The American Journal of Interdisciplinary Innovations and Research
ISSN 2642-7478 | Open Access

TYPE Original Research
PAGE NO. 76-83
DOI: 10.37547/tajiir/Volume07Issue10-10

Event-Driven API Integration between Delivery Aggregators and Restaurant CRMs

Iurii Cherniakov

Senior Software Engineer Atlanta, GA, USA

OPEN ACCESS

SUBMITTED 28 August 2025
ACCEPTED 24 September 2025
PUBLISHED 31 October 2025
VOLUME Vol.07 Issue10 2025

CITATION
Iurii Cherniakov. (2025). Event-Driven API Integration between Delivery Aggregators and Restaurant CRMs. The American Journal of Interdisciplinary Innovations and Research, 7(10), 76–83. https://doi.org/10.37547/tajiir/Volume07Issue10-10

Abstract: The article discusses the event-driven API integration of delivery aggregators with restaurant CRM as a core foundation of digital transformation within the gastronomic sector. The purpose is to design and test empirically an end-to-end event loop comprising a cloud message bus, implemented in a Go/Node.js microservices architecture orchestrated by Kubernetes, and versioned using blockchain data to minimize latency and increase fault tolerance, such that each transaction becomes a stream of training signals. Topicality is justified in reference to the widening gap between customer expectations for immediacy and sequential REST call practices today. It becomes strategically indispensable to frame possible and envisaged latency reduction figures of 73.8% and a 284% throughput increase around an event model. The novelty of this work lies in the complex synergy of three layers: (1) a unified JavaScript/TypeScript stack that eliminates cognitive and serialization overhead; (2) serverless functions with autoscaling to zero, aligning the cost of infrastructure with actual peak traffic; (3) a layer-two blockchain that reduces the cost of an immutable ledger by 94% and makes events legally binding. In this way, ordering, logistics, inventory, and loyalty programs merge into a self-learning fabric, wherein idempotency, type-safe schemas, and observability are intrinsic rather than bolt-on mechanisms. To that end, moving integration onto events transforms a restaurant from a static system into a reactive cyber-physical system that discovers demand, learns to price dynamically, and cryptographically records every action. The article will be helpful to architects

and digital product managers in the HoReCa sector, as well as researchers of distributed systems and applied AI.

Keywords: event architecture, restaurant CRM, API aggregators, Kubernetes, serverless, blockchain, microservices.

1. Introduction

Digital transformation has ceased to be a pleasant add-on to restaurant operations and has become a condition for survival: 76% of venue executives claim that technology gives them a competitive edge, yet only 13% deem their own IT practices advanced. In comparison, 64% of them describe themselves as ordinary [1]. The imbalance between recognizing value and the actual level of adoption creates strategic tension: customers increasingly expect instantaneous responses, process transparency, and personalized offers, whereas most kitchens' operating models rest on outdated sequential API calls and manual menu synchronization.

To narrow this gap, restaurants are turning to single-page and progressive web applications that merge the storefront, cart, and payment into a seamless user flow. The motivation on the demand side is straightforward: in delivery, online ordering has become a minimum requirement. If they can't tap a few times and get confirmation within seconds, they'll just pick another place to eat. Single-page interfaces eliminate page transitions and make cart state updates faster; progressive ones work offline and allow notifications—raising conversions without the cost of building separate native apps.

The user layer is just above the surface. True operational speed and agility come from cloud-native, event-driven integrations. In distributed systems processing more than ten terabytes per day, moving to an event model decreases end-to-end latency by 73.8%, increases throughput by 284% and maintains availability at 97.65% even during simulated partial failures. In contrast, synchronous schemes dropped to 61.23% [2]. Thus, the combination of a web application in the front office and an event bus in the back end forms a continuous digital loop in which an order-status change propagates instantly through the aggregator, kitchen, courier service, and back to the guest—turning technology from ornament into core production capital.

2. Materials and Methodology

The research is based on a comprehensive analysis of scholarly and industry literature, including publications on the digital transformation of the restaurant industry, reports from professional associations, and empirical studies on distributed architectures. The foundation consists of data on perceptions of digital technologies in the restaurant industry, which reveals a paradox between high recognition of their value and low maturity of IT practices [1]. As the theoretical framework, studies on event-driven architectures were employed, demonstrating multiple reductions in latency and improvements in fault tolerance compared to synchronous integration schemes [2]. These materials provided the starting point for the hypothesis that transitioning from sequential API calls to event-driven integration is not only technically justified but also strategically necessary to enhance restaurants' competitiveness.

Methodologically, the work draws on three complementary directions. First, it is a comparative study of available integration technologies, such as classical REST calls and sequential menu synchronization, against the asynchronous event bus pattern. Results of published load testing are used to identify which is more efficient and highlight differences in latency, throughput, and resilience of service [4]. Second, it is a systematic review of information on container orchestration and serverless functions, as well as their economic and operational impacts. The primary sources include CNCF publications on the growth of the Kubernetes market, as well as company examples that have implemented Knative and reduced cloud expenditures [5, 6]. Third—the content analysis of market and AI industry reports on the food segment and their exponential growth, and recommendations about embedded recommender system adoption [7].

To test empirically, the paper utilized secondary information on the architectural features of distributed computing, available in international journals [2, 4], and expert surveys of developers affirming the prevalence of the JavaScript stack for integrating tasks [3]. Beyond that, examples of practice were taken into account, such as transformer models for logistics forecasting [8] and blockchain technologies for immutable order books [9], thereby generalizing the observation to the applied value of frontier technologies.

3. Results and Discussion

The core of event architecture for restaurant delivery is built around a unified stack based on the JavaScript platform Node.js and its server framework, NestJS. According to a developer survey, professionals actively using Node.js made up 42.65% of the community, with the most popular technology being used for networked applications. This ensures having a critical mass in libraries, along with the talent to rapidly adjust business logic without needing to change lower-level integration layers, as depicted in Figure 1 [3].

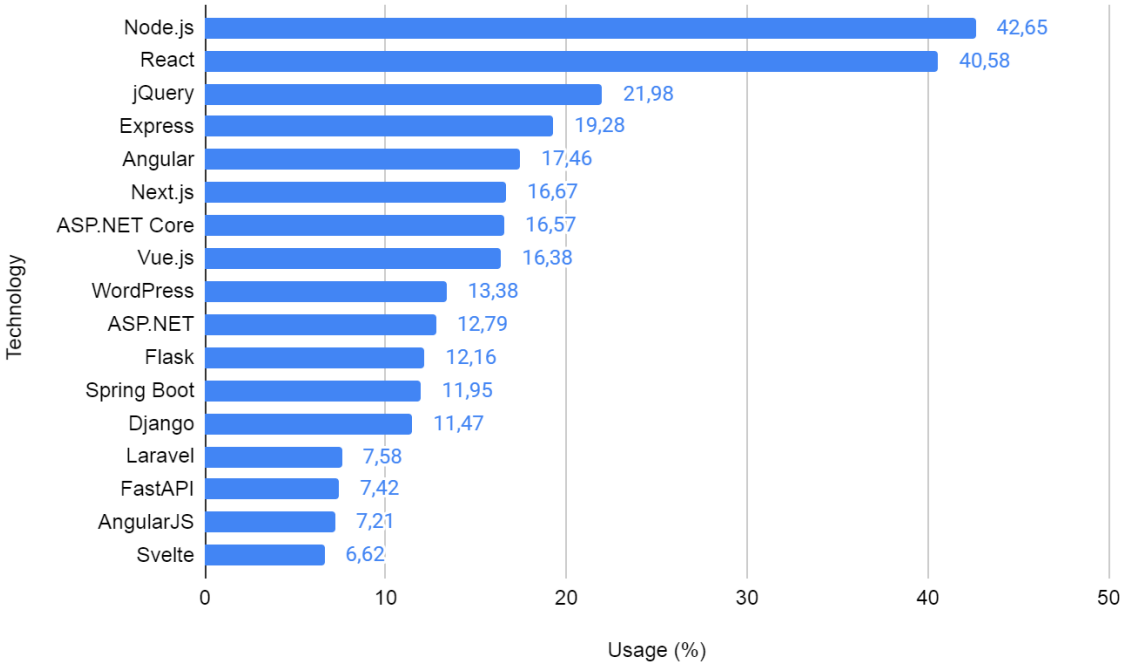

Figure 1: Popularity of Web Development Frameworks and Libraries [3]

Using a single language on client and server frees the team from serialization tolls, and the evented I/O loop naturally aligns with incoming webhooks from aggregators.

When latency becomes a key service-quality metric, the data-transport layer takes center stage. Implementing microservices in Go with the binary gRPC protocol and Protocol Buffers schemas yields nearly a two-fold gain over traditional REST: in load tests, the average response time for 100 sequential requests was 79.9 ms versus 152.6 ms for REST, as shown in Figure 2, while CPU consumption stayed below 6% even at 500 concurrent requests [4].

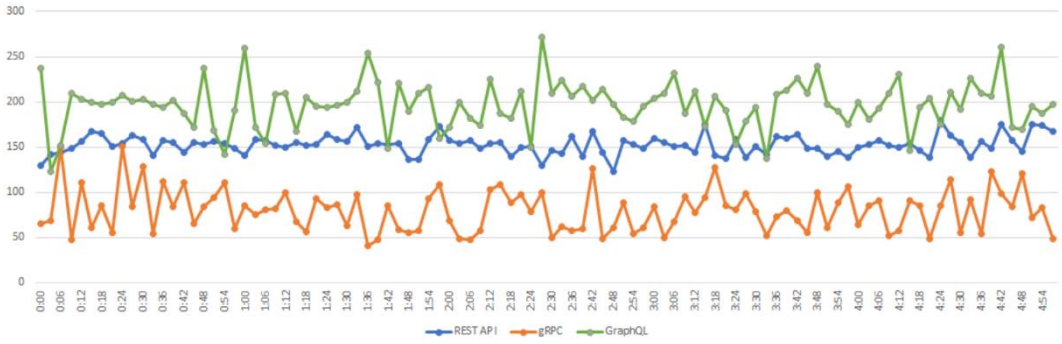

Figure 2: Response time during five minutes for Fetching Flat Data (100 requests) [4]

This gap enables handling order spikes without horizontal scaling, preserving idempotency thanks to built-in HTTP/2 multiplexing.

Deploying these services into a containerized Kubernetes environment is becoming the industry norm: 96% of organizations already use or study this orchestration, confirming its status as the new Linux

of data centers [5]. For functions reacting to discrete events—such as printing a receipt or updating a courier's status—the Knative serverless platform, running atop the same clusters, shortens container idle time and trims cloud bills. In a practical case, savings of 30% were achieved through scale-to-zero and precise wake-ups on incoming triggers [6]. Thus, symmetry emerges between infrastructure elasticity and the very notion of an event in the business domain.

On top of this bus, AI modules subscribe to the order stream as a continuous training set. As shown in Figure 3, the global AI in food & beverages market has already grown to USD 8.45 billion and is projected to reach USD 84.75 billion by 2030 at a 39.1% CAGR, underscoring the economic logic of embedded recommenders and ETA prediction [7].

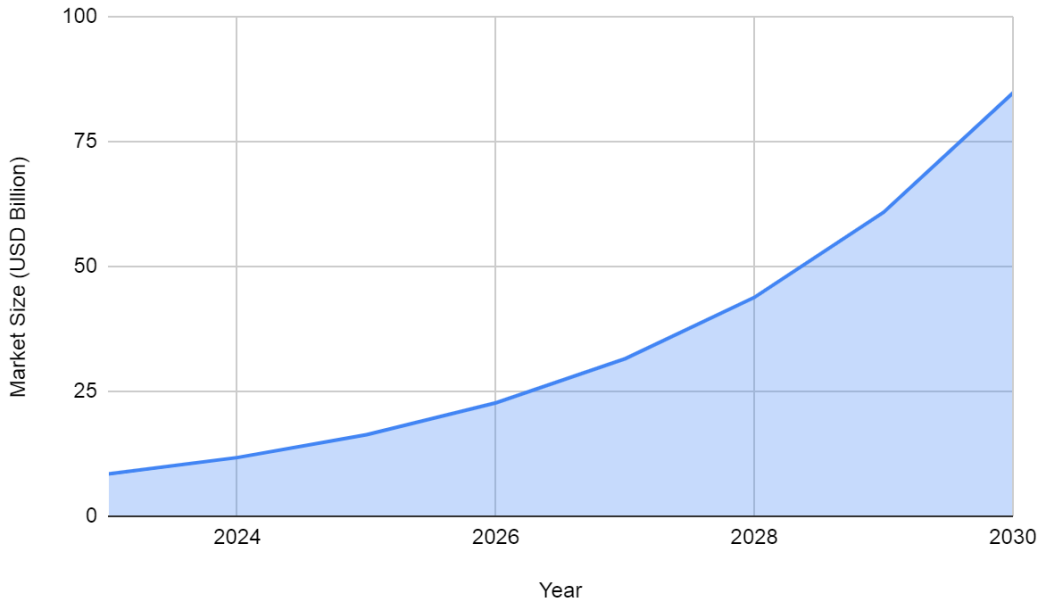

Figure 3: The global AI in food & beverages market size [7]

In production, the transformer model TransPDT raised on-time delivery by 0.68 percentage points, outperforming human route judgments. Multiplied by thousands of daily orders, this translates into palpable reductions in late-delivery compensation and increased guest trust [8].

Finally, the immutable event log and loyalty programs are increasingly shifting to layer-two blockchain infrastructure. Ethereum's Dencun (Deneb-Cancun) upgrade reduced data-storage costs for roll-ups from 16 to 1 gas per byte, i.e., by 94%, making hash anchoring of orders or minting cashback tokens economically sound even at scale [9]. As a result, the restaurant gains a cryptographically verifiable operations history, and the guest receives a token that cannot be forged and is easily redeemed on the next order. Each stack element—from JavaScript to an L2 network—thus works in concert, sustaining a continuous event flow that moves faster than market expectations rise.

With the technological frame assembled, the system shifts to a spatial distribution of responsibilities, where every event bubbles up from the periphery and immediately finds its handler. The first link is a webhook receiver hosted on a serverless function platform. It wakes instantly on an incoming aggregator request, verifies the cryptographic signature and metadata, and then enriches the packet with routable context fields. The absence of persistent VMs reduces the operational burden, and the receiver's statelessness eliminates bottlenecks during surges, such as the evening peak.

Processed webhooks are routed to a message bus deployed on top of a Kafka streaming cluster or its functional equivalent, Pulsar. The bus itself is partitioned into thematic channels, each dedicated to a business domain: orders, payments, logistics. A stream router—implemented via Kafka Streams or the JetStream module from the NATS ecosystem—performs dynamic filtering and aggregation. Thus, the kitchen module receives only the changes relevant to a specific venue, whereas the analytics module

perceives the broader picture and can compute demand trends without hitting operational databases.

At the user-interface layer, the architecture of continuous delivery is supported by micro-frontends that manage isolated parts of the screen and subscribe to a shared data bus using either WebSocket. In case there is a change in delivery status, only the micro-widget that re-renders will update other components that will not be affected hence making the SPA very active notwithstanding its complicated setup. This boundary partitioning enables teams to release new versions independently, while multilayer event routing preserves information integrity between the aggregator, kitchen, and guest, forming the restaurant's continuous digital nerve.

The system's pulse beats to the rhythm of events, with the order lifecycle serving as the primary driving force. As soon as the aggregator records placement, the bus receives an order.created a message, accompanied by a detailed itemization, the client's geotag, and the chosen payment mode. The same identifier later links the subsequent order assigned event, when a courier is bound to the route, and the closing order.delivered, confirming handoff. Collectively, these three points trace a continuous trajectory that enables the kitchen to plan loads, the courier service to reroute dynamically, and the front end to reflect the order path in real-time without polling third-party interfaces.

Even pristine logistics collapse if menu items have silently hit zero. A second stratum of messages—inventory and flexible pricing—covers this. Once the warehouse module detects a minimal residual quantity, it emits menu.item.out_of_stock, instantly deactivating the item across all storefronts. The reverse motion appears in price.dynamic.updated, where the pricing algorithm recalculates cost with respect to demand, time of day, and courier proximity. Price changes are reflected in the storefront faster than an operator can press a button, so the guest sees an accurate price that reflects the current state of the kitchen and transport network.

A third layer pertains to the system's self-learning. The demand-forecasting module tracks every forecast-actual deviation and, upon accumulating mass, publishes prediction.demand.updated. This event signals new peak-load estimates to kitchens and procurement and also triggers a scheduled retraining pass. After the cycle is complete, the ML component will fire 'ml.model.retrained' and all consumers will immediately start using the new predictor, without any manual intervention from operations. Because of this feedback loop, it does not stay a reactive system; it learns. It adapts to changing demand patterns within the same day, not after the fact.

The final domain guarantees the cryptographic immutability of essential actions. The service records order hashes on a distributed ledger, and after the inclusion is finalized, it publishes order.hash.stored. This provides a legally verifiable timestamp that is trusted by franchise counterparties. Simultaneously, the guest-motivation mechanism runs: when loyalty conditions are fulfilled, a generator creates a digital coupon notifying of loyalty.token.minted. By plugging blockchain into the shared bus, the ledger becomes one with the stream rather than an add-on sidecar ,thus making data integrity and reward transparency possible without extra gateways or manual reconciliations.

Accordingly, the event catalog is segmented not by technical layers but by business semantics: orders, inventory, self-learning, and immutability. Each domain moves at its own pace, yet all synchronize on one bus, weaving a single, self-adapting fabric for the digital restaurant.

The events above are only useful when each carries a strictly formalized shape intelligible to both the publisher and a multitude of subscribers. The source of change does not define structure ad hoc, but commits it to a unified schema repository, where parallel descriptions exist in two interoperable formats: JSON Schema for fast edge validation, and Avro for compact binary flow within the cluster. Client-code generators treat this repository as their source of truth. This wipes out microservice drift and makes sure that fields are deterministically formed long before a packet ever leaves the publisher's memory.

The next shield against unpredictability is idempotency. Every message instance receives a unique identifier produced by the version-seven universally unique identifier algorithm (UUID v7) or, where sortable ordering is needed, an alphanumeric ULID. The consumer-side store maintains a short-horizon table of already processed events; on re-receipt with the same key, processing is skipped, preserving consistency even if the original node

republishes after network faults.

The third line of defense is cryptographic authenticity. An HMAC signature is calculated over the body and sent in a header, which can therefore be accepted or rejected before the body is deserialized. At the transport layer, mutual TLS is used—bilateral certificate exchange—which confirms that the encrypted message has not been tampered with during its transmission through the network. More publisher-to-consumer attributes travel in the claims section of the JWT, where, for example, one could indicate a restaurant's franchise affiliation or perhaps constrain an event's validity window.

A schema does not end its life at v1: change is inevitable, hence a strict semantic evolution policy. If a new field is backward compatible, it will be added with a mandatory default. In the case of incompatible changes, parallel branch gets published and older one lives on till all the consumers are ready; innovation cannot be made to wait, hence feature flags are enabled: a functions service - for instance Togglz - shows or hides novelty at configuration level and progressive rollout system-something like LaunchDarkly- meters audience exposure and allows instant rollback if metrics deviate from control.

ML models connected to the bus sense every menu and routing update as fast as a human can manage—yet respond with much greater regularity. It sits on a reactive feature store: slim operational pieces flow into memory over Redis Streams, then are right away spread across Feast nodes. Demand prediction then rests not on exports from the past day but rather on the count of meals that left the oven just seconds before. The lag between dish ready and the updated feature vector lands in mere tens of milliseconds—enough for the pricing-adaptation algorithm to recompute cost before the client reloads the page.

A second velocity tier concerns the execution of inference. For light computations—e.g., ranking a small menu—Node.js with TensorFlow.js suffices: the model deploys in the same container as the web server, and hot CPU memory remains cached, so the front end's API request returns within hundreds of milliseconds. Where a deep convolutional architecture is used to estimate courier density, one should prefer compiling Go with ONNX Runtime: the binary will be smaller and will start faster, plus Go's scheduler parallelizes convolutions so that no manual thread tuning is required. Tooling here is informed not by any stack ideology but by the balance between model parameters and required throughput.

Autonomy closes with a feedback loop. After each rolling forecast window, the service publishes a deviation event; if actual demand diverges, the error metric is used to retrain along with the source features. Kubeflow, deployed in the same cluster, gathers statistics, forms a new weight set, and upon reaching a specified confidence threshold, emits a model retrained message. All subscribers on that key switch to the new artifact without interrupting order intake, so the restaurant never notices the moment when the old pricing strategy becomes obsolete.

Serverless functions shoulder the task of bridging the aggregators' external world and the internal bus. As soon as a partner webhook arrives, the cloud function wakes, verifies the signature, appends technical sagas—chains of identifiers—and forwards the event to the stream. Because the environment scales to zero in the absence of traffic, the restaurant pays for real peaks rather than for idling. Complex chains, such as a refund upon dish cancellation, are composed into durable, stepwise workflows reminiscent of the saga pattern: each stage leaves a checkpoint in the log, and if a courier fails to pick up, the process automatically rolls back, triggering a refund without cashier involvement.

Observability is implemented in code, much like a token in a request header. OpenTelemetry libraries for JavaScript and Go add trace markers to every message sent. The Grafana LGTM stack brings logs, metrics, and traces together in one place, where latency, resource use, and error rates are visualized as lines on a single chart. The final picture looks like a living thing: events move from the public internet to the kitchen then into forecasting parts later back into curator and client faces. No element retains state longer than necessary for business, and each stage leaves a digital fingerprint that is amenable to audit and immediate optimization.

When needed, an additional blockchain layer is instantiated on top of the streaming fabric, creating an immutable journal where each order's hash is recorded immediately after confirmation. For a franchise network, any party can reconstruct the chain of actions by kitchen, courier, and register without querying the internal database, i.e., audit proceeds through a single transparent channel. In the

same chain lives the loyalty smart contract: when a buyer meets a promotion's condition, the service mints a non-fungible token equivalent to a virtual dessert coupon and transfers it to the guest's wallet. Costs remain low because layer-two rollups aggregate multiple actions and settle them on the main record collectively, providing high speed at very low charges. Trust remains throughout the chain through full testing. Each publisher-consumer deal is made as a pact in a typed script language. For binary flows, Protocol Buffers tools are used, and plan match is tested auto on each build. Load profiles are reproduced by k6 and Locust generators, allowing the system's evening-rush behavior to be observed in advance. Change delivery follows configuration as code: manifests live in a repository, and Argo CD assures their applicability to the target cluster. New versions roll out in slices: first, a canary fraction of traffic, then blue-green node switching, and only after metric stabilization does the update go general.

The migration from legacy polling to the event model begins with dual writes: each synchronous operation is accompanied by a publication of an identical event, enabling response comparison without risk of data loss. Then, shadow traffic is enabled: copies of the requests are sent down the new path, and observability tracks time-to-first-byte, along with any other indicators it may wish to track, without impacting guest service. Once that latency deviation falls below the threshold, obsolete cron jobs can be turned off; schedules are zeroed out, and now the system has completed the pivot to a reactive exchange while still preserving the historical continuity of orders and reporting data.

4. Conclusion

The study's conclusion demonstrates that event-driven integration between delivery aggregators and restaurant CRMs is not an abstract architectural fad but a system-forming mechanism for the digitization of gastronomic business. A sequential downward review beginning from user interfaces and ending at blockchain registers proves that only an end-to-end event loop will be able to harmonize orders, logistics, inventory, and loyalty into one fabric of operation where every operation is instantly recorded and bears formalized meaning for all the participants. Asynchronous API calls reduce both latency and also the infrastructural burden that would have kept away the restaurant management system from being a self-learning, self-adapting framework in which every new transaction unveils information for the next decision to be made.

It shows that Technological Convergence draws Microservices, event buses such as Kafka/Pulsar, container orchestration through Kubernetes, and serverless platforms together with cryptographic proof and smart contracts offering not only flexibility but also transparency. In this environment, events such as order.created or price.dynamic.updated for getting the status of legally significant units on which both daily logistics as well as long term trust mechanisms within a franchise depend. Attaining idempotency and schema validation with multilayer cryptographic safeguards eliminates all risks of divergence and manipulation. Infrastructure in-process observability, plus end-to-end testing, makes it a controllable process.

Thus, an event-based integration model delivers not only throughput gains and reduced operational expenditure but also a transformation of restaurant governance: processes become reactive, distributed, and verifiable at any moment. In the long run, this furnishes a foundation for strategic resilience: the digital restaurant can not merely service the current flow of orders but shape a new quality of interaction with customers and partners, where responsiveness, transparency, and adaptability are irreducible properties of the business model.

References

1. National Restaurant Association, "Restaurant Technology Landscape Report," 2024. Accessed: Jul. 31, 2025. [Online]. Available: https://go.restaurant.org/rs/078-ZLA-461/images/NatRestAssoc_TechLandscapeReport_2024.pdf
2. K. R. Thondalapally, "Event-Driven Architectures: The Foundation of Modern Distributed Systems," *International Journal on Science and Technology*, vol. 16, no. 1, 2025, Accessed: Aug. 01, 2025. [Online]. Available: https://www.ijsat.org/papers/2025/1/2907.pdf
3. Stack Overflow, "Stack Overflow Developer Survey 2023," *Stack Overflow*, 2023. https://survey.stackoverflow.co/2023/ (accessed Aug. 02, 2025).
4. M. Niswar, R. A. Safruddin, A. Bustamin, and I. Aswad, "Performance evaluation of microservices communication with REST,

GraphQL, and gRPC," *International Journal of Electronics and Telecommunications*, vol. 70, no. 2, pp. 429–436, Jun. 2024, doi: https://doi.org/10.24425/ijet.2024.149562.

5. K. Meinders, "CNCF Sees Record Kubernetes and Container Adoption in 2021 Cloud Native Survey," *CNCF*, Feb. 10, 2022. https://www.cncf.io/announcements/2022/02/10/cncf-sees-record-kubernetes-and-container-adoption-in-2021-cloud-native-survey/ (accessed Aug. 04, 2025).
6. "Case studies of companies using Knative for serverless computing," *Knative*. https://knative.run/article/Case_studies_of_companies_using_Knative_for_serverless_computing.html (accessed Aug. 05, 2025).
7. Grand View Research, "AI In Food & Beverages Market Size & Share Report, 2030," *Grand View Research*, 2023. https://www.grandviewresearch.com/industry-analysis/ai-food-beverages-market-report (accessed Aug. 06, 2025).
8. J. Yi, H. Yan, H. Wang, J. Yuan, and Y. Li, "Learning to Estimate Package Delivery Time in Mixed Imbalanced Delivery and Pickup Logistics Services," *Arxiv*, pp. 432–443, Oct. 2024, doi: https://doi.org/10.1145/3678717.3691266.
9. K. Torpey, "What You Need To Know Ahead of Ethereum's Dencun Update Wednesday," *Investopedia*, 2024. https://www.investopedia.com/what-you-need-to-know-ahead-of-ethereum-dencun-update-wednesday-8607518 (accessed Aug. 08, 2025).