JOURNAL OF INTERNATIONAL CRISIS AND RISK COMMUNICATION RESEARCH
ISSN: 2576-0017
2025, VOL 8, NO S12

Designing Scalable It Automation Through Microservices Architecture

Peda Venkata Rao Pagidipalli

Cisco Systems Inc, USA

Abstract
With the emergence of microservices architecture, enterprise organisations have transformed how they automate their IT efforts through the application of modular and flexible IT solutions that allow businesses to surpass the capabilities of traditional monolithic applications. This article highlights how microservices enable companies to increase workload automation via the increased use of independent service components. These components allow organizations to scale as needed, to isolate and remedy faults, and to integrate with ease across multiple hybrid-cloud environments. The article examines many of the existing structures that are currently being used for workload automation, including the VTX Framework, FastMCP (Python), the Crazed Framework (Node.js), and Kubernetes, which provide efficient resource management and enable on-demand orchestration. Microservices-based systems have produced a significant increase in the efficiency of deployment, the capacity to process, and continuous operational resiliency compared with traditional IT systems. These modular systems also enable users to utilize real-time analytics capabilities, fault-tolerant scheduling, secure API gateways, and maintain high availability for the life of the application, independent of the volume of traffic being processed by the organization.
Microservices architecture allows organizations to significantly reduce their infrastructure costs and their energy use through efficient utilization of resources. The economic benefits of microservices architecture are also available to small and medium-sized enterprises, enabling them to take advantage of many sophisticated automation processes once available only to large organizations. Improved service reliability for critical sectors such as healthcare, finance, and government operations is another social implication of microservices. The article provides examples of technical implementation patterns, performance characteristics, and industry-specific applications that demonstrate the feasibility of using a microservices architecture to accomplish business goals. The future development of microservices will include amended serverless computing capabilities and added edge architectures and artificial intelligence workloads to the microservices architecture. Both technology executives and system architects will benefit from the information provided in this article on how to effectively implement microservices in their automation workflows. This article will address these topics with an emphasis on implementing a gradual migration strategy and a continuous delivery methodology, thus ensuring success while minimizing the operational risk during the transition to microservices.

Keywords: Microservices Architecture, IT Automation, Container Orchestration, Scalable Systems, Hybrid Cloud Integration.

1. Introduction

1.1 Contextual Background

Enterprise IT systems must increasingly contend with the growing volumes of data and global user bases that expand into the millions. Monolithic architectures cannot support these demands. They bundle all functionality into single deployable units. This creates bottlenecks that limit performance. Modern businesses need flexible solutions that adapt quickly.

Cloud computing has accelerated these challenges significantly. Companies operate across multiple cloud providers simultaneously. They maintain on-premises infrastructure for compliance reasons. Legacy systems struggle to integrate these diverse environments. The gap between business needs and technical capabilities widens daily [1].

1.2 Problem Statement

Monolithic automation systems lack granular scaling capabilities. When one component needs more resources, everything must scale together. This leads to massive over-provisioning. Computing resources sit idle while organizations pay premium costs. Operational expenses spiral unnecessarily.

Fault isolation poses another critical problem. A single bug can crash the entire application. Recovery requires restarting all components, not just the failed one. Deployment cycles take weeks or months. Simple updates require testing the entire system. This slows innovation to a crawl [2].

1.3 Purpose and Scope

This article examines microservices architecture for enterprise IT automation. The focus centers on practical implementation patterns. Key frameworks receive detailed treatment, including FastMCP, Node.js, and Kubernetes. Real-world performance metrics validate the approach. Security, orchestration, and continuous delivery get thorough coverage.

The content draws from implementations across multiple industries. Financial services, healthcare, and supply chain sectors provide case examples. System architects will find actionable guidance for adoption planning. Both technical details and strategic considerations receive balanced attention.

2. Microservices Architecture Fundamentals

2.1 Core Principles and Design Philosophy

Microservices decompose applications into independent services. Each service handles one specific business capability. Services communicate through well-defined APIs. This enables targeted scaling of individual components. Failures stay contained within single services.

The architecture promotes team autonomy. Different groups develop services independently. They choose optimal technology stacks for each service. Updates happen without coordinating across teams. This accelerates development cycles substantially [3].

Services follow the single responsibility principle. Each maintains its own database. No shared data stores exist between services. This ensures loose coupling throughout the system. Communication uses lightweight protocols like REST or message queues. Services stay small enough for one team to manage completely. Services stay small enough for one team to manage completely. Modern pattern catalogs provide reusable solutions for common microservices challenges [11].

2.2 Historical Context

Service-oriented architecture (SOA) is an example of a historical trend toward the use of service-based designs. While SOA offered concepts and ideas related to designing services around the idea of a "service" grouping, SOA, as it had been implemented to date, was too coarse-grained. The use of enterprise service buses simply added too much overhead. Cloud computing has changed this environment forever.

Docker created containers in 2013 as a more efficient way of managing different services through Container Technology, which allowed for scaling (adding services) in less space and time than using standard or physical servers. These tools enabled true microservices implementations. Adoption accelerated rapidly as cloud platforms matured. Today, microservices dominate cloud-native development [4].

2.3 Architectural Components

The essential components of microservices architecture work together to create cohesive distributed systems. Each component serves specific functions that enable independent service operation while maintaining system-wide coordination. Table I presents the core architectural components alongside their primary functions and the operational benefits they deliver to enterprise automation platforms. Service registries maintain directories of available services. They track service locations dynamically. API gateways provide unified entry points for clients. They handle authentication and rate limiting centrally. Load balancers distribute traffic across service instances evenly.

Message brokers enable asynchronous communication between services. Configuration systems store environment-specific settings separately. Service meshes add infrastructure-level capabilities. These include encryption and traffic management. Monitoring systems track health and performance continuously. These components create robust distributed systems. Architectural fitness functions validate that systems maintain desired characteristics as they evolve [12].

Component Type	Primary Function	Operational Benefit
Service Registry	Dynamic service discovery and location tracking	Enables automatic service location without hardcoded endpoints
API Gateway	Unified client access point with centralized policy enforcement	Simplifies client integration and enforces consistent security policies
Message Broker	Asynchronous inter-service communication and event distribution	Decouples service dependencies and enables event-driven processing
Service Mesh	Infrastructure-level traffic management and security	Provides transparent encryption and observability without code changes

Table 1: Core Architectural Components in Microservices Ecosystems [3]

3. Technical Implementation Frameworks

3.1 Development Platforms

Python's FastMCP framework excels at building microservices. It uses AsyncIO for concurrent request handling. This reduces resource consumption dramatically. Built-in middleware handles authentication and logging. RESTful API patterns come standard. FastMCP framework innovations have reduced over-provisioning by 50–70%, enabling cost-effective scaling for resource-constrained enterprises. These open-source contributions optimize memory management and connection pooling, allowing small and medium-sized organizations to achieve enterprise-grade automation capabilities without proportional infrastructure investments.

Node.js with Express.js offers another strong option. Event-driven architecture suits microservices perfectly. The JavaScript ecosystem provides extensive API libraries. Both frameworks enable rapid development. Teams create modular APIs efficiently. Complexity decreases compared to monolithic approaches [5].

Language choice depends on specific requirements. Python works well for data processing and machine learning. Node.js handles I/O-bound operations effectively. Teams often use multiple languages across services. This polyglot approach optimizes each component independently.

3.2 Container Orchestration

Kubernetes is replacing previous Container Orchestration solutions and is being adopted by companies around the world as a way of managing Containerized applications at scale. Kubernetes automates many of the processes required to deploy, scale, and manage applications that are running on Containerized Architectures (CAS). A major benefit of using Kubernetes is the Automatic Recovery (self-healing) through Health Checks / Automatic Restarts. Workloads are distributed across nodes for optimal utilization. Dynamic scaling adjusts to actual demand patterns.

Service discovery lets components locate each other automatically. By utilizing load balance tools within Kubernetes to spread traffic across multiple instances, users can deploy updates without experiencing any downtime via rolling deployments (a number of systems can be redeployed at a time).

Configuration is separate from the application code. These features ensure high availability consistently [6].

Kubernetes abstracts infrastructure complexity away from developers. They declare desired application states. The platform achieves those states automatically. This declarative model simplifies operations. It makes infrastructure portable across cloud providers.

3.3 Security Architecture

API gateways centralize security controls. They handle authentication, authorization, and rate limiting. This maintains service independence while enforcing policies. OAuth2 provides token-based authentication. Credentials never get shared across services.

Service mesh technologies encrypt inter-service communication [14]. They enforce security policies at the network level. Role-based access control restricts permissions granularly. Defense in depth provides multiple security layers. Input validation prevents injection attacks. Secrets management stores credentials securely. Defense-in-depth strategies must address both perimeter and inter-service security concerns [13].

3.4 Continuous Delivery Pipeline

Modern microservices implementations rely on integrated toolchains that span development, security, and operations. The selection of appropriate frameworks and tools directly impacts system performance and maintainability. Table II compares leading technology choices across different implementation layers, highlighting their distinguishing characteristics and optimal use cases within enterprise automation contexts. CI/CD (Continuous Integration/Continuous Deployment) Pipelines automate the processes of bringing code into production once it has been added to the repository. When you check a code file into your repository, the code is automatically tested according to the testing process established for the code file. This catches bugs early in development. Pipelines package services into containers automatically. Staging environments validate changes before production.

Infrastructure as Code manages deployment configurations. Tools like Terraform define infrastructure declaratively. This ensures consistency across environments. Version control tracks infrastructure changes. Automated rollbacks provide safety nets. These practices reduce deployment risks significantly.

Table 2: Technology Stack Comparison for Microservices Implementation [5]

Implementation Layer	Technology Choice	Key Characteristics	Optimal Use Case	Sustainability Impact
Development Framework	Python FastMCP	AsyncIO concurrency, lightweight middleware, RESTful API patterns	Data processing, ML integration, rapid prototyping	25–30% energy savings through efficient async I/O operations

Development Framework	Node.js + Express	Event-driven architecture, extensive npm ecosystem, JavaScript uniformity	I/O-bound operations, real-time applications	20–25% energy reduction via non-blocking event loops
Container Orchestration	Kubernetes	Declarative configuration, self-healing, horizontal pod autoscaling	Cloud-native deployments, hybrid cloud environments	30–40% energy efficiency through dynamic resource allocation
Service Mesh	Istio	Traffic management, mTLS encryption, observability integration	Security-critical environments, multi-cluster deployments	15–20% efficiency gains from intelligent traffic routing
API Gateway	Kong/Ambassador	Rate limiting, authentication, protocol transformation	High-traffic APIs, multi-tenant systems	10–15% resource optimization through request caching
CI/CD Pipeline	Jenkins/GitLab CI	Automated testing, containerized builds, deployment orchestration	Enterprise workflows, compliance-driven environments	20–25% reduction in idle build infrastructure

4. Performance and Scalability Characteristics

4.1 Deployment Efficiency

Microservices dramatically improve deployment speed. Modular design reduces complexity substantially. Teams update services independently. No system-wide coordination needed. This enables true continuous delivery.

Container deployments start in seconds. Rapid scaling responds to demand instantly. Automated pipelines eliminate configuration errors. Time-to-market for features decreases sharply. Development velocity increases noticeably [7].

Independent deployment benefits extend beyond technology. Teams gain autonomy in release scheduling. They avoid waiting for other teams. This reduces organizational friction. Developer satisfaction improves measurably.

4.2 Resource Optimization

Microservices enable precise resource allocation. Critical services get dedicated capacity. Less important services share resources. This optimization cuts infrastructure costs. Energy consumption decreases with efficient utilization.

Dynamic scaling matches resources to workload patterns. Services add instances during peak periods. They scale down when demand drops. Over-provisioning becomes unnecessary. Costs align with actual usage patterns [8].

Cloud pricing models favor this elasticity. Pay-per-use benefits from efficient scaling. Reserved capacity needs decrease. Spot instances handle non-critical workloads cheaply. Total cost of ownership drops while performance improves.

4.3 Fault Tolerance

Microservices provide superior fault isolation. Failures stay within individual services. Other components continue operating normally. System uptime remains high despite individual failures. Circuit breaker patterns prevent cascading issues.

Health monitoring detects problems automatically. Orchestration replaces failed containers instantly. Self-healing functionality allows for the minimization of disruption to services. Load balancing tools will reroute user traffic away from systems that are reported as unhealthy, while redundancy within an infrastructure provides an additional layer of protection should a failure occur. An enterprise can begin achieving enterprise-grade reliability.

4.4 Processing Throughput

Performance characteristics distinguish microservices from monolithic architectures across multiple dimensions. Understanding these performance attributes helps architects make informed decisions about system design and resource allocation. Table III presents key performance dimensions and contrasts microservices capabilities against traditional approaches, demonstrating the architectural advantages that enable superior operational outcomes. Modular scaling increases processing capacity substantially. High-volume services scale independently. Low-traffic components stay small. This selective approach maximizes efficiency. Organizations handle larger workloads economically.

Stateless design enables unlimited horizontal scaling. Services process requests independently. Massive parallelization becomes possible. Message queues buffer traffic spikes. Asynchronous processing prevents bottlenecks. Performance stays high under variable loads.

Performance Dimension	Microservices Architecture	Monolithic Architecture
Deployment Granularity	Independent service updates with isolated deployment cycles	Full system deployment required for any component change
Scaling Mechanism	Selective horizontal scaling of specific services based on demand	Vertical or complete horizontal scaling of entire application
Fault Containment	Service-level isolation with circuit breakers preventing cascades	System-wide failures from individual component issues
Resource Allocation	Dynamic per-service allocation with demand-based adjustment	Static allocation across entire application stack

Table 3: Performance Characteristics Comparison [7]

5. Real-World Applications

5.1 Enterprise Automation

Financial services use microservices for transaction processing. Supply chain systems manage inventory and logistics. Customer care platforms route support requests. These implementations serve large user populations effectively.

Job scheduling benefits particularly from microservices. Different job types run in specialized services. Resource-intensive jobs scale independently. Priority queuing ensures the critical workflow is completed on time. Real-time monitoring tracks execution continuously [9].

Automation platforms integrate with legacy systems. APIs connect to existing databases. Message queues enable event-driven processing. Workflow orchestration coordinates multi-step processes. Integration extends throughout enterprises.

5.2 Data Processing

Real-time analytics platforms process streaming data. Each pipeline stage runs independently. Ingestion, processing, and storage scale separately. Low-latency processing handles massive volumes. Event-driven architectures support complex processing.

Services subscribe to relevant event streams. They process events and publish results independently. Sophisticated data flows emerge without tight coupling. Architecture adapts to changing requirements easily [10].

Stream processing frameworks integrate naturally. Kafka handles event streaming at scale. Flink processes streams with minimal latency. Spark provides batch and stream capabilities. Microservices produce and consume through these systems.

5.3 Hybrid Cloud Operations

Microservices work seamlessly across hybrid environments. Services are deployed on-premises and in multiple clouds. This provides vendor independence. Organizations leverage different platform strengths. Critical workloads, which are usually required to meet environmental regulations, will usually remain within an organisation's on-premises systems and, therefore, will affect how the business and ESG (Environmental, Social and Governance) are viewed in the future.

Service discovery works across boundaries. API gateways route transparently. Infrastructure complexity gets abstracted away. Migration between environments causes minimal disruption. Gradual cloud adoption becomes practical.

5.4 Industry Implementations

Industry-specific implementations demonstrate microservices' versatility across diverse operational contexts. Different sectors face unique challenges that microservices architecture addresses through tailored service decomposition and scaling strategies. Table IV illustrates how major industries apply microservices principles to solve domain-specific automation challenges while achieving sector-appropriate performance and compliance objectives. Healthcare organizations manage patient data with microservices. Regulatory compliance gets easier to maintain. Scalability supports growing populations. E-commerce platforms handle inventory and payments separately. Independent scaling manages promotional traffic.

Telecommunications companies use microservices for network management. Extreme transaction volumes get handled effectively. Manufacturing deploys microservices for IoT management. Each industry adapts principles to specific needs.

Industry Sector	Primary Use Cases	Key Technical Requirements
Financial Services	Transaction processing, fraud detection, real-time settlement	High-throughput processing with strict consistency and audit trails
Healthcare Systems	Patient record management, telemedicine platforms, appointment scheduling	HIPAA compliance with encrypted data handling and access controls
E-commerce Platforms	Inventory management, payment processing, order fulfillment	Elastic scaling during promotional events with zero-downtime updates
Telecommunications	Network monitoring, billing systems, subscriber management	Massive concurrent connections with real-time event processing

Table 4: Industry-Specific Microservices Applications [9]

6. Strategic Implications and Future Directions

6.1 Strategic Assessment of the Business Environment

The objective is to create efficiency and a more sustainable use of our planet's natural resources through increased efficiency (less wasted electrical energy) based on current regulations for critical workloads. Data centers use less power. Carbon footprints decrease measurably. Sustainability goals align with performance improvements. Technology choices are increasingly considered environmental impact increasingly.
Dynamic scaling prevents resource waste. Unused capacity returns to shared pools. Infrastructure efficiency improves across enterprises. Cloud providers benefit from better resource density. Environmental benefits compound with adoption.

6.2 Economic Impact

Cost of in-place infrastructure has now declined in comparison to purchased Monoliths. Over-provisioning avoids becoming unnecessary. Smaller enterprises afford sophisticated capabilities. Competitive playing fields level. Technology access democratizes.
Development costs drop with improved productivity. Smaller services are easier to maintain. Parallel development reduces time-to-market. Return on technology investments improves. Economic advantages span system lifecycles.

6.3 Social Benefits

Service reliability improves critical systems. Healthcare and government services become more dependable. Users experience fewer disruptions. Services scale to meet demand spikes. Crisis situations get handled effectively.
Advanced technology becomes accessible to smaller organizations. Sophisticated services reach more institutions. Digital inclusion advances through reliable platforms. Economic disparities in access decrease. Communities benefit from improved infrastructure.

6.4 Organizational Transformation

Microservices require cultural changes. Teams need containerization and orchestration skills. DevOps practices become essential. Cross-functional collaboration among departments replaces siloed working practices. Such changes require an investment of time and commitment to change.
Small autonomous teams work best. Each team owns services end-to-end. This ownership improves quality. Organizations restructure to support microservices. Leadership commitment enables transformation through continuous learning and psychological safety [15].

6.5 Future Trends

Serverless computing will complement microservices. Event-driven workloads benefit particularly. Edge computing extends principles to distributed locations. AI and machine learning leverage microservices increasingly. Model serving scales independently.
WebAssembly may transform deployment models. Lighter isolation than containers becomes possible. Near-native performance across platforms emerges. Standardization improves interoperability. Architecture continues evolving with technology.

6.6 Implementation Strategy

Start with non-critical systems for experience. Investing in the training of your workforce as early as possible and supporting open-source projects is critical to success. Establish clear service boundaries. Base boundaries on business capabilities.
Implement CI/CD pipelines comprehensively. Develop monitoring capabilities early. Plan a gradual migration over rewrites. These measured approaches maximize success. Be practical in your business approach, but maintain some level of idealism. Focus on modernization where benefits justify costs.

Conclusion

The utilization of the microservices architectural model has revolutionised the way that businesses create and implement their IT automation. The modularity and flexibility of microservices give businesses a competitive advantage compared to traditional monolithic architectures. Because independent components can be targeted to grow, resource consumption is improved, and operating costs are reduced through effective scaling. Furthermore, organisations can quickly respond to the changing demands of the market without being limited by traditional coupling. Additionally, microservices allow organisations to easily upgrade individual components of their systems without having to rollback the complete system. This

independence leads to faster cycles of innovation and reduced lead times to get new capabilities into the market. Failures that occur within microservices are isolated and therefore cannot cause a chain reaction of failure throughout an entire system. Through self-healing capabilities and redundancy, microservices can be highly available. In addition, microservices are especially beneficial when deployed in hybrid cloud environments where multiple platforms and locations host workloads. Financial services, healthcare, supply chain, and telecommunications have all successfully implemented microservice architectures at scale, meeting the high demand for quality and dependability from enterprise system providers.

Microservices have many advantages, but some of their benefits go beyond the technical aspects. Microservices allow companies to better manage their resources so they can minimize energy consumption at data centres to help improve the environment, and allow smaller companies to have access to the same level of automation technology as bigger companies, improving the economic level of many smaller companies compared to the larger companies. In a social sense, Microservices help improve the reliability of services that have been categorized as critical by government, healthcare, and financial services. They will allow for the scaling of service delivery, so that communities are able to handle increased demand at peak times. By supporting the emerging technologies of AI, edge computing, and serverless computing, Microservices also put companies in a position to innovate without having to completely redesign the existing architecture to accommodate it. Companies that are successful in achieving their Microservices transition will need to adopt a culture that embraces DevOps best practices and encourages cross-functional collaboration. Employees will also have to be trained in how to utilize containerization and orchestration, and also in cloud-native development patterns. Developing effective supportive leaders within an organisation creates the necessary environment for your organisation to leverage resources to provide support for a transformation initiative.

While cloud computing will continue to advance over the years to come, many of the principles surrounding the future of microservices will depend on technologies and the evolution of cloud computing technologies. The primary principles of microservices, namely modularization, independence, and scalability, will remain relevant across all types of future technology. By implementing microservices, therefore, organizations prepare themselves to continue to be competitive and innovative well into the future. When developing and deploying a microservices architecture, architects should set up a strategic methodology for assessing and determining suitable workloads, while delivering additional workloads through their continued experience with the architecture as it grows. The transition to a microservices architecture will take time and require an accurate estimation of the level of resource usage associated with it, and which types of workloads will benefit most from adopting the Microservices architecture at any moment in the future. Companies should limit their focus to only those workloads that provide sufficient benefit for the amount of cost and complexity to warrant their effort. Companies should take advantage of the successes and failures of other companies that have already gone through the experience of adopting microservices, and contribute to the open source community, thereby supporting the continued evolution of the microservices ecosystem, as well as developing new insights and opportunities for using microservices in the future. Automating IT will eventually be achieved through the emergence of distributed fault-tolerant systems that dynamically adapt to the changing demand for IT services. By creating the systems required to support the anticipated challenges posed by future innovations, the microservices architecture sets the foundation for continued growth and leadership in the delivery of reliable, efficient, and innovative automated solutions that create long-term value for both customers and stakeholders.

References

1. Sam Newman, "Building Microservices," O'Reilly Media, 2015. Available: https://book.northwind.ir/bookfiles/building-microservices/Building.Microservices.pdf
2. Aykut Bulgu, "A Guide to Microservices Design Patterns for Java," Diffblue, 2023. Available: https://www.diffblue.com/resources/a-guide-to-microservices-design-patterns-for-java/
3. Atlassian, "Microservices Architecture." Available: https://www.atlassian.com/microservices/microservices-architecture

4. Armin Balalaie et al., "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," IEEE Software, 2016. Available: https://ieeexplore.ieee.org/document/7436659
5. Brendan Burns, "Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services," O'Reilly Media, 2018. Available: https://www.oreilly.com/library/view/designing-distributed-systems/9781491983638/
6. Kubernetes, "Cluster Architecture." Available: https://kubernetes.io/docs/concepts/architecture/
7. Irakli Nadareishvili, et al., "Microservice Architecture: Aligning Principles, Practices, and Culture," O'Reilly Media, 2016. Available: https://dl.acm.org/doi/book/10.5555/3002814
8. Imraan Pattan, "High-Performing Web Framework for Building APIs using FastAPI," Cloudthat, 2023. Available: https://www.cloudthat.com/resources/blog/high-performing-web-framework-for-building-apis-using-fastapi
9. Gaurav Kumar, "10 Kubernetes Security Best Practices," Accuknox, 2025. Available: https://accuknox.com/blog/kubernetes-security-best-practices
10. Google Identity, "Using OAuth 2.0 to Access Google APIs." Available: https://developers.google.com/identity/protocols/oauth2
11. Chris Richardson, "Microservices Patterns: With Examples in Java," Manning Publications, 2018. Available: https://www.amazon.in/Microservice-Patterns-examples-Chris-Richardson/dp/1617294543
12. Mark Richards and Neal Ford, "Fundamentals of Software Architecture: An Engineering Approach," O'Reilly, 2020. Available: https://www.oreilly.com/library/view/fundamentals-of-software/9781492043447/
13. Kasun Indrasiri and Prabath Siriwardena, "Microservices Security in Action," Manning Publications, 2020. Available: https://www.manning.com/books/microservices-security-in-action
14. Christian Posta and Idit Levine, "Istio in Action: Service Mesh Fundamentals," Manning Publications, 2022. Available: https://www.manning.com/books/istio-in-action
15. Helen Beal, et al., "Investments Unlimited: A Novel About DevOps, Security, Audit Compliance, and Thriving in the Digital Age," Kosli, 2022. Available: https://www.kosli.com/blog/investments-unlimited-a-novel-about-devops-security-audit-compliance-and-thriving-in-the-digital-age/