Frontiers in Computer Science and Artificial Intelligence
ISSN: 2798-8048
DOI: 10.32996/fcsai
Journal Homepage: www.al-kindipublisher.com/index.php/fcsai

Secure Secret Platform Migration with Application Continuity in Cloud-Native Systems

Rishith Chinthamani

Walmart Global Tech, Bentonville, AR, USA

Corresponding Author: Rishith Chinthamani**, E-mail:** rishithchinthamani1@gmail.com

ABSTRACT

Secret-management technologies are essential for cloud-native apps to safeguard credentials, API keys, passwords, and access tokens. On the other hand, secrets in transit pose risks to services, integrity, and authorisation consistency. This paper presents a safe migration methodology for ensuring application continuity while transitioning from HashiCorp Vault KV v2 to AWS Secrets Manager. The framework includes a SecretProvider interface, a dual-provider fallback mechanism, an 11-state finite-state machine, SHA-256 integrity verification, role-based access control (RBAC), audit logging, and automatic rollback. The framework, which was built using Spring Boot, LocalStack, Keycloak, and Docker, was tested with an OpenH2C-calibrated workload of 7 RPS and security-preserving surrogates derived from SecretBench. Three separate 90-second runs were used to evaluate continuity-aware migration, typical stop-and-restart migration, and baseline operation. A mean migration duration of around 2.9 seconds and throughput exceeding 7.07 secrets/s were obtained by the suggested technique, along with 100% observed request success (1,074/1,074), zero observed container downtime, and a mean latency of 11.32 ms and a P95 latency of 26.82 ms. The average latency dropped from 257.58 ms to 11.32 ms, and the P95 latency dropped from 4,053.97 ms to 26.82 ms, compared with conventional migration. Eight failure-injection scenarios, 23 RBAC checks, and all sixty fingerprint checks were successful. Results show that migration and continuity are both better under these circumstances.

KEYWORDS

Cloud-native security, secret management, application continuity, HashiCorp Vault, AWS Secrets Manager, dual-provider architecture, SHA-256, role-based access control, automated rollback.

ARTICLE INFORMATION

ACCEPTED: 16 October 2026 **PUBLISHED:** 16 November 2026 **DOI:** 10.32996/fcsai.2026.5.12.1

1. Introduction

Sensitive credentials, such as passwords, API keys, access tokens, and digital certificates, are becoming more and more necessary for cloud-native applications to ensure secure communication and authentication across distant services [1][2]. Platforms for managing secrets, including AWS Secrets Manager and HashiCorp Vault, provide centralised ways to store, retrieve, and safeguard these credentials. However, due to cloud adoption, infrastructure modernisation, vendor changes, or changing security needs, organisations may need to migrate between secret-management platforms [3][4]. Data integrity, access-policy consistency, credential secrecy, and secure transfer across diverse environments are all made more difficult by such migration [5].

Applications and microservices must have continuous access to necessary credentials, making application continuity a crucial consideration during secret platform transfer [6]. Service disruptions, manual configuration adjustments, or brief discrepancies between the source and destination systems are all possible outcomes of conventional transfer techniques. These problems may result in operational outages, service degradation, and authentication difficulties [7][8]. The integration of secure secret transfer with continuous application availability throughout the migration lifecycle has received relatively little attention, despite the fact that current research examines cloud migration, cloud-native security, identity and access management, and fault-tolerant architectures [9].

To address these difficulties, this study provides a secure secret-platform migration architecture designed to maintain application continuity in cloud-native systems [10]. The framework adopts a dual-provider architecture, enabling access to the current secret provider and configuration and validation of the target platform. Role-based access control reduces unauthorized migration activity and SHA-256-based integrity checking identifies unintended changes to or corruption of secrets [11]. A migration state

Copyright: © 2026 the Author(s). This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) 4.0 license (https://creativecommons.org/licenses/by/4.0/). Published by Al-Kindi Centre for Research and Development, London, United Kingdom.

machine is a software component that coordinates migration stages, validates migrations, and automatically rolls back if issues arise. Operational traceability is also supported through audit logging and monitoring without recording sensitive secret values.

The proposed architecture is assessed using controlled workloads and test data that maintain security. Its performance is measured using application uptime, request latency, migration time, secret integrity, access control efficacy, and failure recovery performance. The primary objective is to create a reliable migration plan that minimizes service downtime, secures sensitive credentials and maintains continuous access to the applications during secret platform transfers. The study is part of an extensive solution to improve security, operational resiliency, and application continuity in existing cloud-based systems.

1.1 Motivation and Contribution of the paper

The importance of secret-platform transfer is growing as organisations upgrade their cloud infrastructure and adopt new security services. However, with traditional migration, application requests could fail, credentials could be exposed, and incompatibilities could arise between the source and destination platforms. Typically, current methods focus on access control and infrastructure modernization, but do not incorporate automatic recovery, integrity verification, and continued secret access as a single process. This study tackles these issues by creating a secure, continuity-aware migration architecture that minimises service downtime while ensuring authorisation, secret integrity, and operational traceability throughout platform migrations. The major contributions of this research are:

Provider abstraction: A common SecretProvider interface enables Vault to be migrated to AWS Secrets Manager without changing application business logic.

Application continuity: A dual-provider technique uses Vault as the primary provider and AWS Secrets Manager as a fallback when Vault fails, avoiding application-container restarts during migration.

Integrity verification: The SHA-256 fingerprint comparison ensures that the secrets transferred are valid before the providers are switched.

Controlled migration and recovery: An eleven-state finite-state machine allows for automated rollback if errors occur, enforces RBAC, and coordinates migration steps.

Experimental evaluation: Three migration scenarios are considered, taking into account request availability, latency, migration length, integrity checks, authorisation tests and failure-injection results.

1.2 Novelty of the paper

The novelty of this study is that, instead of seeing migration as a separate infrastructure function, application continuity is integrated directly into the secret-platform migration process. The design of the framework is consistent, and it includes features like primary-provider access with fallback, automatic rollback, an 11-step migration process, integrity checks based on SHA-256 and RBAC enforcement. The proposed solution continues to handle requests for applications during migration, unlike stopping and starting, which is the traditional approach. Also, it offers measurable evidence of application performance during migration and security validation, using OpenH2C-tuned workloads and security-preserving surrogates generated by SecretBench in the laboratory environment.

1.3 Organization of the Paper

This is how the rest of the paper is structured. The literature on cloud-native security, secret management, and application migration is reviewed in Section II. The suggested technique, which includes provider abstraction, dual-provider continuity, migration state management, integrity verification, rollback, datasets, and experimental setup, is presented in Section III. Section IV examines the experimental findings for request availability, latency, migration throughput, access control, and failure recovery. Section V closes the study by summarising the main results and suggesting future research areas.

2. Literature Review

Current studies concentrate on centralised secret management, encryption, access control, and secret rotation. Current methods prioritise application continuity, automation, safe migration, and zero downtime.

Yuta Nakamori and Emi Hayasaka (2026) create a strategy framework that integrates architectural transformation concepts, migration techniques, operational optimisation mechanisms, and governance-oriented deployment methods to modernise outdated corporate applications using cloud-based migration models. The report summarises current research on corporate transformation models, cloud-native architectures, security integration, adaptive planning techniques, and cloud migration frameworks [12].

Tuba Arif, Byunghyun Jo and Jong Hyuk Park (2025) explore several approaches to privacy and trust that aim to address the specific security needs of cloud-native systems. A comprehensive case study is also provided to show how security measures are implemented in a cloud-native architecture across many levels, such as application, network, infrastructure, security, and compliance [13].

Neha Surendranath, Sai Annamaiah Basava Raju and Ranita Ganguly (2025) create a structured approach to migrating applications to the cloud and updating them using serverless computing, microservices, and containers. The suggested paradigm outperforms conventional migration frameworks by facilitating cloud-native behaviours that improve system fault-tolerance, optimise

resources, and accelerate deployment schedules. The study illustrates the benefits and drawbacks of cloud-native DevOps by contrasting it with conventional techniques [14].

Chinua Achebe and Tosin Akinwole (2024) highlight how IAM, service mesh technologies, and automated policy enforcement contribute to the security of hybrid systems throughout migration stages. The suggested method incorporates ZTA principles into the process of modernising outdated applications. Simultaneously, cloud-native migration techniques like API-based integration, containerisation, and microservices decomposition are used to gradually revamp outdated systems while ensuring uninterrupted business operations [15].

Nascimento et al. (2024) investigate how containerised apps might be tailored to meet such needs throughout the transition to cloud-native orchestrated environments. A Proof of Concept (PoC) highlighted the technical challenges of transitioning to a Kubernetes environment on Azure. The findings of their research revealed the potential of Kubernetes in meeting requirements for availability and scalability while ensuring security when transferring containerised apps to cloud-native environments [16].

The goal of Theodoros Theodoropoulos et al. (2023) is to provide a thorough examination of the characteristics, features, and solutions connected with cloud-native security. It functions as a structured mapping study that links the essential elements to their corresponding characteristics and these traits to a variety of modern solutions. An additional component is the cataloguing of potential open-source technologies that might facilitate the execution of each investigated solution [10].

Although cloud migration, cloud-native security, containerisation, zero-trust architecture, and application modernisation have advanced significantly, the secure migration of secret management platforms while preserving uninterrupted application continuity has received little attention in the literature. Secret synchronisation, credential rotation, access-policy consistency, and safe secret transfer across diverse settings are unaddressed in current techniques, which mainly focus on infrastructure security, scalability, and deployment optimisation. In addition, rollback techniques, minimisation of downtime for applications, and real-time migration validation have not been studied. Therefore, an integrated architecture is needed to facilitate secure secret movement, application availability across the cloud transition, and uniform application security enforcement. Table I summarizes the past work on cloud migration, cloud security, IAM, and application continuity, and its significance for secure secret-platform migrations.

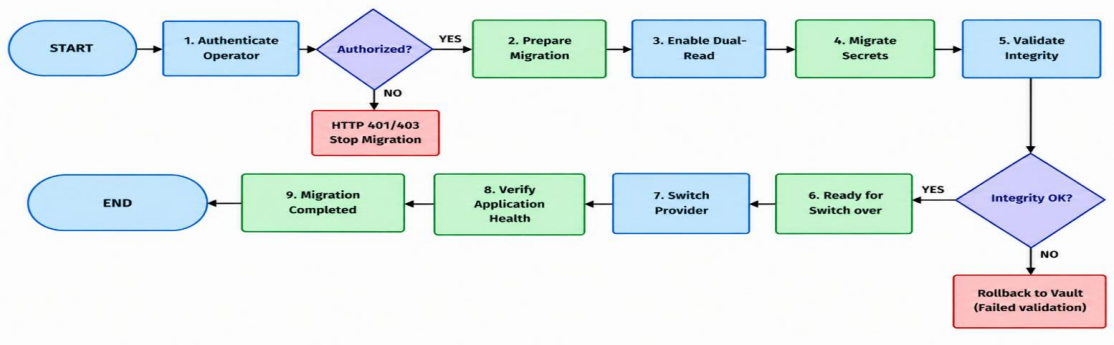
Comparative Analysis of Existing Research on Cloud-Native Security and Migration

Authors	Focus Area	Key Findings	Technologies / Tools Used	Contribution to This Research
Yuta Nakamori and Emi Hayasaka (2026)	Legacy application modernization through cloud migration	Architectural transformation and governance improve enterprise modernization.	Cloud migration frameworks, cloud-native architectures	Provides a migration foundation but does not address secret synchronization and application continuity.
Tuba Arif, Byunghyun Jo, and Jong Hyuk Park (2025)	Privacy-enhancing technologies and cloud-native security	Multilayer security strengthens privacy, trust, and compliance.	Application security, network security, infrastructure security	Supports security integration but does not specifically address continuity-aware secret migration.
Neha Surendranath, Sai Annamaiah Basava Raju, and Ranita Ganguly (2025)	Cloud migration and application modernization	Containers, microservices, and serverless computing improve deployment speed and fault tolerance.	Containers, microservices, serverless computing, cloud-native DevOps	Provides modernization strategies but does not focus on secure secret transfer and credential consistency.
Chinua Achebe and Tosin Akinwole (2024)	Zero-trust security and hybrid cloud modernization	IAM, service meshes, and automated policies strengthen migration security.	IAM, service mesh, Zero Trust Architecture, APIs, containers	Provides security principles but does not fully address secret synchronization and uninterrupted application access.
B. Nascimento et al. (2024)	Containerized application migration to Kubernetes	Kubernetes supports availability, scalability, and security in cloud environments.	Kubernetes, Microsoft Azure, containerization	Demonstrates orchestration benefits but does not investigate secret migration validation and rollback.

Authors	Focus Area	Key Findings	Technologies / Tools Used	Contribution to This Research
Theodoros Theodoropoulos et al. (2023)	Security solutions for cloud-native services	Maps cloud-native security requirements to features and open-source solutions.	Cloud-native security technologies, open-source tools	Provides security insights but lacks an integrated framework for secret platform migration and application continuity.
Proposed Research	Secure secret platform migration with application continuity	Integrates secure secret transfer, synchronization, validation, and recovery to maintain application availability during migration.	Secret management platforms, IAM, encryption, Kubernetes, automated validation, rollback mechanisms	Addresses existing gaps by combining secret confidentiality, access-policy consistency, migration validation, and application continuity within a unified framework.

3. Methodology

The suggested approach is designed to transfer application secrets from HashiCorp Vault to AWS Secrets Manager, while ensuring application availability and secret integrity. This method comprises provider abstraction, dual-provider access, a controlled migration state machine, cryptographic verification, role-based authorisation, and failure recovery. The programme does not cease throughout the migration but continues to process requests. The approach is implemented and evaluated with SecretBench data, ensuring security and characteristics of the workloads on OpenH2C. The three experimental conditions are: normal Vault operation, traditional stop-and-restart migration, and the proposed continuity-aware migration (as illustrated in Figure 1).



3.1 Proposed Secure Secret Migration Workflow with Application Continuity

3.2 Secret Provider Abstraction

The program encapsulates all secret-management activities between AWS Secrets Manager (destination, tested using LocalStack) and HashiCorp Vault KV v2 (source) with a common SecretProvider interface. Platform-specific functions are handled by separate provider implementations, and runtime switching is made possible without restarting the application container by SecretProviderFactory, which also keeps track of the current provider. A DualSecretProvider ensures continuity by first trying Vault and then falling back to AWS Secrets Manager if the source is unavailable or provides an empty result. This abstraction isolates business logic from provider-specific code and enables migration, fallback, and rollback.

3.3 Dual-Provider Application Continuity

During migration, the application invokes DualSecretProvider, which prioritizes HashiCorp Vault KV v2 and falls back to AWS Secrets Manager if Vault is unavailable or returns an empty result. This method allows for maintaining access to secrets without having to restart the application container during migration and validation. The provider factory activates AWS Secrets Manager as the provider after a successful migration and SHA-256 fingerprint verification, but Vault is still available in case a migration fails or the integrity check reveals a mismatch.

3.4 Migration Workflow and State Machine

An eleven-state finite-state machine (FSM) controls the migration process, making sure to follow the steps of authentication and authorisation, migration preparation, activation of dual providers, secret transfer, confirmation of destination integrity, and switchover. Provider transition to AWS Secrets Manager will be dependent upon a successful application health check, validation, and completion. In case of validation or migration failure, Vault can be restored as an active provider with failure and rollback transitions specified, and SHA-256 fingerprints can be used to verify source-destination secrets.

3.5 Secret Migration and Integrity Verification

The secret data are transferred from HashiCorp Vault KV v2 to AWS Secrets Manager, through the provider-specific procedures. The framework computes a SHA-256 hash of the original value, fetches the migrated destination value, and compares their hashes to verify integrity. If validation succeeds, the provider switchover process is started. If there is a mismatch, the default failure and rollback procedure is executed. To ensure that the credentials are not exposed, audit records do not store plaintext secret values, but instead only the metadata and verification results.

3.6 Failure Handling and Rollback Mechanism

The framework also contains recovery procedures for authentication, migration and validation issues. An unauthenticated or unauthorised migration request is denied before critical migration activities begin. If a migrated secret fails the integrity check, it is not recorded as being successfully migrated, but rather as a failed migration. Vault is then restored as the active provider via the rollback technique. This provides a safe recovery mechanism and also allows the program to reach the previously trusted secret source. Therefore, the technique treats migration failure as a recoverable system condition rather than as a reason to restart the application.

3.7 Dataset Integration and Workload Calibration

SecretBench was used to define secret kinds and categories, with 20 security-preserving surrogate secrets chosen for each run from all eight categories to avoid the need for operational credentials. OpenH2C workload observations for load calibration were used, and the MEDIUM tier (262,584 observations) resulted in an average pod CPU use of 2.204 cores. A heuristic exclusive to the research (2.204 times 3 or approximately 6.612) was rounded to 7 requests per second (RPS). Three different 90-second runs for each experimental condition resulted in similar request rates and secret workloads, which enabled fair comparisons of migration strategies. Table II shows the datasets used for experimental evaluation, with SecretBench providing security-preserving secret data and OpenH2C supporting Kubernetes workload calibration.

Datasets used for Experimental Evaluation

Dataset	Purpose	Dataset statistics	Data used	Workload configuration
SecretBench [17]	Characterize secret types and categories	97,479 candidates; 15,084 true secrets; 818 repositories; 49 languages; 311 file types; 8 categories	90 surrogate records; 20 per migration run	Category-based sampling covering all 8 categories
OpenH2C [18]	Calibrate HTTP workload	1,310,549 observations; 98,405 applications; 1,310,524 pods; 20-second intervals	MEDIUM tier: 262,584 observations	Mean CPU: 2.204 cores; heuristic calibration to 7 RPS (\(2.204 \times 3 \approx 7\))

3.8 Experimental Environment & Performance Metrics

The framework was developed in a Docker container environment, using Spring Boot 3.2.5, HashiCorp Vault 1.16, LocalStack 3.4, and Keycloak 24.0. AWS SDK v2 enabled operations for managing secrets at rest, and Keycloak JWTs were used to manage authorisation and authentication. The program used REST APIs for migration and health checks, SHA-256 fingerprinting, and Vault KV v2. Using OpenH2C-calibrated workloads and SecretBench surrogates, three separate 90-second runs were performed at a rate of 7 queries per second per scenario. Note: The report of findings does not contain information about the operating system, host hardware or library versions.

The framework is measured in terms of performance, availability, integrity and security. Mean and P95 delay evaluate responsiveness, while the successful-request ratio measures availability. Measures of secret integrity include SHA-256 fingerprint pass rates, whereas migration efficiency is assessed by migration length, downtime, and throughput. Authentication, RBAC authorisation, and failure-injection tests are used to find vulnerabilities. Under the tested circumstances, the dependability of

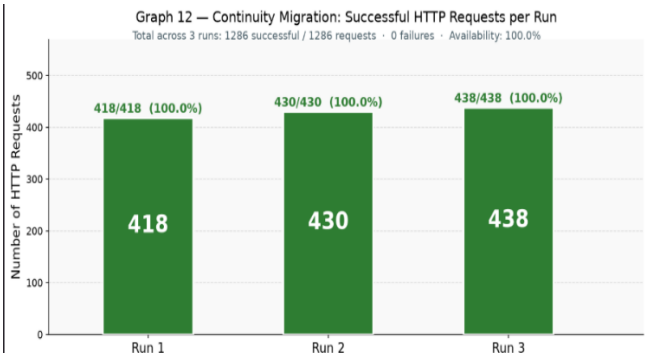
application continuity and migration is evaluated by comparing results from baseline, conventional, and continuity-aware migration.

4. Result Analysis & Discussion

The results prove that the suggested architecture allows safe secret migration without disrupting the operation of the application. Table III shows the results of the three separate runs of the continuity-aware migration method's request latency measurements. The findings reveal persistent low latency, with an overall mean latency of 11.32 ms and a P95 latency of 26.82 ms, showing that the migration procedure ensures steady application performance.

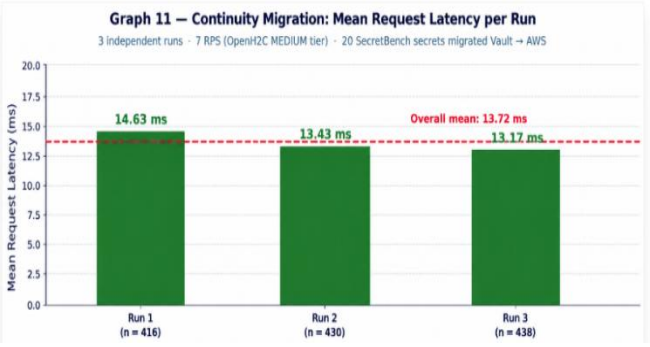
Continuity-Aware Migration Performance Results

Metric	Run 1	Run 2	Run 3	Overall
Total HTTP requests	393	345	336	1,074
Successful requests	393	345	336	1,074
Failed requests	0	0	0	0
Availability (%)	100.00	100.00	100.00	100.00
Mean latency (ms)	12.60	10.72	10.44	11.32
Median latency (ms)	9.33	9.00	9.19	9.17
P95 latency (ms)	28.32	25.90	20.41	26.82
P99 latency (ms)	33.05	33.53	30.71	32.58
Standard deviation (ms)	6.94	5.27	4.51	5.82
Minimum latency (ms)	7.02	7.13	7.11	7.02
Maximum latency (ms)	33.46	40.37	33.82	40.37
Container downtime (seconds)	0	0	0	0



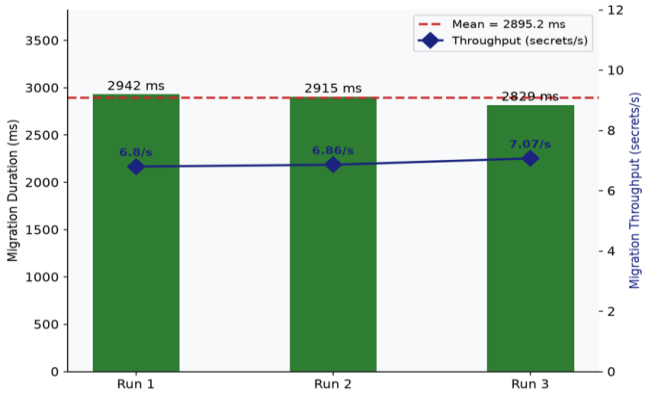
Continuity Migration: Successful HTTP Requests per Run

Figure 2 shows successful HTTP requests across three independent continuity-migration runs. Run 1 recorded 418/418 successful requests, Run 2 recorded 430/430, and Run 3 recorded 438/438. Overall, all 1,286 requests succeeded, with zero failures and 100% observed availability.



Continuity Migration: Mean Request Latency per Run.

Figure 3 shows the mean request latency for three continuity-migration runs at 7 RPS. The latency for Run 1 was 14.63 ms, Run 2 was 13.43 ms and Run 3 was 13.17 ms, giving an overall mean latency of 13.72 ms.



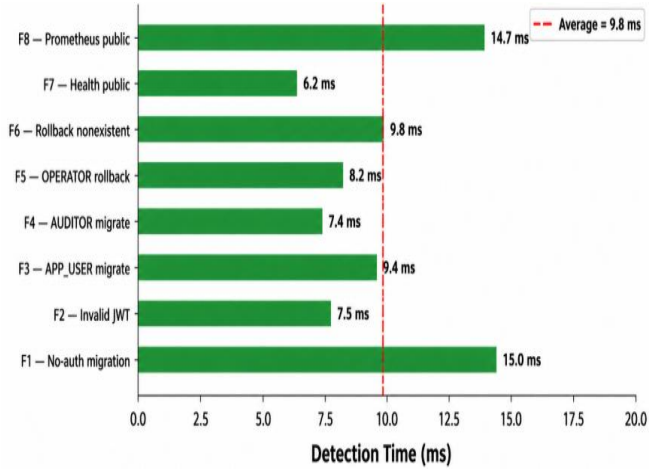
4.1 Migration Duration and Throughput Across Three Runs

Figure 4 shows the duration of migration and throughput for three separate runs. The migration duration decreased from 2,942 ms (Run 1) to 2,820 ms (Run 3), and throughput was improved from 6.8 to 7.07 secrets/s. The average migration time was around 2,895.2 ms.

Table IV shows the outcomes of eight failure and security scenarios, such as unauthenticated requests, invalid JWTs, unauthorized roles and rollback conditions. Detection and recovery were successful in all scenarios, with detection times ranging from 6.2 ms to 20.2 ms, demonstrating effective security enforcement and failure handling.

Failure Injection And Recovery Validation

Failure Scenario	Failure Detected	App Recovered	Detection Time (ms)	Result
Migration with no authentication token	Yes	Yes	6.2	PASS
Migration with invalid/malformed JWT	Yes	Yes	6.6	PASS
APPLICATION_USER role attempts migration	Yes	Yes	8.6	PASS
AUDITOR role attempts migration	Yes	Yes	9.2	PASS
OPERATOR role attempts rollback (ADMIN only)	Yes	Yes	10.9	PASS
ADMIN rollback of a nonexistent migration ID	Yes	Yes	11.9	PASS
Health endpoint accessed with no token	Yes	Yes	7.9	PASS
Prometheus metrics accessed with no token	Yes	Yes	20.2	PASS



Failure Injection and Detection Time per Scenario

The detection time values of the eight injected failure and security scenarios are depicted in Figure 5. All failure scenarios are detected correctly under the proposed framework, which shows that the framework is effective in failure handling and reliable recovery.

Table V shows the RBAC and security validation results for the following contexts: anonymous/invalid JWT, ADMIN, MIGRATION_OPERATOR, AUDITOR, and APPLICATION_USER. A 100% pass rate was achieved for all 23 HTTP responses, matching the expected responses. The average latency was different for each role, with the highest being for migration operations at 213.9 ms.

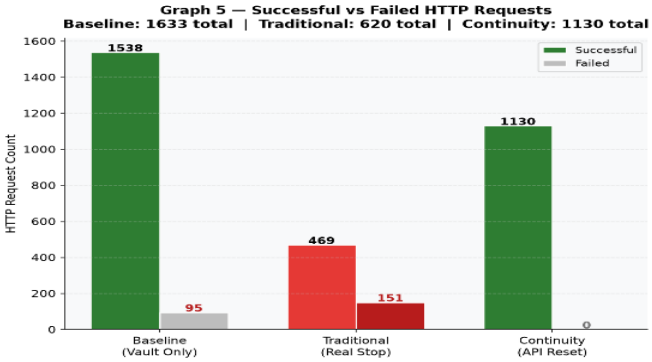
RBAC and Security Validation Results

Role / Context	Checks	Expected vs. Actual HTTP	Average Latency (ms)	Result
Anonymous / Invalid JWT	3	Matched	7.3	PASS
ADMIN	6	Matched	15.9	PASS
MIGRATION_OPERATOR	5	Matched	213.9	PASS
AUDITOR	4	Matched	12.7	PASS
APPLICATION_USER	5	Matched	10.3	PASS
Total	23	All matched	13.4 (reported)	23/23 PASS

A head-to-head comparison of the baseline, traditional stop-and-restart migration, and continuity-aware migration approaches is given in Table VI. The comparison is based on request success, latency, downtime and rollback, and illustrates the operational and performance benefits of the continuity-aware approach.

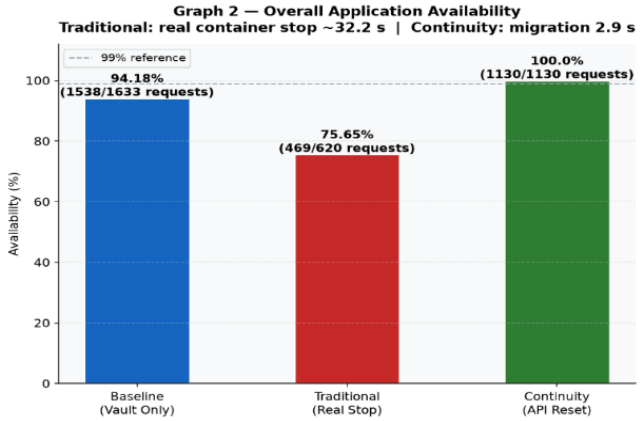
Comparative Evaluation Of Migration Approaches

Metric	Baseline	Traditional (Real Stop)	Continuity (API Reset)	Improvement
Total requests	1,887	374	1,074	—
Successful requests	1,887	246	1,074	—
Failed requests	0	128	0	Continuity: 128 fewer failures
Mean latency (ms)	14.80	257.58	11.32	Continuity: 22.7× lower mean
Median latency (ms)	8.96	13.09	9.17	Comparable
P95 latency (ms)	30.02	4,053.97	26.82	Continuity: 151× lower P95
Real downtime (s)	0	32.2	2.95	Continuity: 10.9× less
Rollback capability	N/A	Manual	Automatic	Continuity only



4.2 Successful and Failed HTTP Requests Across Migration Scenarios

Figure 6 shows the number of successful and unsuccessful HTTP requests in each scenario. Overall there were 1,538 successful requests and 95 failed requests in Baseline, and 469 successful requests and 151 failed requests in traditional migration. As per the test, the continuity-aware migration successfully handled 1130 requests with a 100% success rate.



4.3 Overall Application Availability Across Migration Scenarios

Figure 7 shows a comparison between baseline, traditional migration and continuity-aware migration. Traditional migration achieved 75.65% availability, while baseline availability was 94.18% due to container stoppage. During the tested continuity-aware migration, we observed a 100% request success rate, with 1130 successful requests and 0 failures.

4.4 Discussion

The experimental results demonstrate the advantages of application request handling under continuity-aware migration as compared with the conventional stop-and-restart migration. The proposed solution succeeded in all 1,074 requests with an average latency of 11.32 ms and a P95 of 26.82 ms. Conventional migration had 128 failed requests, a mean latency of 257.58 ms, and a P95 latency of 4,053.97 ms, in comparison. The results indicate that the switch-over may be beneficial for performance after integrity validation, and that dual-provider operation may provide a way to maintain access to the application. Security testing also showed that all 60 fingerprint comparisons succeeded, all 23 RBAC checks matched anticipated HTTP replies, and all eight injected failure scenarios were detected and successfully handled. However, the assessment used a controlled workload and LocalStack instead of actual AWS infrastructure. Therefore, rather than providing a general assurance of continuous service, the data show efficacy under tested conditions.

5. Conclusion And Future Work

The suggested safe secret-platform migration framework shows how to ensure application continuity during secret migration by integrating provider abstraction, dual-provider access, SHA-256 integrity verification, role-based access restriction, and automatic rollback. No container downtime was detected throughout the continuity-aware migration, the mean latency was 11.32 ms, the P95 latency was 26.82 ms, and the observed request success rate was 100% (1,074/1,074). The average migration time was approximately 2.9 seconds, with throughput of 7.07 secrets/s. All 8 failure-injection cases, 23 RBAC checks and all 60 fingerprint checks were successful with mean latency reduced from 257.58 ms to 11.32 ms and P95 latency reduced from 4,053.97 ms to 26.82 ms when compared to typical stop-and-restart migration. These results highlight improvements in request continuity, integrity validation, and migration performance under the evaluated conditions. The framework will be evaluated in other multi-cloud and hybrid-cloud environments with other secret-management systems and AWS infrastructure in future research. Larger secret volumes, increased request volume, concurrent migrations, network failures, and complex microservice relationships will be evaluated in further tests. Automatic secret rotation, Kubernetes-native secret operators, policy-as-code, and enhanced monitoring

could enhance operational resiliency. Additionally, testing and independent security assessments will help identify scalability, reliability, and suitability for production deployments.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

[1] M. S. Rahaman, A. Islam, T. Cerny, and S. Hutton, "Static-Analysis-Based Solutions to Security Challenges in Cloud-Native Systems: Systematic Mapping Study," Sensors, vol. 23, no. 4, p. 1755, Feb. 2023, https://doi.org/10.3390/s23041755.

[2] B. Murugeshwari, "Enhancing Enterprise Cyber Resilience through Intelligent API Security and Cloud Native Infrastructure Optimization," Int. J. Eng. \& Ext. Technol. Res., vol. 3, no. 2, pp. 2796–2805, 2021.

[3] K. R. Kiran, "API Integration Barriers in Case-Based Process Management: A Review of Interoperability and Real-Time Response Constraints Outline," Qual. Res., vol. 25, no. 3, pp. 21–48, 2025, https://doi.org/10.5281/zenodo.16782616.

[4] M. Sethi, "Beyond Ticketing: A Governance-Enforced Autonomous Control Plane for Cloud Database Operations at Enterprise Scale," Am. J. Technol., vol. 5, no. 3, pp. 50–61, Mar. 2026, https://doi.org/10.58425/ajt.v5i3.512.

[5] J. Kurma, J. V. Mamidala, A. Attipalli, S. J. Enokkaren, V. Bitkuri, and R. Kendyala, "A Review of Security, Compliance, and Governance Challenges in Cloud-Native Middleware and Enterprise Systems," Int. J. Res. Appl. Innov., vol. 5, no. 1, pp. 6434–6443, 2022, https://doi.org/10.15662/IJRAI.2022.0501003.

[6] C. Kulkarni, C. Medicherla, B. Vulugundam, T. P. Patel, M. Parikh, and S. R. K. V. Bayyavarapu, "Multi-Layered Security Architecture for Cloudnative Databases- Design Patterns and Implementation," in SoutheastCon 2026, IEEE, Feb. 2026, pp. 1–6. https://doi.org/10.1109/SoutheastCon63549.2026.11476097.

[7] G. Karamchand, "Secure and Privacy-Preserving Data Migration Techniques in Cloud Ecosystems," J. Data Anal. Crit. Manag., vol. 01, no. 2, pp. 67–78, 2025.

[8] R. Eswaramoorthi, "*Secure Container Orchestration Framework for Resilient Cloud-Native Distributed Applications,*" Trans. Internet Secur. Cloud Serv. Distrib. Appl., vol. 3, no. 2, pp. 17–25, 2026.

[9] S. K. Malaraju, G. C. Kakaraparthi, and R. C. Bondalapati, "Efficient Deployment of Scalable AI Models via Kubernetes Orchestration in Hybrid Cloud Setups," in 2025 International Conference on Intelligent and Secure Engineering Solutions (CISES), IEEE, Aug. 2025, pp. 1450–1454. https://doi.org/10.1109/CISES66934.2025.11265182.

[10] T. Theodoropoulos et al., "Security in Cloud-Native Services: A Survey," J. Cybersecurity Priv., vol. 3, no. 4, pp. 758–793, 2023, https://doi.org/10.3390/jcp3040034.

[11] K. R. Singi and M. Konda, "Streaming Telemetry Analytics for AI-Driven Service Performance Prediction in Large-Scale Microservices Architectures," J. Comput. Sci. Technol. Stud., vol. 8, no. 8, pp. 62–72, Jul. 2026.

[12] Y. Nakamori and E. Hayasaka, "*A Strategic Framework For Modernizing Legacy Enterprise Applications Through Cloud-Based Migration Models,*" Int. J. Next-Generation Eng. Technol., vol. 03, no. 04, pp. 80–91, 2026.

[13] T. Arif, B. Jo, and J. H. Park, "A Comprehensive Survey of Privacy-Enhancing and Trust-Centric Cloud-Native Security Techniques Against Cyber Threats," Sensors, vol. 25, no. 8, p. 2350, Apr. 2025, https://doi.org/10.3390/s25082350.

[14] N. Surendranath, S. A. Basava Raju, and R. Ganguly, "*Migrating Traditional Applications to Cloud-Native DevOps: A Framework for Successful Cloud Migration and Modernization,*" in 2025 13th International Symposium on Digital Forensics and Security (ISDFS), Boston, MA, USA: IEEE, 2025, pp. 1–6. https://doi.org/10.1109/ISDFS65363.2025.11012102.

[15] C. Achebe and T. Akinwole, "Secure Legacy System Modernization Using Zero-Trust Principles and Cloud-Native Migration Strategies," 2024, Research Gate.

[16] B. Nascimento, R. Santos, J. Henriques, M. V Bernardo, and F. Caldeira, "Availability, Scalability, and Security in the Migration from Container-Based to Cloud-Native Applications," Computers, vol. 13, no. 8, p. 192, Aug. 2024, https://doi.org/10.3390/computers13080192.

[17] michoo, "secretbench," 2023, Zenodo. doi: https://doi.org/10.5281/zenodo.7571266.

[18] alibaba, "openh2c," 2025, Github.