SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

Architecting Zero-Downtime Re-Platforming: A Decoupled Configuration Driver Model for Legacy-to-Cloud Warehouse Management Migration

Prashant Vikas Patil

Independent Researcher, USA. ORCID ID: https://orcid.org/0009-0007-8651-8386

Abstract
Retail and logistics enterprises operating legacy warehouse management platforms on mainframe infrastructure face a structural migration problem: business rules are hardcoded into vendor source code, forcing an all-or-nothing cutover when moving to cloud-native infrastructure. This architectural rigidity concentrates operational risk into a single cutover event, where a routing or timing defect can halt physical fulfillment operations. This article presents a Decoupled Configuration Driver Model that separates warehouse routing logic from underlying platform code through a runtime configuration layer, enabling incremental, canary-gated migration with automated rollback. The model comprises three phases: parallel-run telemetry baselining, configuration abstraction of hardcoded logic, and gated cutover with continuous validation. Drawing on enterprise re-platforming practice and the continuous delivery literature, this article argues that configuration-driven decoupling converts legacy modernization from a high-risk rewrite into a reversible, incremental engineering process.

Keywords: legacy modernization, warehouse management systems, configuration-driven architecture, canary deployment, cloud migration, zero-downtime, enterprise re-platforming

1. Introduction

Enterprises operating legacy warehouse management systems (WMS) on mainframe infrastructure such as IBM AS400 running PKMS face a specific and recurring architectural problem when modernizing to cloud-native platforms: the business rules that govern order routing, inventory allocation, and shipment prioritization are typically hardcoded directly into vendor source code. Migrating this logic to a distributed cloud environment therefore requires either a full platform rewrite or an all-or-nothing cutover, both of which concentrate operational risk into a narrow window. A routing defect or data translation error discovered only after cutover can halt physical fulfillment operations across an entire distribution network, with consequences that are immediate and costly: missed carrier cutoffs, stalled conveyor lines, and blocked outbound trailers.

This risk profile is well documented in the legacy-to-cloud migration literature. Hasan et al. (2023) note that organizations resist cloud migration of core operational systems specifically because of downtime risk, and that this resistance is compounded when the systems in question coordinate physical processes rather than purely digital ones. Vendor lock-in intensifies the problem: Opara-Martins, Sahandi, and Tian (2016) show that the absence of standardized interfaces between legacy platforms and modern cloud services leaves enterprises with few incremental migration paths, pushing them toward high-risk, big-bang cutovers as the only practical option.

This article proposes an architectural response to this specific problem: a Decoupled Configuration Driver Model that separates warehouse routing and allocation logic from the underlying platform code through an externalized, runtime-modifiable configuration layer. Rather than migrating hardcoded business logic as an inseparable part of a system rewrite, the model extracts that logic into configuration artifacts that can be tested, versioned, and cut over incrementally, with each increment independently validated and reversible. This reframes legacy WMS modernization from a single high-stakes rewrite into a sequence of small, observable, and reversible engineering steps.

1.1 Contribution of this article

This article contributes: (1) a three-phase reference architecture for decoupling legacy warehouse logic from platform code during cloud migration; (2) a configuration-driven mechanism for incremental, canary-gated cutover with automated rollback, informed by continuous delivery and chaos engineering practice; and (3) a qualitative comparison of this approach against traditional all-or-nothing re-platforming, framed as an illustrative architectural analysis rather than a validated empirical case study.

2. Literature Review

The technical difficulty of migrating legacy systems to cloud-native architectures has been examined extensively from an architectural perspective. Hasan et al. (2023) conducted a systematic review of legacy-to-cloud migration research and found that most published architectural guidance recommends microservice decomposition as a target state, but that the literature is comparatively thin on how organizations should sequence the transition from monolithic legacy logic to that target state without service interruption. This gap is directly relevant to warehouse management re-platforming, where the "monolith" is often not custom application code but vendor-controlled WMS logic that cannot be freely refactored.

Vendor lock-in is a second structural obstacle. Opara-Martins, Sahandi, and Tian (2016) surveyed enterprise practitioners and found that lock-in risk is exacerbated, not reduced, as organizations move from on-premises to cloud infrastructure, because proprietary APIs and closed data formats simply shift from one vendor to another. Their findings support the architectural case for a decoupling layer that is independent of any single platform vendor, since such a layer preserves an organization's ability to redirect logic execution without renegotiating vendor contracts.

The microservices migration literature offers relevant precedent for incremental, logic-preserving decomposition. Martínez Saucedo et al. (2025) conducted a systematic mapping study of 114 primary studies on monolith-to-microservices migration and found that migration techniques vary widely in their degree of automation and evaluation rigor, but that a consistent success factor across industrial case studies is the ability to migrate functional slices independently, rather than requiring a single coordinated cutover of the entire system. The Decoupled Configuration Driver Model applies this same principle at the level of business-rule configuration rather than service boundaries.

The continuous delivery and deployment literature provides the empirical foundation for the model's cutover mechanism. Shahin, Zahedi, Babar, and Zhu (2019), in a mixed-methods study of 21 practitioners across 19 organizations, found that architectures enabling continuous delivery share a common property: they decouple the release of a code artifact from the activation of the behavior it implements, allowing new logic to be deployed dormant and activated independently. Rodríguez et al. (2017), in a systematic mapping study of continuous deployment practice, similarly found that organizations adopting continuous deployment consistently pair it with automated rollback mechanisms, since deployment frequency without a reliable rollback path increases rather than decreases operational risk. Laukkanen et al. (2016) examined continuous delivery adoption specifically within a stage-gate managed organization, a governance structure common in large enterprises including regulated retail and logistics operations, and found that bottom-up, incremental adoption succeeded where top-down wholesale process replacement did not, reinforcing the case for an incremental rather than big-bang migration posture even within hierarchical governance environments.

Database-layer migration research offers a directly applicable technique for the model's telemetry and cutover phases. de Jong, van Deursen, and Cleve (2017) developed a method for zero-downtime SQL schema evolution that abstracts schema changes behind a compatibility layer, allowing an application to be redeployed without taking the underlying data store offline. The Decoupled Configuration Driver Model applies an analogous abstraction principle to warehouse routing logic rather than database schema.

Resilience validation during migration is informed by the chaos engineering literature. Basiri et al. (2016), describing Netflix's production resilience testing practice, established the principle that failure modes should be discovered through controlled, reversible experiments in production-adjacent environments rather than assumed away in design documents. This principle motivates the model's use of parallel-run telemetry and canary-gated traffic shifting as mechanisms for discovering integration defects before they affect the full production population.

Finally, the technical debt literature contextualizes the cost argument for decoupling. Guo, Spínola, and Seaman (2016), in a case study of technical debt management costs, found that deferred architectural remediation compounds in cost over time and that the interest paid on unaddressed architectural debt frequently exceeds the original remediation cost. This finding supports treating hardcoded warehouse logic as a form of

architectural debt whose remediation, while requiring upfront investment in a configuration layer, avoids compounding migration risk in future modernization cycles. Foundational warehouse operations research (Gu, Goetschalckx, and McGinnis, 2007) further establishes that release-timing and routing decisions in warehouse operations are highly sensitive to non-linear congestion effects, reinforcing why any migration approach touching routing logic must preserve tight operational control throughout the transition rather than deferring validation to a single post-cutover event.

3. Methodology: The Decoupled Configuration Driver Model

3.1 Design rationale

The central design decision underlying this model is the separation of decision logic (which warehouse order goes where, under what conditions) from execution infrastructure (the platform, cloud or mainframe, that carries out that decision). In conventional legacy WMS deployments, these two concerns are fused: routing rules exist as compiled logic within the vendor platform, so any change to routing behavior requires a platform-level deployment. The Decoupled Configuration Driver Model externalizes decision logic into a configuration layer that both the legacy platform and the new cloud platform can consume identically, making the underlying execution infrastructure substitutable without requiring the decision logic itself to change during the substitution.

3.2 Three-phase architecture

The model operates in three sequential but overlapping phases.

Phase 1 — Parallel telemetry baseline. Before any cutover activity begins, a non-intrusive telemetry layer observes both the legacy platform and a cloud sandbox environment processing mirrored, non-authoritative copies of live order traffic. This phase, analogous to the shadow-mode validation techniques used in other enterprise migration domains, establishes a behavioral baseline: how the legacy platform actually routes orders under real operating conditions, including edge cases that design documentation typically omits. Because the telemetry layer only observes and does not act on traffic, it introduces no additional risk to live operations.

Phase 2 — Configuration abstraction. Hardcoded routing and allocation rules are extracted from the legacy platform's source logic and re-expressed as declarative configuration artifacts, versioned and stored independently of any single platform's deployment lifecycle. This phase is deliberately conservative: rules are extracted and validated against the Phase 1 telemetry baseline to confirm behavioral equivalence before any rule is used to drive live decisions. Extraction proceeds incrementally by order type and routing scenario, rather than as a single monolithic translation effort, consistent with the functional-slice migration pattern observed in the broader microservices literature.

Phase 3 — Canary-gated cutover. Once a configuration slice is validated, live traffic corresponding to that slice is shifted from the legacy platform to the cloud-native execution path in small increments, with the legacy platform continuing to run in parallel as a fallback. Each increment is evaluated against defined stability criteria, drawing on the same automated-rollback discipline documented in continuous deployment practice: if a monitored condition (data validation failure, timeout, unexpected routing outcome) is detected, traffic for that slice reverts automatically to the legacy platform pending investigation. Only after a slice has operated stably across multiple increments is legacy execution for that slice retired.

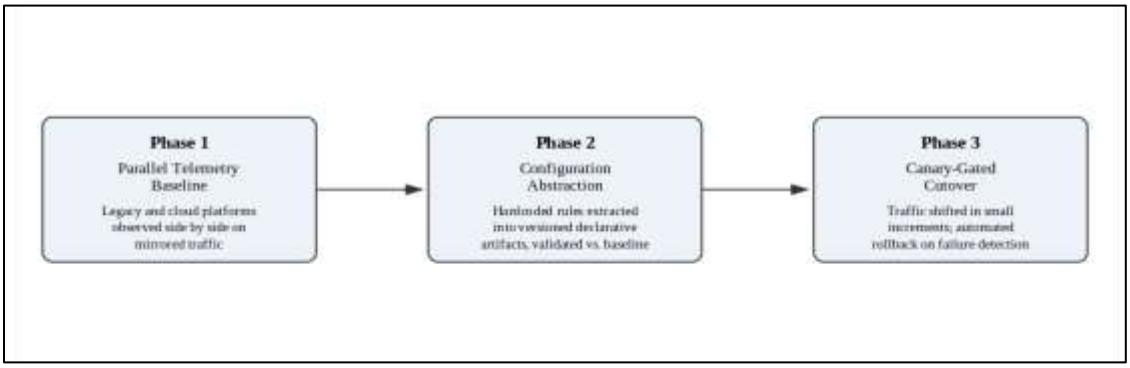

Fig. 1. Three-phase migration sequence of the Decoupled Configuration Driver Model.

3.3 Decoupled topology

Architecturally, the model separates the transactional data plane (legacy and cloud-native workloads actively processing orders) from a distinct capture and verification plane, connected by an event-driven, asynchronous telemetry channel rather than a synchronous dependency. This separation, consistent with established enterprise integration and asynchronous messaging patterns, ensures that migration validation activity cannot introduce latency or failure modes into live transactional processing, regardless of the load placed on the verification plane itself.

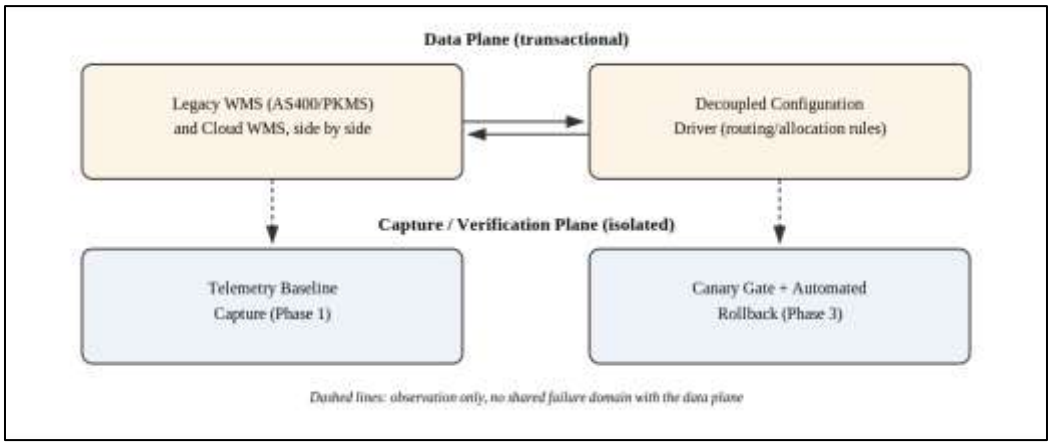

Fig. 2. Decoupled topology separating the transactional data plane from the capture and verification plane.

Table 1. Traditional All-or-Nothing Re-Platforming vs. the Decoupled Configuration Driver Model

Dimension	Traditional Cutover	Decoupled Configuration Driver Model
Logic migration unit	Entire platform, single event	Individual routing/order-type slices
Rollback granularity	Full platform rollback only	Per-slice automated rollback
Pre-cutover validation	Design review, limited live-traffic testing	Parallel telemetry baseline against real traffic
Vendor dependency	High — logic bound to platform code	Reduced — logic externalized to configuration
Failure exposure window	Entire order volume, at single cutover	Limited to the traffic increment under test
Governance fit	Difficult to reconcile with stage-gate review cycles	Compatible with incremental, bottom-up stage-gate adoption

4. Results and Discussion

This section presents an illustrative walkthrough of how the Decoupled Configuration Driver Model addresses the failure modes documented in Section 2, rather than reporting empirical outcomes from a completed deployment; the model is presented as an architectural proposal, and any scenarios described below are illustrative rather than measured results.

Under a traditional cutover approach, a single routing defect discovered after full cutover requires either an emergency full-platform rollback, itself a high-risk operation, or manual intervention under time pressure while orders continue to accumulate. Under the proposed model, the same defect would be constrained to the specific traffic slice in which it was introduced, since other slices continue operating on their already-validated configuration. Automated rollback for that slice alone would restore legacy execution without affecting unrelated order flows, converting what would otherwise be a network-wide incident into a contained, single-slice event.

The parallel telemetry baseline phase is particularly consequential for the class of defects that are hardest to catch through design review alone: those arising from real production edge cases (unusual product

combinations, rare carrier constraints, peak-load timing interactions) that synthetic test traffic does not reproduce. By validating extracted configuration against observed legacy behavior on real traffic before any cutover decision is made, the model shifts defect discovery earlier and reduces the population of defects that can only be found by observing production impact after cutover.

The governance implications are also notable. Laukkanen et al.'s (2016) finding that bottom-up, incremental continuous delivery adoption succeeds where top-down replacement fails in stage-gate managed organizations maps directly onto this model's slice-by-slice cutover structure: each slice constitutes a bounded, independently reviewable change that can pass through existing stage-gate controls without requiring the governance process itself to be restructured around a single high-stakes release event.

The model's principal limitation is that configuration abstraction is only as reliable as the fidelity of the Phase 1 telemetry baseline; edge cases genuinely absent from the observed traffic window will not be captured by extraction, and rare seasonal or promotional traffic patterns may require an extended baseline period to observe. The model also does not eliminate migration effort, it redistributes it: extraction and validation work that would otherwise occur in a compressed pre-cutover period is instead spread across the baseline and abstraction phases, requiring sustained engineering investment over a longer calendar window rather than a shorter, more intense one.

5. Conclusion

Legacy warehouse management modernization is frequently treated as an unavoidable high-risk event because the business logic governing physical fulfillment operations is architecturally fused to platform code that cannot be changed incrementally. This article has presented a Decoupled Configuration Driver Model that separates that logic from platform execution through a versioned, externally validated configuration layer, enabling migration to proceed as a sequence of small, reversible, independently governed steps rather than a single irreversible cutover. Grounded in the continuous delivery, chaos engineering, and microservices migration literature, the model offers enterprise architects a structured alternative to all-or-nothing re-platforming for systems where operational continuity cannot be compromised. Future work should focus on empirically validating the model against a completed enterprise deployment, quantifying the calendar-time and engineering-effort tradeoffs between concentrated and distributed migration effort, and extending the configuration abstraction technique to other classes of legacy enterprise systems beyond warehouse management.

CRediT Author Contribution Statement

Prashant Vikas Patil: Conceptualization, Methodology, Writing – Original Draft, Writing – Review & Editing.

Acknowledgments

The author declares no funding was received for this work. Portions of the manuscript preparation, including reference formatting and prose editing, were assisted by an AI writing tool under the author's direction and review; all technical content, architectural claims, and conclusions are the author's own.

Conflicts of Interest

The author declares no conflicts of interest.

References

1. Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., and Rosenthal, C. (2016). Chaos engineering. IEEE Software, 33(3), 35–41. https://doi.org/10.1109/MS.2016.60
2. de Jong, M., van Deursen, A., and Cleve, A. (2017). Zero-downtime SQL database schema evolution for continuous deployment. 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 143–152. https://doi.org/10.1109/ICSE-SEIP.2017.5
3. Gu, J., Goetschalckx, M., and McGinnis, L. F. (2007). Research on warehouse operation: A comprehensive review. European Journal of Operational Research, 177(1), 1–21. https://doi.org/10.1016/j.ejor.2006.02.025
4. Guo, Y., Spínola, R. O., and Seaman, C. (2016). Exploring the costs of technical debt management – a case study. Empirical Software Engineering, 21(1), 159–182. https://doi.org/10.1007/s10664-014-9351-7

5. Hasan, M. H., Osman, M. H., Admodisastro, N. I., and Muhammad, M. S. (2023). Legacy systems to cloud migration: A review from the architectural perspective. Journal of Systems and Software, 202, 111702. https://doi.org/10.1016/j.jss.2023.111702
6. Laukkanen, E., Lehtinen, T. O. A., Itkonen, J., Paasivaara, M., and Lassenius, C. (2016). Bottom-up adoption of continuous delivery in a stage-gate managed software organization. Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 1–10. https://doi.org/10.1145/2961111.2962608
7. Martínez Saucedo, A., Rodríguez, G., Gomes Rocha, F., and Pereira dos Santos, R. (2025). Migration of monolithic systems to microservices: A systematic mapping study. Information and Software Technology, 177, 107590. https://doi.org/10.1016/j.infsof.2024.107590
8. Opara-Martins, J., Sahandi, R., and Tian, F. (2016). Critical analysis of vendor lock-in and its impact on cloud computing migration: A business perspective. Journal of Cloud Computing, 5(1), 4. https://doi.org/10.1186/s13677-016-0054-z
9. Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J. M., and Oivo, M. (2017). Continuous deployment of software intensive products and services: A systematic mapping study. Journal of Systems and Software, 123, 263–291. https://doi.org/10.1016/j.jss.2015.12.015
10. Shahin, M., Zahedi, M., Babar, M. A., and Zhu, L. (2019). An empirical study of architecting for continuous delivery and deployment. Empirical Software Engineering, 24(3), 1061–1108. https://doi.org/10.1007/s10664-018-9651-4