Empirical Analysis of Cloud-Edge Infrastructure Complexity: Practitioner Pain Points and Architectural Directions

Pawissanutt Lertpongrujikorn[1], Hai Duc Nguyen[2], and Mohsen Amini Salehi[1]
[1]High Performance Cloud Computing (HPCC) Lab, University of North Texas (UNT), USA
[2]Argonne National Laboratory and University of Chicago, USA
{pawissanutt.lertpongrujikorn, mohsen.aminisalehi}@unt.edu, hai.nguyen@anl.gov

***Abstract*—The proliferation of cloud, edge, and Internet of Things (IoT) computing has created unprecedented opportunities for distributed applications. However, this architectural shift introduces profound infrastructural complexity, acting as a significant barrier to developer productivity and innovation. In this paper, we present an empirical analysis based on 101 semi-structured interviews across 86 organizations to investigate the state of cloud-native development practices, pain points, and expectations. Our findings quantitatively validate that deployment complexity (38.6%) and onboarding difficulty (35.6%) are the dominant operational bottlenecks, while developers heavily prioritize productivity (53.5%) and automation (44.6%) over raw performance optimization. Based on these empirical insights, we examine four architectural directions that address the validated pain points: unified object abstractions (Object-as-a-Service), platform engineering via Internal Developer Platforms, declarative AI/ML serving pipelines, and lightweight edge runtimes based on WebAssembly. Furthermore, we detail the multi-stakeholder ecosystem required for adopting novel infrastructure paradigms, emphasizing that security, operational integration, and strict multi-tenant isolation are prerequisites for production readiness. Our results demonstrate that the primary barrier to distributed computing adoption is not execution performance but infrastructural complexity, and that declaratively governed, higher-level abstractions across multiple paradigms offer viable architectural paths toward alleviating it.**

***Index Terms*—Cloud Computing, Edge Computing, Serverless, Platform Engineering, AI Infrastructure, WebAssembly, Orchestration, Empirical Study, Developer Experience**

I. Introduction

The evolution of computing from centralized cloud environments to highly distributed cloud-edge continuums has expanded the capability of modern applications [1], [2]. Yet this capability comes at the cost of profound infrastructural complexity: managing microservice architectures [3], orchestrating state and compute, and configuring fragmented cloud services burden developer productivity [4], with practitioners managing an average of 14 vendor tools and facing 100-day onboarding cycles [5].

This burden is amplified by emerging workloads. AI and machine learning systems add model versioning, GPU scheduling, and inference-pipeline management [6], [7], while platform engineering reflects industry recognition that developer-facing abstractions remain insufficient. These trends suggest that complexity is not confined to one technology but is systemic across modern distributed infrastructure.

While previous work has proposed novel architectures to address individual challenges—such as cold starts in serverless platforms [8], [9] or data replication in edge networks—there remains a critical lack of empirical research examining the holistic pain points experienced by practitioners operating these modern infrastructures. To address this gap, we conducted an extensive customer discovery and empirical validation study involving 101 interviews across 86 organizations, following the NSF I-Corps National program methodologies.

In response to these findings, we examine architectural directions that address validated pain points through higher-level abstractions and declarative governance. This paper makes three contributions: **C1. Empirical grounding.** We present a rigorous empirical analysis identifying deployment complexity and onboarding difficulty as the primary barriers to distributed infrastructure adoption. **C2. Expectation gap.** We quantify the gap between practitioner expectations (prioritizing productivity and automation) and the current state of infrastructure orchestration. **C3. Architectural mapping.** We map the empirical findings to four architectural directions—unified object abstractions (OaaS), platform engineering via Internal Developer Platforms (IDPs), declarative AI/ML serving pipelines, and lightweight edge runtimes (WebAssembly)—and analyze the multi-stakeholder adoption ecosystem, showing that security, operational integration, and multi-tenant isolation are prerequisites for production readiness.

The remainder of this paper is organized as follows. Section II outlines methodology and demographics; Section III reports practitioner challenges and expectations; Section IV maps findings to architectural directions; Section V discusses adoption and production readiness; Section VI reviews related work; and Section VII concludes.

II. Methodology

A. Customer Discovery Approach

We adopted the evidence-based customer discovery framework prescribed by the NSF I-Corps program [10], which emphasizes getting out of the building to test hypotheses about customer problems, needs, and workflows. Our approach consisted of semi-structured interviews designed to explore

current workflows, pain points, attempted solutions, and desired outcomes rather than pitching our specific technology outright.

B. Interview Protocol

Each interview followed a consistent protocol: **Duration:** 30+ minutes per interview; **Format:** semi-structured conversations via video conference, phone, or in-person; **Focus areas:** current cloud/edge-cloud development workflows, deployment practices, primary challenges (technical and organizational), performance and cost concerns, prior attempted solutions, and expectations for improvement; and **Data capture:** participant roles, company industry and size, coded thematic data on specific pain points and desired outcomes, and any quantitative metrics mentioned regarding time savings or performance improvements.

Following each interview, we conducted systematic thematic coding to identify recurring pain points and expectations for improvement. We categorized pain points into eleven primary themes: deployment complexity, onboarding difficulty, system complexity, serverless-specific concerns, security concerns, integration challenges, observability limitations, cost management, scaling issues, documentation gaps, and responsiveness issues. We separately categorized expectations for improvement into thirteen dimensions. This dual coding approach—capturing both current pain points and desired future states—enabled us to validate the problem-solution fit between market needs and edge-cloud orchestration capabilities.

C. Participant Demographics

Over the course of the study, we conducted 101 interviews that spanned 86 unique organizations (some organizations had multiple interviewees).

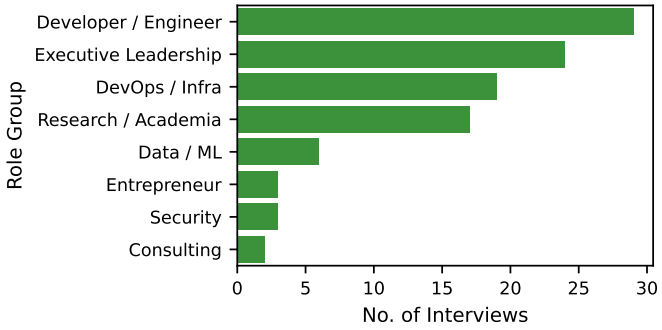

Fig. 1. Distribution of interview participants by role group (N=101 interviews). Developer/Engineer and DevOps/Infra roles comprised nearly half of all interviews, reflecting our focus on technical practitioners directly involved in cloud-native development and deployment.

Role Distribution: As shown in Fig. 1, participants spanned diverse technical and leadership roles. Table I summarizes the distribution of interview participants by role category.

Industry coverage spanned seven sectors led by Education & Academia, Software & Technology, and Financial Services; company sizes ranged from fewer than 10 to over 10,000 employees, ensuring coverage across operational scales.

TABLE I
DISTRIBUTION OF INTERVIEW PARTICIPANTS BY ROLE CATEGORY (N=101)

Role Category	#Interviews	Specific Roles
Dev / Engineer	29	Software Eng, Full-stack Devs, Product Eng, QA Eng
Exec Leadership	24	Managers, Founders, CEOs, CTOs, VPs of Eng, PMs
DevOps / Infra	19	DevOps Eng, Infra Eng, SREs, IT Administrators
Research / Academia	17	Professors, Researchers, Research Scientists, PhDs
Data / ML	6	Data Scientists, ML Eng, Research Scientists
Security	3	Security Eng, DevSecOps
Entrepreneur	3	Independent start-up founders
Consulting	2	Technical consultants

III. Empirical Findings: State of the Continuum

Our empirical analysis revealed a consistent set of critical, data-supported challenges that quantitatively confirm a significant operability gap within modern cloud-native infrastructures.

A. Pervasive Complexity and Operational Overhead

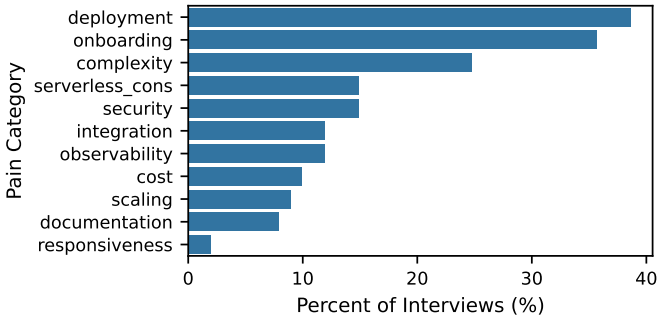

Fig. 2. Frequency of pain points mentioned across 101 interviews. Deployment complexity and onboarding difficulty dominated.

Fig. 2 summarizes the frequency of pain points across all 101 interviews. The top challenges were: **1) Deployment complexity (38.6%).** Fragmented tooling, configuration sprawl, and orchestration across compute, storage, and networking forced practitioners to spend significant time on "glue code" and infrastructure-as-code templates. **2) Onboarding difficulty (35.6%).** Steep learning curves, cloud-specific APIs, and long time-to-productivity created a significant barrier to entry. **3) System complexity (24.8%).** The cognitive load of managing microservices dependencies, distributed state, and failure modes overwhelmed many teams. **4) Security concerns (14.9%).** Security configuration, credential management, and compliance added overhead, especially in regulated industries with zero-trust requirements [11]. **5) Serverless-specific concerns (14.9%).** Cold start latency, execution time limits, and vendor lock-in in Function-as-a-Service platforms [12], [13] hindered performance guarantees. **6) Cost management**

(9.9%). Unpredictable costs and opaque cost-performance trade-offs made it difficult to correlate resource usage with business value.

Less frequent concerns included integration challenges (11.9%), observability limitations (11.9%), scaling issues (8.9%), and documentation gaps (7.9%).

Importantly, these pain points often co-occurred. Our correlation analysis revealed that complexity and onboarding challenges frequently appeared together, identifying the phenomenon of *cognitive overload*—where engineers possess the tools to solve a problem, but orchestrating the tools effectively exceeds reasonable human capital limits.

B. Desired Outcomes and Expectations for Improvement

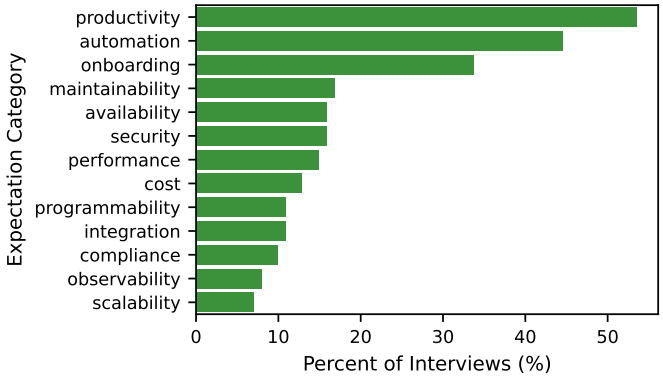

Fig. 3. Frequency of expectations for improvement mentioned across 101 interviews. Productivity and automation outrank raw performance metrics.

Beyond identifying operational constraints, we codified explicit expectations for improvement across thirteen dimensions (Fig. 3): **1) Productivity improvements (53.5%).** Participants sought quantifiable time-to-value gains across the software delivery lifecycle. **2) Automation (44.6%).** They wanted more automation in CI/CD pipelines [14], autoscaling, and rollback processes to remove manual steps and approval bottlenecks. **3) Onboarding improvements (33.7%).** Teams expected intuitive abstractions built into developer workspaces to reduce ramp time from months to weeks. **4) Maintainability (16.8%).** Respondents wanted safer updates and less organizational toil from brittle automation scripts. **5) Availability/Reliability (15.8%) & Security (15.8%).** They emphasized reduced MTTR through innate resiliency and simpler data locality enforcement. **6) Performance (14.9%).** Lower latency and higher throughput, especially tail latency, mattered but ranked below human-centric usability requirements.

Further expectations included cost optimization (12.9%), programmability (10.9%), integration (10.9%), compliance (9.9%), observability (7.9%), and scalability (6.9%).

Participants were notably less interested in raw hardware-level performance optimization than in *developer experience*. This highlights a notable gap in much contemporary edge-cloud systems research, which often optimizes for nanosecond data plane enhancements while underestimating the complexity of the control plane and API abstraction surfaces.

Taken together, these findings point to a common root cause: the fragmentation of abstraction planes forces developers to manually compose independent services for compute, state, and orchestration. Alleviating the dominant pain points—deployment complexity, onboarding difficulty, and cognitive overload—therefore requires higher-level abstractions that consolidate these concerns into declaratively governed deployment units. The following section examines four architectural directions that address this structural deficiency from complementary angles.

IV. Architectural Directions

The empirical evidence presented in Section III identifies a clear demand for abstractions that reduce deployment fragmentation, shorten onboarding cycles, and automate operational toil. No single paradigm is likely to address all practitioner contexts; rather, the findings motivate a family of complementary architectural directions. This section maps the dominant pain points to four such directions: unified object abstractions, platform engineering, AI/ML serving pipelines, and lightweight edge runtimes.

A. Unified Object Abstractions (OaaS)

Object-as-a-Service (OaaS) consolidates compute, state, and orchestration into a unified object abstraction governed by declarative Non-Functional Requirements (NFRs), directly targeting the fragmented abstraction planes identified as the root cause of practitioner pain, as Figure 4 illustrates. OaaS was initially formulated to abstract data-intensive cloud-native workloads [15]. Subsequent work enriched the model with workflow orchestration and execution guarantees for dispersed environments [16], including Oparaca-style encapsulation of orchestrator logic [16], and introduced formal NFR enforcement mechanisms [17]. Most recently, the EdgeWeaver framework extended OaaS to the edge-cloud continuum for IoT scenarios [18].

B. Platform Engineering and Internal Developer Platforms

The rise of platform engineering as a discipline directly mirrors our empirical findings on developer experience. The concept is rooted in the recognition that excessive cognitive load on development teams is a primary bottleneck for software delivery; a recent practitioner survey found that 93% of respondents consider platform engineering beneficial for reducing this burden [19]. Industry surveys report that developers manage an average of 14 distinct vendor tools, contributing to 100-day onboarding cycles [5]—a finding that strongly corroborates our onboarding difficulty statistic. Recent research on developer experience (DevEx) has further established that flow state, feedback loops, and cognitive load are the three key dimensions affecting developer productivity [20]. Platform engineering addresses these dimensions through *Internal Developer Platforms* (IDPs) [21]: curated, self-service layers that abstract infrastructure provisioning behind standardized interfaces, as illustrated by developer portals such as Backstage [22] and the CNCF survey's evidence of widespread Kubernetes production use [23].

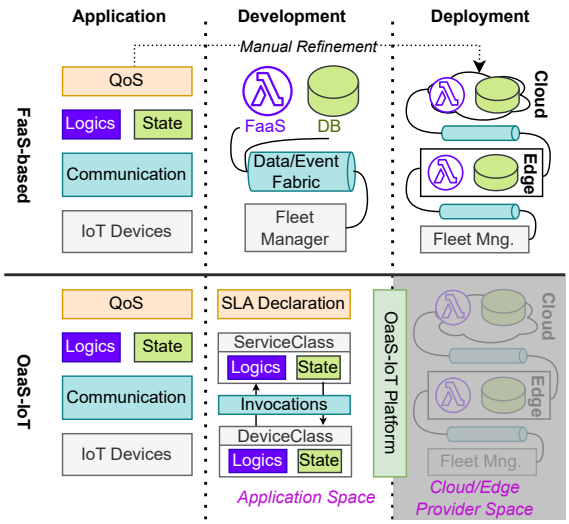

Fig. 4. Comparison of FaaS and OaaS paradigms. In FaaS (top), developers must manually integrate external state stores and coordinate communication via data/event fabrics. In OaaS (bottom), compute, state, and workflow are unified into a single object abstraction managed by the platform.

C. AI/ML Serving Pipelines

The deployment challenges identified in our study manifest acutely in AI/ML infrastructure: MLOps introduces additional orchestration complexity from model versioning, GPU scheduling, and inference pipeline management [6], [7]. Although our interview sample included a modest Data/ML cohort (6 of 101 participants), their pain points—cost unpredictability, scaling difficulties, and integration complexity—echo the general findings, with the productivity gap amplified by the distance between experimental notebooks and production serving. Recent work on LLM serving on preemptible GPUs [24] illustrates how cost-performance trade-offs mirror the broader demand for declarative, outcome-oriented resource governance. The findings motivate AI serving architectures that integrate model artifacts, data pipelines, and inference endpoints into unified deployment units—paralleling OaaS—with declarative interfaces for latency, throughput, and cost constraints, as emerging platforms like BentoML [25] and KServe [26] pursue.

D. Lightweight Edge Runtimes: WebAssembly

The serverless-specific concerns in our study—cold-start latency, execution time limits, and vendor lock-in—intensify at the edge, where resource-constrained devices cannot support full container runtimes. WebAssembly (Wasm) addresses these directly: near-instant cold starts (microsecond-scale versus hundreds of milliseconds for containers), strong sandboxed isolation, and cross-platform portability from a single binary artifact [27] target two validated pain points—performance unpredictability and deployment fragmentation. Comparative studies confirm competitive throughput with reduced resource footprints on constrained edge hardware [28]. Wasm is complementary to OaaS and IDP: replacing the container substrate of OaaS prototypes with Wasm runtimes reduces invocation overhead while preserving unified object abstractions, and IDP golden paths can provision Wasm-based edge functions alongside containerized cloud services for a consistent developer experience across the continuum.

Across these directions, the original design space also exposes three cross-cutting research gaps. First, higher-level abstractions must avoid hiding performance-critical controls when expert operators need them, suggesting adaptive interfaces that default to declarative policies but expose lower-level tuning selectively. Second, cross-tier workflow guarantees remain difficult when edge nodes are intermittently connected and workloads span stateful objects, ML pipelines, and Wasm functions. Third, ecosystem maturity depends on standardized NFR governance interfaces, idiomatic SDKs, integrated debugging, and observability instrumentation; without these operational surfaces, promising runtimes risk remaining research prototypes rather than deployable platform components.

V. Discussion

The architectural directions presented in Section IV address practitioner pain points at the technical level. However, real-world adoption requires simultaneously satisfying three internal personas: *Financial Decision Makers* evaluating ROI, *Technical Decision Makers* assessing architectural safety, and *End Users* evaluating ergonomic benefits. Misalignment among these stakeholders was a recurring theme, with developers favoring cognitive-load reduction while technical leadership demanded production-grade maturity.

Our respondents identified three non-negotiable production prerequisites for any of the four directions: (i) *security*, with multi-tenant isolation and audited IAM controls required in regulated industries (financial services, healthcare) before any consolidation architecture is considered; (ii) *idiomatic developer ergonomics*, with familiar SDKs for Python, Java, Go, and TypeScript and IDE integration required to address onboarding difficulty; and (iii) *observability integration*, with distributed tracing and metrics compatible with existing stacks (Prometheus, OpenTelemetry) as a prerequisite for operations teams. Research prototypes targeting any of the four directions face limited commercial adoption without these operational interfaces—a gap that platform engineering is uniquely positioned to bridge.

We identified two distinct adopter profiles. *SMEs and startups* face acute "infrastructure tax" without dedicated DevOps teams and are natural early adopters of unified abstractions—OaaS or turnkey ML serving platforms—where one deployment unit replaces manual service composition. *Enterprise adopters* require mature IAM, multi-tenant isolation, and audit logging; for them, IDP-based adoption pathways—delivering novel abstractions through an established internal platform with governance controls—are more viable than direct adoption of research prototypes. Across both profiles, respondents favored open, non-proprietary platforms and cloud market-

place integration models that preserve architectural control while reducing configuration overhead.

Practitioners expect *consumption-based* economic models where cost correlates directly to declared quality of service. When a platform accepts a declarative SLA—whether an OaaS NFR, a KServe autoscaling policy, or an IDP resource quota—resource scaling must behave predictably. Admission control mechanisms that dynamically adjust resources without over-provisioning address the demand for transparent cost-performance mapping across all organizational sizes.

VI. Related Work

Our work intersects several active research threads.

Several surveys have examined the practical difficulties of modern distributed systems. Soldani et al. [29] conducted a systematic grey literature review identifying common pains and best practices in microservice architectures, reporting integration testing and fault diagnosis as dominant concerns—findings our interview data corroborates and extends to deployment-level complexity. Luo et al. [30] characterized serverless application patterns and failures, revealing that misconfiguration and tooling fragmentation are leading causes of production incidents. Eismann et al. [31] surveyed serverless adoption and found that vendor lock-in and debugging difficulty rank among the top barriers—consistent with our finding that serverless-specific concerns affected 14.9% of respondents. Our study distinguishes itself by spanning the full cloud-edge continuum rather than focusing on a single platform paradigm, and by systematically coding both pain points and expectations for improvement across 101 practitioner interviews.

Efforts to simplify distributed application development have produced several abstraction models. Function-as-a-Service platforms such as AWS Lambda [32] offload infrastructure management but impose statelessness, forcing developers to externalize state management. Actor-based frameworks, including Microsoft Orleans [33] and Akka, encapsulate state and behavior into virtual actors with location-transparent messaging. While actors address state co-location, they do not provide declarative quality-of-service enforcement or built-in workflow orchestration. CNCF standards like Dapr [34] offer portable building blocks (state stores, pub/sub, bindings) but still require explicit service composition. OaaS differs from these paradigms by unifying compute, state, and workflow into a single deployable object with declarative NFR governance, directly targeting the fragmented abstraction planes our empirical findings identified as the root cause of deployment complexity.

Orchestrating workloads across heterogeneous edge-cloud tiers has been addressed by several frameworks. KubeEdge [35] extends Kubernetes to edge nodes but retains its imperative, container-centric configuration model. OpenFaaS and OpenWhisk bring serverless semantics to edge environments, yet inherit the statelessness limitations of traditional FaaS. Osmotic computing [36] introduced the concept of automatic workload migration between cloud and edge based on resource availability, but lacks formal QoS enforcement mechanisms. WebAssembly has emerged as a lightweight alternative execution substrate; Ménétrey et al. [27] demonstrated Wasm as a common layer for the cloud-edge continuum, achieving microsecond cold starts with strong isolation guarantees. Our work complements these systems by demonstrating, through empirical evidence, that practitioners demand declarative, outcome-oriented abstractions rather than additional imperative configuration surfaces.

The operational challenges of deploying machine learning models in production have been extensively studied. Kreuzberger et al. [6] provide a comprehensive MLOps taxonomy, while Paleyes et al. [7] systematically catalog deployment lifecycle challenges, reporting that infrastructure concerns often dominate over model accuracy issues. The growing complexity of LLM serving infrastructure [24] has further intensified these challenges. While these works examine ML-specific deployment pain, our study provides cross-domain empirical validation demonstrating that the same root causes—fragmented abstractions and manual orchestration—persist across the full cloud-edge continuum, not only in ML pipelines.

Platform engineering has emerged as a discipline focused on reducing developer cognitive load through Internal Developer Platforms [21]. Industry reports document the scale of this problem: developers managing numerous vendor tools with lengthy onboarding cycles [5], and organizations reporting that dominant challenges have shifted from technical to cultural and cognitive dimensions [23]. While Section IV discusses how IDPs address the specific pain points from our study, the related work here positions our empirical contribution: unlike industry surveys that focus on cloud-native early adopters, our interview-based approach captures practitioner perspectives across the full spectrum of distributed infrastructure, including edge and IoT contexts where platform engineering remains nascent.

VII. Conclusion

This paper presented an empirical investigation rooted in structured interviews across 86 technology organizations, characterizing the operational challenges that define the modern cloud-edge continuum. Our findings indicate that the primary barrier to advanced distributed computing is not execution latency, but rather the infrastructure complexity that significantly impedes the practitioners responsible for deployment.

By quantifying that deployment complexity (38.6%) and onboarding difficulty (35.6%) are the dominant pain points—while productivity (53.5%) and automation (44.6%) are the most desired improvements—we have established an empirical foundation for evaluating architectural responses. We examined four complementary directions: Object-as-a-Service (OaaS), which unifies compute, state, and workflow into declaratively governed objects; platform engineering via Internal Developer Platforms, which provides self-service abstraction layers; declarative AI/ML serving pipelines, which ad-

dress the amplified complexity of GPU-bound workloads; and WebAssembly-based edge runtimes, which eliminate cold-start penalties and deployment fragmentation across heterogeneous tiers.

Our analysis of the multi-stakeholder adoption ecosystem further reveals that security, observability integration, and economic transparency are prerequisites for production deployment—requirements that apply across all four directions. Going forward, systems research should prioritize developer experience alongside raw performance, pursue standardized declarative governance interfaces that span paradigms, and bridge the gap between academic prototypes and production-ready platforms.

ACKNOWLEDGEMENT

This project is supported by NSF through CNS CAREER Award# 2419588 and NSF i-Corps Award# 2524083.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," in *IEEE Internet of Things Journal*, vol. 3, no. 5, 2016, pp. 637–646.

[2] P. Gkonis, A. Giannopoulos, P. Trakadas, X. Masip-Bruin, and F. D'Andria, "A survey on iot-edge-cloud continuum systems: Status, challenges, use cases, and open issues," *Future Internet*, vol. 15, no. 12, p. 383, 2023.

[3] V. Velepucha and P. Flores, "A survey on microservices architecture: Principles, patterns and migration challenges," *IEEE access*, vol. 11, pp. 88 339–88 358, 2023.

[4] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," *arXiv preprint arXiv:1812.03651*, 2018.

[5] Harness, "2024 state of developer experience report," https://www.harness.io/state-of-developer-experience, 2024, online; Accessed on 13 Mar. 2026.

[6] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," *IEEE Access*, vol. 11, pp. 31 866–31 879, 2023.

[7] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–29, 2022.

[8] M. Golec, G. K. Walia, M. Kumar, F. Cuadrado, S. S. Gill, and S. Uhlig, "Cold start latency in serverless computing: A systematic review, taxonomy, and future directions," *ACM Computing Surveys*, vol. 57, no. 3, pp. 1–36, 2024.

[9] C. Denninnart and M. Amini Salehi, "SMSE: A Serverless Platform for Multimedia Cloud Systems," *arXiv preprint:220.0194*, 2022.

[10] S. Blank, *The Four Steps to the Epiphany: Successful Strategies for Products that Win*. John Wiley & Sons, 2020.

[11] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," *NIST special publication*, vol. 800, no. 207, pp. 1–52, 2020.

[12] C. Denninnart, T. Chanikaphon, and M. Amini Salehi, "Efficiency in the serverless cloud paradigm: A survey on the reusing and approximation aspects," *Software: Practice and Experience*, vol. 53, no. 10, pp. 1853–1886.

[13] C. Denninnart and M. A. Salehi, "Harnessing the potential of function-reuse in multimedia cloud systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 617–629, 2021.

[14] M. L. Gupta, R. Puppala, V. V. Vadapalli, H. Gundu, and C. Karthikeyan, "Continuous integration, delivery and deployment: A systematic review of approaches, tools, challenges and practices," in *International Conference on Recent Trends in AI Enabled Technologies*. Springer, 2024, pp. 76–89.

[15] P. Lertpongrujikorn and M. A. Salehi, "Object as a service (OaaS): Enabling object abstraction in serverless clouds," in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE, 2023, pp. 315–324.

[16] ——, "Object as a service: Simplifying cloud-native development through serverless object abstraction," *IEEE Transactions on Computers*, vol. 75, no. 2, pp. 423–434, 2026.

[17] P. Lertpongrujikorn, H. D. Nguyen, and M. A. Salehi, "Streamlining cloud-native application development and deployment with robust encapsulation," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '24, 2024, pp. 535–541.

[18] P. Lertpongrujikorn, J. Kwon, H. D. Nguyen, and M. Amini Salehi, "Edgeweaver: Accelerating iot application development across edge-cloud continuum," in *Proceedings of the 40th IEEE International Parallel and Distributed Processing Symposium*, ser. IPDPS '26, 2026.

[19] Puppet, "2023 state of platform engineering report," https://www.puppet.com/resources/state-of-devops-report, 2023, online; Accessed on 13 Mar. 2026.

[20] M. Greiler, M.-A. Storey, and A. Noda, "DevEx: What actually drives productivity," *ACM Queue*, vol. 21, no. 2, pp. 35–53, 2023.

[21] G. P. Rusum and K. K. Pappula, "Platform engineering: Empowering developers with internal developer platforms (idps)," *International Journal of AI, BigData, Computational and Management Studies*, vol. 5, no. 1, pp. 89–101, 2024.

[22] Spotify, "Backstage: An open platform for building developer portals," https://backstage.io, 2024, online; Accessed on 13 Mar. 2026.

[23] Cloud Native Computing Foundation, "Cloud native 2024: Approaching a decade of code, cloud, and change," https://www.cncf.io/reports/cncf-annual-survey-2024/, 2024, online; Accessed on 13 Mar. 2026.

[24] X. Miao, C. Shi, J. Duan, X. Xi, D. Lin, B. Cui, and Z. Jia, "Spotserve: Serving generative large language models on preemptible instances," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 1112–1127.

[25] BentoML, "BentoML: Build, ship, and scale AI applications," https://www.bentoml.com, online; Accessed on 13 Mar. 2026.

[26] KServe Authors, "KServe: Highly scalable and standards-based model inference platform on Kubernetes," https://kserve.github.io/website/, online; Accessed on 13 Mar. 2026.

[27] J. Ménétrey, M. Pasin, P. Felber, and V. Schiavoni, "Webassembly as a common layer for the cloud-edge continuum," in *Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge*, 2022, pp. 3–8.

[28] A. Hall and U. Ramachandran, "WebAssembly and unikernels: A comparative study for serverless at the edge," in *Proceedings of the 9th ACM/IEEE Symposium on Edge Computing (SEC)*. IEEE, 2024, pp. 1–13.

[29] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, "The pains and gains of microservices: A systematic grey literature review," *Journal of Systems and Software*, vol. 146, pp. 215–232, 2018.

[30] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proceedings of the ACM symposium on cloud computing*, 2021, pp. 412–426.

[31] S. Eismann, J. Scheuner, E. Van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "The state of serverless applications: Collection, characterization, and community consensus," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, pp. 4152–4166, 2021.

[32] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[33] S. Bykov, A. Geller, G. Kliot, J. R. Larus, R. Pandya, and J. Thelin, "Orleans: cloud computing for everyone," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, pp. 1–14.

[34] "Dapr: The distributed application runtime," https://dapr.io, 2025, online; Accessed on 3 Mar. 2026.

[35] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *2018 IEEE/ACM Symposium On Edge Computing (SEC)*. IEEE, 2018, pp. 373–377.

[36] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," in *IEEE Cloud Computing*, vol. 3, no. 6, 2016, pp. 76–83.