-new project from scratch. Unit tests are frequently used in-process testing, which creates fake objects and verifies assertions. In the JAVA world, JUnit, and in the NPM world, Mocha. A suite of granular unit tests is how they are intended to operate.
- **Integration Test:** Cross-module testing of the application will be the main emphasis of integration testing in the context of continuous integration. Modern unit and integration testing technologies have a lot of overlap. Since JUnit can chase/follow/invoke the method calls that can be chained together, it can be used for both unit and integration tests. As the newly added features and fresh code modifications are perceived to be interoperable, integration testing continues the confidence continuum.

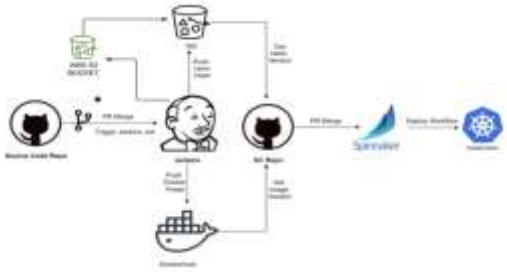

FIG 1: BASIC FLOWCHART OF CI/CD WORKFLOW

VIII. CODE DEPLOYMENT

i. Pushing Helm Chart and Docker Registry

After the build is successfully initiated, an application docker image is built and delivered to Docker Registry. A file called a Docker image is used to run programs within a Docker container. Docker images serve as a template or collection of instructions for creating a Docker container. When utilizing Docker, the starting point is also a set of images. A snapshot and an image are similar concepts in virtual machine (VM) settings.

As "the package management for Kubernetes," Helm is well-known. This implies that if you use the install command for the top-level chart, Helm installs the whole dependency tree of a project. Instead of listing the files to install using kubectl, you only need to run one command to install your complete application. Charts provide us with the same ability to version manifest files as we do with Node.js or any other product. This enables you to install particular chart versions, allowing you to maintain particular code configurations for your infrastructure. Since Kubernetes is natively supported by Helm, getting started with Helm doesn't require writing any elaborate syntax files or other files. The template files may simply be dropped into a new chart to get things started.

ii. Deploy to Kubernetes using Spinnaker

The lifecycle path to deployment is usually as follows:
Development - > Integration - > Staging -> Production

Deployment to Development	• The latest version of the code is found in the development environment. • The goal of the development environment is to allow software developers to batch code changes together and deploy them via CD to the remote Dev environment. • Developers deploy it using spinnaker to see if their code changes are working as expected in a production-like environment.
Deployment to Stage	• Engineers push the code to staging using their deployment pipeline now that they have the build artifact for

	it. • Before our code problems become our customers, this is the last chance to identify bugs, performance regressions, and security risks. • Even if tests have been finished, it's usually a good idea to manually check features out in staging. • The production environment and the staging environment should be as similar as feasible.
Deployment to Prod	• The live application is the production environment. The latest software or items that have been made available for usage by the intended consumers is referred to as the production environment. • Any and all issues need to be resolved before something enters the production environment, and the product or upgrade must function flawlessly. • While new products and upgrades are initially released in the production environment, all testing is done in the development and staging environments. Any flaws that are present in the production environment will be visible to the user.

IX. CONCLUSION

To enable quicker build, delivery, and deployment cycles, you should give CI/CD implementation top priority in your product development setup. It assists in automating the processes required for the end-users to have access to the program and its various versions. The advantages of CI-CD over manual deployments for deploying app versions include an increased focus on core tasks, the growth of an agile mindset, better code quality, faster time to market, and other benefits.

X. REFERENCES

[1] Leite, Leonardo, et al (2020) - A Survey of DevOps Concepts and Challenges, ACM Computing Surveys, vol. 52, no. 6, pp. (1–35)

[2] Maragathavalli, P.(2020) - Automation Pipeline and Build Infrastructure Using DevOps, International Journal for Research in Applied Science and Engineering Technology, vol. 8, no. 11, (pp. 882–886).

[3] Mohammad, Sikender Mohsienuddin. (2018). Improve Software Quality through practicing DevOps Automation. SSRN Electronic Journal. 6, (pp. 251-256).

[4] Erich, F. M., et al. (2017)- A Qualitative Study of DevOps Usage in Practice, Journal of Software: Evolution and Process, vol. 29, no. 6 (pp 1-20)

[5] Weber, I.; Nepal, S.; Zhu, L., (2016) Developing Dependable and Secure Cloud Applications. IEEE Internet Comput. 20, (pp 74–79). [CrossRef]

[6] Len Bass, Ralph Holz, Paul Rimba, An Binh Tran, and Liming Zhu. (2015). Securing a Deployment Pipeline. In Proceedings of the 2015 IEEE/ACM 3rd International Workshop on Release Engineering (RELENG '15). IEEE Computer Society, USA, (pp 4–7).

[7] S. Garg, P. Pundir, G. Rathee, P. K. Gupta, S. Garg and S. Ahlawat, (2021) "On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps," 2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2021, (pp. 25-28)

[8] A. Cepuc, R. Botez, O. Craciun, I. -A. Ivanciu and V. Dobrota,(2020) - Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible and Kubernetes, 19th RoEduNet Conference: Networking in Education and Research (RoEduNet), 2020, (pp. 1-6)

[9] L. Jenkins, "Title: Challenges in deployment of wireless sensor networks," (2014), 9th International Conference on Industrial and Information Systems (ICIIS), 2014, (pp. 1-1).

[10] H. Rajavaram, V. Rajula and B. Thangaraju, (2019) "Automation of Microservices Application Deployment Made Easy By Rundeck and Kubernetes," IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2019, (pp. 1-3)