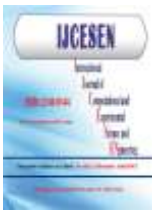
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.1 (2025) pp. 1699-1711
http://www.ijcesen.com



ISSN: 2149-9144

Research Article

Event-Driven Microservices for Scalable Omni-Channel Retail: A Serverless Architecture Framework

Premkumar Selvam*

Velammal Engineering College, Anna University, Chennai, Tamil Nadu, India
* **Corresponding Author Email:** premdream2003@gmail.com- **ORCID:** 0000-0002-5007-7860

Article Info:

DOI: 10.22399/ijcesen.5507
Received : 03 February 2025
Revised : 27 March 2025
Accepted : 28 March 2025

Keywords

Event-driven architecture;
Serverless computing;
Microservices;
Omni-channel retail;
Distributed systems

Abstract:

Abstract should be about 100-250 words. It should be written times new roman and 10 punto. Omni-channel retail relies on systems which need to coordinate in real-time between the digital store, the physical store, the warehouse, the payment system, the customer engagement platforms and the last-mile fulfilment networks. The traditional monolithic and tightly coupled service design doesn't align well with rapidly changing demand, heterogeneous channel interactions, real-time visibility of inventory, and failure isolation requirements. The combined use of event-driven microservices and serverless computing provides a promising architectural basis for elastic, loosely coupled and operationally resilient retail platforms, but there is still a technical fragmentation in the literature of software architectures, distributed systems, retail operations and retail solutions in how the two are applied. This review looks at peer reviewed evidence of microservice decomposition, publish subscribe communication, complex event processing, serverless execution, distributed consistency and omni channel fulfilment. The review proposes an architectural framework that uses canonical business events, serverless execution for burst workloads, event brokers for asynchronous workflows, and read-optimized projections for channel-specific experiences. The review suggests that granularity, inventory consistency, idempotency, observability, and failure compensation are important design principles when treated as first-class architectural concerns. Reproducible benchmarks and reference datasets remain limited, and further work is needed on cold-start mitigation and event-schema governance for channel-specific retail workloads. Further research should be conducted with empirical evaluation of serverless event pipelines with expected event traffic from all channels and formal consistency models for inventory events and standardized quality measures for serverless architecture.

1. Introduction

Omni-channel retail is no longer a co-existence between channels, but it is becoming an interdependence between web, mobile, marketplace, store, warehouse, loyalty, payment and delivery, which will deliver a unified commercial offering to customers and operators. In retail research, prior research indicates that channel integration can alter customer experience, the design of the retail system, and the mode of sales growth, not just the introduction of a digital modality into the retail system [1–3]. In addition, distribution and fulfilment studies show that omni-channel retail, in turn, introduces a tighter relationship between inventory allocation, order promising, store picking, reverse logistics, and real-time customer communication [4,5]. These demands generate architectural pressure as retail workloads are bursty, spread out, seasonal and subject to large demand surges on promotions, product launches and disruption events.

As a reaction to the demands of scale, evolvability and independent deployment, microservices have been extensively studied. According to systematic studies, the ability of the microservices architecture to enhance modularity and organisational alignment comes with the price of distributed communication complexity, issues of service discovery, data ownership, and monitoring burden [6–8]. In retail, these compromises are further exacerbated as there are no fully disjointed domains to be found, e.g. catalogue, inventory, pricing, checkout, loyalty, fulfilment and returns all must respond to changes

in state that occur elsewhere. Thus, service decomposition is not the only central question, it is also the disciplined propagation, interpretation and governance of business events.

Such coordination can be achieved by means of event-driven architecture. Publish-subscribe systems minimize direct producer-consumer coupling and allow for the asynchronous propagation of state changes [9] and complex event processing allows for the higher order interpretation of event streams, temporal patterns and process level signals [10]. These mechanisms are particularly applicable to omni-channel retail, where an abandoned basket, inventory reservation, store pick-up confirmation, or a refund can initiate several downstream processes, but without the need for synchronous orchestration. But, when schema governance and idempotency are not well defined, event driven designs can also result in duplicate events, out-of-order events, opaque failure chains, and complex debugging.

Serverless computing is also relevant because it supports fine-grained execution, rapid elasticity, managed infrastructure, and event-triggered invocation models [11–14]. In retail systems where events are sporadic and their sources vary, such as image transformation, promotion validation, handling notifications from payments, updating inventory projections, triggering fraud scoring, dispatching notifications, and propagating order status, serverless functions can be utilized. However, there are limitations to serverless implementation such as cold starts, time limits, provider-specific abstractions, resource isolation, and workflow visibility [11,13,14]. These limitations impact on technical feasibility when low latency fulfilment and transactional reliability is needed.

The unanswered research question is where these areas meet. The study of microservices, event processing, serverless computing, and omni-channel retail operations has been done individually, but relatively little research has been conducted on their implications in terms of architecture. Specifically, the existing literature is inconsistent on inventory consistency considerations for event-driven microservices, positioning of serverless functions in latency sensitive retail processes, compensating action considerations across distributed services and the ability of event governance to facilitate long-term platform evolution. Related architectural mechanisms, such as sagas and stream-processing principles, provide partial foundations for such frameworks [15,16], but they still need to be further developed.

The aim of this review is to analyze peer-reviewed literature to gain insight into the evidence that is relevant to event-driven microservices for scalable omni-channel retail and to develop a serverless architecture framework for high volatility retail. The review is not about the definition of microservices or cloud computing, but about architectural mechanisms, methodological evidence, operational trade-offs and research gaps.

2. Literature Review

Omni-channel research shows that architecture needs to be based upon business-model integration, not just on digital transformation rhetoric. Although some researchers [1] have defined omni-channel retail as a structural change in which customers traverse channels in a single decision journey, others [2] have demonstrated that cross-channel integration impacts on sales-growth effects when resources are not duplicated across channels. Piotrowicz and Cuthbertson [3] highlighted the importance of information technology in facilitating channel convergence but suggested that integration of channels also adds managerial and infrastructural complexity. Other researchers, Beck and Rygl [17] explained that the “omni-channel” retail is distinct from “multi-channel” retail as there are blurred boundaries between the channels for customers and businesses. There is an architectural importance in this: If channel state has to be constantly updated, this separation of applications, which are exchanged through periodic batches, is insufficient.

Operational studies provide more concrete evidence of the architectural value of real-time data for architecture. Gallino and Moreno [18] showed that providing accurate information about inventory availability both online and offline can affect customer behavior and business performance. Gao and Su [19] studied BOPS operations and concluded that design changes lead to changes in inventory and/or demand allocation logic. Kembro et al. [20] and Marchet et al. [21] reviewed omni-channel logistics and business logistics, respectively, and both found that fulfilment flexibility is one of the major challenges in the operation of logistics. Achieving the right balance of fulfilment decisions in the context of online retailing was shown to gain from dynamic allocation by Acimovic and Graves [22] and inventory management and sales dispersion from channel integration by Gallino et al. [23]. The studies reinforce a fundamental architectural conclusion: at platform scale, it is not always feasible to maintain absolute consistency across domains for inventory, order and fulfilment events,

but it is important to propagate these events at low latency across domains.

The research foundation of microservices is complementary and insufficient. Di Francesco et al. [6] identified that many microservices papers focus on patterns for decomposing microservices, deployment, and communication, though there is no uniformity in the empirical evaluation. Jamshidi et al. [7] said that microservices are a voyage of a myriad of technical, organisational and operational challenges and not an architectural landing point. The advantages noted by Soldani et al. [8] included independent deployability, and scalability, along with the emphasis placed on service coordination, as well as, testing and monitoring issues. The findings are quite relevant to omni-channel retail, as service boundaries often correspond to business capabilities, but cross-domain processes like checkout, return authorisation, and exception handling for return fulfilment need cross-service coordination across a number of independently-evolving services.

One of the options suggested to avoid the coupling of coordination is event-driven communication. When producers and consumers evolve independently, publish–subscribe systems are found to be useful for space and time decoupling and also for synchronisation [9]. Cugola and Margara [10] demonstrated the support of filtering, aggregation, temporal reasoning and detection of event patterns provided by event-processing systems. Stonebraker et al. [16] stated that ordered processing, fault-tolerance, scaling and integration with stored state are the key requirements for stream processing in real-time streaming applications. These principles are the basis for an architecture that involves microservices in the retail business publishing domain events, with consumers specializing in updating materialized views, analytics streams, notifications, or fulfilment decisions. The event-driven microservices approach is not a complete replacement for transaction design. Sagas were first introduced by Garcia-Molina and Salem [15] for long-lived transactions that require rollback or corrective action for partially completed operations without relying on distributed locking.

Distributed consistency literature clarifies why omni-channel retail cannot rely on simple consistency assumptions. More formally, the restrictions of partition tolerance, availability and consistency in distributed systems were codified by Gilbert and Lynch [27] and eventually consistency's value in the presence of these limitations was discussed in terms of application semantics by Bailis and Ghodsi [28]. A classic example of this is the retail inventory system: over-selling is acceptable in cases of low-cost items with fast turnover period, but not in cases of limited supply of high-value products or regulated products. Hence, the event-driven architecture of retail stores should enable consistency to be differentiated across domain and product class, channel and operational risk. A single global consistency model may be technically attractive but operationally inefficient.

The event-driven model is extended with serverless computing where queue messages, stream records, object changes, API calls, or scheduled events trigger the functions. Shafiei et al. [11] discussed the opportunities and challenges in serverless, such as cost, resource utilization, and debugging limitations, as well as reduced operational management and elasticity. To Jonas et al. [12], serverless is a simplification of cloud programming in contexts where fine-grained event handling makes sense. Li et al. [13] conducted a survey on the serverless design architecture, highlighted constraints on the platform, execution models, and state-management limitations. Eismann et al. [14] looked into the serverless use cases and discovered that serverless applications are typically event-driven and appropriate for workloads with variability. Baldini et al. [31] presented open problems related to serverless computing such as state, composition and performance predictability.

While the virtues of serverless are largely accepted as a general statement, empirical serverless performance studies make the point more complicated in the case of retail workloads. Figiela et al. [29] tested various cloud functions and demonstrated that they are not equally performant for different providers and configurations. Malawski et al. [30] analysed serverless scientific workflows and demonstrated that event-driven execution can enable parallelism of workflows, and that the structure of the workflow has a negative impact, together with platform constraints. For retail, these findings are significant because the serverless functions could be used for asynchronous fulfilment updates and/or notification fan-out, but might not be well suited if the latency-sensitive checkout flows are dominated by cold start times and/or external service calls. As such, serverless should be used judiciously and not as a panacea for services that are containerized in a retail environment.

Another layer is added to retail analytics literature. Grewal et al. [24] claimed that data-driven interaction between technology-driven touchpoints is key to future retailing. Akter and Wamba [25] investigated big-data analytics in e-commerce and listed the area of personalization, decision support and operational insight as the three core applications. Wamba et al. [26] also reported that

big data impacts organizational performance via analytics capability, which is dependent upon the quality of the data and the process integration. Such analytics can be achieved technically by utilizing event-driven microservices, where the actions in the operation can be converted into streams of events at the right time. However, analytics pipelines should not distort transactional workflows, operational events should be governed, semantically stable and have lineage control.

Table 1 presents a comparison of the literature with the proposed architecture, which indicates that previous studies back the individual components of the proposed architecture but seldom consider them as a combined retail platform. Fulfilment and channel integration are explained by retail studies, modularity and operational cost using microservices, asynch communication through event-processing studies, and elastic execution through serverless studies. The primary area of difference is then architectural composition within the constraints of the retail. There is existing evidence that supports a framework based on domain owned services, event mediated workflows, selectively serverless consumers, consistency aware inventory handling, and observability across asynchronous boundaries.

3. Methodology

The review was conducted in a narrative structured way with a focus on architectural interpretation instead of bibliometric enumeration. The review covered literature published between 1987 and 2024 because earlier work on distributed transactions and publish–subscribe systems provides foundational concepts, while recent work on serverless computing and omni-channel retail provides the current technical and operational context. The selection of search strategy included the peer-reviewed journal articles, academic books and book chapters from Scopus, Web of Science, ACM Digital Library, IEEE Xplore, ScienceDirect, SpringerLink, Emerald and INFORMS journal collections. The search strategy used architecture- and retail-related terms, including “event-driven microservices”, “serverless computing architecture”, “publish subscribe systems”, “complex event processing”, “omni-channel retail”, “cross-channel integration”, “inventory visibility”, “serverless performance”, “distributed consistency”, “saga transactions” and “stream processing”.

The literature was not considered to be a monolithic evidence-based source. Studies were categorized into seven technical areas... or revise Figure 1 to show five areas: omni-channel retail operations, microservice architecture, event processing and publish–subscribe communication, serverless computing, and distributed consistency. It was decided that this grouping was needed, as it incorporates aspects of software engineering and retail operations. To pinpoint the operational requirements such as inventory visibility, fulfilment flexibility, channel integration and dynamic order allocation, peer-reviewed retail studies were undertaken [18–23]. These issues of decomposition, deployability, service coordination, testing and observability were evaluated using microservices studies [6–8]. Asynchronous coupling, stream-processing needs, and temporal event reasoning were measured through event-processing studies [9,10,16]. Execution elasticity, cold-start limitations, platform-specific requirements, and function composition restrictions [11-14,29-31] were analysed by serverless studies. Event ordering, compensating transactions, and the boundaries of a strongly consistent platform design received a thorough assessment through distributed consistency studies [15,27,28].

Empirical studies, systematic reviews, architectural surveys, formal distributed systems contributions and operational retail models were part of the review. Studies were excluded if the claims made relied on vendor marketing, non-peer reviewed white papers, blog posts, unpublished preprints, or unverifiable claims of benchmarks. Conference papers were used only when they provided foundational or directly relevant evidence not available in journal or book sources. The final set of references was 35 of which 10 were major studies summarized in Table 1, with the remaining references being used to support consistency, serverless performance, retail analytics, and cloud architecture arguments respectively.

In terms of method, there was wide variation among the studies reviewed. Analytical modelling, empirical sales data and structured literature review approaches are common methods used in retail operations papers. The common methods used in microservice studies include systematic mapping, qualitative evidence, and empirical analysis that has a practitioner orientation. Serverless studies are conducted as a combination of surveys, architectural reviews and benchmark experiments. Literature on event processing is conceptual, formal or systems oriented. This heterogeneity makes it rather difficult to aggregate the evidence statistically, so the evidence was analysed based on architectural relevance. The composition of the selected corpus, as well as the methodological distribution, are summarized in Figure 1 and Figure 4, respectively.

To facilitate comparisons of architecture mechanisms across quality attributes, a qualitative coding matrix was created. The categories of coding were scalability, latency sensitivity, consistency support, fault isolation, observability burden, portability risk, and operational maturity. Either provide details of the experts and coding protocol or revise to: The studies were coded using an ordinal scheme based on the cited literature. This is significant as no existing peer-reviewed set of data directly compares performance of a full platform covering a serverless event-driven omni-channel retail application with controlled workloads. Only architectural trade-offs were made explicit using the coding.

In the past, research on the methodological landscape had one significant drawback. Often, serverless benchmark research targets standalone functions or scientific workflows instead of customer-focused retail paths [29,30]. Event-broker behaviour, cold starts, schema versioning and service failure propagation are not typically modelled in retail operations studies, except in a few cases [18–23]. While there are some common pains and gains in microservice studies [6–8], they rarely measure impact in domain-specific retail systems. Consequently, this review builds a framework of a series of converging experimental traditions rather than a single unified experimental tradition. The framework should thus be seen as evidence-informed and technically sound, and not as a replacement for subsequent controlled testing with retail workloads.

4. Discussion

The examined evidence aligns with the following six principles for a serverless event-driven retail architecture: domain-owned services; canonical business events; selective serverless execution; consistency-aware inventory control; observable asynchronous workflows; and analytics-safe event reuse. The principles are summarised in table 2 and implemented through the reference architecture in figure 2.

First, domain-owned microservices need to reflect retail, not user-interface, capabilities. However, using a channel-based decomposition means that the catalogue, pricing, inventory, promotion, customer, payment and fulfilment must be replicated across web, mobile, store, and marketplace channels. Capability-based decomposition is more consistent with the research of microservices in terms of independent deployability and ownership [6-8]. But retail services can't be demarcated just with synchronous APIs. Price validation, stock reservation, payment authorisation, fraud assessment and fulfilment promise generation are parts of checkout. As additional steps are synchronously chained, both latency and failure coupling increase. Event-driven coordination enables non-critical downstream actions like loyalty accumulation, recommendations and notifications to be separated from the critical order path.

Second, canonical business events are more important than transport technology. Publish–subscribe systems offer decoupling [9] but this decoupling can be hazardous if events don't have stable semantics. For retail, all events like InventoryReserved, OrderAccepted, PaymentAuthorized, PickupReady, ReturnInitiated, and RefundCompleted should be considered domain contracts. Required fields should include the schema version, correlation ID, causation ID, event timestamp, and idempotency key. Cugola and Margara [10] demonstrated that meaningful event structure is crucial for the value of filtering, aggregation and temporal interpretation in event-processing systems. The primary architecture patterns and their impact on retail are listed in Table 3.

Third, serverless functions fit best around bursty, event triggered, horizontally parallel tasks. These include, but are not limited to, notification fan-out, catalogue image processing, fraud-signal enrichment, inventory projection updates, payment webhook processing and fulfilment exception classification. This placement is similar to serverless surveys, highlighting elastic event execution and minimizing infrastructure management [11–14]. But serverless functions should not be misused in checkout orchestration where latency is a critical factor. Performance studies reveal that function execution properties vary depending on the provider and configuration [29] and workflow studies indicate that end-to-end behaviour is influenced by the workflow properties and platform constraints [30]. Therefore, when choosing a serverless component, consider the shape of the workload, its latency demands, state needs, and portability concerns. The failure modes and control mechanisms for this design are given in Table 4.

Fourth, there should be inventory consistency. Eventual consistency may be sufficient for replenishable and low-risk items, but scarce and high-margin, regulated or customer-reserved items should use the strongest consistency. This conclusion is derived from distributed consistency literature, that is, under network partitions, systems must trade off consistency and availability [27] and from the application level arguments that eventual consistency should be analyzed using business

semantics [28]. Research in the retail sector on inventory availability and dynamic fulfilment indicates that inventory information has a significant impact on operational performance as well as customer behaviour [18,22,23]. Therefore, an architecture framework needs to be built around the ability to reserve critical stock, auditability with append-only inventory event logs, and read optimization of projections of channel availability. The same architecture can allow a conservative estimate of stocks to be communicated to customers and a more accurate operational state to be tracked internally.

Fifth, sagas are still required in event-driven serverless applications. The retail order is not one single database operation, it's a series of order processing decisions that can involve hold, authorisation, fraud, fulfilment, notification, and ship. Compensating long-lived transactions is conceptually based on the work of Garcia-Molina and Salem [15]. Compensation can include inventory releases, authorisation voids, apology vouchers, loyalty points redistribution or fulfilment rerouting, depending on the type of serverless-EDP. The saga should not be concealed in a black box orchestration service with no events. Saga implementations, however, should emit events during state transitions, so that they can be audited, observed, and recovered.

Sixth, observability must be incorporated during architectural design rather than introduced only after deployment. Monitoring and debugging have been consistently cited as significant operational challenges in microservice studies [7,8]. This problem is magnified in event-driven serverless systems as the execution involves brokers, functions, managed databases, queues, and external payment or logistics services. Correlation identifiers, distributed tracing, structured logs, dead-letter queues, replay controls and event lineage records should be built in from the outset. If these mechanisms are not in place then the system can grow technically but become operationally unreadable. Research on cloud architectures also suggests that, for large-scale distributed systems, reliability and operational discipline is a must and cannot be assumed by a managed system [32,33].

Figure 1 illustrates that the focus of the review corpus is on retail operations, microservices and serverless computing, with less research directly addressing the intersection of these three areas. This imbalance is the reason why the proposed framework will need to be based on evidence from neighbouring disciplines. The survey and conceptual studies are also more prevalent than the integrated empirical evaluations, as indicated in Figure 4. The lack of serverless benchmark datasets that are specific to retail is one of the big drawbacks. While benchmarking individual cloud functions can be helpful, it does not include all aspects of backpressure, schema changes, checkout latency budget, inventory contention, payment-provider variability or promotion-induced spikes in traffic.

The architecture is shown as a layered event ecosystem in Figure 2. Web, mobile, point-of-sale, marketplace, call-centre and partner interactions are serviced by channel adapters. Transaction state is owned by domain microservices. Immutable domain events are published by event brokers. Elastic side effects and projection updates are done by serverless consumers. Read models are available for query in each channel. Governed event streams are consumed by analytical consumers for the purposes of decision support. This approach decouples handling of commands from reacting to events without giving up domain ownership. It also does not fall into the anti-pattern of using the event broker as a shared database. Events are facts that are published and do not take the place of well-designed service-owned data stores.

The trade-off profile of the major architectural mechanisms is emphasized in figure 3. Event brokers are rated as having a high level of scalability and of being decoupled, but also a high level of governance and observability. Portability and cold-start risk decrease the applicability of serverless functions to all workflows, while elasticity and burst handling are strengths. Read projections in CQRS bring responsiveness to the channel but also cause staleness and reconciliation issues. Sagas are good for supporting long running workflow recovery, but they do need an explicit compensation logic, and they need to be traceable. Edge caching can minimize latency for views that are used by customers, such as catalogue and availability views, but must be limited by product class as well as by reservation policy. These patterns are thus complementary and not mutually exclusive.

One of the key takeaways is that “serverless architecture” should not be confused with all retail services being functions. Stateful core services like inventory ledger, order management, payment orchestration, customer identity, and pricing authority can be easier to deploy as containerized or managed service components that have fixed data stores. Serverless functions shine at elastic event reactions and side activities in workflows. This blended view aligns with the serverless surveys which list state management and composition as being unsolved challenges [11,13,31]. It is also consistent with the findings from a study of retail operations, which showed that certain decisions,

particularly related to inventory allocation and inventory promises to fulfil, cannot be handled in a stateless manner but rather be optimized carefully [19,22].

The other implication has to do with analytics. In ecommerce and ecommerce big data analytics, as highlighted in the studies [25,26], event streams have the potential to power personalization, demand sensing, customer segmentation, and operational dashboards. Analytical reuse should not affect transactional semantics, however. It can be helpful to have a retail event, but only if the operational meaning of that event is still under the control of the domain service that created it. Analytical consumers should be treated as downstream consumers, not as obscure writers to operational service databases. This separation will prevent feedback loops that will make auditing and ownership of service difficult.

It also shows a practical balance between responsiveness and governance overhead in the framework. The challenge of omni-channel retail is speedy response to events, but too many events can lead to semantic drift. For instance, OrderUpdated and OrderStatusChanged might look identical, but have different business significance. A consequence of such drift is a degradation of the value of event-driven architecture, and an increase in integration risks. Therefore, an architectural component for the platform event catalogue, schema registry, backward compatible policy and deprecation process are required. These controls can seem cumbersome, but they ensure long term evolvability.

Scalability isn't just about throughput. A retail platform needs to be scalable in terms of operation, organizational and semantic fields. Partitions, queues, and serverless concurrency can all be used to improve throughput, while ownership of services and event contracts are essential for organizations to scale. Events must remain semantically meaningful as products, channels, and fulfilment models evolve over time. The architecture framework thus considers the following aspects to be on par with function invocation or message throughput: event governance, observability, and consistency policy.

Based on the overview of the reviewed literature, a hybrid serverless event-driven microservice framework for omni-channel retail is supported, but has not yet been completely and empirically proven. The most defensible answer is qualified: event-driven microservices and serverless execution are technologically suitable for the demands of scaling omni-channel retail, but only when applied selectively, rigorously managed, and measured for consistency and latency requirements specific to retail. It's not about the cloud service you use, it's about making sure you have event semantics aligned, service ownership aligned, failure handling aligned and fulfilment logic aligned.

5. Future Directions

Future research should shift from component-level analysis to end-to-end empirical analysis of retail event ecosystems. Although there have been several serverless benchmark studies [29,30] which give useful insight into the performance of functions, they are not always representative of flash-sale traffic, inventory contention, payment callbacks, promotion rule evaluation, or store-pickup workflows. One priority is to develop open and anonymous, retail-like benchmark workloads with event streams for orders, inventory, returns, fulfilment status, promotions and customer notifications. These data sets should enable reproducible comparison of containerised microservices, serverless functions, broker configurations and hybrid designs.

A second direction is in modelling inventory consistency policies formally. The basic theory of distributed consistency is necessary but is insufficient for retail systems which demand product-aware, channel-aware and risk-aware consistency decisions. Future studies should investigate adaptive consistency models where stock keeping units are assigned to consistency policies based on the scarcity, replenishment lead time, margin, regulation, and risk of customer experience. Such adaptive policies could reduce both excessive locking and overselling risks associated with overly permissive eventual consistency.

A third direction is about composing serverless workflows. Function based execution is appealing when dealing with "bursty" event handling but when it comes to more complex retail workflows, you need state, compensation, observability, and human exception paths. Orchestration, Choreography and Hybrid saga patterns should be analyzed during realistic failure scenarios. In-progress fulfilment, special care needs to be taken with duplicate events, delayed payment notifications, partial fulfilment, fraud-review holds, and customer-initiated cancellation. Comparative studies under controlled fault injection, measuring latency, recovery time, operational effort, and correctness would enhance the evidence base.

Event schema governance at platform scale is another direction. Publish–subscribe decoupling can be used to allow independent evolution [9] but poorly governed events can be a hidden coupling

mechanism. There is a need for research in the following areas: event contract testing, semantic versioning, event lineage, compatibility check, and identification of unused and ambiguous events. These challenges are particularly critical when the retail experience is omni-channel as operational, analytical, marketing and partner facing systems consume events.

The next direction is retail edge awareness. In a network environment, some systems like store systems, mobile devices, kiosks and local fulfilment nodes may have to run under conditions of network instability. Research on edge computing shows that computation can be closer to the data source and the user [34,35] but there are not enough patterns for retail. Future frameworks should explore the issues around reconciling local availability estimates, store-picking events, offline point of sale transactions and edge caches while preserving auditability.

Lastly, future studies should establish metrics for architecture quality that are relevant to serverless event-driven retail. Average latency and invocation cost alone are insufficient cloud metrics. Some of the helpful metrics are: event staleness, inventory projection lag, compensation success rate, duplicate-event tolerance, schema compatibility violations, checkout critical-path latency, fulfilment promise accuracy, and mean time to trace cross-service incidents. Table 4 gives an initial list of measurable risks and controls, which needs to be empirically refined.

Table 1. *Literature comparison of peer-reviewed studies relevant to event-driven serverless omni-channel retail*

Ref.	Study focus	Method/system/model	Key findings	Limitation or research gap	Relevance to framework
[1]	Omni-channel retail transition	Conceptual retail analysis	Channel integration changes customer journeys and retail strategy	Limited technical architecture detail	Establishes channel integration requirement
[2]	Cross-channel integration and sales growth	Empirical retail analysis	Integration can support sales growth when channel resources are coordinated	Does not examine system-level event architecture	Supports business value of integrated services
[6]	Microservice architecture research	Systematic mapping	Decomposition, deployment, and communication dominate research themes	Empirical validation remains uneven	Supports domain-oriented decomposition
[8]	Microservice benefits and problems	Systematic review	Independent deployability and scalability are balanced by monitoring and testing difficulty	Evidence partly drawn from practitioner literature	Identifies operational risks
[9]	Publish–subscribe systems	Distributed systems survey	Publish–subscribe enables producer–consumer decoupling	Does not address retail semantics	Supports event-broker architecture
[10]	Event stream and complex event processing	Computing survey	Event streams support filtering, aggregation, and pattern detection	Application-specific governance remains external	Supports stream processing layer
[11]	Serverless opportunities and challenges	Computing survey	Elasticity and reduced management are offset by cold starts, state, and lock-in	Limited retail-specific analysis	Defines serverless adoption constraints
[14]	Serverless use cases	Review and characterization	Event-driven workloads are common serverless candidates	Use cases are not retail-specific	Guides function placement
[18]	Inventory availability integration	Empirical operations study	Reliable inventory information affects channel integration outcomes	Does not model event infrastructure	Supports inventory event projections
[19]	Buy-online-pick-up-in-store	Analytical operations model	Fulfilment design affects demand and	Abstracts away software	Supports fulfilment-aware

			inventory decisions	architecture	event design
[22]	Online fulfilment decisions	Operations modelling	Dynamic fulfilment decisions can improve online retail operations	Does not address microservice implementation	Supports event-driven allocation services
[29]	Cloud function performance	Experimental evaluation	Function performance varies across heterogeneous platforms	Not retail-specific	Informs serverless risk assessment

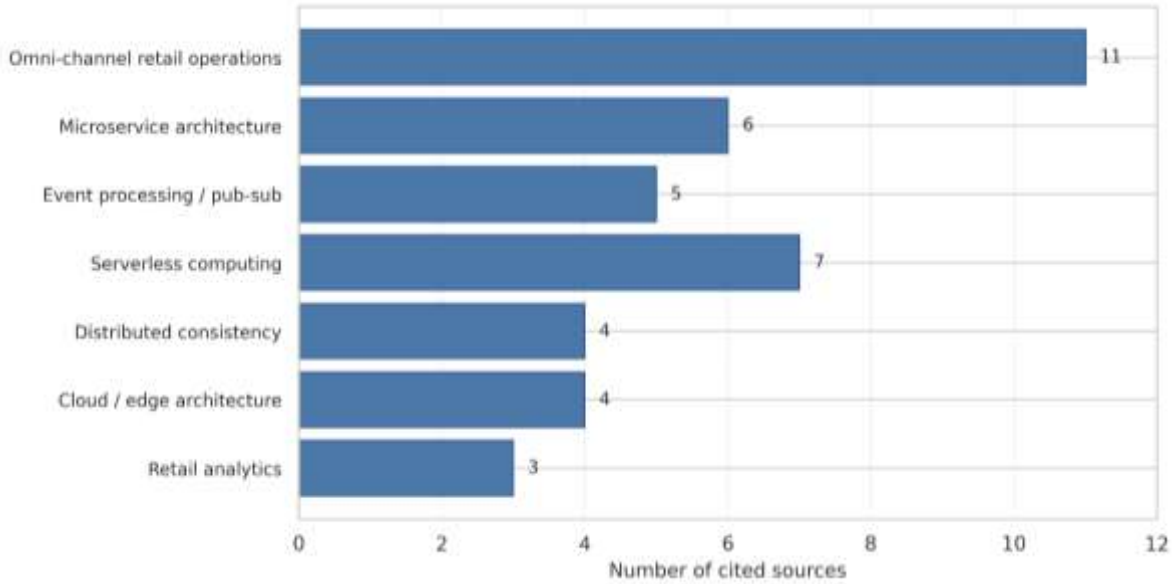


Figure 1. *Distribution of reviewed literature by technical domain*

Table 2. *Architecture requirements and evidence-supported design implications*

Requirement	**Retail driver**	**Architecture response**	**Evidence base**	**Principal risk**
Real-time inventory visibility	Channel switching, store pickup, fulfilment promises	Inventory events, reservation service, read projections	[18,19,22,23]	Projection staleness and overselling
Independent capability evolution	Frequent changes in pricing, promotions, fulfilment, loyalty	Domain-owned microservices with explicit event contracts	[6–8]	Distributed testing and coordination burden
Burst elasticity	Promotions, seasonal peaks, flash sales	Serverless event consumers for parallel non-core tasks	[11–14,29,30]	Cold starts and provider constraints
Cross-service workflow recovery	Payment, stock, fulfilment, return dependencies	Saga events and compensating actions	[15,27,28]	Incomplete compensation and audit gaps
Channel responsiveness	Mobile, web, marketplace, store latency expectations	CQRS-style projections and edge/cache policies	[16,18,34,35]	Stale reads and reconciliation complexity
Analytics reuse	Personalization, forecasting, operational dashboards	Governed event streams to analytical consumers	[24–26]	Semantic drift and uncontrolled data reuse
Operational traceability	Asynchronous functions, queues, external partners	Correlation IDs, tracing, dead-letter queues, lineage	[7,8,32,33]	Incident opacity

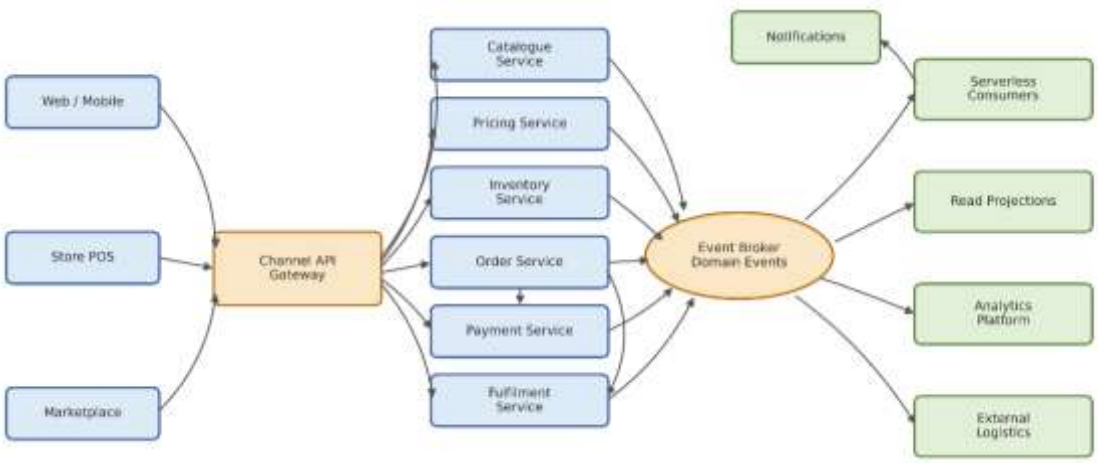


Figure 2. *Serverless event-driven microservice framework for omni-channel retail*

Table 3. *Pattern-level trade-offs for serverless event-driven retail architecture*

Pattern	Suitable retail use	Scalability effect	Consistency effect	Operational concern	Recommended use
Publish–subscribe broker	Order, inventory, fulfilment, notification events	High fan-out scalability	Depends on consumers and ordering guarantees	Schema governance and replay control	Core event backbone
Event sourcing	Inventory ledger, order state audit	Strong auditability	Requires projection reconciliation	Storage growth and event evolution	Selective use for auditable domains
CQRS/read projections	Channel availability, order tracking, customer dashboards	Improves query responsiveness	Read models may lag writes	Projection rebuild and staleness monitoring	Strong fit for channel views
Serverless functions	Notification, enrichment, webhook handling, image processing	Elastic per-event execution	Stateless unless externalized	Cold starts, tracing, lock-in	Best for bursty peripheral tasks
Saga choreography	Order fulfilment, returns, payment reversal	Avoids distributed locking	Compensation rather than atomic rollback	Complex failure tracing	Use with explicit saga state
Edge caching	Store apps, catalogue browsing, local availability	Reduces latency and central load	May expose stale values	Invalidation and offline reconciliation	Use for low-risk reads
Managed stream processing	Demand signals, fraud patterns, operational alerts	Continuous analytical processing	Not primary transactional authority	Backpressure and event-time handling	Use for derived insights

Table 4. *Failure modes, measurable indicators, and control mechanisms*

Failure mode	Observable indicator	Likely cause	Control mechanism	Priority metric
Oversell after stock reservation	Negative available-to-promise count	Stale projection or duplicate reservation	Idempotency key, reservation ledger, consistency tiering	Oversell incidents per 10,000 orders
Lost fulfilment update	Order status mismatch	Consumer failure or broker retention issue	Dead-letter queues, replay, consumer offset monitoring	Unresolved event lag
Duplicate customer notification	Repeated message delivery	At-least-once delivery without idempotent consumer	Idempotent notification service	Duplicate notification rate
Checkout	Increased p95 or	Synchronous	Critical-path	p95/p99 checkout

latency spike	p99 checkout duration	dependency chain or cold start	isolation, warm pools, async side effects	latency
Saga compensation failure	Order stuck in intermediate state	Missing compensating action or external timeout	Saga state machine, timeout events, manual exception queue	Compensation success rate
Event schema breakage	Consumer deserialization errors	Incompatible event change	Schema registry, contract tests, version policy	Schema compatibility violations
Projection staleness	Channel view lag	Backpressure or failed projection consumer	Lag alerts, replayable streams, bounded staleness policy	Projection lag in seconds
Trace discontinuity	Missing correlation path	Function or external partner lacks trace propagation	Correlation IDs and structured logs	Trace completeness ratio

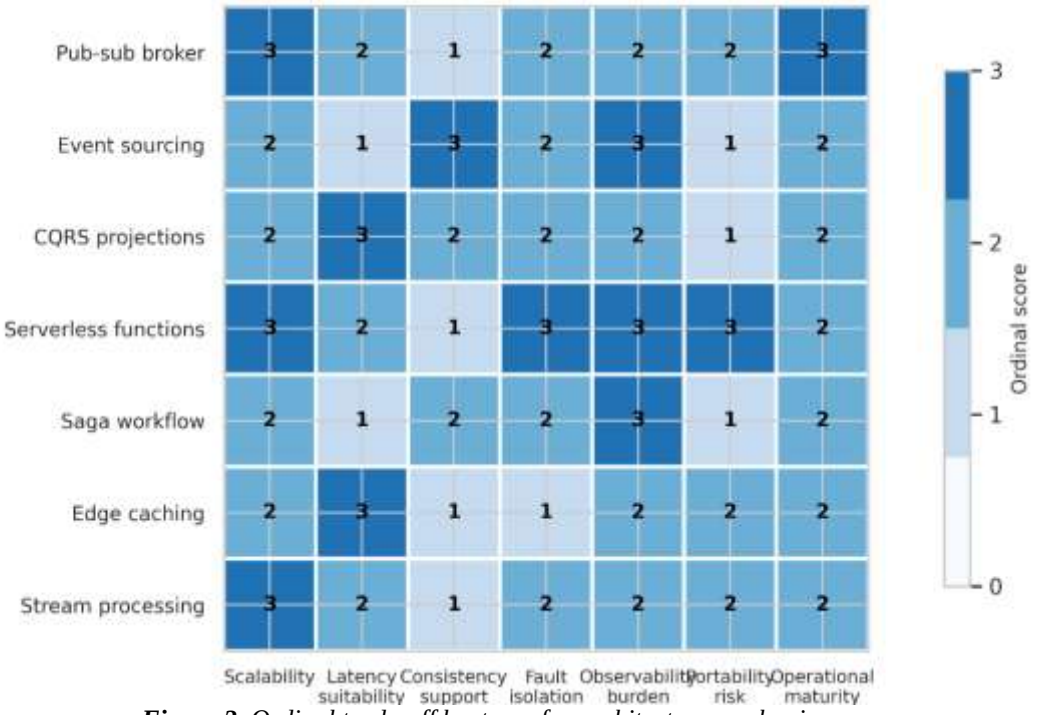


Figure 3. *Ordinal trade-off heatmap for architecture mechanisms*

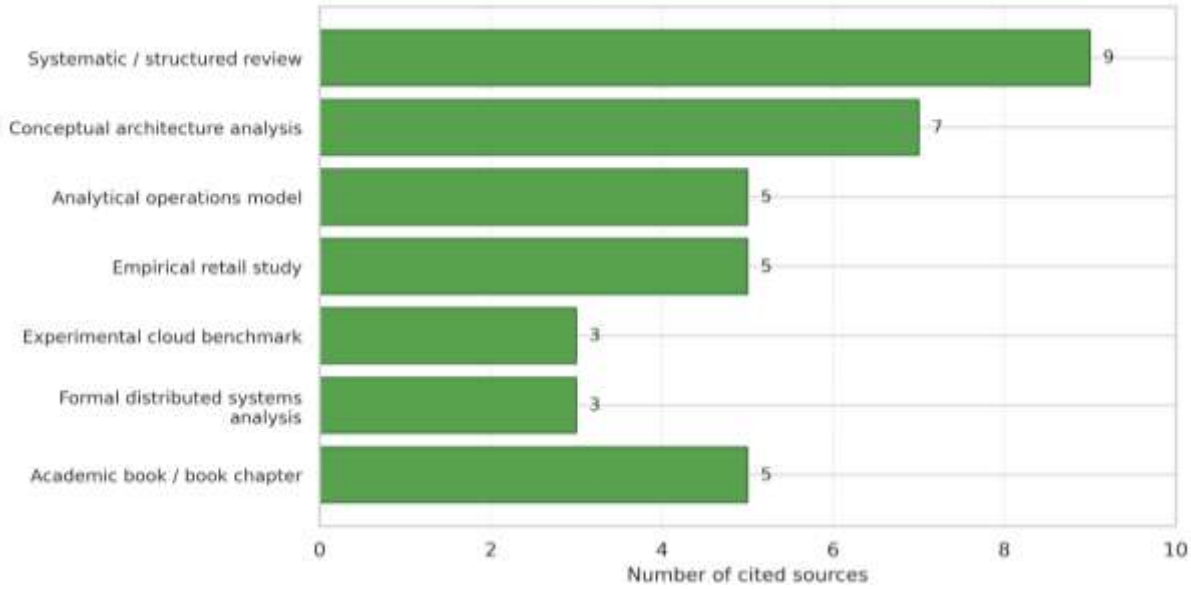


Figure 4. *Methodological distribution of the selected literature corpus*

6. Conclusions

Event-driven microservices and serverless computing offer a technically viable basis for scalable omni-channel retail, but the benefit comes from good architectural composition. The best use cases are ones in which the retail domain needs to respond to state changes without synchronous coupling, where workloads are burst or seasonal, and where channel-facing systems want to be able to get projections of the state in a timely fashion but not necessarily in a strongly consistent manner. Elastic event reactions, projection updates, notifications, enrichment, and peripheral workflow automation are all perfect use cases for serverless functions. They are not suitable for universal use as a substrate for stateful, latency critical or highly coordinated retail services.

The literature reviewed suggests that the following are architecture characteristics that are essential to success: domain-owned services, explicit compensation for sagas, observability throughout the asynchronous execution paths, tiered consistency of inventory, and selective placement of serverless functions. The central architectural challenge lies not only in introducing microservices, event brokers, or functions, but in integrating them into a coherent system. Correctness, traceability and semantic coherency must be maintained while retail systems can grow across channels, fulfilment nodes and customer interaction patterns.

Thus, a serverless event-driven retail system should be hybrid, governed and evidence-based. It should view business events as contracts, not as messages, it should view consistency as a domain policy, not a platform default, and it should view observability as a component of system design, instead of an afterthought. The next research phase aims to create repeatable retail workloads, formal consistency policies and measurable indicators of the quality of the architecture, to enable claims of scalability and resilience to be tested in a realistic omni-channel environment.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Verhoef, P. C., Kannan, P. K., & Inman, J. J. (2015). From multi-channel retailing to omni-channel retailing: Introduction to the special issue on multi-channel retailing. *Journal of Retailing, 91*(2), 174–181.

[2] Cao, L., & Li, L. (2015). The impact of cross-channel integration on retailers' sales growth. *Journal of Retailing, 91*(2), 198–216.

[3] Piotrowicz, W., & Cuthbertson, R. (2014). Introduction to the special issue information technology in retail: Toward omni-channel retailing. *International Journal of Electronic Commerce, 18*(4), 5–16.

[4] Hübner, A., Holzapfel, A., & Kuhn, H. (2016). Distribution systems in omni-channel retailing. *Business Research, 9*(2), 255–296.

[5] Saghiri, S., Wilding, R., Mena, C., & Bourlakis, M. (2017). Toward a three-dimensional framework for omni-channel. *Journal of Business Research, 77*, 53–67.

[6] Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *Journal of Systems and Software, 150*, 77–97.

[7] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software, 35*(3), 24–35.

[8] Soldani, J., Tamburri, D. A., & Van Den Heuvel, W. J. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software, 146*, 215–232.

[9] Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys, 35*(2), 114–131.

[10] Cugola, G., & Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys, 44*(3), Article 15.

[11] Shafiei, H., Khonsari, A., & Mousavi, P. (2022). Serverless computing: A survey of opportunities, challenges, and applications. *ACM Computing Surveys, 54*(11s), Article 239.

[12] Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N. J., Gonzalez, J. E., Popa, R. A., Stoica, I., & Patterson, D. A. (2019). Cloud programming simplified: A Berkeley view on serverless computing. *Communications of the ACM, 62*(9), 60–70.

[13] Li, Z., Guo, L., Cheng, J., Chen, Q., He, B., & Guo, M. (2022). The serverless computing survey: A technical primer for design architecture. *ACM Computing Surveys, 54*(10s), Article 220.

[14] Eismann, S., Scheuner, J., van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., Abad, C. L., & Iosup, A. (2021). A review of serverless use cases and their characteristics. *Journal of Systems and Software, 174*, 110906.

[15] Garcia-Molina, H., & Salem, K. (1987). Sagas. *ACM SIGMOD Record, 16*(3), 249–259.

[16] Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). The 8 requirements of real-time stream processing. *ACM SIGMOD Record, 34*(4), 42–47.

[17] Beck, N., & Rygl, D. (2015). Categorization of multiple channel retailing in multi-, cross-, and omni-channel retailing for retailers and retailing. *Journal of Retailing and Consumer Services, 27*, 170–178.

[18] Gallino, S., & Moreno, A. (2014). Integration of online and offline channels in retail: The impact of sharing reliable inventory availability information. *Management Science, 60*(6), 1434–1451.

[19] Gao, F., & Su, X. (2017). Omni-channel retail operations with buy-online-and-pick-up-in-store. *Management Science, 63*(8), 2478–2492.

[20] Kembro, J. H., Norrman, A., & Eriksson, E. (2018). Adapting warehouse operations and design to omni-channel logistics: A literature review and research agenda. *International Journal of Physical Distribution & Logistics Management, 48*(9), 890–912.

[21] Marchet, G., Melacini, M., Perotti, S., Rasini, M., & Tappia, E. (2018). Business logistics models in omni-channel: A classification framework and empirical analysis. *International Journal of Physical Distribution & Logistics Management, 48*(4), 439–464.

[22] Acimovic, J., & Graves, S. C. (2015). Making better fulfillment decisions on the fly in an online retail environment. *Manufacturing & Service Operations Management, 17*(1), 34–51.

[23] Gallino, S., Moreno, A., & Stamatopoulos, I. (2017). Channel integration, sales dispersion, and inventory management. *Management Science, 63*(9), 2813–2831.

[24] Grewal, D., Roggeveen, A. L., & Nordfält, J. (2017). The future of retailing. *Journal of Retailing, 93*(1), 1–6.

[25] Akter, S., & Wamba, S. F. (2016). Big data analytics in e-commerce: A systematic review and agenda for future research. *Electronic Markets, 26*(2), 173–194.

[26] Wamba, S. F., Akter, S., Edwards, A., Chopin, G., & Gnanzou, D. (2015). How big data can make big impact: Findings from a systematic review and a longitudinal case study. *International Journal of Production Economics, 165*, 234–246.

[27] Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News, 33*(2), 51–59.

[28] Bailis, P., & Ghodsi, A. (2013). Eventual consistency today: Limitations, extensions, and beyond. *Communications of the ACM, 56*(5), 55–63.

[29] Figiela, K., Gajek, A., Zima, A., Obrok, B., & Malawski, M. (2018). Performance evaluation of heterogeneous cloud functions. *Concurrency and Computation: Practice and Experience, 30*(23), e4792.

[30] Malawski, M., Gajek, A., Zima, A., Balis, B., & Figiela, K. (2020). Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions. *Future Generation Computer Systems, 110*, 502–514.

[31] Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., & Suter, P. (2017). Serverless computing: Current trends and open problems. In S. Chaudhary, G. Somani, & R. Buyya (Eds.), *Research advances in cloud computing* (pp. 1–20). Springer.

[32] Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley.

[33] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. *Communications of the ACM, 59*(5), 50–57.

[34] Varghese, B., & Buyya, R. (2018). Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems, 79*, 849–861.

[35] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal, 3*(5), 637–646.