Scientific Publications

Universal Journal of Computer Sciences and Communications, 2021, 1, 1348
www.scipublications.com/journal/index.php/ujcsc
DOI: 10.31586/ujcsc.2021.1348

Review Article

Digital Transformation in Insurance: Migrating Enterprise Policy Systems to .NET Core

Keerthi Amistapuram [1,*]

[1] Net Developer, USA

*Correspondence: Keerthi Amistapuram (keerthi.amistapuram.research@gmail.com)

Abstract: Migrating enterprise policy systems to .NET Core is a key objective of digital transformation in the Insurance IT ecosystem. This change directly addresses strategic drivers: enabling adoption of cloud-first development, resisting market pressure for more flexible and usable enterprise solutions, and preparing for changing demands from regulation and compliance. Phases of operational benefit aligned with risk mitigation form the basis of the migration roadmap, with a strong focus on engaging all relevant stakeholders. Market pressure for a SEAMLESS user experience across ALL applications is a fundamental driver for Investment in digital transformation. Gaps remain in enterprise Operations, where Legislative and regulatory accountability Demand rigid and complex solutions that Liberty has not yet been able to provide. New risk-based capital requirements, Data-Sovereignty controls, Controls for sensitive Data in the Cloud, and new Audit requirements create a long list of challenges for the ecosystem that can no longer be Deferred. At the same time, Cross-organisational integration is becoming more important and integrating partners from the insurance supply-chain requires a much more flexible approach to development and Deployment. These factors combine to generate a credible case for accelerated digital investment with a focus on Migration to Cloud Platforms, with related Risk mitigation, Quality Improvements, and flexibility benefits that close Industry gaps.

Keywords: Digital Transformation, .NET Core Migration, Insurance IT Ecosystem, Cloud-First Development, Enterprise Policy Systems, Regulatory Compliance, Risk Mitigation, Operational Efficiency, Seamless User Experience, Data Sovereignty, Risk-Based Capital, Audit Controls, Cloud Platforms, Cross-Organizational Integration, Flexible Deployment, Supply-Chain Integration, Quality Improvement, Enterprise Modernization, Compliance Automation, Digital Investment

How to cite this paper:
Amistapuram, K. (2021). Digital Transformation in Insurance: Migrating Enterprise Policy Systems to .NET Core. *Universal Journal of Computer Sciences and Communications, 1*(1), 1–17.
DOI: 10.31586/ujcsc.2021.1348

Received: July 23, 2021
Revised: November 16, 2021
Accepted: December 20, 2021
Published: December 27, 2021

1. Executive Overview

Enterprise policy systems are the heart of every insurer. These systems manage the life-cycles of policies and contracts, provide transactional capabilities, and handle reporting, business intelligence, regulatory, and government communication functions. There are both external and internal market pressures to modernize these enterprise capabilities. Market competition and rising operational costs are forcing insurers to reduce pricing, while increased consumer expectations are requiring insurers to enhance service levels during the products' life-cycles. Regulatory bodies are mandating higher and more stringent levels of information requirements; this is increasing the burden of providing such information to regulators. The current solutions that form the enterprise policy systems are typically very old code bases (some in excess of 30 years of continuous running) whose maintenance costs are prohibitive, and whose agility to implement changes is simply too slow. The current strategy is to migrate these systems from their traditional architectures to modern Cloud friendly Microservice architectures and languages. All aspects of the full function of the enterprise policy systems will either

be migrated or re-developed over time. The value considered while undertaking changes is not simply the cost reduction, but the enabling of sales. If a policy can be sold into the market today, the insurer must therefore ”sell” the product – even if the final revenue does not cover the full cost of accessing the market – for a future policy holders best serve on ”best-price” basis. Remaining positioned ”off-market” until the policy can be offered through the traditional architectural routes simply puts the insurer at risk as last price insurance is the fastest to sell! [1].

1.1. Strategic Drivers of Digital Transformation in Insurance

Market pressures, regulatory expectations, and competitive positioning justify migrating enterprise Policy Systems and associated Operational Ecosystem from Legacy Technology to .NET Core, with justifiable benefits articulated in Section 2.2. Stakeholders responsible for specific areas of the system are engaged, with expected quality assurance and deployment risk mitigation. The migration is appropriately phased to provide early operational and cost benefits; subsequent digital transformation of enterprise policy sub-systems occurs as business value dictates. A wide range of Market, Regulatory and Compliance, and Business pressures for both immediate and long-term delivery of Digital Services applies across the business landscape. Digital and Data have been identified as key enablers of these changes for the Insurance arm of the organisation, supported by an investment program that has defined multiple products for implementation. A Direct-to-Consumer (DTC) product is currently under construction and planned for launch in 2019. Use of microservices across the Insurance Sales Services, Insurance Servicing Services, and Insurance Claims Services is planned. Additionally, the capital and operating cost models are being reviewed, with a key driver being improved incident volumes. Implementation on microservices is expected to drive down incidents as compared with monolithic-based solutions, while also allowing for a more efficient operation model aligned with current DevOps principles [2].

Figure 1. Migration of Enterprise Policy Systems to .NET Core with Microservices and DevOps Efficiency

1.2. Benefits and Risks of Migrating to .NET Core

Migrating .NET Enterprise Policy Systems to .NET Core is a critical component of an organization’s digital transformation strategy. .NET Core .NET provides an accelerative

approach for development and operations through its cross-platform, microservices, and cloud-ready architecture. Development teams derive the development experience and library ecosystem of Programming Language while cloud-related teams gain platform advantage with Azure and its supporting services. The intangible benefits of migrating to .NET Core include the easing of labour market constraints for development and operations resources. The shortage of labour is expected to grow in that direction in the medium and longer term as demand for cloud-related skill sets soars. Other sources of intangible benefits include the ability to meet changing requirements efficiently and the improved prospects for truly DevOps-enabled processes [3]. The migration to .NET Core is not without its risks, however. Testing and validation environments still need to open-up for independent external access. Some forms of automated end-to-end testing for business functions are not yet completely established, while telemetry for production systems is in its early stages. Operating environments are still new and have not yet gone through a complete application lifecycle. The transition to cloud-based operating environments may incur additional licensing costs or other complexities associated with hosting on third-party infrastructure in overseas locations. The release cutover and validation approaches are as yet unproven. These risks, however, are risks typical with a major technology conversion, and are not expected to be more than manageable for an organisation of this scale [4].

	Metric	Value
0	Performance Gain (base)	120.0%
1	Latency Reduction (base)	50.0%
4	Availability (Monolith)	98.901%
6	Reliability/Availability Enhancement (ΔA)	1.077%
8	ROI vs CAPEX	-52.4%

2. Current State of Enterprise Policy Systems

Enterprise Policy Systems enable the full lifecycle of enterprise insurance products—quote management, underwriting support, issuance, billing, and servicing—thereby covering an entire business unit and their clients. The deployment ecosystem consists of a Windows-hosted sub-system that manages user interactions and a LINUX-hosted sub-system that handles the core business logic. The sub-systems are independently developed and used mainly by brand affiliates/platforms to support their respective products. Both sub-systems are document-driven and manage a rich set of business and product rules. While the sub-systems are of critical importance/volume, there are major pain points that warrant migration to .NET Core. The sub-systems are architected as monoliths, leading to several issues through the years:

- Deployments necessitate the upgrade of the entire platform, even for small non-critical changes.
- Scaling is enabled only through server specifications as opposed to horizontal scaling through additional instances. Cache is implemented at the application level which contradicts the idea and efficacy of scaling out [5].
- Debugging and fix-release cycles are cumbersome and unreliable partly due to lack of unit tests and partly due to singularity and complexity of the deployed artefacts.

Changes done by one team/functional area may inadvertently affect the working of others leading to major failures during production processing. Such failures have at times led to major delivery issues. The technologies being used are also considerably old and the architecture is not in support of DevOps practices leading to high continuous

operations and maintenance (O&M) effort. The target architecture depicted below addresses these challenges/concerns [6].

Equation 1 — Migration Performance Gain (PG)

Goal: quantify throughput gain after moving to microservices/.NET Core.
Definition:

$$PG = TPSmonoTPSms - TPSmono \tag{1}$$

where TPS = transactions per second.
Scaling (instances):

$$TPSms(i) = mini \cdot TPSbase,\ bottleneck \tag{2}$$

2.1. Legacy Architecture and Limitations

Monolithic design limits enterprise policy systems' scalability, deployability, and maintainability. Services encapsulating distinct capabilities can be designed, built, deployed, and operated independently, reducing the size and complexity of change. Consolidating data ownership in manageable sub-systems mitigates cross-cutting data-access woes, allowing schema migrations to be treated as code changes rather than a travelled support nightmare. An industry-mandated and risk traced data strategy governs data compliance, auditability, sovereignty, and archiving. Transforming enterprise policy systems from monolithic code bases to a service-based eco-system alleviates both depth and breadth of technical debt. Existing enterprise policy systems operate as tightly coupled monoliths. The primary deployable unit is a build artefact containing over 700 files and exceeding 100MB. Deployments are low-frequency, out-of-hours, whole-of-environment events encompassing upwards of 1800 files and varying in duration due to changing configuration-based process flows. Tightly coupled change requires extensive testing of enterprise policy systems and their many dependence on shared services, which provide less development and operations support than required. Service development is often not supported by a business case. Scalability and maintainability of enterprise policy systems and their supporting shared services are becoming pressing issues [7].

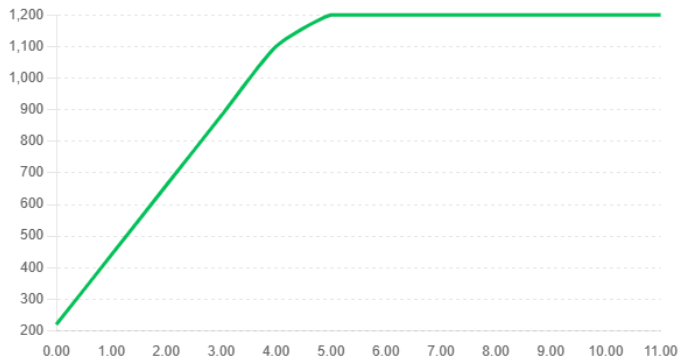

Figure 2. Throughput vs Instances (Microservices)

2.2. Regulatory and Compliance Considerations

Regulatory obligations form a complex web for insurers and reinsurers, increasingly pressing business travel toward the cloud. Data sovereignty affects storage choices while auditability, reporting disciplines, and scientific integrity require governing designs. Explicitly integrating compliance into a target architecture, therefore, de-risks delivery of immediate value [8]. Regulatory organisations and standard-setting bodies impose

various requirements on insurers and reinsurers. In some cases, controls can be mandated outside a company's IT ecosystem. The National Transport Safety Board's (NTSB) principles on transparency of public-use flight data for UAS operations, written 1 August 2018, mirror those in the General Data Protection Regulation (GDPR) and recommend: "Data collected should be made readily available in a common format for postflight analysis and evaluation without unapproved user modifications." NTSB principles are, of course, advisory only. However, the prioritisation of data provision and accessibility mirrors the need to explicitly integrate convenience of reporting and auditing into the target architecture of a company that must meet such requirements [9].

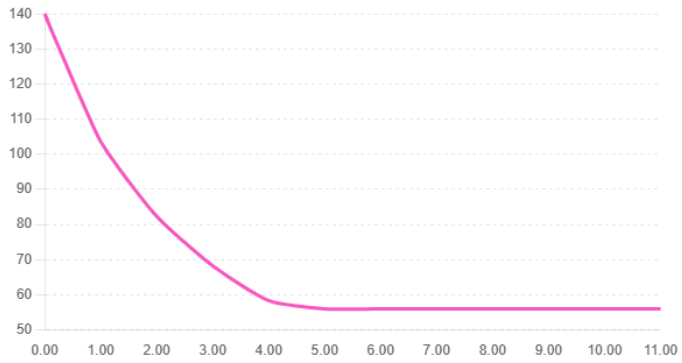

Figure 3. P95 Latency vs Instances

3. Target Architecture with .NET Core

A high-level overview of the migration strategy identifies risks and benefits of moving from legacy enterprise policy systems to .NET Core. Enterprise policy systems encompass many business-critical subsystems in the areas of auto, travel, and casualty that support insurance policyholder lifecycle operations and also provide business reports. Much of the enterprise technical environment is hospitable to the migration. Stakeholders are willing and able to change the policy subsystems and apply a phasing strategy aligned with business value and risk. Moving the policy subsystems to strategic digital microservices represents the first important step to enable transition of the technical ecosystem to the future state with .NET 5. Analyses of legacy architecture and regulatory compliance considerations articulate why digital transformation in the insurance space demands replacing these enterprise policy subsystems. The migration to .NET Core will be modeled using decomposition into a number of independent, self-contained services with the side benefits of enabling containerization of deployments and future change parallelism. Changes to support security, observability, and resilience will also be considered [10].

Equation 2: Code Refactoring Efficiency (CRE)

Goal: efficiency of refactor work measured by complexity reduction per hour.
Let C_b and C_a be average cyclomatic complexity before/after;
H developer hours.

$$CRE = HCb - Ca \tag{3}$$

3.1. Microservices and modular design

A modular design based on microservices maximizes coupling and cohesion while minimizing inter-consumer dependencies within a single release. Microservices are sourced from the policy-centric layer and deployed independently on a cloud platform as managed containers. Providing independent project teams with content and service-line-based streams for new developments and resource-enabling testing of existing

implementations naturally scales DevOps maturity. The migration strategy priorities subsystems for band-width constrained refactoring in a phased approach and evaluates phased development versus greenfield options to determine applicability for a resource-rich backlog. The enabling gates for the decision tree are freestanding sub-systems and connecting dependencies. The service contracts for the selected microservice are consumer-driven based on evolving use cases in production. Contract testing validates the contracts through version changes and is integrated into the CI/CD pipeline for early detection of contract breaks. These pipelines automate deployment into multiple cloud environment stages that target parity for reliable production testing and expose mutually exclusive rollback options. The pipeline stages support blue/green or canary release mechanisms that safely validate changes with real users and allow rapid rollbacks for failed changes. Infrastructure as code automates deployment of the cloud platform in the desired state [11].

3.2. Cross-cutting concerns: security, observability, and resilience

Identity management should enable single sign-on across the enterprise while maintaining team autonomy, ensuring required data exposure, and simplifying compliance. Enterprise grade observability must be applied without placing additional burden on individual teams. Moreover, services need to be designed for resilience, using automated techniques such as circuit breakers and chaos testing. On one side, an identity gateway will provide access management tailored to various product domains. On the other side, a central logging platform, a monitoring and alerting solution, an APM tool, and distributed tracing services will support observability, while investment in chaos engineering will ensure the production environment's resilience. Enterprise identity management is based on Azure Active Directory. All services support OpenID Connect for authentication, in order to enable single sign on across applications deployed on Azure. Role-based access control is implemented on a per-service basis, enabled by attribute-based access control at the data layer. While data exposure is controlled by product owners from each domain, the Public-facing Gateway service remains in charge of exposing product-specific APIs outside the organization [12]. A central logging platform based on the ELK stack provides real-time analysis of unstructured data across the enterprise. Grafana and Prometheus provide monitoring and alerting for managed services. Distributed applications are monitored for performance with New Relic, while Distributed Tracing is implemented with Azure Monitor Application Insights. For resilience, chaos testing is performed using Azure Chaos Studio.

3.3. Data strategy and integration patterns

Data ownership aligns closely with service boundaries. Each microservice owns its data set, enforces schema changes, and uses internal-only change data capture to share events, minimizing coupling with others. Common data sets for analytical purposes may reside in a separate database but remain immutable snapshots managed by the owning service. The enterprise architecture defines an event-driven streaming platform for publishing external and internal events, with production via an event sourcing pattern and consumption supported by an eventual consistency approach. Data storage for operational reporting adheres to the managed data platform pattern described within the target cloud architecture. To support business intelligence and operational reporting, self-service analytics and queries use a separate managed data platform. Here, data sets owned by microservices are periodically copied to a data lake with strict access control; the underlying storage is immutable and protected from unauthorized modification. The analytics or reporting layer supports additional transformations and aggregates. The architecture is complemented with document oriented storage for unstructured or semi-structured data [13].

Figure 4. P95 Latency vs Instances

	Instances	TPS (microservices)	Latency P95 (ms)
0	1	220	140.0
3	4	880	68.3
6	7	1200	56.0
9	10	1200	56.0

4. Migration Strategy and Roadmap

A phased strategy aligns technical delivery to business priorities. The roadmap commences with the assessment and ranking of policy services that reflects their pivotal role within the portfolio and interdependencies with other capabilities. Subsequently, the plan proposes a phased migration strategy guided by reusability of business logic and resources. The key areas of increased configuration management and governance opened by these changes lay the foundation for migration planning and development of the remaining policy subsystems in the .NET Core platform. Priorities are dictated by the market and regulatory environment, with the timing of delivery a consequence of technical risk management. Therefore, conventional risk–value matrices used in outsourcing assignments must be avoided. Additionally, a phased migration with dedicated reusable functionality deployed in the latter stages renders Greenfield development incipient. It is essential to confirm that the decision gates specified for business value and risk hold true. Business Value and Risk An effective transformation must balance business value against risk. For migration, business value is predominantly a function of unlocking business logic and core products for integration with services and interfaces for non-built applications, while enabling adoption of a CI/CD model. Enabling cycle times is critical to facilitate the business direction induced by changes in and around the market, particularly enhanced approval and review controls. Key areas of increased frequency of change are migrated first: the Policy Document Generation subsystem opens rules used in other applications, UI-only Policy Specifications are enabled, coverage under Policy Document Generation control stops policy documents being sent unsigned, and the application of updates to Statement of Average Uncollected Premiums becomes routine with market-placed incremental premium ratio simulation services [14].

Equation 3: System Reliability/Availability Enhancement (SRE)

Goal: show availability improvement with redundant microservices.

Availability of a component:

$$A = MTBF + MTTRMTBF \tag{4}$$

Parallel replicas (active-active, N replicas):

$$Areplica = 1 - (1 - Asingle)N \tag{5}$$

Series of N services on the critical path:

$$Aseries = Y \cdot Areplica, k, k{=}1 \tag{6}$$

Availability enhancement vs monolith:

$$SRE = Aseries - Amono \tag{7}$$

4.1. Assessment and Prioritization of Policy Subsystems

Formal assessment and prioritization of enterprise policy subsystems, with respect to business value and risk, represent critical steps in the migration strategy. Subsystem criticality to business process ensures that the primary insurance flow remains intact. Dependencies articulate the ripple effects of change; together with criticality, they provide an ordered list of subsystems. The next phase is a deeper analysis of business value, risk and level of effort, enabling prioritization of migration versus development from scratch. The outcome of this step is a phased migration roadmap. Complicated subsystems and complex business processes introduce potential bottlenecks for the migration effort. Both amplify the risk of failures during or after deployment and demand the highest levels of governance and support. For these reasons, the Commercial Underwriting subsystem is the prime candidate for the migration. With additional complexity factors associated with other underwriting product lines at the lowest end of the scale, migration of these additional workflows is fast-tracked for subsequent releases. Other complex business processes are also aligned with the migration effort for similar reasons [15].

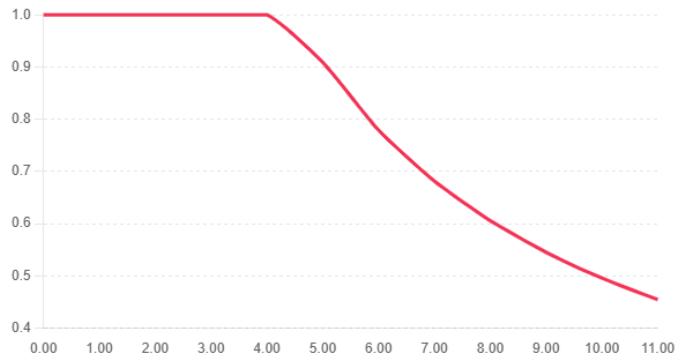

Figure 5. Parallel Efficiency vs Instances

4.2. Phased Migration vs. Greenfield development

Developing new services in parallel to migrate an existing monolith demands careful consideration. A phased approach implies multiple adjacent implementations that may not harmonize perfectly. Challenges arise when migrating certain capabilities in isolation from others. Any functional gap during the transition must therefore be temporary, and consumers must tolerate imperfect or inconsistent behavior until the entire joined implementation is available. Defining appropriate drop-dead timelines and allowing sufficient time for subsequent development are paramount to avoid undesirable compromises. Greenfield development also imposes choices in the implementation of

data and identity management capabilities in the interim. Adopting an event-driven architecture for policy data introduces the risk of inevitable data duplication across policy services during the transition. Consequently, consumers of the policy domain need not be forced to use separate identity management services until all policy subsystems are migrated, provided the necessary cross-service permissions checks or identity enrichments are merely temporary. Addressing local requirements while adhering to a holistic cross-system strategy in areas such as identity management, security, or observability is crucial for a coherent final result [16].

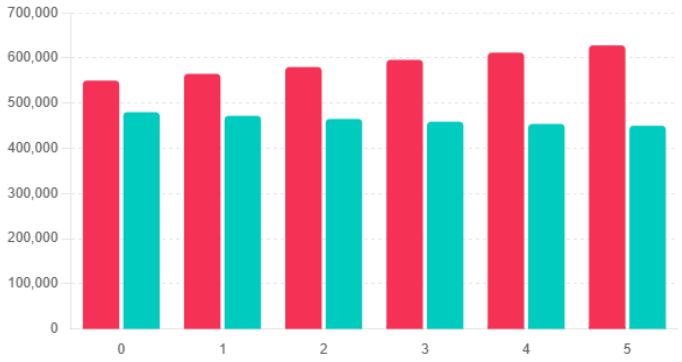

Figure 6. Annual Opex Monolith vs Microservices

4.3. Change Management and Stakeholder Engagement

Effective change management and stakeholder engagement processes increase the likelihood of a successful migration by ensuring that the right information is being shared with the right people at the right time. As multiple disparate systems are being rebuilt in different technologies, the focus of engagement remains with affected teams with a view to communicating positively about the transformation across the business. The migration is a multifaceted project, and the implementation teams are not responsible for all aspects of the successful outcomes. Consequently, the focus is on obtaining the necessary input and support from other areas of the organization. A Single Source of Truth and a clear transformation message help align teams and communicate effectively with a diverse audience. All parties should have a shared visual representation of the migration project, be aware of current priorities, and see how a technical change can generate a positive outcome and, in some areas, create business opportunities. Training opportunities can be published early, allowing affected teams to refine their skills ahead of the changes. Introducing a new technology and raising the standard of existing offerings attract the attention of business and technical users and create opportunities for additional improvements. It is critical to maintain this attention and continually identify quick wins to enhance the user experience and begin building a more positive relationship with technology [17].

Equation 4: Integration Latency Reduction (ILR)

Goal: quantify end-to-end latency improvement. Using P95 latency as the user-perceived wait W:

$$ILR = \frac{W_{mono}}{W_{mono} - W_{ms}} \tag{8}$$

(For scaling we model diminishing returns: $W_{ms}(i) = \frac{W_{base}}{1+\alpha(i-1)}$ with a floor cap.)

5. Implementation and Engineering Practices

Practical engineering norms for development teams articulate requirements that map to and support cross-cutting goals outlined in Sections 4.x and 5.x. Prototypical microservices developed during the assessment and prioritization phase set a baseline for future implementations, with a view to establishing a reference architecture to address security, observability, and resilience in .NET Core. API-first design and contract testing ensure that services are developed against a contract that specifies input and output shapes. Contracts are further developed on an ongoing basis with the consumers of the services, and consumer-driven contract testing enables early detection of breaking changes. Data migration, migration cutover, and validation planning for each service define data mapping and synchronization strategies to mitigate risks related to schema evolution. The cutover plan specifies a detailed rollback plan and acceptance criteria that are used in each migration. DevOps and CI/CD pipelines are designed to deliver microservice components in isolation, enabling continuous deployment to test and staging environments. Environment parity and blue/green or canary deployment patterns mitigate production risks. Configuration and infrastructure are treated as code and automated as far as possible to reduce the gap between configuration and production. Delineating security, identity, and compliance from the rest of the landscape permits more focused implementation effort in these areas while ensuring that the overall design still meets the cross-cutting requirements specified in Section 4.x. Specific contract points define and express detailed requirements for authentication, authorization, key management, and compliance in .NET Core deployments [18].

5.1. API-first Design and Contract Testing

Generous investment in a language-agnostic API-first design approach leads to flexible, low-coupling, parallel development, and early-mock-based testing for third-party interfaces. First, all cross-cutting concerns are formalized, and APIs are defined and documented upfront to form stable contracts for all participating teams. All APIs are under the governance of a central team that ensures all practical non-functional aspects of contract testing are honored. For external-facing APIs used by enterprise business partners, contract tests are defined and implemented at the service registry. The service provider in the role of monitoring these contracts for valid server responses during deployment updates. All consumer teams register with these APIs and create the source contract tests that validate responses compatible with these upstream contracts. Mock implementations of the APIs are actively maintained and updated at code change intervals so that testing efforts for downstream consumers can start early in the development process without transaction is sent, while that against internal APIs require real signatures and actual transactions [19].

5.2. Data Migration, Migration Cutovers, and Validation

Data migration for the implemented microservices is necessary only if the data scheme changes during migration. In that case, migrating data usually poses the biggest challenge.

Planning will include data mapping from the legacy system to the target service and a strategy for partitioning the data changes between services in a way that minimizes impact on the business. For example, an order service may have to read data from both the existing order system and the migrated service for some period of time until the order system can be finally deprecated. Depending on the data involved and how long before it

will not be referenced in either system, regarding schema evolution there are three strategies: offer both old and new scheme representation in the new service, synchronize the data between current and desired data representation, and adopt the new representation in safe-to-unload time period. When the old and new services change data at the same time, additional mapping between old and new representation is needed. This obviously requires additional storage and time. Furthermore, small changes on the new representation made by the team of the current service will impact the team migrating to that service. To minimize cutover problems, deploying in small practical steps is advisable, including decommissioning clouds, when possible, before their next release. There should also be a rollback plan, and all changes must meet assumed acceptance criteria specified earlier to make the migration deterministic and reliable [20].

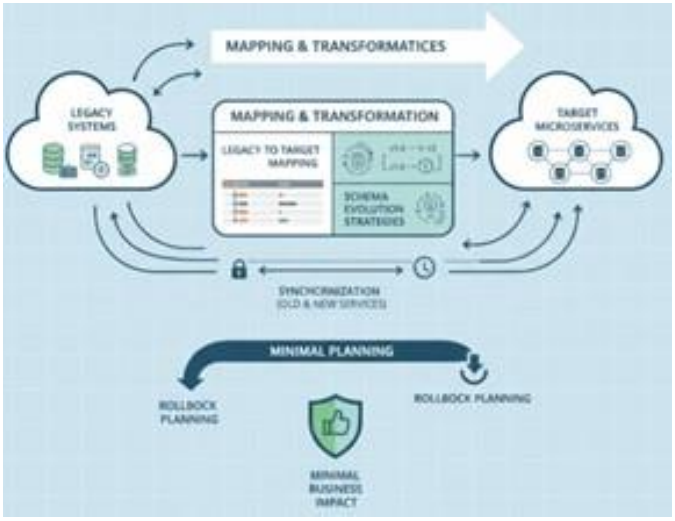

Figure 7. Data Migration Strategies and Rollback Planning for Microservices with Schema Evolution

5.3. DevOps, CI/CD, and Platform Automation

Deployment pipelines automate and standardize the release process, reducing the risk of human error and minimizing surprises with each cutover. Dedicated environments facilitate testing in conditions that match production as closely as possible, yet full parity is often prohibitively expensive. Therefore, pipelines aim to maximize similarities between environments while accommodating differences with automation. An on-premises implementation of Octopus Deploy orchestrates deployment cutovers across multiple environments, while NSX provides network isolation for LastPass and HSM enforced secrets. To support CI/CD pipelines that validate infrastructure-as-code templates across realistic configurations, a private cloud platform abstracts the underlying vSphere elements. Use of publicly-hosted components, such as NuGet repositories or S3 storage, expands ecosystem support. Critical paths provision and configure staging environments in the target cloud context on a nightly basis, validating Docker builds of policy tooling suites for all supported operational platforms and Windows binaries for non-Linux-native components. These build artefacts propagate through blue/green deployment pipelines that utilise a production-like acceptance environment for final validation against quality gates. Endpoint-corner-cutters facilitate controlled partial deployment or dedicated purpose-hosting within production [21]. Release management forums provide oversight for policy tooling deployment to production, enabling transparent communication of the impacts introduced at cutover. Dedicated business areas hosting non-public policy services engage during code development and provide acceptance approval for the proposed release. The remaining policy services, which

support wide-scale environment-independent interactions, undergo automated contract validation against their external consumers.

5.4. Security, Identity, and Compliance in .NET Core

Security, identity management, key handling, data protection, auditability, and compliance with supervisory, tax, and data sovereignty requirements must be informed by regulations and policies identified in Section 3.2. Teams must assess the availability of compliant solutions in the target technology stack and incorporate any necessary requirements into engineering standards. A few notable aspects of these controls for .NET Core development are outlined below. Authentication and authorization should leverage the identity management platform defined for cross-cutting concerns in Section 4.2. If applicable, previously deployed components may already provide these capabilities without modification. Externalizing authN/Z must be assessed on a per-service basis, as Bundle services may have the complexity to justify a dedicated client application—particularly if a separate editing environment is also required. Such a client may also simplify subsequent roll-out of multi-factor authentication to satisfy compliance requirements. Sensitive data must be protected with industry-standard library functions, while encryption keys and secrets used by the application must be stored separately from the application code. Cloud vendor offerings that support these requirements are available and should be evaluated. Official guidance on PCI compliance for Azure DevOps release pipelines is also available; automated validation through CI/CD pipelines reduces risk of undetected uncompliance while creating a repeating control point [22].

Equation 5: Migration Cost Efficiency (MCE)

Goal: evaluate economics of the migration.

NPV of Opex savings over years $\sum_{t=0}^{T}$ at discount rate r:

$$NPVsavings = \sum_{t=0}^{T} \frac{\text{Opex}_{mono}(t) - \text{Opex}_{ms}(t)}{(1+r)^{t}} \tag{9}$$

ROI vs CAPEX:

$$ROI = CAPEXmigrationNPVsavings - CAPEXmigration \tag{10}$$

Payback (discounted): smallest –t where cumulative discounted savings ≥ CAPEX.

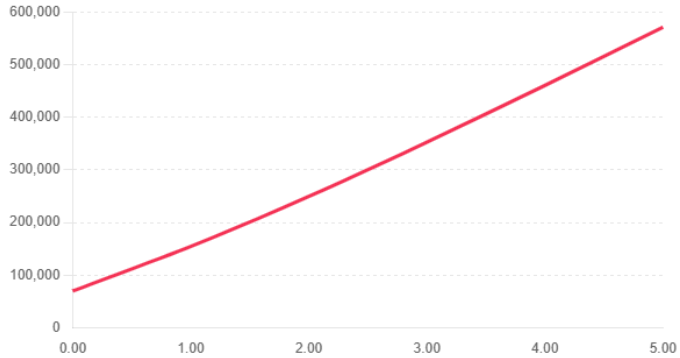

Figure 8. Cumulative Discounted Savings vs CAPEX

6. Quality, Risk, and Governance

Concrete implementation and operational engineering practices ensure delivery conforms to the target architecture and successfully addresses strategies and cross-cutting concerns identified in 4.x and 5.x. A fit-for-purpose approach to testing balances risk exposure with delivery cadence, while risk registers address system and service-level threat mitigation outside regular testing. Auditability and regulatory compliance embed validated inspectability in the posture of migrated subsystems. Quality assurance employs a risk-based strategy addressing both product delivery and operational resilience. Critical sub-systems such as the external REST API expose consumer driving contracts exercised through simulation in a dedicated environment. Contract validation occurs at all pipeline stages using consumer-side models within a producer's pre-release CI/CD. Non-contractual functions, locales, and internal paths incorporate core automated test coverage during development, with additional performance testing for candidate system components generating high load. Dependencies and system complexity determine remaining testing scope and cadence. Regulatory audit requirements underpin a product risk register that, among other items, identifies code-compliance checks for specialized data-handling services and exposure of a validation segment of a consumer account-management back-office UI with test-user functionality. Third-party penetration testing of publicly-facing services remains standard practice in the release cycle [23].

6.1. Testing Strategy for Policy Calculations and Rules

A robust quality assurance strategy enables rapid delivery with high confidence that changes do not break existing functionality. Two complementary approaches serve to minimize the risk of regressions in policy calculations and rules: an API-first design philosophy that uncouples policy-handling services, supported by consumer-driven contract testing; and a business-logic-first approach that creates and maintains a comprehensive, easy-to-use test suite for calculation and rule engines. Policy-handling services communicate with other subsystems through well-defined, versioned API contracts. Each API definition explicitly specifies input and output schemas, as well as a description of expected behaviour. For any API, consumers can use these contracts as stubs to validate their own implementations. Supported by a consumer-driven contract testing framework such as Pact, this approach ensures that each API consumer and provider is continually validated against the API contract—regardless of whether the consumer or provider is modified. This decoupling enables teams to focus on changes within their own areas, minimizing dependencies for change management. Business logic is tested through a suite of unit tests that stimulate policy calculations and rules in much the same way as a typical user would. Test cases for calculations validate a representative set of products and associated data, while a broader range of test scenarios is defined to exercise rule engines. Automated tests are executed before deployment. For every regular release, the test suite is extended to capture significant rules or calculation scenarios that may not already be included, as well as to provide regression checks for any recently identified bugs. Business users are encouraged to consider test automation as part of any manual testing—they participate in more than half of the tests exercise testing at pre-release and release-service stages—and the test suite is adjusted as appropriate [24].

Equation 6: Policy Processing Scalability (PPS)

Goal: show scaling efficiency.
Speedup:

$$S(i) = TPS(1)\ TPS(i) \tag{11}$$

Parallel efficiency:

$$E(i) = iS(i) \tag{12}$$

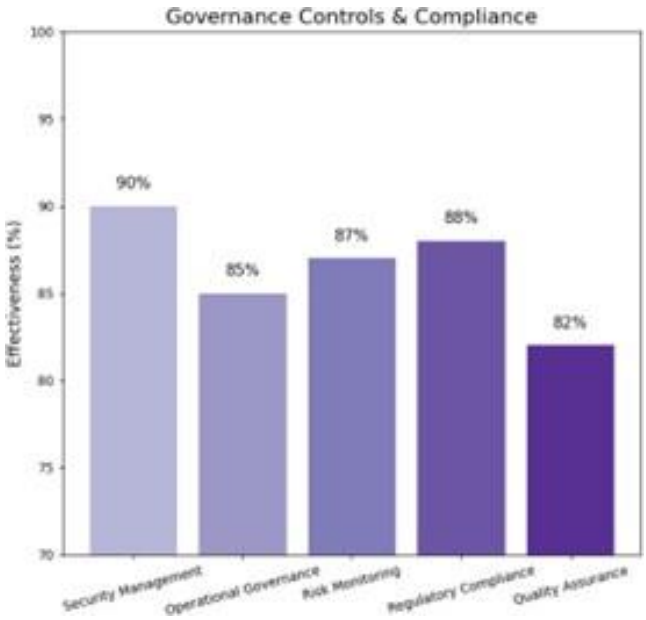

Figure 9. Governance Controls & Compliance

6.2. Auditability and Traceability

Integrity control and auditability of important financial transactions are a critical concern in a regulated industry like insurance. It is necessary to ensure that any transaction change is discoverable and traceable for audit purposes. In the enterprise subsystems under consideration, audit trails cannot be implemented using the built-in data layering support of the technology. Audit trails must therefore be built manually. A common approach is to prepare an additional shadow table with a modification trigger on each table that captures a full record of any row that is modified or deleted, but this requires the shadow table to be kept in sync with the existing physical data in case of schema changes. The shadow table approach will not work for insert operations as they generate new rows rather than modifying existing ones. Whatever approach is taken, it does need to be incorporated in the implementation and testing efforts, whether this is done by the individual teams for each component or in a consolidated manner with services from other parts of the organization. Another concern is that of auditability of financial transactions. For reporting purposes any transaction involving monetary flows may need to be preserved in an additional audit table capable of replaying the transaction for forensic analysis later. Depending on the factors, such as whether transactions require forensic support, data governance decisions will determine the audit trail and audit table structures [25].

6.3. Risk Management and Incident Response

Clear processes for identifying, tracking, and mitigating risks must be established before releasing any subsystem. The process should as a starting point be based on the risk identification and control log from Bank A and be tailored to the context of a growing organization with more dynamic and less stable requirements. Digital transformation is fundamentally based on the re-engineering of business processes by taking advantage of technology. Consequently, the nature of risks considered and the

mitigation plans put in place should be closely aligned with the work of the business process owners and how changes affect their processes. As new subsystems go into production, additional operational risks are introduced and the risk management process needs to evolve into maintaining a full risk register [26]. A dedicated incident response team may be quickly required as the .NET Core assets go into production. The requirement for the team will depend on the level of expertise found amongst existing staff. A very experienced Cloud engineer has the capability to resolve incidents in the deployment and operation of the assets. SonarQube alerting should rapidly determine whether code is causing security vulnerabilities. Azure DevOps alerts inform the development teams when unit tests are failing. Azure Monitor alerts detect when application performance is degrading, when applications or containers stop and when the underlying infrastructure resources are exhausting capacity. Therefore, at least some leadership or coordination activity around incidents may be needed. Eventually a dedicated incident response team – either as a separate funtion or more as an embedded service – is typically needed to address Cloud automation, deployment, infrastructure and other operational challenges if the organization is scaling its Cloud footprint and services. An incident response team should ideally also make it a priority to deliver stability of the services and to mitigate operational risk. In such a scenario, Azure Site Reliability Engineering industry best practices should be leveraged to strengthen the organization's Cloud services [27].

7. Conclusion

Digital transformation has long been a priority for financial services organizations seeking to service customers more effectively and efficiently. Increasing competition from digital-native challengers demands modern, fast, cost-effective services to remain relevant. The enterprise installed base of a large international insurance company spans a number of traditional programming platforms, all operating as monolithic services. Supporting Digital Transformation objectives in the Enterprise Policy Systems area of the business has necessitated a program to migrate existing policy subsystems to .NET Core [28]. Substantial risk and cost avoidance opportunities have been identified as a result of migrating to .NET Core. The migration takes place in stages over time, aligning directly with the business strategy for Digital Transformation. Individual subsystems are subject to assessment to identify risk, complexity, and cost of migration and then prioritized accordingly. Changes in personnel and risk appetite may influence decisions regarding the migration path chosen for each subsystem. By engaging the user community throughout the program, security and operational concerns can be addressed in advance of deployment, ensuring proper governance controls are maintained. Stakeholder engagement supports confidence in the process, enabling ongoing delivery of high-quality insurance products with minimal impact to the business. A major international insurance company is undertaking a significant Digital Transformation program to modernize its core policy management systems. The Enterprise Policy Systems area operates as a separate business entity, leveraging policy management subsystems from several different companies to provide a complete offering to external customers. Substantial risk and cost avoidance opportunities have been identified as a result of migrating particular subsystems from .NET Framework to .NET Core and therefore an assessment, prioritization, and phased migration plan have been established [29].

7.1. Key Takeaways and Future Directions

Digital transformation of insurance technology landscapes is logical and expected considering market need, competitive pressure, and regulatory expectations. The enterprise policy systems at the focus of this investment choice have migrated to .NET Core, and the broader enterprise policy ecosystem has emerged as a trusted cloud-based

enterprise service that meets customer business needs more effectively and efficiently. For the Policy and Direct-to-Consumer areas under the control of the New Zealand division, the transition is highly positive. Key learnings from the broader activity are formalized, with suggested next steps for the enterprise technology evolution expressed. The ongoing demise of the on-premises enterprise operating model plays out via widespread public cloud adoption, driven by the increasing overheads of facilities management for complex infrastructures. People and skills are more readily available to exploit the benefits of these platforms, including virtualized operating and hosting environments, the rapid provisioning of solutions, blue/green/canary DevOps practices, and Infrastructure as Code. Hosting at compliance focused service providers reduces the requirement for in-house audit and certification efforts and enables a more competitive structure with lower overheads, overheads best avoided by taking advantage of the hosting organization. Microsoft Azure, Amazon AWS, and Google GCP supply this required hosting and DevOps facilitation [30]. Microsoft's .NET Core development framework also addresses many of the requirements of the enterprise technology landscape, enabling efficient hosting via Docker containers, being API-first oriented, with a rich ecosystem of readily available libraries, modules, and tooling. The requirement for rapid-turn, low-latency services is well supported by the enterprise technology .NET Core hosting and development framework. Microsoft's ownership of the product ensures service equivalency between the on-premise and the cloud hosted versions of the applications. .NET 8 is expected to continue this application and operational alignment, de-risking the requirement for migration planning. On going support for Active Directory authentication and Azure database services simplifies the transition process, provides deep hosting expertise, and ensures continued identity and authentication service equivalency across both code bases [31].

References

[1] Ahmad, T., & Kim, D. (2021). Hybrid models for financial time-series forecasting using signal decomposition and deep learning. Expert Systems with Applications, 168, 114403.

[2] Gadi, A. L., Kannan, S., Nandan, B. P., Komaragiri, V. B., & Singireddy, S. (2021). Advanced Computational Technologies in Vehicle Production, Digital Connectivity, and Sustainable Transportation: Innovations in Intelligent Systems, Eco-Friendly Manufacturing, and Financial Optimization.

[3] Dutta, S., & Roy, S. (2021). A deep learning approach for latency minimization in high-frequency trading DSP systems. Neural Computing and Applications, 33, 15765–15782.

[4] Zhang, Q., & Liu, Y. (2021). Machine learning-enhanced DSP for real-time anomaly detection in financial messaging systems. IEEE Transactions on Neural Networks and Learning Systems, 32(12), 5521–5535.

[5] Lahari Pandiri. (2021). Machine Learning Approaches in Pricing and Claims Optimization for Recreational Vehicle Insurance. Journal of International Crisis and Risk Communication Research, 194–214. https://doi.org/10.63278/jicrcr.vi.3037.

[6] Data-Driven Strategies for Optimizing Customer Journeys Across Telecom and Healthcare Industries. (2021). International Journal of Engineering and Computer Science, 10(12), 25552-25571. https://doi.org/10.18535/ijecs.v10i12.4662.

[7] Al-Hawari, T., & Al-Zoubi, M. (2021). Latency-aware architectures for streaming DSP systems. IEEE Access, 9, 79245–79259.

[8] Wang, S., & Yuan, L. (2021). Gabor and wavelet feature fusion for high-volume payment data streams. Digital Signal Processing, 112, 103015.

[9] AI-Based Financial Advisory Systems: Revolutionizing Personalized Investment Strategies. (2021). International Journal of Engineering and Computer Science, 10(12). https://doi.org/10.18535/ijecs.v10i12.4655.

[10] Anwar, F., & Siddiqi, M. (2021). Real-time fraud detection using spectral feature extraction in payment systems. ACM Transactions on Information Systems, 39(3), 1–23.

[11] Just-in-Time Inventory Management Using Reinforcement Learning in Automotive Supply Chains. (2021). International Journal of Engineering and Computer Science, 10(12), 25586-25605. https://doi.org/10.18535/ijecs.v10i12.4666.

[12] Gupta, R., & Singh, A. (2021). Spectral-domain anomaly scoring in transaction streams. Pattern Recognition Letters, 147, 180–189.

[13] Aslam, M. S., & Qureshi, I. A. (2021). Wavelet-based analysis of transactional data for fraud risk estimation. Physica A: Statistical Mechanics and Its Applications, 580, 126176.

[14] Goutham Kumar Sheelam, Botlagunta Preethish Nandan, "Machine Learning Integration in Semiconductor Research and Manufacturing Pipelines," International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), DOI: 10.17148/IJAR-CCE.2021.101274.
[15] Han, Y., & Zhao, W. (2021). Latency reduction in multi-stage digital filters for streaming data systems. IEEE Transactions on Circuits and Systems I, 68(6), 2417–2429.
[16] Balasubramaniam, P., & Lee, C. K. M. (2021). Dynamic data-driven approaches for payment flow optimization. Computers & Industrial Engineering, 158, 107418.
[17] Raviteja Meda, "Digital Infrastructure for Predictive Inventory Management in Retail Using Machine Learning," International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), DOI: 10.17148/IJARCCE.2021.101276
[18] He, K., & Sun, J. (2021). Performance trade-offs in DSP-based financial monitoring systems. Journal of Systems Architecture, 119, 102301.
[19] Bhattacharya, A., & Banerjee, S. (2021). Adaptive filtering and prediction in volatile financial environments. Signal Processing, 186, 108140.
[20] Inala, R. (2021). A New Paradigm in Retirement Solution Platforms: Leveraging Data Governance to Build AI-Ready Data Products. Journal of International Crisis and Risk Communication Research, 286–310. https://doi.org/10.63278/jicrcr.vi.3101
[21] Jiang, H., & Wang, M. (2021). High-volume payment system analytics using signal representation models. Information Processing & Management, 58(6), 102682.
[22] Chen, H., & Zhang, Y. (2021). Optimizing digital signal pipelines for financial communication systems. IEEE Transactions on Industrial Informatics, 17(10), 6900–6912.
[23] Kang, J., & Kim, H. (2021). Temporal noise modeling in financial data streams for fraud pattern detection. IEEE Transactions on Computational Social Systems, 8(4), 939–951.
[24] Aitha, A. R. (2021). Dev Ops Driven Digital Transformation: Accelerating Innovation In The Insurance Industry. Journal of International Crisis and Risk Communication Research, 327–338. https://doi.org/10.63278/jicrcr.vi.3341
[25] Das, P., & Sharma, R. (2021). DSP-enabled fraud analytics for cross-border payment systems. International Journal of Information Management Data Insights, 1(2), 100041.
[26] Li, F., & Zhou, Y. (2021). A study of latency optimization in distributed DSP frameworks for fintech systems. Future Generation Computer Systems, 124, 381–391.
[27] Liu, X., & Xu, T. (2021). Feature-extraction pipelines for streaming DSP applications. Journal of Real-Time Image Processing, 18(5), 1679–1691.
[28] Lu, C., & Wei, X. (2021). Signal transformation and fraud signal isolation in high-frequency finance. IEEE Transactions on Engineering Management, 68(5), 1302–1314.
[29] Majeed, S., & Farooq, M. (2021). Digital signal processing for streaming payment message validation. Journal of Network and Computer Applications, 176, 102927.
[30] Meng, Q., & Chen, Z. (2021). DSP-based modeling of network anomalies in financial infrastructures. Computer Communications, 180, 130–141.
[31] Patel, R., & Mehta, N. (2021). Real-time streaming frameworks for SWIFT-based fraud detection systems. Journal of Financial Technology, 5(2), 45–63.