Rebecca Dias 2019, 7:2
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology

An Open Access Journal

# From Monolith to Microservices a Jboss and Tomcat Migration Guide for Cloud Computing

**Rebecca Dias**
Goa University

**Abstract- Enterprises running monolithic Java applications on JBoss and Tomcat face increasing challenges in scalability, maintainability, and integration with cloud-native technologies. Microservices architecture offers a modular, flexible alternative that enables independent deployment, improved fault isolation, and operational agility. This review examines strategies for migrating monolithic JBoss and Tomcat applications to microservices, covering code refactoring, containerization, middleware integration, and orchestration in hybrid and cloud environments. It highlights automation through CI/CD pipelines, security and compliance considerations, and monitoring for performance optimization. Case studies from both large enterprises and mid-sized organizations demonstrate practical applications, lessons learned, and business impacts, including operational efficiency, cost optimization, and accelerated innovation. Emerging trends, such as serverless computing, AI-driven DevOps, and evolving middleware ecosystems, are discussed to provide guidance for organizations seeking sustainable and scalable modernization. This article serves as a comprehensive roadmap for IT architects, developers, and enterprise decision-makers pursuing microservices migration and cloud readiness.**

**Keywords- Monolithic Applications, Microservices Architecture, JBoss, Tomcat, Cloud Computing, Containerization, CI/CD Automation, Middleware Integration, Security and Compliance, IT Modernization.**

## I. INTRODUCTION

### Background on Monolithic Architectures

Monolithic applications have historically dominated enterprise IT environments, particularly for Java-based workloads running on application servers such as JBoss and Tomcat. In monolithic architectures, all components of an application—presentation, business logic, and data access layers—are tightly coupled and deployed as a single unit. While this design simplifies initial development and deployment, it introduces significant challenges over time, particularly as applications grow in complexity. Scaling monoliths often requires scaling the entire application, leading to inefficient resource usage and increased infrastructure costs. Additionally, tightly coupled components can slow down deployment cycles, impede fault isolation, and create bottlenecks for teams attempting to implement agile practices. These challenges have prompted enterprises to explore more modular, flexible alternatives that can accommodate rapid innovation and cloud adoption.

### Rise of Microservices in Cloud Computing

Microservices architecture has emerged as a preferred solution for modern enterprises seeking agility, scalability, and resilience in cloud environments. By decomposing applications into loosely coupled, independently deployable services, microservices enable organizations to scale specific components as needed, improve fault tolerance, and streamline continuous delivery pipelines. Cloud computing further enhances the benefits of microservices by providing elastic infrastructure, automated provisioning, and seamless orchestration. Application servers such as JBoss and lightweight containers like Tomcat are increasingly utilized in microservices deployments, providing enterprise-grade support for Java EE applications while facilitating containerization and cloud-native

integration. The combination of microservices and cloud computing empowers organizations to respond rapidly to changing business requirements, implement DevOps practices, and optimize resource utilization across hybrid and public cloud environments.

**Objective and Scope**

This review aims to provide a comprehensive guide for migrating monolithic Java applications hosted on JBoss and Tomcat to microservices-based architectures optimized for cloud computing. It examines technical strategies for application decomposition, containerization, and middleware integration, along with operational considerations including automation, CI/CD pipelines, monitoring, and security. The article also addresses organizational and business impacts, highlighting the benefits of increased agility, scalability, and efficiency. By presenting real-world examples, best practices, and lessons learned, this review serves as a practical roadmap for IT architects, developers, and decision-makers seeking to modernize enterprise applications while aligning technology adoption with strategic cloud initiatives.

## II. OVERVIEW OF JBOSS AND TOMCAT

**JBoss Application Server**

JBoss, now known as WildFly, is a robust Java EE application server widely used in enterprise environments to host complex, mission-critical applications. It provides full support for Java EE standards, including EJB, JMS, JPA, and web services, making it suitable for large-scale, transactional applications. JBoss offers features such as clustering, high availability, and integrated middleware capabilities, which are essential for enterprises seeking reliability and performance in distributed environments. Its modular architecture allows developers to extend functionality and integrate with external services, while built-in management tools simplify administration and monitoring. In the context of microservices migration, JBoss can host decomposed service components, providing enterprise-level features while supporting containerization and cloud deployment.

**Apache Tomcat**

Apache Tomcat is a lightweight, open-source servlet container and web application server that primarily supports Java Servlet, JSP, and WebSocket technologies. Unlike JBoss, Tomcat does not provide full Java EE support but offers a simpler, more resource-efficient platform ideal for hosting web applications or individual microservices. Its lightweight nature makes it highly suitable for containerization and deployment in cloud environments, allowing organizations to run multiple instances efficiently while optimizing resource usage. Tomcat's simplicity and speed have made it a popular choice for microservices that require low overhead, fast startup times, and easy scaling in containerized architectures.

**Comparative Analysis**

JBoss and Tomcat serve complementary roles in enterprise Java environments. JBoss is ideal for applications requiring full Java EE functionality, transactional integrity, and enterprise-grade clustering, while Tomcat excels in lightweight, web-focused microservices deployments. During migration from monolithic architectures, organizations often leverage JBoss for critical backend services and Tomcat for stateless, front-end, or API-driven components. Both platforms support containerization and integration with CI/CD pipelines, making them suitable for hybrid cloud adoption. Understanding their strengths and limitations is crucial for designing an efficient, modular microservices architecture that balances performance, scalability, and operational simplicity.

## III. MIGRATION STRATEGIES

**Assessment and Planning**

Successful migration from monolithic Java applications to microservices begins with comprehensive assessment and planning. Organizations must first perform a detailed inventory of all application components, dependencies, and workflows. Understanding which modules are tightly coupled or critical to business operations is essential for prioritizing migration efforts. Dependency mapping helps identify integration points, database interactions, and shared

resources that require careful consideration during decomposition. Additionally, risk analysis and downtime planning ensure minimal disruption to business operations. A phased migration plan, including pilot projects and incremental deployment, allows teams to validate strategies and adjust approaches before scaling across the entire application stack.

**Migration Methodologies**

Several methodologies can guide the transition from monolith to microservices. The strangler pattern gradually replaces parts of the monolithic application with microservices, allowing incremental migration without complete system downtime. Decomposition by business capability involves breaking the monolith into independent services aligned with specific business functions, improving modularity and maintainability. API-first approaches emphasize designing services around well-defined APIs, enabling interoperability between new microservices and remaining monolithic components. Enterprises may choose a combination of these strategies depending on application complexity, criticality, and resource availability. Incremental migration often reduces risk while allowing teams to learn and refine deployment practices throughout the process.

**Tooling and Automation**

Automation plays a pivotal role in modern microservices migration. CI/CD pipelines enable continuous integration, testing, and deployment, ensuring that services can be released independently without impacting other components. Build automation and containerization tools, including Docker and Kubernetes, provide standardized environments, simplify scaling, and facilitate hybrid or cloud deployment. Configuration management platforms such as Ansible, Puppet, and Terraform help automate infrastructure provisioning, service configuration, and policy enforcement. By leveraging these tools, organizations can accelerate migration, reduce human errors, maintain consistency across environments, and achieve repeatable, reliable deployments.

## IV. TECHNICAL IMPLEMENTATION

**Code Refactoring and Modularization**

Migrating monolithic Java applications to microservices requires extensive code refactoring and modularization. The monolith must be decomposed into independent, loosely coupled services, each handling a specific business capability. This process involves separating presentation layers, business logic, and data access components while ensuring minimal disruption to existing functionality. Refactoring may include rewriting tightly coupled modules, decoupling shared libraries, and adopting standard APIs for inter-service communication. Proper modularization improves maintainability, facilitates independent deployments, and enables teams to scale services individually based on demand.

**Containerization and Deployment**

Containerization is a key enabler for deploying microservices in hybrid or cloud environments. Both JBoss and Tomcat services can be packaged into Docker containers, providing isolated, consistent runtime environments that simplify deployment and scaling. Kubernetes or OpenShift orchestration platforms manage container scheduling, service discovery, and horizontal scaling, ensuring high availability and resilience. Containers also allow for rapid rollback in case of failures and facilitate continuous integration and deployment pipelines. By standardizing deployment environments, containerization reduces operational complexity and ensures reproducibility across development, testing, and production.

**Middleware and Integration**

Effective middleware and integration strategies are critical to maintain interoperability between microservices and remaining monolithic components during migration. Messaging systems such as RabbitMQ, Kafka, or JMS enable asynchronous communication and decouple service dependencies, improving scalability and fault tolerance. API gateways manage routing, load balancing, and security, providing a single entry point for service consumption. Integration with legacy databases may involve data replication,

service facades, or temporary bridging solutions until full microservices adoption is achieved. These practices ensure continuity, minimize disruption, and facilitate incremental migration while enabling a smooth transition to cloud-ready architectures.

## V. SECURITY AND COMPLIANCE

**Application Security**

Security is a critical consideration when migrating monolithic Java applications to microservices. Each microservice must implement authentication, authorization, and secure communication protocols independently. Techniques such as OAuth2, JWT (JSON Web Tokens), and role-based access control (RBAC) can enforce fine-grained access policies. Additionally, inter-service communication should utilize encrypted channels (TLS/SSL) to prevent interception or tampering. Containerized services must also adhere to secure coding practices, vulnerability scanning, and runtime security policies to reduce exposure to attacks in hybrid or cloud environments.

**Data Protection**

Distributed microservices architectures introduce complexities in managing sensitive data. Enterprises must ensure data confidentiality, integrity, and availability across multiple services and deployment environments. Encryption at rest and in transit, secure key management, and tokenization strategies are essential to protect data during migration and in production. Data consistency and transaction management across decoupled services require careful planning, including the use of eventual consistency models, distributed transaction protocols, or middleware-driven orchestration to avoid data anomalies.

**Regulatory Considerations**

Compliance with regulatory standards such as GDPR, HIPAA, PCI-DSS, and industry-specific mandates is paramount in cloud and hybrid deployments. Enterprises must ensure that microservices adhere to data residency, audit, and retention policies. Centralized logging, monitoring, and audit trails are critical to demonstrate compliance during inspections and internal audits. Automation tools, including Ansible, Terraform, and OpenSCAP, can enforce security baselines, configuration policies, and continuous compliance checks, helping organizations maintain regulatory alignment without sacrificing operational efficiency.

## VI. PERFORMANCE AND MONITORING

**Metrics and Observability**

Monitoring and observability are critical for ensuring the reliability and efficiency of microservices deployed on JBoss and Tomcat. Enterprises must capture key performance metrics such as response times, throughput, error rates, and resource utilization for each service. Observability tools like Prometheus, Grafana, and ELK Stack provide real-time insights, enabling administrators to identify bottlenecks and optimize system performance. Distributed tracing tools, including Jaeger and Zipkin, help track requests across multiple services, revealing dependencies and latency sources in complex architectures.

**Load Balancing and Scaling**

Microservices architectures require dynamic load balancing and scaling to accommodate fluctuating workloads. Kubernetes, OpenShift, and cloud-native load balancers automatically distribute requests across service instances, ensuring high availability and fault tolerance. Horizontal scaling allows additional service replicas to be deployed when demand increases, while vertical scaling optimizes resource allocation for individual containers. Autoscaling policies, informed by monitoring metrics, help maintain performance under variable loads while reducing infrastructure costs.

**Logging and Tracing**

Centralized logging is essential for troubleshooting and operational transparency in microservices deployments. Logs from JBoss and Tomcat services should be aggregated, normalized, and correlated using log management platforms. This approach simplifies error diagnosis, performance tuning, and audit compliance. Distributed tracing further enhances visibility by tracking transactions across services, allowing teams to pinpoint latency issues or failed interactions. Implementing structured logging,

log retention policies, and anomaly detection mechanisms ensures operational efficiency and proactive issue resolution.

## VII. BUSINESS IMPACT

**Operational Efficiency**

Migrating from monolithic applications to microservices on JBoss and Tomcat significantly enhances operational efficiency. Decoupled services enable independent deployment and maintenance, reducing downtime and minimizing the impact of changes on other components. Automation through CI/CD pipelines and containerized deployments streamlines repetitive tasks, accelerates release cycles, and ensures consistency across development, testing, and production environments. Teams can respond faster to business requirements, iterate on new features more effectively, and maintain higher service reliability, resulting in improved overall operational productivity.

**Cost Optimization**

Microservices architectures support cost-effective resource utilization. Lightweight containers and cloud-based infrastructure allow enterprises to scale services on-demand, reducing the need for overprovisioned on-premises hardware. Horizontal and vertical scaling, combined with autoscaling policies, optimizes computing resources, ensuring that organizations pay only for what they use. Additionally, open-source platforms such as Tomcat and JBoss help reduce licensing costs compared to proprietary enterprise software. By aligning technical efficiency with financial considerations, organizations can lower total cost of ownership (TCO) while enhancing the agility and scalability of their IT infrastructure.

**Organizational Benefits**

Beyond technical and financial gains, microservices adoption fosters organizational agility. Smaller, autonomous development teams can own specific services, promoting accountability, faster decision-making, and better alignment with business goals. Collaboration between development, operations, and business units is facilitated through shared visibility, monitoring, and standardized deployment practices. Teams gain exposure to modern DevOps practices, cloud-native technologies, and container orchestration, strengthening skill sets and supporting talent development. Collectively, these benefits drive innovation, accelerate time-to-market, and empower enterprises to respond rapidly to evolving customer needs and competitive pressures.

## VIII. CASE STUDIES AND LESSONS LEARNED

**Enterprise Adoption**

A global financial institution illustrates the benefits of migrating monolithic JBoss applications to a microservices architecture. The organization faced challenges in scaling its legacy systems to support high-volume transactions and frequent regulatory updates. By decomposing its monolithic Java applications into microservices and deploying them on Tomcat and JBoss containers, the enterprise achieved improved scalability, fault isolation, and accelerated release cycles. CI/CD pipelines automated deployment, while Kubernetes orchestration ensured high availability and efficient resource utilization. The migration reduced downtime, enhanced operational flexibility, and allowed the IT team to respond more quickly to changing business requirements.

**Mid-Market Implementation**

A mid-sized e-commerce company demonstrates how smaller organizations can benefit from microservices migration. Its monolithic application experienced performance bottlenecks during peak shopping periods, resulting in lost revenue and poor customer experiences. Migrating to microservices on Tomcat for web services and JBoss for core business logic enabled the company to distribute workloads efficiently, scale services dynamically, and implement independent service updates without impacting the entire system. The adoption of containerization and monitoring tools improved system observability, enabling proactive troubleshooting and optimized resource allocation.

**Lessons Learned**

These case studies reveal several key lessons for organizations undertaking monolith-to-microservices migrations. First, thorough assessment and planning, including dependency mapping and risk evaluation, are critical to mitigate technical and operational risks. Second, incremental migration strategies, such as the strangler pattern, reduce disruption and allow validation of new services in production environments. Third, automation and CI/CD pipelines are essential for maintaining consistency, repeatability, and quality across services. Finally, organizational readiness—including staff training in DevOps practices, cloud-native tools, and container orchestration—plays a pivotal role in ensuring a successful and sustainable migration. By applying these lessons, enterprises of varying scale can optimize performance, reduce operational complexity, and accelerate innovation while modernizing their IT infrastructure.

## IX. FUTURE TRENDS

**Serverless and Cloud-Native Microservices**

The evolution of microservices is increasingly intertwined with serverless computing and cloud-native architectures. Enterprises are adopting Function-as-a-Service (FaaS) platforms to execute individual microservice functions without managing underlying infrastructure, allowing dynamic scaling, reduced operational overhead, and cost optimization. Cloud-native platforms such as Kubernetes, OpenShift, and managed container services provide automated orchestration, service discovery, and self-healing capabilities. Integrating JBoss and Tomcat services with serverless or cloud-native environments can enhance agility, reduce latency, and improve resilience in hybrid or multi-cloud deployments.

**AI and Automation in DevOps**

Artificial intelligence and advanced automation are poised to transform microservices management. Predictive analytics, anomaly detection, and AI-driven scaling can optimize resource allocation and performance while reducing manual intervention. Intelligent CI/CD pipelines enable automated testing, deployment, and rollback strategies, accelerating development cycles and improving system reliability. AI-powered monitoring and observability platforms allow enterprises to proactively detect issues, optimize load balancing, and maintain compliance, creating a self-adaptive IT ecosystem that aligns technical operations with business objectives.

**Evolving Middleware Ecosystem**

The middleware landscape supporting JBoss, Tomcat, and microservices continues to advance. Modern middleware platforms offer improved container integration, lightweight deployment options, and enhanced cloud-native compatibility. Tools for API management, service mesh implementation, and distributed tracing are becoming standard components in enterprise architectures. As organizations embrace hybrid cloud and edge computing scenarios, middleware evolution ensures seamless integration, operational consistency, and scalable performance. Enterprises adopting these emerging technologies will be better equipped to meet dynamic business requirements and accelerate digital transformation initiatives.

## X. CONCLUSION

Migrating from monolithic Java applications to microservices using JBoss and Tomcat represents a strategic imperative for enterprises seeking agility, scalability, and operational efficiency in cloud computing environments. Monolithic architectures, while historically effective for centralized application deployment, present significant challenges as applications grow in complexity. Tight coupling, resource inefficiencies, and slow release cycles limit organizational responsiveness, making modernization essential for enterprises aiming to remain competitive in fast-evolving markets. The migration process involves careful assessment, planning, and execution. Breaking down monolithic systems into independent, loosely coupled microservices requires code refactoring, modularization, and the implementation of standardized APIs. Containerization with platforms such as Docker, coupled with orchestration tools like Kubernetes or OpenShift, enables reliable, scalable deployments across hybrid and cloud environments.

Middleware and integration strategies ensure continuity between legacy components and new services, while CI/CD pipelines and automation streamline testing, deployment, and monitoring. Security, data protection, and regulatory compliance are critical considerations, ensuring that the modernized architecture meets enterprise governance standards. From a business perspective, microservices adoption delivers measurable benefits, including faster release cycles, improved fault isolation, cost-efficient resource utilization, and enhanced operational agility. Teams gain autonomy and exposure to modern DevOps practices, while organizations can respond more effectively to changing business requirements, customer demands, and competitive pressures. Case studies from large enterprises and mid-sized organizations demonstrate the practical value of incremental migration strategies, best practices, and lessons learned, reinforcing the feasibility and impact of microservices transformations. Looking ahead, emerging trends such as serverless computing, AI-driven automation, and evolving middleware ecosystems will further enhance the efficiency and scalability of microservices architectures. Enterprises that embrace these technologies, combined with JBoss and Tomcat platforms, can achieve future-ready IT environments that support innovation, resilience, and digital transformation. In conclusion, migrating from monolithic to microservices architectures is not merely a technical upgrade—it is a comprehensive strategic initiative that aligns enterprise IT with modern business objectives, optimizing performance, scalability, and operational excellence in the cloud era.

## REFERENCE


1. Battula, V. (2015). Next-generation LAMP stack governance: Embedding predictive analytics and automated configuration into enterprise Unix/Linux architectures. International Journal of Research and Analytical Reviews, 2(3).
2. Battula, V. (2016). Adaptive hybrid infrastructures: Cross-platform automation and governance across virtual and bare metal Unix/Linux systems using modern toolchains. International Journal of Trend in Scientific Research and Development, 1(1).
3. Battula, V. (2017). Unified Unix/Linux operations: Automating governance with Satellite, Kickstart, and Jumpstart across enterprise infrastructures. International Journal of Creative Research Thoughts, 5(1). Retrieved from http://www.ijcrt.org
4. Battula, V. (2018). Securing and automating Red Hat, Solaris, and AIX: Provisioning-to-performance frameworks with LDAP/AD integration. International Journal of Current Science, 8(1). Retrieved from http://www.ijcspub.org
5. Gowda, H. G. (2017). Container intelligence at scale: Harmonizing Kubernetes, Helm, and OpenShift for enterprise resilience. International Journal of Scientific Research & Engineering Trends, 2(4), 1–6.
6. Kota, A. K. (2017). Cross-platform BI migrations: Strategies for seamlessly transitioning dashboards between Qlik, Tableau, and Power BI. International Journal of Scientific Development and Research, 3(?). Retrieved from http://www.ijsdr.org
7. Kota, A. K. (2018). Dimensional modeling reimagined: Enhancing performance and security with section access in enterprise BI environments. International Journal of Science, Engineering and Technology, 6(2).
8. Kota, A. K. (2018). Unifying MDM and data warehousing: Governance-driven architectures for trustworthy analytics across BI platforms. International Journal of Creative Research Thoughts, 6(?). Retrieved from http://www.ijcrt.org
9. Madamanchi, S. R. (2015). Adaptive Unix ecosystems: Integrating AI-driven security and automation for next-generation hybrid infrastructures. International Journal of Science, Engineering and Technology, 3(2).
10. Madamanchi, S. R. (2017). From compliance to cognition: Reimagining enterprise governance with AI-augmented Linux and Solaris frameworks. International Journal of Scientific Research & Engineering Trends, 3(3).
11. Madamanchi, S. R. (2018). Intelligent enterprise server operations: Leveraging Python, Perl, and

shell automation across Sun Fire, HP Integrity, and IBM pSeries platforms. International Journal of Trend in Research and Development, 5(6).
12. Maddineni, S. K. (2016). Aligning data and decisions through secure Workday integrations with EIB Cloud Connect and WD Studio. Journal of Emerging Technologies and Innovative Research, 3(9), 610–617. Retrieved from http://www.jetir.org
13. Maddineni, S. K. (2017). Comparative analysis of compensation review deployments across different industries using Workday. International Journal of Trend in Scientific Research and Development, 2(1), 1900–1904.
14. Maddineni, S. K. (2017). Dynamic accrual management in Workday: Leveraging calculated fields and eligibility rules for precision leave planning. International Journal of Current Science, 7(1), 50–55. Retrieved from http://www.ijcspub.org
15. Maddineni, S. K. (2017). From transactions to intelligence by unlocking advanced reporting and security capabilities across Workday platforms. TIJER – International Research Journal, 4(12), a9–a16. Retrieved from http://www.tijer.org
16. Maddineni, S. K. (2017). Implementing Workday for contractual workforces: A case study on letter generation and experience letters. International Journal of Trend in Scientific Research and Development, 1(6), 1477–1480.
17. Maddineni, S. K. (2018). Automated change detection and resolution in payroll integrations using Workday Studio. International Journal of Trend in Research and Development, 5(2), 778–780.
18. Maddineni, S. K. (2018). Governance driven payroll transformation by embedding PECI and PI into resilient Workday delivery frameworks. International Journal of Scientific Development and Research, 3(9), 236–243. Retrieved from http://www.ijsdr.org
19. Maddineni, S. K. (2018). Multi-format file handling in Workday: Strategies to manage CSV, XML, JSON, and EDI-based integrations. International Journal of Science, Engineering and Technology, 6(2).
20. Maddineni, S. K. (2018). XSLT and document transformation in Workday integrations: Patterns for accurate outbound data transmission. International Journal of Science, Engineering and Technology, 6(2).
21. Mulpuri, R. (2016). Conversational enterprises: LLM-augmented Salesforce for dynamic decisioning. International Journal of Scientific Research & Engineering Trends, 2(1).
22. Mulpuri, R. (2017). Sustainable Salesforce CRM: Embedding ESG metrics into automation loops to enable carbon-aware, responsible, and agile business practices. International Journal of Trend in Research and Development, 4(6). Retrieved from http://www.ijtrd.com
23. Mulpuri, R. (2018). Federated Salesforce ecosystems across poly cloud CRM architectures: Enabling enterprise agility, scalability, and seamless digital transformation. International Journal of Scientific Development and Research, 3(6). Retrieved from http://www.ijsdr.org
24. Ahmad, N., Naveed, Q.N., & Hoda, N. (2018). Strategy and procedures for Migration to the Cloud Computing. 2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS), 1-5.
25. Khan, N., & Al-Yasiri, A. (2016). Framework for cloud computing adoption: A road map for Smes to cloud migration. ArXiv, abs/1601.01608.
26. Buzachis, A., Galletta, A., Carnevale, L., Celesti, A., Fazio, M., & Villari, M. (2018). Towards Osmotic Computing: Analyzing Overlay Network Solutions to Optimize the Deployment of Container-Based Microservices in Fog, Edge and IoT Environments. 2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC), 1-10.
27. Carvalho, J.O., Trinta, F.A., & Vieira, D. (2018). PacificClouds: A Flexible MicroServices based Architecture for Interoperability in Multi-Cloud Environments. International Conference on Cloud Computing and Services Science.
28. Alkhalil, A., Sahandi, R., & John, D. (2017). A decision process model to support migration to cloud computing. Int. J. Bus. Inf. Syst., 24, 102-126.

29. Odun-Ayo, I., Oladimeji, T., & Odede, B. (2018). Cloud Computing Economics: Issues andDevelopments.
30. Lloyd, W.J., Vu, M., Zhang, B., David, O., & Leavesley, G.H. (2018). Improving Application Migration to Serverless Computing Platforms: Latency Mitigation with Keep-Alive Workloads. 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), 195-200.
31. Liu, X., Zhan, Z., Deng, J.D., Li, Y., Gu, T., & Zhang, J. (2018). An Energy Efficient Ant Colony System for Virtual Machine Placement in Cloud Computing. IEEE Transactions on Evolutionary Computation, 22, 113-128.
32. Gulushanian, A. (2017). Cloud Computing: New Opportunities or New Risks? How to Mitigate Cloud Risks in Oil & Gas Industry. British Journal of Applied Science and Technology, 19, 1-17.