A Survey of Microservices Autoscaling and Container Orchestration in Cloud Computing

www.surveyx.cn

Abstract

This survey paper explores the transformative role of microservices architecture in cloud computing, highlighting its advantages in scalability, availability, and fault tolerance. Microservices, by decomposing monolithic applications into independently deployable services, enhance flexibility and scalability, crucial for dynamic cloud environments. The integration of autoscaling and container orchestration, particularly through platforms like Kubernetes, is pivotal in optimizing resource management, reducing operational costs, and improving application performance. Autoscaling ensures resource efficiency by dynamically adjusting to workload fluctuations, while container orchestration automates deployment and scaling, thus overcoming traditional scaling limitations. Decision models and advanced orchestration techniques, including AI-driven frameworks, further enhance microservices architecture by providing structured guidance for pattern selection and resource allocation. Despite these benefits, challenges such as security vulnerabilities, resource allocation trade-offs, and the complexity of distributed systems remain. The paper underscores the need for robust frameworks and methodologies to address these challenges, emphasizing the importance of security mechanisms and informed decision-making in cloud deployments. Future research directions include refining decomposition criteria, integrating AI for predictive resource management, and expanding benchmarks to evaluate design pattern impacts on performance metrics. By leveraging innovative strategies, organizations can achieve resilient, efficient, and scalable digital infrastructures, aligning with the evolving demands of modern computing environments.

1 Introduction

1.1 Significance of Microservices in Cloud Computing

Microservices architecture is pivotal in cloud-native applications, revolutionizing the development and deployment of scalable web applications [1]. This approach allows for the decomposition of monolithic applications into smaller, independently deployable services, facilitating various architectural patterns that enhance scalability and flexibility [2]. In cloud environments, microservices effectively manage complex interactions, concurrent events, and component failures, which are prevalent challenges in cloud-native applications [3].

The modularity of microservices promotes collaborative development across teams, enabling organizations to build applications that adapt to dynamic workloads while maintaining service quality [4]. This adaptability is essential in cloud computing, where cost-effectiveness and performance are critical, necessitating careful resource allocation to optimize performance [5]. Furthermore, microservices simplify the development process for computational scientists, allowing them to concentrate on their specific expertise while delegating integration tasks to software engineers [6].

Microservices also play a crucial role in addressing common migration issues from monolithic systems, providing a framework for identifying and avoiding anti-patterns [7]. By strategically

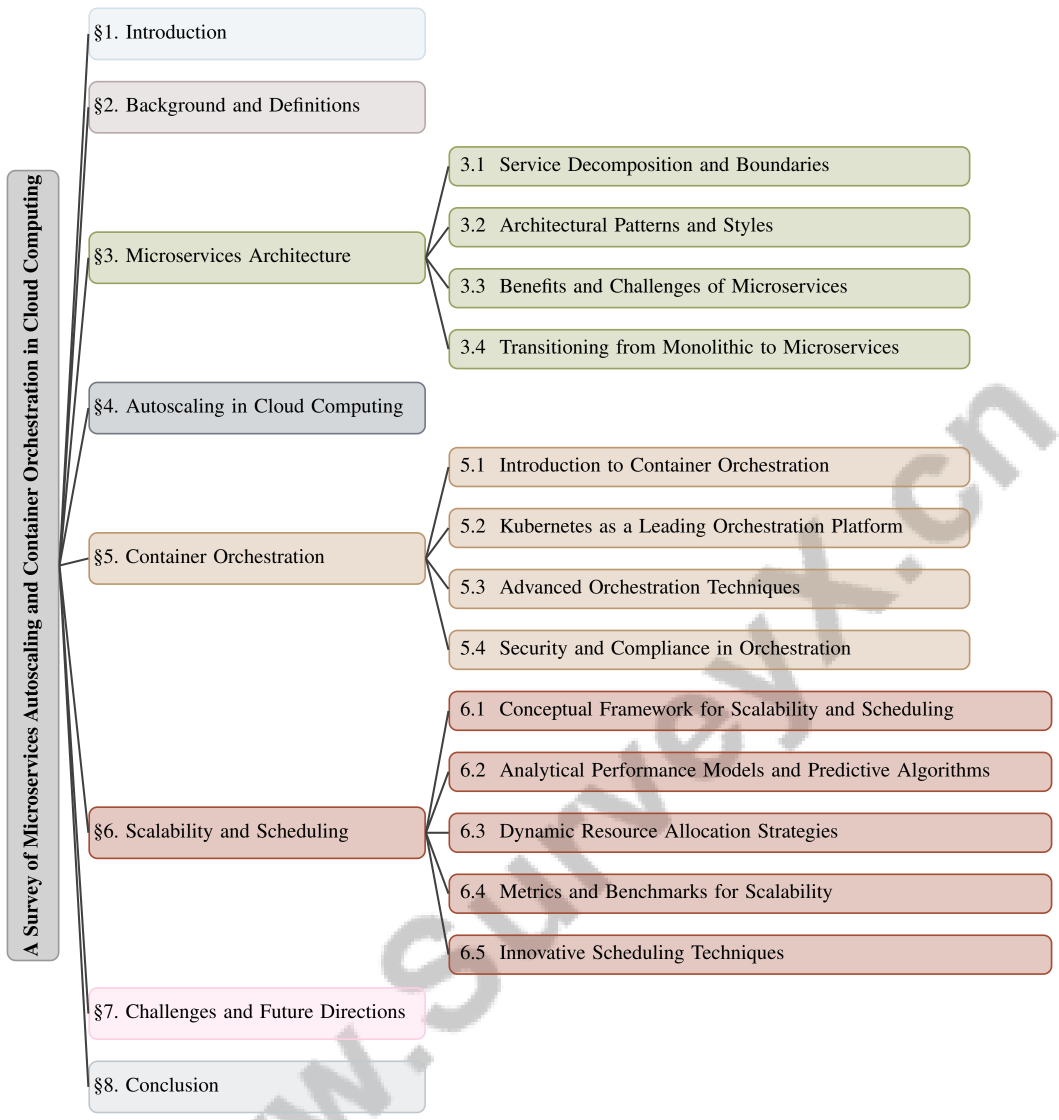

A Survey of Microservices Autoscaling and Container Orchestration in Cloud Computing
§1. Introduction
§2. Background and Definitions
§3. Microservices Architecture
3.1 Service Decomposition and Boundaries
3.2 Architectural Patterns and Styles
3.3 Benefits and Challenges of Microservices
3.4 Transitioning from Monolithic to Microservices
§4. Autoscaling in Cloud Computing
§5. Container Orchestration
5.1 Introduction to Container Orchestration
5.2 Kubernetes as a Leading Orchestration Platform
5.3 Advanced Orchestration Techniques
5.4 Security and Compliance in Orchestration
§6. Scalability and Scheduling
6.1 Conceptual Framework for Scalability and Scheduling
6.2 Analytical Performance Models and Predictive Algorithms
6.3 Dynamic Resource Allocation Strategies
6.4 Metrics and Benchmarks for Scalability
6.5 Innovative Scheduling Techniques
§7. Challenges and Future Directions
§8. Conclusion

Figure 1: chapter structure

applying architectural design patterns, microservices enhance system performance metrics, driving innovation and efficiency in cloud computing [8].

1.2 Importance of Autoscaling and Container Orchestration

Autoscaling and container orchestration are vital for managing microservices within dynamic cloud environments. Autoscaling enables automatic adjustment of computational resources, essential for maintaining system reliability and performance amidst varying software versions and workloads [9]. This capability is particularly important for addressing complex interdependencies and runtime challenges in microservices, especially during live migration scenarios [10]. AI techniques further enhance autoscaling by tackling complexities and improving system quality attributes, optimizing resource allocation [11].

Container orchestration platforms, such as Kubernetes, automate the deployment, scaling, and management of containerized applications, significantly alleviating the operational complexities associated with microservices [12]. This automation is crucial for managing intricate service relationships and dependencies, often inadequately handled by traditional methods [13]. Container orchestration supports efficient resource utilization and addresses the limitations of conventional scaling techniques through optimal distribution and dynamic reconfiguration of microservices [14].

AI-generated, for reference only.

The interplay between autoscaling and container orchestration is critical in managing the architectural complexity and deployment coordination challenges inherent in microservices [15]. Together, these technologies provide a robust framework to meet the performance and scalability demands of modern microservices architectures, fostering sustainable application development [16]. They facilitate the transition from monolithic systems to microservices, highlighting their significance in evolving to more flexible architectures [17]. The orchestration of microservices simplifies debugging and maintenance, promoting a streamlined development process [3]. Moreover, the complexity of end-to-end performance as influenced by resource allocation can lead to overprovisioning, which autoscaling can mitigate, ensuring efficient resource usage [5]. The literature emphasizes the importance of selecting and implementing microservices architecture patterns and strategies in addressing practitioners' challenges [2]. The collaborative model-driven development approach further underscores the critical role of these technologies in managing distributed development complexities [4].

1.3 Paper Objectives and Structure

This survey provides a comprehensive examination of microservices architecture, focusing on the integration of autoscaling and container orchestration within cloud computing environments. A primary objective is to elucidate the principles and practices that underpin microservices architecture while intentionally excluding unrelated software development paradigms [18]. The survey aims to propose novel algorithms for architecture-level run-time adaptation, ensuring optimal deployment orchestrations to maintain a target Maximum Computational Load (MCL) [14].

Additionally, the survey introduces KubeNow as a cloud-agnostic platform that enhances the management of scientific applications through a flexible and scalable microservice-oriented architecture [19]. By clarifying the role of AI techniques in the development, optimization, and operation of microservices, the survey addresses automation challenges in deploying and managing complex cloud-native applications [11].

The survey also provides insights into architectural considerations, infrastructure support, and code management in microservices, drawing on the experiences of seasoned practitioners in the field [20]. The paper is structured as follows: Section 2 presents essential background and definitions of key concepts such as microservices, autoscaling, and container orchestration. Section 3 delves into microservices architecture, exploring service decomposition, architectural patterns, and the transition from monolithic systems. Section 4 focuses on autoscaling in cloud computing, discussing strategies and their applications in microservices. Section 5 examines container orchestration, particularly emphasizing Kubernetes and advanced orchestration techniques. Section 6 explores scalability and scheduling, outlining frameworks, models, and innovative techniques from recent research. Finally, Section 7 identifies current challenges and future research directions, offering insights into the evolving landscape of microservices in cloud computing.The following sections are organized as shown in Figure 1.

2 Background and Definitions

2.1 Background and Definitions

Grasping the foundational concepts of microservices, autoscaling, scheduling, container orchestration, and scalability is pivotal for designing and managing distributed systems in cloud computing. Microservices architecture decomposes applications into independently deployable services, enhancing flexibility and scalability, necessitating meticulous orchestration to manage interactions and inefficiencies while emphasizing decentralized governance and autonomous teams. Model-driven Development (MDD) supports this by promoting collaborative processes that enhance team interactions [4].

Autoscaling is vital in cloud environments, enabling dynamic resource adjustments to meet fluctuating demands, optimizing resource utilization in asynchronous, computationally intensive applications through predictive metrics [1]. Challenges like hardware heterogeneity and performance pattern drifts necessitate sophisticated frameworks to sustain system performance.

Scheduling in cloud computing involves strategic resource and task allocation to optimize workload distribution, crucial for enhancing performance and minimizing latency in microservices architectures. These architectures face challenges in performance predictability and resource management, especially when deploying across multiple containers or integrating with Edge Computing for low-latency benefits [21, 22]. Effective scheduling frameworks are essential for efficient scaling in local and cloud environments.

Cloud computing, as the delivery of computing services via the internet, offers on-demand access to resources like servers, storage, and applications, enhancing scalability and flexibility. Organizations can dynamically allocate resources to meet real-time demands, ensuring SLA compliance and optimizing operational costs. Advanced models, such as the Temporal Fusion Transformer for predictive analytics, enable proactive resource management based on workload intensity and inter-service dependencies, enhancing performance and user experience in microservice architectures [23, 18, 24, 25, 19]. Deploying microservices in resource-constrained environments like edge computing underscores the need for robust foundational understanding.

Container orchestration, exemplified by Kubernetes, automates deployment, scaling, and management of containerized applications, ensuring operational efficiency and optimal performance in large-scale environments. It addresses microservices architecture complexities, including tenant isolation in multi-tenant cloud environments, resource management, and shared infrastructure support. It facilitates the deployment of independent services, enforces security policies, and enhances application adaptability, aiding transitions from monolithic to microservices-based architectures [20, 17, 26]. The role of microservices as components in complex applications underscores their significance in modern software architecture.

Scalability, a key characteristic of cloud-native applications, refers to a system's ability to manage increased load by adding resources. This capability is crucial for evolving applications to meet changing user demands without sacrificing performance or reliability. Performance evaluations of microservices architectures use benchmarks to address scalability challenges, offering insights into the effectiveness of various architectural patterns and strategies [1]. These concepts collectively form the backbone of contemporary cloud computing systems, facilitating the development of scalable, resilient, and efficient applications that meet today's dynamic digital landscape demands.

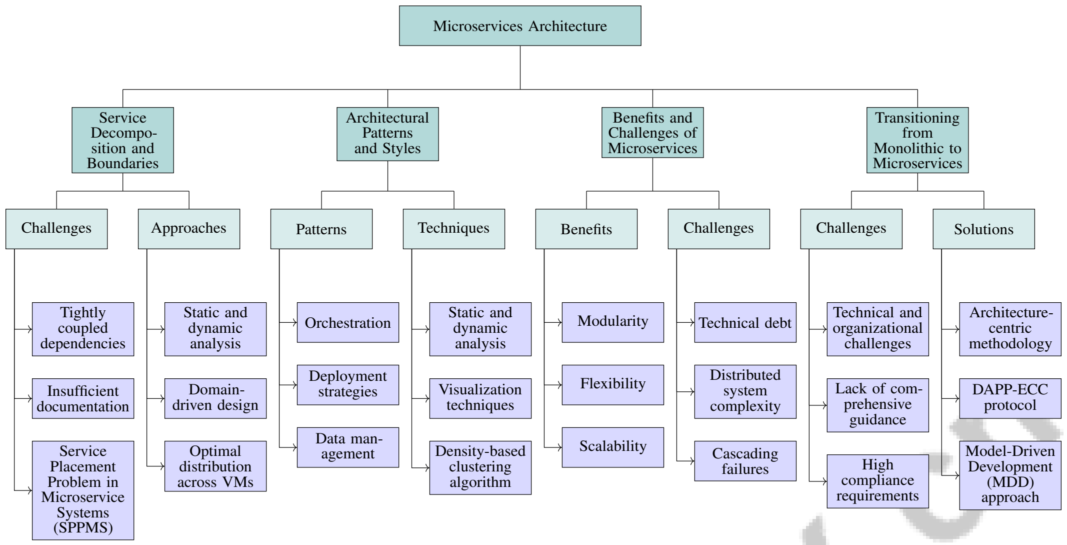
In recent years, the adoption of microservices architecture has transformed the landscape of software development, offering significant advantages over traditional monolithic systems. This shift necessitates a comprehensive understanding of the hierarchical structure underlying microservices. Figure 2 illustrates this hierarchical structure, highlighting key aspects such as service decomposition, architectural patterns, benefits and challenges, and the transition from monolithic systems. Specifically, it categorizes the challenges and approaches involved in service decomposition, the patterns and techniques prevalent in architectural styles, and the associated benefits and challenges of microservices. Furthermore, it addresses the complexities and solutions related to the transition from monolithic architectures to microservices, thereby providing a holistic view of this architectural paradigm. Such an understanding is crucial for practitioners aiming to leverage the full potential of microservices in their development processes.

3 Microservices Architecture

3.1 Service Decomposition and Boundaries

Service decomposition is a core aspect of microservices architecture, transforming monolithic applications into discrete, loosely coupled services to enhance scalability and maintainability [27]. This process is challenged by tightly coupled dependencies and insufficient documentation, complicating the identification of service boundaries [27]. A systematic approach employing static and dynamic analysis is crucial to derive candidate microservices from monolithic systems, ensuring that the resulting services are well-defined for independent development and deployment [28]. The effectiveness of these techniques is gauged by their ability to produce scalable and maintainable services.

Domain-driven design offers a structured method for service decomposition, emphasizing clear boundaries to maintain service autonomy and minimize dependencies [29]. This approach categorizes research into various decomposition strategies, underscoring the importance of service boundaries in enhancing orchestration and resource management within microservices systems [30]. Key considerations include microservice granularity and ownership, affecting the independence and control of each service [20]. Practitioners often face challenges in selecting appropriate patterns and strategies, leading to inefficiencies in service design, security, communication, and discovery [2]. The Service Placement Problem in Microservice Systems (SPPMS) complicates this by requiring



Microservices Architecture
Service Decomposition and Boundaries
Architectural Patterns and Styles
Benefits and Challenges of Microservices
Transitioning from Monolithic to Microservices
Challenges
Approaches
Patterns
Techniques
Benefits
Challenges
Challenges
Solutions
Tightly coupled dependencies
Insufficient documentation
Service Placement Problem in Microservice Systems (SPPMS)
Static and dynamic analysis
Domain-driven design
Optimal distribution across VMs
Orchestration
Deployment strategies
Data management
Static and dynamic analysis
Visualization techniques
Density-based clustering algorithm
Modularity
Flexibility
Scalability
Technical debt
Distributed system complexity
Cascading failures
Technical and organizational challenges
Lack of comprehensive guidance
High compliance requirements
Architecture-centric methodology
DAPP-ECC protocol
Model-Driven Development (MDD) approach

Figure 2: This figure illustrates the hierarchical structure of microservices architecture, highlighting key aspects such as service decomposition, architectural patterns, benefits and challenges, and the transition from monolithic systems. It categorizes the challenges and approaches involved in service decomposition, the patterns and techniques in architectural styles, the benefits and challenges of microservices, and the challenges and solutions in transitioning to microservices.

the deployment of microservices with complex dependencies and multiple instances to ensure stable Quality of Service (QoS) [13].

Deploying microservices involves optimal distribution of fine-grained components across Virtual Machines (VMs) and dynamic reconfiguration to effectively manage workload fluctuations [14]. This dynamic nature necessitates a robust framework that categorizes research into various deployment approaches and communication patterns, facilitating effective microservice architecture [15].

3.2 Architectural Patterns and Styles

Architectural patterns and styles are pivotal in determining the efficiency and effectiveness of microservices applications. These patterns guide the decomposition of applications into microservices, ensuring functional coherence and manageability. Taibi et al. classify these patterns into orchestration, deployment strategies, and data management, providing a comprehensive framework for their application in microservices architecture [31]. The integration of static and dynamic analysis is crucial for refining microservice candidates. Krause et al. propose a method that combines these analyses with visualization techniques to enhance service granularity and autonomy by improving microservice boundary identification [32]. Decision models, such as Waseem et al.'s, offer structured guides for selecting patterns and strategies based on quality attributes, aligning architectural decisions with organizational goals [33].

Migrating from monolithic systems to microservices involves complex decision-making. Chaieb et al. present a method using a density-based clustering algorithm, considering static analysis and inter-class relationships to produce functionally coherent microservices [34]. This highlights the importance of selecting architectural patterns that accommodate unique dependencies and interactions within microservices systems.

3.3 Benefits and Challenges of Microservices

Microservices architecture offers advantages in modularity, flexibility, and scalability, crucial for modern software development [35]. It enables decomposition into smaller, independently deployable services, enhancing adaptability and flexibility [29]. By allowing independent development, deployment, and scaling, microservices improve system robustness and responsiveness to workload fluctuations [36]. The architecture's flexibility allows organizations to leverage diverse technologies, accelerating development cycles [37]. Employing various software versions within a microservices architecture enhances resilience against failures and performance degradation, bolstering overall system robustness [9]. Collaboration between experts and engineers is improved, facilitating complex integrations and enhancing software quality [6].

However, microservices architecture also presents challenges, such as managing technical debt and achieving appropriate service granularity [36]. The complexity of distributed systems complicates testing and debugging, especially concerning distributed transactions [38]. The risk of cascading failures among services necessitates robust failure management strategies [39]. Network contention and QoS violations are significant concerns in large-scale applications with complex dependencies [39].

Deployment challenges include managing independently failed services, deployment coordination, distributed logs, and associated costs [15]. Establishing service ownership, limiting language diversity, and implementing robust logging and monitoring frameworks early in development are crucial [20]. Solutions like the HA State Controller integrated with Kubernetes aim to manage state replication and redirect services to healthy instances, ensuring availability and reliability [12]. Traditional metrics like CPU utilization may not accurately reflect microservice performance, leading to inefficient scaling decisions that can violate SLAs [37].

AI has the potential to enhance microservices performance and reliability, particularly in operational contexts, despite limitations during development phases [11]. The MSTG framework addresses the often-overlooked networking layer in microservices evaluation, providing enhanced flexibility and scalability [40]. While microservices architecture presents substantial benefits, addressing inherent challenges requires systematic methodologies, innovative metrics, and comprehensive frameworks to fully realize its potential.

3.4 Transitioning from Monolithic to Microservices

Transitioning from monolithic architectures to microservices is complex, with technical and organizational challenges. This shift is driven by the limitations of monolithic systems, such as tightly coupled components that hinder scalability and adaptability [27]. Identifying suitable microservice boundaries requires meticulous dependency analysis and architectural restructuring to maintain low coupling and high cohesion.

A primary challenge in this transition is the lack of comprehensive guidance and best practices, exacerbated by tight IT budgets and high compliance requirements [41]. An architecture-centric methodology has been proposed to address these challenges, offering a structured framework for migrating to microservices. This methodology guides architects through the migration process, providing a systematic approach to assess legacy systems, define migration strategies, and implement microservices incrementally [41].

The migration process is further complicated by the need for efficient placement and migration of microservices across data centers, which can be facilitated by protocols like DAPP-ECC to ensure stability and performance during the transition from centralized approaches [42]. Adopting an orchestrated approach, as opposed to a choreographed one, significantly eases debugging efforts, making it a worthwhile investment for developers [3].

As illustrated in Figure 3, the transition from monolithic to microservices architecture encompasses the main challenges, proposed solutions, and the critical role of automation and tools in facilitating this process. Automation plays a critical role in streamlining the migration process, as highlighted by the complexities involved in migrating monolithic applications to microservices architecture [27]. Transitioning from traditional development methods to a collaborative Model-Driven Development (MDD) approach for Microservices Architecture (MSA) can help manage the complexity of distributed systems, facilitating more efficient and effective migrations [4].

4 Autoscaling in Cloud Computing

Autoscaling is pivotal in cloud computing, allowing dynamic resource adjustments to accommodate fluctuating workloads, thus enhancing efficiency and performance in cloud-based applications. Table 1 presents a detailed overview of the autoscaling strategies and techniques, as well as their applications

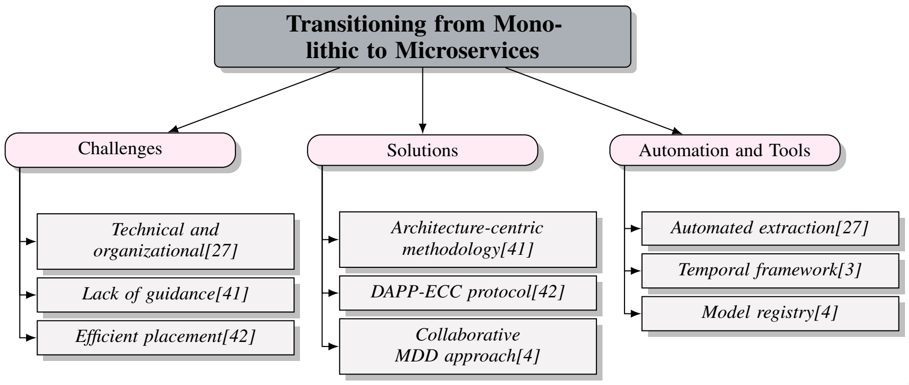


Figure 3: This figure illustrates the transition from monolithic to microservices architecture, highlighting the main challenges, proposed solutions, and the role of automation and tools in facilitating this process.

Category	Feature	Method
Autoscaling Strategies and Techniques	Dynamic Resource Management	SASAF[43]
	Algorithmic Optimization	RS[44]
	Performance and Reliability	DAPP-ECC[42]
	Continuous Integration and Deployment	MDI[6]
Applications in Microservices	System Continuity and Reliability	MS2M[10], DAA[9]
	Predictive and Decision-Making Models	CPMS[35], MMPA[45]
	Optimization and Efficiency	COLA[46], TBOM[47]

Table 1: This table provides a comprehensive summary of various autoscaling strategies and techniques, alongside their respective applications in microservices. It categorizes the methods into two main areas: Autoscaling Strategies and Techniques, and Applications in Microservices, highlighting key features and methodologies employed in each category. The table serves as a reference for understanding the diverse approaches to optimizing resource management and system performance in cloud computing environments.

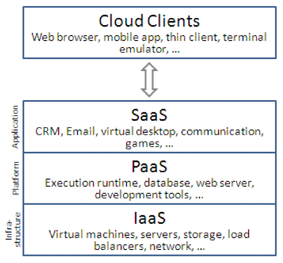
in microservices, which are crucial for optimizing resource management and enhancing performance in cloud computing. Additionally, Table 3 offers a comprehensive comparison of various autoscaling methods, demonstrating their unique approaches to scaling, resource management, and predictive analytics, which are critical for enhancing efficiency and performance in cloud computing systems. This section examines various strategies and techniques developed to optimize resource management and system performance.

4.1 Autoscaling Strategies and Techniques

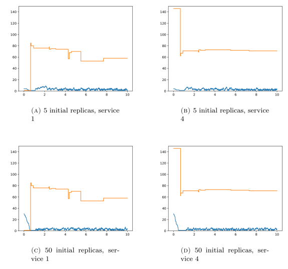
Autoscaling is crucial for adapting computational resources to changing demands, ensuring cost-effectiveness and performance, especially in microservices architectures, which benefit from independent service scaling [46]. A variety of strategies have been devised to address resource management and workload prediction challenges.

A prominent approach integrates deep periodic workload prediction models with meta-learning frameworks, guiding optimal scaling actions for efficiency and adaptability [45]. This highlights the significance of predictive analytics in anticipating demand fluctuations. Incorporating ecological principles into autoscaling, as discussed by Chen et al., enhances system stability and sustainability, demonstrating the potential of interdisciplinary strategies to enhance robustness [48]. The Diversity-Aware Autoscaling Algorithm by Akhtarian adjusts replica counts based on reliability scores, optimizing both performance and reliability [9].

Cohen et al. introduced the DAPP-ECC protocol, which reduces communication overhead through a distributed approach, enhancing cloud responsiveness [42]. The Token-Bucket Overbooking Method (TBOM) by Kesidis offers a framework for overbooking Lambda functions, allocating resources based on estimated demand and execution time distributions [47]. Additionally, continuous integration practices emphasized by Debayser facilitate rapid deployment and scaling of microservices, ensuring swift adaptation to changing demands [6].


Cloud Clients
Web browser, mobile app, thin client, terminal emulator, ...
SaaS
CRM, Email, virtual desktop, communication, games, ...
PaaS
Execution runtime, database, web server, development tools, ...
IaaS
Virtual machines, servers, storage, load balancers, network, ...
Application
Platform
Infra-structure

(a) Cloud Clients[43]



(b) Performance of a Distributed System with Different Initial Replicas and Services[49]

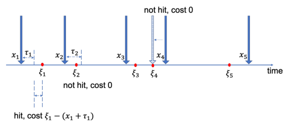


Fig. 1. The scaling process. Red dots denote query arrivals and blue arrows denote instance creation.

Algorithm 1 Scaling dynamics

```
   Given the instance creation times $x_1, x_2, \ldots$,
1: for $i = 1, 2, \ldots$ -th query do
2:     if $x_i + \tau_i \le \xi_i$ then
3:         the query is processed immediately
4:     else if $x_i \le \xi_i < x_i + \tau_i$ then
5:         the query waits until $x_i + \tau_i$ and then processing starts
6:     else
7:         an instance is created to process the query
8:     end if
9: end for
```

(c) The Scaling Process[44]

Figure 4: Examples of Autoscaling Strategies and Techniques

As illustrated in Figure 4, autoscaling is vital for optimal resource allocation in cloud computing. The first subfigure, "Cloud Clients," depicts the interaction between various clients and cloud services, highlighting the need for adaptable scaling solutions. The second subfigure shows the performance of distributed systems with varying initial replicas, emphasizing strategic planning's impact on efficiency. Finally, "The Scaling Process" illustrates how systems dynamically respond to query arrivals, maintaining performance and resource efficiency. This comprehensive view underscores the complexities involved in implementing effective autoscaling strategies.

4.2 Applications in Microservices

Method Name	**Resource Optimization**	**Predictive Techniques**	**Deployment Strategies**
COLA[46]	Constrained Optimization Problem	Bandit Algorithm	Centralized Approach Autoscaling
MMPA[45]	Optimal VM Allocations	Deep Workload Prediction	Model-based RL
MS2M[10]	Asynchronous Messaging	-	Service Placement Optimization
CPMS[35]	Dynamic Reconfiguration	-	Service Placement Optimization
DAA[9]	Autoscaling Algorithm	Predictive Scaling Techniques	Balanced Deployment Strategy
TBOM[47]	Dynamic Scheduling	Statistical Models	Service Placement Optimization

Table 2: This table provides a comparative analysis of various autoscaling methods applied in microservices architecture. It outlines the distinct approaches each method employs in resource optimization, predictive techniques, and deployment strategies. The table highlights the diversity of solutions and innovations aimed at enhancing performance and resource utilization efficiency in dynamic microservice environments.

In microservices architecture, autoscaling is essential for optimizing resource usage, enabling dynamic adjustments to workloads while maintaining performance and cost efficiency. Techniques like predictive analytics and AI-driven frameworks enhance system adaptability. The COLA framework exemplifies effective compute resource allocation across microservices, maintaining performance levels and optimizing resource usage [46].

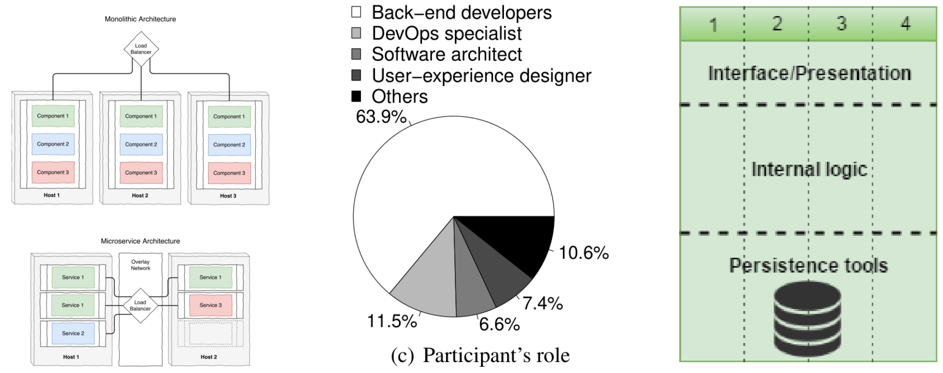
Existing solutions often rely on reactive strategies that fail to address the dynamic nature of microservices, resulting in inefficient resource utilization [50]. Advanced methods like Meta Model-based Predictive Autoscaling (MMPA) utilize deep learning for accurate workload predictions and meta-learning for efficient task adaptation [45].

Experiments using diverse load profiles demonstrate autoscaling's effectiveness in maintaining service performance under varying conditions [16]. The MS2M approach minimizes service downtime by allowing original instances to continue processing requests during migration, optimizing resource usage [10].

In manufacturing, defining processes as compositions of cyber-physical microservices optimizes resource usage, illustrating microservices' versatility in diverse applications [35]. The integration of software diversity in autoscaling, monitored through metrics such as Pod restarts and memory consumption, enhances resource allocation efficiency [9].

AI-generated, for reference only.

Formulating the service placement problem as Quadratic Semi-Relaxed Fractional Programming (QSRFP) and employing greedy algorithms optimize service deployment based on current states, showcasing effective management of microservice environments [13]. Furthermore, dynamic scheduling of Lambda functions through resource overbooking based on statistical demand models optimizes resource usage, demonstrating innovative scheduling methods' potential [47].



(a) Microservices Architecture vs Monolithic Architecture[51]

(b) Participants' Roles in a Software Development Project[52]

(c) Architecture Layers in Software Development[53]

Figure 5: Examples of Applications in Microservices

As shown in Figure 6, autoscaling significantly enhances system efficiency and resilience within microservices. The figure illustrates the hierarchical structure of applications in microservices, focusing on autoscaling techniques, resource optimization strategies, and their integration across various industries. The first image contrasts monolithic and microservices architectures, emphasizing the latter's ability to distribute services across hosts for improved fault tolerance and scalability. The second image highlights the collaborative nature of microservices development, involving diverse roles such as back-end developers and DevOps specialists. The third image illustrates the layered architecture of software development, reinforcing the principle of separation of concerns supported by microservices. Collectively, these visuals underscore autoscaling's transformative impact in cloud computing, enabling dynamic resource allocation and improved reliability. Additionally, Table 2 presents a comparative overview of key autoscaling methods in microservices, emphasizing their resource optimization, predictive techniques, and deployment strategies.

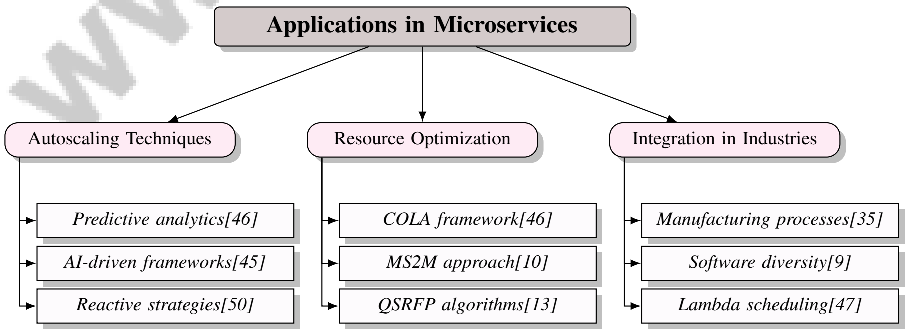


Figure 6: This figure illustrates the hierarchical structure of applications in microservices, focusing on autoscaling techniques, resource optimization strategies, and their integration in various industries. The autoscaling techniques include predictive analytics and AI-driven frameworks, while resource optimization features the COLA framework and MS2M approach. Integration in industries highlights manufacturing processes and software diversity.

AI-generated, for reference only.

Feature	Deep Periodic Workload Prediction	Ecological Principles Integration	Diversity-Aware Autoscaling Algorithm
Scaling Approach	Meta-learning Framework	Interdisciplinary Strategies	Reliability-based Adjustment
Resource Management	Optimal Scaling Actions	System Stability	Performance Optimization
Predictive Techniques	Deep Learning Models	Not Specified	Reliability Scores

Table 3: This table provides a comparative analysis of three distinct autoscaling strategies in cloud computing, focusing on their scaling approaches, resource management capabilities, and predictive techniques. The comparison highlights the integration of meta-learning frameworks, interdisciplinary strategies, and reliability-based adjustments, showcasing the diverse methodologies employed for optimizing resource management and performance in cloud environments.

5 Container Orchestration

5.1 Introduction to Container Orchestration

Container orchestration is vital in cloud computing, automating the management, deployment, and scaling of containerized applications to enhance operational efficiency in dynamic, distributed cloud-native systems. It optimizes resource utilization and ensures elasticity for scalable, fault-tolerant architectures [54]. Platforms like Kubernetes and Docker are crucial for orchestrating these applications, enabling seamless integration across diverse environments. Orchestration is particularly advantageous for deploying complex applications comprising microservices and Virtual Network Functions (VNFs), ensuring cohesive operation within cloud infrastructures [54]. Tools such as KubeNow demonstrate the adaptability of container orchestration by enhancing the creation of flexible research environments [19]. The Automated Microservices Extraction (AME) method exemplifies orchestration's role in modernizing legacy systems by efficiently extracting microservices from monolithic applications [27].

In hybrid cloud scenarios, orchestration is essential for managing serverless batch-processing pipelines, dynamically allocating resources to meet varying demands [55]. Distributed protocols like DAPP-ECC enhance resilience and performance in orchestrated systems [42]. Additionally, tools like Timed SmartDeployer optimize configurations for microservice applications by integrating timed architectural modeling with deployment orchestration [14]. Security is another critical consideration in container orchestration, as architectures like CAPODAZ enhance security controls in Cloud-IoT ecosystems by integrating microservices [56]. Performance antipattern detection tools, such as PADRE, improve the performance and security of cloud applications when used with orchestration platforms like Docker and Spring Boot [57].

5.2 Kubernetes as a Leading Orchestration Platform

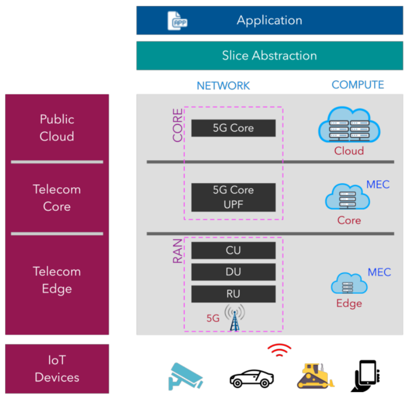
Kubernetes is a premier platform for orchestrating containerized applications, particularly for microservices deployment and management, due to its robust architecture that enhances scalability and maintainability [58]. It supports customizable autoscaling solutions, such as the Proactive Prediction-based Autoscaling (PPA) framework, which is crucial for maintaining performance and cost-efficiency amidst fluctuating workload demands [59]. Kubernetes excels in resource management through advanced orchestration techniques, enhancing resource allocation for microservices [60]. Its flexibility in integrating various autoscaling methods underscores its adaptability in diverse computing scenarios [61].

Kubernetes optimizes deployment processes through methods like the Cost-Efficient Container Resource Management (CCRM), employing a plugin-scheduler for efficient deployment and scaling in public cloud environments [62]. Its compatibility with containerization technologies like Docker is demonstrated in projects managing polyglot microservice applications, highlighting its versatility [63]. The FIRM framework, which adjusts resource allocation on Kubernetes clusters, exemplifies its relevance in fine-grained resource management [64]. In scientific development, Kubernetes is vital for managing microservices, streamlining processes through containerization technologies and orchestration tools [6]. Its comprehensive features, including automated deployment and scaling, establish Kubernetes as a leading orchestration platform in cloud computing.

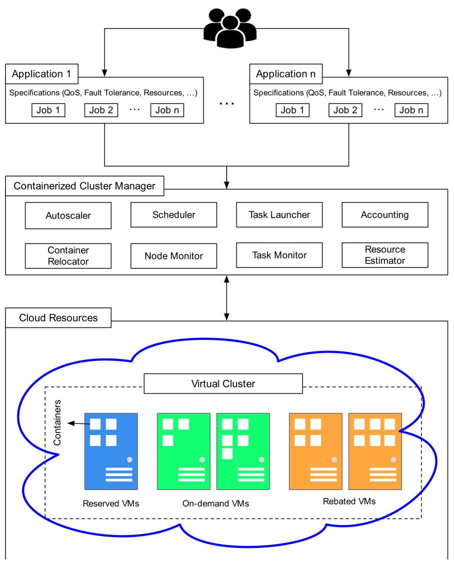
5.3 Advanced Orchestration Techniques

Advanced orchestration techniques are critical for optimizing microservices management, enhancing deployment, scaling, and operational efficiency. These techniques leverage innovative methodologies and tools to tackle the complexities of microservices architectures. The integration of machine learning and AI-driven frameworks into container orchestration facilitates predictive analytics and automated decision-making, improving resource allocation and system adaptability [11]. Decentralized orchestration approaches, like the DAPP-ECC protocol, minimize communication overhead and enhance scalability in microservices systems [42]. Hybrid orchestration models that combine centralized and decentralized techniques provide a flexible framework for managing microservices, allowing dynamic service reconfiguration to adapt to changing workloads [14].

Advanced scheduling algorithms, such as those used in the Service Placement Problem in Microservice Systems (SPPMS), optimize microservice deployment based on complex dependencies, ensuring stable Quality of Service (QoS) in response to user demands [13]. Integrating performance antipattern detection tools like PADRE with container orchestration platforms enhances application performance and security by identifying inefficiencies and vulnerabilities [57]. These techniques facilitate the transformation of monolithic enterprise systems into microservices architectures, addressing the complexities of traditional applications and enhancing performance in cloud-native environments [65, 17].



(a) 5G Network Architecture with Slice Abstraction[24]



(b) Containerized Cloud Management System[60]

Figure 7: Examples of Advanced Orchestration Techniques

As shown in Figure 7, advanced orchestration techniques are pivotal for enhancing the efficiency and scalability of modern network and cloud infrastructures. The figures illustrate two sophisticated orchestration techniques: "5G Network Architecture with Slice Abstraction" and "Containerized Cloud Management System." The first example emphasizes slice abstraction in 5G network architecture, delineating the network into three components: Network, Compute, and Application, facilitating seamless interaction among them. The second example showcases a containerized cloud management system designed to optimize application management and job execution, featuring a cluster manager with components like an autoscaler, scheduler, task launcher, and resource estimator, demonstrating a comprehensive approach to resource orchestration in cloud environments [24, 60].

5.4 Security and Compliance in Orchestration

Security and compliance are paramount in orchestrating containerized applications, ensuring the integrity, confidentiality, and availability of cloud services. Platforms like Kubernetes provide robust mechanisms for managing security policies and compliance requirements, essential for maintaining trust in microservices architectures [56]. A primary challenge in securing containerized environments is managing authentication and authorization processes. The CAPODAZ architecture exemplifies a containerized authorization framework that enhances security through policy-driven access controls within Cloud-IoT ecosystems [56], mitigating unauthorized access risks.

Network security concerns, such as securing communication between microservices, must also be addressed. Implementing service meshes and network policies within Kubernetes enhances security by providing granular control over service interactions and traffic management. This approach isolates individual microservices and allows for the dynamic enforcement of security policies tailored to the sensitive nature of IoT data in multi-domain environments [66, 67]. Compliance with regulatory standards like GDPR and HIPAA is another critical aspect of container orchestration. Orchestration platforms must support compliance controls to ensure data protection and privacy. The ability to audit and monitor activities within a containerized environment is crucial for maintaining transparency and accountability [68, 69, 70, 71].

Integrating performance antipattern detection tools, such as PADRE, with container orchestration platforms enhances security by identifying vulnerabilities and inefficiencies in the system [57]. Addressing security and compliance in orchestration involves comprehensive strategies encompassing access control, network security, and regulatory compliance. By utilizing sophisticated security frameworks and tools, container orchestration platforms can enhance application and data protection, ensuring microservices architectures maintain robust security and compliance in rapidly evolving cloud environments [72, 26, 62].

6 Scalability and Scheduling

6.1 Conceptual Framework for Scalability and Scheduling

The framework for scalability and scheduling in cloud environments optimizes resource allocation and performance in microservices architectures. It addresses resource dependencies, which can cause contention and bottlenecks, necessitating effective scalability strategies [73]. Key components include service discovery, load balancing, and configuration management, aiding migration to cloud-native architectures [65]. Dynamic resource allocation techniques enhance system performance and cost-efficiency, emphasizing resource utilization, startup times, and scalability [63]. Intelligent orchestration aligns resources with real-time demands, minimizing costs while maximizing efficiency. The framework evaluates trade-offs between microservices and serverless architectures, especially in IoT applications, strategically placing containers to optimize interaction affinity and network latency, enhancing scalability and performance [74]. Collecting and analyzing metrics is essential for informed decisions, avoiding unnecessary migrations. Future research should explore resource heterogeneity's effects on autoscaling and evolving workload patterns on allocation strategies. Integrating dynamic resource allocation, strategic container placement, and intelligent orchestration enables efficient cloud-based application deployment, enhancing scalability and optimizing resource utilization for diverse workloads [75, 19, 60, 62].

6.2 Analytical Performance Models and Predictive Algorithms

Analytical performance models and predictive algorithms optimize scalability and efficiency in microservices architectures by utilizing metrics such as response time, throughput, and error rate to assess system performance [76]. These models integrate CPU and RAM usage, facilitating comprehensive evaluations in simulated environments to identify bottlenecks and optimization opportunities [40]. Predictive algorithms are crucial for workload prediction, vital for effective auto-scaling and resource allocation [50]. Advanced statistical techniques forecast workload demands, enabling dynamic resource adjustments.

Figure 8 illustrates the key components of analytical performance models and predictive algorithms in microservices, focusing on performance metrics, predictive algorithms for workload and resource management, and the role of performance feedback autoscaling. For instance, integrating predictive analytics within OpenStack enhances scaling policies, ensuring efficient resource management. The proactive auto-scaling model by Merkouche et al. leverages microservices interdependencies to forecast resource needs, optimizing allocation and enhancing scalability [20, 17, 50]. Performance Feedback Autoscaling (PFA) uses historical performance data for resource management, improving predictive models and autoscaling strategies [5, 77, 50]. Predictive algorithms developed by Flunkert et al. use advanced statistical techniques for workload forecasts, facilitating informed scaling decisions [78, 79, 50, 77, 45]. Tailored predictive models for serverless platforms, as noted by Mahmoudi et al., are needed to address technical debt and sustain high system performance [80, 20, 81].

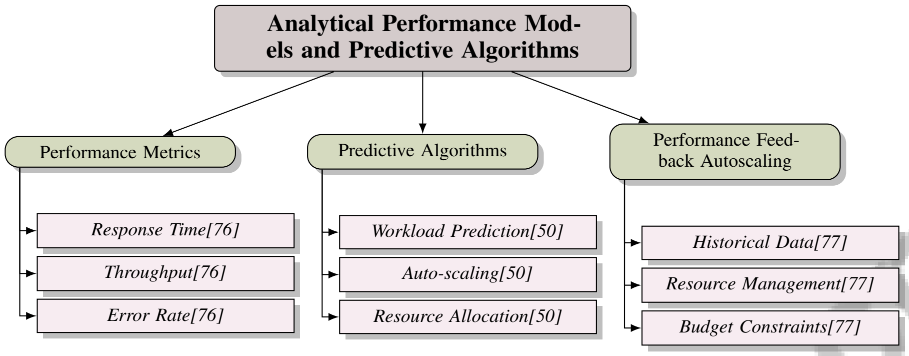


Figure 8: This figure illustrates the key components of analytical performance models and predictive algorithms in microservices, focusing on performance metrics, predictive algorithms for workload and resource management, and the role of performance feedback autoscaling.

6.3 Dynamic Resource Allocation Strategies

Dynamic resource allocation optimizes computational resource utilization in cloud environments, adapting to fluctuating demands. The transition to microservice architecture enhances scalability and maintainability for mission-critical systems, as shown in the FX Core case study [82]. The Pattern-based Microservice Workflow Scheduling to Containers (PMWSC) framework identifies workflow patterns to inform task scheduling and resource allocation, reducing computation time and enhancing scheduling accuracy [83]. The token-bucket mechanism, discussed by Kesidis, allocates resources based on demand patterns, improving efficiency and service availability [47]. Tailoring resource allocation strategies to application architectures is essential, as noted by Seedat et al. [29]. Effective dynamic resource allocation considers each microservice's demands to ensure optimal distribution, maintaining performance and reliability.

6.4 Metrics and Benchmarks for Scalability

Benchmark	Size	Domain	Task Format	Metric
MPE-DP[8]	15,000	Microservices	Performance Evaluation	Spearman's rank correlation coefficient, Mean Absolute Error
MSDB[84]	20	Microservices	Dependency Analysis	SLOCcount, MicroDep-Graph
DSB[85]	1,000,000	Cloud Computing	End-to-end Service Evaluation	Tail Latency, Throughput
MSB[86]	9	Microservices	Performance Benchmarking	Latency, Cost
MEBE[87]	1,000	Smart Public Safety	Performance Evaluation	FLOPS, MB/s
MF-Benchmark[88]	10,000	Urban Mobility	Performance Evaluation	End-to-End Latency, Resource Consumption
ML-AutoScale[89]	1,000	Machine Learning Inference	Performance Evaluation	Inference Time, Resource Utilization
ASB[90]	3,551,000	Engineering	Workflow Scheduling	Makespan, Normalized Schedule Length

Table 4: This table presents a comprehensive overview of various benchmarks utilized in evaluating the scalability and performance of cloud systems, particularly within microservices architectures. It details the size, domain, task format, and metrics associated with each benchmark, providing a crucial resource for understanding performance evaluation methodologies across different domains.

Implementing key metrics and benchmarks is essential for assessing cloud systems' scalability, particularly in microservices architectures. Table 4 provides a detailed summary of benchmarks employed in the assessment of scalability metrics for cloud systems, highlighting their relevance in microservices and other domains. Metrics like response time, throughput, and reliability evaluate performance aspects, addressing bottlenecks and optimization techniques [91, 76]. Understanding these metrics optimizes resource utilization and ensures application reliability. CPU utilization provides insights into processing capacity under varying loads. The dataset from Meijer et al. includes performance data such as service latency and resource usage, essential for evaluating architectural design patterns' impact on scalability [8]. Amaral et al. emphasize evaluating CPU execution times and network throughput/latency under various configurations [92]. Network latency and throughput affect microservices' responsiveness and efficiency, as highlighted by Shadija et al. [21]. Resource utilization insights help optimize management strategies, enhancing scalability [93]. Deployment speed and scalability are crucial in cloud environments, where rapid provisioning is often required. Capuccini et al. demonstrate evaluating KubeNow's performance concerning deployment speed and scalability [19]. AI techniques improve performance efficiency and reliability, as noted by Moreschini et al., despite a lack of focus on quality attributes like compatibility and usability [11]. A significant limitation is the lack of consensus on sustainability metrics, hindering comprehensive assessments [94].

6.5 Innovative Scheduling Techniques

Innovative scheduling techniques optimize resource allocation and enhance microservices architectures' performance in cloud computing. These techniques manage distributed resources and workloads, ensuring applications adapt to dynamic demands. AI-driven frameworks use predictive analytics to anticipate workload variations, adjusting scheduling decisions accordingly [11]. Decentralized scheduling protocols, like the DAPP-ECC protocol, minimize communication overhead and enhance scalability [42]. The PMWSC framework uses pattern recognition to optimize scheduling decisions, reducing computation time and enhancing accuracy [83]. The token-bucket mechanism allocates resources based on demand patterns, enhancing performance and reliability [47]. Integrating performance feedback mechanisms, such as Performance Feedback Autoscaling (PFA), into scheduling processes refines decisions using historical data [50]. AI-driven resource management reduces costs by 30-40

7 Challenges and Future Directions

7.1 Technical Challenges and Considerations

Implementing autoscaling and orchestration in microservices architectures involves numerous technical challenges that affect system performance, resource management, and reliability. The complexity of distributed systems requires effective collaboration among teams while maintaining the autonomy of individual microservices [4]. This complexity is compounded by the dispersed nature of strategies across various publications, making it difficult for practitioners to find tailored solutions [2]. The NP-hard nature of resource allocation and scheduling optimization further complicates effective autoscaling strategies, as traditional local scaling techniques often lead to performance degradation [10]. Additionally, strict quotas on active Lambda invocations per tenant can hinder dynamic resource allocation, resulting in inefficient resource utilization [47].

Centralized orchestrators can create communication bottlenecks and single points of failure, affecting the scalability and resilience of autoscaling systems [42]. In dynamic environments with fluctuating workloads, achieving optimal performance is challenging, as existing methods may not adapt effectively [5]. Moreover, the lack of empirical validation in benchmarks limits the understanding of performance behaviors of design patterns in real-world scenarios [8]. Continuous integration and deployment practices may not be feasible for all scientific projects, restricting the applicability of certain methodologies [6]. Emphasizing quality assurance and systematic assessment of architectural refactoring techniques is crucial for informed decision-making during the transition to microservices architectures [41].

Advanced frameworks that automate performance assessments and visualize complex architectures are essential to address these challenges. By tackling the technical challenges associated with microservices architecture, organizations can significantly enhance system efficiency and reliability in dynamic cloud environments, thereby supporting software development agility. Insights from industry practitioners emphasize the importance of robust infrastructural support early in development and effective management of common code and product variants, facilitating smoother transitions from monolithic systems and promoting best practices for successful microservice implementations [20, 17].

7.2 Challenges and Opportunities in Container Orchestration

Container orchestration presents various challenges and opportunities crucial for the efficiency and scalability of cloud-native applications. A primary challenge is optimizing resource allocation across diverse microservices, which often have varying demands and dependencies. Traditional resource management methods may fall short, leading to inefficiencies and heightened operational costs. The COLA framework exemplifies an advanced approach to resource optimization by considering the entire application's context, resulting in lower costs and improved latency performance compared to conventional strategies [46].

Balancing scalability with operational complexity is another significant challenge, particularly as the dynamic nature of cloud workloads increases orchestration difficulties. Advanced resource scheduling and autoscaling algorithms capable of adapting to varying application demands are necessary to optimize resource utilization and minimize costs [68, 60, 62]. The orchestration system must manage an increasing number of containers, each with unique dependencies and resource requirements, complicating the maintenance of consistent performance and reliability. However, this complexity presents opportunities to leverage advanced orchestration tools and frameworks that automate these processes, enhancing system efficiency and alleviating developer burdens.

Security and compliance are critical concerns in container orchestration, particularly in shared environments where robust security measures are essential to protect sensitive data and maintain user trust. The complexity and interdependencies of microservices can heighten vulnerabilities, making effective security protocols vital for data integrity and overall application performance [69, 75]. Opportunities exist to develop sophisticated security frameworks that seamlessly integrate with orchestration platforms, enhancing protection and compliance capabilities.

The dynamic nature of cloud environments also presents both challenges and opportunities for container orchestration. The ability to dynamically scale resources in response to changing workloads is a significant advantage. Effective resource management in microservices architecture requires advanced algorithms capable of accurately forecasting workload demands and dynamically adjusting resource allocations. This complexity arises from the intricate interdependencies among microservices and the need to comply with Service Level Agreements (SLAs) while optimizing resource utilization. Employing models like the Temporal Fusion Transformer can enhance prediction accuracy and facilitate proactive scaling, ensuring each microservice is provisioned with appropriate resources to meet varying demand levels and maintain high Quality of Service (QoS) [23, 5, 19, 50]. Integrating AI and machine learning techniques into orchestration platforms holds significant potential for enhancing predictive capabilities and improving resource management efficiency.

7.3 Challenges and Trade-offs in Resource Allocation

Resource allocation in microservices architectures involves a complex interplay of challenges and trade-offs that significantly impact system performance and efficiency. Balancing resource utilization with system performance is a primary challenge, particularly in dynamic cloud environments where unpredictable workloads can fluctuate rapidly [47]. The traditional approach of overprovisioning resources to ensure performance reliability often leads to inefficient resource utilization and increased operational costs [5].

Trade-offs in resource allocation are further complicated by the need to maintain Quality of Service (QoS) while optimizing costs. The distributed nature of microservices necessitates careful coordination of resources to prevent bottlenecks and ensure seamless operation. This requires sophisticated scheduling algorithms capable of dynamically adjusting resource distribution based on real-time demand, as demonstrated by frameworks like the Pattern-based Microservice Workflow Scheduling to Containers (PMWSC) [83].

Another significant trade-off involves choosing between centralized and decentralized resource management strategies. While centralized approaches offer straightforward control and oversight, they often encounter scalability issues and single points of failure [42]. Conversely, decentralized strategies, such as those enabled by the DAPP-ECC protocol, provide enhanced scalability and fault tolerance but may introduce complexities in coordination and decision-making processes [42].

Integrating AI-driven frameworks into resource allocation strategies presents both opportunities and challenges. AI techniques can improve predictive capabilities, allowing for more accurate workload forecasts and resource adjustments [11]. However, reliance on AI models introduces additional complexity regarding model training, maintenance, and the necessity for robust data management practices to ensure accuracy and reliability.

7.4 Future Directions and Research Opportunities

The future trajectory of microservices and cloud computing is set for significant advancements, with several promising research avenues aimed at enhancing system resilience, scalability, and efficiency. A primary focus area is refining decomposition criteria and exploring dynamic analysis methods to improve the accuracy of microservices migration from monolithic architectures. Developing a hybrid approach that incorporates both static and dynamic analysis could significantly enhance service decomposition precision [27].

Further research is needed to establish robust frameworks for deployment and communication that can adapt to the evolving needs of microservice architectures. This includes integrating machine learning techniques to predict service requests and enhance migration process efficiency [42]. Exploring specific ecological principles that can be effectively translated into cloud autoscaling mechanisms also represents a promising area of study [48].

The future of microservices also involves expanding benchmarks to include additional design patterns and examining their impact on performance metrics such as scalability and energy consumption, indicating potential research opportunities [8]. Moreover, refining migration methodologies based on practitioner feedback and expanding tools' functionality will enhance usability and effectiveness [41].

Extending diversity-aware autoscaling approaches to a broader range of microservices and refining reliability assessment metrics are critical for optimizing resource management strategies. Future research should also concentrate on enhancing statistical models for demand prediction and integrating the token-bucket overbooking method with other cloud resource management strategies [47].

Additionally, simplifying and accelerating debugging processes through orchestration platforms like Temporal suggests a promising direction for future research and development, particularly in enhancing microservices orchestration [3]. Long-term evaluations of identified anti-patterns and controlled experiments to study the effectiveness of proposed solutions are necessary for improving microservices architecture [7].

Evaluating collaborative model-driven software engineering (CMDSE) approaches and adapting modeling techniques for distributed environments will further enhance microservices development [4]. Additionally, refining continuous integration (CI) processes for data and schema management and investigating the scalability of microservices in various scientific domains will contribute to advancing microservices in specialized fields [6].

8 Conclusion

Microservices architecture is pivotal in advancing modern software development by offering enhanced scalability, availability, and fault tolerance, which are essential for contemporary applications. The integration of autoscaling and container orchestration into cloud computing solutions significantly boosts efficiency and cost-effectiveness. The utilization of frameworks that optimize resource allocation ensures that microservice applications maintain high performance while minimizing server usage. Decision models provide practitioners with strategic guidance for selecting appropriate patterns and strategies, thereby refining software architecture quality.

Incorporating robust security mechanisms within microservices architectures is crucial to address vulnerabilities and ensure the effectiveness of cloud computing solutions. The selection of cloud services plays a critical role in determining both performance and cost, underscoring the importance of informed decision-making in cloud deployments. The research supports the effectiveness of decision models in assisting practitioners to enhance software architecture by selecting suitable patterns and strategies.

The study highlights the substantial impact of microservice granularity on application performance, particularly in cloud environments. Containerized microservice architectures demonstrate superior resource utilization, scalability, and responsiveness compared to monolithic architectures. By leveraging advanced techniques and frameworks, organizations can achieve optimal resource management, reduce operational costs, and enhance the scalability and reliability of their cloud-native applications.

References

[1] Masters thesis.

[2] Muhammad Waseem, Peng Liang, Aakash Ahmad, Mojtaba Shahin, Arif Ali Khan, and Gastón Márquez. Decision models for selecting patterns and strategies in microservices systems and their evaluation by practitioners, 2022.

[3] Anas Nadeem and Muhammad Zubair Malik. A case for microservices orchestration using workflow engines, 2022.

[4] Jonas Sorgalla, Florian Rademacher, Sabine Sachweh, and Albert Zündorf. On collaborative model-driven development of microservices, 2018.

[5] Michael Alan Chang, Aurojit Panda, Hantao Wang, Yuancheng Tsai, Rahul Balakrishnan, and Scott Shenker. Autotune: Improving end-to-end performance and resource efficiency for microservice applications, 2021.

[6] Maximillien de Bayser, Vinicius Segura, Leonardo Guerreiro Azevedo, Leonardo P. Tizzei, Raphael Melo Thiago, Elton Soares, and Renato Cerqueira. Devops and microservices in scientific system development, 2021.

[7] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. Microservices anti patterns: A taxonomy, 2019.

[8] Willem Meijer, Catia Trubiani, and Aldeida Aleti. Experimental evaluation of architectural software performance design patterns in microservices, 2024.

[9] Nazanin Akhtarian, Hamzeh Khazaei, and Marin Litoiu. Employing software diversity in cloud microservices to engineer reliable and performant systems, 2024.

[10] Hai Dinh-Tuan and Felix Beierle. Ms2m: A message-based approach for live stateful microservices migration, 2022.

[11] Sergio Moreschini, Shahrzad Pour, Ivan Lanese, Daniel Balouek-Thomert, Justus Bogner, Xiaozhou Li, Fabiano Pecorelli, Jacopo Soldani, Eddy Truyen, and Davide Taibi. Ai techniques in the microservices life-cycle: A systematic mapping study, 2025.

[12] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. A kubernetes controller for managing the availability of elastic microservice based stateful applications, 2020.

[13] Xiang He, Zhiying Tu, Markus Wagner, Xiaofei Xu, and Zhongjie Wang. Online deployment algorithms for microservice systems with complex dependencies, 2021.

[14] Lorenzo Bacchiani, Mario Bravetti, Saverio Giallorenzo, Jacopo Mauro, Iacopo Talevi, and Gianluigi Zavattaro. Microservice dynamic architecture-level deployment orchestration (extended version), 2021.

[15] Işıl Karabey Aksakalli, Turgay Çelik, Ahmet Burak Can, and Bedir Tekinerdoğan. Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software*, 180:111014, 2021.

[16] Floriment Klinaku, Martina Rapp, Jörg Henss, and Stephan Rhode. Beauty and the beast: A case study on performance prototyping of data-intensive containerized cloud applications, 2022.

[17] Alessandra Levcovitz, Ricardo Terra, and Marco Tulio Valente. Towards a technique for extracting microservices from monolithic enterprise systems, 2016.

[18] Manuel Mazzara, Ruslan Mustafin, Larisa Safina, and Ivan Lanese. Towards microservices and beyond: An incoming paradigm shift in distributed computing, 2016.

[19] Marco Capuccini, Anders Larsson, Matteo Carone, Jon Ander Novella, Noureddin Sadawi, Jianliang Gao, Salman Toor, and Ola Spjuth. On-demand virtual research environments using microservices, 2019.

[20] Yingying Wang, Harshavardhan Kadiyala, and Julia Rubin. Promises and challenges of microservices: an exploratory study. *Empirical Software Engineering*, 26(4):63, 2021.

[21] Dharmendra Shadija, Mo Rezai, and Richard Hill. Microservices: Granularity vs. performance, 2017.

[22] Lucas Fernando Souza de Castro and Sandro Rigo. Relating edge computing and microservices by means of architecture approaches and features, orchestration, choreography, and offloading: A systematic literature review, 2023.

[23] Amadou Ba, Pavithra Harsha, and Chitra Subramanian. Leveraging interpretability in the transformer to automate the proactive scaling of cloud resources, 2024.

[24] Anousheh Gholami, Kunal Rao, Wang-Pin Hsiung, Oliver Po, Murugan Sankaradas, and Srimat Chakradhar. Roma: Resource orchestration for microservices-based 5g applications, 2022.

[25] Linfeng Wen, Minxian Xu, Sukhpal Singh Gill, Muhammad Hafizhuddin Hilman, Satish Narayana Srirama, Kejiang Ye, and Chengzhong Xu. Statuscale: Status-aware and elastic scaling strategy for microservice applications, 2024.

[26] Marcela S. Melara and Mic Bowman. Enabling security-oriented orchestration of microservices, 2021.

[27] Meryam Chaieb and Mohamed Aymen Saied. Automate migration to microservices architecture using machine learning techniques, 2023.

[28] Bernardo Andrade, Samuel Santos, and António Rito Silva. From monolith to microservices: Static and dynamic analysis comparison, 2022.

[29] Momil Seedat, Qaisar Abbas, and Nadeem Ahmad. Systematic mapping of monolithic applications to microservices architecture, 2023.

[30] Florian Rademacher, Sabine Sachweh, and Albert Zündorf. Analysis of service-oriented modeling approaches for viewpoint-specific model-driven development of microservice architecture, 2018.

[31] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. Architectural patterns for microservices: a systematic mapping study. In *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018*. SciTePress, 2018.

[32] Alexander Krause, Christian Zirkelbach, Wilhelm Hasselbring, Stephan Lenga, and Dan Kröger. Microservice decomposition via static and dynamic analysis of the monolith, 2020.

[33] Muhammad Waseem, Peng Liang, Gastón Márquez, Mojtaba Shahin, Arif Ali Khan, and Aakash Ahmad. A decision model for selecting patterns and strategies to decompose applications into microservices, 2021.

[34] Meryam chaieb and Mohamed Aymen Saied. Migration to microservices: A comparative study of decomposition strategies and analysis metrics, 2024.

[35] Kleanthis Thramboulidis, Danai C. Vachtsevanou, and Alexandros Solanos. Cyber-physical microservices: An iot-based framework for manufacturing systems, 2018.

[36] Justus Bogner, Jonas Fritzsch, Stefan Wagner, and Alfred Zimmermann. Assuring the evolvability of microservices: Insights into industry practices and challenges, 2019.

[37] Tommaso Praturlon. Intelligent autoscaling in kubernetes: the impact of container performance indicators in model-free drl methods, 2023.

[38] Hai Dinh-Tuan, Katerina Katsarou, and Patrick Herbke. Optimizing microservices with hyperparameter optimization, 2022.

[39] Yu Gan and Christina Delimitrou. The architectural implications of microservices in the cloud, 2018.

[40] Emilien Wansart, Maxime Goffart, Justin Iurman, and Benoit Donnet. Mstg: A flexible and scalable microservices infrastructure generator, 2024.

[41] Jonas Fritzsch, Justus Bogner, Markus Haug, Stefan Wagner, and Alfred Zimmermann. Towards an architecture-centric methodology for migrating to microservices, 2022.

[42] Itamar Cohen, Paolo Giaccone, and Carla Fabiana Chiasserini. Asynchronous distributed protocol for service provisioning in the edge-cloud continuum, 2023.

[43] Tao Chen. Self-aware and self-adaptive autoscaling for cloud based services, 2016.

[44] Huajie Qian, Qingsong Wen, Liang Sun, Jing Gu, Qiulin Niu, and Zhimin Tang. Robustscaler: Qos-aware autoscaling for complex workloads, 2022.

[45] Siqiao Xue, Chao Qu, Xiaoming Shi, Cong Liao, Shiyi Zhu, Xiaoyu Tan, Lintao Ma, Shiyu Wang, Shijun Wang, Yun Hu, Lei Lei, Yangfei Zheng, Jianguo Li, and James Zhang. A meta reinforcement learning approach for predictive autoscaling in the cloud, 2022.

[46] Vighnesh Sachidananda and Anirudh Sivaraman. Collective autoscaling for cloud microservices, 2022.

[47] George Kesidis. Overbooking microservices in the cloud, 2019.

[48] Tao Chen and Rami Bahsoon. Bridging ecology and cloud: Transposing ecological prespective to enable better cloud autoscaling, 2016.

[49] Harold Ship, Evgeny Shindin, Chen Wang, Diana Arroyo, and Asser Tantawi. Optimizing simultaneous autoscaling for serverless cloud computing, 2023.

[50] Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. The power of prediction: microservice auto scaling via workload learning. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 355–369, 2022.

[51] Nicola Dragoni, Ivan Lanese, Stephan Thordal Larsen, Manuel Mazzara, Ruslan Mustafin, and Larisa Safina. Microservices: How to make your application scale, 2017.

[52] Markos Viggiato, Ricardo Terra, Henrique Rocha, Marco Tulio Valente, and Eduardo Figueiredo. Microservices in practice: A survey study, 2018.

[53] Manuel Mazzara, Antonio Bucchiarone, Nicola Dragoni, and Victor Rivera. Size matters: Microservices research and applications, 2019.

[54] Pankaj Saha, Angel Beltre, Piotr Uminski, and Madhusudhan Govindaraju. Evaluation of docker containers for scientific workloads in the cloud, 2019.

[55] Aristotelis Peri, Michail Tsenos, and Vana Kalogeraki. Orchestrating the execution of serverless functions in hybrid clouds, 2024.

[56] Dimitrios Kallergis, Zacharenia Garofalaki, Georgios Katsikogiannis, and Christos Douligeris. Capodaz: A containerised authorisation and policy-driven architecture using microservices, 2020.

[57] Vittorio Cortellessa, Daniele Di Pompeo, Romina Eramo, and Michele Tucci. A model-driven approach for continuous performance engineering in microservice-based systems, 2023.

[58] Haoyu Zhang, Jianming Zhang, and Yaozong Wang. Design and implementation of an advanced planning and scheduling system based on microservices. In *3rd International Conference on Management Science and Software Engineering (ICMSSE 2023)*, pages 593–601. Atlantis Press, 2023.

[59] Li Ju, Prashant Singh, and Salman Toor. Proactive autoscaling for edge computing systems with kubernetes, 2021.

[60] Rajkumar Buyya, Maria A. Rodriguez, Adel Nadjaran Toosi, and Jaeman Park. Cost-efficient orchestration of containers in clouds: A vision, architectural elements, and future directions, 2018.

[61] Yuxuan Zhao and Alexandru Uta. Tiny autoscalers for tiny workloads: Dynamic cpu allocation for serverless functions, 2022.

[62] Maria A. Rodriguez and Rajkumar Buyya. Containers orchestration with cost-efficient autoscaling in cloud computing environments, 2018.

[63] Vamsi Krishna Yepuri, Venkata Kalyan Polamarasetty, Shivani Donthi, and Ajay Kumar Reddy Gondi. Containerization of a polyglot microservice application using docker and kubernetes, 2023.

[64] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. Firm: An intelligent fine-grained resource management framework for slo-oriented microservices, 2020.

[65] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Migrating to cloud-native architectures using microservices: An experience report, 2015.

[66] Samodha Pallewatta and Muhammad Ali Babar. Towards secure management of edge-cloud iot microservices using policy as code, 2024.

[67] Adam Rubak. Dimensioning microservices on kubernetes platforms using machine learning techniques, 2023.

[68] Zhiheng Zhong, Minxian Xu, Maria Alejandra Rodriguez, Chengzhong Xu, and Rajkumar Buyya. Machine learning-based orchestration of containers: A taxonomy and future directions, 2021.

[69] Prathamesh Muzumdar, Amol Bhosale, Ganga Prasad Basyal, and George Kurian. Navigating the docker ecosystem: A comprehensive taxonomy and survey, 2024.

[70] Mujahid Sultan. Linking stakeholders' viewpoint concerns and microservices-based architecture, 2020.

[71] Vikram Nitin, Shubhi Asthana, Baishakhi Ray, and Rahul Krishna. Cargo: Ai-guided dependency analysis for migrating monolithic applications to microservices architecture, 2022.

[72] Abdelhakim Hannousse and Salima Yahiouche. Securing microservices and microservice architectures: A systematic mapping study, 2020.

[73] Biman Barua and M. Shamim Kaiser. Ai-driven resource allocation framework for microservices in hybrid cloud platforms, 2024.

[74] Mohak Chadha, Victor Pacyna, Anshul Jindal, Jianfeng Gu, and Michael Gerndt. Migrating from microservices to serverless: An iot platform case study, 2022.

[75] Nane Kratzke. About microservices, containers and their underestimated impact on network performance, 2017.

[76] Ganesh Chowdary Desina. Evaluating the impact of cloud-based microservices architecture on application performance, 2023.

[77] Alexey Ilyushkin, André Bauer, Alessandro V. Papadopoulos, Ewa Deelman, and Alexandru Iosup. Performance-feedback autoscaling with budget constraints for cloud-based workloads of workflows, 2019.

[78] Valentin Flunkert, Quentin Rebjock, Joel Castellon, Laurent Callot, and Tim Januschowski. A simple and effective predictive resource scaling heuristic for large-scale cloud applications, 2020.

[79] Hanieh Alipour. *Model-Driven Machine Learning for Predictive Cloud Auto-scaling*. PhD thesis, Concordia University, 2019.

[80] Quintin Fettes, Avinash Karanth, Razvan Bunescu, Brandon Beckwith, and Sreenivas Subramoney. Reclaimer: A reinforcement learning approach to dynamic resource allocation for cloud microservices. *arXiv preprint arXiv:2304.07941*, 2023.

[81] Muhammad Waseem, Peng Liang, Aakash Ahmad, Arif Ali Khan, Mojtaba Shahin, Pekka Abrahamsson, Ali Rezaei Nasab, and Tommi Mikkonen. Understanding the issues, their causes and solutions in microservices systems: An empirical study, 2023.

[82] Nicola Dragoni, Schahram Dustdar, Stephan T. Larsen, and Manuel Mazzara. Microservices: Migration of a mission critical system, 2017.

[83] Wenzheng Li, Xiaoping Li, and Long Chen. Pattern learning for scheduling microservice workflow to cloud containers. *International Journal of Machine Learning and Cybernetics*, 15(9):3701–3714, 2024.

[84] Mohammad Imranur, Rahman, Sebastiano Panichella, and Davide Taibi. A curated dataset of microservices-based systems, 2019.

[85] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. An open-source benchmark suite for cloud and iot microservices, 2019.

[86] Juan Mera Menéndez, Jose Emilio Labra Gayo, Enrique Riesgo Canal, and Aitor Echevarría Fernández. A comparison between traditional and serverless technologies in a microservices setting, 2023.

[87] Qian Qu, Ronghua Xu, Seyed Yahya Nikouei, and Yu Chen. An experimental study on microservices based edge computing platforms, 2020.

[88] Hai Dinh-Tuan, Maria Mora-Martinez, Felix Beierle, and Sandro Rodriguez Garzon. Development frameworks for microservice-based applications: Evaluation and comparison, 2022.

[89] Christian Schroeder, Rene Boehm, and Alexander Lampe. Comparison of autoscaling frameworks for containerised machine-learning-applications in a local and cloud environment, 2024.

[90] Laurens Versluis, Mihai Neacşu, and Alexandru Iosup. Technical report: A trace-based performance study of autoscaling workloads of workflows in datacenters, 2017.

[91] Carlos M Aderaldo, Nabor C Mendonça, Claus Pahl, and Pooyan Jamshidi. Benchmark requirements for microservices architecture research. In *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*, pages 8–13. IEEE, 2017.

[92] Marcelo Amaral, Jordà Polo, David Carrera, Iqbal Mohomed, Merve Unuvar, and Malgorzata Steinder. Performance evaluation of microservices architectures using containers, 2015.

[93] Gerta Sheganaku. *Optimized auto scaling of elastic processes in the cloud using docker containers*. PhD thesis, Wien, 2017.

[94] Xingwen Xiao. Architectural tactics to improve the environmental sustainability of microservices: A rapid review, 2024.