Sarcouncil Journal of Engineering and Computer Sciences

ISSN(Online): 2945-3585

Volume- 04| Issue- 11| 2025

Research Article

Received: 10-10-2025| Accepted: 05-11-2025 | Published: 19-11-2025

Incremental Modernization: APIs, DevOps, and Cloud-Readiness in IBM i Environments

Srinivas Allam

Core ITS LLC, USA

Abstract: The digital transformation of enterprise computing infrastructure requires IBM i platforms to evolve from isolated legacy systems to integrated parts of modern digital ecosystems. This transformation is intended to meet the critical challenge of preserving decades of proven business logic and institutional knowledge while making possible contemporary capabilities like API-driven integration, DevOps automation, and cloud deployment flexibility. Anchored by the Strangler Application pattern, the modular evolutionary approach allows for incremental porting of monolithic applications to microservices architectures without impairing operational continuity. API enablement is a basic enabler for exposing IBM i functionality to mobile applications, cloud services, and partner systems as RESTful interfaces and event-driven architectures. A combination of distributed version control systems, continuous integration pipelines, and modern development environments changes the repetitive IBM i development processes into agile collaborative development processes in accordance with industry best practices. The cloud migration strategies that include lift-and-shift migrations, hybrid frameworks, as well as fully offered services offer companies the versatile mechanisms to harness elastic capacity and geographic dispersion, besides catering to security and compliance requirements. Such a combination of technological and methodological innovations establishes a sustainable architecture of evolutionary change that successfully strikes a balance between innovation needs and the needs of operational stability to keep the IBM i platforms strategic to enterprise architectures and support digital business agendas.

Keywords: IBM i Modernization, Api Enablement, Devops Integration, Cloud Migration, Microservices Architecture.

INTRODUCTION

The digital transformation requirement has put legacy systems at a crossroads where they need to evolve without losing the vital business value that decades of operational perfection have accumulated. The AS/400 or iSeries environments, now called IBM i, constitute a major portion of enterprise computing infrastructure, particularly in financial services, manufacturing, and retail. According to the recent marketplace survey data, IBM i platform is still demonstrating incredible resilience and strategic significance because 58 percent of organizations report that their dependency on their IBM i systems has increased in the last three years, and 21 percent of organizations reported no change in their dependence with the platform and only 8 percent of organizations indicate the reduced dependency on the platform (Huntington, T. 2025). Moreover, the survey shows that 84% of organizations consider their IBM i systems as critical or very important to business operations, pointing to their sustaining role in enterprise computing architectures (Huntington, T. 2025). These systems continue to run mission-critical workloads with exceptional reliability, yet are under increasing pressure to integrate into modern architectures, support contemporary development practices, and tap cloud computing capabilities that have become fundamental for competitive positioning across digital markets.

Traditional approaches to system modernization have often advocated for wholesale replacement or "rip-and-replace" strategies, approaches that carry substantial risk, require extensive capital investment, and all too often result in project failures or significant cost overruns. Empirical analysis of legacy system migration projects reveals that big-bang replacement methodologies find profound challenges: the complexity of reimplementing decades of accumulated business logic, regulatory compliance rules, and operational workflows is found to be far more difficult than ever imagined (Availability Digest, 2007). Organizations attempting wholesale system replacements frequently find themselves facing a situation where comprehensive requirements documentation does not exist, institutional knowledge resides with personnel approaching retirement age, and subtle intricacies of business rule implementations cannot be reduced to reverse engineering attempts (Availability Digest, 2007). The incremental approach to the migration of legacy systems presents a very different paradigm, recognizing that evolutionary transformation through gateway interfaces and progressive modernization reduces risk while maintaining operational continuity (Availability Digest, 2007). This approach allows organizations to preserve the substantial investment in proven business logic and incrementally introduce modern capabilities,

*Corresponding Author: Srinivas Allam

DOI- https://doi.org/10.5281/zenodo.17652964

avoiding catastrophic failures that have characterized many high-profile big-bang migration attempts in the literature across industries (Availability Digest, 2007).

The paper represents an academic investigation into the principles, methods, and technology enablers that form the basis of incremental modernization strategies for IBM i platforms. It looks at API enablement as a core building block, integration with modern DevOps toolchains, migration paths to the cloud and hybrid cloud architectures, and the emerging role of artificial intelligence in enhancing system reliability along the steps of the modernization process. The aim is to provide a theoretical and practical framework for evolutionary transformation that would preserve institutional knowledge while enabling digital capabilities required for competitive positioning in modern markets, on the basis of empirical evidence that demonstrates the technical feasibility and economic viability of graduated transformation approaches.

Table 1: IBM i Platform Characteristics and Migration Challenges (Huntington, T. 2025; Availability Digest, 2007)

Aspect	Traditional Environment	Modernization Challenge
System Criticality	Mission-critical operations	Integration with contemporary architecture
Business Logic	Decades of refined workflows	Preservation during transformation
Documentation	Limited or absent	Institutional knowledge capture
Personnel	An aging expert workforce	Knowledge transfer requirements

THE MODULAR EVOLUTIONARY APPROACH TO LEGACY SYSTEM TRANSFORMATION

The modular evolutionary approach represents a paradigm shift from traditional modernization strategies focused on disruptive replacement. This methodology recognizes that a lot of significant business logic, mechanisms for ensuring regulatory compliance, and operational workflows have been embedded in those legacy systems, which have become refined with several years of production use. The strangler application pattern provides a structured approach to this transformation; it describes how organizations can incrementally migrate monolithic applications to a microservices architecture through incremental build-ups of new functionality around the core of legacy systems and concurrently decommissioning old components (Brown, K. 2020). It is an effective pattern for IBM i environments where total replacement of systems is fraught with unacceptable business risk, given that it allows the coexistence of new microservices with existing monolithic applications through well-defined API interfaces intercepting calls and progressively routing them to modernized components (Brown, K. 2020). Instead of discarding accumulated intellectual capital, the evolutionary approach aims at identifying, refactoring, and recontextualizing existing functionalities into modern architectural patterns. This helps an enterprise ensure continuity within operations while systematically addressing the technical debt developed during a couple of decades of system evolution (Brown, K. 2020).

Central to this is the principle of incremental refactoring, wherein large monolithic applications are systematically analyzed for discrete functional components that need to be restructured as independent, loosely coupled services. The strangler pattern works by using an intercepting façade layer that mediates requests to the legacy system, intelligently routing traffic to either the newly developed microservices or existing monolithic components based on migration progress 3. This has proven to be an effective architectural strategy because it allows development teams to target high-value business capabilities for early migration, realizing early value from the transformation effort while building critical organizational competency in modern development practices 3. This pattern works effectively because it reduces migration risk through the incremental testing of new services in production environments ahead of retiring selected legacy functionality 3.

Architectural considerations within this evolutionary approach go beyond mere technical restructuring. The domain-driven design principles guide the decomposition of monolithic applications to bounded contexts, making sure that the resulting components will be aligned with business domains rather than arbitrary technical boundaries. Contemporary research in the area of microservices architecture has made it clear that successful implementations have to pay close attention to service granularity, inter-service communication patterns, data consistency management, and deployment orchestration

Publisher: SARC Publisher

(Dragoni, N. *et al.*, 2017). In this respect, quite a few organizations adopting microservices architectures have reported significant gains in development velocity, system resilience, and operational flexibility. These benefits only materialize, however, when architectural principles receive rigorous application and the organizational structures adapt in order to support autonomous service teams (Dragoni, N. *et al.*, 2017). Microservices architecture has validated the evolutionary approach as a dominant paradigm for distributed systems. This illustrates that loosely coupled, independently deployable services offer much better flexibility, scalability, and resilience compared to their monolithic counterparts, as long as the migration strategy respects the inherent complexity of distributed system design (Dragoni, N. *et al.*, 2017). In this way, the transition from monolithic to microservices architectures can be achieved via strangler patterns, enabling organizations to maintain business continuity while gradually modernizing their respective technical infrastructures and, thus, balancing innovation imperatives with requirements for operational stability (Dragoni, N. *et al.*, 2017).

Table 2: Evolutionary Modernization Patterns (Brown, K. 2020; Dragoni, N. *et al.*, 2017)

Pattern Component	Implementation Approach	Primary Benefit
Strangler Façade	Request interception layer	Progressive migration capability
Incremental Refactoring	Component-by-component transformation	Risk reduction through validation
Domain-Driven Design	Business-aligned decomposition	Maintainability enhancement
Microservices Architecture	Loosely coupled services	Deployment independence

API ENABLEMENT AS THE FOUNDATION OF MODERNIZATION

Application Programming Interfaces are the very building blocks of how IBM i systems are able to fit into modern digital ecosystems. API enablement changes these legacy applications from isolated monoliths to integrated parts that can communicate bidirectionally with contemporary systems, mobile applications, web services, and third-party platforms. Modernization through seamless API integration addresses key business challenges, such as real-time data access, cloud integration, and mobile application development, without requiring wholesale replacement of proven business logic (Khatri, N. 2024). It is more than a technological refresh but rather the repositioning of enterprise assets as composable services that can be orchestrated to meet evolving business requirements, where organizations leverage REST APIs and web services to expose IBM i functionality and attain significantly improved levels of agility in meeting market demands and customer expectations (Khatri, N. 2024).

The technical enablement of APIs on IBM i platforms usually adopts RESTful web services, which became a de facto standard for system-to-system integration because of their simplicity, statelessness, and alignment with HTTP protocols. The process of API enablement into an IBM i environment can be done in several ways, such as an Integrated Web Services Server that lets RPG and COBOL be published as web services, without change, open-source frameworks, and more specific middleware that liberates the legacy applications and puts them in front of the new consumers. Since it uses standard HTTP methods, including GET to access resources, POST to create resources, PUT to update the entire resource, PATCH to update a part, and DELETE to delete resources, the REST architecture has become predominant as it produces interfaces that are easy to use and follow, and which are intuitive based on web semantics. This exposure is done without necessarily introducing fundamental alterations to the preexisting business logic and maintaining the stability and reliability attributes that have historically characterized IBM i systems without sacrificing patterns of integration that drive a great diversity of modern digital endeavors.

Not only is REST an important consideration, but also the various architectural patterns and security frameworks are increasingly taken into consideration for enterprise-grade API implementations. In designing RESTful APIs, an important aspect is resource-oriented architecture, where URIs are used to identify resources, HTTP methods to determine operations, and hypermedia controls enable clients to navigate through application state transitions (Masse, M. 2011). Organizations that implement APIs need to consider key design aspects such as conventions for constructing URIs, which foster consistency and discoverability; adequate use of HTTP status codes that faithfully convey operation outcomes; content negotiation mechanisms that enable multiple forms of representation; and strategies for caching that optimize network effectiveness (Masse, M. 2011). Security considerations become

Publisher: SARC Publisher

paramount in API design and are concerned with the implementation of authentication schemes, authorization systems that implement fine-grained access control, transport layer encryption through HTTPS to protect data during transmission, and input validation procedures that prevent injection attacks and processing of malformed requests (Masse, M. 2011). The principle of least privilege underlines access control design, ensuring that a given consumer receives only those permissions needed to perform the legitimate operation while keeping audit trails that support compliance requirements and investigation of security incidents (Masse, M. 2011). API enablement has strategic value beyond immediate integration needs. APIs that are well-designed are reusable resources, which may be used by many consumption channels without duplicating the business logic, including mobile applications, web portals, partner systems, and analytics platforms. This reusability accelerates the creation of new capabilities, provides uniformity in the application of business rules across a broad range of touchpoints, and empowers organizations to utilize their investments in IBM i to digital transformation efforts to enhance customer experience and operations. Besides, the APIs also facilitate the gradual transition to microservices architectures by providing explicit contracts between components of the system-enabling, over time, the replacement of underlying implementations without impacting the consumers and maintaining the backward compatibility-crucially important to business continuity where long-term transformation initiatives are concerned.

Table 3: API Enablement Strategies (Khatri, N. 2024; Masse, M. 2011)

API Component	Technical Implementation	Strategic Value
REST Endpoints	HTTP method mapping	Standard web integration
Resource Orientation	URI-based identification	Intuitive interface design
Security Framework	OAuth and HTTPS	Enterprise-grade protection
Reusability	Multi-channel support	Accelerated capability development

DEVOPS INTEGRATION AND MODERN DEVELOPMENT PRACTICES

The integration of IBM i development workflows with modern DevOps practices is a significant cultural and technical transformation. Traditional IBM i development environments have conventionally worked within paradigms characterized by centralized source management on the host system, limited version control capabilities, and manual deployment processes that have created bottlenecks in software delivery cycles (Eradani, 2024). The adoption of DevOps principles, with a focus on automation, continuous integration, continuous delivery, and collaborative development practices, requires changes at the very foundation of established workflows but respects the peculiar characteristics of IBM i platforms. The organizations moving toward the adoption of DevOps methodologies must bridge the gap in skills between traditional IBM i developers proficient in RPG and COBOL and the current state of development encompassing Git version control, pipelines for CI/CD, containerization concepts, and infrastructure automation (Eradani, 2024). This transformation entails comprehensive training programs and gradual adoption approaches within which veteran developers can gain contemporary skills while tapping deep domain expertise and an understanding of critical business logic locked inside legacy applications (Eradani, 2024).

Source code management through distributed version control systems, in particular Git, is a foundational part of this transformation. In this respect, modern IBM i development tooling facilitates integration that synchronizes source members from IBM i libraries with Git repositories. Such integration allows developers to apply industry-standard version control workflows such as branching strategies, pull requests, and the resolution of merge conflicts. The shift from legacy source physical file management to Git-based workflows demands knowledge in the principles of distributed version control, along with knowledge of branching models such as GitFlow or trunk-based development, and collaborative code review practices, which are fundamentally different from the sequential development approaches common in legacy environments (Eradani, 2024). This approach keeps the production source libraries stable while allowing for experimentation and parallel development in isolated branches, easing feature flags, A/B testing capabilities, and progressive deployment strategies that reduce the risk associated with production releases (Eradani, 2024).

Publisher: SARC Publisher

The development of the Integrated Development Environment has provided alternatives to the green-screen editors as a tooling system for developing programs. IBM i-specific extensions are provided to the Visual Code software that provides syntax highlighting, code navigation, debugging, and easy integration with Git repositories. This move, as a result of the adoption of modern IDEs, represents a fundamental paradigm shift among the developers who were accustomed to command-line interfaces and source entry utilities, and may demand investment in training and managing change, to strike a balance against the decades of established practices (Eradani, 2024). This modern development experience attracts newer talent who are more familiar with contemporary tooling while increasing productivity thanks to features such as intelligent code completion, refactoring assistance, and integrated testing frameworks, further aligning the IBM i development practices with industry standards evident in other technology stacks (Eradani, 2024).

Continuous Integration and Continuous Delivery pipelines are automated systems used to construct, test, and deploy IBM i applications, which cut down on manual work and human error and also improve the speed at which they are delivered. The DevOps Handbook suggests three foundational principles of effective implementations of DevOps: The First Way is based on a fast flow between development and operations through automated deployments and minimized batch sizes, The Second Way is built on rapid feedback loops, achieved through comprehensive monitoring and testing, and The Third Way on cultures of continuous experimentation and learning (Kim, G. *et al.*, 2016). Organizations that adopt these principles experience transformative improvements in software delivery performance: high-performing technology organizations have deployment frequencies 200 times faster, lead times from commit to deploy that are 2,555 times faster, and change failure rates that are three times lower than those of low performers (Kim, G. *et al.*, 2016). These dramatic differentials in performance translate directly into competitive advantage, with organizations able to respond more rapidly to market opportunities and customer needs while sustaining superior reliability and stability (Kim, G. *et al.*, 2016).

The automation of system configurations, library management, and the orchestration of deployment are some of the Infrastructure as Code principles applied to the IBM i environment. These cultural features of DevOps adoption-collaboration among both development and operations teams, blameless post-mortems, and continuous improvement not the secondary components to such an implementation; successful transformations in DevOps require executive support, investment in tools and education, and simple rearrangements around value streams instead of functional silos 8.

Table 4: DevOps Transformation Elements (Eradani, 2024; Kim, G. *et al., 2016*)

DevOps Practice	Traditional Contrast	Transformation Requirement
Version Control	Host-based management	Git distributed systems
Development Environment	Green-screen editors	Modern IDEs with extensions
Deployment Process	Manual procedures	Automated CI/CD pipelines
Organizational Culture	Functional silos	Cross-functional teams

CLOUD MIGRATION STRATEGIES AND HYBRID ARCHITECTURE PATTERNS

IBM i workload migrations to cloud environments come with unique challenges and opportunities that demand very thoughtful consideration of application characteristics, data sovereignty requirements, performance expectations, and cost implications. There are multiple migration pathways, from approaches based on Infrastructure as a Service, which essentially relocate existing systems to cloud-hosted environments, to more transformative methods that decompose applications into cloud-native services with ongoing use of core IBM i functionality for specialized workloads. According to in-depth security and modernization research, modernization has become an increasingly important activity for IBM i organizations, with survey data showing that organizations are under increasing pressure to integrate their legacy systems into contemporary cloud infrastructures while sustaining the security postures and compliance frameworks that have typified IBM i environments up to this time (Fortra, 2024). This research indicates that security considerations hold center stage during journeys of modernization; for example, 78% of respondents put security as the top priority as they consider cloud migration

Publisher: SARC Publisher

strategies, while 64% cite concerns about maintaining regulatory compliance during transitions to either hybrid or cloud-native architecture (Fortra, 2024).

Lift-and-shift migrations, in which entire IBM i logical partitions migrate to cloud infrastructure providers offering IBM Power Systems capacity, are the least invasive type of migration. All the large cloud providers currently have IBM i hosting solutions that enable the organization to continue with operations and still have the benefits in the areas of cloud elasticity, geographic distribution, and reducing disaster recovery. This kind of practice suits well in circumstances where significant change cannot be done immediately because of limited resources, governmental factors, or business continuity needs. Nevertheless, lift-and-shift solutions are only capturing part of the cloud value because applications remain running in the patterns of traditional architecture and fail to leverage the native benefits of the cloud. The security study indicates that for lift-and-shift approaches, organizations struggle to adapt traditional security controls to the cloud, with 56% of executives saying that it is difficult to translate on-premises security policies into cloud infrastructure and 48% concerned about having visibility into activities within cloud-based systems (Fortra, 2024).

Hybrid cloud architectures position IBM i systems as components of broader distributed systems and provide additional flexibility. These patterns continue to leverage the IBM i platforms for core transaction processing and maintaining systems of record, adding new cloud-native services for analytics, customer engagement, integration, and other capabilities. The hybrid integration reference architecture defines the end-to-end patterns for integrating on-premises systems with cloud services across multiple integration layers, including API management gateways that secure, throttle, and perform analytics duties, message-oriented middleware that enables asynchronous integration patterns, and data replication services for bidirectional synchronization (IBM). APIs provide a mechanism for integrations and bidirectional communications between on-premises IBM i systems and cloud-resident components, and reference architecture highlights the importance of consistent security models, comprehensive monitoring across hybrid topologies, and unified governance frameworks that span on-premises and cloud-resident components (IBM).

Data architecture considerations assume critical importance in hybrid scenarios. Strategies must address data synchronization, consistency maintenance, and latency management across distributed components. The Hybrid Integration Architecture provides numerous data integration patterns including synchronous request/reply to meet immediate consistency needs, asynchronous messaging to meet eventual consistency needs, and bulk transfer of data via batch files; it is up to the user to decide which of these to use based on the toleration of latency, the volume of data, the consistency requirements, and network bandwidth considerations 10. Change data capture systems enable near real-time replication of IBM i data to cloud-based data warehouses or data lakes, to support the needs of analytics and reporting without affecting the performance of transactional systems. The event streaming platforms enable low-latency data distribution, which enables business events to be responsive in real-time. To this effect, the reference architecture suggests the use of Apache Kafka or other event streaming technologies that have been engineered to support throughput capacity of over millions of events per second and an order guarantee and exactly-once delivery semantics 10.

Cloud-native IBM i services are starting to emerge as another migration pathway, where vendors provide a fully managed IBM i environment, with automated scaling, patching, and operational management. Such offerings minimize operational overhead while sustaining application compatibility. Organizations can focus on enhancing the business logic rather than managing the infrastructure, thus speeding up modernization efforts, although security research indicates that 52% of organizations express concerns about shared responsibility models in managed service environments and require clear delineation of security obligations between providers and consumers (Fortra, 2024).

CONCLUSION

Gradual upgrading of IBM i systems provides a sensible route for digital transformation—one that honors the significant commercial value contained in old systems while enabling modern features needed for competitive positioning adoption. Using the strangler application pattern and API enablement, an evolutionary change framework lets businesses smoothly move single applications into distributed architectures without interrupting operational continuity or throwing away decades

Publisher: SARC Publisher

of carefully honed corporate logic. Combining DevOps techniques with distributed version control, continuous integration pipelines, and modern development environments transforms traditional IBM i development methods into agile ones, satisfying industry needs. A flexible deployment option addressing performance, security, and compliance needs, hybrid architectures and lift-and-shift relocations provide Every one of these methods demonstrates how IBM i platforms can be turned from solitary, legacy systems into an integrated component of a current digital ecosystem—hence not only maintaining their strategic role in enterprise architecture but also providing mobile apps, cloud services, and partner integrations. This being said, incremental modernization reduces the risk of transformation, shortens the time frame for return on investment, and allows the business to receive continuous value during the modernization journey rather than at the end of it after complete system replacement. Evidence presented throughout this examination confirms the technical feasibility and economic viability of evolutionary transformation approaches based on striking a proper balance between innovation imperatives and operational stability requirements while preserving institutional knowledge and proven business processes constituting irreplaceable organizational assets.

REFERENCES

1. Huntington, T. "2025 IBM i Marketplace Survey Results." *Fortra*, (2024).
2. Availability Digest, "Migrating Legacy Systems: Gateways, Interfaces, & the Incremental Approach." (2007).
3. Brown, K. "Apply the Strangler Fig Application pattern to microservices applications." *IBM Developer*, (2020).
4. Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. "Microservices: yesterday, today, and tomorrow." *Present and ulterior software engineering* (2017): 195-216.
5. Khatri, N. "Modernize IBM i: Seamless Integration Through APIs." *IBM Community Blog*, (2024).
6. Masse, M. "REST API design rulebook: designing consistent RESTful web service interfaces." *O'Reilly Media, Inc.*, (2011).
7. Eradani, "DevOps for IBM i: The Ultimate Guide to Modernizing Developer Skills." (2024).
8. Kim, G. *et al.*, "The DevOps Handbook: Introduction, Part I and Part II," *IT Revolution Press*, (2016).
9. Fortra, "Download the 'State of IBM i Security Study,' (2024).
10. IBM, Cloud Architecture Center, "Hybrid Integration Reference Architecture," *IBM Cloud Architecture*.

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:
Allam, S. " Incremental Modernization: APIs, DevOps, and Cloud-Readiness in IBM i Environments." *Sarcouncil Journal of Engineering and Computer Sciences* 4.11 (2025): pp 168-174.

Publisher: SARC Publisher