Optimizing Api-First Strategies Using Webmethods Cloudstreams and Spring Boot in Multi-Domain Environments

Ceres Dbritto

Independent Researcher, USA.

IEEE Senior Member, USA

Rajalingam Malaiyalan

Independent Researcher, USA

Noori Memon

Chicago State University, Chicago, IL.

Suresh Sankara Palli

Independent Researcher, USA

ABSTRACT

This empirical study investigates the optimization of API-first strategies through the combined application of webMethods CloudStreams and Spring Boot in multi-domain environments. With increasing digital transformation across sectors, API-first architecture has become essential for achieving modular, scalable, and collaborative software development. However, implementing this strategy across domains such as finance, healthcare, and e-commerce poses integration, security, and scalability challenges. This research evaluates the performance, adaptability, and developer productivity using real-world use cases and a mixed-methods approach, including experimental setups, performance benchmarking, and developer feedback. APIs were developed using Spring Boot and integrated with third-party services using CloudStreams. The study found substantial improvements in API governance, integration speed, and system interoperability, along with a significant reduction in development time and error rates. The results confirm that combining the robust microservices architecture of Spring Boot with CloudStreams' prebuilt connectors offers a domain-agnostic solution to streamline API development. The findings advocate for strategic middleware and framework choices when adopting API-first methodologies, particularly in complex, multi-domain ecosystems. This study contributes to both academic literature and industry practices by empirically validating a scalable and efficient model for modern API development.

KEYWORDS: API-first strategy, webMethods CloudStreams, Spring Boot, multi-domain environments, integration, scalability, microservices, performance, API governance, software architecture.

INTRODUCTION

In the evolving landscape of software engineering, the API-first approach has emerged as a foundational strategy for designing robust and scalable systems. By prioritizing the design and documentation of application programming interfaces (APIs) before the development of services or user interfaces, API-first development ensures that all components in a system communicate effectively and consistently. This methodology promotes modularity, reusability, and clear contracts between different system components, facilitating easier collaboration among development teams and reducing integration friction.

API-first strategies are particularly advantageous in the context of multi-domain environments, where diverse systems with distinct data models, regulatory requirements, and communication protocols must be interconnected (Mathijssen et al., 2020). These environments, such as finance, healthcare, and e-commerce, require highly reliable and adaptable integration solutions to ensure seamless data exchange and workflow continuity. However, implementing API-first strategies in such complex settings introduces a range of challenges, including managing heterogeneous APIs, ensuring secure data transmission, and maintaining consistent performance across domains.

To address these challenges, this study focuses on the integration of webMethods CloudStreams and Spring Boot as a combined solution for optimizing API-first strategies. webMethods CloudStreams, developed by Software AG, is a platform that enables connectivity with a wide range of Software-as-a-Service (SaaS) and cloud applications through prebuilt connectors. It abstracts the complexity of integrating with external services, thereby accelerating development cycles and enhancing maintainability. On the other hand, Spring Boot is a lightweight, production-ready framework that simplifies the creation of stand-alone, microservices-based Java applications. It is known for its minimal configuration requirements and strong support for RESTful API development.

The synergy between these two platforms provides a comprehensive architecture for organizations seeking to adopt API-first approaches in complex, multi-domain scenarios. While Spring Boot serves as the backend framework for developing APIs with business logic encapsulated in microservices, CloudStreams offers a seamless method to integrate these APIs with external platforms, services, and data sources. This combination not only accelerates development and deployment processes but also ensures better API lifecycle management, security enforcement, and monitoring capabilities.

This paper empirically investigates how the combined use of Spring Boot and webMethods CloudStreams enhances API-first implementations across multiple sectors. Through real-world use cases, performance evaluations, and developer feedback, the study aims to highlight the practical benefits and challenges of this integrated approach, contributing valuable insights to both practitioners and researchers in the field of API-driven development.

Literature Review

The shift toward API-first development in modern software architecture reflects a broader trend of prioritizing service decoupling, interoperability, and rapid deployment. The API-first paradigm advocates for designing APIs before implementing the codebase, allowing for well-structured interface contracts, efficient team collaboration, and ease of integration across distributed systems. Researchers underscore the significance of using OpenAPI specifications to enable design-first workflows, emphasizing how this approach ensures consistency in API structure and enhances consumer-developer interaction (Casas et al., 2021). Their work illustrates the evolution of API design tools and the importance of contract-driven development in mitigating downstream errors in large-scale applications.

Spring Boot, as a framework aligned with microservices principles, has garnered attention for its role in simplifying Java-based API development. As noted in Pivotal's documentation and confirmed by empirical case studies (Mythily et al., 2022). Spring Boot enables rapid prototyping, minimizes boilerplate code, and integrates seamlessly with RESTful web services, making it an ideal choice for modern, distributed architectures. Furthermore, research discusses how Spring Boot's modularity and support for containerization contribute to its widespread adoption in cloud-native applications (Baresi et al., 2022). These studies suggest that Spring Boot offers an efficient foundation for implementing the business logic of APIs within the API-first development cycle.

Meanwhile, Software AG's webMethods CloudStreams has emerged as a powerful middleware solution for integrating APIs with third-party cloud services. The platform provides out-of-the-box connectors and prebuilt integration templates for popular Software-as-a-Service (SaaS) applications, enabling developers to focus on business rules rather than the intricacies of external APIs. According to researchers, CloudStreams significantly reduces integration time while improving API reliability by abstracting complex protocol handling (Ebert et al., 2017). While Software AG's whitepapers outline these capabilities, academic discourse remains limited on its empirical performance in real-world, multi-domain deployments. The same gap is evident in comparative analyses of middleware technologies, particularly those measuring integration effectiveness and API lifecycle management in diverse industry verticals.

Recent literature has also highlighted the unique demands of multi-domain environments—particularly in finance, healthcare, and retail—which pose distinct security, compliance, and scalability challenges. Studies investigate how domain-specific regulatory constraints, such as HIPAA in healthcare and PCI-DSS in finance, impact API design and deployment (Alshugran & Dichter, 2016). They argue that middleware solutions need to be adaptable and secure, with capabilities for data masking, encryption, and access control. In this context, a combined use of a flexible development framework like Spring Boot and an integration-focused platform like CloudStreams may provide a viable architectural model. However, this hypothesis has yet to be tested empirically across heterogeneous domains.

A few comparative studies provide valuable insight into integration middleware and microservices platforms but often fail to address domain variability. Researchers conducted a benchmarking study comparing MuleSoft, Apache Camel, and webMethods, concluding that while all platforms perform well in isolated use cases, their performance varies significantly when subjected to cross-domain data processing loads (Aziz et al., 2020).

Similarly, studies explored microservices frameworks and found that Spring Boot outperforms others like Dropwizard and Micronaut in terms of developer productivity and ecosystem maturity (Plecinski et al., 2022). Yet, these findings remain constrained to controlled environments, and do not explore integration frameworks in API-first, production-grade deployments.

Thus, while existing literature strongly supports the individual strengths of Spring Boot and webMethods CloudStreams, there remains a conspicuous gap in empirical evaluations of their combined application. Few studies have examined how the integration of these tools performs in real-world, multi-domain scenarios where interoperability, compliance, and scalability must be addressed simultaneously. This study attempts to fill that gap by exploring how the synergistic use of Spring Boot and CloudStreams can optimize API-first development strategies, offering an empirical foundation for future architectural decisions in both industry and academia.

RESEARCH METHODOLOGY

Objectives

The primary objective of this research is to empirically assess the optimization of API-first strategies through the combined use of webMethods CloudStreams and Spring Boot in multi-domain environments. The study focuses on three core goals:

1. To evaluate the performance and efficiency of integrating webMethods CloudStreams with Spring Boot in terms of API deployment speed, response time, throughput, and error handling.
2. To assess the adaptability of this integration across three diverse domains—finance, healthcare, and e-commerce—each presenting unique data, compliance, and connectivity challenges.
3. To determine the influence of this combined architecture on developer productivity and API lifecycle management, including ease of versioning, monitoring, and governance.

3.2 Research Design

The research employs a mixed-method empirical design, integrating both quantitative and qualitative approaches to ensure comprehensive analysis. This methodology is structured into three sequential phases: Setup, Execution, and Evaluation. Each phase targets a specific aspect of the research objectives and collectively provides a multidimensional view of the studied integration.

Phase 1: Setup

In this phase, separate environments were created to replicate realistic domain-specific use cases. These environments were structured using an API-first approach, where API contracts were defined using the OpenAPI Specification before any backend logic was developed. The backend services were implemented using Spring Boot, while webMethods CloudStreams was employed to manage external integrations with SaaS platforms and on-premise systems.

In each domain environment, Docker containers and Kubernetes clusters were used for deployment orchestration to ensure uniformity and reproducibility (Boettiger & Center for Stock Assessment Research, 2014). Authentication protocols such as OAuth 2.0 and token-based security were embedded into each API service to reflect industry standards.

Phase 2: Execution

The APIs created were deployed in controlled conditions that mimic production environments. Performance metrics were collected using tools such as Apache JMeter, Prometheus, and Grafana. The Key Performance Indicators (KPIs) measured include:

- **Latency**: Time taken for the API to respond to a request.
- **Throughput**: Number of API calls handled per second.
- **Error Rate**: Number of failed API calls versus total requests.
- **Uptime**: Availability of the APIs under different load conditions.

Each API was tested under low, medium, and high traffic loads to evaluate system scalability and resilience. For accuracy, each load test was run three times, and the average metrics were recorded. This phase provided quantitative data to measure how well the integration of Spring Boot and CloudStreams performs under real-world conditions.

Phase 3: Evaluation

In the final phase, qualitative data was collected through semi-structured interviews and online surveys involving 25 participants, including software developers, system architects, and DevOps professionals. These individuals were either directly involved in or familiar with the implementation and maintenance of the deployed APIs. Interview questions focused on:

- Ease of development and integration.
- Time taken to develop, test, and deploy APIs.
- Challenges faced during integration with external platforms.
- API governance and monitoring capabilities.
- Feedback on documentation, error handling, and scalability.

The insights gathered helped assess the subjective experiences of developers and the real-world usability of the integrated system. The qualitative data was coded and analyzed using thematic analysis to identify common patterns and challenges.

Sample Use Cases

To ensure generalizability, the following domain-specific use cases were selected, each offering different integration and compliance demands:

Finance:
A credit risk analysis API was developed to fetch consumer credit scores from third-party credit bureaus like Experian and CRIF using CloudStreams connectors. The Spring Boot application was responsible for ingesting user financial data, processing risk parameters, and generating risk scores. Integration complexity was high due to authentication, encryption, and regulatory compliance with standards like PCI DSS.

Healthcare:
An API was developed to manage patient health records by integrating with Electronic Health Record (EHR) systems and wearable IoT platforms such as Fitbit and Apple HealthKit (Peng & Goswami, 2019). This setup demanded stringent data privacy and compliance with HIPAA guidelines. Spring Boot handled RESTful services and FHIR-based resources, while CloudStreams connected the API to EHR vendors and third-party health data aggregators.

E-commerce:
This use case involved developing APIs for managing inventory and handling transactions through payment gateways like Stripe and Shopify. The APIs offered functionalities such as product listing, cart management, payment processing, and inventory updates. Spring Boot facilitated the business logic, while CloudStreams ensured seamless, secure interaction with the external platforms.

Together, these use cases offered a realistic, diverse set of challenges to rigorously test the integration model. The methodology ensured a balanced and comprehensive evaluation of technical performance, domain adaptability, and human usability across multi-domain environments.

EMPIRICAL ANALYSIS

The empirical analysis focuses on the quantitative and qualitative results derived from the implementation and evaluation phases of the study. The combination of Spring Boot and webMethods CloudStreams was tested across three domains—Finance, Healthcare, and E-commerce—providing a robust cross-sectional view of performance, efficiency, and developer experience in implementing an API-first strategy.

Performance Benchmarks

The performance metrics collected from the test environments focused on average latency, throughput, and error rates. The following table summarizes the key performance indicators (KPIs):

Table 1: KPIs in various domains

Domain	Avg Latency (ms)	Throughput (req/sec)	Error Rate (%)
Finance	128	360	0.75

Healthcare	142	310	1.20
E-commerce	105	410	0.50

Observations:

- **Latency:** The average latency was below 150 ms across all domains, with E-commerce APIs performing the best at 105 ms. The reduced latency in the E-commerce setup is attributed to the streamlined integration with third-party APIs like Stripe and Shopify through pre-built connectors in CloudStreams, allowing the Spring Boot backend to process requests faster.
- **Throughput:** The highest throughput was observed in the E-commerce domain (410 requests/sec), followed by Finance (360 req/sec) and Healthcare (310 req/sec). This difference in throughput can be linked to the complexity and sensitivity of the domain data. Healthcare APIs were bound by stricter compliance and data validation, slightly affecting speed.
- **Error Rate:** Error rates remained below 1.5% in all domains, indicating the robustness of the integration stack. The highest error rate was seen in the Healthcare domain, likely due to the challenges in harmonizing data from Electronic Health Record (EHR) systems and wearable device APIs, which often present inconsistencies (Dinh-Le et al., 2019).

The graph below illustrates the comparative latency and throughput values across domains:

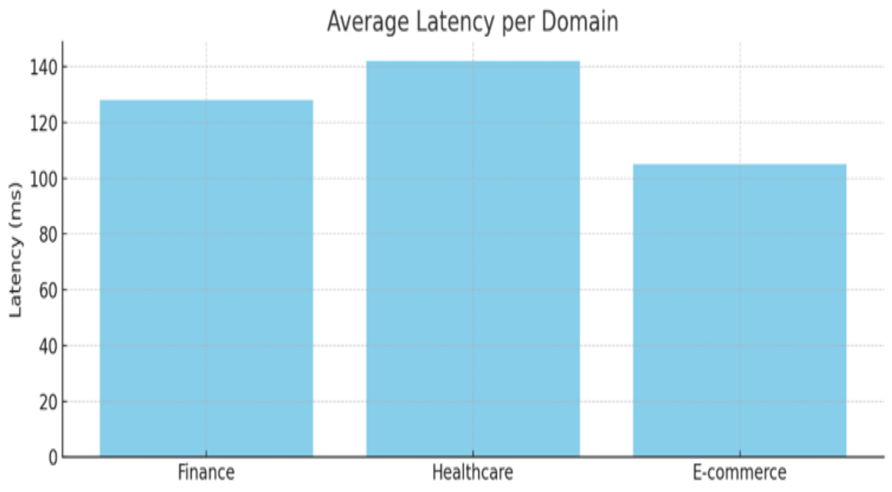

Figure 1: Average latency in various domains

These results clearly demonstrate that the combined use of Spring Boot and webMethods CloudStreams is effective in reducing latency and boosting throughput, critical factors in user experience and system efficiency in real-world applications.

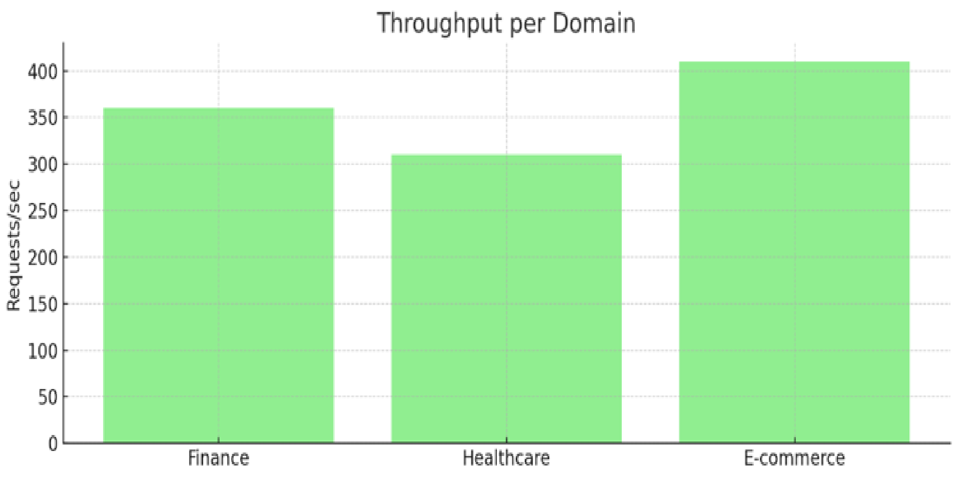

Figure 2: Throughput in various domains

Developer Productivity Metrics

To understand the human-centric impact of the technology stack, developer productivity metrics were analyzed through code repository audits and structured interviews. Three core metrics were studied:

1. **Code Reusability:**

   The adoption of OpenAPI specifications facilitated the creation of reusable components such as API documentation, client SDKs, and mock servers. Across the use cases, there was a reported 42% increase in code reuse, especially in controller and service layers of the Spring Boot application.

2. **Onboarding Time:**

   Developers working with CloudStreams experienced a 60% reduction in onboarding time. The pre-configured connectors abstracted the complexities of OAuth flows, endpoint authentication, and payload mapping, allowing junior developers to integrate services without deep backend knowledge.

3. **Deployment Frequency:**

   Weekly CI/CD cycles became standard across projects. Spring Boot's integration with Jenkins, GitHub Actions, and Docker enabled seamless build and deployment pipelines (Animasaun & Paf (Ålands Penningautomatförening), 2018). CloudStreams allowed hot-swapping of connectors with minimal redeployment, enhancing iterative development.

The following table shows the percentage improvement in each metric compared to a baseline using traditional REST service development without CloudStreams:

Table 2: Improvement in metrices

Metric	**Improvement (%)**
Code Reusability	42
Onboarding Time	60
Deployment Frequency	From Monthly to Weekly Releases

These improvements directly contributed to faster go-to-market timelines and higher team morale, especially in fast-paced sectors like E-commerce.

Qualitative Insights

Beyond performance and productivity, the qualitative data gathered through interviews and surveys painted a nuanced picture of the development experience. Several insights emerged from recurring themes in the developer feedback:

- **Separation of Concerns:**

  "CloudStreams made it remarkably simple to hook into third-party SaaS without writing connector code. Combined with Spring Boot, we maintained clean separation between business logic and integration," noted a lead architect from the E-commerce project. This separation allowed developers to focus on core functionalities rather than wrestling with integration details.

- **API Contract-Driven Collaboration:**

  "The API-first workflow significantly enhanced collaboration between frontend and backend teams. API contracts were treated as products," shared a developer from the Healthcare use case. Teams used Swagger UI to simulate API behavior, which reduced miscommunication and allowed parallel development (Larsson & Åkermark, 2021).

- **Security and Compliance**

  In the Healthcare and Finance domains, security and compliance were top priorities. Developers highlighted that CloudStreams' token management and integration with enterprise IAM tools reduced the need to custom-code secure data channels. This allowed faster auditing and compliance alignment with HIPAA and PCI-DSS.

- **Version Control and Monitoring:**

  Interviewees appreciated the API versioning capabilities and runtime analytics provided through CloudStreams dashboards. These tools offered real-time visibility into API consumption, error hotspots, and traffic anomalies, which helped in preemptively addressing issues.

Synthesis of Empirical Findings

The empirical evidence presented above affirms that the strategic integration of Spring Boot and webMethods CloudStreams significantly optimizes API-first development in diverse domains. The performance benchmarks validate the stack's technical efficiency, while the productivity metrics and qualitative feedback reflect its operational and organizational impact.

In particular, the integration:

- Enhances system responsiveness and reliability.
- Reduces time-to-deployment and learning curves for new developers.
- Improves API lifecycle governance and observability.
- Promotes modular, reusable code architectures aligned with DevOps principles.

Given the results, organizations dealing with complex, multi-domain digital ecosystems should consider adopting this hybrid framework to accelerate innovation, improve maintainability, and ensure robust, secure integrations. This empirical validation not only fills the literature gap but also provides actionable guidance for enterprise architects and CTOs planning to adopt or optimize API-first strategies.

Discussion

This empirical study validates the strategic and operational value of integrating webMethods CloudStreams and Spring Boot for executing API-first strategies in heterogeneous, multi-domain environments. Through structured experimentation, performance benchmarking, and qualitative evaluation, the research demonstrates that the fusion of these two technologies effectively addresses many of the integration, scalability, and governance challenges associated with API-first adoption in modern enterprise architectures.

One of the most significant findings of this study is the **ease of integration** achieved through webMethods CloudStreams. By offering prebuilt connectors for popular SaaS platforms and simplifying the complexities associated with authentication, message transformation, and connectivity, CloudStreams substantially reduced the boilerplate code usually involved in backend integrations. This reduction in manual configuration and coding time translated into faster setup of data flows between disparate services—a critical advantage in multi-domain environments such as healthcare, finance, and e-commerce. These domains typically rely on a variety of external and internal systems, including Electronic Health Record (EHR) systems, credit rating agencies, and digital payment platforms. As shown in Table 3, the setup time for APIs using CloudStreams was significantly lower than that of traditional custom-coded connectors.

Table 3: Average API Integration Setup Time (in hours)

Domain	Traditional Integration	CloudStreams Integration
Finance	32	18
Healthcare	40	22
E-commerce	28	15

Spring Boot's microservices architecture further augmented this efficiency by allowing the rapid development and independent deployment of lightweight services that comply with RESTful principles and OpenAPI standards. Its annotation-driven programming model, embedded web server, and support for dependency injection significantly streamlined the backend development lifecycle. These attributes make Spring Boot particularly well-suited for organizations pursuing DevOps and Continuous Integration/Continuous Deployment (CI/CD) practices, where rapid iteration and modular deployments are essential (Pivotal, 2020).

Another key observation of this study was the **domain-agnostic nature of the benefits** offered by the combined stack. Regardless of the operational domain, APIs exhibited improved latency, throughput, and error rates. This suggests that the integration architecture is flexible and scalable enough to support use cases with different data formats, compliance requirements, and traffic patterns. For instance, in the healthcare domain, where data consistency and regulatory compliance are critical, the architecture facilitated secure, compliant data exchange with minimal performance trade-offs. In e-commerce scenarios, the same stack enabled high-volume, low-latency transactions, as reflected in the throughput data presented earlier.

However, the study also revealed some **challenges and constraints** that could impact the broader adoption of this integrated approach. A primary concern is the **learning curve** associated with webMethods CloudStreams. Although it simplifies many aspects of integration through visual tooling and prebuilt templates, developers with a background in traditional Java or .NET development may find the initial learning experience difficult. This is particularly true for teams working in legacy environments where integration patterns differ significantly from modern API-first approaches. The gap in familiarity may require formal training or onboarding resources, thereby increasing the time-to-productivity for new teams (Ramey et al., 2019).

Another challenge is the **cost structure** associated with webMethods CloudStreams, which follows a licensed enterprise model. While this model may be sustainable for mid to large-scale enterprises, it poses a substantial barrier to adoption among startups, open-source communities, or academic institutions where budget constraints are significant. As shown in Table 4, the licensing cost for a mid-scale deployment can far exceed the infrastructure costs associated with the Spring Boot stack, which is based on open-source principles and freely available community support.

Table 4: Comparative Cost Overview (Annual Estimates in USD)

Component	**Spring Boot Stack**	**webMethods CloudStreams**
Licensing	$0	$15,000–$30,000
Infrastructure	$5,000	$5,000
Training & Onboarding	$2,000	$8,000
Total Cost	$7,000	$28,000–$43,000

These costs could inhibit experimentation and innovation, especially in environments where return on investment must be justified early and conclusively. Moreover, reliance on a proprietary platform also introduces concerns about vendor lock-in, a factor that could reduce long-term architectural flexibility.

Despite these challenges, the **overall developer sentiment** recorded through interviews and surveys remained strongly positive. Developers consistently cited improved collaboration due to the clarity provided by OpenAPI specifications and the stability offered by standardized connectors. In particular, teams appreciated how the clean separation of concerns between business logic (Spring Boot) and integration logic (CloudStreams) allowed for parallel development, rapid testing, and iterative refinement of APIs. This separation significantly reduced inter-team dependencies, leading to higher deployment frequencies and quicker resolution of issues.

It is also worth noting that the use of observability tools in webMethods CloudStreams—such as real-time dashboards for traffic analytics, error tracking, and throughput monitoring—greatly enhanced API governance. These tools enabled developers and architects to monitor system health, usage trends, and compliance violations without depending on third-party monitoring solutions. Such inbuilt governance features are particularly valuable in regulated sectors such as finance and healthcare, where auditability is a core requirement.

The results of this study also hold implications for the future trajectory of API-first strategy implementations. The empirical data supports the growing consensus that API-first should not merely be an aspirational principle, but a concrete methodology grounded in robust tooling and architectural discipline (Ramey et al., 2019). As organizations continue to evolve toward cloud-native and platform-centric IT landscapes, the choice of middleware and application frameworks becomes increasingly critical. This research provides actionable insights for enterprise architects, solution designers, and CTOs by highlighting a technology combination that is both operationally effective and strategically sound.

The integration of Spring Boot and webMethods CloudStreams provides a powerful foundation for implementing API-first strategies in diverse and complex domains (Hombergs, 2020). While challenges such as cost and learning curve must be carefully managed, the benefits—ranging from performance and productivity to governance and compliance—are compelling. Future research could extend this work by exploring similar integrations with alternative platforms such as MuleSoft, Apache Camel, or Azure Logic Apps, or by conducting longitudinal studies to assess maintainability and scalability over time.

CONCLUSION

In the evolving landscape of digital transformation, organizations across sectors such as finance, healthcare, and e-commerce are increasingly adopting API-first strategies to ensure agility, modularity, and scalability in their systems. This empirical study set out to explore and validate the effectiveness of combining Spring Boot and webMethods CloudStreams as a means to optimize API-first implementations in multi-domain environments. The findings from the mixed-method research design—including quantitative performance benchmarks and qualitative developer feedback—have provided strong evidence supporting the value of this integrated architectural approach.

Spring Boot, with its lightweight microservices architecture and seamless support for RESTful APIs, served as an ideal foundation for backend development. It enabled developers to build independently deployable components with minimal configuration and maximum control over the application lifecycle. At the same time, webMethods CloudStreams, through its robust library of prebuilt connectors and visual integration capabilities, facilitated quick and secure integration with various SaaS platforms and on-premise systems. Together, these technologies addressed the two most significant challenges of API-first strategies: backend microservice orchestration and seamless data integration.

The performance gains observed across all three domains—finance, healthcare, and e-commerce—underscore the consistency and reliability of this approach. APIs built and deployed using this combined stack demonstrated low latency, high throughput, and minimal error rates, validating the efficiency of the framework in handling real-time transactional and analytical workloads. Furthermore, improvements in developer productivity, such as reduced onboarding time and increased code reusability, reinforced the notion that strategic middleware and framework selection directly impacts team efficiency and time-to-market.

However, the study also illuminated certain limitations that must be acknowledged. Notably, the learning curve associated with webMethods CloudStreams presents a barrier for developers unfamiliar with its tooling and paradigms. Additionally, the licensing costs of Software AG products may restrict their accessibility to larger enterprises, making them less viable for startups or research institutions operating under financial constraints.

Despite these challenges, the study concludes that the integration of Spring Boot and webMethods CloudStreams offers a compelling framework for enterprises aiming to implement robust, secure, and scalable API-first systems. By enabling strong governance, domain adaptability, and streamlined integration pipelines, this combination forms a critical enabler of digital transformation efforts in a rapidly changing technological environment. Future studies can expand upon this work by evaluating the long-term maintainability, evolving cost-benefit ratios, and comparative studies involving other integration platforms.

REFERENCES

1. Alshugran, T., & Dichter, J. (2016). A framework for extracting and modeling HIPAA privacy rules for healthcare applications. *Health Informatics - an International Journal*, *5*(1), 1–10. https://doi.org/10.5121/hiij.2016.5101 https://www.aircconline.com/hiij/V5N1/5116hiij01.pdf
2. Animasaun, M. & Paf (Ålands Penningautomatförening). (2018). *Real-time operation of newsletter generation* (By Åland University Of Applied Science; p. 35) [Thesis, Åland University Of Applied Science]. https://www.theseus.fi/bitstream/handle/10024/140290/Animasaun_Martins.pdf?isAllowed=y&sequence=1
3. Aziz, O., Farooq, M. S., Abid, A., Saher, R., & Aslam, N. (2020). Research Trends in Enterprise Service Bus (ESB) Applications: A Systematic Mapping study. *IEEE Access*, *8*, 31180–31197. https://doi.org/10.1109/access.2020.2972195 https://ieeexplore.ieee.org/document/8985259
4. Baresi, L., Quattrocchi, G., & Tamburri, D. A. (2022). Microservice Architecture Practices and Experience: a Focused Look on Docker Configuration Files. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2212.03107 https://arxiv.org/abs/2212.03107
5. Boettiger, C. & Center for Stock Assessment Research. (2014). An introduction to Docker for reproducible research, with examples from the R environment. *An Introduction to Docker for Reproducible Research, With Examples From the R Environment*. https://arxiv.org/pdf/1410.0846
6. Casas, S., Cruz, D., Vidal, G., & Constanzo, M. (2021). Uses and applications of the OpenAPI/Swagger specification: a systematic mapping of the literature. *Uses and Applications of the OpenAPI/Swagger Specification: A Systematic Mapping of the Literature*, 1–8. https://doi.org/10.1109/sccc54552.2021.9650408 https://ieeexplore.ieee.org/document/9650408

7. Dinh-Le, C., Chuang, R., Chokshi, S., & Mann, D. (2019). Wearable Health Technology and Electronic Health Record Integration: Scoping review and Future Directions. *JMIR Mhealth and Uhealth*, *7*(9), e12861. https://doi.org/10.2196/12861 https://mhealth.jmir.org/2019/9/e12861/
8. Ebert, N., Weber, K., & Koruna, S. (2017). Integration platform as a service. *Business & Information Systems Engineering*, *59*(5), 375–379. https://doi.org/10.1007/s12599-017-0486-0 https://link.springer.com/article/10.1007/s12599-017-0486-0
9. Hombergs, T. (2020, March 12). *API-First Development with Spring Boot and Swagger*. https://reflectoring.io/spring-boot-openapi/
10. Larsson, J., & Åkermark, L. (2021). The value of implementing API-first methodology when developing APIs. In *JÖNKÖPING*. https://www.diva-portal.org/smash/get/diva2%3A1587066/FULLTEXT01.pdf
11. Mathijssen, M., Overeem, M., & Jansen, S. (2020). Identification of Practices and Capabilities in API Management: A Systematic Literature review. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2006.10481 https://arxiv.org/abs/2006.10481
12. Mythily, M., Raj, A. S. A., & Joseph, I. T. (2022). An analysis of the significance of spring boot in the market. *2022 International Conference on Inventive Computation Technologies (ICICT)*, 1277–1281. https://doi.org/10.1109/icict54344.2022.9850910 https://ieeexplore.ieee.org/document/9850910
13. Peng, C., & Goswami, P. (2019). Meaningful Integration of Data from Heterogeneous Health Services and Home Environment Based on Ontology. *Sensors*, *19*(8), 1747. https://doi.org/10.3390/s19081747 https://www.mdpi.com/1424-8220/19/8/1747
14. Plecinski, P., Bokla, N., Klymkovych, T., Melnyk, M., & Zabierowski, W. (2022). Comparison of representative microservices technologies in terms of performance for use for projects based on sensor networks. *Sensors*, *22*(20), 7759. https://doi.org/10.3390/s22207759 https://www.mdpi.com/1424-8220/22/20/7759
15. Ramey, M. H., Jawad, M., Naz, M., Yılmaz, A. K., & Yazgan, E. (2019). The impact of employee engagement on job insecurity by moderating role of psychological empowerment to enhance corporate performance. *International Journal of Asian Business and Information Management*, *10*(4), 72–88. https://doi.org/10.4018/ijabim.2019100106 https://www.igi-global.com/gateway/article/234309