

Designing a Dual-Run Migration Architecture for Medicare CRM: Event Synchronization, System-of-Record Transition, and Legacy Platform Retirement

Susil Kumar Sahu

Solutions Engineer Executive Advisor
Elevance Health, Inc.

Cite this paper as: Susil Kumar Sahu (2021). Designing a Dual-Run Migration Architecture for Medicare CRM: Event Synchronization, System-of-Record Transition, and Legacy Platform Retirement. *Frontiers in Health Informatics*, Vol.10, 452-468

Abstract—Today, Medicare sales and customer-management platforms rely on a series of interconnected legacy systems that enable prospect management, quote support, eligibility, broker support, enrollment, communications, identity management and reporting. Customer relationship management (CRM) is a mission-critical application and replacing it is challenging because those upstream and downstream applications rely on the legacy CRM application during transformation. This systematic review is a synthesis of 32 sources of legacy modernization, cloud and microservice migration, interoperability in the health domain, distributed consistency, and data quality, and provides a dual run migration architecture where the two environments of legacy and cloud are in parallel while the business capabilities migrate gradually over time. Ownership of a system of record is explicit per business domain and application programming interfaces and event-driven synchronizations communicate information via canonical data models for the customer, lead, broker, eligibility, quote, application, and enrollment status. The architecture features handling of duplicate-event prevention, transactions idempotency, transaction correlation, retry processing, reconciliation, exception management, and controlled replay. The analytic model suggests that 3 bounded retries lead to a residual delivery failure probability of 1.25×10^{-4} , compared to 0.05 for transient failure and 0.033 for an incremental gateway, and 0.026 for a cold-turkey cutover, on a five-point rating scale. With a system-of-record transition instead of a full legacy retirement, an architectural solution can be adopted to mitigate the operational risk of a large-scale healthcare CRM and cloud transformation.

Keywords—*dual-run migration, Medicare CRM, system of record, event-driven synchronization, idempotency, canonical data model, reconciliation, cloud migration, legacy retirement*

Introduction

These interconnected systems enable sales platforms and customer-management platforms for Medicare to handle prospect management, quote generation, eligibility verification, broker services, enrollment, communications, identity and reporting. Legacy information systems of this nature are vital to mission, expensive to maintain, hard to modify and replacement a constant worry [1]. There are two conflicting philosophies for migration: the one-step cutover (cold-turkey approach) and gradual migration over time, in which the legacy system remains and functionality is gradually transferred over time.

This is in line with the evolution of software. Programs in a changing environment should be changed regularly and these programs tend to become more complex unless effort is made to

keep them simpler [2]. As products, channels and partner systems are continually changing, they expand the scope of integration for the customer relationship management (CRM) system, and every upstream or downstream dependency increases the cost and risk of a single replacement.

Practitioner studies focus on organizational reality of ground modernization, not only technology aspects, through consideration of the perception of legacy systems and software modernization by practitioners [3]. Given these observations, a dual-run approach is suggested in which both legacy and cloud CRM systems can coexist, ownership of a system of record is specific to a business domain, and retirement of the legacy system is separate from ownership transition. The contributions consist of a reference architecture, an analytic model of the synchronization behavior, and a phased transition model. The rest of the paper is organized as follows: Section II outlines the review method, Section III summarizes related work, Section IV presents the architecture, Section V presents the phased transition, Section VI presents the evaluation and Section VII presents the discussion.

Review Methodology

The study is design-science, which involves creating a purposeful artifact and assessing it in light of a relevant problem. A set of seven design-science research guidelines, which focus on the relevance of the problem, design of the artifact, evaluation, rigor, and communication of the results, structure the work [4]. The artifact is a migration architecture, including its synchronization, reconciliation and transition processes.

A systematic review of the supplied literature is combined with the design-science activities. The adopted process model consists of six activities: problem identification, setting up goals, design and development, demonstration, evaluation and communication [5]. Thirty-two sources were identified and grouped into 8 themes as shown in Table I and each source was given a code for the architectural concern that it addresses.

TABLE I. DISTRIBUTION OF REVIEWED SOURCES BY THEME

Theme	Sources	Share (%)	Reference Numbers
Legacy systems and evolution	3	9.4	[1]–[3]
Research methodology	2	6.3	[4], [5]
Healthcare interoperability	4	12.5	[7]–[9], [16]
Cloud computing and migration	4	12.5	[6], [10]–[12]
Service and microservice migration	9	28.1	[13]–[15], [25]–[28], [31], [32]

Theme	Sources	Share (%)	Reference Numbers
Distributed consistency and events	6	18.8	[19]–[24]
API and web architecture	2	6.3	[17], [18]
Data quality	2	6.3	[29], [30]
Total	32	100.0	

Note: Percentages may not sum exactly because of rounding.

There is a methodological precedent for this, in a previous systematic review that selected 23 primary papers to characterise the state of the field and open challenges of migration to the cloud [6]. A similar screening logic was used in a more limited fashion: sources were included because they reported to the existence of old and new systems, data movement, consistency, or controlled retirement, and the evidence base is comprised of publications between 1978 and 2019.

Background and Related Work

Healthcare Platforms and Interoperability

For years, health information technology (HIT) has been blamed for shoving providers into a single-fix-it application. This used to be the rationale behind “substitutable applications on a stable and standardized platform”, which would free applications from that dependency [7]. The same goes for CRM modernization; a platform that can be replaced, capability by capability, does not require a single, irrevocable replacement event.

The digitization of healthcare is already established and can be seen in trends of adoption. After federal incentives, there was a significant increase in the adoption of EHRs among hospitals, from 9.4% to 83.8% between 2008 and 2015 [8]. Adopting this magnitude too quickly generates a big installed base of interdependent systems, and the integration demands remain long after implementation.

Canonical representations are provided by interoperability standards. Health Level Seven Fast Healthcare Interoperability Resources (HL7 FHIR) [9] brings together a resource-based data model with RESTful exchange, enabling healthcare information to be accessed via a common interface. The principles used to design the canonical customer, eligibility, and enrollment resources for CRM synchronization are the same as those used for clinical data.

Cloud Computing and Migration

The National Institute of Standards and Technology (NIST) definition of cloud computing defines five essential characteristics, three service models and four deployment models [10] that provide a shared vocabulary for the target platform. A cloud CRM is usually a software service, which implies that the operational model for the intended environment is shaped by on-demand self-service, a wide range of access to networks and measured services.

Three new facets of cloud computing were detected in an early synthesis: the emergence of unlimited resources, the elimination of upfront commitment and the provision of pay for use on a short term basis [11]. These factors are conducive to incremental adoption because the new capability can be added on top of the existing investment.

An existing application will have to be adapted at multiple layers of its architecture if it is to be migrated. Problems and solutions about adapting applications to the cloud environment were centralized in these layers [12] and that seemed to be a good starting point for splitting up a CRM platform into the individual parts that can be migrated separately.

Service and Microservice Migration

Loosely coupled, interface-defined services became the building blocks of enterprise applications with the advent of service-oriented computing [13]. The use of the capability-based decomposition adopted here is fundamental and is based on the notion of services with explicit contracts, such as lead management, quoting, eligibility, broker services, and enrollment. Another advantage of explicit contracts is that the legacy and cloud versions of a capability can be replaced under the same contract.

Small services that can be deployed independently and are based on business capabilities are called microservices [14]. Independent deployability is the feature that enables one CRM function to be deployed to the cloud—the others to the legacy system. The data belongs to each such unit and the data ownership is consistent with data system-of-record assignment developed in Section IV.

In a systematic grey literature review of 51 industrial sources, the benefits and costs of microservices were discussed [15]. Maintainability and delivery were cited as gains, while complexity of distributed systems and the complexity of data management were cited as pains, with the latter being directly applicable to dual-run synchronization. The chronology of these concepts is followed in Fig. 1.

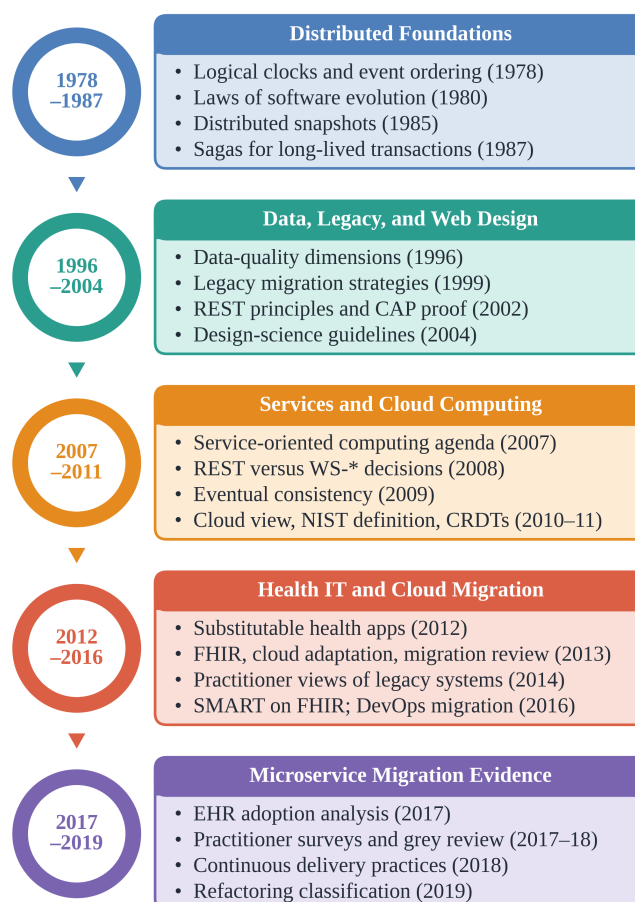


Fig. 1. Chronological development of the concepts underlying dual-run migration, synthesized from the reviewed literature.

Dual-Run Migration Architecture

System-of-Record Ownership and API Mediation

The architecture also places an application programming interface (API) mediation layer between the consuming applications and both the CRM environments. A standard-based apps platform for electronic health records (EHR) was successfully used to prove that apps can be developed and deployed with a consistent interface and across disparate back ends [16]. Similarly, upstream channels and downstream systems are for one CRM interface, with routing to the legacy or cloud platform handled behind the interface, as shown in Fig. 2.

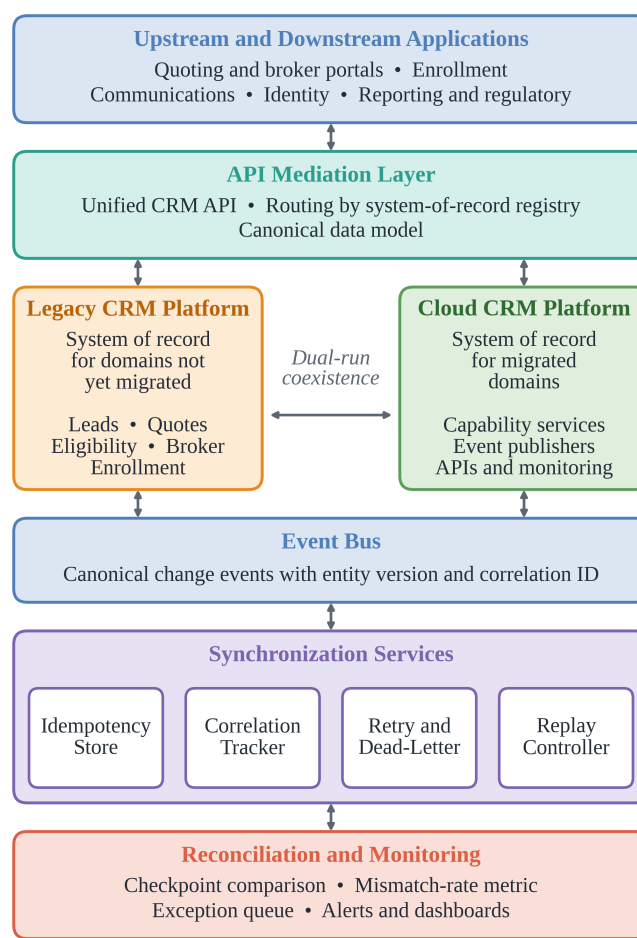


Fig. 2. Reference architecture for dual-run operation of legacy and cloud CRM platforms.

The constraints of the representational state transfer (REST) style are adhered in order to design an interface, which includes client-server separation, statelessness, cacheability, uniform interface, and layered composition [17]. A uniform interface makes it possible to insert a routing component without changing clients, which is what is leveraged in the process of progressive re-pointing from legacy platform to cloud platform.

RESTful and web services standards (WS-*) are compared on the architectural level, where REST is preferred for ad hoc integration for simplicity and loose coupling, and WS-* is for advanced quality-of-service requirements [18]. Resource oriented APIs are thus used by synchronous queries and commands, while state changes are transmitted asynchronously as events. Each business domain is owned separately in a separate registry, and the ownership sequence is summarized in Table II.

TABLE II. SYSTEM-OF-RECORD OWNERSHIP SEQUENCE AND CANONICAL ENTITIES

Domain	Canonical Entities	Wave	Downstream Dependency
Lead management	Customer, lead	1	Narrow; few readers
Broker services	Broker, appointment	2	Moderate; quoting and enrollment read
Eligibility	Eligibility result	3	Moderate; quoting reads
Quoting	Quote	4	Broad; broker portal and enrollment
Enrollment	Application, enrollment status	5	Broadest; communications, reporting, regulatory

Note: Dependency breadth is a qualitative judgment used to order the waves.

Event Synchronization and Ordering

Changes to state are reported as unchangeable events that contain a canonical payload, a source identifier and an entity version. The physical clocks that are used in distributed components are not guaranteed to agree, so ordering is achieved by using a happened-before relation, in which logical clocks assign timestamps that respect causality [19]. A consumer only applies event e to entity x_e , if v_e is greater than the last applied version $V[x_e]$ (as in (1)).

$$A(e) = \begin{cases} 1, & \text{if } v_e > V[x_e] \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

For consistency verification, a consistent view across both platforms is needed. The distributed snapshot algorithm showed that it is possible to record a consistent global state in a distributed system without interrupting the computation and using marker messages to define the range of events before and after the snapshot [20]. The event streams of both platforms are marked with an analogous checkpoint marker, so that record sets that are compared for reconciliation occur at the same logical point. The marker based cut does not require a platform stop, thus maintaining availability for validation.

Consistency, Idempotency, and Replay

Eventual consistency ensures that, once a new update is made, all replicas will eventually agree on the new value and that there was a window during which the updates were not visible to all replicas [21]. The latency in dual-run is the sum of the publication latency and broker latency and consumption latency which are described in (2) in seconds, respectively.

$$T_c = L_p + L_b + L_c \quad (2)$$

The CAP conjecture is the statement that it is impossible to guarantee all three properties of

consistency, availability and partition tolerance in an asynchronous network in general, which means that at least two of them can be guaranteed [22]. Partitions between legacy and cloud environments are not excluded, so that the architecture is more resilient to loss of availability for customer-facing operations and relies on reconciliation to restore consistency; the submission of enrollments are under stricter ownership rules. If there is a divergence accepted in partition, the reconciliation result will then be considered as the right path to correct the divergence.

Conflict-free replicated data types proved that even in the case of merge operations that are commutative, associative, and idempotent, it is possible to have replicas converge without coordination [23]. The idempotency property is adopted directly, but not the same for the types as for the records in CRM. The value of the determinant key is constructed from the value of the source system s_e , entity identifier x_e , and version v_e , as displayed in (3) where $\parallel$ denotes concatenation and H is a hash function. The key of events already present in the idempotency store are discarded which allows events in the store to always be safely replayed, and the failure modes covered are summarized in Table III.

$$k_e = H(s_e \parallel x_e \parallel v_e) \quad (3)$$

TABLE III. FAILURE MODES AND CORRESPONDING SYNCHRONIZATION CONTROLS

Failure Mode	Control	Mechanism
Duplicate delivery	Idempotent consumption	Key k_e checked in the idempotency store
Out-of-order delivery	Version check	Per-entity version compared with $V[x_e]$
Lost or delayed event	Retry with backoff	Bounded exponential delay d_n
Persistent failure	Dead-letter queue	Exception record, review, and replay
Cross-system drift	Reconciliation	Checkpoint comparison and rate ρ
Partial business transaction	Compensation	Correlation identifier and saga steps

Transaction Correlation and Compensation

Long-lived business transactions (such as an enrollment that includes quoting, eligibility, and submission) cannot cross platform. The saga model breaks down such a transaction into its sub-transactions and then creates a compensating transaction to semantically reverse the effect of the sub-transactions [24]. There is a correlation identifier that correlates all the events of a business transaction for compensation and diagnosis. Delivers are retried starting with a

minimum delay of d_0 seconds, but with delay increasing exponentially with the number of attempts, n , but capped at a maximum delay of d_{max} seconds as defined in (4). When the redo budget is consumed, events go to a DLQ, see Fig. 3.

$$d_n = \min(d_{max}, d_0 \cdot 2^{n-1}) \quad (4)$$

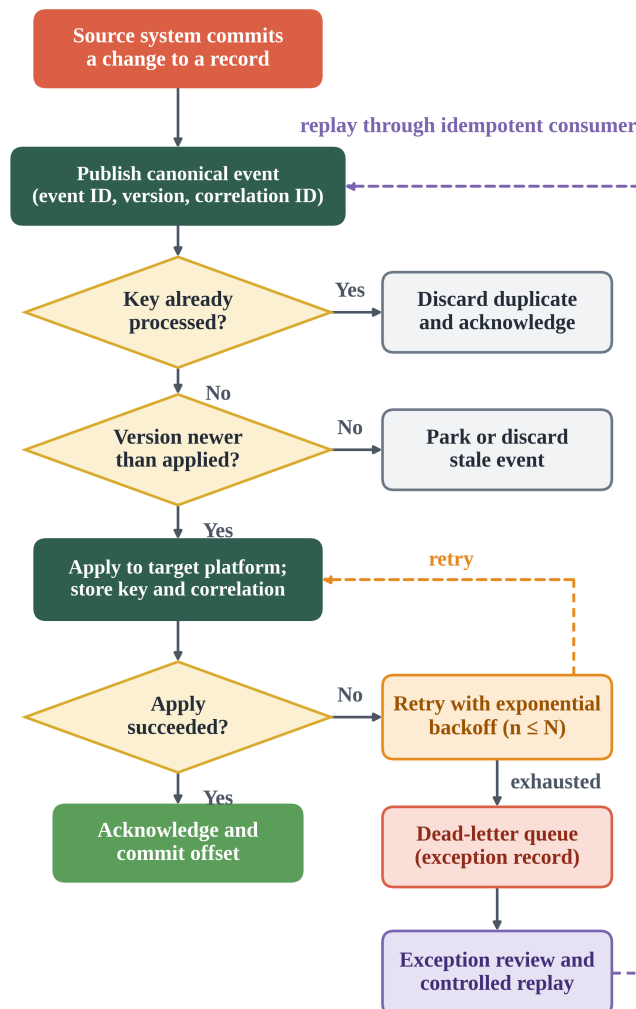


Fig. 3. Event-processing flow with duplicate suppression, version check, retry, dead-letter handling, and controlled replay.

Phased Capability Transition

It is reported that incremental migration has been used in industry. A report from a cloud experience to transform an application to a cloud-native microservice architecture reported a step-by-step method that involved gradually moving parts of the functionality over time while continuing to run the system, with the help of automated delivery pipelines [25]. The same incremental approach applies to the evolution of CRM capabilities, which unfold in the eight steps and five waves listed in Fig. 4. The marginal effort of later waves is expected to decline, as each wave is using the same delivery pipeline.

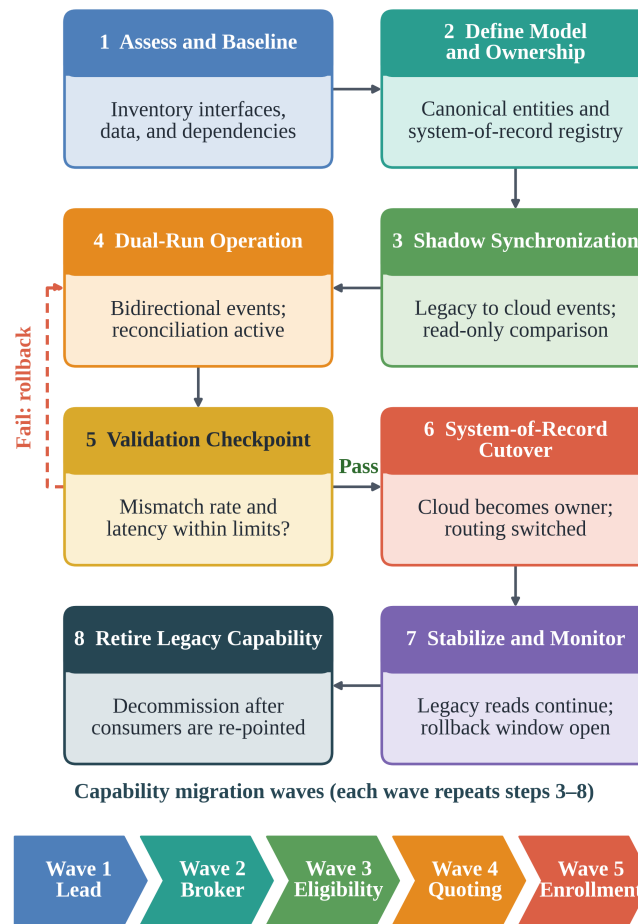


Fig. 4. Phased capability-transition process with validation checkpoint, rollback path, and migration waves.

The motivations and problems associated with migration have been discussed with 21 experienced practitioners [26] using interviews. The reported reasons for migration ranged from maintainability, scalability, and delivery speed and issues faced highlight the size of the migration effort. The wave sequence in Table IV is thus arranged such that the narrow dependency capable migrations precede those with broad dependency capabilities.

A survey of industrial practitioners, based on 18 practitioners, described the practice of migration through the reverse engineering and migration, transformation and forward engineering phases [27]. The proposed model shows these phases at a smaller scale: The existing behaviour is recovered during assessment, is transformed in the model and re-implemented in the cloud platform with validation checkpoints prior to cutover. Thus, validation checkpoints are a mixture of functional parity tests and mismatch-rate criterion and a latency criterion based on the inconsistency window.

A classification of refactoring approaches for decomposing monolithic systems was reported when transitioning towards microservices [28]. Since the decomposition here is done by business capabilities, progress is measured in terms of the weighted coverage in (5), where w_i is the weight of business capability i , $m_i \in [0,1]$ is the fraction of capability i migrated and C is a number between 0 and 1. Table IV gives illustrative weights and exit thresholds.

$$C = \frac{\sum_{i=1}^5 w_i m_i}{\sum_{i=1}^5 w_i} \quad (5)$$

TABLE IV. ILLUSTRATIVE WAVE PLAN WITH WEIGHTS AND EXIT THRESHOLDS

Wave	Capability	Weight w_i	Coverage C	Exit Threshold ρ (%)
1	Lead management	0.20	0.20	≤ 0.50
2	Broker services	0.15	0.35	≤ 0.30
3	Eligibility	0.20	0.55	≤ 0.20
4	Quoting	0.25	0.80	≤ 0.10
5	Enrollment	0.20	1.00	≤ 0.05

Note: Weights and thresholds are planning values, not measurements; coverage assumes each completed wave has $m_i = 1$.

Reconciliation and Comparative Evaluation

Reconciliation and Data Quality

Reconciliation is presented as a Data Quality exercise. Data quality was described from the user's point of view using 15 dimensions that are classified into four categories: intrinsic, contextual, representational, and accessibility [29]. The mismatch rate in (6) is a measure of accuracy and completeness of synchronized records after checkpointing with a coordinated checkpoint, with M denoting the set of records that mismatch between platforms and R denoting the set of records being compared. The dimensions most directly impacted by synchronization are completeness, accuracy, and timeliness, each of which is tracked for each entity type.

$$\rho = \frac{|M|}{|R|} \times 100\% \quad (6)$$

Methodologies for data quality assessment and improvement were compared in a survey that distinguishes assessment from improvement activities [30]. Assessment results guide exception management, in which each unresolved mismatch is classified, assigned to an owner, and corrected through controlled replay. The reliability of retried delivery is expressed by the cumulative wait in (7) and the residual failure probability in (8), where p is the probability of a transient failure per attempt. For $p = 0.05$, $d_0 = 1$ s, and $d_{\max} = 60$ s, three attempts leave a residual probability of 1.25×10^{-4} , as shown in Table V and Fig. 5.

$$D_N = \sum_{n=1}^N d_n \quad (7)$$

$$P_N = p^N \quad (8)$$

TABLE V. RETRY POLICY OUTCOMES FOR INITIAL DELAY OF 1 S AND CEILING OF 60 S

N	d_N (s)	D_N (s)	$P_N, p = 0.05$	$P_N, p = 0.20$
1	1	1	5.00×10^{-2}	2.00×10^{-1}
2	2	3	2.50×10^{-3}	4.00×10^{-2}
3	4	7	1.25×10^{-4}	8.00×10^{-3}
5	16	31	3.13×10^{-7}	3.20×10^{-4}
7	60	123	7.81×10^{-10}	1.28×10^{-5}
8	60	183	3.91×10^{-11}	2.56×10^{-6}

Note: Values computed from (4), (7), and (8); N is the number of attempts.

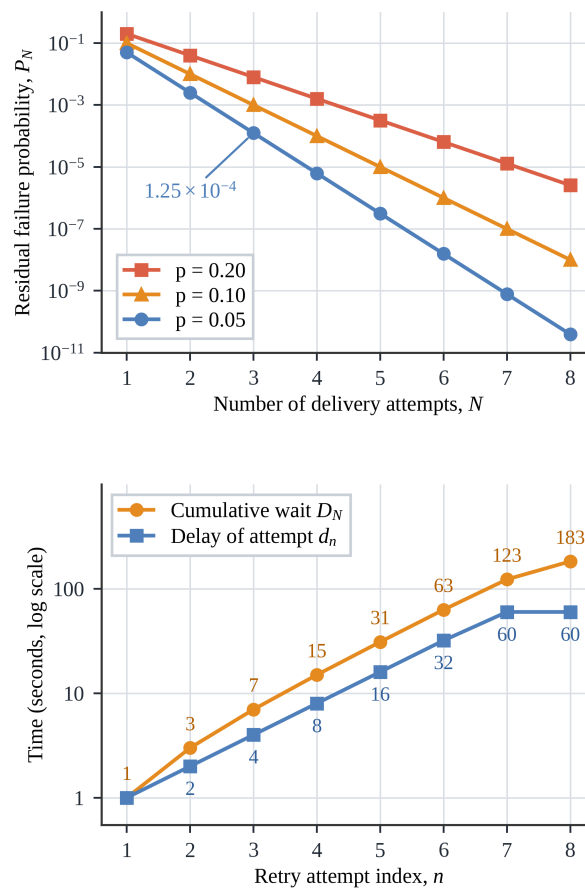


Fig. 5. Residual delivery failure probability (upper) and retry delay profile (lower) under bounded exponential backoff.

Comparative Evaluation

Another factor affecting strategy choice is delivery practice—when delivery is continuous and based on DevOps, release cycles become shorter and small, frequent changes are supported [31]. The weighted score is computed by (9), with the weights w_i of the criteria and the ratings s_{ij} of the strategies, based on the ordinal rating in Table VI, which is used to compare 3 strategies for 5 weighted criteria. The dual-run architecture achieves highest score of 4.05, but at the cost of delivery speed and effort of implementation as compared to incremental gateway (3.30) and cold-turkey cutover (2.60) as illustrated in Fig. 6. The weights are indicative of the importance of continuity and reversibility in a mission critical environment and sensitivity to alternative weights should be explored prior to adoption.

$$S_j = \sum_{i=1}^5 w_i s_{ij} \quad (9)$$

TABLE VI. ORDINAL RATINGS AND WEIGHTED SCORES OF MIGRATION STRATEGIES

Criterion	Weight	Cold-Turkey	Incremental Gateway	Dual-Run
Business continuity	0.30	2	4	5
Rollback capability	0.25	1	3	5
Consistency control	0.20	3	3	4
Delivery speed	0.15	5	3	2
Implementation effort	0.10	4	3	2
Weighted score S_j	1.00	2.60	3.30	4.05

Note: Ratings are ordinal judgments synthesized from the reviewed literature (5 = strongest); the weights are assumed.

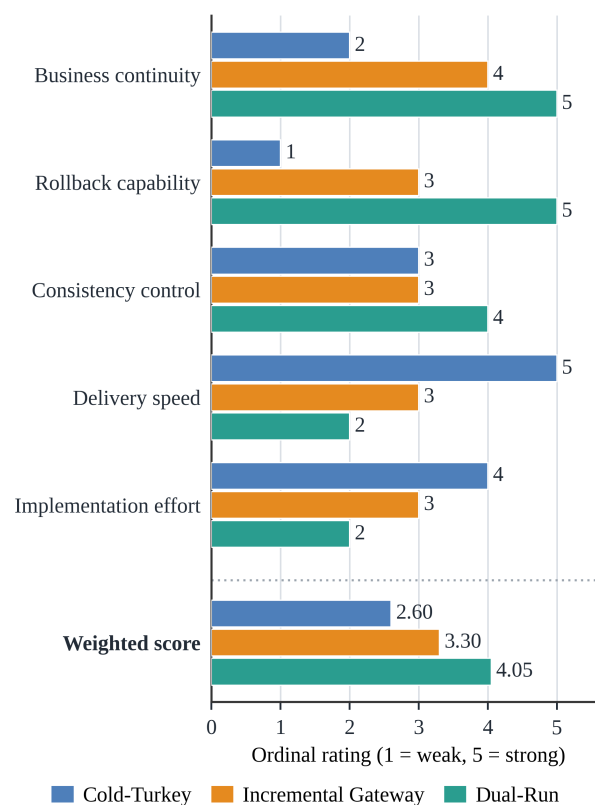


Fig. 6. Ordinal ratings and weighted scores of three migration strategies.

Discussion

Issues noted along the microservice journey such as handling distributed data and tooling

maturity have remained relevant to dual-run migration [32]. As seen in the synthesis, the evidence base for each design element is well-developed, but there is no source directly addressing dual-run operation of a healthcare CRM, so the ratings, weights, and probabilities applied to the analytic model are planning parameters that would need to be calibrated against production telemetry. However, the costs of duplicated infrastructure and reconciliation effort also need to be considered for dual-run operation, as does the reduction in cutover risk. From the operational point of view, the following three conditions enable a retirement of a legacy capability: all consumers have been re-pointed, the rollback window has been closed, and reconciliation within threshold. The literature reviewed is correlated with the architectural concepts in Table VII.

TABLE VII. MAPPING OF REVIEWED LITERATURE TO ARCHITECTURAL CONCEPTS

Architectural Concept	Sources	Contribution to the Architecture
Legacy risk and evolution	[1]–[3]	Motivates incremental replacement
Review and design method	[4]–[6]	Structures the artifact and its evaluation
Healthcare interoperability	[7]–[9], [16]	Canonical resources and API mediation
Cloud target environment	[10]–[12]	Operational model of the target platform
Service decomposition	[13]–[15], [25]–[28]	Capability-based migration waves
API design	[17], [18]	Uniform interface and traffic routing
Ordering and consistency	[19]–[23]	Versioning, convergence, idempotency
Transaction correlation	[24]	Saga compensation and diagnosis
Data quality	[29], [30]	Mismatch rate and exception management
Delivery practice and challenges	[31], [32]	Release automation and residual risks

Conclusion

A dual-run migration strategy was suggested to replace a Medicare CRM platform without a

single cutover. During the five waves, capabilities migrate from legacy to cloud environments while maintaining the ability for legacy environments to coexist through explicit system-of-record ownership, canonical event synchronization, idempotent consumption, correlation-based compensation and checkpoint-based reconciliation. The analytic model predicts that the residual delivery failure rate is 1.25×10^{-4} at 0.05 transient failure rate after three bounded retries, versus a transient failure rate of 0.05 and a residual failure rate of 6.60×10^{-4} for a cold-turkey cutover, with a weighted score of 4.05 for the dual-run approach compared to 2.60 for the cold-turkey cutover. System-of-record transition is not the same as legacy retirement, so risk is less because consumers won't roll back until after transition. The model parameters should be calibrated using production measurements, the model be extended to other domains and replay governance should be assessed under regulatory audit constraints.

References

- [1] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: Issues and directions," *IEEE Softw.*, vol. 16, no. 5, pp. 103–111, 1999, doi: 10.1109/52.795108.
- [2] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proc. IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980, doi: 10.1109/PROC.1980.11805.
- [3] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen, and J. Hage, "How do professionals perceive legacy systems and software modernization?," in *Proc. 36th Int. Conf. Softw. Eng.*, ACM, 2014, pp. 36–47, doi: 10.1145/2568225.2568318.
- [4] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.
- [5] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *J. Manage. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, 2007, doi: 10.2753/MIS0742-1222240302.
- [6] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud migration research: A systematic review," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 142–157, 2013, doi: 10.1109/TCC.2013.10.
- [7] K. D. Mandl and I. S. Kohane, "Escaping the EHR trap—The future of health IT," *N. Engl. J. Med.*, vol. 366, no. 24, pp. 2240–2242, 2012, doi: 10.1056/NEJMp1203102.
- [8] J. Adler-Milstein and A. K. Jha, "HITECH Act drove large gains in hospital electronic health record adoption," *Health Aff.*, vol. 36, no. 8, pp. 1416–1422, 2017, doi: 10.1377/hlthaff.2016.1651.
- [9] D. Bender and K. Sartipi, "HL7 FHIR: An agile and RESTful approach to healthcare information exchange," in *Proc. 26th IEEE Int. Symp. Comput.-Based Med. Syst.*, IEEE, 2013, pp. 326–331, doi: 10.1109/CBMS.2013.6627810.
- [10] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, NIST Special Publication 800-145, 2011, doi: 10.6028/NIST.SP.800-145.
- [11] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010, doi: 10.1145/1721654.1721672.
- [12] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to adapt applications for the cloud environment: Challenges and solutions in migrating applications to the cloud," *Computing*, vol. 95, no. 6, pp. 493–535, 2013, doi: 10.1007/s00607-012-0248-2.

- [13] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007, doi: 10.1109/MC.2007.400.
- [14] N. Dragoni *et al.*, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Springer, 2017, pp. 195–216, doi: 10.1007/978-3-319-67425-4_12.
- [15] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, "The pains and gains of microservices: A systematic grey literature review," *J. Syst. Softw.*, vol. 146, pp. 215–232, 2018, doi: 10.1016/j.jss.2018.09.082.
- [16] J. C. Mandel, D. A. Kreda, K. D. Mandl, I. S. Kohane, and R. B. Ramoni, "SMART on FHIR: A standards-based, interoperable apps platform for electronic health records," *J. Am. Med. Inform. Assoc.*, vol. 23, no. 5, pp. 899–908, 2016, doi: 10.1093/jamia/ocv189.
- [17] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, 2002, doi: 10.1145/514183.514185.
- [18] C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful web services vs. 'big' web services: Making the right architectural decision," in *Proc. 17th Int. Conf. World Wide Web*, ACM, 2008, pp. 805–814, doi: 10.1145/1367497.1367606.
- [19] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978, doi: 10.1145/359545.359563.
- [20] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 1, pp. 63–75, 1985, doi: 10.1145/214451.214456.
- [21] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, 2009, doi: 10.1145/1435417.1435432.
- [22] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002, doi: 10.1145/564585.564601.
- [23] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Stabilization, Safety, and Security of Distributed Systems*, X. Défago, F. Petit, and V. Villain, Eds. Springer, 2011, pp. 386–400, doi: 10.1007/978-3-642-24550-3_29.
- [24] H. Garcia-Molina and K. Salem, "Sagas," *ACM SIGMOD Rec.*, vol. 16, no. 3, pp. 249–259, 1987, doi: 10.1145/38713.38742.
- [25] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, 2016, doi: 10.1109/MS.2016.64.
- [26] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 22–32, 2017, doi: 10.1109/MCC.2017.4250931.
- [27] P. D. Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in *Proc. 2018 IEEE Int. Conf. Softw. Archit.*, IEEE, 2018, pp. 29–39, doi: 10.1109/ICSA.2018.00012.
- [28] J. Fritzsche, J. Bogner, A. Zimmermann, and S. Wagner, "From monolith to microservices: A classification of refactoring approaches," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, J.-M. Bruehl, Ed., Springer, 2019, pp. 1–12, doi: 10.1007/978-3-319-98888-8_1.

- M. Mazzara, and B. Meyer, Eds. Springer, 2019, pp. 128–141, doi: 10.1007/978-3-030-06019-0_10.
- [29] R. Y. Wang and D. M. Strong, “Beyond accuracy: What data quality means to data consumers,” *J. Manage. Inf. Syst.*, vol. 12, no. 4, pp. 5–33, 1996, doi: 10.1080/07421222.1996.11518099.
- [30] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, “Methodologies for data quality assessment and improvement,” *ACM Comput. Surv.*, vol. 41, no. 3, Art. no. 16, 2009, doi: 10.1145/1541880.1541883.
- [31] L. Chen, “Microservices: Architecting for continuous delivery and DevOps,” in *Proc. 2018 IEEE Int. Conf. Softw. Archit.*, IEEE, 2018, pp. 39–46, doi: 10.1109/ICSA.2018.00013.
- [32] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, “Microservices: The journey so far and challenges ahead,” *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, 2018, doi: 10.1109/MS.2018.2141039.