International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. #, No.# (Mon-20..)

Continuous Integration And Continuous Deployment Pipeline using GitOps, Jenkins and ArgoCD

Dr.Ramakanthkumar P[1], Dr.Pavithra H[2], Mr.Mirza Baig[3], Mr.Sumanth Hegde[4], Mr. Furqan Abdul Khadar Ramadurg[5], Mr.Virendra Naik[6]

[1,2,4,5,6]Computer Science & Engineering, RV College of Engineering, Bengaluru, India
[3]SAMSUNG R&D Institute, Bengaluru, India

E-mail address: ramakanthkp@rvce.edu.in, pavithrah@rvce.edu.in , mirza.daud@samsung.com, sumanthhegde.cs19@rvce.edu.in, furqanabdulk.cs18@rvce.edu.in, virendranaik.cs18@rvce.edu.in,

Received ## Mon. 20##, Revised ## Mon. 20##, Accepted ## Mon. 20##, Published ## Mon. 20##

Abstract: With companies incorporating agile software practices and with the increase in the number of developers collaborating on a given product, it becomes important to automate the development pipeline (build, test and deploy). Automation helps in reducing manual effort involved in software development, thereby increasing efficiency and making the entire system more robust. The entire development cycle can be viewed as a process of Continuous Integration (CI) and Continuous Deployment (CD), which can be achieved through several open-source tools. In the pipeline discussed in this paper, CI is achieved by creating a Jenkins job. Jenkins provides a wide range of plugins, which can be used to ease the processing required in the build tasks. A successful build results in an update of the helm chart corresponding to the application. These configuration files, which are stored in a Git repository, are constantly monitored by the ArgoCD controller, which automatically deploys the Kubernetes components of the application to the target cluster when a difference is observed between the state of the application as desired in the configuration files and current deployment in the target cluster. Thus with minimal manual intervention, developers can independently make changes to a given product and the corresponding artifacts are built and automatically deployed to production using CI/CD automation.

Keywords: Automation, CI/CD, Jenkins, plugins, Helm Chart, Kubernetes, ArgoCD

1. Introduction (Heading 1)

With traditional software development practices like the waterfall model, the lifecycle of the product is spanned over a larger period of time, which allows for manual logging of the version of the software, which is built and released. However, with agile practices taking over the enterprise world, the entire process has become iterative wherein there are frequent releases with each release targeting specific issues or client interests. Hence, time is of the essence and the process needs to be efficient with minimal errors involved. Thus, the sequence of steps, which are taken to build, test and deploy the application, needs to have minimal human involvement. Any large-scale product life cycle generally involves multiple environments for different purposes like development, test and production environments. Each release may also need a separate delivery pipeline depending on the design and dependencies of the software. Manually monitoring the execution of each stage and promoting to the next pipeline stage is inefficient and unreliable. Automation of the pipeline helps to overcome all these difficulties. Creation of a pipeline automation generally involves steps like Planning, Coding, and Building of the source code, Testing, Release and Deployment. These steps can be categorized into three distinct stages: Continuous Integration (CI), Continuous Delivery and Continuous Deployment (CD).

E-mail: ramakanthkp@rvce.edu.in

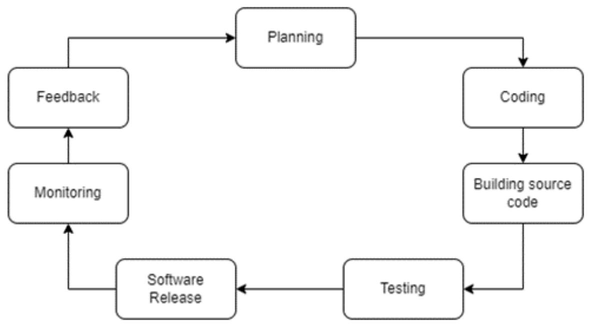


Figure 1. DevOps lifecycle

Continuous Integration is the practice of automatically integrating the code changes made by any of the multiple contributors involved in the development of the product, building the merged codebase and performing tests to assert correctness before release. A Version Control System like git helps each developer work independently and merge his contribution with the central repository. This repository can then be built into a logical artefact. This can be done by containerization of the application by creating Docker images. These images can be stored in a private registry, which also serves the purpose of logging of the successful build process. Validation of the software can be done using test suites created in an automated testing framework. Build pipelines can be developed using tools like Jenkins where jobs can be created which define actions necessary for the integration of the software being developed. Advantages include availability of plugins to easily achieve complex workflows, and the ease of creation as Jenkins provides a declarative approach for declaring build action steps.

Continuous Deployment is the practice wherein a change, which successfully passes all the stages of the pipeline, is released to the customer environment without any manual intervention. The artifacts generated as part of the build process are deployed as infrastructure in the client cluster using a container orchestration system like Kubernetes. This helps in automating tasks like rolling out new versions of software, container management and monitoring, scaling of infrastructure as per changing needs etc. Through this process, developers can initiate a faster feedback loop from the customers, as there is no waiting period for a release to happen.

Advantages of Using Jenkins, GitOps, and ArgoCD:

Consistency: GitOps ensures that the infrastructure and application deployments are consistent and reliable.

Auditability: All changes are tracked in Git, providing a clear audit trail.

Scalability: Jenkins can be used to manage complex CI pipelines, while ArgoCD scales to manage multiple clusters and environments.

Reduced Manual Intervention: Automation reduces the need for manual intervention, minimizing the risk of human errors.

Implementing a CI/CD pipeline using GitOps, Jenkins, and ArgoCD provides a robust, scalable, and automated approach to managing software delivery. This approach ensures that your applications are continuously tested, built, and deployed with minimal manual effort, resulting in faster, safer, and more reliable software releases.

2. LITERATURE SURVEY

Jenkins is an open source Continuous Integration tool widely used for automating build and test processes. Additionally Jenkins also provides a wide range of plugins, which makes it possible to use Jenkins in any given environment to execute any given, build step in the shortest time. Plugins also increase interpretability of the pipeline as simple calls provided in a plugin can be made to process even complicated steps. With the rise in test driven development (TDD)[1]automation of the testing process would become highly beneficial to companies. A. Deshpande et.al [2] have discussed the benefits of using CI for testing and the advantages of automated testing over manual testing. The paper which makes use of Jenkins discusses the automation pipeline that is possible because of the master slave architecture of Jenkins and how testing can be efficiently done with the numerous plugins available with Jenkins.

With software being utilized for different purposes in multiple fields, its development needs to be done for multiple sites and tested on multiple platforms. N. Seth et. al.[3] discuss the use case of Jenkins in a real-life scenario and how CI has been a crucial development for software development. The paper implements CI for an embedded system environment through which successive patches of software are efficiently integrated and released to the client.

Zebula Sampedro et. al., [4] illustrates how Jenkins has been adapted to ease software maintenance of High Performance computing (HPC) tools through high-level procedures. The paper provides the viewpoints of both the software architect who is creating the pipeline and a researcher who is maintaining and utilizing the codebase developed by the former.

Sriniketan Mysari et. al., [5] discusses an automation system with Jenkins used for CI and Ansible used for CD. The build process results in the creation of packaged artifacts. Ansible is an open source platform for configuration management. Ansible helps in provisioning of resources for the built artifacts and can be run on the Jenkins node itself and remote access is also provided.

Valentina Armenise et. al., [6] put forward how Jenkins, which is conventionally used for CI, has been extended to perform continuous delivery. The paper highlights how automation of product release and distribution could be achieved thereby unifying all the automation tasks in a single platform. The issues, which need to be addressed to efficiently monitor and achieve maximal output, are also discussed.

The main step in CI is building the source code. With the rise of microservice architecture, developers have started packaging applications as containers. Docker [7] is a platform, which enables the creation, storage and shipping of these containers. Containerization of the source application can be achieved using the Docker plugin available in Jenkins.

With the widespread adoption of containers, there comes a need to manage the containers in a sophisticated manner. Different products have different dynamic requirements and the number of containers can itself be a huge number. Hence, manual monitoring and maintaining of the containers becomes a highly inefficient process. Kubernetes is an open source platform, which helps in the orchestration of these containers and helps in automating the processes of monitoring and resource provisioning [8], deployment and scaling of applications as per the requirements.

Tesliuk et. al.,[9] present an efficient system making use of kubernetes infrastructure for data analysis. They efficiently deploy the application for each independent data payload and illustrate the advantages of Kubernetes.

L. Abdollahi Vayghan et. al.,[10] have experimented with deploying microservice based applications with Kubernetes. The paper highlights the results of how availability of the application varies with the configuration of the infrastructure.

Helm is a package manager of Kubernetes, which is highly useful as the complexity of Kubernetes [12] components increases. This tool is most useful when there is a high level of pattern and redundancy in configuration requirements of Kubernetes components. Helm charts streamline the process of deployment and managing Kubernetes apps. S. Gokhale et al., [11] make use of an experimental setup to compare how effective helm charts are for the deployment of enterprise level applications. They note that there is a considerable improvement in time taken for deployment and the maintenance process is simplified.

Fowler, M. [14] . In his seminal work on Continuous Integration, Martin Fowler describes CI as a practice where developers frequently integrate their code into a shared repository, which is then automatically tested. This practice aims to identify integration issues early, improving software quality and reducing time to market.

Shahin, M., Babar, M. A., & Zhu, L. [15]. The authors analyze various CI/CD practices and their impacts on software quality. They highlight the challenges in implementing CI/CD, such as tool integration, and emphasize the need for automated pipelines to reduce human errors and accelerate delivery.

Weaveworks [16]. GitOps, a term coined by Weaveworks, is a set of practices that uses Git as the source of truth for the entire system's desired state. The article introduces the concept of GitOps, emphasizing its role in simplifying operations and improving the reliability of CI/CD pipelines. It discusses how GitOps leverages existing CI/CD tools to automate infrastructure management and application deployment.

Pena, M. [17]. The paper provides an in-depth analysis of how GitOps can be integrated into traditional CI/CD pipelines. It compares GitOps with Infrastructure as Code (IaC) approaches and highlights the benefits of using GitOps for managing Kubernetes environments, including version control, auditability, and improved collaboration between development and operations teams.

Hilton, M., Tunnell, T., Huang, K., Marinov, D., & Dig, D. [18]. This study investigates the adoption of Continuous Integration practices, focusing on Jenkins as a widely used tool. It discusses the challenges organizations face in adopting Jenkins, including plugin management and pipeline complexity. The paper concludes that while Jenkins is a powerful tool, its effectiveness depends on proper configuration and management.

Fitzgerald, B. [19]. This article highlights Jenkins' role in automating CI/CD pipelines. It examines Jenkins' flexibility in integrating with various tools and platforms, making it an ideal choice for diverse development environments. The paper also explores the concept of Pipeline as Code and how Jenkins facilitates versioning and managing CI/CD pipelines as code.

Argo Project [20]. The ArgoCD documentation provides an overview of ArgoCD as a GitOps-based continuous delivery tool for Kubernetes. It explains how ArgoCD automates the deployment of applications by synchronizing the live state of applications with the desired state defined in Git repositories. The documentation also discusses ArgoCD's features, such as automated rollbacks, multi-cluster management, and integration with existing CI/CD tools like Jenkins.

Brikman, Y. [21]. The author explores the practical implementation of ArgoCD in a Kubernetes-based environment. The paper discusses how ArgoCD complements CI/CD pipelines by providing continuous monitoring and automated synchronization of application states. It also highlights the benefits of using ArgoCD for managing microservices architectures and multi-cloud deployments.

Netflix Technology Blog [22]. This blog post discusses how Netflix adopted GitOps and ArgoCD to manage its large-scale, multi-cloud infrastructure. The post outlines the challenges Netflix faced with traditional CI/CD practices and how GitOps and ArgoCD helped streamline their deployment processes, improve scalability, and enhance system reliability.

Intuit Engineering [23]. Intuit's engineering team provides a case study on implementing GitOps using Jenkins and ArgoCD. The paper discusses how Intuit integrated these tools to create a fully automated CI/CD pipeline for their Kubernetes-based applications. The case study highlights the benefits of using GitOps principles, such as reducing deployment times and increasing deployment frequency.

Rahman, M. A., & Williams, L. [24]. The paper identifies the challenges of adopting GitOps, Jenkins, and ArgoCD in CI/CD pipelines, including the complexity of tool integration, the learning curve for development teams, and the need for robust security practices. It also suggests

future research directions, such as improving the scalability of GitOps-based pipelines and enhancing the integration between CI/CD tools.

Xu, Y., & Bass, J. M. [25]. This paper explores the future of CI/CD pipelines, focusing on the evolution of GitOps and its potential to simplify operations and improve collaboration between development and operations teams. The authors suggest that future CI/CD pipelines will likely integrate AI and machine learning to further automate decision-making processes and enhance pipeline efficiency.

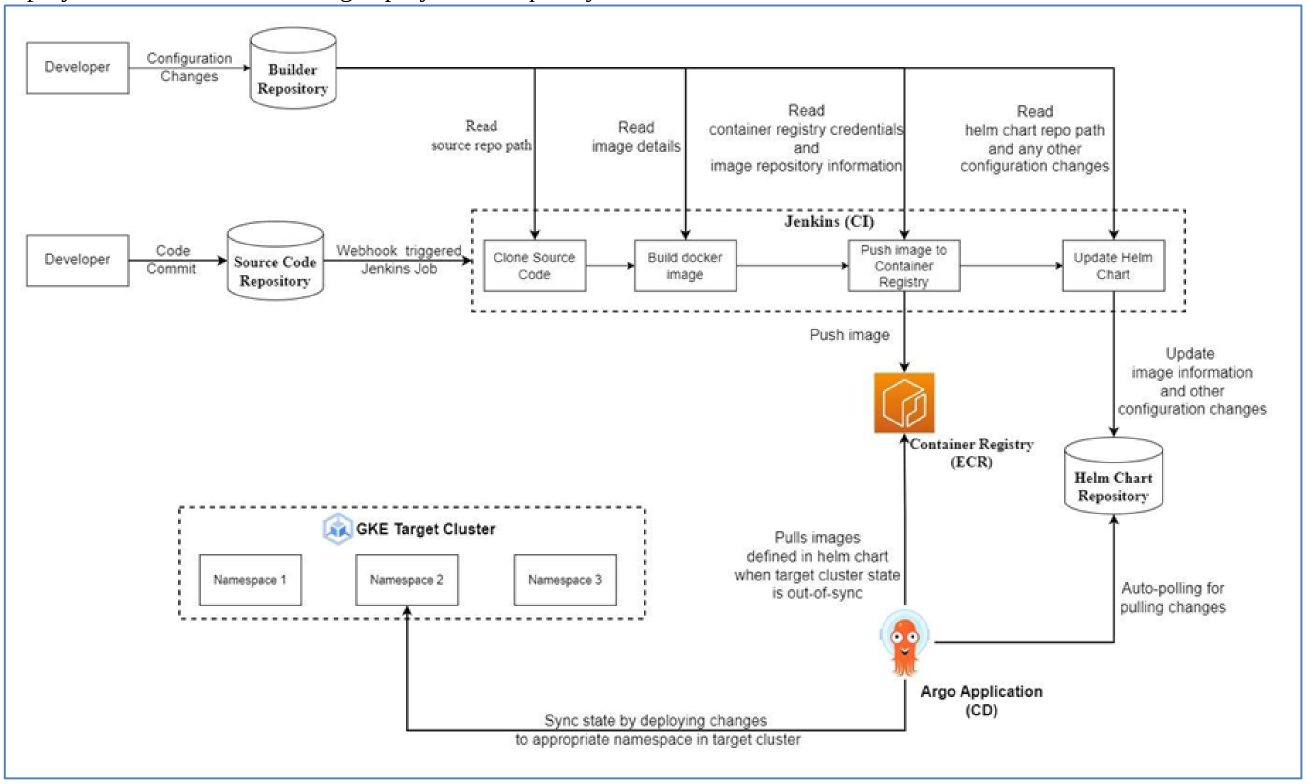

Developer
Configuration Changes
Builder Repository
Read source repo path
Read image details
Read container registry credentials and image repository information
Read helm chart repo path and any other configuration changes
Developer
Code Commit
Source Code Repository
Webhook triggered Jenkins Job
Jenkins (CI)
Clone Source Code
Build docker image
Push image to Container Registry
Update Helm Chart
Push image
Update image information and other configuration changes
Container Registry (ECR)
Helm Chart Repository
GKE Target Cluster
Namespace 1
Namespace 2
Namespace 3
Pulls images defined in helm chart when target cluster state is out-of-sync
Auto-polling for pulling changes
Argo Application (CD)
Sync state by deploying changes to appropriate namespace in target cluster

Figure 2. Systems Architecture

The literature surveyed demonstrates that GitOps, Jenkins, and ArgoCD are powerful tools that significantly enhance CI/CD pipelines. GitOps provides a robust framework for managing infrastructure and application deployments through version control, while Jenkins offers a flexible automation server that integrates well with various tools. ArgoCD complements these by providing continuous delivery capabilities tailored for Kubernetes environments. Together, these tools enable organizations to implement highly automated, reliable, and scalable CI/CD pipelines, leading to faster and more frequent software releases.

3. METHODOLOGY

Creating a Continuous Integration (CI) and Continuous Deployment (CD) pipeline using GitOps, Jenkins, and ArgoCD involves several steps. Below is a high-level guide to building such a pipeline.

The automation proposed here is based on the Gitops[13] principle where git acts as the single source of truth. The infrastructure required for the application is represented using configuration files, which are stored on a version control system like github. This helps in logging the infrastructure details in a structured manner, which can be used to roll back to a stable version of the software in case of any failures. Additionally it eases the process of automation of infrastructure provisioning.

A Jenkins server is set up and a job is created for all the projects to be built. Since a custom declarative pipeline is being used, automatic polling to recognize any changes in the source repository is achieved by associating the job with a web hook created on the repository. Thus, when a developer pushes code, a request is sent to the Jenkins server, which then is routed to the required job where the pipeline is triggered.

Additionally, a builder repository and helm chart repositories are also maintained on git as shown in Figure. 3. The builder repo is essentially a YAML file which contains all the necessary information for the entire CI/CD pipeline like paths of source codes, name of the repository on the cloud registry where the built image has to be stored, tags for the desired image, namespace in which the application needs to be deployed, helm chart repository paths and any other infrastructure configuration information. The helm chart repo consists of template configuration files for Kubernetes components and a value.yaml file, which injects the actual values into the placeholders in the template files.

Developer pushes code, a request is sent to the Jenkins server, which then is routed to the required job where the pipeline is triggered.

Additionally, a builder repository and helm chart repositories are also maintained on git as shown in Figure. 3. The builder repo is essentially a YAML file which contains all the necessary information for the entire CI/CD pipeline like paths of source codes, name of the repository on the cloud registry where the built image has to be stored, tags for the desired image, namespace in which the application needs to be deployed, helm chart repository paths and any other infrastructure configuration information. The helm chart repo consists of template configuration files for Kubernetes components and a values.yaml file, which injects the actual values into the placeholders in the template files.

A Jenkins server is set up and a job is created for all the projects to be built. Since a custom declarative pipeline is being used, automatic polling to recognize any changes in the source repository is achieved by associating the job with a web hook created on the repository. Thus when a

The Jenkins pipeline is divided into stages: cloning of the source code repository whose path is obtained from the builder repository, building the Docker image for the source code based on a predefined Docker file (which is decided based on the tech stack used in the project), pushing the built image to the desired cloud registry and updating the image information in the values. yaml file in the helm chart repository. Private git repositories cannot be read or updated without authorization. Public repositories on the other hand, can be cloned but cannot be updated. Thus for authorization purposes, auth tokens are created and are stored in Jenkins credentials. Jenkins provides a plugin which authenticates with github and within whose call scope, clone and commit operations can be made to the corresponding repository.

If the image is pushed to a private repository, then ArgoCD will not be able to pull images from the container registry and deploy the application to the target cluster. As part of the update stage in the Jenkins pipeline, a token is

E-mail: ramakanthkp@rvce.edu.in

generated using the AWS plugin, which enables the image to be pulled. This token is stored in the helm repository as a Kubernetes Secret. This secret is referenced in other Kubernetes components, which have to pull the desired image. As the token is valid only for a period of 12 hours, a cronjob can be set up to update the secret every 12 hours.

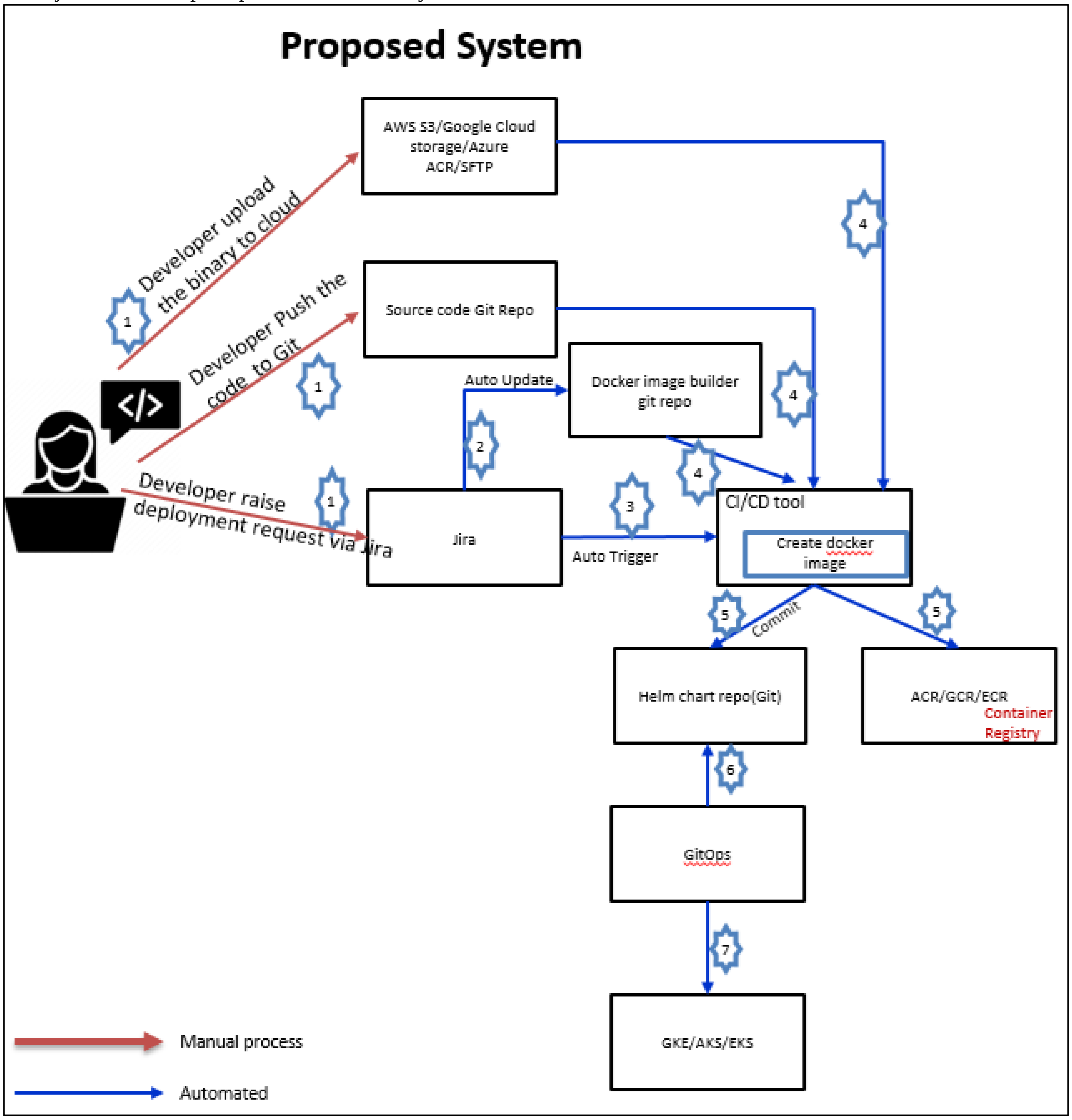

Proposed System
AWS S3/Google Cloud storage/Azure ACR/SFTP
Developer upload the binary to cloud
1
Source code Git Repo
Developer Push the code to Git
1
Auto Update
Docker image builder git repo
2
4
4
4
Developer raise deployment request via Jira
1
Jira
3
Auto Trigger
CI/CD tool
Create docker image
5
Commit
5
Helm chart repo(Git)
ACR/GCR/ECR Container Registry
6
GitOps
7
GKE/AKS/EKS
Manual process
Automated

Figure 3. Proposed System

Continuous deployment is achieved using ArgoCD, which follows the GitOps pattern. Each helm repository, which is to be monitored, is associated with an argo application. As a pre-configuration step, the local Kubernetes cluster in which the ArgoCD server exists is connected to all the external cloud clusters in which a deployment might be necessary. Thus, each argo application is connected to its respective target cluster. With the successful completion of the Jenkins job, a new commit is made to the helm repository. The ArgoCD controller which continuously polls all the argo applications (those which have been configured with automatic sync policy) recognizes the new commit and compares the state of the target cluster and that of the state which is defined in the helm repository. A deployment is triggered in the case where the states are different.

Summary Workflow
Developer Workflow:

- The developer pushes code to a feature branch.
- Jenkins pipeline is triggered to run tests and build the application.
- Upon successful build, the application is containerized, and the Docker image is pushed to a registry.

CI/CD Pipeline:

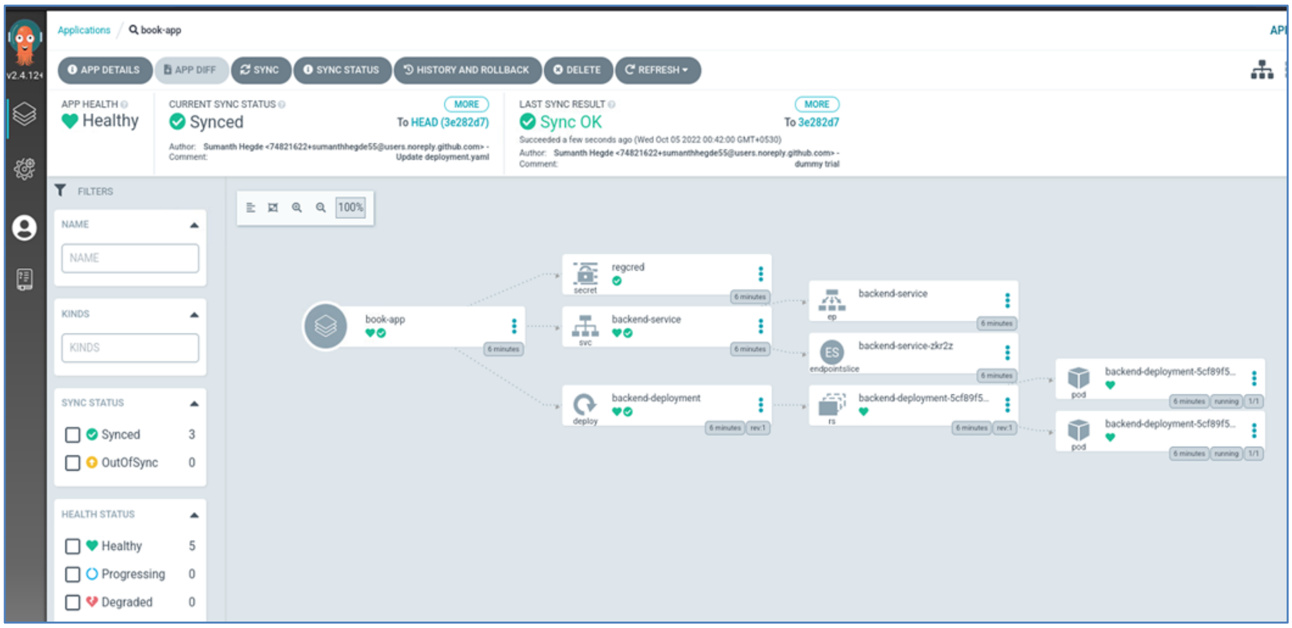
- Jenkins updates the Git repository with new manifests.
- ArgoCD detects changes in the Git repository.
- ArgoCD synchronizes the Kubernetes cluster to match the desired state defined in Git.

Deployment:

- The application is deployed to the Kubernetes cluster.
- Monitoring and logging tools keep track of the application's health and performance.

Tools Summary:
- Jenkins: Automates the CI process.
- ArgoCD: Manages the CD process via GitOps.
- Git: Acts as the single source of truth for the application's desired state.

This pipeline provides an efficient, automated, and robust CI/CD process, leveraging the strengths of GitOps for infrastructure management and the flexibility of Jenkins for CI.



APP HEALTH: Healthy

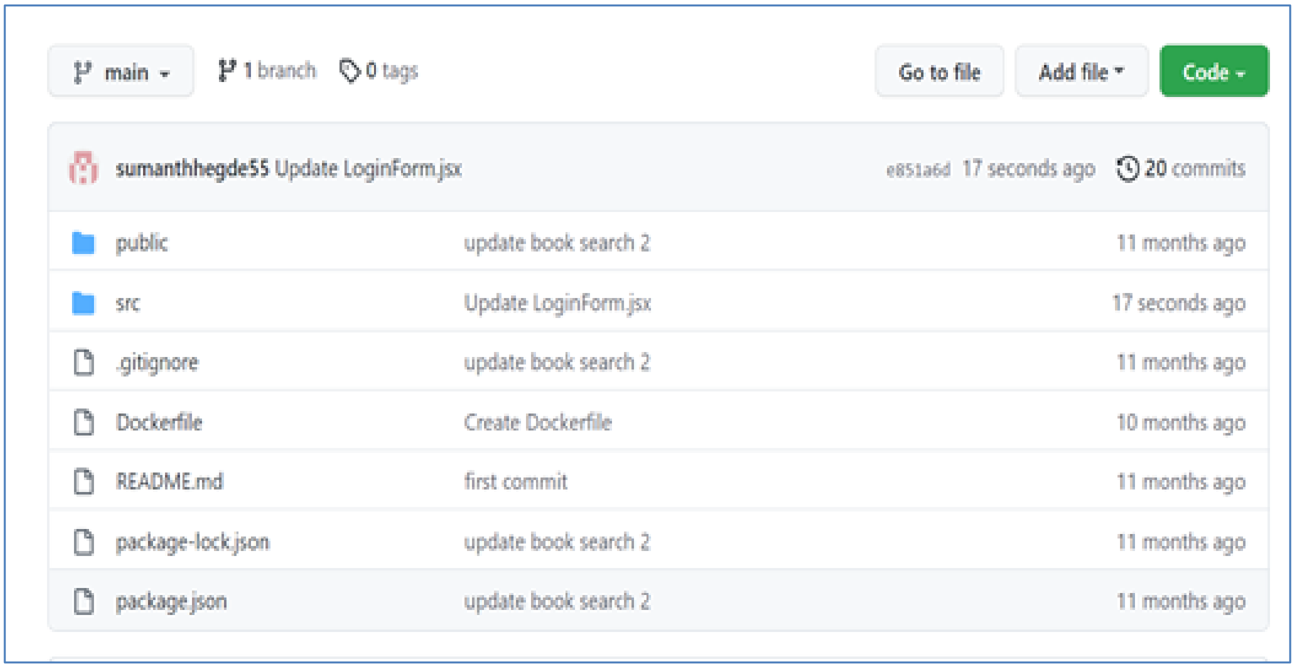
CURRENT SYNC STATUS: Synced — To HEAD (3e282d7)
Author: Sumanth Hegde <74821622+sumanthhegde55@users.noreply.github.com> -
Comment: Update deployment.yaml

LAST SYNC RESULT: Sync OK — To 3e282d7
Succeeded a few seconds ago (Wed Oct 05 2022 00:42:00 GMT+0530)
Author: Sumanth Hegde <74821622+sumanthhegde55@users.noreply.github.com> -
Comment: dummy trial

Figure 4. ArgoCD application, which polls the helm repository and automatically synchronizes the target cluster with the desired changes

3. RESULTS

Jenkins requires a manual triggering of a job for the very first time. Apart from that, the entire development to deployment process has been achieved with minimal manual effort.

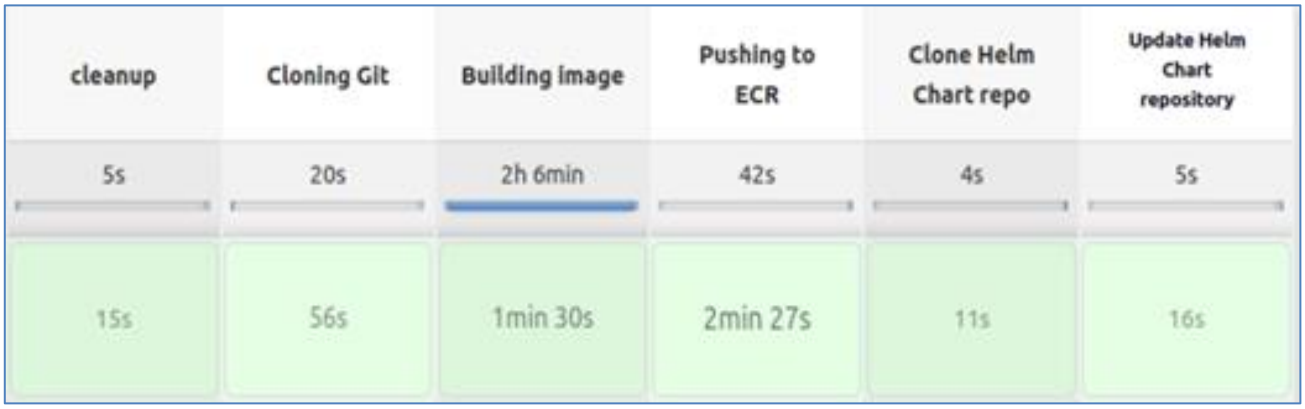


main · 1 branch · 0 tags · Go to file · Add file · Code

sumanthhegde55 Update LoginForm.jsx · e851a6d 17 seconds ago · 20 commits

public	update book search 2	11 months ago
src	Update LoginForm.jsx	17 seconds ago
.gitignore	update book search 2	11 months ago
Dockerfile	Create Dockerfile	10 months ago
README.md	first commit	11 months ago
package-lock.json	update book search 2	11 months ago
package.json	update book search 2	11 months ago

Figure 5. Source code commit, the repository has a webhook, which is triggered whenever any commit is made.



cleanup	Cloning Git	Building image	Pushing to ECR	Clone Helm Chart repo	Update Helm Chart repository
5s	20s	2h 6min	42s	4s	5s
15s	56s	1min 30s	2min 27s	11s	16s

Figure 6. Jenkins pipeline with 6 stages executed successfully.

E-mail: ramakanthkp@rvce.edu.in

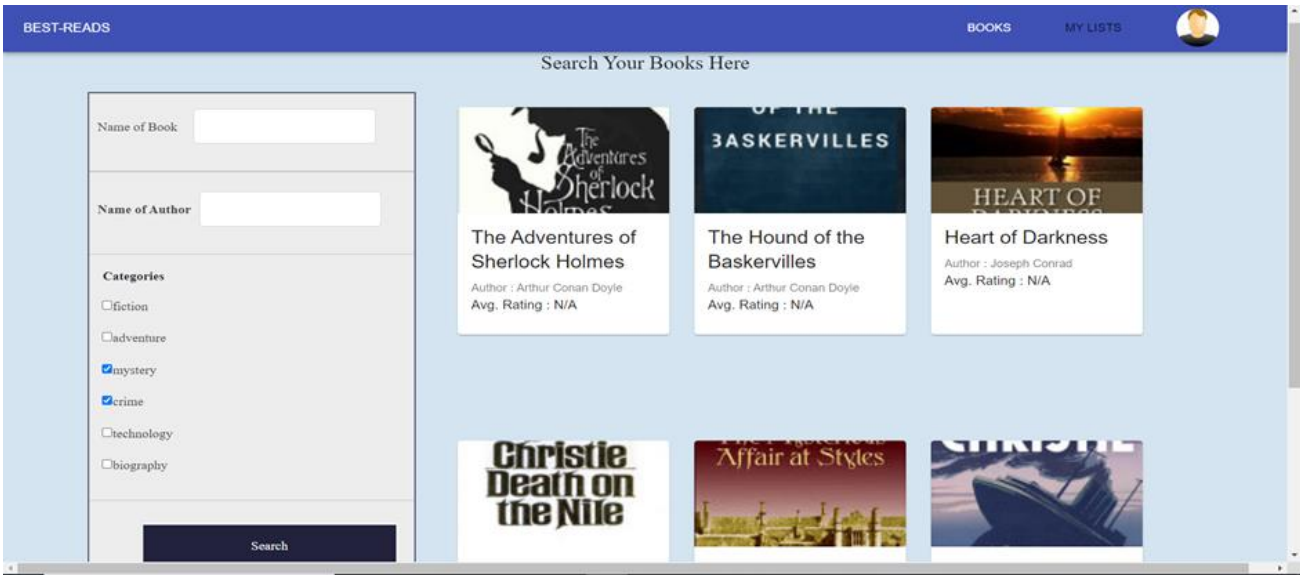


Figure 7. Deployed application

As shown in Figure. 7, the application being deployed is a react application with the data being fetched from Mongodb in the backend application, which is a dependency. A list of dockerfile templates is maintained for all the applications, which are currently being provided to the client. Thus, map each application to one of these predefined templates. Both the main application and the dependency trigger independent pipelines, which result in separate builds. The artifacts are linked at the time of deploying of the infrastructure.

Once a new commit is made as in Figure. 5, a jenkins job is triggered. The source code is cloned using the git plugin. The image is built using the docker plugin according to the blueprint of the dockerfile. The aws plugin is used to push the image to the ECR registry. Finally, update the helm repository by cloning and making the required changes in the values. yaml file using the utility plugin. This triggers deployment to the target cluster. The target cluster is identified via information like the cluster region, GCP project ID etc obtained through the builder repository. The context of the cluster is available by authenticating with gcloud cli and fetching the credentials for the target cluster. This results in the context information being added to the local kube.config file which can be made accessible to the ArgoCD controller using argo-cli add context command.The built application is accessible through the service created as part of the configuration. The application is shown in Figure 7. The health of the various components of the application is also automatically monitored in ArgoCD, which is visible in the dashboard as shown in Figure. 4.

By maintaining a global configuration file, the manual task of providing all the information at various stages of the build process has been eliminated. The manual process of configuring Kubernetes components also has been reduced significantly by using helm charts with placeholders, especially for single deployment applications with no networking between components. As the Gitops methodology is being used, the entire infrastructure is systematically logged in the git repository, hence we can easily shift between different versions of infrastructure, and hence the system becomes more reliable.

ArgoCD applications have the property of multi-tenancy. This can be used to introduce a checkpoint in the automation pipeline where a manual checking can be done before the product is released to the customer. The build from jenkins can be updated in a secondary branch. By having ArgoCD deploy to a test cluster, the infrastructure, which would be provided to the customer, can be verified from an authorized personnel. The branch can then be merged to the master branch, which would automatically trigger the deployment to the desired client cluster.

4. CONCLUSION AND FUTURE WORK

In this paper, an automation pipeline has been proposed making use of open-source tools like Jenkins and ArgoCD.

The manual efforts, which are conventionally required for building and deploying applications, have been reduced by making use of a builder repository and the features of the CI/CD tools. The blueprint for the Docker images are also coupled with the application code, which enables easy building of the application. By following the GitOps pattern, the entire infrastructure is maintained as code in the form of Helm charts, which can easily be mapped to logical artifacts.

In future enhancements, the entry point for the automation could be extended to include Jira tickets from which the required information can be extracted to trigger the pipeline. The pipeline could also be modified to download compressed source packages in addition to cloning code from repositories.

ACKNOWLEDGMENT

This work is carried out as part of SAMSUNG Prism project. Thanks to SAMSUNG Research India.

REFERENCES

[1] A. S. Dookhun and L. Nagowah, "Assessing The Effectiveness Of Test-Driven Development and Behavior-Driven Development in an Industry Setting," 2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), 2019, pp. 365-370, doi: 10.1109/ICCIKE47802.2019.9004328.

[2] A. Deshpande, S. V. Veenadevi and S. Aleti, "Test Automation and Continuous Integration using Jenkins for Smart Card OS," 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2021, pp. 01-05, doi: 10.1109/ICCCNT51525.2021.9580021.

[3] N. Seth and R. Khare, "ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development," 2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS), 2015, pp. 1-6, doi: 10.1109/RAECS.2015.7453279.

[4] Zebula Sampedro, Aaron Holt and Thomas Hauser, "Continuous Integration and Delivery for HPC," Practice and Experience in Advanced Research Computing, pp. 22-26, 2018

[5] S. Mysari and V. Bejgam, "Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1-4, doi: 10.1109/ic-ETITE47903.2020.239.

[6] Valentina Armenise, "Continuous Delivery with Jenkins," IEEE/ACM 3rd International Workshop on Release Engineering, pp. 24-27, 2015.

[7] V. Sharma, H. K. Saxena and A. K. Singh, "Docker for Multi-containers Web Application," 2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), 2020, pp. 589-592, doi: 10.1109/ICIMIA48430.2020.9074925.

[8] E. Kim, K. Lee and C. Yoo, "On the Resource Management of Kubernetes," 2021 International Conference on Information Networking (ICOIN), 2021, pp. 154-158, doi: 10.1109/ICOIN50884.2021.9333977.

[9] A. Tesliuk, S. Bobkov, V. Ilyin, A. Novikov, A. Poyda and V. Velikhov, "Kubernetes Container Orchestration as a Framework for Flexible and Effective Scientific Data Analysis," 2019 Ivannikov Ispras Open Conference (ISPRAS), 2019, pp. 67-71, doi: 10.1109/ISPRAS47671.2019.00016.

[10] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe and F. Khendek, "Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018, pp. 970-973, doi: 10.1109/CLOUD.2018.00148.

[11] S. Gokhale et al., "Creating Helm Charts to ease deployment of Enterprise Application and its related Services in Kubernetes," 2021 International Conference on Computing, Communication and Green Engineering (CCGE), 2021, pp. 1-5, doi: 10.1109/CCGE50943.2021.9776450.

[12] S. Telenyk, O. Sopov, E. Zharikov and G. Nowakowski, "A Comparison of Kubernetes and Kubernetes-Compatible Platforms," 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2021, pp. 313-317, doi: 10.1109/IDAACS53288.2021.9660392.

[13] S. Gupta, M. Bhatia, M. Memoria and P. Manani, "Prevalence of GitOps, DevOps in Fast CI/CD Cycles," 2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON), 2022, pp. 589-596, doi: 10.1109/COM-IT-CON54601.2022.9850786.

[14] Fowler, M. (2006). Continuous Integration. ThoughtWorks.

[15] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous Integration, Delivery, and Deployment: A Systematic Review on Approaches, Tools, Challenges, and Practices. IEEE.

[16] Weaveworks (2017). GitOps - What you need to know.

[17] Pena, M. (2020). Integrating GitOps into Continuous Integration/Continuous Deployment Pipelines.

[18] Hilton, M., Tunnell, T., Huang, K., Marinov, D., & Dig, D. (2016). Continuous Integration Needs Continuous Testing: A Study of CI Usage in Open-Source Projects. ACM.

[19] Fitzgerald, B. (2015). Jenkins: The Definitive Guide. O'Reilly Media.

[20] Argo Project (2018). ArgoCD Documentation.

[21] Brikman, Y. (2020). ArgoCD: Implementing Continuous Delivery with GitOps.

[22] Netflix Technology Blog (2019). How Netflix Leverages GitOps for Multi-Cloud Deployment.

[23] Intuit Engineering (2020). CI/CD at Scale with Jenkins and ArgoCD.

[24] Rahman, M. A., & Williams, L. (2019). Challenges and Best Practices in Adopting Continuous Delivery. IEEE.

[25] Xu, Y., & Bass, J. M. (2021). The Future of Continuous Delivery: Beyond Pipelines. IEEE.

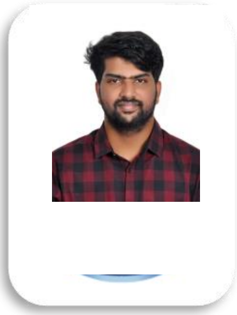
Dr Ramakanth Kumar P is a Professor and HOD in the Computer Science and Engineering department at RVCE. His research interests are Digital Image Processing, Pattern Recognition and Natural Language processing. He has published over 100 research papers. He has executed several funded research and consultancy projects sponsored by DRDO, ISRO, AICTE, GE India Pvt.Ltd, CABS, HP, Nihon Communication Solutions Pvt.Ltd etc.

Dr.Pavithra H Assistant Professor in the Computer Science and Engineering department at RVCE.Her research interests are Software Defined Networks, Machine Learning, Deep Learning, Software Engineering. She has executed projects sponsored by Samsung, ToyotO etc.

Mirza Abu Daud Baig is an Architect at Samsung R&D Institute India, Bengaluru

Sumanth Hegde is a student at CSE, RV College of Engineering, Bengaluru.

Furqan Abdul Khadar Ramadurg is a student at CSE, RV College of Engineering, Bengaluru.



Virendra Naik is a student at CSE, RV College of Engineering, Bengaluru.