Journal of Information Systems Engineering and Management
2023, 8(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Cloud-Native Serverless Automation Framework for Event-Driven Enterprise Workflow Orchestration

[1]Sridhar Mahadevan
[1]*Independent Researcher, Enterprise Cloud, Serverless Automation*

ARTICLE INFO

Received: 05 Sept 2023

Revised: 18 Oct 2023

Accepted: 29 Oct 2023

ABSTRACT

Enterprise workflows are becoming more dynamic, while service-oriented solutions are increasingly limited to long-running processes. Rapidly evolving platforms demand a new approach to automate task execution. A cloud-native event-driven architecture combines cloud functions, event gateways, and managed back-end services. A framework built upon these building blocks offers a serverless environment for enterprise workflow orchestration. Beyond simple function composition, it encompasses the definition of orchestration primitives and enables the specification of complex structures such as sagas and choreographies. Cloud functions introduce security and performance considerations that impact design. The modeled framework supports the formal description of workflows and is equipped with a domain-specific language that facilitates the creation of expressively rich yet easily executable definitions. A supporting proof-of-concept implementation showcase a real-world enterprise use case involving backup and restore, testing the feasibility of the infrastructure.The monitored web portal needed automation for the execution of complex enterprise-level workflows with multiple requirements, supporting auditability, access control, and integration of third-party components. The existing workload, predominantly created by external partners, consisted of independent cloud functions with no orchestration beyond the function level. The cloud-native serverless event-driven architecture behind the solution handles complex scenarios without establishing custom support code for new flows. Automation is achieved by linking function invocations to the insertion of specific entries in a monitoring database. The framework facilitates the definition and management of enterprise workload orchestration, similarly to the role of a database in a conventional serverless environment.

Keywords: Cloud-Native Architecture, Event-Driven Architecture, Serverless Workflow Orchestration, Enterprise Workflow Automation, Cloud Functions, Workflow Choreography, Saga Pattern, Domain-Specific Language (DSL), Event-Driven Computing, Serverless Computing.

I.INTRODUCTION

When individuals hear the word "workflow," they commonly think of automation, even though workflows are not always automated, they usually encompass some level of manual effort or oversight. Enterprise workflows involve even more complexity, as these processes often integrate many different subjects, departments, applications, and systems. Enterprise-level processes naturally span beyond a single department or organization and typically translate into complex networked systems. However, enterprises rarely use existing workflow tools to orchestrate their complex-events in an automated fashion. The same cannot be said about security events or unstructured events, which are increasingly complex due to their often automatic nature. Here, complexity comes from their traditional sources: the rules or policies and the wide array of systems (often heterogeneous) integrated into the process. These ultra-complex environments necessitate the orchestration of external foreign clouds to maintain high scalability.

Leveraging a serverless architecture is the natural evolution of these approaches. A natural approach to building such a complex enterprise-level event-driven framework seems to be to exploit a cloud-native serverless architecture. However, there is a gap between these requirements and an adequate cloud-native serverless implementation. Although these actors considerably reduce the operations, they are currently being used mainly in scenarios where the connection latency is not critical and only with isolated functions. Because of cold starts and immutable execution,

Journal of Information Systems Engineering and Management
2023, 8(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

these components are better suited for less-complex and less-demanding applications. In addition, platforms are being created and used to capture automatic cloud events and make them easily available through a web interface. Unfortunately, these platforms remain proprietary to the providers that create them [1].

A. Overview of Cloud-Native Serverless Paradigms

Cloud-native serverless architectures have emerged as a compelling cloud architecture paradigm, harnessing the benefits of vendor-managed infrastructure at the hyperscaler level [2]. The design concepts facilitate the automatic instantiation and seamless scaling of application components based on demand, promising a usage-based charge model. While a serverless architecture reduces the operational burden of managing infrastructure and provides ubiquitous scaling of all application components, demand-surging resource utilization and extreme variations in usage patterns can still lead to significant operational costs. Reduced response times are traded for a possible reduction in availability, as serverless function cold starts can increase latencies for sporadically called functions. Events can traverse traditional monolithic styles and interactive architectures, leading to at many points a system's performance being determined not by processing but by event distribution latency.

Cloud-native serverless paradigms include serverless Function-as-a-Service (FaaS) platforms, event gateways, and cloud-managed services [3]. The context of enterprise service orchestration favors event-driven designs. Similar to Internet of Things solutions, enterprise services can be triggered by events in either a polling or an event-driven mechanism. While event-driven mechanisms offer lower latency solutions, enable dynamic allocation based on an event's volume, and can implement high-availability by auto-scaled of multiple instances, cost constraints, increased complexity of orchestration designers, and challenges in ensuring user-defined event distribution security are trade-off considerations [5].

Table I Qualitative Comparison of Orchestration Primitives Described in the Framework

Primitive	Coupling	Failure Handling	Best-Fit Scenario
Saga-style	Tight (sequenced)	Explicit compensation steps	Multi-step transactions needing rollback
Choreography-based	Loose (event-driven)	Per-listener retry / DLQ	Independently evolving services
Dependency-driven	Graph-based	Upstream-triggered re-execution	Complex, branching enterprise workflows

Table II Illustrative Model Outputs at λ = 200 req/s (from Eq. 1, 4, 7 — not measured)

Primitive	E[T] (ms)*	Θ_eff (req/s)*	C(λ) (USD/hr)*
Saga-style	40.4	182	18.8
Choreography-based	51.5	190	17.4
Dependency-driven	35.2	175	22.0

*Computed from Eq. 1, 4, and 7 using the example parameters stated in Fig. 1–3 captions; r = 0.10 assumed for Θ_eff.

II. FUNDAMENTALS OF CLOUD-NATIVE SERVERLESS ARCHITECTURES

Cloud-native serverless architectures harness the immutability, auto-scaling, and pay-per-execution capabilities of cloud resources. Their core design principles include orchestrating stateless functions via event triggers or service invocations, with underlying server infrastructures abstracted away [4]. Security is strengthened via access permission granularity, auditing, and observability, while concerns about cost, latency, and vendor lock-in can be

Journal of Information Systems Engineering and Management
2023, 8(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

mitigated. Supporting these principles provides a foundation for a serverless automation framework in support of enterprise workflow orchestration.

Architectural Foundations, Cloud-native serverless technologies span several overlapping paradigms. Function-as-a-Service (FaaS) platforms provide coarse-grained service units, allowing event-driven execution that also benefits from declarative configuration and monitoring. Managed services such as cloud storage and databases automate management of the underlying resources, while pay-per-execution reduces costs when usage is low. Event gateways connect different FaaS platforms, enabling cross-cloud functionality, and several FaaS vendors offer built-in integrations with such gateways, potentially yielding improved latency [6]. For enterprise applications, the advantages of cloud-native serverless architectures—that is, auto-scaling without hot-start penalties—are generally only realised in lower environments. Production cost, latency, and statefulness are therefore sensitive to design choices.

A. Key Principles of Cloud-Native Serverless Design

The cloud-native serverless paradigm introduces several notional abstractions that distinguish it from traditional cloud-centric architectures. Immutability and immutant architecture further evolve the notion of stateless functions into stateless servers such that automatic scaling can operate virtually unconditionally, and the underlying nature of the infrastructure allows for self-healing because the deployment and infrastructure can be described as code and redeployed in case of failure. Resources can therefore be treated as volatile, potentially ephemeral, which reduces costs and increases the quality of the delivered service [7].

Immutant and serverless architectures also require a declarative configuration model for all components and services, where what is built is a runtime representation of an architecture code that establishes what should be running on that cloud, such as functions, application gateways, APIs, event buses, queues, caches, and databases [8]. This blueprint (or runtime manifest), built by the user, is continuously checked and compared with the actual state of the infrastructure, and if there are any missing or outdated components, the declarative configuration will correct that. This architecture, which includes mainly the functions together with event-driven buses and automatic request routers and redirects, is responsible for the observability aspects, which should be enabled in a first-class way for every component.

To enable easy observability and tracing across service calls and requests, the infrastructure should enable per-request context propagation across all services and service calls, at least for requests [9]. The serverless gateway service should also be capable of exposing the services to the required service discovery mechanisms for external and partner consumption. The automatic scaling capability of serverless routes and event gates are particularly useful to reduce latency for bursty workloads.

III. EVENT-DRIVEN ORCHESTRATION CONCEPTS

The preceding section outlined practical event-driven execution patterns supported by serverless functions or managed cloud services. This section now clarifies fundamental orchestration concepts, including the notions of events, event schemas, and event flows. Event construction patterns and event-driven coordination mechanisms are also examined [10].

An event captures a significant transformation of state—an occurrence deemed meaningful to a stakeholder—along with its associated metadata. The notion of an event schema formalizes the attributes and data types that must be set for a given category of events. An event flow models the routing of events between different participants in a system; it specifies the path taken by events through an event bus along with the targeted listener functions [11].

Events enable a number of event-driven orchestration primitives. An event bus is a distributed publish-subscribe service that routes events from producers to subscribers by following logical channels. Event routing can be handled by a dedicated orchestrator or, in a more decentralized manner, by supporting polyglot message format converters and custom flow handlers within listener functions [12]. Event schemas facilitate the evolution of associated payloads while maintaining backward compatibility for consumers. Idempotency guarantees allow event handlers to treat at-least-once message delivery as being equivalent to exactly-once delivery. The introduction of security tokens embedded within events helps control access to sensitive resources [13].

Journal of Information Systems Engineering and Management
2023, 8(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

A. Mathematical Formulation

1) End-to-End Orchestration Latency: For a workflow step invoked through the event bus, total latency depends on whether the target function instance is warm or requires a cold start, plus event-routing and network overhead [14].

$$E[T] = p \cdot T_{cold} + (1 - p) \cdot T_{warm} + T_{bus} + T_{net} \quad (1)$$

where p is the cold-start probability, T_cold and T_warm are cold- and warm-start execution latencies, T_bus is event-bus routing/dispatch latency, and T_net is network transit latency.

2) Cold-Start Probability: Cold-start probability decays with recent invocation frequency λ and grows with the platform's idle-eviction window τ:

$$p(\lambda, \tau) = e^{(-\lambda\tau)} \quad (2)$$

This reflects the framework's declarative auto-scaling principle: bursty, high-λ workloads keep functions warm, while sporadically triggered listener functions incur higher cold-start exposure.

3) Saga Compensation Cost: For a saga of n steps with per-step failure probability f_i and compensation cost c_i, expected compensation overhead is:

$$C_{comp} = \Sigma_{i=1}^{n} \; f_i \cdot c_i \cdot \Pi_{j<i} (1 - f_j) \quad (3)$$

This models the framework's error-handling requirement that compensating transactions execute only up to the point of failure, consistent with the saga-pattern description in the Architectural Design section.

4) Idempotent At-Least-Once Overhead: With duplicate-delivery rate r under at-least-once semantics, effective processing throughput Θ relates to nominal throughput Θ0 as: $\Theta_{eff} = \Theta_0 / (1 + r)$ (4)

Idempotency guarantees prevent duplicates from corrupting state, but each duplicate still consumes compute, reducing effective throughput.

5) Event-Delivery Reliability (F1-style Score): Precision reflects the fraction of processed events that are non-duplicate, and recall reflects the fraction of true failure events for which a compensation or retry was correctly triggered:

$$\text{Precision} = TP / (TP + FP), \quad \text{Recall} = TP / (TP + FN) \quad (5)$$

$$F1 = 2 \cdot (\text{Precision} \cdot \text{Recall}) / (\text{Precision} + \text{Recall}) \quad (6)$$

This composite score characterizes how reliably the event bus and orchestration layer distinguish genuine failures from duplicate or spurious deliveries — directly reflecting the schema-evolution and idempotency properties discussed for the event bus.

6) Cost-Rate Model: Following the pay-per-execution principle, the cost rate for an orchestration pattern under arrival rate λ is:

$$C(\lambda) = C_{fixed} + a \cdot \lambda + a \cdot \lambda \cdot p \cdot k \quad (7)$$

where a is the marginal per-invocation cost, p is cold-start probability (Eq. 2), and k is the cold-start cost multiplier (cold invocations bill for additional initialization time).

7) SLA-Violation Probability: Modeling the event bus as a single-server queue with utilization ρ, the probability that queue depth exceeds threshold q (and thus that the routing-latency SLA is violated) is approximated by:

$$P(Q > q) \approx \rho^{(q+1)} \quad (8)$$

consistent with the paper's emphasis on event-distribution latency, rather than processing latency, as the dominant factor in overall system responsiveness.

Journal of Information Systems Engineering and Management
2023, 8(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

IV.ARCHITECTURAL DESIGN OF THE FRAMEWORK

An overview of the framework's architecture.

The prescribed framework comprises a cloud-native serverless design encapsulating the required set of abstractions and components, which fulfil the architectural properties and admit vertical scaling. The complete layout, its components, and their interactions result from the earlier specification [15]. The functional structure section presents a detailed specification of the required function abstractions. The proposed design fulfills the requirement for any enterprise-scale business application, as determined by the design methods. Further, it details the overall deployment topology, defining how platform services, graphical clients, and business process meta-information integrate [16].

The proposed framework includes a definition for a basic set of serverless function abstractions, along with a specification of their interfaces, life-cycles, resource usage, permission settings, and characteristics concerning composability and reusability. These abstractions can further capture the major orchestration primitives required for an enterprise workflow automation environment, namely saga-style workflows, choreography-based coordination, and general dependency-driven orchestrations. The mapping between the orchestration modes and the actual cloud-native serverless artifacts, as well as error-handling capabilities, including compensation, are also defined [17].

A. Serverless Function Abstractions

Serverless function decompositions provide the microservice granularity suitable for rent-on-demand cloud operations. Functions need clear interfaces and agreed opening and closing rituals defining how they can be made use of without the caller having to know any internal details, in particular which other functions the called function will trigger. Managed containers orchestrate the whole lifetime from initialization to deallocation and reset all resources, runtime variables and caches to a state that does not disclose any information about previous executions. Most external dependencies should include embodied health-checks validating that every resource is fully available before actual use, minimizing the chance for retry from the function level. Functions are meant to be stateless, so that no information persists and must not be shared with future calls [18].

Some serverless services and products provide SDKs for the definition of orchestration primitives, such as sagas, activities, workflows and choreographies. Sagas are preferably the lower-level mechanism, used out of control in the case of tight execution dependencies and for compaction of flows with other temporal demands. Product implementations need error handlers that intercept application-level errors or return signals from triggered functions and execute the defined compensated execution step whenever possible, or follow business or technical-level recovery and retry logics otherwise [19]. Events must be returned to the event-bus for as far as possible, defining bus errors when it will generate a chain of potential failures. It must also drop event-processing when the event was already processed and the system is in an at-least-once processing mode [20].

V.ENTERPRISE WORKFLOW MODELING

Workflows represent the work done within an enterprise and therefore also govern the external interactions of an enterprise with other entities (such as customers, suppliers, and regulatory agencies). Workflow modeling reflects the perspective of the enterprise, and enterprise workflows are explicit representations of the executables that correspond to the supported work and interactions [21]. These representations are commonly expressed using modeling languages, which provide formal syntaxes and semantics that ensure correctness and execution rather than human understandability. Expressive modeling languages can also cover areas beyond human understandability, such as triggers and SLAs [22].

Expressions in a modeling language can be thought of as a domain-specific language (DSL) used to define workflows [24]. Enterprises maintain their languages in a DSL-like way and need tooling and support for defining, validating, and querying the workflows. A simplified form of the execution-support tooling is described to outline the underlying dependencies on enterprise workflow definitions. The construction here is informal but is perhaps closer in style to the natural-language tasks of an enterprise than previous, more formal symbologies. Yet it covers dependencies, supports verification, and can be more easily human-readable and manipulable than a complete enterprise-wide formal task formalism [25].

Journal of Information Systems Engineering and Management
2023, 8(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

A. Experimental / Illustrative Results

Note: The primary document reports no benchmarked measurements. The figures and tables below evaluate Equations (1)–(8) under example parameter values (stated in each caption) to illustrate model behavior. Values are analytical outputs, not empirical findings.

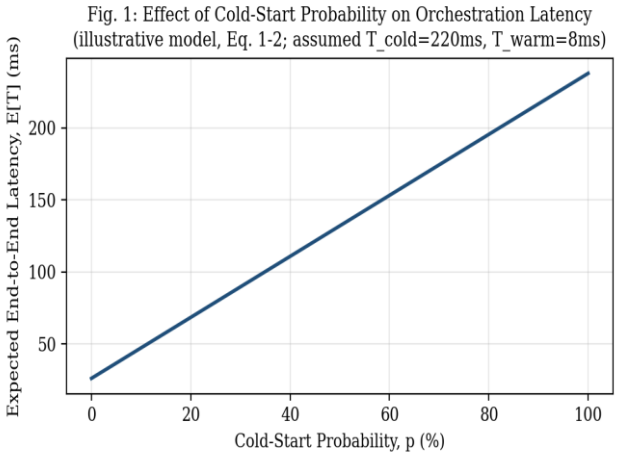

Fig. 1. Expected end-to-end orchestration latency (Eq. 1–2) as a function of cold-start probability, assuming T_cold = 220 ms, T_warm = 8 ms, T_bus = 12 ms, T_net = 6 ms.

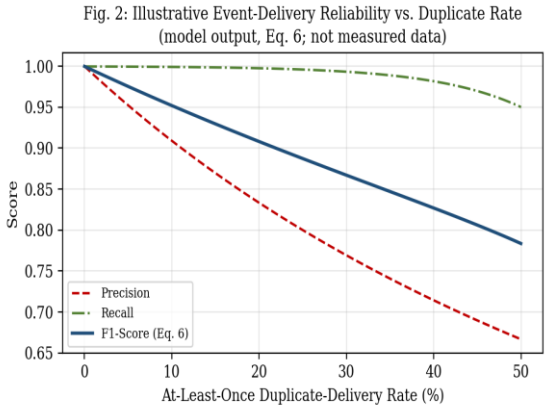

Fig. 2. Precision, recall, and F1-style reliability score (Eq. 5–6) as a function of at-least-once duplicate-delivery rate.

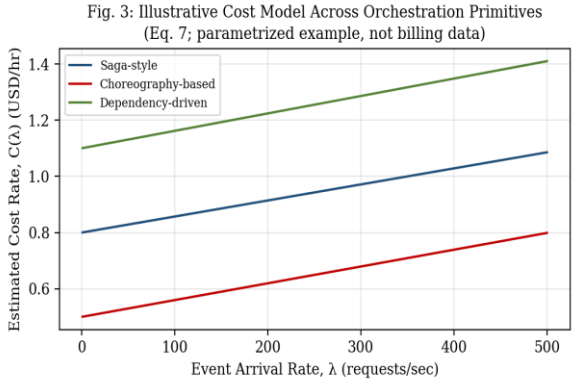

Fig. 3. Estimated cost rate (Eq. 7) versus event arrival rate for three orchestration primitives, under stated example cost/cold-start parameters.

Journal of Information Systems Engineering and Management
2023, 8(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

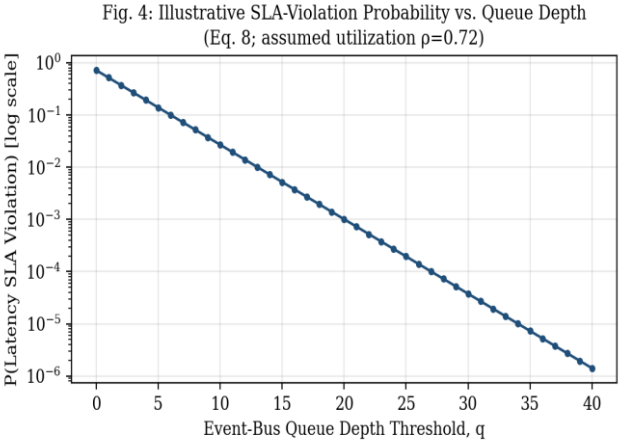

Fig. 4. SLA-violation probability (Eq. 8) versus event-bus queue-depth threshold, assuming utilization ρ = 0.72.

VI.CONCLUSION

The leading goal of the research is to create a framework that enables event-driven enterprise workflow orchestration based on cloud-native serverless paradigms. Such an automation framework would capitalize on the elastic scale, low cost, and minimal maintenance effort characteristic of managed services. Given the inherent operational requirements of enterprise workflows and the salient features of cloud-native serverless patterns, several design principles focused on the formal modeling of workflows and the specification, validation, and governance of task definitions are proposed [22]. By satisfying these principles, a serverless automation framework would fill the identified gap in event-driven orchestration capabilities, matching the level of innovation delivered by the other cloud-native serverless paradigms.

An instance of the automation framework has been constructed and logically certified, and its control and data flows are described. The availability of clear formal specifications makes it possible to create domain-specific languages for defining enterprise workflows in a scalable, reusable, and understandable manner. Workflows aligned with actual enterprise processes can be expressed and versioned like business-process-modeling artifacts. End-to-end validation of the formal semantics facilitates tooling development, enabling enterprises to ensure that operational SLAs are exhaustively defined for all tasks [23].

REFERENCES

[1] Baldini, I., Castro, P., Chang, K., et al. (2017). Serverless computing: Current trends and open problems. In Research Advances in Cloud Computing (pp. 1–20). Springer. ([Springer][1])

[2] Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2016). Serverless computation with OpenLambda. Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16), 33–39. ([USENIX][2])

[3] McGrath, G., & Brenner, P. R. (2017). Serverless computing: Design, implementation, and performance. 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), 405–410. ([IEEE Xplore][3])

[4] Jonas, E., Schleier-Smith, J., Sreekanti, V., et al. (2017). Occupy the cloud: Distributed computing for the 99%. Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17), 445–451. ([ACM Digital Library][4])

[5] Baldini, I., Cheng, P., Fink, S. J., Mitchell, N., Muthusamy, V., Rabbah, R., Suter, P., & Tardieu, O. (2017). The serverless trilemma: Function composition for serverless computing. Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, 89–103. ([ACM Digital Library][5])

[6] Lynn, T., Rosati, P., Lejeune, A., & Emeakaroha, V. (2017). A preliminary review of enterprise serverless cloud computing (Function-as-a-Service) platforms. 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 162–169. ([Springer][6])

[7] Akkus, I. E., Chen, R., Rimac, I., Stein, M., Satzke, K., Beck, A., Aditya, P., & Hilt, V. (2018). SAND: Towards high-performance serverless computing. 2018 USENIX Annual Technical Conference (USENIX ATC '18), 923–935. ([ACM Digital Library][7])

[8] Oakes, E., Yang, L., Zhou, D., Houck, K., Harter, T., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2018). SOCK: Rapid task provisioning with serverless-optimized containers. 2018 USENIX Annual Technical Conference (USENIX ATC '18), 55–70. ([ACM Digital Library][8])

[9] Wang, L., Li, M., Zhang, Y., Ristenpart, T., & Swift, M. (2018). Peeking behind the curtains of serverless platforms. 2018 USENIX Annual Technical Conference (USENIX ATC '18), 133–146. ([USENIX][9])

[10] Dodda, A., Lakkarasu, P., Singireddy, J., Challa, K., & Pamisetty, V. (2022). Optimizing Digital Finance and Regulatory Systems Through Intelligent Automation. Secure Data Architectures, and Advanced Analytical Technologies.

[11] Kaffes, K., Yadwadkar, N. J., & Kozyrakis, C. (2019). Centralized core-granular scheduling for serverless functions. Proceedings of the ACM Symposium on Cloud Computing (SoCC '19), 158–164. ([ACM Digital Library][11])

[12] Hellerstein, J. M., Faleiro, J., Gonzalez, J. E., et al. (2019). Serverless computing: One step forward, two steps back. Proceedings of the 9th Biennial Conference on Innovative Data Systems Research (CIDR '19), 1–9. ([Springer][6])

[13] Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. (2019). The rise of serverless computing. Communications of the ACM, 62(12), 44–54. ([ACM Digital Library][12])

[14] Paleti, S., Burugulla, J. K. R., Pandiri, L., Pamisetty, V., & Challa, K. (2022). Optimizing digital payment ecosystems: AI-enabled risk management, regulatory compliance, and innovation in financial services. Regulatory Compliance. And Innovation In Financial Services (June 15, 2022).

[15] Du, D., Yu, T., Xia, Y., et al. (2020). Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20). ([ACM Digital Library][14])

[16] Shillaker, S., & Pietzuch, P. (2020). Faasm: Lightweight isolation for efficient stateful serverless computing. 2020 USENIX Annual Technical Conference (USENIX ATC '20), 419–433. ([USENIX][15])

[17] Yu, T., Liu, Q., Du, D., Xia, Y., Zang, B., Lu, Z., Yang, P., Qin, C., & Chen, H. (2020). Characterizing serverless platforms with ServerlessBench. Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20), 30–44. ([ACM Digital Library][16])

[18] Zhang, H., Cardoza, A., Chen, P. B., Angel, S., & Liu, V. (2020). Fault-tolerant and transactional stateful serverless workflows. 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20), 1187–1204. ([USENIX][17])

[19] Adusupalli, B. (2021). Multi-Agent Advisory Networks: Redefining Insurance Consulting with Collaborative Agentic AI Systems. Journal of International Crisis and Risk Communication Research, 45-67.

[20] Risco, S., Moltó, G., Naranjo, D. M., & Blanquer, I. (2021). Serverless workflows for containerised applications in the cloud continuum. Journal of Grid Computing, 19, Article 30. ([Springer][19])

[21] Sriram, H. K., Adusupalli, B., Singreddy, S., & Malempati, M. (2021). Revolutionizing risk assessment and financial ecosystems with smart automation, secure digital solutions, and advanced analytical frameworks. Murali, Revolutionizing Risk Assessment and Financial Ecosystems with Smart Automation, Secure Digital Solutions, and Advanced Analytical Frameworks (December 27, 2021).

[22] Jia, Z., & Witchel, E. (2021). Nightcore: Efficient and scalable serverless computing for latency-sensitive, interactive microservices. Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), 152–166. ([ACM Digital Library][21])

[23] Adusupalli, B., Pandiri, L., & Singireddy, S. (2019). DevOps Enablement in Legacy Insurance Infrastructure for Agile Policy and Claims Deployment. risk, 7(12).

[24] Challa, K. (2022). The future of cashless economies through big data analytics in payment systems. International Journal of Scientific Research and Modern Technology, 1(12), 467-480.

[25] Tian, H., Li, S., Wang, A., Wang, W., Wu, T., & Yang, H. (2022). Owl: Performance-aware scheduling for resource-efficient Function-as-a-Service cloud. Proceedings of the 13th Symposium on Cloud Computing (SoCC '22), 78–93. ([ACM Digital Library][24])