ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may695

Toward Self-Adaptive REST APIs: Enhancing Spring Boot Microservices with Intelligent Runtime Optimization

Muka Kabeya Arsene[1]; Oshasha Oshasha Fiston[2]; Musas A Musas Andre[3]; Kabeya Mukosayi Jospeh[4]; Tshielo Koka Souvient[5]; Kobalanga Liamba Pathy Cedrick[6]

[1,2,3,4,5,6] Commissariat General A L'energie Atomique / Centre Regional D'etudes, Nucleaires De Kinshasa (Cgea/Cren-K)

Publication Date: 2026/05/23

Abstract: The complexity of distributed systems is ever growing, and cloud-native architectures have become the de facto standard for modern software engineering, further solidifying REST APIs as its centerpiece. Thanks to the availability of frameworks like Spring Boot, programmers can develop microservices that are scalable and modular at a rapid pace. However, although flexible, most API frameworks are statically configured and do not support dynamic adaptation at runtime to changes in workload, user behaviour, or infrastructure environment. In this paper, we present a complete self-adaptive REST API paradigm by the means of incorporating live observability, smart decision-making, and runtime reconfiguration within Spring Boot-based microservices. The architecture proposed in this paper brings a feedback-driven approach which keeps track of the system's behavior, context-aware data analysis, and incremental adaptions at runtime so as to enhance its performance, resiliency, and resource consumption. By complementing classical API design techniques with adaptative intelligence, this work paves the way towards the potential next generation of autonomous software systems that are expected to run successfully in highly dynamic and unforeseeable environments.

How to Cite: Muka Kabeya Arsene; Oshasha Oshasha Fiston; Musas A Musas Andre; Kabeya Mukosayi Jospeh; Tshielo Koka Souvient; Kobalanga Liamba Pathy Cedrick (2026) Toward Self-Adaptive REST APIs: Enhancing Spring Boot Microservices with Intelligent Runtime Optimization. *International Journal of Innovative Science and Research Technology*, 11(5), 1328-1330. https://doi.org/10.38124/ijisrt/26may695

I. INTRODUCTION

The industry-wide digitalization has contributed to a growing dependence on distributed systems, where applications are broken down into loosely coupled services that interact with each other via REST APIs. Such APIs form the backbone of the modern software ecosystem and facilitate the interaction of services running on disparate environments such as cloud, edge, hybrid, and more. The development of microservices architecture has increased the importance of APIs, as they are also be seen as the smallest scalable and maintainable unit.

Spring Boot and similar frameworks have made API development manageable without the need for excessive configuration. Today, developers can build production-ready services with little fuss, thanks to such concepts as embedded servers, dependency injection, and auto-configuration. But the design principles of these systems are largely unchanging, and their configuration is done once at deployment and is very rarely altered in real time.

This rigid nature imposes a grievous limitation which is glaring, if not fatal, in an age of volatility and uncertainty. User demand is unpredictable, network conditions may deteriorate, and system resources may be limited. In these environments, where such slowdowns are occurring with increasing frequency, fixed APIs tend to become under-optimized, with corresponding increases in latency, decline in user experiences, and reduction in resource efficiencies. We make the case that the next generation of API design should be to embrace adaptability as a first principle design path to allow systems to intelligently respond to situational changes in real time.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may695

II. EVOLUTION OF API ARCHITECTURES AND EMERGING CHALLENGES

The development of API architectures has been driven by the requirements of scalability, modularity and fault tolerance. The components within early monolithic systems were tightly coupled, so they were difficult to scale and maintain. While the move to service-oriented architecture brought another layer of modularity, it was the rise of microservices that really disrupted the market by enabling services to be independently deployed and scaled.

Yet, related problem in addition to those challenges, microservices architectures also pose additional challenges in two areas: coordination, communication overhead and system observability. As the services grow in numbers, so do the difficulty in managing the interactions between them. REST APIs, despite having standardized communication interface, often become bottlenecks under heavy load or poorly designed.

One of the major difficulties with modern API platforms is that they cannot dynamically respond to runtime situations. For example, cache policies are usually set in stone and are not adaptive to changes in access patterns. In the same way, rate limitations are static and do not respond to changes in the user features. These constraints suggest that a new type of API is required that can be continuously evolving in response to its environment.

III. CONCEPTUAL FOUNDATIONS OF SELF-ADAPTIVE SYSTEMS

Self-adaptive systems have the following capabilities: they can perceive the environment, detect changes, and adapt themselves to the environment. In fact, this idea was borrowed from autonomic computing [3], which proposed systems that are self-configuring, self-healing, self-optimizing and self-protecting. Implementing these ideas for REST APIs demands a paradigm shift in design--from static, one-size-fits-all policies to dynamic, real-time feedback based policies.

The fundamental concept in self-adaptive APIs is a continuous feedback loop that bridges system observation to decision making and action. Through this loop the system is able to learn from its conduct and to adapt itself. In API systems, it means measuring performance, detecting anomalies, and applying tuning kick dynamically to their behaviors.

- *Advanced Architecture for Self-Adaptive APIs*

The architecture proposed here generalizes traditional microservices into an invasive multi-layer adaptive framework. At the highest level, the architecture is centered on a continuous feedback loop comprised of observability, intelligence and dynamic control.

The observability plane gathers fine-grained runtime information from a variety of sources, such as application logs, metrics and distributed traces. Such a layer gives a high-level representation of the system from which performance bottlenecks or system disturbances can be identified. Integration of observability tools in Spring Boot enables smooth data aggregation and analysis.

The data that is collected is then processed by the intelligence tier using sophisticated analytics and machine learning methods. This level finds patterns, predicts future states of the system, and selects the best adaptation actions. The system uses predictive models to anticipate future changes in workload and to take preemptive actions.

Decisions of adaptation are realized in the control layer, through real-time changes of system configurations. This involves modification of thread pools, reconfiguration of routing strategies, or dynamic resource scaling. The dynamics between the two layers define a stabilizing feedback loop with an effective performance on the time-varying phenomena.

- *Observability and Monitoring in Adaptive APIs*

Observability also contributes to making self-adaptation possible. With no accurate, timely information, the system is blind to making educated decisions. The observability of today is more than just monitoring - it gives you a very high-level view of system behavior via metrics, log, and traces.

Observability in Spring Boot There are a number of tools that enable observability in Spring Boot-based applications by providing detailed performance information. These monitors give developers visibility into critical metrics including response time, throughput, and error rates. By performing pattern analysis on these metrics, the system can discover trends and anomalies that are worthy of adaptation.

- *Dynamic Configuration and Runtime Adaptation*

Dynamic configuration is a necessity in order to realize self-adaptive APIs. Dynamic config can be changed at runtime without restarting the app, contrary to static config which are set at deployment time. This allows the system to adapt rapidly to changing environments.
When using Spring Boot, dynamic configuration can be achieved by leveraging centralized configuration services that distribute changes to multiple nodes. This produces consistency and allows for coordinated adaptation among a suite of services.

- *Scalability and Load Management*

Scalability is too important a question for modern API systems, especially in cloud-native environments. Self-Adaptive API (SA-API) improves the API scalability by adapting the API provisioning according to the current demand for resources. This may involve scaling instances, moving workloads around, or consolidating resources.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may695

The adaptive mechanism can be incorporated with the container orchestration platform to allow an automated scaling and load balancing. The API monitors system performance continuously and knows the best time to scale up or down for an efficient use of the resources.

- *Resilience and Fault Tolerance*

Resilience is yet another advantage of self-adaptive APIs. Through system health/performance monitoring in a loop, the system can identify the failure and attempt recovery immediately. Such actions include rerouting requests, restarting completed services and changing configurations to relieve problems.

Real-time failure adaptation greatly enhances system dependability and performance. This is especially critical for critical applications in which availability is a must.

- *Security Implications of Adaptive APIs*

The Emergence of Dynamicity and Behavior in API Systems Poses the Following Critical Security Questions. Dynamic, adaptive mechanisms must ensure that they are not subject to tampering (i.e., prevent unauthorized modification of system configurations) and that they can verify their integrity. Also, relying on data at runtime to make decisions opens potential privacy issues that need to be solved employing strong data protection.

- *DevOps and Continuous Delivery Integration*

The development of self-adaptive APIs certainly affects the DevOps process. Continuous integration and deployment pipelines need to be modified to accommodate dynamic configuration and runtime modification. This calls for new tools and techniques for integration of development and operations.

- *Multi-Cloud and Distributed Environments*

Nowadays, applications tend to be distributed on multiple clouds, resulting in new challenges for API management. Self-adaptive approaches have the potential to tackle these issues by the real-time adaptation of system functioning in each environment.

- *Experimental Evaluation and Case Study*

In order to prove the proposed method, scenarios of experiments can be planned considered real conditions. Such experiments are to quantify the effect of adaptation mechanisms on performance, scalability, and resilience. Real application case-studies may also gain useful insight in to what it means to implement self-adaptive APIs in practice.

IV. DISCUSSION

The evolution to self-adaptive APIs is a leap forward in software engineering that allows systems to become more efficient and effective in their operation. However, it also brings new problems to complexity, maintainability, and trust. Meeting these challenges will necessitate sustained research and interdisciplinary collaboration.

V. FUTURE RESEARCH DIRECTIONS

Future work may investigate the the combination of more advanced AI techniques, such as deep learning and multi-agent systems, to improve system adaptivity. Also, the creation of generalized frameworks of adaptive APIs will be crucial for the diffusion of such technologies.

VI. CONCLUSION

To the best of our knowledge, this is the first work in the direction of self-adaptive REST APIs in the Spring Boot context. Through bringing observability, intelligent analysis and dynamic adaptation as layers of the APIs stack, the approach provides APIs with the possibility to run in a dynamic fashion in dynamic environments. This contribution demonstrates that adaptability should be considered as a first-class concept in modern software engineering and opens new directions for research in this area.

REFERENCES

[1]. R. Johnson et al., *Spring Framework Reference Documentation*, 2022.
[2]. P. Jamshidi, C. Pahl, and N. C. Mendonça, "Self-Adaptive Microservices: A Systematic Literature Review," *IEEE Software*, vol. 35, no. 3, pp. 56–63, 2018.
[3]. B. Burns and D. Oppenheimer, *Designing Distributed Systems*, O'Reilly Media, 2018.
[4]. M. Fowler, "Microservices: A Definition of This New Architectural Term," 2014.
[5]. N. Dragoni et al., "Microservices: Yesterday, Today, and Tomorrow," Springer, 2017.
[6]. S. Newman, *Building Microservices*, O'Reilly Media, 2021.
[7]. G. Hohpe and B. Woolf, *Enterprise Integration Patterns*, Addison-Wesley, 2003.