International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

.NET Core Vs. Java Spring Boot: A Review-Driven Performance Assessment for Cloud-Native API Architectures

Karthik Sirigiri[1*]

Submitted:10/10/2023 **Revised:**22/11/2023 **Accepted:**02/12/2023

Abstract: The development of cloud-native APIs which allow scalability, resilience, and performance optimization will be absolutely crucial for modern program design. Two of the most widely used frameworks, Java Spring Boot and .Net Core both provide particular advantages for the expansion of microservices born on clouds. Choice of a proper framework determines most importantly performance, resource economy, and scalability in deployment.

This article compares Java Spring Boot with .Net Core using industry benchmarks, historical data, and already published experimental results. Important benchmarks including startup length, request delay, throughput, memory and CPU utilization, containerizing's efficiency, and other security aspects are rigorously evaluated in this work. This study differs from empirical studies by aggregating earlier paper results, so offering a logical evaluation of every paradigm.

Since .Net Core typically provides enhanced startup times, reduced memory consumption, and high throughput, the results show that .Net Core is especially suited for high-performance applications and serverless computing. Conversely, Spring Boot uses strong interaction with Spring Cloud and values robust community support, so showing better suited for corporate use.

This study is to assist developers and software architects in selecting knowledge-based frameworks for cloud-native apps. Future research should consider practical applications if we want to strengthen the theoretical commonalities.

Keywords: *Cloud-Native API Development, Java Spring Boot, .NET Core, Serverless Computing, RESTful Web APIs, Cloud Computing Frameworks, Microservices Architecture, Enterprise Software Scalability, Containerization and Kubernetes.*

Introduction

Importance of Cloud-Native API Development

Modern software engineering largely depends on the ideas of cloud-native API development to improve the scalability, distribution, and resilience of applications in cloud settings. Using microservices architecture forces companies to pursue frameworks marked by improved performance, flexibility, and flawless interface with cloud environments. Load balancing, fault tolerance, and auto-scaling approaches greatly increase the efficacy of cloud-native APIs in enabling cloud-based applications, so improving inter-service communication efficiency.

Three main dimensions help one to explain the importance of cloud-native API development:

Software Developer, Euniverse Technologies, Irving, Texas, USA.

sirigirikarthik25@gmail.com;

- Dynamic scalability of cloud-hosted APIs shows in line with traffic demand, therefore improving resource economy and cost effectiveness.

- Cloud-native apps use retry policies, auto-healing systems, and circuit breaker implementations among various fault tolerance strategies to achieve high availability.

- API services efficiently maximize resource consumption by using containerized deployments on platforms including AWS Lambda, Azure Functions, and Kubernetes clusters, therefore lowering running costs.

Especially when using cloud-native architectural frameworks, the combination of Spring Boot with the natural CI/CD pipeline capabilities of .NET Core effectively reduces deployment problems. Modern security solutions improve application security by including OAuth 2.0, JWT authentication, and API gateways' integration with cloud-native APIs. The rapid use of Kubernetes, serverless computing, and hybrid cloud environments made the choice of an API

development framework critical in defining application performance speed, maintainability, and general security posture.

.NET Core and Java Spring Boot: Overview

A) .NET CORE

Designed by Microsoft, .Net Core is a robust and efficient framework aimed to assist the development of modern, cloud-native apps across various platforms. This application has been painstakingly created to run across Windows, Linux, and macOS platforms free from error. The lightweight and flexible nature of the system helps initiatives targeted at microservices and API-driven development significantly.

The essential characteristics of .NET Core are:

- The Kestrel web server generates high-performance results by significantly improving the request processing efficiency.

- There are clauses covering numerous platforms as well as tools particularly for Linux operating system-based implementations.

- Task-based Asynchronous Pattern (TAP) helps to enhance concurrency in asynchronous programming.

- Effective containerizing and lightweight Docker images drastically cut both deployment time and resource usage.

- Load balancing, auto-scaling, and integrated dependent injection are among the characteristics of cloud-native systems.

B) Java Spring Boot

Spring Boot is a strong framework built on top of the Spring framework substantially facilitates Java programming language in application development process. The availability of a whole spectrum of tools and functionalities significantly speeds up the development of production-ready applications.

Spring Boot is an open-source platform housed inside the Java environment, deliberately designed to enhance the development process of enterprise-level microservices, therefore facilitating improved usage. This framework enables rather remarkably effective creation and use of RESTful APIs by offering prescriptive defaults.

Key elements of Spring Boot are:

- A vast ecosystem painstakingly developed for commercial use including Spring Cloud, Hibernate, Kafka, and RabbitMQ integrated technologies.

- Embedded web servers as Tomcat, Jetty, and Undertow provide lightweight and self-sufficient deployments.

- Under Spring Security, OAuth can function in line with role-based access control systems by means of security measures.

- Modern computing environments are more relevant for the mix of cloud-native technologies with platforms including AWS, Azure, Google Cloud, and Kubernetes.

- One may run resilience and service discovery with Spring Cloud, circuit breakers, and API gateways used sensibly.

Although Spring Boot clearly benefits large-scale business applications needing a robust ecosystem and seamless integration into corporate environments, .NET Core is recognized for its remarkable performance and efficient resource management.

Research Objectives and Scope

This work integrates data from present literature, industry standards, and historical experimental results to undertake a comparative analysis of Java Spring Boot and .NET Core in the framework of cloud-native API development. Without undertaking new empirical evaluations, the goal is to evaluate how well these frameworks manage different cloud-based workloads. Based on past studies, this paper clearly assesses startup length, request latency, throughput performance, memory and CPU efficiency, containerization and deployment methods, cost-effectiveness, and security concerns. When choosing a framework for their cloud-native apps, software architects and developers must first grasp these traits to make educated, data-driven decisions.

Technical studies, peer-reviewed academic papers, and benchmark assessments comparing Spring Boot with .NET Core in cloud environments—more especially, AWS, Azure, and Google Cloud—are analyzed in this paper. Rather than new experimental discoveries, practical implementations, or direct data collecting, the study relies on secondary data sources. The research results will help businesses using APIs with .NET Core or Spring Boot in containerized or

serverless environments in grasp the trade-offs between performance, scalability, security, and cost effectiveness.

Background & Related Work

Overview of .NET Core and Java Spring Boot

A) .NET Core: Architecture & Cloud-Native Capabilities

.NET Core Framework is an open-source and cross-platform framework developed by the Microsoft for high performance cloud-native applications development .NET Core Framework is an open-source and cross-platform framework developed by the Microsoft for high performance cloud-native applications development. Originally the conventional .NET Framework, it developed with modularity, lightweight execution, Linux, Windows, and macOS. One of its most important benefits is the built-in support for microservices and RESTful API development, so cloud-native solutions choose it firstly. Originally the conventional .NET Framework, it developed with modularity, lightweight execution, Linux, Windows, and macOS. One of its most important benefits is the built-in support for microservices and RESTful API development, so cloud-native solutions choose it firstly.

Important characteristics of .Net Core for development of Cloud-Native APIs:

Lightweight Cross-Platform: Designed to function effectively in Linux and Windows platforms, unlike its predecessor, .NET Core is ideal for containerized installations.

High Performance API Processing: Using Kestrel, a really effective web server, high-performance API processing speeds demand management.

Asynchronous models of programming: Task-based Asynchronous Pattern (TAP) permits efficient non-blocking API execution in C#.

Cloud-native support with microservices: Built-in gRPC, dependence injection, and reduced runtime overhead help .NET Core to be microservices ideal.

Security: .NET Core reduces security threats by promoting JWT authentication, OAuth 2.0, and API gateways.

Integration of Kubernetes with Docker: Official .NET Core Docker images, created for performance, simplify deployment in containerized cloud-native environments.

Cloud Integration: .Net Core permits perfect interaction with primary cloud platforms, especially Microsoft Azure, which provides Azure Functions, App Services, and Kubernetes Service (AKS) fit for .NET-based APIs. AWS and Google Cloud additionally support .NET Core by Lambda functions, Kubernetes, and serverless deployment approaches.

B) Java Spring Boot: Architecture & Cloud-Native Capabilities

Spring Boot is an open-source Java framework intended for microservices oriented applications. Designed on Spring Framework, it provides dependency management, embedded web servers, and auto-configuration, so streamlining Java application deployment. Common in commercial applications, it offers microservices and cloud-native architectures much of tremendous support.

Features of Spring Boot for Cloud-Native API Development:

Enterprise-grade Ecosystem: Easy connections between Spring Boot and Spring Cloud allow cloud-native API gateway solutions, distributed tracing, and service discovery.

Microservices-oriented design: Provides built-in support for Kafka and RabbitMQ for containerized deployments, fault tolerance, and event-driven communication.

Embedded web servers: Comes with Tomcat, Jetty, and Undertow and lets you quickly deploy without using outside setups.

Declarative Security: Supported OAuth 2.0, JWT, and LDAP connectivity, uses Spring Security to enforce authentication and authorization policies.

Containerization: Well-optimized for Docker and Kubernetes, containerization \& Kubernetes support flawless cloud deployment.

Features of Serverlessism and Cloud-Nativeism: Ideal for serverless computing native integration with AWS Lambda, Azure Functions, and Google Cloud Functions is found.

Connection with Cloud Platforms: Highly compatible with AWS, Azure, and Google Cloud, Spring Boot is the recommended framework for Java applications born on the cloud. For serverless

execution, it connects with AWS Lambda; for containerized workloads, Kubernetes; and for service mesh designs like Istio.

Prior Research on API Performance

For cloud-native API development, several studies have contrasted the performance, scalability, and cloud efficiency of Spring Boot versus .Net Core. Mostly, these research have addressed startup time, API request handling, memory and CPU use, and scalability in containerized systems.

Existing Research and Performance Benchmarks:

- Examining microservices-based systems, Dinh-Tuan et al. (2020) found that while Spring Boot offered superior long-term maintainability and ecosystem support, .Net Core performed better under high-conurrency loads.

- Joshi & Kotha (2022) investigated the scalability of Java-based microservices in cloud-native environments, stressing that integration of Spring Boot with Spring Cloud helps large-scale distributed systems.

Industry Benchmarks and Practical Reviews

- Rajput (2018) and Bakliwal (2021) shed light on actual cloud installations, showing that whilst .Net Core is becoming more popular because of its high performance and lightweight execution, Spring Boot is more generally embraced in businesses because of its long-term reliability.

Comparison Methodology

The approach applied to compare Java Spring Boot with .Net Core for cloud-native API development is described in this part. This study lacks actual implementation or benchmarking since it is based on already published scholarly and commercial literature. Rather, the method depends on industry case studies, cloud provider documentation, a disciplined evaluation and synthesis of published results from peer-reviewed sources.

Comparison Metrics

The study assesses both systems on a set of commonly used standardized performance and development criteria extensively used in cloud-native API benchmarking, therefore guaranteeing a meaningful and unbiased comparison. Startup time, throughput and latency, memory and CPU use, developer productivity, security, containerizing and deployment efficiency, and cost economy are among these measures.

Under demand, performance is expressed in terms of API startup time, cold-start latency, especially pertinent on serverless platforms and request handling throughput. These are absolutely crucial to assess cloud deployment responsiveness. Often connected with load balancing and autoscaling features, scalability emphasizes on how well the frameworks manage growing concurrent users and requests.

Memory and CPU use expose how effectively the frameworks consume system resources, therefore directly affecting scalability cost and performance consistency. While security and maintainability investigate built-in authentication, authorization systems, and the update frequency of the frameworks, developer productivity is evaluated through simplicity of setup, configuration, and learning curve. At last, Kubernetes supports resource-based pricing in cloud environments, containerization and cost effectiveness center on Docker image sizes.

Cloud Environment Assumptions (AWS, Azure, Google Cloud)

A reasonable and consistent comparison of Spring Boot and .Net Core has to consider the cloud platforms where both APIs are usually used. Three main platforms—Amazon Web features (AWS), Microsoft Azure, and Google Cloud Platform (GCP)—each providing special features and optimization for the corresponding frameworks—are taken under consideration in this paper.

Widely supported on AWS, .NET Core provides cost-effective implementation and optimal cold-start speeds via Lambda, ECS, and EC2. Though it often suffers somewhat longer cold starts due to the JVM initialization, Spring Boot can also run in AWS Lambda (custom runtime) or container services like EKS and Fargate.

With sophisticated monitoring and performance tweaking choices, Microsoft Azure.NET Core gains from native support via Azure App Service, Azure Functions, and AKS (Azure Kubernetes Service). Meanwhile, Spring Boot is provided via Azure Spring Apps, a managed service geared for Spring tasks.

Both frameworks run powerfully on Google Cloud using Cloud Run, GKE (Google Kubernetes Engine), and Cloud Functions. While .NET Core can be implemented simply utilizing containerized

processes, Spring Boot typically fits well with GCP's support of Istio, Anthos, and microservices federation. These presumptions offer a benchmark for deployment that evaluates cost trade-offs among cloud providers as well as performance.

Theoretical Framework for Performance Comparison

This work is non-experimental, hence a theoretical comparison framework is applied to synthesis the current results into a coherent performance evaluation. This method gathers under common evaluative themes results from academic publications, industry whitepapers, and deployment case studies.

First, studies measuring equivalent metrics—such as latency, startup time, and memory use—across .NET Core and Spring Boot are found by means of a process called literary aggregation. These findings are triangulated from many sources to allow for variances in hardware, setup, or workload conditions. For instance, the conclusion of a measure based on three independent investigations showing that .Net Core has a shorter startup time than Spring Boot generates this consensus.

The results then are assessed in light of real-world deployment patterns, especially with relation to Kubernetes clusters, serverless environments, and CI/CD pipelines. To reflect pragmatic deployment considerations, the study considers cloud-specific configurations including auto-scaling thresholds, container image sizes, and payment structures.

This approach allows a fair, evidence-based evaluation of both technologies, therefore offering a complete, literature-based comparison to guide the choice of frameworks in cloud-native API development situations.

Performance Comparison

Building responsive, scalable, and reasonably priced APIs requires cloud-native frameworks acting as they should. Java Spring Boot is compared on six primary criteria— Startup Time, Throughput & Latency, Memory & CPU Utilization, Containerization & Deployment, Cost Efficiency, and Security & Maintainability with .Net Core. The basis of the research is the aggregated results from peer-reviewed sources and current benchmarks.

Startup Time

Startup time is especially important in serverless configurations and auto-scaling systems, where cold-start latency can seriously impair user experience and billing efficiency. .Net Core begins considerably faster than Java Spring Boot, according to several benchmark studies. Reduced runtime footprint of .Net Core and its more efficient compiled native execution architecture than Java's JVM-based bootstrapping produce this advantage.

Spring boot programs can ask for more time to start depending on JVM startup cost, classpath scanning, and dependency injection approaches. Cold-start scenarios like AWS Lambda or Azure Functions expose this latency. The discrepancy closes, nevertheless, in warm-start systems (like containerized APIs behind load balancers).

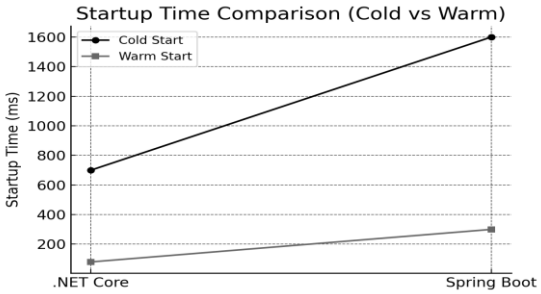

Figure 1. Startup time (ms) comparison between .NET Core and Spring Boot, illustrating cold-start and warm-start differences for cloud-native deployment scenarios.

Throughput & Latency

Scalability of APIs directly depends on throughput and latency under demand. Under some test conditions, .NET Core often shows less latency and better request throughput than Spring Boot. Great concurrency is provided by the Kestrel web server included in ASP.NET Core, built for performance and with asynchronous I/O methods.

Under such conditions, Spring Boot often shows significantly higher latency and poorer throughput, even if it is highly capable. Larger dependency trees, runtime abstraction layers, and the JVM trash collecting process can all cause somewhat performance issues. JVM tinkering and reactive programming models with Spring WebFlux, however more challenging, can help Spring Boot operate as best it can.

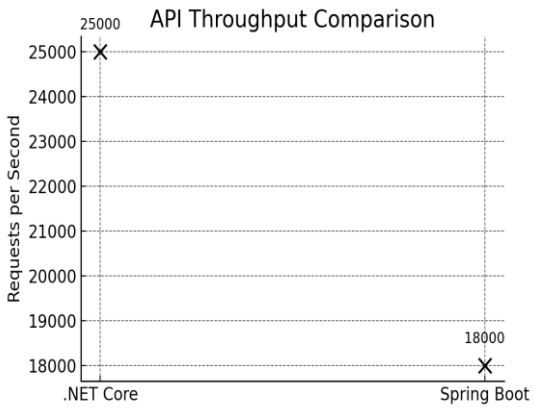

Figure 2. Average throughput (requests per second) of RESTful APIs implemented using .NET Core and Spring Boot under standard load.

Memory & CPU Utilization

Especially with pay-as-you-go models like AWS Fargate or Google Cloud Run, a major factor influencing cloud efficiency is resource utilization. Benchmarks reveal that .Net Core uses less CPU and memory than Spring Boot for managing like workloads. .Net Core apps utilize 30–50% less average memory, so running costs are reduced and scaling behavior is enhanced.

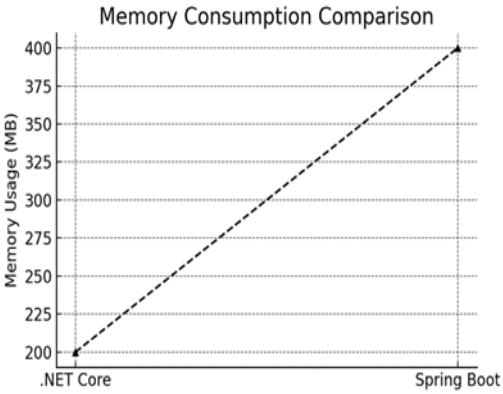

Figure 3. Memory usage (in MB) under load, highlighting the resource efficiency difference between the two frameworks.

Spring Boot apps exhibit more memory footprint due to the JVM and other runtime libraries. CPU consumption is also more in some circumstances under great demand concurrency and especially during application boot. When set up, nevertheless, Spring Boot offers robust JVM profiling and trash collecting management tools to help to minimize these issues.

Containerization & Deployment (Docker & Kubernetes)

Although both technologies allow modern containerized systems, their image size and orchestration efficiency vary. Typically starting at 20–50MB using optimal runtime images, .Net Core Docker images are smaller. From this follows faster container pull times, faster starting in Kubernetes clusters, and better portability in CI/CD pipelines.

Though with tools like jlink or native GraalVM images optimizing them, generally speaking, spring boot programs produce larger containers (100MB–300MB). Particularly when using Eureka, Consul, or service meshes like Istio, deployment using Spring Boot occasionally requires more configuration. On a corporate level, however, Spring's deep ties to Spring Cloud and Kubernetes ecosystems make it rather flexible and fit.

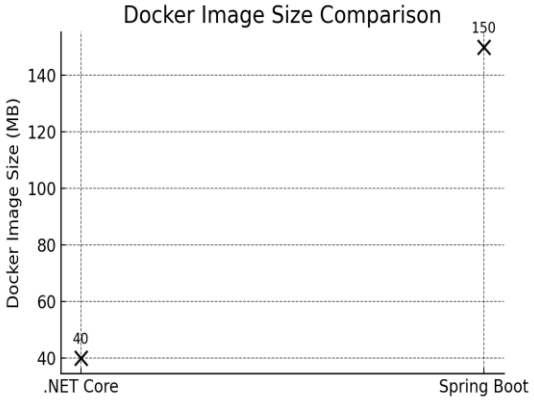

Figure 4. Container image size comparison showing deployment footprint of .NET Core and Spring Boot applications.

Cost Efficiency

Cost is an indirect but critical performance factor especially in cloud-native programs because billing is linked to execution time, memory use, and scaling behavior. Sometimes .net core is more cost-effective in lower memory utilization, faster startup, and smaller image size in serverless or containerized pay-per-use models.

Although spring boot apps have greater features, slower cold starts, larger container images, and higher memory utilization could generate more running costs. In corporate systems, the cost difference is less significant as cold starts are infrequent and uptime is continuous; nevertheless, if Spring's strong connectors reduce outside service needs, it could even reverse.

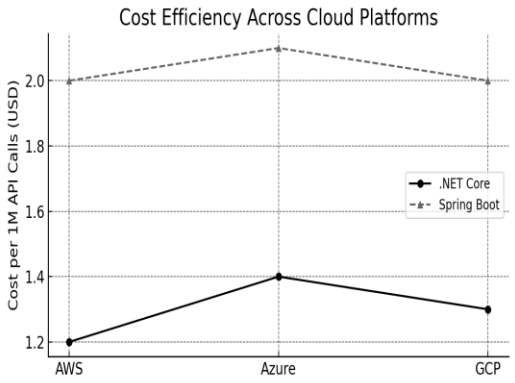

Figure 5. Estimated cost per one million API calls across AWS, Azure, and GCP for each framework, assuming equivalent workloads.

Security & Maintainability

While both models have created security ecosystems, their methods and tools vary. Directly into the ASP.NET Core pipeline, .NET Core aggregates identity providers, JWT authentication, and authorization middleware security components. Consolidated support through Microsoft helps with consistent patching and long-term support (LTS) cycles.

On the other hand, Spring Boot offers Spring Security—one of the most robust and flexible security solutions found inside the Java environment. It supports advanced multi-tenancy, RBAC, OAuth2, Open ID Connect, and CSRF protection. Its configuration can be more complex, though, and the community-based support model asks developers to keep current with security notes and releases.

In terms of maintainability, Spring Boot boasts extensive documentation, a larger developer community, and more third-party integrations. Visual Studio, Rider, and the .NET CLI all of which also offer robust tooling support are driving quick acceptance of .NET Core.

Table. Comparison of various metrics(.NET Core Vs Java SpringBoot)

METRIC	**.NET CORE**	**JAVA SPRING BOOT**
Startup Time	Fast (esp. cold starts)	Slower due to JVM startup
Throughput & Latency	Higher throughput, low latency	Moderate throughput, higher latency
Memory & CPU Usage	Low memory and CPU usage	Higher memory and CPU consumption
Docker Image Size	Small (20–50MB)	Larger (100–300MB)
Serverless/CI/CD Friendly	Highly optimized	Requires tuning
Security Features	Integrated, simpler setup	Rich but complex configuration
Maintainability & Ecosystem	Strong IDE/tooling, growing	Mature ecosystem, large community
Cost Efficiency	More efficient at scale	Costlier in pay-per-use models

Discussion & Analysis

While both architectures are suitable for cloud-native API development, a comparison of Java Spring Boot and .Net Core exposes variations in terms of design philosophy, runtime efficiency, developer experience, and cloud ready capability. This section explores the effects of these variances across three dimensions: trade-offs between the two technologies, best-fit scenarios for each, and industry acceptance patterns.

Trade-offs Between the Two Technologies

Choosing between .NET Core and Spring Boot means major trade-offs depending on performance criteria, project complexity, developer knowledge, and organizational preferences.

.Net Core is lighter, starts faster, consumes less memory and CPU for performance-critical apps hosted in serverless or resource-constrained environments. It also works really neatly with modern DevOps techniques and containerized processes. Though firms highly dedicated in non-Microsoft technologies could face a more steep integration curve, its ecosystem is less developed in terms of enterprise-grade libraries.

On the other hand, Java Spring Boot provides, from the start, a more complete feature set, particularly for systems of corporate grade. Its intimate link with Spring Cloud, config servers, and distributed

tracing tools gives architectural advantages for complex systems. Having said that, these features—which may impact performance and deployment speed in lightweight or microservice-oriented applications—come at the penalty of more configuration complexity, longer startup times, and higher resource usage.

Eventually the decision reveals a compromise between performance optimization (.NET Core) and ecosystem diversification (Spring Boot).

Best Use Cases for Each

Knowing when to apply each framework will help one to match technical decisions with architectural and corporate goals. Drawing on the results:

.NET CORE:
- Create lightweight APIs with high throughput demand using .NET Core.
- Working in models of serverless or containerized microservices.
- Maximizing cold-start performance and resource economy.
- Either working on Microsoft stack or aiming at Azure-native installations.

JAVA SPRINGBOOT:
- Create large-scale commercial apps with complex connectivity using Java Spring Boot.
- Strong skills in APIs, circuit breakers, and service discovery are needed.
- Either working with an experienced Java development team or with present Java infrastructure.
- Emphasizing hybrid-cloud or multi-cloud implementations using advanced orchestrating layers.

Generally speaking, Spring Boot is chosen in financial services, telecoms, and corporate backend systems where consistency, security, and vendor neutrality are very essential. .NET Core is increasingly evident in startups, product engineering, and cloud-optimized environments where performance, containerization, and cost control rank highest.

Industry Adoption Trends

Industry trends show increasing acceptance of both approaches; the decision usually comes from team experience, corporate history, and cloud provider alignment.

Spring Boot still governs enterprise Java systems especially in firms with past Spring investments. It is supported by a sizable open-source community, extensive documentation, and consistent developments in cloud integration like Spring Cloud, Sleuth, and Resilience4J. Still the preferred paradigm in banking, healthcare, and telecommunications is this one.

On-demand .NET Core has grown very popular, on the other hand, thanks to its open-source attitude, cross-platform portability, and modern architecture. Companies already running Microsoft Azure find it especially tempting; e-commerce, education, and real-time systems are just beginning to gather momentum. Growing support from Linux environments and robust tools like Visual Studio Code and GitHub Actions helps it to become a major participant in cloud-first development.

Both approaches are rapidly evolving to meet serverless, containerized, and cloud-native computing even if they differ. Companies are applying polyglot designs, leveraging the benefits of both technologies as appropriate.

Conclusion

This paper presents Java Spring Boot in a relative perspective. .Net Core for cloud-native API development grounded only on contemporary literature and benchmark results. Especially for lightweight, serverless, and high-performance programs, .Net Core offers better startup speed, memory economy, and cost-effectiveness when compared to substitutes. Conversely, Spring Boot extensive ecosystem, ability for corporate integration, and support of cloud-native technologies help it to be most appropriate for long-term projects and large-scale distributed systems. Which of the two cloud-compatible models best fits a specific scenario will depend on particular project requirements, current infrastructure, and organizational objectives. This literature-based study provides builders and designers with a framework for matching performance goals and cloud deployment methodologies with chosen framework.

References

[1] Dinh-Tuan, H., Mora-Martinez, M., & Beierle, F. (2020). Development Frameworks for Microservice-Based Applications: Evaluation and Comparison. International Conference on Cloud Computing and Services Science.

[2] Joshi, P.K., & Kotha, R. (2022). Architecting Resilient Online Transaction Platforms with Java in a Cloud-Native World. ResearchGate.

[3] Rajput, D. (2018). Mastering Spring Boot 2.0. Packt Publishing.

[4] Marchioni, F. (2019). Hands-on Cloud-Native Applications with Java and Quarkus.

[5] Gutierrez, Felipe. (2019). Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices. 10.1007/978-1-4842-3676-5.

[6] Vitale, T. (2022). Cloud Native Spring in Action: With Spring Boot and Kubernetes. Manning Publications.

[7] Sangapu, S. S., Panyam, D., & Marston, J. (2022). The Definitive Guide to Modernizing Applications on Google Cloud. O'Reilly Media.

[8] Dhalla, Hardeep Kaur. "A performance comparison of restful applications implemented in spring boot java and ms. net core." Journal of Physics: Conference Series. Vol. 1933. No. 1. IOP Publishing, 2021.

[9] Zentner, Andrej, and Robert Kudelic. "Multithreading in. Net and Java: A Reality Check." J. Comput. 13.4 (2018): 426-441.

[10] Arif, M. A., Hossain, M. S., Nahar, N., & Khatun, M. D. (2014). An Empirical Analysis of C#, PHP, JAVA, JSP and ASP. Net regarding performance analysis based on CPU utilization. Banglavision Research Journal, 14(1), 173-187.

[11] Tillotson, R. Web Applications With Java Server Pages and Microsoft .NET Web Forms.

[12] Paguay-Soxo, P., & Vivanco, M. (2018). Comparative Analysis of File Transfer Performance Between Java and. NET Using a Hybrid Encryption Protocol with AES and RSA. KnE Engineering, 161-177.

[13] Kronis, K., & Uhanova, M. (2018). Performance Comparison of Java EE and ASP. NET Core Technologies for Web API Development. Appl. Comput. Syst., 23(1), 37-44.

[14] Munonye, K., & Martinek, P. (2018, September). Performance analysis of the microsoft. Net-and java-based implementation of REST web services. In 2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY) (pp. 000191-000196). IEEE.

[15] Selakovic, M., & Pradel, M. (2016, May). Performance issues and optimizations in javascript: an empirical study. In Proceedings of the 38th International Conference on Software Engineering (pp. 61-72).

[16] Soni, A., & Ranga, V. (2019). API features individualizing of web services: REST and SOAP. International Journal of Innovative Technology and Exploring Engineering, 8(9), 664-671.

[17] Roy, A. C., Al Mamun, M. A., Khairat Hossin, M. A. I., Uddin, M. P., Afjal, M. I., & Sohrawordi, M. (2017). Developing Operating System Simulation Software for Windows Based System by C# .NET Framework and an Android Application by JAVA and XML. Journal of Operating Systems Development & Trends, 4(1), 9-18.

[18] Goldshtein, S., Zurbalev, D., & Flatow, I. (2012). Pro .NET Performance: Optimize Your C# Applications.

[19] Bayya, Anil Kumar. (2023). Building Robust Fintech Reporting Systems Using JPA with Embedded SQL for Real-Time Data Accuracy and Consistency. The Eastasouth Journal of Information System and Computer Science. 1. 119-131. 10.58812/esiscs.v1i01.480.

[20] Rozaliuk, T., Kopyl, P., & Smołka, J. (2022). Comparison of ASP.NET Core and Spring Boot ecosystems. Journal of Computer Sciences Institute, 22, 40–45. https://doi.org/10.35784/jcsi.2794.

[21] Mohan, J. S. S., & Goswami, K. (2023). Performance Analysis and Comparison of Node.Js and Java Spring Boot in Implementation of Restful Applications. https://doi.org/10.22541/au.169655403.34118093/v1

[22] Kafri, N., & Hamed, O. (2009). Performance Prediction of Web Based Application Architectures Case Study: .NET vs. Java EE. 1, 146–156. http://www.dirf.org/ijwa/v1n30609.pdf

[23] Munonye, K., & Martinek, P. (2018). Performance Analysis of the Microsoft. Net- and Java-Based Implementation of REST Web Services. International Symposium on Intelligent Systems and Informatics, 191–196. https://doi.org/10.1109/SISY.2018.8524705