International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 06 | June - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Continuous Integration for New Service Deployment and Service Validation Script for Vault

Pritish Raj
Department of ISE
R. V. College of Engineering®
Bengaluru, India
pritishraj11013@gmail.com

Dr. Kavitha S.N.
Department of ISE
R. V. College of Engineering®
Bengaluru, India
kavithasn@rvce.edu.in

***Abstract*—Modern cloud-native applications demand robust security measures to safeguard sensitive data such as passwords, API keys, and encryption keys. Managing these secrets securely within Kubernetes clusters presents a significant challenge. In response, this project proposes a comprehensive solution leveraging HashiCorp Vault, Kubernetes, and Docker to enhance secret management and strengthen overall security posture.HashiCorp Vault serves as a centralized secrets management tool, providing encryption, access control, and auditing functionalities. By integrating Vault with Kubernetes, secrets can be dynamically generated, securely stored, and automatically injected into application pods at runtime. This approach reduces the exposure of sensitive information within containerized environments and mitigates the risk of unauthorized access.**

The project architecture involves deploying Vault within the Kubernetes cluster, utilizing Docker containers for seamless encapsulation and portability. Kubernetes' native integrations with Vault, such as the Kubernetes Auth method and the Vault Agent Injector, streamline the authentication and authorization processes, ensuring secure communication between applications and Vault. The project involves deploying Vault in Kubernetes for secrets management, ensuring High Availability. It focuses on generating, storing, and managing secrets securely, leveraging Vault's dynamic secrets engine for automatic rotation. Integration with Kubernetes employs authentication methods like Service Accounts and RBAC for granular access control.

Dockerization ensures application consistency and portability, with Vault Agent containers enabling seamless secret injection. Security best practices, including least privilege access and encryption, are prioritized, along with regular auditing and monitoring. Overall, the project aims to establish a robust secrets management solution within Kubernetes while emphasizing resilience, security, and compliance in handling sensitive information.

***Index Terms*—Docker, DevOps, CI/CD, Automation, Secrets, Kubernetes, Vault, Security**

I. Introduction

The burgeoning world of microservices demands a multi-pronged approach to application security. While Docker containers streamline application packaging and portability, Kubernetes orchestrates containerized deployments at scale. Yet, a crucial challenge persists: securing the storage, access, and management of sensitive data within these dynamic environments. This project tackles this challenge head-on by forging a multi-layered security architecture that integrates HashiCorp Vault, Kubernetes, and Docker.

HashiCorp Vault serves as the fortified vault in this security landscape. It acts as a centralized command center for all secrets, safeguarding sensitive information like API keys, database credentials, and other classified data. Vault enforces strict access control through granular role-based policies, akin to a sophisticated security guard system. This centralized approach minimizes the risk of secrets being scattered across various applications or configurations, akin to leaving valuables hidden in unsecured corners.

Kubernetes, the container orchestration platform, serves as the deployment engine for the microservices, akin to a meticulously planned city layout. Integrating Kubernetes with Vault allows applications to securely access the secrets they require, similar to granting residents access to specific utilities like water and electricity. This integration enables dynamic secret injection, where secrets are automatically injected into application containers as environment variables at runtime. This eliminates the need to embed secrets directly in application code, similar to having secure, pre-configured utility connections in each building, enhancing both security and portability.

Docker containers act as the building blocks for the microservices, akin to individual homes within the city. Leveraging Docker images ensures consistent application environments across development, testing, and production stages. The integration with Vault empowers applications running within these containers to securely access the secrets they need to function effectively, similar to residents having secure access to utilities within their homes.

This project aspires to achieve the following key objectives:

- Centralized Secret Management: Establish Vault as the central repository for all sensitive information, eliminating sprawl and simplifying management – akin to having a single, secure vault for all valuables in the city.
- Enhanced Security: Implement robust access control through Vault policies, ensuring that only authorized applications can access specific secrets, similar to issuing personalized access cards to residents for specific areas within the city.
- Automated Secret Rotation: Utilize Vault's capabilities for automated secret rotation, minimizing the window of vulnerability for compromised credentials – akin to

regularly changing the city's security codes and access keys.

- Improved Application Security: Eliminate the need to embed secrets within application code, reducing the attack surface and enhancing code portability – akin to having secure, central utility connections instead of storing flammable gas canisters inside each home.
- Streamlined Deployment: Leverage Kubernetes for efficient containerized application deployments and integrate with Vault for secure secret injection – akin to a pre-planned, efficient city construction process with secure and pre-configured utility connections.

By successfully integrating these technologies, this project will establish a secure and scalable foundation for the microservice architecture. The project will deliver a robust environment that fosters innovation while ensuring the confidentiality and integrity of sensitive data within our containerized applications.

II. Literature Review

The evolving landscape of information technology has necessitated robust security measures to protect data integrity and privacy. This literature review delves into the key areas of key management, policy-based access control, and storage security, providing insights into recent advancements and methodologies employed to enhance security in various environments. Key management is a critical aspect of cryptographic systems, as it involves the generation, distribution, storage, and disposal of cryptographic keys. Effective key management ensures that keys are secure from unauthorized access while remaining accessible to authorized users.

Gebremichael et al. [1] focus on the challenges posed by IoT networks, which often have limited processing power and resources. They propose lightweight group key management techniques tailored to these constraints. These techniques aim to reduce the computational and memory overhead associated with key management while maintaining security standards. The proposed methods include efficient key distribution protocols and periodic key updates to mitigate the risk of key compromise.

J. Gustafsson et al. [2] emphasize the importance of safe secret management for small enterprises, which typically face limitations in funding, knowledge, personnel, and hardware. Their article compares various software-based key management technologies, highlighting those that facilitate the automatic and secure handling of secrets. The recommended practices include the use of centralized key management systems that automate key rotation and revocation, thereby reducing the risk of human error and improving overall security posture.

Focardi et al. [3] describe the use of keystores to ensure the integrity of shared keys within cryptographic mechanisms. Keystores provide a secure repository for storing cryptographic keys, offering protection against unauthorized access and tampering. The authors discuss various keystore implementations, including hardware-based and software-based solutions, and their respective strengths and weaknesses in maintaining key integrity.

Sven Plaga et al. [4] introduce Hardware Security Modules (HSMs) as a robust solution for the secure and efficient storage of cryptographic keys. HSMs are dedicated hardware devices designed to protect keys from physical and logical attacks. They offer enhanced security features, such as tamper-resistance and secure key generation, making them ideal for high-security environments. The authors highlight the role of HSMs in supporting various cryptographic operations, including encryption, decryption, and digital signing.

In contrast to hardware-based solutions, John Patrick McGregor et al. [5] propose virtual secure coprocessing to improve processor performance and secure key access. This approach leverages distributed computing techniques to maintain and secure keys, reducing the reliance on dedicated hardware modules. The virtual secure coprocessing model offers flexibility and scalability, enabling efficient key management across distributed systems. However, the authors note that this method can result in increased hardware demands and costs, particularly for high-performance applications.

M.V. Srinath et al. [6] provide a comprehensive overview of key management approaches for secure multicast environments. Multicast communication, which involves the transmission of data to multiple recipients, presents unique security challenges. The authors discuss various key management schemes, including group key distribution protocols and key agreement techniques, designed to ensure secure and efficient multicast communication. These approaches aim to minimize the overhead associated with key distribution and update processes while maintaining robust security.

The advent of cloud computing has introduced new opportunities and challenges for key management. Cloud environments offer scalable and cost-effective solutions for storing and managing cryptographic keys, but they also raise concerns about data security and privacy.

Amar Buchade et al. [7] analyze symmetric key cryptography techniques and key management in cloud environments. They evaluate the performance and security of various symmetric key algorithms, such as AES and DES, in the context of cloud-based applications. The authors highlight the importance of efficient key management practices, including secure key generation, storage, and distribution, to mitigate the risks associated with cloud-based data storage and processing.

Dharam Raj Kumar et al. [9] conducted an experimental assessment of key generation techniques in cloud data storage over the internet. Their study identifies several issues with existing cryptographic methods used for data storage and retrieval, such as latency, key management overhead, and vulnerability to attacks. The authors propose improvements to key generation and management processes, including the use of distributed key management systems and advanced cryptographic algorithms, to enhance the security and performance of cloud-based storage solutions.

Secure storage of data is a fundamental requirement for

maintaining the confidentiality, integrity, and availability of information. This section explores various storage security measures, including policy-based access control and attribute-based encryption, which provide fine-grained access control and protect sensitive data.

Policy-based access control mechanisms allow organizations to define and enforce access policies that govern who can access specific keys and data sets. These mechanisms provide flexibility and scalability, enabling organizations to manage access rights based on various criteria, such as user roles, attributes, and environmental conditions.

S. Rajeswari et al. [10] discuss the challenges and solutions associated with storage security in cloud-based services. Cloud storage requires exposing sensitive data to a cloud provider, which may not be appropriate for legal and regulatory purposes. The authors explore various encryption techniques and access control mechanisms that can be employed to protect data stored in the cloud. They emphasize the need for robust key management practices and secure data transmission protocols to mitigate the risks associated with cloud storage.

Nesrine Kaaniche et al. [11] propose a multi-level access control approach using attribute-based encryption (ABE) algorithms. ABE enables secure data exchange and decentralized access control by encrypting data based on user attributes. This approach allows organizations to implement fine-grained access control policies, ensuring that only authorized users can access specific data sets. The authors demonstrate how ABE can provide multiple degrees of protection, enhancing the security and privacy of sensitive information.

The literature on key management, policy-based access control, and storage security measures highlights the importance of implementing robust security practices to protect data in various environments. Lightweight key management techniques are essential for resource-constrained IoT networks, while centralized and automated key management systems are beneficial for small enterprises. Hardware Security Modules (HSMs) and virtual secure coprocessing offer secure and efficient key storage solutions, albeit with different trade-offs in terms of cost and complexity.Cloud-based key management introduces new challenges, necessitating efficient and secure key generation, storage, and distribution practices.

Policy-based access control and attribute-based encryption provide flexible and scalable mechanisms for managing access to sensitive data, ensuring that only authorized users can access specific information. Future research should focus on addressing the limitations of existing key management and storage security solutions, exploring new cryptographic algorithms and distributed computing techniques to enhance security and performance. Additionally, the integration of emerging technologies, such as blockchain and artificial intelligence, could provide innovative approaches to key management and data protection in an increasingly interconnected world.

III. BACKGROUND AND OBJECTIVES

A. Problem Statement

Developing validation and clean-up scripts for existing services entails a methodical approach. Initially, it's crucial to thoroughly assess the current service landscape, understanding functionalities, dependencies, and existing issues. Subsequently, clear validation criteria must be delineated, encompassing input verification, API responses, and data integrity checks. Leveraging scripting languages like Python or Shell, automation scripts can be crafted to execute these validations seamlessly, integrated into regular workflows. Implementing robust error handling and reporting mechanisms ensures graceful failure management and comprehensive insight into validation outcomes. Scheduled execution of these scripts, alongside the development of clean-up counterparts to rectify identified issues or perform routine maintenance tasks, maintains service health and efficiency.

B. Purpose

The project integrating HashiCorp Vault, Kubernetes, and Docker aims to create a secure, scalable, and efficient secrets management solution for cloud-native environments. Vault offers robust encryption, access control, and auditing to manage sensitive data such as passwords and API keys. Kubernetes orchestrates the deployment of containerized applications, while Docker ensures consistency and portability across environments. This integration allows applications to securely access secrets at runtime, enhancing security by encrypting data at rest and in transit with fine-grained access controls. The project seeks to improve security posture, reduce the risk of data breaches, and ensure regulatory compliance. By centralizing secrets management and automating processes like dynamic secrets generation and rotation, it empowers organizations to confidently build and operate cloud-native applications, ensuring trust and reliability in their digital infrastructure.

C. Motivation

The project addresses the complex challenges of secrets management and security in cloud-native environments. With micro-services and containerization, organizations face security threats and compliance requirements for managing sensitive data. By leveraging Vault's robust secrets management, Kubernetes' orchestration, and Docker's containerization, the project enhances data security, confidentiality, and integrity through encryption, access controls, and auditing. It streamlines operations, improves agility, and reduces secrets management overhead. Ultimately, it empowers organizations to confidently embrace cloud-native technologies, knowing their sensitive data is protected and compliant with industry regulations, fostering innovation and reliability in their digital infrastructure.

D. Scope and Relevance

The project covers the entire secrets management lifecycle, including encryption, access control, dynamic secrets generation, and auditing. It aims to secure sensitive data across distributed environments by integrating Vault with Kubernetes and Docker. This project addresses critical challenges in securing and managing secrets in cloud-native ecosystems, particularly with the rise of microservices and containerized deployments. By providing a centralized and scalable solution, it ensures robust security, regulatory compliance, and streamlined operations. Its relevance extends to the broader tech community, advancing best practices in cloud-native security and infrastructure management, enhancing organizational security, and operational efficiency.

IV. METHODOLOGY

1. Pod Restart Simulation:
   - Simulate pod restarts to evaluate the resilience of secret injection mechanisms.
   - Monitor the behavior of application pods after restart to ensure seamless secret retrieval from Vault.
2. Manual Testing:
   - Conduct manual testing post-pod restart to validate application functionality.
   - Verify that secrets are injected correctly and application behavior remains consistent.
   - Test various scenarios, including different secret types and access patterns.
3. End-to-End Testing for Existing Features:
   - Perform comprehensive end-to-end testing to validate all existing features.
   - Ensure that the integrated environment meets functional and security requirements.
   - Test scenarios covering secret generation, rotation, and revocation.
   - Validate access control policies and compliance with regulatory standards.
4. Automated Workflow Integration:

Configure GitHub Actions to trigger automated workflows upon new service addition or secret updates to existing services.

   - Define a workflow YAML file to specify the actions to be executed, such as build, test, and deployment steps.
   - Utilize GitHub's event-based triggers, such as push events, to initiate the workflow when changes are pushed to the repository.
5. Continuous Integration Pipeline Setup:
   - Establish a CI pipeline to automate the build and testing process for the new service.
   - Use tools like Docker or Kubernetes for containerized builds to ensure consistency across environments.
   - Implement unit tests, integration tests, and code quality checks to validate the integrity of the codebase before deployment.

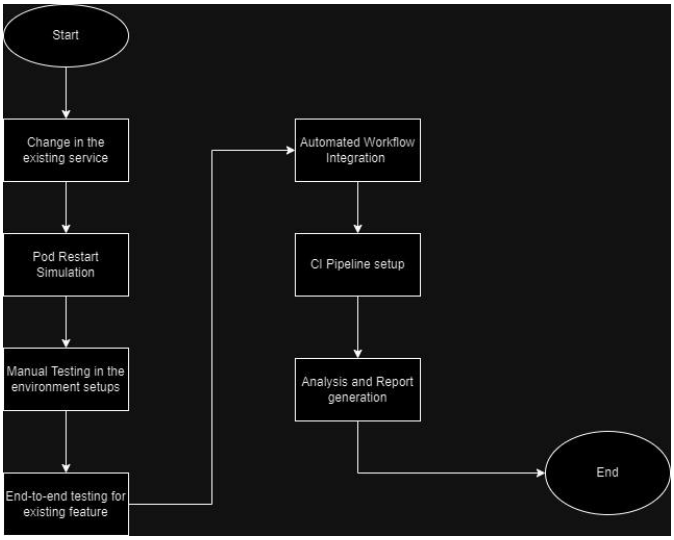

Fig. 1. Methodology

6. Analysis and Reporting:
   - Analyze test results to identify any issues or discrepancies.
   - Document findings and provide recommendations for improvements.
   - Report on the overall resilience, functionality, and security of the integrated environment.
   - Incorporate feedback into continuous improvement efforts for ongoing enhancement.

V. IMPLEMENTATION

The implementation involves deploying applications within Kubernetes clusters orchestrated by Docker. Vault manages secrets securely, integrating with Kubernetes for authentication and secret injection. Applications are containerized using Docker, with Vault Agent containers facilitating seamless secret retrieval. Security measures like encryption and access control are enforced, ensuring robust secrets management within the dynamic and scalable Kubernetes environment.

A. Implementation Requirements

Implementation requirements for a project involving Kubernetes, Docker, and Vault include:

1) Infrastructure Setup: Provisioning servers or cloud instances to host Kubernetes clusters and Vault servers.
2) Kubernetes Configuration: Configuring Kubernetes clusters with appropriate networking, storage, and security settings.
3) Docker Installation: Installing Docker Engine on Kubernetes nodes to containerize applications.
4) Vault Deployment: Deploying Vault servers within the Kubernetes environment or as standalone instances.
5) Integration Setup: Configuring Kubernetes authentication methods for Vault integration.
6) Secrets Management: Defining Vault policies and secrets engines to manage secrets securely.

7) Application Containerization: Creating Docker images for applications, ensuring compatibility with Vault secret injection.
8) Monitoring and Logging: Setting up monitoring and logging solutions to track system health and security events.

The project's implementation entails setting up Kubernetes clusters and Docker environments to orchestrate and containerize applications. Vault manages secrets securely, integrating with Kubernetes for authentication and secret injection. Infrastructure provisioning, configuration, and deployment automation are pivotal, ensuring seamless integration. Security measures like encryption and access controls are enforced. Thorough testing, monitoring, and documentation complement the process to guarantee robustness and reliability of the solution.

VI. Architecture Diagram

The diagram shows how HashiCorp Vault secures access to secrets. Users authenticate with an IdP and get a Vault token. This token is used to access secrets stored in a Vault secret engine. The diagram also shows how different SSH roles can be created with varying permissions.

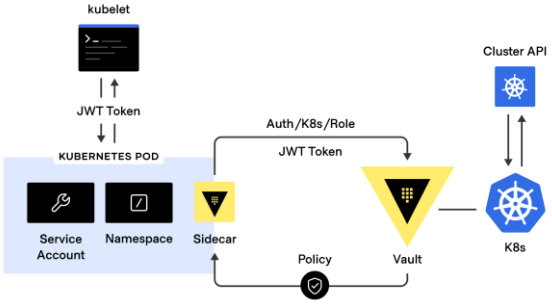

Fig. 2. Vault Architecture

1) A service account is created in Kubernetes. A service account is a special type of account used to identify a pod or service within a Kubernetes cluster. Service accounts are used to grant pods access to resources in the cluster.
2) A JWT token is issued to the service account. A JWT, or JSON Web Token, is a cryptographic token that contains claims about the identity of the service account. This token is used by the pod to authenticate with Vault.
3) The pod uses the JWT token to authenticate with Vault. When the pod needs to access a secret from Vault, it presents the JWT token to Vault for authentication.
4) Vault validates the JWT token and grants access to the requested secret if the token is valid. If the token is valid, Vault will grant the pod access to the requested secret.
5) The pod uses the secret to access the resource. The pod can then use the secret to access the resource that it needs, such as a database or an API.

This process helps to ensure that only authorized pods have access to secrets in Vault.

VII. Authentication of Vault

Vault utilizes an authentication method to confirm your identity. User present credentials (username/password, token, etc.) through a chosen auth backend (LDAP, GitHub, etc.). Vault validates these against the backend and issues a short-lived JWT token upon successful authentication. This token acts as your key, granting access to authorized secrets within Vault.

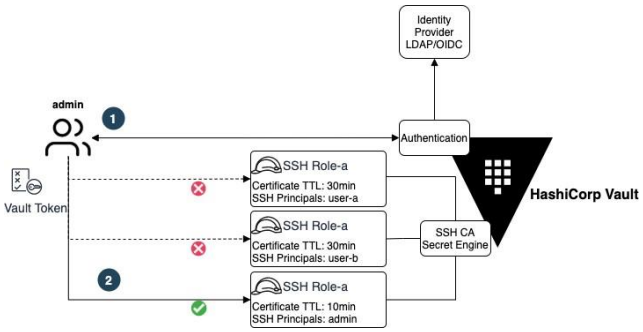

Fig. 3. Authenticating Vault

1) Identity Provider: The top left corner of the image shows an Identity Provider labeled "LDAP/OIDC". An Identity Provider (IdP) is a trusted authority that verifies the identity of users or services. In this case, LDAP or OpenID Connect (OIDC) are being used as the Identity Provider.
2) Admin: Below the IdP is a box labeled "admin". This likely represents an administrative account that has the highest level of privilege within the system.
3) Authentication: An arrow points from the Identity Provider to a box labeled "Authentication". This indicates that the user or service authenticates with the IdP.
4) SSH Role-a: Below the Authentication box is a box labeled "SSH Role-a". An SSH role defines the permissions that are associated with an SSH identity.
5) Vault Token: An arrow points from the Authentication box to a box labeled "Vault Token". This indicates that once a user or service is authenticated, they are issued a Vault Token. A Vault token is a credential that is used to access resources within HashiCorp Vault.
6) SSH CA: To the right of the Vault Token box is a box labeled "SSH CA". An SSH CA, or Certificate Authority, is a trusted entity that signs SSH certificates. These certificates are used to authenticate SSH connections.
7) Secret Engine: An arrow points from the Vault Token box to a box labeled "Secret Engine". A secret engine is a part of HashiCorp Vault that stores specific types of secrets. In this case, the secret engine likely stores SSH certificates.

8) SSH Role-a with details: The bottom left corner of the image shows details about SSH Role-a. The details include:
9) Certificate TTL: 30min - This means that SSH certificates issued for this role are valid for 30 minutes.
10) SSH Principals: user-a - This means that user-a is authorized to use SSH certificates issued for this role.
11) Another SSH Role-a with details: The bottom right corner of the image shows another SSH Role-a with details that differ slightly from the first SSH Role-a. The details include:
12) Certificate TTL: 10min - This means that SSH certificates issued for this role are valid for 10 minutes, which is less than the first SSH Role-a.
13) SSH Principals: admin - This indicates that the admin user is authorized to use SSH certificates issued for this role.

This diagram shows how HashiCorp Vault can be used to securely manage SSH access. Users first authenticate with an Identity Provider. Once authenticated, they are issued a Vault Token. This token can be used to access SSH certificates stored in a Vault secret engine. The SSH certificates can then be used to authenticate SSH connections. The diagram also shows that different SSH roles can be created with different permissions. For example, SSH Role-a for user-a has a longer certificate TTL than SSH Role-a for the admin user.

VIII. Results

The project successfully established a robust Continuous Integration (CI) pipeline for new service deployment and implemented effective service validation scripts for HashiCorp Vault. The CI pipeline, built using GitLab CI, automates the entire deployment process, including code build, testing, and deployment within Docker containers orchestrated by Kubernetes. Automated tests, including unit, integration, and end-to-end tests, ensure that new services function correctly and securely before deployment.

Additionally, service validation scripts were created to verify the interaction between new services and Vault. These scripts ensure that services correctly retrieve, use, and manage secrets, adhering to defined access controls and encryption policies. The validation process includes dynamic secrets generation and rotation, ensuring that the services use up-to-date credentials. Compliance checks were also integrated to verify that deployments meet security and regulatory standards.

The implementation of the CI pipeline and validation scripts yielded significant improvements in deployment efficiency and security. By automating the deployment process, the project reduced the risk of human error and ensured consistent deployments across different environments. This automation also accelerated the deployment cycle, allowing for more rapid iteration and innovation.

The service validation scripts for Vault provided an additional security layer, ensuring that new services interact with Vault as intended. This interaction includes correct retrieval and use of secrets, adherence to access controls, and proper encryption, which are critical for maintaining data integrity and confidentiality. Dynamic secrets management further enhanced security by ensuring services always use the most secure credentials.

Overall, the project demonstrated the value of integrating CI practices with robust secrets management, enhancing both the security posture and operational efficiency of the organization. It serves as a model for best practices in secure service deployment and management in cloud-native environments, contributing to the broader technology community by fostering improved security and operational standards.

IX. Conclusion

The project has been instrumental in ensuring the resilience, functionality, and security of the integrated environment. By implementing a comprehensive approach, encompassing various testing strategies and automation techniques, we have successfully validated the integrity of the system and its ability to handle different scenarios effectively.

The Pod Restart Simulation served as a crucial mechanism for evaluating the resilience of secret injection mechanisms. Through this simulation, we were able to assess how the application pods behaved after restarts, ensuring seamless secret retrieval from Vault. This step was pivotal in guaranteeing that the application could maintain its functionality even under challenging circumstances.

Manual testing further validated the correctness of secret injection and ensured consistent application behavior post-pod restart. By conducting tests across various scenarios, including different secret types and access patterns, we could confidently assert the robustness of the system in handling diverse situations.End-to-End Testing for Existing Features provided a comprehensive validation of all functionalities within the integrated environment. This step ensured that not only were secrets generated, rotated, and revoked correctly, but also that access control policies were enforced and compliance with regulatory standards was maintained. By scrutinizing every aspect of the system, we could guarantee its readiness for deployment in production environments.

The integration of Automated Workflow through GitHub Actions streamlined the development process significantly. By automating build, test, and deployment steps upon new service pushes, we ensured a consistent and efficient development workflow. Leveraging event-based triggers like push events enabled seamless integration with the development lifecycle, promoting agility and reliability in the software delivery process.

In conclusion, the project has not only validated the resilience, functionality, and security of the integrated environment but has also paved the way for ongoing enhancement and improvement. By adopting a holistic approach to testing and automation, we have created a robust foundation for the continued evolution and success of the software system.

X. Future Scope

The project's future presents several opportunities for further enhancement and expansion:

1) Advanced Kubernetes Integration: Explore advanced Kubernetes features such as StatefulSets, DaemonSets, and Custom Resource Definitions (CRDs) to optimize the deployment and management of applications within Kubernetes clusters. Implementing Kubernetes Operators for Vault could streamline secret management and automate common tasks, further enhancing the integration between Kubernetes and Vault.
2) Vault Integration Enhancements: Enhance the integration between Vault and Kubernetes by leveraging Vault Agent Injector for automatic injection of secrets into application pods. Implement dynamic secrets generation and leasing capabilities provided by Vault to enhance security and reduce manual overhead in managing secrets.
3) Docker Security Enhancements: Implement best practices for securing Docker containers, such as image vulnerability scanning, runtime security monitoring, and container signing. Explore the use of Docker Content Trust (DCT) to ensure the integrity and authenticity of container images throughout the deployment lifecycle.
4) Continuous Delivery Pipelines: Extend the Continuous Integration (CI) pipeline to include Continuous Delivery (CD) capabilities, enabling automated deployment of applications to Kubernetes clusters after successful testing. Implement blue-green or canary deployment strategies to minimize downtime and risk during application updates.

By exploring these future scopes, the project can further leverage the capabilities to enhance security, scalability, and automation in modern cloud-native environments. Additionally, it can adapt to evolving industry trends and requirements, ensuring the continued success and relevance of the integrated solution.

References

[1] Gebremichael Teklay, "Lightweight Cryptographic Group Key Management Protocols for the Internet of Things," presented at IEEE International Workshop of Security and Trust, Luxembourg, (2019).
[2] Jacob Gustafsson and Adam T rnkvist, "Secure handling of encryption keys for small businesses : A comparative study of key management systems," presented at International Cybersecurity Congress , Moscow, (2019).
[3] Riccardo Focardi, Francesco Palmarini, Marco Squarcina, Graham Steel and Mauro Tempesta, "Mind Your Keys? A Security Evaluation of Java Key-stores," presented at Network and Distributed System Security Symposium , California, (2018).
[4] Sven Plaga, Norbert Wiedermann, Gerhard Hansch, and Thomas Newe, "Secure your SSH Keys! Motivation and Practical Implementation of a HSM-based Approach Securing Private SSH-Keys," presented at 17th European Conference on Cyber Warfare and Security ECCWS, Norway, (2018). J
[5] ohn McGregor and Ruby Lee, "Protecting cryptographic keys and computations via virtual secure coprocessing," in SIGARCH Computer Architecture News, (2005).
[6] B. T. Geetha and M. V. Srinath, "A Study on Various Cryptographic Key Management and Distribution System in Secure Multicast Communications," in International Conference on Advances in Mobile Network Communication and Its Applications, (2012), pp. 64-69.
[7] Amar Buchade and Rajesh Ingle,"Key Management for Cloud DatA Storage: Methods and Comparisons," in International Conference on Advanced Computing and Communication Technologies, (2014), pp. 263-270.
[8] F. Mohamed, B. AlBelooshi, K. Salah, C. Y. Yeun and E. Damiani, "A Scattering Technique for Protecting Cryptographic Keys in the Cloud," presented at IEEE 2nd International Workshops on Foundations and Applications of Self Systems ,University of Arizona, (2017).
[9] Dharam Kumar and Jitendra Sheetlani, "Review of Key Management and Distribution Technique for Data Dynamics for Storage Security in Cloud Computing , " IOSR Journal of Computer Engineering, (2017), pp. 38-49.
[10] S. Rajeswari and R. Kalaiselvi , "Survey of data and storage security in cloud computing," in IEEE International Conference on Circuits and Systems (ICCS), (2017), pp. 76-81.
[11] Zakaria Igarramen, Ahmed Bentajer, and Mustapha Hedabou ," TPM Based Schema for Reinforcing Security in IBE's Key Manager, "presented at International Conference on Data and Model Engineering, Toulouse, (2019).
[12] Changji Wang and Jianfa Luo, "An Efficient Key-Policy Attribute-Based Encryption Scheme with Constant Ciphertext Length," presented at International Conference on Computational Intelligence and Security, Hong Kong, (2017).
[13] Nesrine Kaaniche and Maryline Laurent,"Attribute based Encryption for Multi-level Access Control Policies," in International Conference on Security and Cryptography, (2017), pp. 67-78.
[14] K Bhargavan, Richard Barnes, and Eric Rescorla,"TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS), "presented at INRIA, Paris, (2019).
[15] Sam Kim and David J. Wu, "Access Control Encryption for General Policies from Standard Assumptions," presented at International Conference on the Theory and Application of Cryptology , Brisbane, (2018).