International Journal of Scientific Research in Science, Engineering and Technology

Print ISSN - 2395-1990
Online ISSN : 2394-4099

OPEN ACCESS

Available Online at :www.ijsrset.com
doi : https://doi.org/10.32628/IJSRSET2410614

Optimizing DevOps Pipelines with Automation: Ansible and Terraform in AWS Environments

Venkat Marella
Independent Researcher, USA

ARTICLEINFO

Article History:

Accepted: 18 Nov 2024
Published: 13 Dec 2024

Publication Issue :
Volume 11, Issue 6
November-December-2024
Page Number :
285-294

ABSTRACT

In order to improve operational efficiency and agility in contemporary IT systems, this study investigates the integration of DevOps methods with cloud management. It offers a thorough rundown of the main DevOps tools and technologies needed to manage cloud infrastructure, including as monitoring systems, CI/CD pipelines, Infrastructure as Code (IaC) tools, and configuration management systems. The inability to allow concurrent project development and deployment on the same operational infrastructure (such as a cluster of Docker containers) is a practical shortcoming of current DevOps systems. In order to fill the gaps in the current Development and Operations (DevOps) methods, this paper offers a thorough study and explores how such integrations in Amazon Web Services might enhance deployment efficiency, dependability in software and infrastructure delivery, and security. Thus, the goal of this research is to use the AWS platform to automate the processes of developing and maintaining IT infrastructure. Therefore, we provide a mathematical model in this research to ascertain the ideal arrangement for IT infrastructure. This study investigates in detail how Kubernetes clusters in the AWS environment may be efficiently automated, scaled, and managed using Terraform, an Infrastructure as Code (IaC) tool. It thoroughly examines the advantages of using Terraform, highlighting how it may enhance productivity, automation, scalability, and security while managing Kubernetes clusters. To demonstrate Terraform's capabilities in infrastructure management, the paper contrasts it with other popular (IaC) tools and techniques. It also explores how Terraform works with AWS services to streamline procedures and cut down on complexity. Trends and possible developments in combining Kubernetes and Terraform to improve the administration of cloud-native apps are also covered in the paper.

Keywords: - (IaC), Kubernetes Clusters, AWS Platform, Cloud Native, Mathematical Model, IT Environments, CI/CD Pipelines, Simultaneous

Project, Cloud Management, (DevOps).

1. Introduction

DevOps and continuous integration/continuous delivery (CI/CD) methodologies are highly valued by businesses and organizations in the present day [1]. These approaches seek to improve their companies' operational efficiency, productivity, agility, and [1, 2]. Additionally, they want to provide a dependable software development environment with quick response times. DevOps promotes cooperation to produce products of greater quality and dependability by facilitating the convergence of formerly distinct responsibilities, including as development, IT operations, quality engineering, and security [2]. Teams will be better equipped to handle client needs, feel secure in the applications they create, and accelerate the achievement of corporate goals by adopting a DevOps mind-set and the techniques and tools that go along with it. Development teams may deliver code changes quickly and reliably when CI/CD approaches are used because DevOps and CI/CD work together to support the shared objective of accelerating and simplifying software development [2, 3].

Micro services and containers may be a perfect addition to an enterprise's DevOps ecosystem by increasing agility and flexibility [3]. Developer teams may design loosely linked, lightweight, independently deployable services that are simple to manage, build, and implement using DevOps and microservices. Software with a micro service-based design often uses containers because they provide platform independence and interoperability, two features that are ideal for micro service architecture. Without being impacted by services hosted in other containers, each containerized micro service may be set up and maintained separately. Because Docker containers provide an easy method to package code and send it to the release pipeline, which expedites the DevOps process, organizations primarily utilize them to package and deploy micro service-based applications.

Microservices and containers may be a perfect addition to an enterprise's DevOps ecosystem by increasing agility and flexibility. Developer teams may design loosely linked, lightweight, independently deployable services that are simple to manage, build, and implement using DevOps and microservices. The objectives of microservice architecture are well aligned with the platform freedom and interoperability provided by containers, [2], which are often used in applications with a microservice-based design. Services hosted in other containers won't have an impact on the deployment and management of each containerized microservice. Microservice-based applications are often packaged and deployed using Docker containers because they provide an easy-to-use method for packaging code and pushing it to the release pipeline, which expedites the DevOps process [2, 3].

1.1 Elastic Kubernetes Service (EKS)

Installing, running, and maintaining a personal Kubernetes control plane or nodes is not necessary when using Amazon EKS, a managed service for running Kubernetes on AWS. To provide high availability, EKS scales and operates the Kubernetes control plane across many AWS availability zones. EKS offers a reliable Kubernetes solution on-premises that is completely compatible with integrated tools

and easy deployment to bare metal servers, virtual machines, and AWS Outposts [3, 4]. Users may expand and operate native AWS services on-site using AWS Outposts solutions. Amazon EKS has historically been used for deployment in hybrid settings [3].

Regardless of whether they are operating in public clouds or on-premise data centers, apps running on EKS are completely interoperable with those running on any other standard Kubernetes environment. This implies that moving any typical Kubernetes application to EKS is simple and requires no changes to the code [4]. EKS is linked with several AWS services to provide customers' applications security and scalability. AWS Elastic Container Registry (ECR) for container images and Amazon VPC for isolation are two instances of the capabilities offered [4, 5]. Users may use all of the current plugins and tools from the Kubernetes community since it runs updated versions of the open-source Kubernetes software.

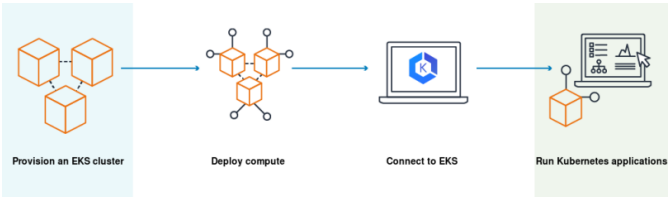

Fig. 1 Amazon EKS setup process. [5]

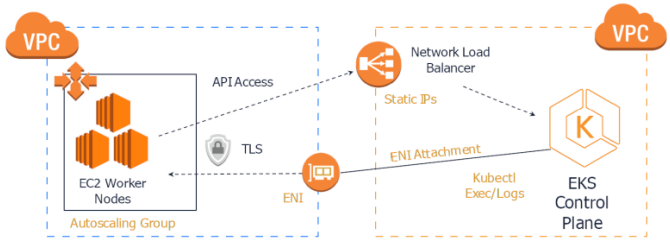

Fig. 2 Amazon EKS architecture. [5, 6]

- Other AWS services, such as AWS Identity and Access Management (IAM) for authentication [6], AWS ECR for container images, and AWS Elastic Load Balancer (ELB) for load distributions, are accessible to users that utilize EKS;
- As previously stated, EKS offers a highly available and scalable Kubernetes control plane that operates over many AWS availability zones [6, 7]. In addition to automatically identifying and replacing problematic control plane nodes, it ensures high availability by executing the Kubernetes control plane throughout the availability zones;
- EKS is often used in hybrid installations. Containerized apps that need minimal latency to on-premises systems may be operated on AWS Outposts. This functionality allows customers to manage containers on-premises just as easily as they do in the cloud; [8]

1.2 Key DevOps Tools for Cloud Infrastructure Management

Optimizing resource use, preserving performance, and guaranteeing scalability all depend on efficient cloud infrastructure management [5, 8]. DevOps technologies are essential for automating and optimizing these procedures, including features that improve the administration of cloud infrastructure. Some of the most well-known DevOps tools and technologies designed to support cloud systems are examined in this section [8].

- **Infrastructure as Code (IaC) Tools:** Terraform, an open-source program created by Hashi Corp., makes it possible to manage cloud infrastructure using declarative configuration files. AWS, [8], Azure, and Google Cloud Platform (GCP) are among the few cloud providers it supports. Terraform ensures reliable and repeatable deployments by enabling users to specify infrastructure using an advanced configuration language that can be versioned and repurposed.
- **AWS Cloud Formation:** With Cloud Formation, which was created especially for AWS, customers may utilize JSON or YAML templates for building and administering AWS resources [8, 9]. It offers a complete solution to administer complex cloud settings by automating resource provisioning and integrating with other AWS services [9].

- **Continuous Integration and Continuous Deployment (CI/CD) Tools:**
  - **Jenkins:** Jenkins, a popular open-source CI/CD solution, automates the deployment, testing, and building processes [9]. It facilitates easy deployment to cloud settings by supporting integration with a variety of plugins and technologies. Jenkins makes continuous delivery easier by automating tedious procedures and giving instant feedback on changes to the code.
  - **GitLab CI/CD:** GitLab CI/CD, a component of the GitLab platform, provides integrated CI/CD features that let developers create, test, and publish code straight from the GitLab repository [9, 10]. It enhances cooperation and productivity in the development process by integrating with several cloud providers and offering strong support for cloud installations.
- **Container Orchestration Tools:**
  - **Kubernetes:** An open-source framework called Kubernetes makes it possible to automate the deployment, scaling, and administration of containerized applications. Effective administration of containerized workloads and services in a cloud context is made possible by its strong orchestration architecture, which connects with several cloud providers [10, 11].
  - **Docker Swarm:** The native cluster and orchestrating tool for Docker is called Docker Swarm. By offering an intuitive interface for installing and expanding software across numerous hosts, it streamlines the administration of Docker containers [11, 12]. In cloud settings, Docker Swarm is often used to efficiently manage containerized applications.
- **Security and Compliance Tools:**
  - **Aqua Security:** A set of tools for protecting cloud-native environments and containerized apps is provided by Aqua Security. It helps businesses secure their cloud infrastructure from attacks and maintain regulatory compliance by offering services like vulnerability assessment, runtime protection, and compliance monitoring [12].
  - **Hashi Corp Vault:** Vault is a solution for sensitive data protection and secret management. In order to ensure security in cloud settings, it offers safe storage and access restrictions for passwords, [11], API keys, and other sensitive data.

An open source platform called Kubernetes, also referred to as K8s, is used to automate the deployment, scaling, and operation of application containers [12, 13]. By efficiently managing containerized apps across many servers in a cluster, it contributes to application deployment. Increased scalability, dependability, and efficiency are the outcomes of this. Amazon offers a managed solution called Amazon Elastic Kubernetes solution (EKS) that makes it easier to operate Kubernetes on AWS [13]. Because EKS manages the Kubernetes control plane automatically, users don't need to bother about setting it up or keeping it up. To guarantee uptime and security, EKS handles duties like patching and upgrading. Developers may focus on creating apps and using connectors with AWS services by letting AWS handle the Kubernetes control plane [14].

2. Literature Review

2.1. Terraform and Infrastructure as Code

Hashi Corp.'s Terraform has emerged as a tool in the Infrastructure as Code (IaC) space. It enables the use of an understandable language for the definition, provisioning, and management of cloud infrastructure [15]. Terraform's ability to provide a repeatable procedure for configuring and maintaining infrastructure across cloud platforms and service providers is a crucial component of its infrastructure management function [15, 16]. Terraform facilitates

the automation of environment setup by converting infrastructure into code, which lowers the possibility of mistakes, increases efficiency, and permits version control over infrastructure settings [16].

2.2. Kubernetes cluster management

- **Review of traditional methods for managing kubernetes clusters:** In this regard, managing Kubernetes clusters usually entails using scripts and configuration files to manage deployments, services, and networking in addition to command line tools like kubectl for cluster orchestration [17]. Although these techniques provide a certain amount of flexibility and control, they may be complex and prone to mistakes in large settings. Expertise is often required for manual cluster administration [17, 18]. Might lead to irregularities and difficulties in sustaining infrastructure as it grows.
- **Advantages of using terraform for kubernetes cluster management on AWS:** There are benefits to using Terraform to manage Kubernetes clusters on AWS. Users may define the infrastructure using Terraform's declarative technique, to start. EKS clusters are included in the code. This ensures that infrastructure changes are tracked and replicated while streamlining version control, auditing, and collaboration procedures [18]. Additionally, Terraform's extensive module library, such as the AWS EKS module, simplifies Kubernetes cluster setup by providing templates that adhere to industry standards. The ease with which Terraform may be linked with AWS services is another important advantage [18]. To guarantee that the Kubernetes cluster is not only installed successfully but also seamlessly connects to the AWS environment for enhanced security and scalability, it can manage IAM roles, VPC settings, and auto scaling groups, for instance [11].

3. Methodology

3.1. Setting up terraform for AWS

- **Installation and configuration of terraform:** Installing Terraform is the first step in using it for Kubernetes cluster management on AWS. The Terraform binary is first downloaded from the official Hashi Corp website, its integrity is checked, and it is added to the system's PATH [17, 19]. Following installation, the terraform unit command creates and initializes a functioning directory for Terraform scripts by downloading required provider plugins and configuring the backend for state file storage [19].
- **Setting up AWS credentials and configuring the AWS provider in terraform:** Configuring AWS credentials is required in order to allow Terraform to communicate with AWS [19, 20]. This entails setting up the AWS CLI using these credentials and establishing an IAM user with the necessary rights. Terraform utilizes a configuration file that contains the credentials for authentication. To provide safe and effective administration of AWS resources, the Terraform configuration file specifies the AWS provider with the relevant region and credentials profile.

3.2. Provisioning Kubernetes Clusters

- **Using the AWS EKS Module to Define the Desired State of the Cluster:** Kubernetes cluster deployment is made easier by the AWS EKS module [20, 21]. This module offers a reusable framework for building EKS clusters and summarizes recommended practices. The Terraform configuration file specifies the cluster's intended state, which includes the availability zones, VPC CIDR block, and Kubernetes version [22]. This configuration guarantees that the cluster is built in compliance with the standards.
- **Defining Subnets, Node Groups, and IAM Roles:** Setting up the Kubernetes cluster requires defining subnets, node groups, and IAM roles.

IAM roles provide safe access to AWS resources, node groups specify which instances will execute Kubernetes workloads, and subnets supply the required networking infrastructure [22]. The setup includes mapping IAM roles for user access and permissions and defining the quantity and kind of instances for node groups.

- **Execution of Terraform Commands to Provision Resources:** Terraform commands are used to start the provisioning process when the configuration has been established. Terraform's it command downloads the required provider plugins and initializes the configuration directory [23]. The configuration is then applied and the resources are created using the terraform apply command. This command asks for approval before moving on and shows a plan of the changes that need to be made [23]. Terraform provides the EKS cluster and related resources after confirmation, giving real-time updates and the finished state.

3.3. Managing Kubernetes resources with terraform

- **Configuring the Kubernetes provider in terraform:** Terraform may be used to manage Kubernetes resources after the EKS cluster is up and running [21]. This entails setting up Terraform's Kubernetes provider to communicate with the Kubernetes API. Terraform can handle resources inside the cluster thanks to the provider configuration, which contains information like the cluster end point, CA certificate, and authorization token [18].
- **Managing Kubernetes resources like pods, services, and deployments:** Kubernetes resources like pods, services, and deployments are defined and managed using Terraform scripts. An application's deployment, for example, may be specified with particular metadata, selectors, container specs, and replica count. Terraform oversees these resources' lifespan, making sure they are set up and maintained in accordance with the configuration guidelines [18].

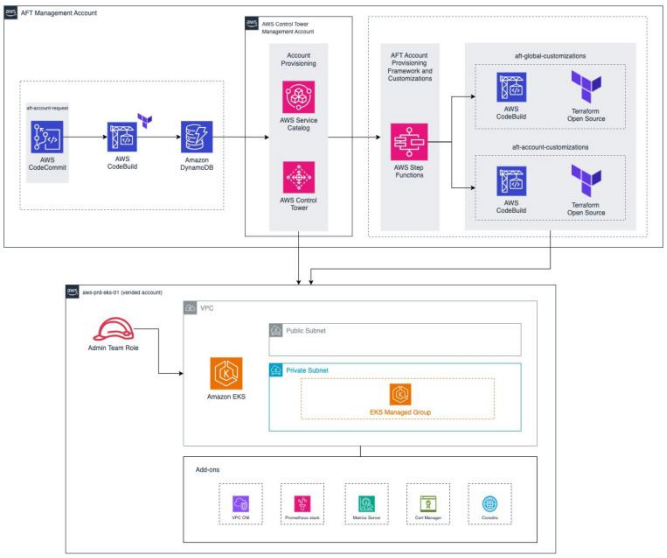

Fig. 3 An illustration of a Terraform process for AWS Kubernetes cluster setup and management. [19]

To sum up, there are comprehensive procedures for installing Terraform, configuring AWS, allocating an EKS cluster, and using Terraform scripts to manage Kubernetes resources when using Terraform for Kubernetes cluster management on AWS [19, 20]. This approach improves cloud-native application management's automation, consistency, and scalability.

4. Result

4.1. Provisioning Efficiency

- **Assessment of Terraform's ability to deploy Kubernetes clusters quickly and efficiently:** The time and resources needed to build up Kubernetes clusters using Terraform are measured in order to assess provisioning efficiency [20, 21]. Terraform's declarative methodology enables the specification of the intended infrastructure state, which it then compares to the current situation [21]. Because Terraform uses the configuration files to automate the construction, update, and deletion of infrastructure components, this procedure is very efficient. Terraform's ability to parallelize resource generation where dependencies permit

drastically speeds up provisioning and lowers the total deployment time.

- **Comparison with Traditional Methods of Cluster Management:** Manual setup and scripting are common components of traditional Kubernetes cluster management techniques. This method is prone to human mistake and might be time-consuming. Terraform, on the other hand, offers a more automated and efficient procedure [22]. Terraform makes ensuring that infrastructure configurations are repeatable and consistent by using Infrastructure as Code (IaC). Terraform is a better option for quickly and effectively establishing Kubernetes clusters because it offers scalability and repeatability that older techniques do not.

4.2. Automation and Scalability

- **Benefits of automation in cluster management using terraform:** Terraform-based cluster management automation has many advantages, such as less human intervention, fewer mistakes, and consistent environments. Because Terraform scripts are reusable and version-controlled, teams can effectively and cooperatively handle infrastructure changes [23]. Terraform's automation features also enable scaling operations, enabling resources to be dynamically adjusted in response to demand. Without requiring human reconfiguration, this automatic scaling guarantees that the cluster can manage fluctuating workloads.
- **Evaluation of performance enhancements and scalability in Kubernetes cluster management:** Terraform is excellent at scalability, which is a crucial component of Kubernetes cluster management. Scalable administration of big, complex systems is made easier by Terraform's modularity and use of IaC concepts [23]. As Terraform optimizes the infrastructure according to the specified settings, performance gains are achieved via the effective distribution and administration of resources. Scaling processes may be automated to guarantee optimum resource use, improving cost and performance.

4.3. Security and Compliance

- **Evaluation of Terraform-implemented security best practices:** Terraform's configuration files enable the application of security best practices. Encrypting data both in transit and at rest, establishing secure access controls, and making sure organizational standards are followed are some examples of these measures [23]. Terraform lowers the possibility of misconfigurations and vulnerabilities by ensuring that all infrastructure components follow the designated security requirements via the codification of security settings.
- **Assessment of Kubernetes Cluster Compliance Management on AWS:** For businesses in regulated sectors, compliance management is essential. By providing an auditable and transparent record of infrastructure modifications, Terraform facilitates compliance [24]. The infrastructure setup is documented in the configuration files, which may be examined and inspected to make sure it complies with rules and industry standards. Furthermore, Terraform's interface with AWS services makes it possible to use AWS compliance tools and capabilities, which improves the capacity to efficiently manage compliance.

In conclusion, there are notable gains in efficiency, automation, scalability, security, and compliance when using Terraform to supply and manage Kubernetes clusters on AWS. Because of these improvements, Terraform is now a very useful tool for managing contemporary infrastructure, providing a reliable way to install and maintain Kubernetes clusters in a safe, repeatable, and consistent manner [21].

5. Conclusion and Future Scope

5.1. Summary of findings

- ✓ **Terraform's benefits for managing Kubernetes clusters on AWS:** There are many advantages to using Terraform to manage Kubernetes clusters on AWS. By allowing users to declare the intended state of resources, Terraform's Infrastructure as Code (IaC) technique facilitates the management of infrastructures [23]. Terraform then aligns the state with the specified state. This approach guarantees consistency and uniformity and encourages teamwork. Additionally, Terraform uses the cloud platform's characteristics to seamlessly interact with AWS services, enhancing infrastructure management [21].

- **Efficiency:** Compared to doing it by hand, Terraform simplifies infrastructure configuration, saving a significant amount of time and effort [21, 23]. This mechanism ensures that resources are effectively generated, modified, and deleted in accordance with the designated settings.
- **Automation:** Beyond infrastructure setup, Terraform's automation tools also cover operations like resource updates and scale adjustments. This guarantees that the infrastructure remains as intended, minimizes mistakes, and lessens the need for repair [23].
- **Scalability:** Terraform makes managing infrastructure at scale simple with its architecture and Infrastructure as Code (IaC) concepts. Businesses may swiftly modify resources to satisfy changing demands, ensuring efficiency and economy.
- **Security:** Terraform guarantees that all components of the infrastructure adhere to regulations and industry standards by enabling security best practices to be included into code [24]. This increases security measures and reduces the likelihood of misconfigurations.

5.2. Future trends

- ✓ **Possible advancements in the combination of Terraform and Kubernetes:** It is anticipated that Kubernetes and Terraform's interaction will get even more robust and smooth as they both develop further. Future advancements may include improved multi-cloud environment compatibility, which would enable enterprises to centrally manage Kubernetes clusters across several cloud providers. Furthermore, [25] new modules and providers added to Terraform's ecosystem will probably provide even more reliable Kubernetes management options.

Prospects for future research and areas for development in Kubernetes administration and IaC tools:

- **Enhanced automation and AI integration:** Future studies may examine how IaC technologies like Terraform can be integrated with machine learning and artificial intelligence. Predictive scaling, more intelligent automation, and improved resource management might result from this [14, 16].
- **Security and compliance automation:** Tools that can automatically enforce standards and guarantee compliance are becoming more and more necessary as security and compliance requirements rise. Advanced features that make compliance management easier might be developed as a result of this research [18].
- **User experience and collaboration:** It will be essential to improve collaborative capabilities and the IaC tools' user experience. Future improvements may include enhanced version control systems, greater collaborative tools, and more user-friendly interfaces [18].
- **Performance optimization:** It will be crucial to continue investigating how to optimize the performance of Kubernetes clusters and the supporting infrastructure [18]. This might include creating fresh methods for load balance, latency reduction, and resource allocation.

6. Conclusion

Many of the issues with contemporary DevOps processes may be effectively resolved by integrating Terraform and AWS services to optimize CI/CD pipelines. Organizations may lower manual overhead, get rid of human mistake, and increase the consistency and dependability of their systems by automating infrastructure provisioning, deployment, and scaling. In addition to efficiently eliminating software development cycle obstacles, the MPME methodology streamlines and automates the complex process of establishing and overseeing many environments in a self-managed private DevOps infrastructure.

We included a project-naming component to make the MPME technique easier to apply. This element is essential for facilitating the development of dynamic environments that can accommodate many ongoing project activities. Project start, branching tactics, version control systems, repositories, and the CI/CD pipeline process are all seamlessly integrated into the DevOps workflow.

To sum up, Terraform has shown itself to be a useful tool for Kubernetes cluster management on AWS, providing notable gains in security, scalability, automation, and efficiency. The interaction between Terraform and Kubernetes is expected to get increasingly stronger as technology develops, spurring more innovation in infrastructure management. Future studies in this area might completely change how businesses implement and maintain their cloud-native apps, enabling them to adapt to the rapidly evolving technological environment.

REFERENCES

[1]. Yu, Y.; Silveira, H.; Sundaram, M. A microservice based reference architecture model in the context of enterprise architecture. In Proceedings of the 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Xi'an, China, 3–5 October 2016; pp. 1856–1860.

[2]. Villamizar, M.; Garcés, O.; Castro, H.; Verano, M.; Salamanca, L.; Casallas, R.; Gil, S. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In Proceedings of the 2015 10th Computing Colombian Conference (10CCC), Bogota, Colombia, 21–25 September 2015; pp. 583–590.

[3]. Kalske, M.; Mäkitalo, N.; Mikkonen, T. Challenges When Moving from Monolith to Microservice Architecture. In Current Trends in Web Engineering. ICWE 2017; Garrigós, I., Wimmer, M., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2018; Volume 10544.

[4]. Bernstein, D. Containers and Cloud: From LXC to Docker to Kubernetes. IEEE Cloud Comput. 2014, 1, 81–84.

[5]. Wan, X.; Guan, X.; Wang, T.; Bai, G.; Choi, B.-Y. Application deployment using Microservice and Docker containers: Framework and optimization. J. Netw. Comput. Appl. 2018, 119, 97–109.

[6]. Armenise, V. Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery. In Proceedings of the 2015 IEEE/ACM 3rd International Workshop on Release Engineering, Florence, Italy, 19 May 2015; pp. 24–27.

[7]. Bass, L.; Clements, P.; Kazman, R. Software Architecture in Practice, 3rd ed.; Addison-Wesley: Westford, MA, USA, 2012.

[8]. Saito, H.; Lee, H.-C.; Wu, C.-Y. DevOps with Kubernetes; Packt Publishing: Birmingham, UK, 2017; ISBN 978-1-78839-664-6.

[9]. Jenkins, D.; Arnaud, J.; Thompson, S.; Yau, M.; Wright, J. Version Control and Patch Management of Protection and Automation Systems. In Proceedings of the 12th IET

International Conference on Developments in Power System Protection, Copenhagen, Denmark, 31 March–3 April 2014.

[10]. Chandrasekara, C.; Herath, P. Branching with Azure Git Repos. In Hands-On Azure Repos; Apress: Berkeley, CA, USA, 2020.

[11]. Chen, L. Microservices: Architecting for Continuous Delivery and DevOps. In Proceedings of the 2018 IEEE International Conference on Software Architecture (ICSA), Seattle, WA, USA, 30 April–4 May 2018; pp. 39–397.

[12]. Abrar Mohammad Mowad, Hamed Fawareh, Mohammad A. Hassan, Effect of Using Continuous Integration (CI) and Continuous Delivery (CD) Deployment in DevOps to reduce the Gap between Developer and Operation, International Arab Conference on Information Technology (ACIT),22-24 November 2022.

[13]. Gowtham Mulpuri, CI/CD Pipeline Optimization for Accelerated Development Cycles,European Journal of Advances in Engineering and Technology, 2023.

[14]. A. Rahman, R. Mahdavi-hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," Information and Software Technology, vol. 108, 12 2018.

[15]. D. K. Sharma, "Terraform vs ansible: Key differences between devops tools," Mar 2024.

[16]. K. MORRIS, Infrastructure as code: Dynamic Systems for the cloud age. O'REILLY MEDIA, 2021. ISBN 9781098114671.

[17]. R. Alif and L. Munggaran, "Implementation of gitops in containerized infrastructure," Rabit: Jurnal Teknologi dan Sistem Informasi Univrab, vol. 9, no. 1, pp. 154–161, Feb. 2024.

[18]. Ekanayaka E, Thathsarani J, Karunanayaka D, Kuruwitaarachchi N, Skandakumar N. Enhancing devops infrastructure for efficient management of microservice applications. 2023 IEEE International Conference on e-Business Engineering (ICEBE) 2023; 63-68.

[19]. Sheikh S, Suganya G, Premalatha M. Automated resource management on AWS cloud platform. Proceedings of 6th International Conference on Big Data and Cloud Computing Challenges 2019.

[20]. Sindhu G, N. Mtech, and R. Pavithra D. Deploying a Kubernetes Cluster with Kubernetes Operation (kops) on AWS Cloud: Experiments and Lessons Learned. Int J Engineering Advanced Technology 2020.

[21]. Campbell B. Terraform In-Depth. The definitive guide of AWS infrastructure automation 2019; 123-203.

[22]. Sahana B, Kumaraswamy T, Nachiketh RG, Navadeep S, Noronha J. Weight based load balancing in Kubernetes using AWS. 2023 International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT) 2023; 629-634.

[23]. Bailuguttu S, Chavan A, Pal O, Sannakavalappa K, Chakrabarti D. Comparing performance of bastion host on cloud using Amazon web services vs terraform. Indonesian J Electrical Engineering Computer Science 2023.

[24]. Ganeshan M, Malathi S. Building and deploying a static application using Jenkins and Docker in AWS, Int J Trend in Scientific Research and Development 2020.

[25]. Haragi LD, Mahith S, Sahana B. Infrastructure Optimization in Kubernetes Cluster. J University of Shanghai for Science and Technology 2021.