2023 8th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA) | 978-1-6654-5533-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCCBDA56900.2023.10154891

Integrated log-aware CI/CD pipeline with custom bot for monitoring

A Narendiran
Dept of CSE
PES University
Bengaluru, India
narendiran.work@gmail.com

Abhishek D
Dept of CSE
PES University
Bengaluru, India
abhishekdinesh21042001@gmail.com

Adithya P
Dept of CSE
PES University
Bengaluru, India
adithya.p2001@gmail.com

Debaditya Ray
Dept of CSE
PES University
Bengaluru, India
debadityaray00@gmail.com

Prafullata K. Auradkar
Computer Science
PES University
Bengaluru, India
Prafullatak@pes.edu

H. L. Phalachandra
Computer Science
PES University
Bengaluru, India
hlphala@gmail.com

***Abstract*—Agile is currently the most prevalent life cycle used in software development due to its ability to scale and support changes. This has led to DevOps, which further led to a need for an optimized and integrated approach to implementing CI/CD. This research paper showcases proof of concepts for minimizing human intervention in the operations cycle for software development. It discusses several additions and/or modifications to the traditional implementations of CI/CD pipelines and explores how several DevOps tools can be leveraged to improve them from continuous development and integration to continuous deployment. Our aim is to come up with an improved CI/CD and monitoring process which is convenient for the developers to set up and leverage with the least amount of configuration and manual intervention. Additionally, it proposes a customizable python application to process and maintain the inherently complex logs generated by the pipeline, contributing to the ease of monitoring the pipeline. This paper ignored the application-centric continuous testing phase of the pipeline which is left to the developers to construct.**

***Index Terms*—CI/CD Pipeline, DevOps, Agile, Cloud Computing, Load Balancer, Log Processing, Bot Services**

I. Introduction

While waterfall is a linear and sequential development process formerly used by software teams, Agile is more adaptable because it is built on continual iterative sprints, which are more favourable to change.[1] Organizations realised they needed to handle cross-team communication and cut software needs down into smaller parts in order to implement Agile and DevOps. Agile has led to approaches towards code integration, build, testing and deployment which can seamlessly and agilely support the evolution of a product. This has led to DevOps which supports continuous integration and continuous delivery (CI/CD) to be used in most agile environments.

Agile started evolving as a way for organizations to deliver value to customers faster. With time, the size of the projects and frequency of changes kept on increasing gradually which made way for DevOps. Over the years, many organizations around the world have implemented several custom end-to-end CI/CD Pipelines to meet their various needs pertaining to delivering value to their clients faster. Most such implementations are conceptually straightforward. [2, 3, 4, 5]. However, After studying the different approaches to implementing CI/CD, it was found that a few stages in the pipeline require manual intervention. In this proof of concept, the principles of DevOps have been adopted to propose a complete CI/CD pipeline that can support end-to-end automation and monitoring with minimal manual interventions.

A CI/CD pipeline enables the automation of the entire software delivery process. Just a single push or commit to the source code repository is enough to trigger the entire workflow, resulting in automatically deploying the updated application. This concept can be leveraged to further decrease the manual intervention required in the operations cycle of the software engineering process.

Our major contribution is an integrated CI/CD pipeline as a POC using a combination of tools and services. Any manual intervention in the operations cycle is prone to errors. Eg,, manual configuration of DNS records for each subdomain. Our work proposes an automated approach to create and configure the records. Additionally, it proposes an approach for better communication between the components in the different phases of the CI/CD pipeline and the developer in terms of error handling and reporting, etc. using bot services. Thus, the work covers the automated deployment of the application and its delivery to the end users. It is assumed that application development and testing are done solely by the developers either manually or in an automated fashion depending upon the application. The work focuses only on maximizing throughput and minimizing manual

interventions in the operations part. For the purpose of this POC, delivering and deploying a simple front-end application sufficed. However, this approach can be extended to the back-end or any other component that the user wants to add to the application.

The following introductions are proposed to the CI/CD pipeline:

1) **In the continuous delivery phase**: Automating the creation and configuration of DNS records for domains and subdomains of the application to be deployed.
2) **In the continuous monitoring phase**: A customizable software (hereby referred to as "bot" or "bot service") that receives, processes, labels, groups and persists the logs from the running CI/CD workflow. If possible, it handles errors in the pipeline using naive techniques.
3) **In the continuous deployment phase**: Implementing a mechanism for zero-downtime switching between older and newer versions of the deployed application using load-balancing.

Section 2 discusses background theory to set up context for our work. Section 3 discusses the related work and literature survey exploring several examples of previously implemented workflows by organizations. Section 4 discusses the experimental setup, high-level and low-level architecture, and design along with the configurations and several tools used for the implementation of the CI/CD pipeline. Section 5 showcases our implementation of the CI/CD workflow. Section 6 discusses the implementation of the POC custom software crafted for pipeline monitoring and error notifications (i.e Bot Service) followed by Section 7, which discusses the results and future work.

II. Background

DevOps is a software engineering methodology combining software development and IT operations and blurs the lines between the long clearly marked phases of development, testing, and deployment in contrast to the initial development life cycles adopted in the industry. As soon as any small module or component is translated into code, it is tested for various parameters and after some successful runs, it is instantly released and deployed into production. A CI/CD pipeline enables the automation of the entire software delivery process. Just a single push or commit to the source code repository is enough to trigger the entire workflow and result in the updated application being deployed automatically. The various stages of the CI/CD pipeline are :

- **Continuous Integration & Build** : The first phase of the pipeline where all the code changes and commits from various developers are merged from single or multiple repositories and along with the necessary dependencies, the source code is compiled into an executable followed by basic unit or sanity testing
- **Continuous Testing** : Subsequent phase where the features or the products to be extensively tested are deployed in the necessary environments imitating real working conditions. All the test scripts are generally automated with various types of scenarios and all types of tests taking place from component, and load testing to user acceptance testing.
- **Continuous Deployment** : On successfully passing all the tests, the required infrastructure is allocated and created or modified and subsequently provisioned with the necessary software and tools for installing and executing the tested builds. This is in contrast to continuous delivery where the code is manually deployed into the necessary environment after reviewing it. Care is taken to avoid downtime when switching between different application versions

III. Related work

There have been a large number of applications adopted in the industry. One such industrial experience has been explained in [2], in which a Finnish software development company documents the terms, challenges, methods, and benefits associated with shifting from an in house delivery system to a cloud-native continuous delivery pipeline consisting of open source tools such as Docker, GoCD, Salt, AWS and Jenkins. Another such case study is observed in [6], where an Austrian online business company elaborates on the use of automated testing in the CI/CD pipeline with extensive focus on the various stages such as commit, acceptance, user acceptance, and capacity testing stages. Moving to the usage of tools in [7], an in depth analysis of different tools used in CI/CD is performed where scenarios to use Gitlab over Jenkins stands out. This motivates the use of GitHub Actions for triggering and running the workflow explained in the current paper which is very similar to Gitlab CI/CD. Similarly [8] explains CI/CD using dockerized builds along with the necessary architecture/infrastructure required in order to ensure dockerized builds and best practices for container security. Taking inputs from the paper, with respect to building, dockerized containers are proposed to ensure cross-platform cloud deployability

A recent literature survey identifies the need to process logs for better detection of failures in the continuous engineering process and talks about how similar logs can be clustered together using a variation of the LogCluster tool by Lin et al. [9] for better error handling [10]. In line with this idea of log processing, this paper proposes an alternative approach for the same, using a custom server application that supplements the pipeline.

Log management is an important aspect of any development or deployment process. To track the status of operations in the various phases, extensive logging is configured and performed to give a verbose description of the execution and errors or

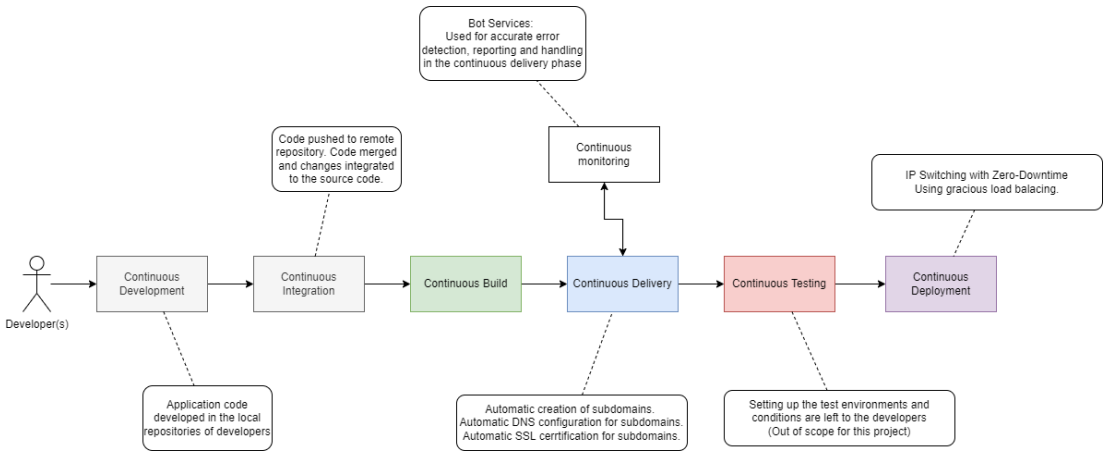

Fig. 1. High Level Workflow Diagram of the proposed CI/CD Pipeline

TABLE I
COMPARISON OF TOOLS CHOSEN

Tool used	***Phase***	***Purpose***	***Popular counterparts***	***Reason of choice***
Github, Git, DockerHub	Continuous Development	Code integration & version control	Aws Codecommit, BitBucket	The popularity and community support provided by GitHub
Docker, Github Actions	Continuous Build	Containerise build and trigger workflow	Jenkins	GitHub Actions offers a hybrid cloud approach by hosting its own runners and is easier to set up.
Terraform, Cloudflare API	Continuous Delivery	Infrastructure configuration and automated creation	Ansible, chef, puppet	Terraform uses a declarative approach in its code, i.e., if an infrastructure resource is already existing in our cloud due to a previous workflow run, the resources will not be re-initialized.
CloudFlare (DNS Server)	Continuous delivery	Name server that provides free SSL encryption for domains mapped	Amazon Route 53, GoDaddy Premium DNS.	Cloudflare is free of cost.
NameCheap (Domain Name registrar)	Continuous delivery	Register the domain for the application	GoDaddy.com	Some domains in NameCheap are free of cost which is best for our purpose of making a proof of concept.
AWS EC2, Load Balancer	Continuous Deployment	Deploying applications in instances and employing load balancing	Heroku, GCP (Google Cloud Platform)	AWS has significantly more services, and more features within those services, than any other cloud provider. It is also the most widely used.
GitHub Webhooks	Continuous Monitoring	Delivery payload to bot for logs	Hw-kafka-avro, Servant-to-elm, etc	Flexible and well-documented API.

failures which pop up. However, locating them sometimes becomes an ordeal with millions of log entries. To combat this issue this section discusses the implementation of a fully automated CI/CD pipeline with a custom python bot for parsing and locating errors in the logs created in the pipeline.

IV. Pipeline Architecture and Configuration

- A central master source code repository for the application is maintained and hosted in the cloud. All the necessary changes are committed here. A workflow is associated and connected with the repository such that on each push or commit, it gets triggered and the new code is built, tested, and deployed.
- The application is dockerized after the build and undergoes necessary testing as required after which it is pushed to a container image repository. For any deployment purposes, the application's executable image is obtained from here.
- For deployment, an infrastructure automation tool is used as part of the workflow that sets up the necessary environment in the cloud. The application requirements are predetermined as to the number of resources or bandwidth it will require.
- The virtual machines if deployed are assigned a static IP address whose DNS-IP records mappings are automatically updated in the DNS nameservers. The domain to be used for the application should be linked with ssl certificates for ensuring security.
- During deployment, a load-balanced approach is employed to smoothly replace the revised application version

with the existing version in servicing.

V. Pipeline Implementation

1) Experimental Setup: Users can develop, test, and deploy user code directly from GitHub using the CI/CD pipeline and GitHub Actions, which automate all software workflows. An Infrastructure as Code (IaC) service like Terraform assists users in setting up their infrastructure on cloud platforms like AWS. In this study, terraform is used to create a docker image and upload it to an online repository like dockerhub. Next, an EC2 instance must be set up to serve as a platform for the docker container to run on. Once the application is deployed, a DNS record is created. The application was a simple react front-end webpage. Cloudflare was utilized as our nameservers. Fig.1 shows the workflow run of the pipeline.

2) Design and implementation:

- A yaml file contains the CI/CD pipeline automation workflow, which is executed when a push event is initiated to the code repository. This file contains a series of jobs to be run.
- Build : This is the initial step in the pipeline, wherein for any change made to the application, a new docker image of the same is built automatically and pushed to a container repository like DockerHub.
- Infrastructure: An EC2 instance is created as specified in the declarative Terraform code. The security group and network configuration are also specified as code. An Elastic IP address is automatically allocated to the created EC2 instance.
- Deployment: The EC2 instance pulls and runs the updated Docker container. A DNS record is created in Cloudflare using the Elastic IP address allocated. Cloudflare provides the SSL certificate required to make requests over HTTPS.
- Load Balancer: For subsequent updates of the application blue-green deployment strategy is used. A new EC2 instance is created and it runs the updated docker container, with the help of the load balancer on AWS, the traffic is directed from one instance to the other and the old instance is destroyed.

After all these steps the application can be accessed on the web browser using the domain name provided earlier.

VI. Bot Services

1) Experimental Setup: GitHub webhooks is a feature that provides a way for notifications to be delivered to an external web server as a payload whenever certain events occur on a remote repository/organization. We leverage this feature by subscribing to the workflow run and workflow job run events. When the events occur, the webhook is triggered and the workflow meta-data is sent to the external payload server. In this paper, the payload server is configured to be the bot, which uses the payload details to get the logs of the current workflow and process them. The developers can subscribe to this Webhook for details of the workflow, free of cost. For persisting the processed and labeled logs a cloud database storage service like MongoDB Atlas is used. A pre-trained ML model for keyword and keyphrase extraction is used for the labeling of logs. In this way, on each commits to the remote repository, the bot processes and persists the logs in real-time. As an extra feature, the error messages (if any) along with processed logs and details of the committer are sent to the developers of the organization through email automatically by the bot.

2) Design and implementation: Fig. 2 depicts the low-level details of the implementation:

- A web hook is configured with the pipeline to send a payload to the bot every time the workflow runs. The payload contains details about the workflow run. When the workflow completes, its logs are downloaded from the details provided in the payload.
- Since the downloaded logs are in a compressed format, they are automatically extracted to retrieve the actual log files. The files are then combined into a single file for the convenience of processing.
- The logs are then automatically pre-processed to remove particular control commands and color coding schemes that interfere with the parsing. This is followed by error extraction by looking for a particular sequence of terms.
- In case of an erroneous workflow run, the error part is extracted. In order to get the context of the error the previous few commands too, if possible, are extracted from the log. A pre-trained Keyword search algorithm is used to find out some possible causes related to the error by processing the extracted error as its input.
- All the logs are persisted in an external cloud database. For erroneous logs, the error details are automatically sent to the developers of the organization by mail.
- Extensive usage of system observers can be set up to parallelize the operations such as download completion detection, log extraction, etc. One such example of the system observers is the watchdog module which is used in the python implementation of the bot.

VII. Results

As a result of building the CI/CD pipeline and exploring different tools to work with, it was found that certain ones were more effective and customizable in an integrated CI/CD pipeline. This is depicted in Table 1.
By implementing this pipeline concept

- **build process is completely automated and just a single trigger or push commit is sufficient** from the developer's side for the deployment of existing or new code from large repositories on a real-time basis.
- **Only incremental or differential changes are implemented** with respect to infrastructure changes by employing infrastructure automation tools that adopt a declarative language approach

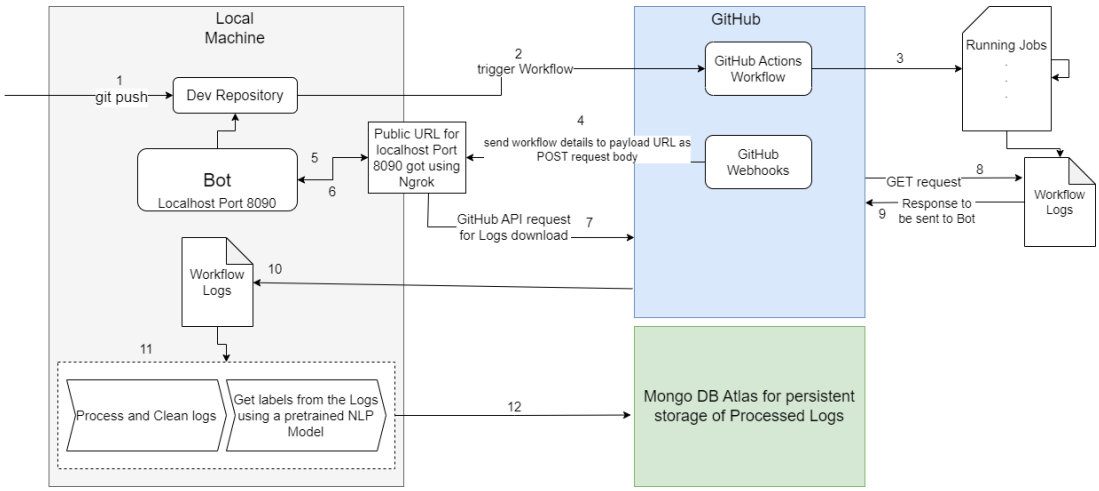

Fig. 2. Diagram showing the working of the bot server

- **Removes the hassle of obtaining different IP addresses for the instances each time** they are created by the usage of static IP addresses for deployment and ensures automatic DNS record creation in the DNS name servers using the infrastructure automation tools.
- **Automation of SSL certification activation is considered to be ineffective** since certificate activation and linking to the domain is only a one-time process for every domain which cannot be changed and can only be canceled.
- **Zero downtime deployments with load balancer**, zero downtime switching between different versions of the application was achieved using the blue-green deployment strategy.

By implementing the bot services

- **The logs became queryable and contributed to the ease of log readability**, by configuring the bot service to respond to each push to the source code repository on which the logs generated by the workflow during the continuous build and continuous delivery phases are downloaded, processed and error messages are labeled before persisting them onto a cloud database. If any errors are detected they are automatically emailed to the concerned people or teams who committed or authored the code.

VIII. Conclusion & future work

In this paper, we have showcased the POC of an improved automated CI/CD pipeline with a log-aware monitoring system. It is important for organizations to enable the developer(s) to write, manage, test, and deliver their software to their customers with minimal manual intervention in the operations phase of continuous engineering. This paper showcases the power of integrating DevOps tools to customize the workflow stages that the CI/CD pipeline performs and automate the creation of infrastructure by writing declarative code. GitHub Actions, Terraform, and AWS when used together with docker and any DNS provider proves to be effective for practical usage in the real world. Additionally, the paper proposed the usage of a customized application to complement the CI/CD pipeline. This bot-like service makes the monitoring easier for the developers and makes the workflow logs queryable for better debugging. We consider the quantitative measure metric for convenience to be the click count of the developer for various logins, setting up the infrastructure, monitoring the pipeline, and finding the cause of any underlying errors in the logs errors. The extent of automation of the continuous engineering processes shown in this paper reduces the click count significantly. This implies an improvement in convenience. For future work, we plan to further explore continuous engineering phases in the hope of finding opportunities for further automation of the pipeline. Furthermore, we plan to implement the proposal in a real-life business use case and thereby introspect the results of our approach and further enhance the pipeline according to changing business needs.

References

[1] Pekka Abrahamsson et al. “Agile software development methods: Review and analysis”. In: *arXiv preprint arXiv:1709.08439* (2017).

[2] Aleksi Häkli, Davide Taibi, and Kari Systä. “Towards Cloud Native Continuous Delivery: An Industrial Experience Report”. In: First International Workshop on Cloud-Native Applications Design and Experience (Zurich). 2018. DOI: 10.1109/UCC-Companion.2018.00074.

[3] Lianping Chen. “Towards architecting for continuous delivery”. In: *2015 12th Working IEEE/IFIP Conference on Software Architecture*. IEEE. 2015, pp. 131–134.

[4] Constantin Viorel Marian. “DNS Records Secure Provisioning Mechanism for Virtual Machines automatic management in high density data centers”. In: *2021 IEEE International Black Sea Conference on Commu-*

nications and Networking (BlackSeaCom). IEEE. 2021, pp. 1–5.

[5] A Khiyaita et al. “Load balancing cloud computing: state of art”. In: *2012 National Days of Network Security and Systems* (2012), pp. 106–109.

[6] Johannes Gmeiner, Julian Haslinger, and Rudolf Ramler. “Automated testing in the continuous delivery pipeline: A case study of an online company”. In: IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). 2015. DOI: 10.1109/ICSTW.2015.7107423.

[7] Charanjot Singh et al. “Comparison of Different CI/CD Tools Integrated with Cloud Platform”. In: 9th International Conference on Cloud Computing, Data Science Engineering (Confluence). 2019.

[8] Satvik Garg and Somya Garg. “Automated Cloud Infrastructure, Continuous Integration and Continuous Delivery using Docker with Robust Container Security”. In: IEEE Conference on Multimedia Information Processing and Retrieval (MIPR) (San Jose, CA). 2019.

[9] Qingwei Lin et al. “Log clustering based problem identification for online service systems”. In: IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). 2016.

[10] Carl Martin Rosenberg and Leon Moonen. “Improving problem identification via automated log clustering using dimensionality reduction”. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2018.