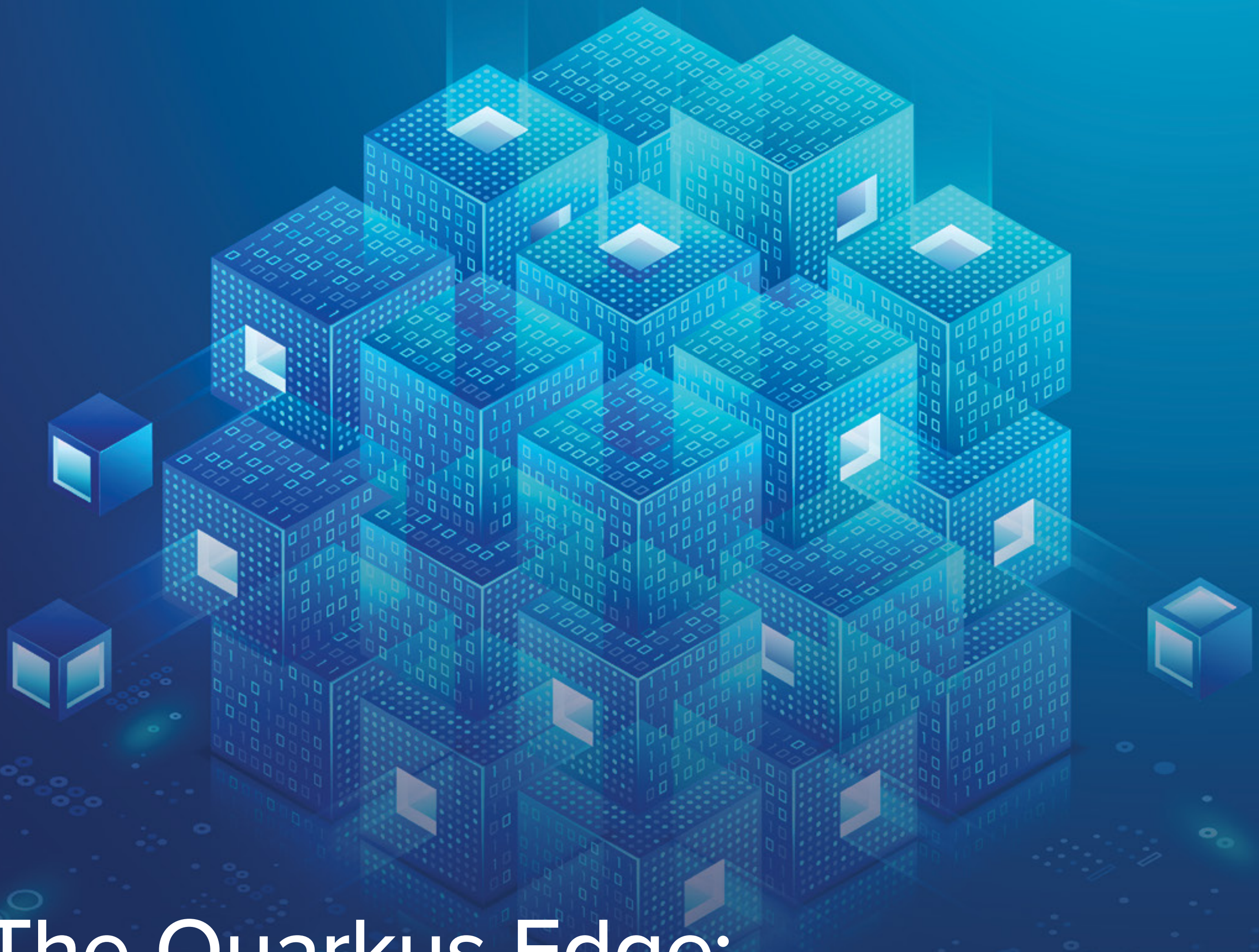
White Paper, sponsored by Red Hat | January 2025



The Quarkus Edge: How Real Customers Achieve Speed, Performance, and Agility

Arnal Dayaratna
Research Vice President,
Software Development, IDC

Table of Contents

CLICK ANY HEADING TO NAVIGATE DIRECTLY TO THAT PAGE.

Situation Overview

Quarkus is a modern Java framework for cloud-native development that prioritizes development-related agility, automation, scalability, and resilience as inherent characteristics rather than optional enhancements. As such, Quarkus prioritizes the ability to build, extend, and refine applications optimized for containerized environments, microservices, and serverless architectures. The shift toward cloud-native development, which Quarkus enables, ensures that applications can leverage faster start-up times, lower memory consumption, and seamless integration with cloud ecosystems as fundamental attributes.

As enterprises increasingly adopt cloud-native development practices to drive digital transformation, the need for frameworks that support agile, resilient, and scalable applications has intensified. Quarkus has emerged as a critical enabler in this landscape by delivering a tailored Kubernetes-native experience for developers and IT decision-makers. By integrating directly into existing development pipelines, Quarkus helps teams build applications that can respond rapidly to changing business requirements, optimize resource utilization, and enhance end-user experiences. This white paper examines the business and development-related benefits of Quarkus by synthesizing findings from interviews with IT leaders who adopted Quarkus for their organization. As such, the white paper elaborates on findings from three case studies in conjunction with generalized commentary on motivations for its adoption, the decision-making process for its selection, and its benefits.

Definitions

Cloud-Native Development

Cloud-native development refers to a development paradigm that brings the agility, automation, scalability, and resilience of cloud computing to applications and software development.

The paradigm is the conjunction of the following attributes:

Microservices Architecture:
Cloud-native development structures applications into independent, modular services known as microservices that companies can develop, deploy, and scale independently. This architectural style increases flexibility and enables rapid updates and specific scaling of services without impacting the entire application.

Containerization:
Technologies such as Docker package microservices into containers that ensure consistency across development, testing, and production environments. Containerization promotes application isolation and portability by making it easier to deploy services across various cloud environments.

Container Orchestration Frameworks:
Technologies such as Kubernetes automate the deployment, scaling, and management of containerized applications. These frameworks optimize resource use, manage load balancing, and enhance the resilience of cloud-native systems.

Development Operations and Continuous Integration/ Continuous Development Pipelines:
Cloud-native development integrates DevOps practices and continuous integration/continuous deployment (CI/CD) pipelines to streamline the build, test, and deployment processes. This approach reduces the time it takes to deliver new features and ensures reliable, consistent application delivery.

Quarkus

Quarkus is a Kubernetes-native Java framework for building cloud-native applications.

Optimized for GraalVM and HotSpot, it enables fast start-up times, low memory consumption, and a developer-friendly experience. Quarkus supports seamless integration with Kubernetes and other cloud ecosystems, making it a strong choice for enterprises seeking scalable, efficient solutions. These capabilities make Quarkus particularly beneficial in AI and generative AI contexts, where performance, scalability, and responsiveness are critical.

CASE STUDY#1:

Use of Quarkus in Telco for the Edge and the Internet of Things

A software engineer from a large U.S. telecommunications company discussed its adoption of Quarkus to meet stringent performance requirements resulting from the shift toward 5G and edge computing. After initially exploring Quarkus in 2018 as an alternative to Spring, the company decided to adopt Quarkus for production workloads in 2020. The need to significantly improve performance and latency influenced the decision. Extensive benchmarking showed that Quarkus significantly outperformed other frameworks, such as Spring WebFlux and Vert.x, particularly in CPU efficiency and latency. The company successfully integrated Quarkus into its development stack by leveraging its capabilities for various use cases, including IoT and data processing.

Achieved up to 10x cost reduction in CPU performance,

Significantly improved P99 latency

Benefits of Using Quarkus:

- **Performance Improvement:** Achieved up to 10x cost reduction in CPU performance, crucial for high-throughput and low-latency requirements
- **Latency Reduction:** Significantly improved P99 latency, which is essential for real-time applications
- **Cost Efficiency:** Reduced infrastructure costs, especially in CPU usage, which is nine times more expensive than random-access memory in public clouds
- **Comprehensive Framework:** Provided a complete solution that integrated well with existing technologies, such as Kafka and RabbitMQ
- **Developer Productivity:** Enhanced the developer experience with features such as live reload and fast start-up times
- **Scalability:** Facilitated the development of scalable, edge-computing solutions, supporting the company's strategic initiatives
- **Flexibility:** Allowed the use of Vert.x directly within Quarkus and correspondingly offered a hybrid approach for various use cases
- **Operational Efficiency:** Simplified integration with multiple protocols and reduced the complexity of managing dependencies

CASE STUDY#2:

Adoption of Quarkus for Event-Driven Architectures in Transportation Services Such as Taxis and Airlines

A software engineer working at a global distribution system company discussed the company's adoption of Quarkus to leverage event-driven architecture (EDA). The company chose Quarkus for its reduced memory and CPU usage and improved developer experience. The transition began with new applications and gradually included existing ones. By 2022, the company was using Quarkus in production, significantly improving application performance and developer productivity. The company plans to expand its use of Quarkus and potentially replace the legacy applications entirely.

Significant reduction in memory and CPU usage

Reduced operational costs due to lower resource consumption

Benefits of Using Quarkus:

- **Faster Start-Up Times:** Reduced from 5–10 minutes to a few seconds
- **Lower Resource Consumption:** Significant reduction in memory and CPU usage
- **Improved Developer Experience:** Enhanced feedback loop and ability to run applications on laptops
- **Cost Savings:** Reduced operational costs due to lower resource consumption
- **Event-Driven Architecture:** Integrated EDA capabilities with Kafka and reactive messaging.
- **Scalability:** Facilitates cloud-native development and microservices architecture.
- **Support and Integration:** Strong support from Red Hat and easy integration with existing systems, such as OpenShift and Splunk

CASE STUDY#3:

Adoption of Quarkus to Modernize Banking Services

An enterprise architect at a leading European bank discussed its transition to using Quarkus for its core banking system. The bank divided its IT structure into 16 functional domains, each of which is responsible for specific banking services. These domains expose services through a common platform that the bank has implemented Quarkus into. They began migrating from the legacy application to Quarkus in early 2023 due to support discontinuation. Quarkus' performance benefits and strong support from Red Hat influenced the decision. The migration process included reskilling developers and addressing integration challenges with legacy systems.

Allows applications to start up 85% faster

Allows the platform to host 30% more applications

Boosts developer productivity

Benefits of Using Quarkus:

- **Faster Start-Up Times:** Allows applications to start up 85% faster, improving deployment and recovery times
- **Reduced Memory Consumption:** Saves costs by using less memory than JBoss Fuse
- **Increased Hosting Capacity:** Allows the platform to host 30% more applications within the same infrastructure
- **Enhanced Developer Experience:** Boosts developer productivity with live coding capabilities and rapid start-up times
- **Scalability:** Eases migration to public cloud infrastructure, aligning with the bank's future IT strategy
- **Operational Efficiency:** Reduces operational overhead and improves system resilience
- **Cost Savings:** Avoids additional licensing costs by hosting more applications on existing infrastructure

Motivations for Adopting Quarkus

Enterprises adopt Quarkus to address core challenges associated with cloud-native development, such as optimizing performance optimization, reducing infrastructure overhead, and ensuring a modern developer experience.

For example, a U.S. telecommunications company achieved a 10x reduction in CPU costs and significantly improved latency. The reduction in costs and improvement in latency illustrate how Quarkus meets stringent performance requirements for edge and IoT use cases. Similarly, a transportation services provider reduced start-up times from minutes to seconds, optimizing resource consumption for EDAs.

These examples underscore the quantifiable benefits organizations achieve by adopting Quarkus, including enhanced scalability, faster time-to-market, and cost efficiency. By offering rapid start-up times, lower latency, and increased throughput, Quarkus effectively meets the demands of modern microservices and serverless architectures. Additionally, its compatibility with Kubernetes-native environments simplifies the deployment and scaling of applications.

Modernizing Traditional Frameworks

Many enterprises transition from legacy Java frameworks because they struggle to meet modern scalability demands due to higher memory footprints and slower start-up times. Quarkus provides a lightweight, efficient alternative that integrates seamlessly with container orchestration platforms, improves overall application performance, and reduces infrastructure costs. This makes it a preferred choice for organizations looking to modernize their technology stacks while minimizing migration risks.

Decision-Making Process

Enterprises evaluating Quarkus conduct parallel case studies to compare its performance with other frameworks by examining metrics such as start-up times, memory consumption, and throughput. For example, a leading European bank observed an 85% reduction in application start-up times during its evaluation phase, which directly influenced its decision to adopt Quarkus for modernizing its core banking services. Additionally, a transportation services company leveraged Quarkus' rapid feedback loops and lower resource consumption to support EDAs.

These case studies demonstrate how Quarkus provides measurable advantages that inform decision-making processes, enabling organizations to make data-driven choices that align with strategic objectives. As the case studies section of this white paper illustrates, cost–benefit analyses reveal how Quarkus can reduce infrastructure costs, improve developer productivity, and increase application responsiveness. Key decision criteria include enhanced performance, cost efficiency, developer experience, and enterprise-level support. Organizations often consider existing relationships with Red Hat, as Quarkus integrates well within the Red Hat ecosystem, which is already a trusted ecosystem for enterprise-grade support and long-term stability.

As the case studies section of this white paper illustrates,
cost–benefit analyses reveal how Quarkus can reduce infrastructure costs, improve developer productivity, and increase application responsiveness.

Business Benefits of Adopting Quarkus

Cost Reduction

Quarkus enables enterprises to significantly lower infrastructure costs through its resource-efficient design. It minimizes memory consumption and CPU usage by making applications lighter and more cost effective across cloud and on-premises environments. For example, Quarkus' native compilation with GraalVM can reduce memory usage, reducing container sizes and cloud runtime costs. This optimization is particularly advantageous in microservices or serverless contexts, where resource consumption directly affects pricing models. Additionally, Quarkus' faster start-up times reduce costs by decreasing the active runtime in serverless deployments, which directly impacts cost metrics such as AWS Lambda's per-invocation pricing.

Faster Time-to-Market

In the fast-paced landscape of digital transformation, accelerating application delivery is a key differentiator. Quarkus enhances developer productivity by supporting features such as live coding, which enables real-time testing and rapid iteration. The live-reloading feature eliminates the need to restart the application for every code change, enabling developers to see updates instantly and adjust on the fly. This capability extends to CI/CD pipelines, where Quarkus streamlines automated testing and deployment processes, reducing cycle times from days to hours. By integrating smoothly with tools such as GitLab CI/CD, Jenkins, or GitHub Actions, Quarkus allows for seamless parallel deployment that reduces development-related friction and enables faster delivery of new features.

Enhanced Application Performance

Quarkus can handle high-performance requirements in enterprise environments. By optimizing for both JVM and GraalVM, it ensures rapid start-up, low latency, and high throughput. For applications with real-time needs — such as financial trading platforms, online gaming, or AI/ML workloads — Quarkus' architecture enables faster data processing and quicker response times. This capability is especially critical for applications that must process large volumes of transactions simultaneously. The non-blocking I/O and reactive programming model in Quarkus facilitate asynchronous data handling by improving responsiveness and allowing applications to scale horizontally with minimal performance degradation. Additionally, for businesses operating in regulated industries, Quarkus helps maintain compliance by supporting transaction consistency and secure data handling.

Legacy Modernization

Legacy modernization with Quarkus allows enterprises to bring existing Java applications into the cloud-native era with minimal disruption. The framework offers compatibility with existing Java libraries, frameworks, and application programming interfaces (APIs), making it easier for developers to migrate legacy Java Enterprise Edition or Spring Boot applications to modern microservices or serverless models. By leveraging Quarkus' extension-based architecture, enterprises can gradually transition legacy systems into a containerized ecosystem that improves performance without extensive rewrites. For example, Quarkus' support for Hibernate ORM facilitates seamless database migrations while maintaining data integrity. This modernization strategy reduces the technical debt associated with legacy systems and allows enterprises to extend the lifespan of their digital assets while optimizing them for cloud-native scalability.

Support for Cloud-Native Development

Quarkus' architecture aligns with core cloud-native principles that enable scalability, resilience, and continuous delivery. By supporting Kubernetes-native features, such as service discovery, load balancing, and declarative configuration, Quarkus enables seamless integration with container orchestration technologies. The built-in fault tolerance and circuit breaker patterns enhance application reliability and enable systems to expediently recover from failures without affecting the user experience. Quarkus supports function-based architectures that enable serverless deployment on platforms such as AWS Lambda, Google Cloud Functions, or Azure Functions. This adaptability to cloud infrastructure ensures applications remain resilient, scalable, and cost effective across various deployment scenarios.

Legacy modernization with Quarkus allows enterprises to bring existing Java applications into the cloud-native era with minimal disruption.

Development-Related Benefits of Quarkus

Faster Start-Up Time

Quarkus achieves rapid start-up times by leveraging GraalVM's native image compilation and innovative build-time metadata processing. By moving as much processing as possible to build time, Quarkus reduces runtime overhead and enables applications to start in milliseconds rather than seconds. This reduction of application runtime is particularly effective for serverless deployments and microservices architectures that require applications to scale up and down quickly in response to demand. For developers, faster start-up times mean quicker development cycles and more responsive applications, enabling them to test and deploy code changes more rapidly. In latency-sensitive environments, this improves user experiences and meets stringent performance requirements.

Lower Latency

Quarkus' reactive programming model, built on frameworks such as Vert.x and Mutiny, lowers latency by supporting asynchronous, non-blocking operations. This allows applications to handle more simultaneous requests with minimal delay, improving the user experience in latency-sensitive applications, such as streaming analytics, chat services, and real-time data processing. Developers benefit from a simplified approach to handling concurrency and backpressure because Quarkus allows them to build applications that are responsive and resilient under high-load conditions. The ability to process data streams efficiently is crucial for applications in AI/ML, IoT, and financial services, where timely data handling can significantly impact outcomes.

Increased Throughput

High throughput is a core attribute of Quarkus that it realizes by supporting event-driven, non-blocking architectures and efficient resource utilization. By leveraging reactive programming principles and efficient execution models, Quarkus can handle a high volume of transactions without compromising performance. For developers building data-intensive applications — such as IoT platforms, online marketplaces, or real-time analytics dashboards — Quarkus' architecture ensures that applications can handle concurrent operations smoothly. The framework's integration with technologies such as Apache Kafka and AMQP enables efficient message processing and data streaming at scale, allowing developers to build systems that can ingest and process vast amounts of data in real time.

Developer Productivity

Quarkus enhances developer productivity through features that streamline coding, testing, and deployment. The live coding capability provides instant feedback that allows developers to test new functionalities in real time without restarting the application. This accelerates the development process by reducing context switching and debugging time. Quarkus supports a unified configuration model and leverages familiar programming paradigms that make it easier for developers to manage different environments (e.g., development, staging, and production) without manual configuration adjustments. The framework's emphasis on convention over configuration reduces boilerplate code and allows developers to focus on business logic rather than infrastructure code. These features collectively enhance the developer experience by facilitating faster iteration cycles and higher-quality code.

Integration with Developer Tools

Quarkus integrates with an extensive ecosystem of developer tools and supports everything from integrated development environments to CI/CD pipelines and monitoring solutions. For instance, Quarkus' extensions for IntelliJ IDEA and Visual Studio Code include features such as code completion, syntax highlighting, and built-in debugging. The framework's support for Maven and Gradle allows developers to leverage familiar build automation tools that facilitate tasks such as dependency management, code compilation, and packaging. Quarkus also integrates with testing frameworks that enable developers to write unit and integration tests effectively. Its observability features, compatible with Prometheus, Grafana, and Jaeger, offer real-time insights into application performance, allowing developers to quickly identify bottlenecks and optimize performance. This integration reduces friction across the development life cycle and aligns well with agile and DevOps practices.

Support for Cloud-Native Development

Quarkus is designed with cloud-native development in mind, emphasizing microservices architecture, containerization, and serverless deployment. Its extension-based architecture supports critical cloud-native features such as service resilience, declarative configuration, and service mesh integration (e.g., Istio or Linkerd). For example, Quarkus simplifies building microservices that integrate seamlessly with service meshes, enhancing observability, security, and load balancing. The framework's support for OpenAPI and MicroProfile makes it straightforward to develop representational state transfer (REST) services that comply with industry standards. Quarkus also supports reactive streams and asynchronous messaging through integration with Reactive Messaging and SmallRye Reactive Messaging, which are essential for creating event-driven applications. This comprehensive support for cloud-native development allows developers to build robust, scalable, and adaptable applications that align with the demands of distributed systems.

Developer-Friendly Ecosystem

Quarkus' extension-based architecture enables developers to quickly add functionality and enhance application capabilities while maintaining a familiar development experience. The ecosystem supports popular APIs and libraries, such as RESTEasy for RESTful web services, Hibernate ORM for database interactions, Apache Kafka for messaging, and Vert.x for reactive applications, simplifying the integration of features without steep learning curves. Quarkus offers a rich set of community-driven extensions, which facilitate faster adoption of emerging technologies and frameworks that keep developers at the forefront of cloud-native innovations. The framework's intuitive design, comprehensive documentation, and active community support reduce the learning curve, enabling developers to onboard quickly and accelerate delivery cycles.

Moreover, Quarkus encourages best practices through its opinionated approach, guiding developers toward efficient coding patterns and architectural decisions. This helps maintain code quality and consistency across teams, which is particularly valuable in large enterprise environments with multiple development teams.

Quarkus offers a rich set of community-driven extensions, which facilitate faster adoption of emerging technologies and frameworks that keep developers at the forefront of cloud-native innovations.

Considerations for Quarkus Adoption

Adopting Quarkus within an organization requires a strategic approach that accounts for technical and organizational factors. Successful adoption involves careful evaluation, seamless integration, and a clear understanding of best practices to maximize value.

Evaluation Phase

Organizations should begin with proof-of-concept projects to evaluate Quarkus' compatibility, performance, and scalability within their specific environments. During this phase, it is crucial to identify target use cases, such as modernizing legacy applications, building new microservices, or deploying serverless solutions. Assessing existing infrastructure and application dependencies will help determine how easily organizations can integrate Quarkus and whether they require additional tooling or refactoring. Benchmarking Quarkus against existing frameworks in terms of performance metrics, such as start-up time, memory usage, and throughput, provides tangible data to support decision-making.

Integration with Existing Infrastructure

Quarkus can integrate with existing DevOps tools, CI/CD systems, databases, and cloud-native tools. Before full-scale adoption, organizations should ensure compatibility with current workflows that include build automation, deployment pipelines, and monitoring setups. Adopting Quarkus incrementally — starting with less-critical applications — can help organizations optimize resource utilization and adapt to the new development environment. This step-by-step integration allows teams to adjust configurations, test extensions, and train developers gradually, minimizing potential disruptions. Leveraging containerization tools, such as Docker, and orchestration platforms, such as Kubernetes, facilitates the deployment of Quarkus applications alongside existing services.

Training and Skill Development

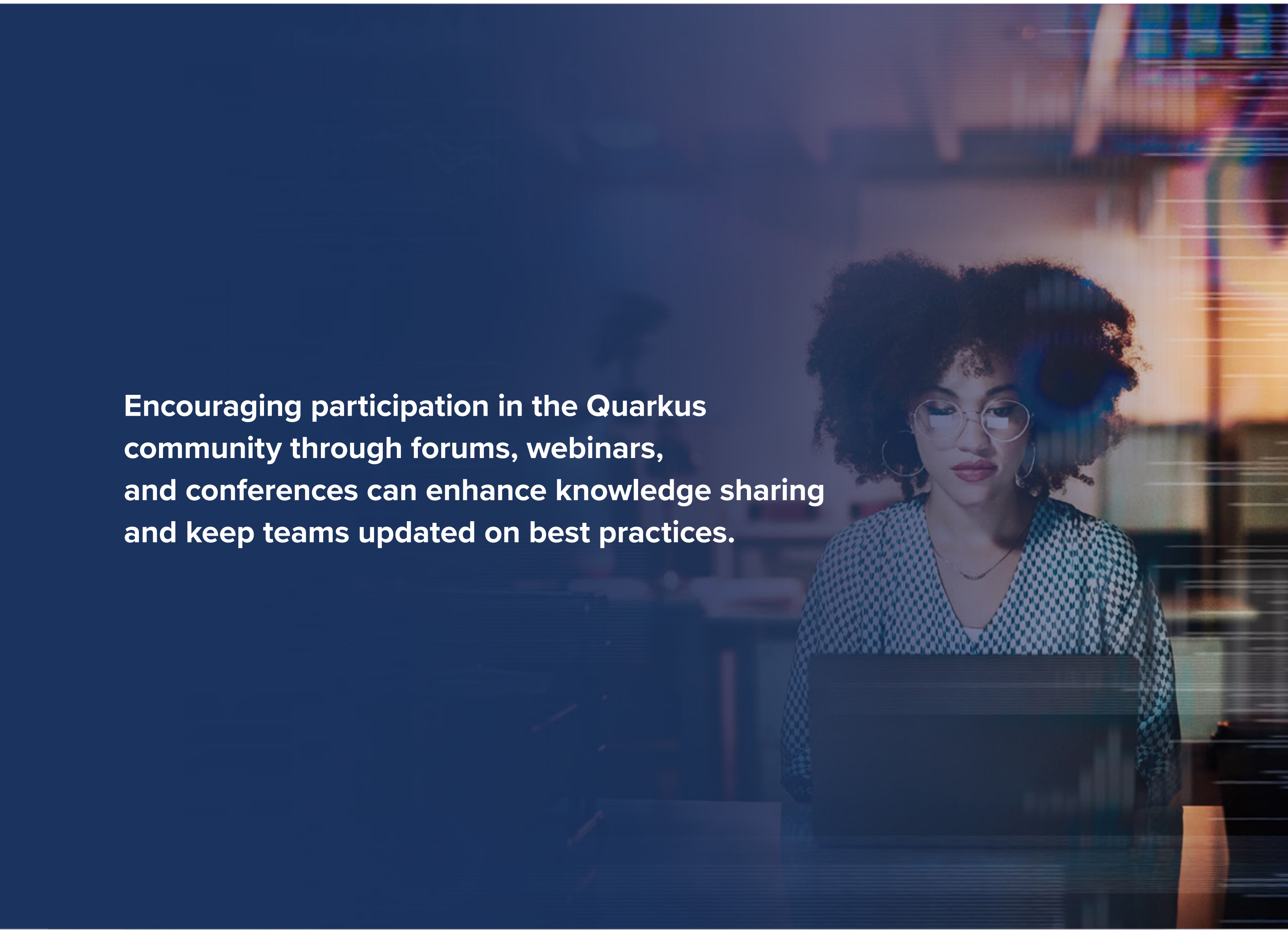
Effective adoption of Quarkus requires upskilling developers in areas such as reactive programming, cloud-native patterns, and Quarkus-specific extensions. Organizations should prioritize workshops, hands-on training, and developer guides to familiarize teams with Quarkus' features and functionalities. Encouraging participation in the Quarkus community through forums, webinars, and conferences can enhance knowledge sharing and keep teams updated on best practices. A strong focus on documentation and collaborative learning will enable developers to leverage Quarkus' capabilities more effectively and ensure its readiness for production environments.

Monitoring and Performance Optimization

As organizations deploy Quarkus' applications, it is essential to establish robust monitoring and observability practices. Companies can use tools such as Prometheus, Grafana, and Jaeger to track performance metrics, identify bottlenecks, and ensure applications meet service-level agreements. Additionally, organizations should conduct performance testing to validate Quarkus' benefits in terms of start-up time, latency, and throughput in production scenarios. Optimizing configurations and infrastructure settings based on monitoring data will help ensure optimal performance as applications scale. Developers should implement logging and tracing mechanisms to facilitate debugging and maintain application health.

Managing Change and Adoption

Adopting Quarkus often requires a cultural shift within development teams, particularly for those transitioning from traditional Java frameworks. Organizations should foster a culture of experimentation, encouraging developers to explore Quarkus' features and capabilities. IT leaders clearly communicating the strategic benefits of adopting Quarkus — such as improved performance, scalability, and cost efficiency — can help align teams around shared goals, facilitating smoother adoption and long-term success. Establishing champions or lead developers who can guide others and serve as internal experts can accelerate the adoption process.



Encouraging participation in the Quarkus community through forums, webinars, and conferences can enhance knowledge sharing and keep teams updated on best practices.

Conclusion

Quarkus represents a strategic choice for enterprises seeking to accelerate cloud-native development, enhance application performance, and reduce the total cost of ownership. By prioritizing efficiency, scalability, and developer productivity as default attributes, Quarkus empowers developers and IT decision-makers to drive digital transformations with confidence. Its ability to optimize Java for the cloud-native era positions it as a forward-thinking framework that aligns with modern development practices.

As enterprises continue to navigate the complexities of cloud-native adoption, Quarkus stands out as an effective framework for building the next generation of Java applications. Its combination of performance optimizations, developer-friendly features, and seamless integration with cloud ecosystems makes it a compelling option for organizations looking to modernize their applications and infrastructure. By embracing Quarkus, enterprises can achieve faster time-to-market, improved application performance, and greater operational efficiency, positioning themselves competitively in an increasingly digital landscape.

About the IDC Analyst

Arnal Dayaratna
Research Vice President, Software Development, IDC

Dr. Arnal Dayaratna is research vice president of Software Development at IDC. Arnal focuses on software developer demographics, trends in programming languages and other application development tools, and the intersection of these development environments and the many emerging technologies that are enabling and driving digital transformation. Arnal's research examines how the changing nature of software development relates to broader trends in the technology landscape.

More about Arnal Dayaratna

Message from the Sponsor

Red Hat

The Red Hat build of Quarkus is an open, Kubernetes-native Java framework that empowers organizations to modernize their Java workloads or build new cloud-native applications with ease.

As the largest contributor to Quarkus, Red Hat has shaped the framework to be developer-centric, efficient, and highly scalable for cloud environments. With open standards at its core, Quarkus embraces the Java ecosystem and seamlessly integrates with popular libraries and frameworks, making it easier for developers to adopt while preserving existing investments.

For organizations, this open, community-driven approach means leveraging a flexible, innovation-friendly framework, backed by Red Hat's expertise and enterprise support. Combined with OpenShift, Quarkus offers a powerful solution for accelerating digital transformation, fostering agility, and delivering high-performing Java applications across hybrid and cloud-native environments.

For more information, check out "4 reasons to try Quarkus for Kubernetes-native Java apps".

IDC Custom Solutions

IDC Research, Inc.
140 Kendrick Street, Building B, Needham, MA 02494, USA
T +1 508 872 8200

idc.com

@idc

@idc