Scientific Research Publishing

Journal of Software Engineering and Applications, 2025, 18(7), 286-302
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Architecting SuperApps: Microservices vs. Mini-Apps Container Models

—Technical Frameworks for Managing Scalability, Modularity, and Security in SuperApps

Zaki Ali Bayashot

NHC Innovation, Balady, Riyadh, KSA
Email: zakibayashot@gmail.com

How to cite this paper: Bayashot, Z.A. (2025) Architecting SuperApps: Microservices vs. Mini-Apps Container Models. *Journal of Software Engineering and Applications*, **18**, 286-302.
https://doi.org/10.4236/jsea.2025.187017

Received: June 19, 2025
Accepted: July 22, 2025
Published: July 25, 2025

Abstract

SuperApps are reshaping digital ecosystems by integrating multiple services into unified platforms, offering users seamless access to diverse functionalities such as messaging, payments, e-commerce, and utilities. This paper provides a comparative analysis of two dominant architectural paradigms underpinning SuperApps: microservices and mini-app container models. While microservices represent a mature, modular backend approach characterized by independent deployment and horizontal scalability, mini-apps offer a lightweight, front-end integration model designed for extensibility within a centralized application framework. The analysis is structured around three core dimensions critical to SuperApp performance: scalability, modularity, and security. Microservices facilitate fine-grained resource allocation, support independent scaling, and promote heterogeneous technology stacks, making them ideal for backend-heavy operations requiring continuous deployment. Conversely, mini-apps, embedded within the SuperApp environment, leverage centralized cloud infrastructure to achieve operational scalability and rapid feature deployment, though they introduce significant privacy concerns due to pervasive user data collection. Empirical insights from recent studies underscore the contrasting implications of each model. For instance, while microservices are well-suited for CI/CD pipelines and resilient fault isolation, mini-apps have been shown to leak sensitive user information through interaction histories, posing non-traditional security risks. This paper also explores emerging hybrid architectures that combine microservices for backend logic with mini-apps for modular front-end experiences, advocating for an integrated strategy that balances flexibility, scalability, and data privacy. Through this comparative framework, the study informs developers, architects, and policymakers on optimal design strategies for future SuperApp ecosystems. It

concludes by outlining directions for improving privacy protections in mini-apps, enhancing microservices orchestration, and aligning industry practices with regulatory standards.

Keywords

SuperApps Architecture, Microservices, Mini-Apps Container Model, Scalability and Modularity, Privacy and Security in Mobile Platforms

1. Introduction

SuperApps are redefining how digital services are accessed and consumed by consolidating multiple functionalities—such as messaging, payment systems, transportation, and e-commerce—into a single, unified mobile platform. Popular SuperApps like WeChat, Alipay, Paytm, and Gojek serve as multifunctional ecosystems that facilitate user convenience while driving digital engagement and retention [1]. The scale and complexity of these platforms require robust architectural frameworks capable of handling massive user loads, modular feature development, and secure data interactions.

Two principal container models have emerged in the architecture of SuperApps: microservices and mini-apps. Each offers unique benefits and trade-offs in terms of scalability, modularity, and security, which are critical design considerations for developers and enterprise architects. This paper provides a focused comparative analysis of these models, assessing their technical suitability and implications for future SuperApp development.

The microservices architecture decomposes monolithic applications into loosely coupled, independently deployable services. These services typically communicate via lightweight protocols and are deployed using containerization platforms such as Docker, with orchestration handled by tools like Kubernetes. This model enables horizontal scaling, resilience, and service-specific CI/CD (Continuous Integration/Continuous Deployment) pipelines, making it ideal for rapidly evolving backend infrastructure [2]. Microservices have been widely adopted across domains requiring granular scaling and high availability, particularly in cloud-native systems.

In contrast, mini-apps represent lightweight, modular application units that reside within a host SuperApp. Typically built using web technologies (e.g., JavaScript, HTML5) and rendered through components like WebView, mini-apps provide user-facing functionalities while depending on the core SuperApp for services such as authentication, payment processing, and data storage. This approach simplifies third-party development and encourages a standardized, extensible front-end framework [3]. The mini-app model is especially prevalent in Asia-Pacific markets where platforms like Alipay and WeChat have enabled thousands of developers to integrate services without deploying full-scale applications.

However, mini-apps also introduce privacy and security concerns due to their close coupling with user interaction data. A recent study showed that mini-app usage and operation histories can be used to infer sensitive personal attributes using inference attacks with over 95% accuracy [4]. Notably, of 31 SuperApp vendors analyzed in the same study, only one (WeChat) explicitly acknowledged these data privacy risks, highlighting a lack of industry awareness regarding this attack vector.

The urgency of addressing architectural trade-offs between these two container models is underscored by the SuperApp market's exponential growth. Analysts project the global market for SuperApps will surpass $400 billion by 2030, driven by expanding mobile internet access and a demand for platform-based digital experiences [5]. As these platforms expand their user bases and diversify their feature sets, it becomes increasingly vital to understand how their architectural foundations affect operational efficiency, developer productivity, and user trust.

This paper evaluates microservices and mini-apps based on their contributions to scalability, modularity, and security within the SuperApp ecosystem. Drawing on empirical evidence, architectural best practices, and industry case studies, we aim to provide software architects and policymakers with practical guidance for selecting and refining container strategies. Additionally, the paper explores hybrid models where microservices manage backend logic and mini-apps handle frontend service delivery—a trend gaining traction as platforms seek to optimize both operational efficiency and user experience.

Ultimately, this comparative analysis offers actionable insights into containerized system design for next-generation SuperApps, facilitating better engineering decisions and more responsible digital ecosystems.

Methodology

This study adopts a conceptual comparative analysis approach grounded in secondary data sources. The evaluation is based on an extensive review of peer-reviewed academic literature, industry white papers, and real-world SuperApp case studies, including platforms such as WeChat, Alipay, and Gojek. These platforms were selected due to their global prominence, multi-functional architectures, and widespread adoption, offering rich insights into the practical deployment of microservices and mini-app container models.

The analysis framework centers on three critical architectural dimensions: scalability, modularity, and security. These criteria were chosen based on their relevance to system performance, development flexibility, and data protection—factors that are especially vital in large-scale, user-facing applications like SuperApps. Sources were selected based on credibility, recency (published between 2015-2024), and relevance to containerized system design. The aim is to distill practical implications and trade-offs for developers and architects evaluating container strategies in complex, modular platforms.

2. Scalability

Scalability is a critical architectural attribute for SuperApps, which often serve hundreds of millions to billions of users. These platforms must ensure responsiveness, low latency, and uninterrupted service during high-demand periods. While both microservices and mini-app container models enable scalability, they do so through fundamentally different architectural and operational mechanisms.

2.1. Scalability in Microservices

The microservices architecture is designed for high scalability by decomposing applications into independently deployable and loosely coupled services. Each service, typically encapsulating a specific business capability, can be developed, tested, deployed, and scaled independently. This architectural autonomy facilitates precise resource allocation and dynamic scaling, especially in response to fluctuating demand across various services.

Containerization technologies such as Docker and orchestration platforms like Kubernetes are foundational to the scalability of microservices. They allow services to be replicated horizontally, migrated across environments, and auto-scaled based on CPU/memory thresholds or traffic volume. For instance, organizations leveraging Kubernetes clusters in a microservices environment have reported up to 80% improvement in resource utilization due to automated scaling and load balancing features.

Moreover, the adoption of CI/CD pipelines plays a significant role in supporting real-time scaling and continuous availability. CI/CD pipelines have been reported to reduce downtime to under one minute in benchmarked environments; however, exact figures may vary depending on workload complexity, deployment tools, and system configuration. Additionally, resource utilization improvements of up to 80% have been observed in certain orchestrated container environments with fine-grained autoscaling capabilities, although such metrics are highly context-specific [6]. In mission-critical systems such as digital banking or online retail, this capability ensures minimal service disruption and a seamless user experience.

Additionally, microservices enable targeted scalability. For example, a payment gateway service might be scaled out during peak shopping seasons without affecting other components such as authentication or user profile services. This granularity reduces overprovisioning and improves cost efficiency [7].

However, managing a large number of microservices introduces operational complexity. Service discovery, configuration management, network latency, and monitoring must be tightly orchestrated to avoid bottlenecks. Despite these challenges, the microservices model remains well-suited for SuperApps that emphasize backend extensibility, autonomous team development, and continuous integration.

2.2. Scalability in Mini-Apps

Mini-apps, in contrast, are lightweight application modules that operate within the SuperApp's user interface layer. Their scalability stems not from individual autonomy but from their integration into the broader infrastructure of the host SuperApp. The SuperApp's centralized backend and cloud infrastructure provide high-throughput processing, database services, and user session management that support massive concurrency.

AliPay, for example, supports more than 1.3 billion users and manages thousands of mini-apps within its ecosystem. Rather than independently scaling each mini-app, AliPay leverages shared resources like distributed databases and elastic cloud functions to scale mini-app services collectively [8]. This centralized model simplifies deployment and allows mini-apps to benefit from global platform upgrades in performance and capacity.

Mini-apps are often developed using web technologies (e.g., JavaScript, HTML5) and executed via embedded WebView containers. This approach offers speed and flexibility in development but restricts the degree of independent control over backend performance tuning. Mini-app scalability is therefore indirectly governed by how efficiently the SuperApp scales its infrastructure, such as its load balancers, content delivery networks (CDNs), and database replication strategies.

Despite these limitations, mini-apps scale effectively within SuperApps that are engineered for elastic infrastructure provisioning. Features such as usage analytics, dynamic content caching, and global CDN integration enable mini-apps to deliver consistent performance even under high user loads. In high-traffic scenarios, SuperApps apply queuing systems and asynchronous processing to ensure mini-app responsiveness.

However, the centralized nature of this model also means that performance issues in the core platform may cascade across multiple mini-apps, affecting availability and latency. Additionally, the lack of independent scaling makes it difficult to apply differential resource allocation to individual mini-apps during traffic spikes.

2.3. Comparative Analysis of Scalability

While both container models enable scalability, their design philosophies and operational mechanisms differ significantly. The table below (**Table 1**) presents a comparative summary of their scalability characteristics:

Microservices offer fine-grained control, horizontal scalability, and independent resource scaling, making them suitable for backend-intensive systems where reliability and service isolation are critical. They empower engineering teams with flexibility and autonomy, though at the cost of operational complexity.

Mini-apps, meanwhile, leverage the overarching scalability of the host SuperApp platform. They enable rapid front-end deployment and global reach without the burden of managing backend infrastructure. However, this approach can lead to performance coupling and lacks per-service scalability control.

Table 1. Scalability characteristics of Microservices and Mini-apps (in SuperApps).

Aspect	Microservices	Mini-apps (in SuperApps)
Architecture	Independent services, containerized, horizontally scalable	Embedded web-based applications supported by centralized infrastructure
Scaling Mechanism	Service-level scaling via orchestration (e.g., Kubernetes, Docker Swarm)	Platform-level scaling through centralized SuperApp infrastructure
Resource Allocation	Tailored per microservice; supports traffic-based autoscaling	Shared cloud resources; economies of scale managed by platform
Deployment Flexibility	Independent deployment and versioning of services	Deployment constrained by SuperApp container and update cycles
Downtime & Availability	CI/CD pipelines reduce service downtime (<45 seconds)	Centralized resiliency strategies enable global uptime
Developer Control	High control over resource tuning and performance optimization	Limited control; scalability reliant on host SuperApp infrastructure

3. Modularity

Modularity is a foundational principle in modern software architecture, enabling systems to evolve through decoupled development, independent deployment, and flexible maintenance. It allows developers to isolate changes, accelerate feature rollouts, and minimize disruption to the overall system. In the context of SuperApps, both microservices and mini-apps offer modular structures, yet differ significantly in implementation scope, control, and flexibility.

3.1. Modularity in Microservices

Microservices architecture inherently promotes high modularity by decomposing a large application into discrete, independently deployable services. Each microservice encapsulates a specific business capability and interacts with others through well-defined APIs, often following REST or gRPC communication standards [9]. This architectural decoupling empowers development teams to work in parallel, use independent version control, and select the most appropriate technology stacks for each service.

One of the main advantages of this model is that it supports continuous delivery and agile development practices. Services can be tested and deployed independently without requiring a full system rebuild. This isolation also allows for faster debugging and fault recovery, since errors can be contained within a single module without cascading failures across the system [10].

The containerization of services—most commonly through Docker—further reinforces modularity by encapsulating each service in a standardized, reproducible environment. Developers can replicate microservices across development, testing, and production environments with minimal configuration drift, thus improving system reliability and reproducibility. Kubernetes and other orchestration platforms support this modular deployment by enabling autoscaling, service discovery, and fault tolerance at the container level.

However, this independence can introduce integration complexity. Service-to-service communication, API versioning, and distributed tracing must be carefully managed to maintain coherence. Still, in terms of modular flexibility, microservices remain one of the most robust approaches for building large-scale, dynamic systems like SuperApps.

3.2. Modularity in Mini-Apps

Mini-apps present a different but equally important approach to modularity, particularly in the SuperApp user interface layer. These lightweight, self-contained applications are embedded within the SuperApp and are commonly built using web-based technologies such as HTML5, CSS, and JavaScript. Mini-apps are rendered within containers like WebView, which enables them to run across platforms with minimal modifications.

The modularity of mini-apps stems from their ability to provide feature-specific functionality without requiring changes to the core SuperApp codebase. For instance, an e-commerce mini-app can be introduced or updated independently of the main platform. This capability supports rapid feature integration and third-party developer participation, which are hallmarks of SuperApp ecosystems like WeChat and Gojek [11].

Additionally, the mini-app model promotes interface standardization across the platform. Uniform development guidelines, design patterns, and API protocols are typically enforced by the host SuperApp, enabling consistent user experiences and easier onboarding for new developers. This level of consistency helps SuperApps scale their developer ecosystem and accelerate time-to-market for new services.

Nevertheless, this modularity is bounded by platform constraints. Mini-apps must operate within the SuperApp's runtime environment, which limits their control over backend logic, security policies, and resource management. They rely on APIs exposed by the host application and cannot introduce independent architectural decisions at the infrastructure level. In this sense, their modularity is mostly superficial, restricted to the front-end logic and integration point with the platform [12].

3.3. Comparative Analysis of Modularity

A direct comparison between microservices and mini-apps reveals key distinctions in how modularity is implemented and leveraged within SuperApps:

- Microservices offer deep modularity at the infrastructure and logic level. Each service functions as an independent module with its lifecycle, development pipeline, and resource allocation.
- Mini-apps, while modular from a user interaction and feature deployment perspective, are dependent on the core SuperApp architecture. Their independence is limited to the front-end and user interface layer.

Aspect	Unit of Deployment	Mini-apps (in SuperApps)
Unit of Deployment	Standalone services are deployed independently	Embedded mini-apps integrated within the SuperApp
Development Flexibility	Teams select independent stacks and libraries	Requires compliance with platform-specific SDKs and APIs
Service Isolation	Full isolation ensures localized failures	Shared runtime may cause cascading effects
Update and Maintenance	Targeted updates with CI/CD	Dependent on the SuperApp release cycle or approval processes
Control Over Backend	Full backend logic and database control	Limited to front-end logic; backend managed by host app

Both container models offer modular advantages, but their application scope differs. Microservices empower backend modularity with full control over service lifecycles, making them suitable for complex, dynamic systems requiring granular control. Mini-apps, on the other hand, facilitate rapid front-end modularization and service integration within a predefined platform, enabling broad developer participation but limiting architectural autonomy.

In SuperApp design, the choice of modular architecture depends on the development priorities. For large enterprises prioritizing flexibility, fault isolation, and innovation velocity, microservices provide superior modularity. For platforms seeking rapid front-end extensibility and a consistent user experience, mini-apps offer effective modular encapsulation—albeit with infrastructure trade-offs.

4. Security

Security is a central concern in designing and deploying modern application architectures. In the context of SuperApps, platforms that aggregate diverse services and manage extensive volumes of user data—the architectural choice between microservices and mini-apps carries significant implications for both system-level security and user data privacy. As these applications scale to serve billions of users, ensuring protection from external threats and safeguarding internal data handling mechanisms becomes essential to building trust and regulatory compliance.

4.1. Security in Microservices

Microservices architectures are inherently more secure in several respects due to their modular and isolated nature. Each microservice is containerized and deployed independently, allowing it to enforce its security protocols. This design supports service-level isolation, which limits the lateral spread of vulnerabilities across the system. For example, a compromised authentication service does not necessarily endanger the integrity of unrelated services like payments or user profiles.

Security best practices in microservices include role-based access control (RBAC), encrypted communication channels (such as TLS), API gateways, and service mesh frameworks (e.g., Istio) that enforce authentication and traffic con-

trol policies. In conjunction with CI/CD pipelines, these mechanisms support automated testing, vulnerability scanning, and continuous monitoring—thereby increasing resilience against evolving threats.

Despite these advantages, microservices introduce operational complexity that can itself be a security risk. The frequent communication between services opens multiple attack surfaces, such as insecure APIs or unvalidated message payloads. Improper configuration of service discovery protocols, container runtimes, or third-party libraries may lead to privilege escalation or denial-of-service (DoS) attacks [13]. However, these risks can be mitigated with industry-standard security practices, such as applying the principle of least privilege, container hardening, and zero-trust network architectures.

Another important aspect is security orchestration. As the number of microservices grows, maintaining consistency in authentication, logging, and encryption across services becomes more difficult. Platforms that adopt DevSecOps principles—integrating security throughout the development lifecycle—tend to maintain stronger defenses across distributed microservice environments.

4.2. Security in Mini-Apps

In contrast, mini-apps embedded within SuperApps face a distinct set of security challenges, particularly related to user data privacy. Mini-apps typically function as front-end modules rendered within a WebView container and rely on APIs provided by the host SuperApp for backend functionality. While this model simplifies development and integration, it centralizes sensitive data collection, which presents a significant privacy risk if not managed properly.

Recent research has highlighted a critical vulnerability: the passive collection and misuse of mini-app interaction history, including Mini-app Usage History (Mini-H) and Operation History (Op-H), the latter of which refers to logs detailing users' in-app actions such as taps, navigation paths, and function calls. These interaction logs, which track user behavior across various mini-apps, can be leveraged to infer sensitive attributes such as age, gender, financial status, or even political views [14]. A notable inference attack, THEFT (Target History Exploitation for Feature Tracking), demonstrated that it was possible to accurately predict sensitive user attributes with over 95% accuracy based on interaction histories and minimal training data.

The security concern is compounded by the lack of industry recognition of this threat. A study of 31 leading SuperApps found that only one vendor—WeChat—explicitly acknowledged the potential privacy implications of collecting Mini-H and Op-H data. Most SuperApps do not inform users about these practices, nor do they offer opt-out mechanisms or adequate anonymization protocols. This oversight creates a hidden and unregulated attack surface, particularly vulnerable to machine learning-based inference attacks and data misuse.

Furthermore, since mini-apps operate within a shared runtime environment, isolation between different apps is limited. If the SuperApp fails to enforce strict

sandboxing policies, one mini-app could potentially access session data or local storage from another, resulting in unauthorized data access.

4.3. Comparative Analysis of Security

The table below highlights the key differences in security posture between microservices and mini-apps:

Aspect	Microservices	Mini-apps (in SuperApps)
Isolation and Containment	High isolation; breaches are localized	Shared runtime may expose data across apps
Access Control	RBAC, secure APIs, and service mesh	API access controlled by host app; limited app-level control
Data Protection	Encrypted service communication; container-level encryption	Interaction data often unencrypted or insufficiently anonymized
Privacy Risks	Limited if proper governance is applied	High due to centralized behavior tracking and lack of consent mechanisms
Security Tooling	DevSecOps integration, CI/CD with security tests	Minimal per-app security tooling; centralized audit requirements
Industry Awareness	High due to maturity of microservices practices	Low awareness of novel privacy threats (e.g., THEFT attack)

Microservices provide a well-established security framework through modular isolation, encrypted communications, and DevSecOps practices. Although their distributed nature can introduce coordination complexity, their compartmentalized structure offers resilience against wide-scale breaches. These features make microservices well-suited for backend operations where confidentiality and system integrity are paramount.

Mini-apps, while offering ease of integration and rapid development, suffer from structural vulnerabilities related to centralized data collection and insufficient user privacy protections. The emergence of privacy inference attacks like THEFT has exposed an urgent need for industry-wide reassessment of data governance practices in SuperApp platforms. The lack of transparency, limited user control, and shared runtime environments contribute to a risk landscape that is poorly addressed in current implementations.

To address these issues, SuperApp developers must incorporate data anonymization, user consent mechanisms, and security auditing frameworks tailored for the mini-app paradigm. Without these safeguards, the continued growth of SuperApps could erode user trust and trigger regulatory scrutiny under privacy legislation such as the GDPR and China's Personal Information Protection Law (PIPL).

5. Integrated Discussion: Contrasting Container Models in SuperApps

The comparative analysis of scalability, modularity, and security reveals the complex trade-offs involved in selecting microservices or mini-apps as container mod-

els for SuperApps. These two paradigms embody contrasting design philosophies: microservices prioritize backend autonomy and scalability, whereas mini-apps emphasize front-end modularity and user engagement. In real-world applications, SuperApps often integrate both models, striking a balance between operational efficiency, development flexibility, and user-centric experience.

5.1. Architectural Integration in SuperApps

SuperApps operate within unique technological and business ecosystems, where architectural decisions are influenced by both performance requirements and strategic imperatives. Increasingly, a hybrid approach has emerged—combining microservices for backend operations with mini-apps at the user interface layer. In this architecture, microservices manage mission-critical services such as authentication, payments, and user data storage, while mini-apps extend user-facing functionality in a modular, low-friction manner. It is important to note, however, that while microservices enable modular backend operations, managing numerous interdependent services introduces operational burdens such as dependency management, service mesh orchestration, and failure tracing, all of which can significantly complicate development, debugging, and system reliability.

For example, WeChat employs a robust microservices infrastructure to support scalable backend processes, while hosting thousands of mini-apps for social interaction, e-commerce, and government services. This separation of concerns allows SuperApps to maintain operational resilience while iterating rapidly on customer-facing features (see Figure 1).

A conceptual architecture typically includes core microservices communicating via secure APIs, a mini-app container interface (often WebView-based), and an API gateway to orchestrate data flow and access control. This hybrid integration allows for seamless horizontal scaling, developer independence, and feature-level modularity, all within the same ecosystem.

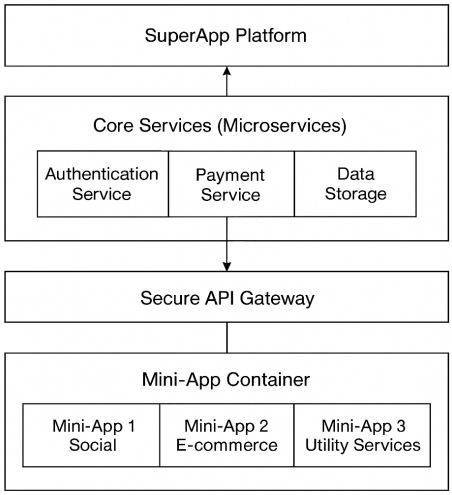

Figure 1. Hybrid SuperApp architecture integrating microservices for backend operations with a mini-app container for front-end modules.

5.2. Scalability and Operational Efficiency

Microservices are proven to scale horizontally by distributing services across containers and nodes. This capability is particularly beneficial for managing dynamic workloads and independent service scaling. For instance, a payments microservice can scale independently from a messaging module, optimizing compute resources and maintaining service-level agreements (SLAs).

Mini-apps, in contrast, scale as part of the larger SuperApp platform. Their performance and availability are heavily dependent on the centralized infrastructure provided by the host application. AliPay, for example, supports over 1.3 billion users by leveraging a resilient cloud backend and global CDNs, enabling its mini-apps to function reliably under massive concurrency. While effective, this model lacks the fine-grained control over individual mini-app resource allocation, potentially leading to platform-wide constraints during spikes in demand.

The hybrid model benefits from combining these strengths: the granular control and automation of microservices, with the extensibility and interface modularity of mini-apps. However, it also introduces orchestration complexity, particularly in aligning deployment pipelines, error handling, and monitoring across the two layers.

5.3. Modularity and Development Agility

Modularity in microservices enables independent development, deployment, and scaling, making it well-suited for agile workflows and DevOps environments. Services can be maintained, upgraded, or replaced without affecting the entire system, thus promoting faster innovation and continuous delivery.

Mini-apps support modularity through UI-level separation, allowing different features to be embedded or removed independently of the core SuperApp. However, they are constrained by the host app's SDK, runtime policies, and release cycles. This results in less autonomy for third-party developers, who must conform to predefined interaction patterns and platform limitations.

From a development agility standpoint, microservices support parallel pipelines and heterogeneous technology stacks, enabling faster iteration across teams. Mini-apps provide rapid feature deployment, especially for business units or external developers looking to integrate narrowly scoped services.

5.4. Security Trade-Offs and Privacy Concerns

Security considerations further distinguish these models. Microservices offer service-level isolation, secure communication, and customizable access control mechanisms. This modularization inherently reduces the blast radius of a potential breach and facilitates targeted remediation. Moreover, industry best practices—such as service meshes and CI/CD-integrated security checks—enhance overall robustness.

Mini-apps, while simpler to integrate, pose substantial privacy risks due to centralized data collection and limited user transparency. As highlighted in recent

research, interaction histories (e.g., Mini-H and Op-H) collected by SuperApps can be used to infer sensitive user attributes via machine learning-based inference attacks like THEFT. Alarmingly, only one out of 31 reviewed SuperApp vendors acknowledged this privacy threat, underscoring a critical gap in industry awareness and regulation.

Hybrid systems must therefore integrate both technical defenses (e.g., data anonymization, sandboxing) and policy-level safeguards (e.g., GDPR/PIPL compliance, explicit user consent) to mitigate privacy concerns while maintaining performance and usability.

6. Future Directions and Considerations

While the present comparative analysis offers a comprehensive view of microservices and mini-app container models within SuperApps, several areas merit further exploration. Future work must address the evolving technical, security, and regulatory challenges that arise from integrating these models at scale. This section outlines key directions for research and development aimed at strengthening privacy protections, enhancing security, refining hybrid architectures, and aligning with global regulatory standards.

6.1. Addressing Privacy Vulnerabilities in Mini-Apps

A significant concern in the mini-app paradigm is the exploitation of user interaction data—including Mini-app Usage History (Mini-H) and Operation History (Op-H)—to infer sensitive personal attributes. Future studies should investigate the following areas:

- **Data Anonymization Techniques:** Techniques such as differential privacy can help reduce the risk of re-identification. However, current implementations often trade off data utility for privacy, particularly in high-frequency mobile interactions [15]. Hybrid methods that combine local anonymization with secure aggregation may offer more robust solutions [16].
- **Trusted Hardware Solutions:** Trusted Execution Environments (TEEs) like Intel SGX can isolate sensitive operations and prevent unauthorized access during execution. Though TEEs show promise, their integration with large-scale SuperApps introduces cost, scalability, and energy-consumption concerns [17].
- **User Awareness and Consent:** Transparent privacy policies and granular consent controls can improve user trust and ensure compliance with data protection laws. Interfaces should be designed to clearly inform users when their interaction data is being collected and how it is used [18].

6.2. Enhancing Microservices Security

For microservices, future security improvements should focus on the following:

- **Secure Inter-Service Communication:** Implementing mutual TLS (mTLS) and token-based authentication (e.g., JWT) can enhance trust and integrity in

microservices communications [19].

- **Automated Vulnerability Scanning:** Integrating tools such as Aqua Security, Snyk, or Trivy into CI/CD pipelines ensures early detection of misconfigurations and third-party vulnerabilities [20].
- Intelligent Scaling Policies: Research into adaptive scaling mechanisms that consider both performance metrics and threat detection data can create more secure and responsive environments. Machine learning-based decision engines for scaling may help prevent denial-of-service (DoS) scenarios by throttling vulnerable endpoints [21].

6.3. Hybrid Approaches and Integration Strategies

Given that many SuperApps already blend microservices and mini-app architectures, exploring hybrid strategies is essential:

- **Backend–Frontend Separation:** Architectures that use microservices for core transactional services (e.g., payments, user identity, recommendation engines) and mini-apps for front-end engagement (e.g., e-commerce, social media) offer performance and development flexibility.
- **Unified Governance Frameworks:** SuperApps need cross-layer policies for access control, logging, and incident response. Unified governance ensures consistency in auditing, monitoring, and regulatory compliance across container layers.
- **Collaborative Research and Benchmarking:** Industry-academia collaborations can generate empirical benchmarks and public testbeds for SuperApp security, performance, and modularity. Such collaborations can help develop and validate best practices across diverse deployment environments.

6.4. Implications for Regulation and Industry Standards

The privacy and operational risks associated with mini-apps call for updated regulatory frameworks:

- **Standards for Data Collection and Processing:** Governments and industry bodies must define transparent guidelines around what user data can be collected, for how long, and under what legal justification. This includes clarifying how aggregated interaction histories should be anonymized and stored.
- **Auditing and Certification Programs:** Independent security audits, akin to SOC 2 or ISO/IEC 27001 certifications, can help ensure that SuperApp platforms adhere to industry-standard controls. Regular third-party evaluations can enhance both security and consumer trust.
- **User Empowerment Mechanisms:** Tools that give users control over their data—such as personal data dashboards, revocation mechanisms, and activity transparency reports—are vital for aligning SuperApps with privacy regulations like the GDPR and China's PIPL.

7. Conclusion

This study has presented a comprehensive evaluation of microservices and mini-

apps as competing container models in the architecture of SuperApps, with specific attention to their performance in scalability, modularity, and security. Drawing from recent research and real-world implementations, the analysis provides a well-rounded foundation for understanding how each model supports—or constrains—the rapidly growing ecosystem of digital SuperApps.

Scalability

Microservices support independent horizontal scaling of discrete services. Their compatibility with container orchestration platforms and CI/CD pipelines enables responsive resource management, fault tolerance, and high system availability. In contrast, mini-apps scale collectively within the broader SuperApp infrastructure. While they benefit from centralized cloud services optimized for mass concurrency, they lack the granular control over resource allocation that microservices offer.

Modularity

Microservices deliver strong backend modularity by enabling parallel development, technology heterogeneity, and independent service updates. This fosters rapid innovation and system agility. Mini-apps contribute interface-level modularity, allowing third-party developers to add features to the platform without disrupting core SuperApp services. However, their modularity is constrained by platform dependencies and runtime limitations imposed by the SuperApp framework.

Security

Microservices facilitate security through isolation, where vulnerabilities in one service can be contained without affecting the rest of the system. The use of DevSecOps pipelines, secure API gateways, and encrypted communications enhances defense-in-depth strategies. Mini-apps, however, introduce a novel category of privacy risk. Their inherent collection of behavioral data—such as interaction and operation histories—can be exploited to infer sensitive personal attributes. The lack of industry awareness and absence of robust consent mechanisms exacerbate these concerns.

Summary of Key Findings

- **Scalability:**
  - Microservices offer dynamic, independent scaling per service.
  - Mini-apps leverage platform-wide scalability but lack service-level flexibility.
- **Modularity:**
  - Microservices support backend decoupling and autonomous development.
  - Mini-apps enable modular user experiences with standardized interfaces.
- **Security:**

- Microservices isolate risks and support granular security controls.
- Mini-apps pose emerging privacy challenges due to centralized data collection and insufficient safeguards.

In conclusion, both microservices and mini-apps provide significant value for SuperApp design, but each comes with its own limitations. Microservices are more suitable for backend operations requiring flexibility, scale, and isolation, while mini-apps excel at front-end integration and rapid deployment within unified user experiences. However, to ensure secure and sustainable growth, developers must proactively address the privacy gaps inherent in mini-app models and enhance interoperability between container models.

Future research and development should focus on implementing hybrid architectures, strengthening regulatory compliance, and investing in privacy-preserving technologies such as differential privacy, federated learning, and trusted hardware. These strategies will be critical to reconciling the trade-offs between performance, security, and usability in the next generation of SuperApps.

Ultimately, this paper serves as a blueprint for system architects, developers, and policymakers who aim to build resilient and user-centric SuperApp platforms that align with both technical best practices and emerging privacy standards.

Limitations

This study is conceptual in nature and primarily relies on secondary data from academic literature, white papers, and publicly available case studies. While this approach enables broad comparative insights into container architectures, it lacks direct empirical validation, such as deployment metrics, interviews, or benchmarking within live SuperApp environments. Additionally, the focus on three major dimensions—scalability, modularity, and security—provides breadth but limits the depth of analysis in each category. As such, the conclusions drawn should be interpreted as exploratory and foundational. Future work should build on this framework by incorporating empirical studies and evaluating hybrid architectures under real-world operational conditions.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Jain, V., Arora, S. and Mehta, K. (2022) Digital Ecosystems and the Rise of Super-Apps: Case of India and Southeast Asia. *Asian Journal of Business Research*, **12**, 45-59.

[2] Dragoni, N., Lanese, I., Larsen, S.T., Mazzara, M., Mustafin, R. and Safina, L. (2018) Microservices: How to Make Your Application Scale. In: Petrenko, A.K. and Voronkov, A., Eds., *Perspectives of System Informatics*, Springer International Publishing, 95-104. https://doi.org/10.1007/978-3-319-74313-4_8

[3] Liu, Y., Zheng, H. and Hu, W. (2021) Application Modularization in Super Apps: Exploring the Role of Mini-App Platforms. *Journal of Systems and Software*, **180**,

Article ID: 111014.

[4] Wang, S., He, J. and Zhao, L. (2022) I Can Tell Your Secrets: Inferring Privacy Attributes from Mini-App Interaction History in SuperApps. *Proceedings of the IEEE Symposium on Security and Privacy*, San Francisco, 23-25 May 2022, 1-14.

[5] Statista (2023) SuperApp Market Value Forecast Worldwide from 2022 to 2030. Statista Research Department. https://www.statista.com/

[6] Fazio, M., Puliafito, A., Villari, M. and Dustdar, S. (2020) Containers and Micro-Services for Scalable IoT Applications: A Performance Evaluation. *Journal of Internet Services and Applications*, **11**, 1-14.

[7] Sharma, M. and Soni, N. (2022) Optimizing CI/CD Pipelines in Microservices Using Containerization Tools. *Software Engineering Journal*, **37**, 67-75.

[8] New Relic (2021) Microservices Scaling: Performance Tips and Trade-Offs. https://newrelic.com/

[9] Zhou, L., Yang, Y. and Chen, H. (2022) Scalable Architecture of the AliPay Mini-App Ecosystem. *IEEE Access*, **10**, 84791-84804.

[10] Newman, S. (2015) Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.

[11] Taibi, D., Lenarduzzi, V. and Pahl, C. (2018) Continuous ARCHITECTING with micro-Services and DevOps: A Systematic Mapping Study. *Journal of Systems and Software*, **146**, 120-135.

[12] Zhang, X. and Xu, J. (2020) Mini-App Platforms in SuperApps: Enablers of Digital Innovation. *ACM Computing Surveys*, **53**, 1-35.

[13] Berardi, D., Giallorenzo, S., Mauro, J., Melis, A., Montesi, F. and Prandini, M. (2022) Microservice Security: A Systematic Literature Review. *PeerJ Computer Science*, **8**, e779. https://doi.org/10.7717/peerj-cs.779

[14] Mohanty, A. and Tripathy, A. (2022) DevSecOps: A Practice to Integrate Security in DevOps Lifecycle. *International Journal of Security and Software Engineering*, **13**, 45-59.

[15] Dwork, C. and Roth, A. (2013) The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, **9**, 211-407. https://doi.org/10.1561/0400000042

[16] Beaulieu-Jones, B.K., Wu, Z.S., Williams, C., Lee, R., Bhavnani, S.P., Byrd, J.B., *et al.* (2019) Privacy-Preserving Generative Deep Neural Networks Support Clinical Data Sharing. *Circulation*: *Cardiovascular Quality and Outcomes*, **12**, e005122. https://doi.org/10.1161/circoutcomes.118.005122

[17] Costan, V. and Devadas, S. (2016) Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086.

[18] European Commission (2021) General Data Protection Regulation (GDPR). https://gdpr.eu/

[19] Javed, M., Ullah, I. and Abbas, H. (2021) A Review of Microservices Security: Challenges and Best Practices. *Journal of Cloud Computing*, **10**, 1-17.

[20] Snyk Ltd. (2022) The State of Open Source Security. https://snyk.io

[21] Tang, C., Wang, Q., Liu, W. and Yu, Y. (2021) AI-Based Autoscaling for Micro-Services in Cloud Environments. *Future Generation Computer Systems*, **125**, 572-584.