Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Enterprise Application Modernization in the Cloud Era: A Case Study of Ticket Management System Migration

Harris Peter Baskaran
Google Inc., USA
Corresponding Author: Harris Peter Baskaran, **E-mail**: harrispeterbaskaran@gmail.com

| ABSTRACT

The migration and modernization of enterprise applications from on-premises infrastructure to cloud environments presents significant technical and organizational challenges. This article documents the transformation journey of a large-scale ticket management system from a monolithic architecture on VMware to a cloud-native implementation using containerization and microservices. Through a strategic two-phase process, the project first established baseline operations via lift-and-shift migration to Google Compute Engine before evolving to a fully containerized architecture on Google Kubernetes Engine. The transformation included database migration from Oracle to PostgreSQL, implementation of advanced network security policies, comprehensive disaster recovery mechanisms, and modern observability solutions. The resultant architecture demonstrated marked improvements in system reliability, performance, security posture, and operational efficiency. The documented strategies, challenges, and outcomes provide valuable insights for organizations undertaking similar modernization initiatives in enterprise environments.

| KEYWORDS

Cloud migration, containerization, microservices, enterprise modernization, disaster recovery

| ARTICLE INFORMATION

ACCEPTED: 20 October 2025 **PUBLISHED:** 06 November 2025 **DOI:** 10.32996/jcsts.2025.7.11.30

1. Introduction

Enterprise application modernization represents one of the most significant technological challenges facing organizations today. As legacy systems reach their operational limits, businesses increasingly turn to cloud environments to enhance scalability, reliability, and cost-efficiency. The transition to microservices architecture has emerged as a crucial enabler for SRE practices, facilitating the migration to cloud-native architectures through incremental modernization processes that allow for continuous delivery and deployment. However, the journey from on-premises infrastructure to cloud-native architectures involves complex technical decisions and organizational changes that extend far beyond the initial migration process.

The prevalent "lift-and-shift" approach—where applications are moved to cloud infrastructure with minimal modifications—offers an expedient path to cloud adoption but often fails to capitalize on the fundamental advantages of cloud computing. This strategy, while reducing immediate migration complexity, merely relocates existing technical debt and architectural limitations to a new environment. When designing cloud-based solutions, organizations must carefully evaluate their specific requirements across infrastructure, platform, and software service models to determine the optimal architecture that balances control, flexibility, and operational efficiency. Applications that are simply relocated without redesign continue to carry forward their original constraints in scalability, reliability, and operational efficiency.

This article examines a comprehensive modernization initiative for a large-scale ticket management system serving a multinational enterprise. The system, originally deployed in an on-premises VMware environment, processed substantial volumes

of support tickets daily with strict SLA requirements. The legacy architecture featured a monolithic application structure tightly coupled with an Oracle database, creating significant operational bottlenecks and maintenance challenges as transaction volumes increased.

The research objectives of this study center on documenting the technical and operational challenges encountered during the multi-phase modernization process, quantifying the performance improvements achieved through cloud-native architecture adoption, and developing a generalizable framework for similar enterprise modernization initiatives. The evolution toward microservices architecture necessitates both architectural refactoring and organizational restructuring to accommodate new development and operational paradigms that support scalability and resilience.

The transformation journey proceeded in two distinct phases. The initial phase employed a lift-and-shift approach, migrating the application from on-premises VMware infrastructure to Google Compute Engine virtual machines. This established a baseline while maintaining the existing architecture. The second phase involved a comprehensive redesign for cloud-native operation, including containerization of services, database migration, implementation of cloud-native security controls, transition to integrated monitoring solutions, and development of comprehensive disaster recovery capabilities. Architectural decisions throughout this process require careful consideration of various cloud service models and their implications for infrastructure management, application development, and operational control.

2. Modernization Framework and Methodology

The modernization of enterprise applications requires a structured approach that balances technical innovation with business continuity. This section presents the methodological framework employed in transforming the ticket management system, beginning with a comparative analysis of migration strategies.

Enterprise application migration strategies exist on a spectrum from minimal modification to complete redesign. The lift-and-shift approach represents the most conservative strategy, maintaining application architecture while changing only the hosting infrastructure. While this approach minimizes initial effort and risk, it fails to address fundamental architectural limitations and cannot fully leverage cloud-native capabilities. In contrast, modernization involves re-architecting applications to exploit cloud-native services, containerization, and microservices patterns. The migration from legacy monolithic systems to microservice architecture requires systematic process steps, including architecture recovery, service identification, dependency analysis, and service extraction, with particular attention to database decomposition challenges that often present the most significant migration obstacles. The two-phase approach adopted in this case study allowed for risk mitigation through an initial lift-and-shift phase. The theoretical framework guiding this modernization initiative focused primarily on establishing a robust cloud deployment for an existing third-party application rather than extensive architectural redesign. Unlike custom-developed applications where microservices decomposition might be the primary approach, this project centered on creating a reliable cloud environment that maintained compatibility with the ticket management system's existing architecture. The framework emphasized infrastructure provisioning, configuration management, and system validation to ensure functional equivalence between on-premises and cloud deployments.

Given the third-party nature of the application, the modernization approach prioritized out-of-place system building rather than internal architectural modifications. The decision-making criteria focused on cloud environment configuration, network design, security implementation, and operational considerations that would support the application without requiring changes to its core codebase. This included determining appropriate instance sizing, network topology, access controls, and monitoring solutions that aligned with cloud best practices while maintaining application compatibility.

In the second phase, the evaluation framework for modernization success incorporated quantitative and qualitative metrics across multiple dimensions that were relevant to a third-party application migration. Reliability metrics included system availability, mapping all dependencies, and recovery time objectives that matched or exceeded the on-premises environment. Performance metrics encompassed request latency, throughput capacity, and resource utilization efficiency to ensure the cloud deployment delivered equivalent or superior performance. Operational metrics measured deployment success, environment provisioning time, and system restoration capabilities. Security metrics evaluated the cloud perimeter protection, access control implementation, and vulnerability management processes appropriate for the third-party application.

A critical success factor in the migration process was the early establishment of robust deployment pipelines that supported independent development and deployment of services while maintaining system integrity throughout the transformation. The migration sequence carefully considered service coupling, ensuring that tightly integrated components were either migrated together or temporarily bridged through facade patterns until complete migration could be achieved.

This methodological framework provided the foundation for the technical implementation described in subsequent sections, ensuring that architectural decisions remained aligned with business objectives throughout the transformation process.

3. Technical Implementation and Architecture Evolution

The transformation of the ticket management system proceeded through two distinct phases, each addressing specific technical challenges while maintaining system availability. This section details the technical implementation and architectural evolution throughout this process.

Phase 1: Initial Migration (VMware to GCE)

The first phase focused on establishing the application in a cloud environment with minimal architectural changes. The assessment of the legacy infrastructure revealed a three-tier application architecture deployed on VMware ESXi hosts, with web servers and application servers running on Linux, and multiple Oracle databases running on physical servers. The application comprised substantial code developed over many years, incorporating multiple programming languages and frameworks. System dependencies included message queues, file system dependencies, and integrations with external services.

Migration planning incorporated both technical and organizational considerations. A comprehensive dependency mapping exercise identified all integration points, data flows, and infrastructure requirements. The migration strategy employed a parallel deployment approach, where the cloud infrastructure was established while maintaining the existing on-premises system. This approach allowed for thorough testing and validation before traffic cutover. Resource sizing was determined through performance profiling of the existing environment, with additional capacity allocated to account for growth projections.

The migration execution followed a structured approach, beginning with the establishment of the networking infrastructure on the Google Cloud Platform (GCP). Virtual private clouds (VPCs) were configured with appropriate subnet allocation to maintain logical separation between application tiers. Through a secure Cloud Storage bucket, a drop box transfer point between the on-premises data center and the GCP environment was established. Virtual machines in Google Compute Engine (GCE) were provisioned to match the specifications of existing VMware instances, with appropriate modifications to leverage cloud-specific optimizations.

Data migration represented a critical path in the migration timeline. The Oracle database was migrated using a combination of backup/restore operations and continuous replication mechanisms to minimize downtime. During the transition period, a unidirectional replication solution maintained data consistency between the on-premises and cloud environments. Performance benchmarking throughout the migration process established baseline metrics for subsequent optimization efforts. Key performance indicators included transaction throughput, request latency, database query performance, and system resource utilization.

Phase 2: Cloud-Native Transformation (GCE to GKE)

The second phase focused on architectural transformation to fully leverage cloud-native capabilities. The database migration from Oracle to CloudSQL PostgreSQL represented a fundamental shift in the application's data layer. Initially, the open-source tool ora2pg was evaluated for the migration, but it proved insufficient for the complexity and scale of the production database. Subsequently, a vendor-provided specialized migration software was employed to ensure data integrity and schema compatibility. The migration process involved comprehensive assessment of schema complexity, compatibility analysis between Oracle and PostgreSQL environments, schema conversion with attention to data type differences, and thorough validation of migrated data. The date-time data types in particular needed thoughtful reviews and considerations. Particular attention was given to handling Oracle-specific features such as packages, procedures, and sequences that required alternative implementation patterns in PostgreSQL. The vendor solution provided optimized tooling for these conversions while maintaining referential integrity and ensuring application functionality remained consistent post-migration.

Rather than decomposing the application into microservices, which was not feasible due to the vendor-provided nature of the software, the modernization effort focused on establishing appropriate operational boundaries and controls within the Kubernetes environment. Significant effort was dedicated to defining failure domains for the service to ensure appropriate resilience planning, establishing clear security boundaries through network segmentation and access controls, and implementing comprehensive monitoring solutions to maintain visibility into system health and performance. The team invested considerable resources in disaster recovery testing to validate system recoverability under various failure scenarios, implementing automated backup solutions with verified restoration procedures, and documenting operational runbooks for maintaining the application in its containerized state. While the core application architecture remained consistent with the vendor's design, the surrounding infrastructure leveraged Kubernetes capabilities for deployment consistency, resource management, and operational standardization.

Deployment automation was implemented using a combination of infrastructure-as-code tools and continuous integration/continuous deployment (CI/CD) pipelines. Helm chart defined the Kubernetes resources required for each service, with environment-specific configuration managed through value files. The CI/CD pipeline automates building, testing, and

deploying containerized services, with progressive deployment strategies (blue-green and canary), minimizing risk during updates.

Network security transformation represented a significant enhancement to the system's security posture. GCP firewalls implement perimeter security at the network level, with rules defined based on the principle of least privilege. Within the Kubernetes cluster, network policies provide granular control over pod-to-pod communication. We had to design this for over 200 inter pod communication and we used Cilium, a BPF-based networking solution. This implementation allowed us to enforce these policies efficiently while providing enhanced visibility into network flows.

Monitoring modernization focused on establishing comprehensive observability across the transformed architecture. The monitoring solution incorporated three key pillars: metrics, logs, and traces. Infrastructure and application metrics were collected through a combination of Kubernetes-native monitoring tools and application instrumentation. Centralized logging aggregates container logs, application logs, and system events with appropriate tagging for correlation. The container orchestration platform provided native capabilities for health monitoring, detecting, and responding to application and infrastructure failures automatically while maintaining the desired system state.

The transition to cloud-native architecture delivered significant improvements in system scalability, reliability, and operational efficiency. The containerized services automatically scale based on workload demands, optimizing resource utilization while maintaining performance under variable load conditions.

4. Operational Impact and Performance Analysis

The migration and modernization of the ticket management system yielded substantial improvements across multiple operational dimensions. This section presents a comprehensive analysis of these improvements, focusing on system reliability, security posture, disaster recovery capabilities, and operational efficiency.

Comparative Analysis of System Reliability Metrics

System reliability represents a critical success metric for the modernization initiative. Prior to modernization, the legacy system operated with a measured availability percentage that increased significantly following the cloud-native transformation. This improvement stemmed from multiple architectural enhancements, including the implementation of self-healing Kubernetes clusters, automated instance recovery, and load-balanced service deployment across multiple availability zones.

Latency measurements demonstrated significant performance improvements across key user interactions. Average ticket creation latency decreased considerably, while search operations across the ticket database showed even more dramatic improvements. These performance gains resulted from a combination of factors, including the optimized database schema in PostgreSQL, efficient containerization of application components, and the implementation of caching mechanisms for frequently accessed data.

Throughput capacity increased substantially, with the system's ability to process concurrent transactions improving significantly. This enhancement was particularly evident during peak usage periods, where the legacy system had previously encountered infrastructure scaling limits despite maintaining acceptable performance. The modernized cloud architecture removed these scaling constraints, enabling the system to handle substantially higher transaction volumes without requiring manual intervention. The comparative analysis of container orchestration platforms indicates that Kubernetes provides superior auto-scaling capabilities and more efficient resource utilization compared to alternatives like Docker Swarm, particularly for workloads with variable demand patterns and complex service dependencies, explaining the substantial performance improvements observed in the modernized system.

Security Posture Enhancement

The security posture of the ticket management system underwent a fundamental transformation through the implementation of cloud-native security controls. The legacy environment relied primarily on perimeter security and network segmentation, with limited visibility into internal traffic patterns. The modernized architecture implemented a defense-in-depth strategy, incorporating multiple layers of security controls.

Cloud-native security requires a fundamentally different approach that addresses the unique challenges of dynamic, distributed environments by implementing security across four key dimensions: cloud infrastructure, Kubernetes clusters, containerized workloads, and application code. This comprehensive approach enabled the ticket management system to implement security controls that were integrated directly into the infrastructure supporting the application. Within the Kubernetes environment, network policies enforced through Cilium provide protocol-aware traffic filtering between pods, detecting and blocking potential attack patterns at the application layer.

The security approach focused primarily on external protections and infrastructure-level controls, as the vendor-provided software limited the ability to modify internal application security mechanisms. Rather than implementing service identities and mutual TLS for internal communications, security efforts concentrated on establishing robust perimeter controls, implementing comprehensive network segmentation, enforcing strict access management for administrative functions, and ensuring proper isolation between environments. Vulnerability management focused on maintaining current patch levels for the underlying infrastructure and container hosts, while working within the vendor's release schedule for application updates. This infrastructure-centric security approach allowed for significant security improvements without requiring modifications to the vendor-provided application code.

A. Disaster Recovery Capabilities and Testing

The modernization initiative included a comprehensive overhaul of disaster recovery capabilities, moving from basic database backups to a fully tested, multi-region recovery solution. The DR architecture implemented cross regional replica databases to a secondary region, with containerized application components deployable through infrastructure-as-code templates. This approach enabled substantial improvements in recovery point objectives and recovery time objectives compared to the legacy system's metrics.

A structured disaster recovery testing framework was developed to validate these capabilities under various failure scenarios. Current research on disaster recovery preparedness indicates that organizations with regular testing programs that include full-scale simulations across multiple scenarios demonstrate recovery times up to three times faster than those conducting limited or theoretical exercises, validating the comprehensive testing approach adopted for the ticket management system. The testing methodology incorporated three categories of scenarios: infrastructure failures, application failures, and operational failures, with each scenario including clearly defined success criteria, recovery procedures, and verification steps.

Key learnings from the DR exercises led to several refinements in the recovery process. These refinements improved recovery time performance significantly between the first and most recent tests.

B. Operational Efficiency Gains

The modernization initiative delivered substantial operational efficiency gains, reducing management overhead while improving service quality. Manual intervention requirements decreased significantly, with routine operational tasks such as scaling, patching, and backup management largely automated through the Kubernetes platform and associated tooling. Incident response efficiency improved, with the mean time to resolution for critical incidents decreasing considerably.

Comparative analysis of cloud-native architectures demonstrates that Kubernetes-based deployments offer significant operational advantages for complex enterprise applications through standardized deployment patterns, declarative configuration management, and comprehensive ecosystem integration, while serverless approaches may provide greater efficiency for specific workload types with intermittent execution patterns. For the ticket management system, the Kubernetes-based approach delivered optimal benefits due to the persistent nature of the workload and complex integration requirements.

Resource utilization efficiency improved markedly through the implementation of container-based deployment and dynamic scaling. The legacy environment maintained excess capacity to accommodate peak loads, resulting in significant resource underutilization during normal operation. The modernized system dynamically adjusted resource allocation based on actual demand, reducing infrastructure costs while maintaining performance objectives. Deployment efficiency showed dramatic improvement, with the average time to implement and release new features decreasing substantially, while change management risk was significantly reduced through the implementation of canary deployment strategies.

Conclusion

The modernization of the ticket management system from a traditional on-premises deployment to a cloud-native architecture exemplifies how strategic technical transformation can deliver substantial operational benefits. The two-phase approach—beginning with lift-and-shift migration followed by architectural redesign—provided a balanced path that minimized risk while enabling progressive optimization. Critical success factors included thorough dependency mapping, incremental service decomposition based on business capabilities, automated deployment pipelines, comprehensive security controls, and regular disaster recovery testing. The database migration from Oracle to PostgreSQL presented notable challenges, particularly in handling proprietary features and ensuring data consistency, yet yielded significant performance advantages. Organizations undertaking similar modernization initiatives should prioritize early investment in observability tooling, establish clear service boundaries before containerization, implement comprehensive testing frameworks, and develop team capabilities aligned with cloud-native operational models. As containerization and orchestration technologies continue to mature, future opportunities exist for enhanced service mesh implementations, advanced machine learning for operational optimization, and deeper integration between development and security workflows.

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] Armin Balalaie et al., "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," IEEE Explore, 2016. https://ieeexplore.ieee.org/document/7436659

[2] Michael J. Kavis, "Architecting the Cloud: Design Decisions for Cloud Computing Service Models(SaaS, PaaS, IaaS)," SLogix, 2024. https://slogix.in/cloud-computing/architecting-the-cloud-design-decisions-for-cloud-computing-service-modelssaas-paas-iaas/

[3] Kristian Tuusjärvi et al., "Migrating a Legacy System to a Microservice Architecture," ResearchGate, 2024. https://www.researchgate.net/publication/377044123_Migrating_a_Legacy_System_to_a_Microservice_Architecture

[4] Brent Frye, "8 Steps for Migrating Existing Applications to Microservices," Software Engineering Institute, Carnegie Mellon University, 2020. https://insights.sei.cmu.edu/blog/8-steps-for-migrating-existing-applications-to-microservices/

[5] EnterpriseDB, "The Complete Oracle to Postgres Migration Guide: Tools, Schema, and Data." https://www.enterprisedb.com/blog/the-complete-oracle-to-postgresql-migration-guide-tutorial-move-convert-database-oracle-alternative?lang=en

[6] Red Hat, "What is container orchestration?" 2025. https://www.redhat.com/en/topics/containers/what-is-container-orchestration

[7] Venkat Marella, "Comparative Analysis of Container Orchestration Platforms: Kubernetes vs. Docker Swarm," ResearchGate, 2024. https://www.researchgate.net/publication/387028160_Comparative_Analysis_of_Container_Orchestration_Platforms_Kubernetes_vs_Docker_Swarm

[8] Palo Alto Networks, "What Is Cloud-Native Security?" https://www.paloaltonetworks.com/cyberpedia/what-is-cloud-native-security

[9] Brent Ellis, "The State of Disaster Recovery Preparedness 2024," Disaster Recovery Journal, 2024. https://drj.com/journal_main/the-state-of-disaster-recovery-preparedness-2024/

[10] Gireesh Kambala, "Cloud-Native Architectures: A Comparative Analysis of Kubernetes and Serverless Computing," ResearchGate, 2023. https://www.researchgate.net/publication/388717188_Cloud-Native_Architectures_A_Comparative_Analysis_of_Kubernetes_and_Serverless_Computing