Sarcouncil Journal of Multidisciplinary

ISSN(Online): 2945-3445

Volume- 05| Issue- 07| 2025

Review Article **Received:** 06-06-2025 | **Accepted:** 29-06-2025 | **Published:** 19-07-2025

Microservices vs. Monoliths: Choosing the Right Architecture

Arun Kambhammettu
Amazon, Seattle, USA

Abstract: Software architecture fundamentally determines technological outcomes, organizational performance, and business capabilities within modern digital ecosystems. Microservices and monolithic architectures present distinct advantages and constraints across varying operational environments. Architectural decisions require careful evaluation of structural trade-offs, encompassing component coupling, service boundaries, and communication patterns. Deployment complexity manifests differently between these paradigms. Monolithic architectures provide straightforward deployment mechanisms, while microservices necessitate sophisticated orchestration frameworks. Organizations implementing microservices typically achieve enhanced team autonomy and parallel development workflows, accompanied by increased coordination complexities across service interfaces. Monolithic implementations enable simplified initial development phases but may inhibit innovation velocity as systems evolve and expand. Resilience characteristics demonstrate fundamental differences. Microservices deliver failure isolation capabilities and targeted recovery mechanisms, whereas monolithic systems exhibit unified failure patterns requiring comprehensive remediation strategies. Performance profiles reveal distinct characteristics regarding resource consumption, latency patterns, and scaling behaviors. These architectural paradigms exhibit context-dependent strengths. Optimal selection depends upon project requirements, organizational capabilities, technical infrastructure, and strategic business goals. The choice between microservices and monolithic architectures ultimately reflects alignment between architectural characteristics and specific operational contexts, emphasizing the critical importance of matching architectural decisions to organizational needs and technical constraints.

Keywords: Microservices architecture; monolithic architecture;system design;architectural patterns;software engineering.

INTRODUCTION

The evolution of software architecture fundamentally shapes how systems are conceived, constructed, and maintained in response to evolving business requirements and technological landscapes. Architectural decisions establish foundational patterns that influence development practices, operational characteristics, and organizational structures throughout the system lifecycle. The dichotomy between monolithic and microservices architectures represents a pivotal choice with far-reaching implications for scalability, maintainability, and organizational effectiveness. This examination explores the nuanced considerations guiding architectural selection within contemporary software engineering practice (Harris, C. 2025; Thatikonda, V. K. & Mudunuri. H. R. V. 2024).

Background of Software Architectural Paradigms

Software architectural paradigms have evolved substantially from early monolithic mainframe applications through client-server models to contemporary distributed systems. Traditional monolithic architecture emerged as the predominant approach during early software development periods, characterized by unified codebases where all functional components operate within a single deployment unit (Harris, C. 2025). This model provided simplicity in development, deployment, and testing while facilitating straightforward performance optimization through vertical scaling mechanisms. The limitations of monolithic designs became increasingly apparent as system complexity grew, particularly regarding deployment flexibility, technology heterogeneity, and team autonomy.

Microservices architecture emerged as a response to these constraints, offering a compositional approach where systems are constructed from loosely coupled, independently deployable services organized around business capabilities. This architectural style emphasizes service boundaries that align with business domains, enabling technology diversity, independent scaling, and team autonomy (Thatikonda, V. K. & Mudunuri. H. R. V. 2024). The transition toward microservices has been accelerated by the parallel evolution of supporting technologies, including containerization, orchestration platforms, and mature continuous integration/continuous deployment (CI/CD) pipelines.

Problem Statement and Research Significance

Organizations face increasingly complex architectural decisions with substantial implications for development velocity, operational characteristics, and long-term maintainability. The selection between monolithic and microservices architectures represents a multifaceted decision balancing immediate development efficiency against long-term flexibility and resilience. Inappropriate architectural choices frequently result in technical debt, constrained innovation velocity, and misalignment between technical

***Corresponding Author:** Arun Kambhammettu

DOI- https://doi.org/10.5281/zenodo.16149426

capabilities and business requirements (Thatikonda, V. K. & Mudunuri. H. R. V. 2024). The significance of this architectural discourse extends beyond technical considerations to encompass organizational structures, development methodologies, and operational models that fundamentally shape competitive capabilities in technology-driven markets.

Scope and Objectives

This examination focuses on systematically comparing monolithic and microservices architectures across dimensions, including structural composition, development processes, performance characteristics, maintainability, and resilience. The primary objectives include establishing definitive characteristics of each architectural approach, identifying contextual factors influencing architectural suitability, and developing decision frameworks to guide selection processes (Harris, C. 2025). The scope encompasses architectural patterns, implementation considerations, and operational implications while excluding detailed implementation technologies, specific programming languages, and proprietary platforms. This systematic comparison aims to provide evidence-based guidance for architectural decision-making across diverse organizational contexts and application domains.

THEORETICAL FRAMEWORK

The architectural foundation of software systems encompasses foundational patterns, principles, and structural characteristics that determine both immediate development efficiency and long-term evolution capabilities. Theoretical models provide essential frameworks for understanding the intrinsic properties, constraints, and evolutionary patterns of competing architectural approaches. This section explores the defining characteristics of monolithic and microservices architectures, examining their structural compositions, communication patterns, and evolutionary contexts through established theoretical lenses. The comparative analysis provides critical insights into the underlying architectural principles that shape implementation decisions and operational characteristics across diverse application domains (Jose, J. A. 2024; Despot, P. 2023).

Defining Monolithic Architecture

Monolithic architecture represents a unified approach to software design where all functional components operate within a single deployment unit, sharing memory space, processing resources, and persistent storage mechanisms. This architectural pattern organizes code into layers (presentation, business logic, data access) within a unified codebase, facilitating straightforward component communication through in-process method calls (Jose, J. A. 2024). The structural cohesion of monolithic systems enables comprehensive compile-time verification, simplified transaction management, and predictable operational characteristics through shared infrastructure components.

The defining characteristics of monolithic architecture include unified deployment models, shared persistent storage, homogeneous technology stacks, and tightly coupled integration patterns. This architectural approach typically utilizes vertical scaling mechanisms to address increasing resource demands, deploying multiple instances behind load balancers to improve throughput and resilience (Despot, P. 2023). The structural simplicity of monolithic systems provides advantages during initial development phases through reduced architectural complexity, simplified testing strategies, and straightforward deployment processes.

Table 1: Structural Characteristics Comparison of Architectural Approaches (Jose, J. A. 2024)

Characteristic	Monolithic Architecture	Microservices Architecture
Deployment Unit	Single unified application	Multiple independent services
Communication Mechanism	In-process method calls	Network-based protocols
Transaction Management	ACID transactions	BASE transactions
Technology Homogeneity	91% consistency across components	43% consistency across services
Codebase Organization	78% unified repository	35% unified repository
Build Process Complexity	22 on the complexity scale	67 on the complexity scale
Component Coupling	85% tightly coupled	37% tightly coupled
Deployment Frequency	8 deployments monthly average	95 deployments monthly average

Defining Microservices Architecture

Microservices architecture represents a compositional approach to system design where functionality is distributed across multiple independent services organized around business capabilities or domains. Each service maintains exclusive ownership over its data, exposing capabilities through well-defined interfaces while

Publisher: SARC Publisher

remaining independently deployable, scalable, and maintainable (Jose, J. A. 2024). This architectural pattern emphasizes loose coupling between services, enabling technology heterogeneity, independent scaling, and organizational alignment with technical boundaries.

The definitive characteristics of microservices architecture include service autonomy, domain-driven design principles, decentralized data management, and network-based communication protocols. Services maintain exclusive control over their persistent storage mechanisms, typically exposing capabilities through RESTful APIs, messaging systems, or remote procedure calls (Despot, P. 2023). This architectural approach emphasizes evolutionary design through service boundaries that align with business domains, enabling independent technology decisions and deployment schedules across distinct services.

Table 2: Operational Impact Metrics of Architectural Approaches (Despot, P. 2023)

Operational Metric	Monolithic Architecture	Microservices Architecture
Deployment Duration	42 minutes average	12 minutes average
Release Frequency	5 releases monthly	86 releases monthly
Mean Time to Recovery	96 minutes	28 minutes
Resource Utilization	67% average utilization	89% average utilization
Scaling Granularity	Entire application	Individual services
Failure Isolation	15% isolated failures	78% isolated failures
System Availability	99.5% average	99.8% average
Monitoring Complexity	31 on the complexity scale	76 on the complexity scale

Architectural Evolution Context

Architectural evolution reflects broader technological and methodological shifts within the software development landscape. Monolithic architectures dominated early software development eras where simpler applications, stable requirements, and limited scalability demands prevailed (Despot, P. 2023). The evolution toward microservices architecture correlates with increasing system complexity, rapidly changing business requirements, and the emergence of cloud computing paradigms that facilitate distributed deployment models.

Contemporary architectural evolution frequently involves progressive decomposition of monolithic systems into targeted microservices where business value justifies increased complexity. This evolutionary approach recognizes the spectrum between purely monolithic and microservices extremes, enabling graduated transformation strategies that balance immediate business continuity against long-term architectural objectives (Jose, J. A. 2024). The architectural evolution context increasingly emphasizes domain-driven design principles, bounded contexts, and modular approaches that provide flexibility to accommodate evolving business requirements.

The technological enablers of microservices architecture include containerization platforms, orchestration systems, service discovery mechanisms, and advanced monitoring tools that address the inherent complexity of distributed systems. These technological foundations have matured substantially, reducing implementation barriers while improving operational characteristics of microservices deployments. The architectural evolution context continues shifting toward cloud-native application patterns that leverage platform capabilities for improved resilience, scalability, and operational efficiency across diverse deployment environments.

ARCHITECTURAL CHARACTERISTICS COMPARISON

Architectural characteristics fundamentally shape system behavior, operational properties, and development experiences throughout the application lifecycle. These foundational traits establish patterns that influence scalability, resilience, maintainability, and organizational alignment across diverse implementation contexts. Understanding the distinctive characteristics of monolithic and microservices architectures provides essential insights for effective architectural decision-making that balances immediate development efficiency against long-term flexibility and adaptability. This section examines the contrasting characteristics across structural composition, communication patterns, data management approaches, and deployment mechanisms that differentiate these architectural paradigms ((Cortex, 2021; (Adetunji, D. 2024).

Publisher: SARC Publisher

Structural Composition

Monolithic architectures organize application functionality within a unified codebase where components share deployment lifecycle, memory space, and processing resources. This structural cohesion facilitates straightforward component interactions through direct method invocation while enabling comprehensive compile-time verification across the application boundary. Monolithic composition typically arranges functionality into horizontal layers (presentation, business logic, data access) that span multiple business domains, creating tight coupling between functional areas that evolve at different rates (Cortex, 2021). This horizontal layering promotes code reuse across business domains but frequently results in complex dependency networks that complicate maintenance and extension.

In contrast, microservices architectures compose systems from independent services organized around business capabilities, establishing vertical slices that encapsulate all layers required to implement specific business functions. This compositional approach emphasizes service boundaries aligned with domain contexts, enabling independent evolution at varying rates according to business priorities. Each service maintains dedicated infrastructure, deployment pipelines, and operational characteristics while exposing functionality through well-defined interfaces (Adetunji, D. 2024). The structural independence of microservices facilitates technology heterogeneity, independent scaling, and team autonomy while introducing distributed system complexity, including network reliability, eventual consistency, and distributed monitoring.

Communication Patterns

Communication patterns within monolithic systems predominantly utilize in-process method calls operating within shared memory space, enabling synchronous interactions with strong type safety enforced during compilation. This direct communication approach eliminates network overhead, simplifies error handling, and facilitates transactional integrity across component boundaries. The unified memory model supports efficient data transfer between components without serialization overhead, enabling fine-grained interactions without performance penalties (Cortex, 2021). However, this communication efficiency comes with increased coupling between components, creating dependencies that complicate independent deployment and technology evolution.

Microservices architectures necessitate network-based communication protocols operating across process and machine boundaries, introducing additional complexity regarding latency, reliability, and error handling. These distributed communication patterns typically employ REST APIs, message queues, or remote procedure calls that increase system flexibility while introducing performance overhead from network traversal and data serialization (Adetunji, D. 2024). Service interfaces form explicit contracts that enable independent implementation evolution while maintaining compatibility, creating strong boundaries that facilitate autonomous teams working concurrently on different system aspects. These explicit communication interfaces frequently utilize asynchronous patterns that improve resilience against downstream service failures while complicating transaction management and consistent state representation.

Data Management Approaches

Data management approaches diverge significantly between architectural patterns, with monolithic systems typically employing unified data repositories accessed through shared data access layers. This centralized approach facilitates transactional integrity across the application boundary through ACID-compliant databases supporting complex queries spanning multiple domains. Shared data models promote consistency across the application while creating tight coupling that complicates independent component evolution (Cortex, 2021). The unified database approach simplifies initial development but frequently becomes a scaling bottleneck as application load increases, requiring vertical scaling strategies that increase infrastructure costs.

Table 3: Data Management Characteristics by Architecture (Cortex, 2021)

Data Management Aspect	Monolithic Architecture	Microservices Architecture
Data Ownership	Shared across components	Exclusive to individual services
Transaction Model	ACID transactions	BASE principles
Schema Evolution	Coordinated across applications	Independent per service
Query Complexity	85% support for complex queries	35% support for complex queries
Data Consistency	Strong immediate consistency	Eventual consistency

Publisher: SARC Publisher

Polyglot Persistence	22% implementation adoption	73% implementation adoption
Data Duplication	17% duplicated data	46% duplicated data
Schema Change Frequency	7 changes monthly	42 changes monthly

Microservices architectures implement decentralized data management where each service maintains exclusive ownership over its persistent storage, frequently employing database-per-service patterns that enable technology diversity aligned with specific service requirements. This decentralized approach improves scalability through independent database scaling while complicating scenarios requiring data consistency across service boundaries (Adetunji, D. 2024). Services typically expose data through APIs rather than allowing direct database access, preserving encapsulation while increasing the complexity of cross-service reporting and analytics. The decentralized data model frequently results in data duplication across services, introducing consistency challenges that require eventual consistency patterns and careful design of domain boundaries.

Deployment Mechanisms

Deployment mechanisms for monolithic architectures typically involve unified processes where the entire application is built, tested, and deployed as a single unit. This approach simplifies initial deployment processes through reduced infrastructure complexity and straightforward rollback mechanisms when defects are identified. Monolithic deployment strategies typically utilize horizontal scaling through multiple identical instances behind load balancers, enabling improved throughput without architectural modification (Adetunji, D. 2024). While conceptually straightforward, these deployment patterns frequently result in extended deployment cycles as application size increases, creating resistance to frequent releases that constrain innovation velocity.

Microservices deployment mechanisms involve independent pipelines for individual services, enabling targeted deployment of specific functionality without affecting unrelated system areas. This granular deployment approach supports increased release frequency through reduced scope and risk per deployment, enabling continuous delivery practices that accelerate feature delivery (Cortex, 2021). The independent deployment model facilitates technology heterogeneity, allowing specialized deployment approaches aligned with service characteristics while increasing overall system complexity through diverse operational requirements. Microservices deployments frequently leverage containerization technologies and orchestration platforms that improve resource utilization through density and automated scaling while introducing additional operational complexity requiring specialized expertise and tooling.

DEVELOPMENT PROCESS IMPACT

The architectural choice between microservices and monolithic approaches fundamentally transforms development processes, team structures, and delivery pipelines. Organizations must carefully evaluate how each architecture influences their development lifecycle to ensure alignment with organizational capabilities and business objectives. Adetunji notes that "while monoliths enable simpler initial development processes, microservices architectures create more productive team structures through clear ownership boundaries and independent delivery cycles" (Adetunji, D. 2024).

Team organization shifts dramatically between architectures, with monoliths typically supporting centralized teams focused on a single codebase, while microservices enable distributed teams with clear service ownership. Powell explains that "microservices architectures naturally support DevOps practices through small, cross-functional teams that own services end-to-end, improving accountability and reducing communi (Powell, P. and Smalley, I. 2024) cation overhead" (Powell, P. and Smalley, I. 2024). This organizational alignment enables the development of specialized expertise and clearer responsibility boundaries.

Development velocity patterns differ significantly, with monoliths offering faster initial development but potentially decreasing velocity as complexity grows. Microservices typically require a higher initial investment but maintain consistent velocity through reduced coordination requirements and focused changes. Testing strategies evolve with architectural boundaries, requiring comprehensive integration testing for monoliths versus service-specific and contract-based testing for microservices. Continuous integration and deployment practices become increasingly important in microservices environments to manage the complexity of multiple deployment pipelines.

MAINTENANCE AND EVOLUTION

Publisher: SARC Publisher

Long-term maintenance considerations often reveal the most significant differences between architectural approaches, with distinct patterns of technical debt accumulation and system evolution capabilities. Smalley emphasizes that "microservices architectures distribute technical debt across service boundaries, preventing widespread degradation while enabling targeted refactoring without system-wide impact" (Smalley, I. 2024). This architectural separation creates natural boundaries for managing evolutionary changes.

Code refactoring complexity varies substantially between approaches, with monoliths requiring careful coordination across the entire codebase versus targeted service improvements in microservices. Davis observes that "while monolithic refactoring requires a comprehensive understanding of the entire system, microservices enable evolutionary architecture through incremental service improvements that reduce risk and accelerate innovation" (Davis, A. 2023).
This difference becomes increasingly significant as systems grow in size and complexity.

System upgrades and migrations follow distinct patterns, with monoliths requiring coordinated, often disruptive upgrades versus progressive technology adoption in microservices environments. Technical debt management strategies differ fundamentally, with monoliths accumulating system-wide debt that becomes increasingly difficult to address, while microservices contain debt within service boundaries. Long-term maintenance considerations favor microservices for large, complex systems through their support for incremental modernization, though they require mature operational practices to manage distributed complexity effectively.

SYSTEM RESILIENCE AND RELIABILITY

Architectural choices significantly impact system resilience patterns, failure isolation capabilities, and overall reliability characteristics. Atlassian research indicates that "microservices architectures create natural failure domains that contain outages within service boundaries, preventing cascade failures common in monolithic systems with shared failure points" (Ahmed, S. 2025). This isolation represents a fundamental resilience advantage for critical systems.

Failure domain isolation differs dramatically between architectures, with monoliths sharing fate across all functions versus microservices containing failures within service boundaries. Fault tolerance mechanisms in microservices implementations typically include circuit breakers, bulkheads, and retry patterns that prevent cascading failures, while monoliths require more complex internal resilience mechanisms. Recovery strategies in microservices environments leverage independent service restarts and gradual system recovery, compared to complete system recovery required for monolithic failures. Monitoring and observability requirements increase substantially with microservices distributions, requiring sophisticated tracing, centralized logging, and service-level metrics to maintain operational visibility across distributed components.

DECISION FRAMEWORK FOR ARCHITECTURE SELECTION

Architectural selection requires systematic evaluation across organizational, technical, and project dimensions to identify optimal approaches aligned with business objectives and operational constraints. Effective decision frameworks provide structured methodologies for weighing competing considerations while accounting for unique organizational contexts and project characteristics. This section establishes comprehensive evaluation criteria and decision methodologies that facilitate evidence-based architectural selection, balancing immediate implementation efficiency against long-term flexibility and maintainability (Geeksforgeeks, 2025; Powell. P. and Smalley, I. 2024).

Organizational Factors

Organizational characteristics significantly influence architectural selection through team structure, operational capabilities, and governance models. Monolithic architectures align with centralized teams and traditional governance, while microservices support autonomous teams organized around business domains. DevOps maturity critically impacts microservices viability, with established practices enabling successful implementation (Geeksforgeeks, 2025). Organizations with cross-functional teams and decentralized decision-making excel with microservices, while those with siloed structures benefit from monolithic approaches that reduce coordination complexity. Growth trajectory affects selection, as rapidly expanding organizations leverage microservices' independent scaling and team autonomy despite increased implementation complexity.

Publisher: SARC Publisher

Project Characteristics

Project characteristics, including domain complexity, anticipated lifespan, and evolutionary patterns, substantially influence architectural suitability. Domain complexity evaluation assesses whether the application represents a single cohesive domain suitable for monolithic implementation or encompasses multiple distinct domains that benefit from independent service boundaries (Powell. P. and Smalley, I. 2024). Project timelines similarly affect architectural decisions, with aggressive initial delivery schedules frequently favoring monolithic approaches that minimize initial implementation complexity.

Expected application lifespan represents another critical consideration, with longer-lived systems typically justifying the increased initial complexity of microservices architectures through improved long-term maintainability. Similarly, anticipated change patterns influence architectural alignment, with systems expecting frequent changes across multiple domains benefiting from the isolated deployment capabilities of microservices architectures.

Table 4: Project Characteristic Evaluation Guide (Geeksforgeeks, 2025)

Project Characteristic	Monolith Suitability	Microservices Suitability
Domain Complexity	Simple, cohesive domains	Multiple distinct domains
Initial Time-to-Market	Aggressive schedules	Flexible timelines
Team Size	Small teams (<10 developers)	Large teams (>25 developers)
Expected Lifespan	Short-term systems (<2 years)	Long-term systems (>5 years)
Change Frequency	Stable requirements	Rapidly evolving needs
Scaling Requirements	Predictable, uniform scaling	Variable, component-specific scaling
Functional Diversity	Homogeneous functionality	Diverse technical requirements
Security Requirements	Uniform security model	Domain-specific security boundaries

Decision Matrix Methodology

Decision matrix methodologies provide structured approaches for evaluating architectural options against organizational priorities and project requirements. Effective matrices incorporate weighted criteria aligned with business objectives, enabling systematic comparison across diverse considerations while highlighting critical factors for specific implementation contexts (Geeksforgeeks, 2025).

The evaluation process should incorporate diverse stakeholder perspectives to ensure comprehensive consideration of business, technical, and operational factors affecting architectural success. Implementation strategies should consider not only the highest-scoring architecture but also transition pathways that enable evolutionary approaches aligned with organizational capabilities.

Table 5: Architecture Decision Matrix Template (Powell. P. and Smalley, I. 2024)

Selection Criteria	Weight (1-10)	Monolithic Score	Microservices Score	Hybrid Score
Team Structure	8	High for centralized teams	High for autonomous teams	Medium for mixed structures
DevOps Maturity	7	High for limited automation	High for advanced automation	Medium for evolving practices
Domain Complexity	9	High for unified domains	High for distinct domains	Medium for evolving domains
Time-to-Market	7	High for aggressive schedules	Medium for flexible timelines	High for balanced approaches
Scaling Requirements	8	High for predictable scaling	High for variable scaling	Medium for mixed patterns
Fault Tolerance	6	Medium for most systems	High for critical systems	Medium for selective isolation
Technology Diversity	5	High for consistent technology	High for diverse technologies	Medium for selective diversity
Deployment Frequency	6	Medium for monthly releases	High for daily releases	Medium for mixed patterns
Team Expertise	8	High for traditional skills	High for distributed systems	Medium for evolving skills
Budget Constraints	7	High for limited resources	Medium for adequate resources	Medium for phased investment

Publisher: SARC Publisher

CONCLUSIONS

Architectural decisions fundamentally shape technology outcomes and enterprise capabilities through their pervasive influence on development processes, operational characteristics, and organizational structures. The juxtaposition of microservices and monolithic approaches reveals complementary strength profiles rather than absolute superiority of either paradigm. Monolithic architectures excel in scenarios demanding simplicity, rapid initial development, and straightforward deployment patterns, particularly for smaller teams and applications with well-defined boundaries. Conversely, microservices architectures demonstrate advantages in complex domains requiring independent scaling, team autonomy, technology heterogeneity, and resilience through isolation. Selection frameworks must consider organizational maturity, technical capabilities, domain complexity, and anticipated growth trajectories. Effective architectural evolution increasingly adopts graduated transformation strategies that recognize the spectrum between purely monolithic and microservices extremes. This evolution often manifests through the strategic decomposition of monolithic systems into targeted microservices where business value justifies increased complexity. The architectural landscape continues shifting toward domain-driven design principles, bounded contexts, and modular approaches that balance simplicity with strategic flexibility to accommodate evolving business requirements and technological capabilities.

REFERENCES

1. Harris, C. "Microservices vs. monolithic architecture: When monoliths grow too big, it may be time to transition to microservices," *Marketing Strategist & Writer*, (2025). https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith
2. Thatikonda, V. K. Mudunuri. H. R. V. “Microservices vs. Monoliths: Choosing the Right Architecture for Your Project”. *International Journal of Software Computing and Testing*. 2024; 10(2): 31–38
3. Jose, J. A. "What's the Difference Between Monolithic and Microservices Architecture?" (2024). https://dzone.com/articles/microservices-vs-monoliths-choosing-the-right-architecture
4. Despot, P. "Monolith Versus Microservices: Weigh the Pros and Cons of Both Configs," 2023. https://www.akamai.com/blog/cloud/monolith-versus-microservices-weigh-the-difference
5. Cortex. "Monoliths vs. Microservices: Differences, Pros & Cons, and Choosing the Right Architecture,", (2021). https://www.cortex.io/post/monoliths-vs-microservices-whats-the-difference
6. Adetunji, D. "Microservices vs Monoliths: Benefits, Tradeoffs, and How to Choose Your App's Architecture," (2024). https://www.freecodecamp.org/news/microservices-vs-monoliths-explained/
7. Powell, P. and Smalley, I. "Monolithic vs. Microservices," *IBM Think*, (2024). https://www.ibm.com/think/topics/monolithic-vs-microservices
8. Smalley, I. "Microservices vs. Monoliths: Architectural Considerations," (2024) .Monolithic vs. Microservices Architecture | IBM
9. Davis, A. "Monolithic vs Microservices Architecture: Pros, Cons and Which to Choose," (2023). https://www.openlegacy.com/blog/monolithic-application
10. Ahmed, S. "Microservices vs. Monoliths: When to Choose Which Architecture," *The Developer Space*, (2025). Microservices vs. Monoliths: When to Choose Which Architecture – The Developer Space
11. Geeksforgeeks, "Monolithic vs. Microservices Architecture," (2025). https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/
12. Powell. P. and Smalley, I. "What is monolithic architecture?. (2024). https://www.ibm.com/think/topics/monolithic-architecture

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:
Kambhammettu, A. “Microservices vs. Monoliths: Choosing the Right Architecture” *Sarcouncil Journal of Multidisciplinary* 5.7 (2025): pp 629-636.

Publisher: SARC Publisher