ORIGINAL RESEARCH ARTICLE UHD JOURNAL OF SCIENCE AND TECHNOLOGY

# Benchmarking Traditional Web Servers and Container-Based Serverless Platforms for a Django E-Commerce System

**Chapk Rzgar Mohamad[1], Kamaran Hama Ali Faraj[1,2]**

*[1]Department of Computer, College of Science, University of Sulaimani, Sulaymaniyah, Kurdistan Region, Iraq, [2]Department of Computer Engineering, College of Engineering and Computer Science, Lebanese French University, Erbil, Kurdistan Region, Iraq.*

## ABSTRACT

We benchmark a Django 5.2 e-commerce application (Python 3.11, SQLite3 as a controlled database constant) on four hosting configurations – Apache 2.4 on Ubuntu 22.04 via WSL2, Nginx 1.27.2 on Windows 11, Azure Container Apps (ACA), and Koyeb – under five load profiles (C = 10–10,000; $n$ = 500–50,000 requests). Sixteen harmonized performance metrics were evaluated, covering mean and tail latency, throughput, error rate, resource utilization, energy-per-request proxy, security timing, and four cloud elasticity metrics. Under the local loopback topology, Apache achieved lower client-observed latency and higher throughput; Nginx exhibited zero errors and tighter tail spread across all load levels. In the cloud topology, Koyeb outperformed ACA in steady-state performance and scaling responsiveness; ACA provided faster cold-start activation. Non-parametric statistical tests (Mann–Whitney U and Kruskal–Wallis with Bonferroni-corrected *post hoc* comparisons), together with effect-size analysis (Cliff's δ and Cohen's d), support the main conclusions of the study. Statistically significant differences are observed for most load levels, whereas intermediate concurrency levels show non-significant differences between ACA and Koyeb. We also introduce the deployment topology-aware benchmarking model, which decomposes client-observed end-to-end latency into topology-dependent and topology-independent components; the associated topology contribution ratio (TCR) – ranging from TCR ≈ 0 for local loopback to TCR ≈ 0.37 for ACA at C = 10 – enables principled cross-environment comparison without conflating network-path effects with platform processing performance.

**Index Terms:** Django E-Commerce, Web-Server Benchmarking, Container-Based Serverless, Tail Latency, Non-Parametric Benchmarking, Topology-Aware Benchmarking, Elasticity

## 1. INTRODUCTION

E-commerce platforms require hosting capable of sustaining low latency, high throughput, and dependable availability under steady-state and flash-crowd demand [1]. Empirical evidence from 10 billion user visits indicates that a 100-ms increase in latency reduces conversion rates by 2.4–7.1% and that users are 15% more likely to complete purchases on high-performance sites [1]. Beyond throughput, hosting systems must deliver consistent response times, as tail latency variability significantly degrades the perceived quality of interactive web applications [2], [3].

Traditional web servers, Apache HTTP server and Nginx, remain widely deployed for dynamic web applications due to mature ecosystems and demonstrated performance characteristics under controlled load [4]. They rely on

Access this article online

DOI:10.21928/uhdjst.v10n1y2026.pp101-115

E-ISSN: 2521-4217
P-ISSN: 2521-4209



**Corresponding author's e-mail:** Dr. Kamaran Hama Ali Faraj, Department of Computer, College of Science, University of Sulaimani, Sulaymaniyah, Kurdistan Region, Iraq. E-mail: KAMARAN.FARAJ@UNIVSUL.EDU.IQ

Received: 27-12-2025 Accepted: 27-03-2026 Published: 26-04-2026

statically provisioned resources; under heavy concurrency, performance becomes sensitive to capacity-planning decisions [4] where over-provisioning increases costs and under-provisioning leads to request queueing, elevated tail latency, and potential service instability [3], [5].

Container-based serverless platforms such as Azure Container Apps (ACAs), Google Cloud Run, AWS App Runner, and Koyeb offer platform-managed automatic scaling with scale-to-zero support [5]. While vendor documentation is cited only to report the precise configuration parameters used in our deployments, the scientific implications of this model – cold-start latency, scaling delay, and quality of service (QoS) trade-offs under bursty demand – are well established in peer-reviewed serverless literature [6]–[8].

Existing empirical research has largely characterized traditional web servers or serverless platforms in isolation, rarely both under unified experimental conditions. This creates five research gaps: (1) Apache/Nginx studies rarely integrate tail percentiles and security timing under a harmonized protocol [4], [9]; (2) serverless benchmarking is dominated by function-as-a-service (FaaS) evaluations rather than full long-running web applications [10], [11]; (3) tail-latency variability is not consistently integrated into cross-paradigm benchmarking for realistic workloads [3], [7]; (4) managed container platforms – especially Koyeb – are largely absent from peer-reviewed empirical benchmarking [5], [12]; (5) cross-paradigm studies rarely integrate energy-aware indicators and security timing in the same unified protocol [13], [14].

This study addresses these gaps through a controlled experimental comparison of traditional and container-based serverless hosting for an identical Django e-commerce workload. The core research question is: How do traditional web servers and container-based serverless platforms differ in latency, throughput, robustness, resource behavior, security responsiveness, and elasticity when hosting the same e-commerce workload under increasing concurrency?

The contributions of this work are fourfold. First, a unified, reproducible cross-paradigm benchmarking framework applying a harmonized 16-metric evaluation model across both hosting paradigms (addresses Gaps 1–5). Second, the first peer-reviewed empirical evaluation of Koyeb as a managed container-based serverless platform under sustained high-concurrency e-commerce workloads (Gaps 2, 4). Third, integrated performance–elasticity–robustness–security characterization, including cumulative distribution functions (CDF)-based tail analysis and formal non-parametric statistical validation (Mann–Whitney U, Kruskal–Wallis, Bonferroni correction, Cohen's d, Cliff's delta) (Gaps 1, 3, 5). Fourth, the deployment topology-aware benchmarking (DTAB) model – a formal latency decomposition with topology contribution ratio (TCR) providing a reusable analytical framework for any cross-environment benchmarking study (Gaps 1, 3). The remainder of this paper is organized as follows: Section 2 reviews related work; Section 3 describes the methodology; Section 4 presents and analyzes the empirical results; Section 5 concludes.

## 2. RELATED WORK

Prior studies have examined web-server and serverless performance across several research threads [4], [6], [11], [15], but the evidence base remains fragmented because traditional evaluations and serverless characterizations rely on different applications, workload generators, and metric sets, limiting direct cross-paradigm comparison [10], [16], [17]. Moreover, serverless research is dominated by FaaS benchmarking [8], [10], [11]. At the same time, managed container platforms for long-running web applications appear less frequently and are typically studied from usability rather than sustained high-concurrency perspectives [5].

### 2.1. Traditional Web-Server Benchmarking

Traditional web-server studies commonly compare Apache and Nginx, emphasizing throughput and average latency under concurrent loads [4], [16]. Wicoksono *et al.* benchmarked Apache, Nginx, and Lighttpd on Debian, reporting request-rate and transfer-rate results [4]. Farhadian *et al.* confirmed that web-server performance depends significantly on the hosting environment across Docker, Podman, and LXC containers [9]. Recent 2025 evaluations confirm persistent architectural trade-offs between event-driven and thread-based models under high concurrency [4], [16].

### 2.2. Serverless Platform Characterization (FaaS-Focused)

Yu *et al.* proposed ServerlessBench for AWS Lambda, Apache OpenWhisk, and Fn [11]. Copik *et al.* introduced SeBS for reproducibility [10]. Scheuner *et al.* proposed CrossFit for fair cross-provider benchmarking [17]. Golec *et al.* reviewed cold-start latency mitigation techniques [8]. A 2025 systematic review further highlights the transition of serverless research toward high-performance computing and artificial intelligence (AI) workloads, while noting limited evaluation of long-running web applications [6]. Schirmer

**TABLE 1: Comparison of prior benchmarking studies with the present work**

| Study | Year | Platforms | Type | Metrics/focus | Key limitation | Gap |
|---|---|---|---|---|---|---|
| Wicoksono *et al*. [4] | 2025 | Apache, Nginx, Lighttpd | Traditional | Throughput, transfer | No tail or elasticity metrics | Gap 1 |
| Farhadian *et al*. [9] | 2025 | Apache, Nginx on Docker/Podman/ LXC | Traditional | Latency, throughput, and resources under containers | Heterogeneous methods | Gap 1 |
| Li *et al*. (TailBench++) [2] | 2025 | Multiclient/ multiserver | Cross-platform | Tail-latency benchmarking methodology | No elasticity, security, or container comparison; arXiv preprint | Gap 3 |
| Kasture and Sanchez (TailBench) [21] | 2016 | 8 latency-critical apps | Cross-platform | Tail-latency benchmark suite, open-loop methodology | No web-server or serverless comparison | Gap 3 |
| Henning *et al*. [20] | 2025 | AWS Kubernetes | Cross-platform | Cloud performance variability (longitudinal) | Stream processing only; no web-server or serverless container comparison | Gap 3 |
| Yu *et al*. ServerlessBench) [11] | 2020 | Lambda, OpenWhisk, Fn | FaaS | Platform characterization | No web-server comparison | Gap 2 |
| Copik *et al*. (SeBS) [10] | 2021 | Multiple FaaS | FaaS | Benchmark suite | FaaS-only | Gap 2 |
| Scheuner *et al*. (CrossFit) [17] | 2022 | AWS, Azure | FaaS | Fair benchmarking | No containers | Gap 2 |
| Schirmer *et al*. [7] | 2023 | Cloud serverless | FaaS | Time-dependent performance variability | Not long-running web apps | Gap 3 |
| Golec *et al*. [8] | 2024 | Serverless (survey) | FaaS | Cold-start taxonomy | Survey only; no managed containers | Gap 2 |
| Abraham and Yang [5] | 2024 | Cloud Run, app runner, ACA | Serverless containers | Usability, features | No Koyeb; no sustained load | Gap 4 |
| Pratama [18] | 2026 | Cloud Run, app runner | Serverless containers | p95 latency, throughput, error rate | No traditional stacks, no Koyeb, max C=1000 | Gap 4 |
| Besozzi *et al*. [6] | 2025 | Serverless (broad) | FaaS | Systematic review (HPC/AI) | Non-web workloads dominate | Gap 2/ Motivation |
| Ghattas *et al*. [15] | 2025 | Website metrics | Cross-domain | Metrics+research gaps | Fragmented benchmarking landscape | Motivation |
| Werner *et al*. [13] | 2025 | Cloud-native | Cross-platform | Energy-aware experimentation | Framework only | Gap 5 |
| Legler *et al*. [14] | 2025 | Cloud-native microservices | Cross-platform | Energy modeling | Not unified benchmarking | Gap 5 |
| GreenCloud [22] | 2012 | Data centers | Cross-platform | Energy modeling | Simulation-level | Gap 5 |
| Present study | 2026 | Apache, Nginx, ACA, Koyeb | Unified | 16 harmonized metrics+CDF tail analysis+non-parametric statistical validation | — | Gaps 1–5 |

HPC/AI: High performance computing/artificial intelligence, CDF: Cumulative distribution functions, ACA: Azure Container Apps

**TABLE 2: Baseline network latency measured from benchmarking client to cloud endpoints (interpretation only; not subtracted from metrics; RTT)**

| Target endpoint | Baseline type | Summary (ms) |
|---|---|---|
| Local Apache (loopback) | RTT baseline | ≈ 0 (negligible) |
| ACA | ICMP RTT (ping -n 20) | Min=175, Avg=181, Max=191 |
| ACA | HTTPS setup+first response (Invoke-WebRequest×20) | Median=464.9, IQR=23.8 (includes first-hit=33,285.7 ms) |
| ACA | HTTPS steady-state (exclude first hit; n=19) | Median=461.7, IQR=24.2 |
| Koyeb | ICMP RTT (ping -n 20) | Min=20, Avg=25, Max=32 |
| Koyeb | HTTPS setup+first response (Invoke-WebRequest×20) | Median=391.4, IQR=137.2 |

ACA: Azure Container Apps, RTT: Round-trip time, IQR: Interquartile range

*et al*. demonstrated time-dependent performance variability in serverless platforms [7]. All of these studies focus on FaaS models and do not compare against traditional web-server stacks under unified workloads (Gap 2).

### 2.3. Container-Based Serverless Platforms

Yang and Abraham analyzed Cloud Run, App Runner, and ACA from usability and feature perspectives, but they provide limited sustained high-concurrency benchmarking data [5]. Pratama (2026) benchmarked Cloud Run and App Runner under up to 1000 virtual users, reporting p95 latency, throughput, and error rate, but excluded traditional web-server stacks and Koyeb [18]. Henning and Hasselbring benchmarked containerized stream processing on Kubernetes and confirmed that cloud performance is time-dependent and affected by multitenant variability [19], [20]. Dean and Barroso established the fundamental importance of tail latency at scale [3]. Kasture and Sanchez introduced TailBench for latency-critical applications [21]. Li *et al.* (2025; arXiv preprint) extend this framework to multiclient cloud environments [2]. Emerging research on energy-aware benchmarking proposes central processing unit (CPU)-proxy energy models for cloud sustainability [13], [14], with GreenCloud's linear power model remaining widely adopted when direct instrumentation is unavailable [22]. Koyeb does not appear as an evaluated platform in any peer-reviewed benchmarking study, representing a specific gap in managed serverless container evaluation (Gap 4).

### 2.4. Positioning of the Present Study

This study advances existing benchmarking literature through a unified cross-paradigm comparison that eliminates application-level and workload heterogeneity, enabling results that reflect architectural and platform-level behavior rather than implementation differences as summarized in Table 1. Whereas most prior studies evaluate traditional web servers or serverless platforms in isolation, this work benchmarks both paradigms using the same Django e-commerce application, identical (N, C) load profiles, and a harmonized 16-metric framework. It extends serverless benchmarking beyond FaaS by incorporating container-based platforms hosting long-running web applications, and provides the first peer-reviewed empirical evaluation of Koyeb under sustained high-concurrency conditions. This aligns with established experimental systems research, where novelty derives from controlled comparative evaluation rather than algorithmic invention.

## 3. MATERIALS AND METHODS

### 3.1. Application under Test and Hosting Platforms

All experiments used the same Django 5.2 application (Python 3.11, SQLite3), providing product catalog, cart, checkout, and authentication. Traditional stacks: (1) Apache 2.4 with mod_wsgi (daemon mode, worker MPM) on Ubuntu 22.04; (2) Nginx 1.27.2 as a reverse proxy with Waitress 3.0.1 on Windows 11, least-connection upstream. Cloud stacks: a Python 3.11-slim container with Gunicorn

**TABLE 3: Load profiles**

| Category | Concurrency | Total requests | Interpretation |
|---|---|---|---|
| Very low (VL) | 10 | 500 | Development/ staging load |
| Low (L) | 100 | 2,000 | Small production site |
| Medium (M) | 500 | 10,000 | Busy period |
| High (H) | 2,000 | 25,000 | Daily peak traffic |
| Very high (VH) | 10,000 | 50,000 | "Black Friday" surge |

**TABLE 4: Evaluated metrics (12 core+4 serverless-only) *Cloud elasticity metrics for ACA and Koyeb**

| Metric | Definition | Source |
|---|---|---|
| Mean latency (ms) | Average end-to-end response time | AB/aiohttp client |
| P99 latency (ms) | 99$^{th}$-percentile response time | AB/client |
| Tail latency stability (ms) | P99−P95 | Derived |
| Throughput (req/s) | Completed requests per second | AB/client |
| Error rate (%) | (N_failed/N_total)×100 | AB/client |
| CPU utilization (%) | Peak CPU during run | psutil/provider telemetry |
| Memory usage | Peak RSS MB (local)/quota % (cloud) | psutil/provider telemetry |
| Scalability efficiency (%) | RPS/(RPS_ref×C)×100 | Derived |
| Energy per request | Utilization-based proxy (J/req); see §3.6 | Derived |
| Security detection time (ms) | Time from first malicious request to first detection indicator | Custom scripts/logs |
| Security response time (ms) | Time from first malicious request to protective outcome | Custom scripts/logs |
| Security success rate (%) | (N_blocked/N_malicious)×100 | Custom scripts |
| Cold-start latency (ms)* | First response after scale-to-zero (≥10 min idle) | Provider logs/timing |
| Warm-start latency (ms)* | Mean latency with already-active instance | Timing script |
| Auto-scaling latency (ms)* | Time from demand spike to effective new capacity | Provider telemetry |
| Burst-recovery time (ms)* | Time to restore normal performance after a burst | Timing script |

CPU: Central processing unit, ACA: Azure Container Apps

on ACA (0–10 replicas, East US) and Koyeb (≤20 instances, Frankfurt). Hardware: Dell laptop, Intel Core i7-8650U (4 cores/8 threads), 16 GB RAM, and SSD storage. The workload covers navigation-style endpoints that are representative of standard e-commerce benchmarking scenarios. Local experiments used loopback; cloud tests over the public Internet.

Regional deployment justification. ACA was deployed in the East US, the provider's primary North American region, with the largest service capacity and lowest baseline latency for managed container workloads. Koyeb was deployed in Frankfurt, which is the provider's primary European region and the geographically closest available location to the Middle East-based client environment used in this study. These selections aim to evaluate each platform under best-practice deployment conditions, as recommended by the providers; consequently, cross-region comparisons reflect realistic end-to-end, user-observed latency, including wide-area network effects and provider routing policies, rather than isolated server-side processing time alone.

Round-trip time (RTT) baseline and interpretation as reported in Table 2. To quantify the transport and connection overhead absent from loopback tests, baseline latency was measured from the benchmarking client to each cloud endpoint before load testing. ICMP RTT captures the underlying network-path delay, while HTTPS setup timing provides a practical client-observed baseline. ACA exhibited a large first-hit timing (33.3 s), consistent with a scale-from-idle cold initialization; therefore, ACA steady-state HTTPS timing is reported separately after excluding the first request. These baselines contextualize cross-topology results as deployment-level outcomes; RTT is not subtracted from end-to-end latency to claim intrinsic server-processing time. Such baseline measurements are consistent with prior cloud-to-user latency characterizations for major providers [23].

### 3.2. DTAB Model

To enable principled comparison across heterogeneous deployment environments, we propose the DTAB interpretation model, which formalizes the decomposition of client-observed latency into topology-dependent and topology-independent components:

$$T\_E2E = T\_topo + T\_platform + \varepsilon \quad (1)$$

where T_topo = T_RTT + T_TLS + T_routing (network round-trip, TLS handshake, provider routing) and T_platform = T_queue + T_app (server queuing and application processing). The TCR quantifies topology influence on observed latency:

$$TCR = T\_topo/T\_E2E \quad (2)$$

TCR ≈ 0 indicates platform-dominated results; TCR ≈ 1 indicates network-dominated results. For this study: TCR_Apache ≈ 0 (loopback), TCR_Koyeb ≈ 0.09, TCR_ACA ≈ 0.37 at C = 10. Three rules govern interpretation: Rule 1 (Within-topology) – platforms sharing the same topology may be compared directly; Rule 2 (Cross-topology) – observed differences reflect deployment outcomes, including network-path effects; Rule 3 (Boundary condition) – no claim is made that local stacks are intrinsically faster in server processing (T_platform). The DTAB model is not specific to the platforms evaluated and provides a reusable analytical framework for any cross-environment study.

### 3.3. Database Selection and Endpoint Scope

SQLite was selected to preserve identical application behavior across all platforms, treating the persistence layer as a controlled constant [24]. SQLite was used solely to maintain application portability across platforms and was not the focus of performance evaluation. The primary benchmark target was HTTP GET/login/; authenticated sessions return HTTP 302 redirects, exercising session-check logic rather than write-intensive transactions. Write activity was negligible. WSL2 has been reported to introduce CPU overhead of approximately 5–15% [25], [26]; since both traditional stacks ran on the same hardware under identical load generation, comparative results remain valid for stack-level evaluation.

### 3.4. Benchmark Load Profiles

ApacheBench (ab -k -n N -c C) generated load for traditional stacks; a custom Python asyncio/aiohttp script with equivalent (N, C) semantics was used for cloud platforms to accommodate HTTPS requirements. Cold-start and security-timing measurements used separate dedicated runs as defined in Table 3.

ApacheBench implements a closed-model workload generator, where new requests are issued only after prior requests complete, matching the controlled concurrency semantics (fixed N total requests, C maximum in-flight) required by this study's load profiles. Modern tools such as k6 and Locust support open-model (constant arrival rate) and scenario-based workload patterns that better represent realistic user behavior with think times and mixed endpoints; however, the open-model approach introduces additional variability that complicates controlled cross-platform

comparison under fixed concurrency targets. Validation using open-model load generators with multi-endpoint user journeys is planned as future work.

### 3.5. Performance and Elasticity Metrics

Metric selection was guided by systematic reviews of web and cloud performance evaluation, which identify tail percentiles, throughput, and error behavior as core QoS indicators and emphasize that mean-only reporting is insufficient for interactive systems [15]. Serverless literature further highlights cold-start latency, scaling delay, and burst-recovery dynamics as critical user-visible behaviors that must be measured explicitly for elastic platforms as listed in Table 4 [6]–[8].

Accordingly, 12 core metrics are reported across all platforms, complemented by four elasticity-specific metrics for managed container platforms. Metrics were collected from three complementary sources: client-side measurements using ApacheBench and custom Python asyncio/aiohttp scripts; host-level monitoring through psutil for traditional stacks, where CPU and memory values represent peak observed usage over each run; and provider telemetry and logs for cloud platforms, where CPU and memory are expressed as percentages of the allocated quota and interpreted as platform-reported resource-pressure indicators rather than directly comparable absolute units.

Security timing metrics are derived from scripted malicious-request injections: detection time is the elapsed time from the first injected request to the first server-side detection indicator, and response time is the elapsed time to the first protective outcome observed by the client and confirmed in logs [27]. For managed platforms, percentile values (P95, P99) were computed empirically from per-request latency samples collected by the aiohttp client rather than from ApacheBench output.

### 3.6. Energy-Per-Request Proxy (EPR) Model and Data Sources

Direct power instrumentation is unavailable for managed container platforms as tenants do not have access to the underlying physical hosts [13]. Following GreenCloud [22] and Werner *et al.* [13], we adopt a linear server power model:

$$P(t) = P\_max \times (k + (1 - k) \times u(t)) \quad (3)$$

where $u(t) \in [0,1]$ is instantaneous CPU utilization, P_max is maximum server power, and $k = P\_idle/P\_max \approx 0.66$ (GreenCloud values: Idle ≈ 198 W, peak ≈ 301 W). For traditional stacks, u(t) is sampled via psutil at $\Delta t = 1$ s; for managed platforms, $\Delta t$ corresponds to provider telemetry granularity (typically 30–60 s). Total estimated energy and EPR are then:

$$E\_total = \Sigma\, P(t\_i)\, \Delta t \quad EPR = E\_total/N\_completed \quad (4)$$

EPR is a utilization-based proxy used strictly for relative workload comparison under identical sampling procedures; it does not represent measured physical energy consumption of the underlying cloud hosts. Sensitivity analysis confirms that varying k between 0.50 and 0.80 preserves the relative EPR ranking across all platform comparisons, demonstrating that the proxy is robust to reasonable uncertainty in the idle-power fraction.

### 3.7. Statistical Analysis Framework

Shapiro–Wilk tests confirmed significant non-normality ($P < 0.001$) for all latency distributions, consistent with the right-skewed, heavy-tailed behavior widely reported in web-system latency research, and justifying the use of non-parametric methods throughout: (i) Mann–Whitney U for pairwise within-paradigm comparisons; (ii) Kruskal–Wallis H-test for omnibus cross-platform comparison; (iii) Bonferroni correction for multiple *post hoc* comparisons; (iv) 95% confidence intervals for mean latency estimates; and (v) Cliff's delta ($\delta$) and Cohen's d as effect size measures; Cliff's delta quantifies stochastic dominance without assuming normality and is more appropriate than standardized mean differences for skewed distributions, whereas Cohen's d is included for comparability with prior literature. CDF analysis complements percentile summaries with full distributional characterization of tail behavior.

### 3.8. Normalized Score Calculation

Normalized scoring is a common approach in multicriteria benchmarking comparison [15]. For compact cross-metric comparison, each metric is converted into a 0–100 normalized score per platform, where 100 indicates the better value for that metric, and the other opposing platform receives a proportional score.

For lower-is-better metrics (latency, error rate, etc.):

$$Score\_i = 100 \times [\min(x\_A, x\_B)/x\_i] \text{ (lower-is-better)} \quad (5)$$

$$Score\_i = 100 \times [x\_i/\max(x\_A, x\_B)] \text{ (higher-is-better)} \quad (6)$$

where x_i is the measured value for platform i and x_A, x_B are the values for platforms A and B, respectively. Where

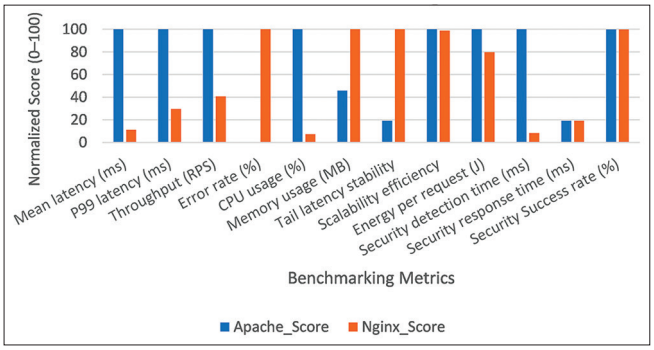


**Fig. 1.** Average normalised scores (0–100) of Apache and Nginx across twelve benchmarking metrics.

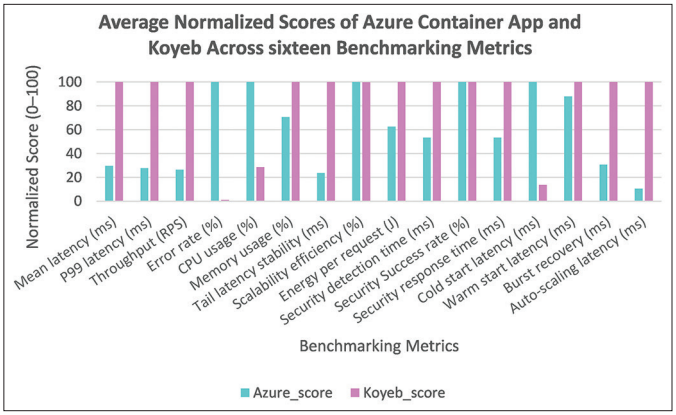


**Fig. 2.** Average normalized scores (0–100) of Azure Container Apps and Koyeb across 16 metrics.

min(x_A, x_B) = 0 for a lower-is-better metric, the platform with zero receives 100, and the other receives 0. Average normalized scores are the arithmetic mean across all metrics with equal weight, visualized in Figs. 1 and 2.

### 3.9. Experimental Procedure

Each benchmark followed three sequential phases within a controlled experimental session, not calendar days. Phase 1 covered application startup and warm-up to stabilize runtime behavior before measurement collection; Phase 2 comprised systematic load execution across all five profiles with metric collection, with the warm-up phase omitted for cold-start experiments; Phase 3 included cool-down intervals, scale-to-zero observation, and cold-start measurements after ≥10 min idle, with cold-start runs initiated only after scale-to-zero behavior was confirmed for ACA and Koyeb [28], [29]. Each test case was executed in repeated runs, and the results are reported as arithmetic means. Software versions, scripts, and raw data are maintained at https://github.com/Ch4pkRzg4r/django-ecommerce-benchmarking; the repository will be made publicly accessible upon paper acceptance.

## 4. RESULTS AND DISCUSSION

Performance metrics were collected from ApacheBench outputs and custom HTTP scripts; resource- and elasticity-related metrics were derived from psutil sampling and cloud-provider telemetry. Each test case was executed in repeated runs; reported values are arithmetic means. Metrics were first aggregated per test case and then summarized by load level (very low to very high) to support consistent comparison across platforms. Results for traditional stacks are presented in Figs. 1 and 3, and Table 5, managed platforms in Figs. 2 and 4 and Table 6, statistical validation in Tables 7-9, and cross-architecture outcomes in Table 10.

### 4.1. Traditional Hosting Stacks: Apache Versus Nginx

Fig. 3 compares Apache + mod_wsgi on Ubuntu WSL2 with Nginx + Waitress on Windows 11 across five load categories; Table 5 reports aggregated averages. Both platforms were evaluated under identical local loopback conditions (DTAB Rule 1: Within-topology); the following results represent stack-level behavior. Latency and throughput: Apache achieves approximately ×9 lower mean latency (658 ms vs. 5,911 ms), ×3.4 lower P99 latency, and ×2.5 higher throughput under the tested loopback workload profiles. All differences are statistically significant ($P < 0.001$, Cohen's d ≥1.23 – large effect).

Reliability: Nginx records 0% error rate across all load profiles; Apache peaks at 8.79% under very high concurrency. Nginx shows ×5.2 more stable tail latency (P99-P95: 117 ms vs. 610 ms) and ×2.2 lower memory footprint. These trade-offs are summarized by the normalized comparison (Fig. 1): Apache dominates latency, throughput, CPU efficiency, energy proxy, and security timings, whereas Nginx leads in error rate, memory footprint, and tail latency stability.

Security timing: Apache achieves ×12 faster detection (610 ms vs. 7,370 ms) and ×5.2 faster response (1,610 ms vs. 8,370 ms), though both platforms achieve ~99.8% security success rate. Both stacks were evaluated under identical loopback conditions (DTAB Rule 1), making direct comparison valid.

### 4.2. Container-Based Serverless Platforms: ACA Versus Koyeb

Fig. 4 and Table 6 compare ACA and Koyeb under identical load profiles. Both platforms operate over the public Internet (WAN topology), so results reflect within-cloud comparison (DTAB Rule 1), with TCR_Koyeb ≈ 0.09 and TCR_ACA ≈ 0.37 at C = 10 providing upper bounds on topology contribution. Because ACA (East US) and Koyeb (Frankfurt)

**TABLE 5: Metric comparison of traditional hosting stacks (Apache vs. Nginx)**

| Metric | Apache_Avg | Nginx_Avg | Winner | Advantage |
|---|---|---|---|---|
| Mean latency (ms) | 658.11 | 5910.76 | Apache | ≈×9 lower latency |
| P99 latency (ms) | 1789.41 | 6046.15 | Apache | ≈×3.4 lower tail latency |
| Throughput (RPS) | 3057.60 | 1245.16 | Apache | ≈×2.5 higher throughput |
| Error rate (%) | 2.59 | 0.00 | Nginx | No errors versus 2.6% failures |
| CPU usage (%) | 3.67 | 50.11 | Apache | ≈×13.7 lower CPU utilization |
| Memory usage (MB) | 5680.30 | 2602.56 | Nginx | ≈×2.2 lower memory footprint |
| Tail latency stability | 610.31 | 117.25 | Nginx | ≈×5.2 more stable tail latency |
| Scalability efficiency | 10.02 | 9.91 | Apache | Slightly better scaling (~1%) |
| Energy per request (J) | 2.85 | 3.57 | Apache | ≈×1.25 more energy-efficient |
| Security detection time (ms) | 610 | 7370 | Apache | ≈×12 faster attack detection |
| Security response time (ms) | 1610 | 8370 | Apache | ≈×5.2 faster security response |
| Security success rate (%) | 99.93 | 99.76 | Apache | Marginally higher |

**TABLE 6: Metric comparison of container-based serverless platforms (ACA vs. Koyeb)**

| Metric | Azure average | Koyeb average | Better (Internet-hosted comparison) | Relative difference |
|---|---|---|---|---|
| Mean latency (ms) | 6344.49 | 1886.80 | Koyeb | ≈×3.3 faster |
| P99 latency (ms) | 12,541.26 | 3,487.03 | Koyeb | ≈×3.6 lower tail latency |
| Throughput (RPS) | 53.29 | 200.67 | Koyeb | ≈×3.7 higher |
| Error rate (%) | 8.12 | 0.09 | Koyeb | Much lower failures |
| CPU usage (%) | 3.16 | 11.01 | Azure | Lower reported CPU utilization |
| Memory usage (%) | 12.91 | 9.11 | Koyeb | ≈×1.4 lower footprint |
| Tail latency stability (ms) | 2093.17 | 495.98 | Koyeb | ≈×4.2 more stable tail |
| Scalability efficiency (%) | 9.91 | 9.93 | Tie | Similar efficiency |
| Energy per request (J) | 2.89 | 1.81 | Koyeb | ≈×1.6 more efficient |
| Security detection time (ms) | 764.55† | 407.70 | Koyeb | ≈×1.9 faster |
| Security success rate (%) | 100 | 100 | Tie | Full success |
| Security response time (ms) | 764.55† | 407.71 | Koyeb | ≈×1.9 faster |
| Cold start latency (ms) | 855.51 | 6,243.01 | Azure | ≈×7 faster |
| Warm start latency (ms) | 838.88 | 737.46 | Koyeb | ≈13% faster |
| Burst recovery (ms) | 9,198.47† | 2,821.14 | Koyeb | ≈×3.2 faster |
| Auto-scaling latency (ms) | 9,198.47† | 978.49 | Koyeb | ≈×9.4 faster |

CPU: Central processing unit, ACA: Azure Container Apps. †For ACA, Security detection time and security response time are identical (764.55 ms) because the platform's protective response was triggered atomically within the same request-handling cycle – detection and blocking occurred as a single observable event. Auto-scaling latency and Burst-recovery time are likewise identical (9,198.47 ms) because ACA's scaling mechanism did not produce a distinct post-burst stabilization phase separable from the scaling event itself; both metrics were derived from the same measurement window

**TABLE 7: Mann–Whitney U test results for mean latency: Apache versus Nginx (selected concurrency levels)**

| Concurrency | Apache mean (ms) | 95% CI apache | Nginx mean (ms) | 95% CI nginx | n_A | n_B | U | P-value | Cohen's d |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 4.6 | (4.0, 5.2) | 81.2 | (59.1, 103.3) | 36 | 36 | 0 | <0.000001 | 1.60 |
| 50 | 23.8 | (8.9, 38.6) | 395.9 | (289.2, 502.6) | 36 | 36 | 21 | <0.000001 | 1.60 |
| 100 | 86.8 | (29.7, 143.8) | 624.0 | (470.0, 778.1) | 54 | 54 | 169 | <0.000001 | 1.23 |
| 500 | 106.7 | (82.8, 130.6) | 2009.0 | (1841.4, 2176.6) | 31 | 42 | 0 | <0.000001 | 4.82 |
| 1000 | 237.4 | (154.2, 320.7) | 3785.6 | (3540.1, 4031.2) | 27 | 36 | 0 | <0.000001 | 6.40 |
| 5000 | 2998.9 | (2233.8, 3764.1) | 21924.6 | (20358.0, 23491.3) | 22 | 24 | 0 | <0.000001 | 6.19 |

CI: Confidence interval

have different RTT baselines (181 ms vs. 25 ms, respectively), these results reflect end-to-end deployment outcomes within the WAN class rather than strict within-topology equivalence, though both TCR values remain below 0.40.

Steady-state performance. Koyeb achieves ×3.3 lower mean latency (1887 ms vs. 6344 ms), ×3.6 lower P99, and ×3.7 higher throughput. Koyeb also shows a much lower error rate (0.09% vs. 8.12%) and a tighter tail spread (P99–P95: 496 ms vs. 2,093 ms). Scalability efficiency is similar for both platforms (≈10%),

**TABLE 8: Mann–Whitney U test results for mean latency: Koyeb versus ACA (selected concurrency levels)**

| Concurrency | Koyeb mean (ms) | 95% CI koyeb | ACA mean (ms) | 95% CI ACA | n_K | n_A | U | *P*-value | Cohen's d |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 269.8 | (262.1, 277.6) | 492.6 | (461.1, 524.2) | 18 | 18 | 0 | <0.000001 | 4.48 |
| 50 | 378.9 | (325.1, 432.8) | 662.1 | (550.8, 773.5) | 18 | 18 | 30 | 0.000032 | 1.50 |
| 100 | 529.5 | (449.2, 609.8) | 759.1 | (641.4, 876.9) | 27 | 27 | 164 | 0.000540 | 0.86 |
| 500 | 1953.1 | (1598.4, 2307.9) | 3904.6 | (1199.6, 6609.6) | 21 | 21 | 254 | 0.406 | 0.43 |
| 1000 | 3918.6 | (3180.5, 4656.7) | 6720.5 | (2827.3, 10613.7) | 18 | 18 | 167 | 0.887 | 0.46 |
| 5000 | 7222.3 | (6458.0, 7986.7) | 24154.2 | (15022.8, 33285.7) | 12 | 12 | 14 | 0.000901 | 1.48 |

ACA: Azure Container Apps, CI: Confidence interval

**TABLE 9: Kruskal–Wallis omnibus test *post hoc* comparisons (Bonferroni corrected)**

| Pair | $Mean_1$ (ms) | $Mean_2$ (ms) | Raw *P*-value | Bonferroni-adjusted *P* | Cohen's d |
|---|---|---|---|---|---|
| Apache versus Nginx | 744.5 | 7309.7 | $4.76\times10^{-51}$ | $2.85\times10^{-50}$ | 0.64 |
| Apache versus Koyeb | 744.5 | 3075.9 | $1.45\times10^{-45}$ | $8.73\times10^{-45}$ | 0.74 |
| Apache versus ACA | 744.5 | 7161.1 | $5.94\times10^{-50}$ | $3.56\times10^{-49}$ | 0.73 |
| Nginx versus Koyeb | 7309.7 | 3075.9 | 0.551 | 1.000 | 0.40 |
| Nginx versus ACA | 7309.7 | 7161.1 | 0.006 | 0.037 | 0.01 |
| Koyeb versus ACA | 3075.9 | 7161.1 | $2.59\times10^{-4}$ | $1.55\times10^{-3}$ | 0.45 |

ACA: Azure Container Apps

**TABLE 10: Cross-architecture comparison of the best traditional stack (Apache) and the best container-based serverless platform (Koyeb)**

| Metric | Apache (local loopback) average | Koyeb (Internet-hosted) average | Observed end-to-end outcome (topology-dependent) |
|---|---|---|---|
| Mean latency (ms) | 658.11 | 1886.80 | Lower end-to-end latency observed under loopback topology |
| P99 latency (ms) | 1789.41 | 3487.03 | Lower tail latency under loopback conditions |
| Throughput (RPS) | 3057.60 | 200.67 | Higher achieved throughput under local topology |
| Error rate (%) | 2.59 | 0.09 | Lower observed failure rate under the managed platform |
| CPU usage (% of allocated) | 3.67 | 11.01 | Lower measured CPU utilization under the local sampling method |
| Memory usage (MB for Apache/% quota for Koyeb)* | 5680.30 | 9.11% quota | Different units; footprint comparison is qualitative only |
| Tail latency stability (ms) | 610.31 | 495.98 | Slightly tighter tail distribution under the managed platform |
| Scalability efficiency (%) | 10.02 | 9.93 | Similar scaling efficiency under respective topologies |
| Energy per request (J) | 2.85 | 1.81 | Lower proxy energy per request under the managed platform |
| Security detection time (ms) | 610.00 | 407.70 | Faster client-observed detection under managed deployment |
| Security success rate (%) | 99.93 | 100.00 | Near-equivalent success rates |
| Security response time (ms) | 1610.00 | 407.71 | Faster client-observed response under managed deployment |

CPU: Central processing unit,* Memory usage values are reported in different units: local RSS (MB) for Apache and quota percentage (%) for Koyeb; cross-paradigm comparison is qualitative only.

but at very different absolute throughput and latency levels. Most differences are statistically significant (Table 8), except at intermediate concurrency (C = 500–1,000), where *P*-values do not reach significance and effect sizes are small-to-moderate.

Elasticity: Koyeb shows ×9.4 lower auto-scaling latency (978 ms vs. 9,198 ms), ×3.2 faster burst recovery, and 13% faster warm-start. ACA provides ×7.3 faster cold-start activation (856 ms vs. 6,243 ms), which is advantageous for infrequently accessed deployments where cold-start is the dominant latency.

Resources, security, and CDF analysis: ACA reports lower CPU usage (3.16% vs. 11.01%), but this occurs alongside markedly higher failure rates, suggesting constrained delivery capacity rather than efficiency. CDF analysis (Figs. 5-10) confirms Koyeb's tighter tail distribution: ACA exhibits broader spread and higher extreme latencies under stress. Effect sizes are large at low and extreme concurrency (Cohen's d ≥ 1.48 at C = 5000); intermediate concurrency shows smaller effects consistent with non-significant *P*-values. Both platforms achieve a 100% security success rate, with Koyeb detecting and responding faster (≈408 ms vs. ≈765 ms for ACA).

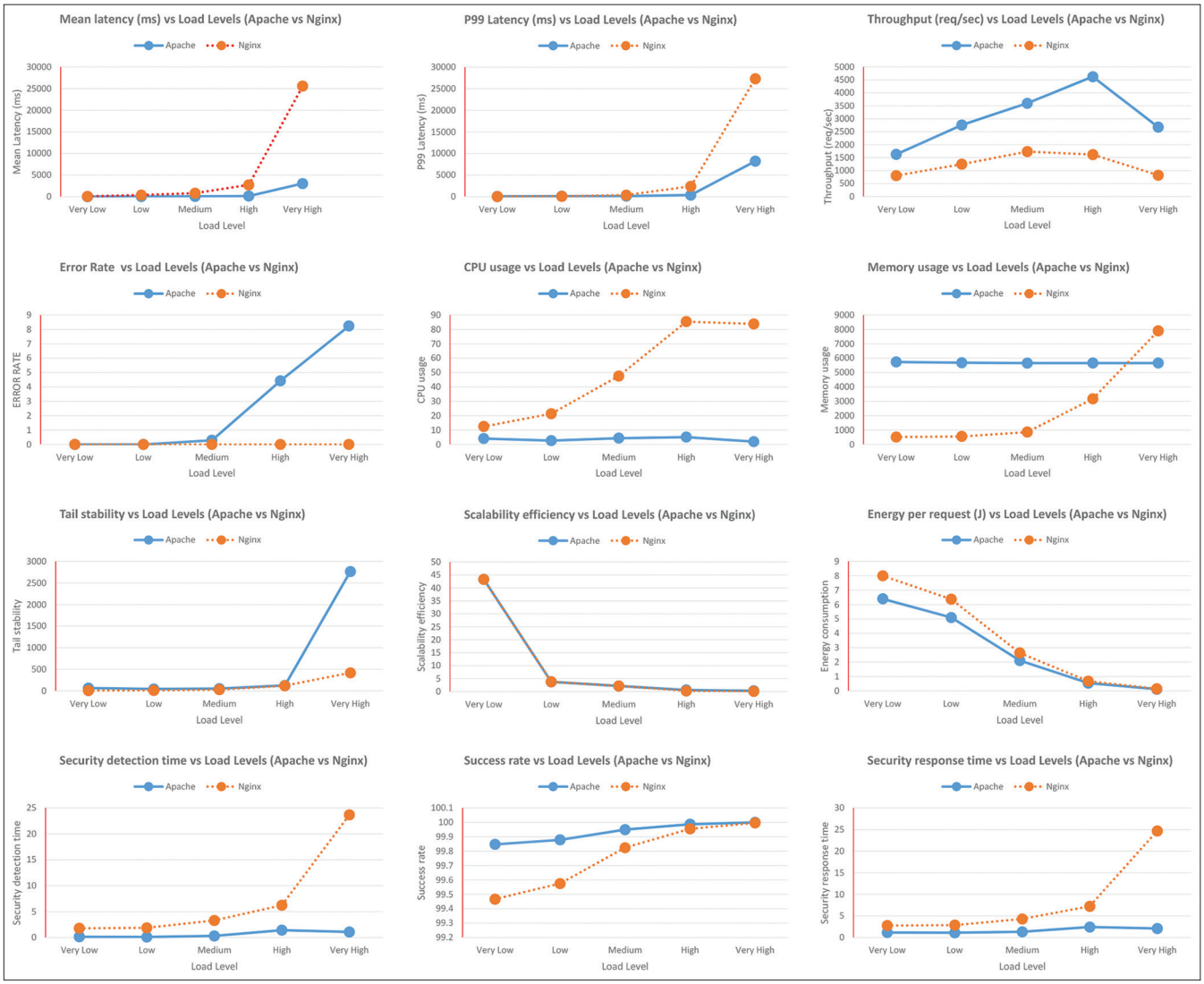


**Fig. 3.** Multi-metric comparison of Apache and Nginx across five load categories (12 metrics).

Figs. 5-10 present empirical CDFs for mean and P99 latency on a log-scale x-axis, complementing the percentile summaries by revealing the full shape of each platform's latency distribution, including the frequency and severity of rare high-latency events. Apache's distribution lies left of Nginx across the majority of the CDF range, confirming lower latency under most conditions, while Nginx exhibits a comparatively heavier tail under high load; for managed platforms, Koyeb shows a tighter tail than ACA, with ACA exhibiting broader spread and higher extreme latencies under stress, a pattern that is amplified when the CDF is restricted to high-concurrency conditions ($C \geq 500$) in Fig. 9.

### 4.3. Statistical Validation Results

Tables 7-9 present the results of formal non-parametric hypothesis testing across all platform comparisons. For most concurrency levels, the differences are statistically significant ($P < 0.05$), with the strongest effects observed at low and high concurrency ($P < 0.001$). However, at intermediate concurrency levels ($C = 500$ and $C = 2000$), the Mann–Whitney U test yields non-significant $P$-values ($P = 0.406$ and $P = 0.887$), indicating comparable steady-state performance under moderate load. Effect sizes confirm that the observed differences are not only statistically detectable but practically meaningful: Apache versus Nginx comparisons yield large-to-very-large effects across all concurrency levels (Cohen's $d \geq 1.23$, reaching

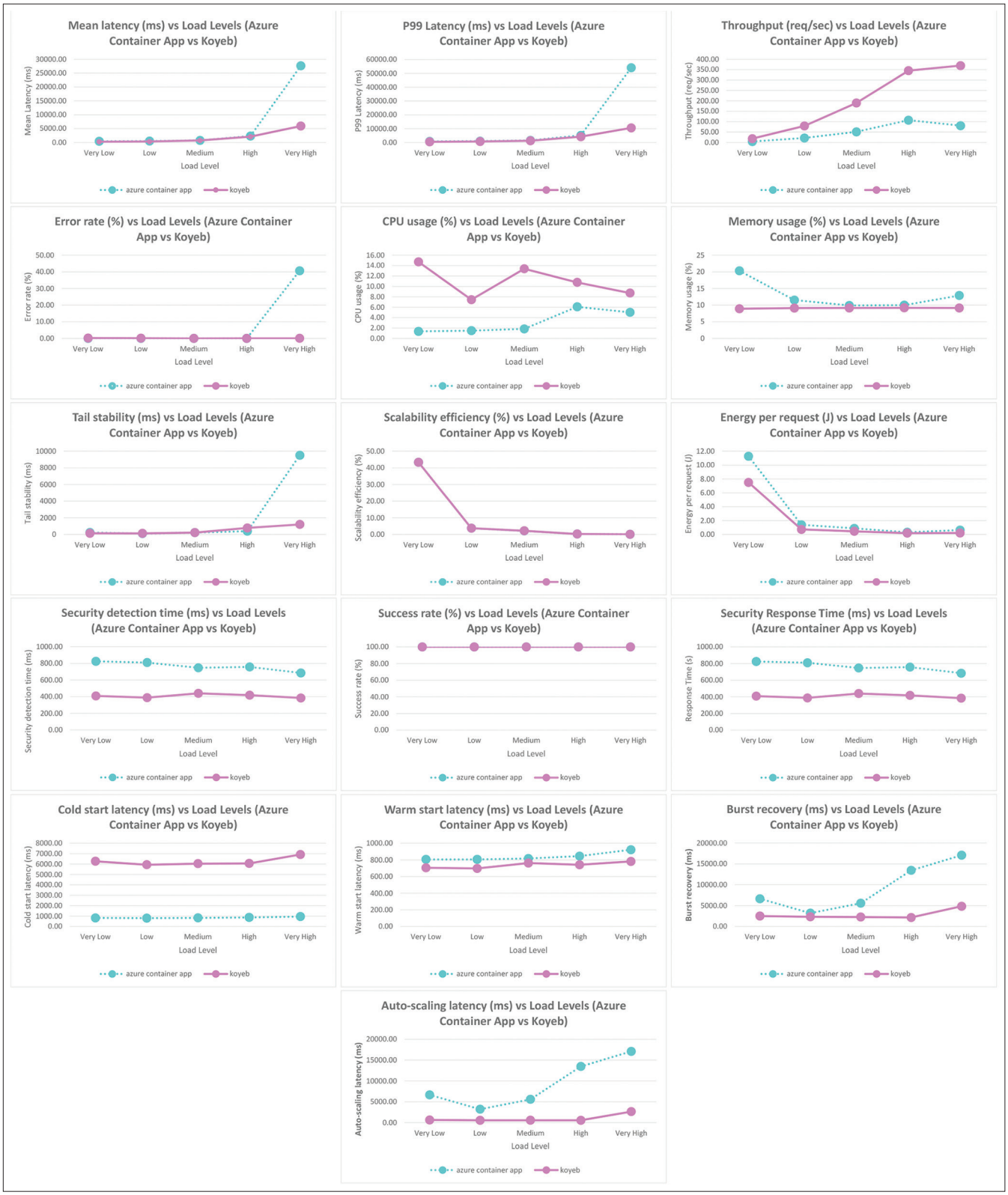


**Fig. 4.** Multi-metric comparison of Azure Container Apps and Koyeb across five load categories (16 metrics).

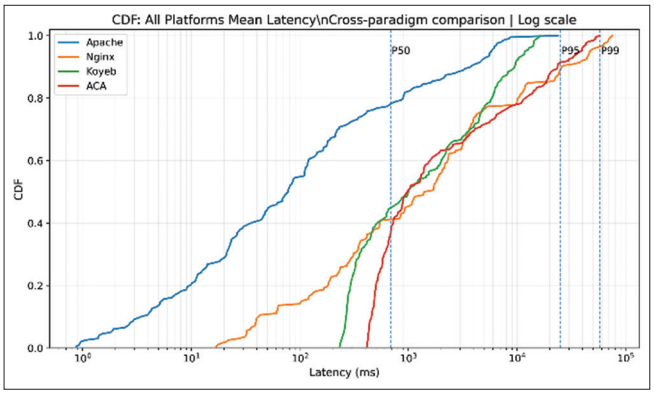


**Fig. 5.** All-platform cumulative distribution functions.

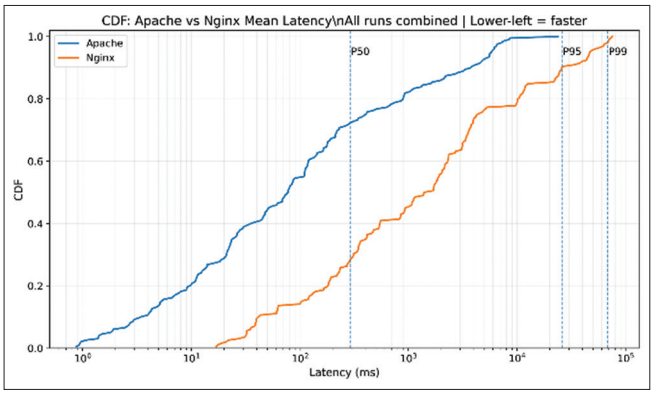


**Fig. 6.** Apache versus Nginx cumulative distribution functions.

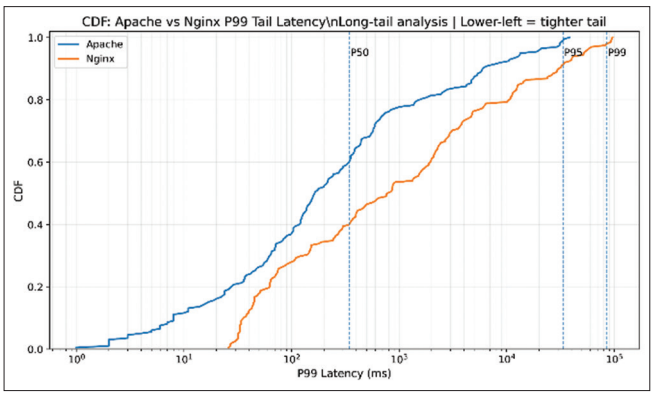


**Fig. 7.** Apache versus Nginx P99 cumulative distribution functions.

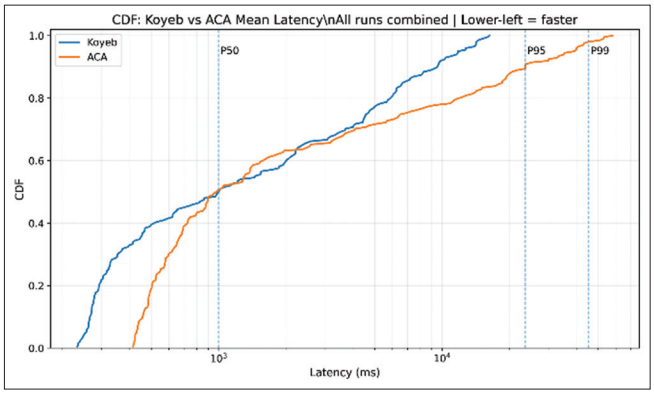


**Fig. 8.** Koyeb versus Azure Container Apps cumulative distribution functions.

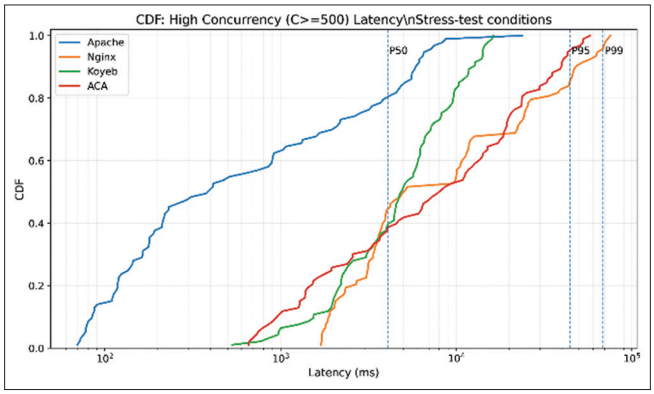


**Fig. 9.** High-concurrency cumulative distribution functions.

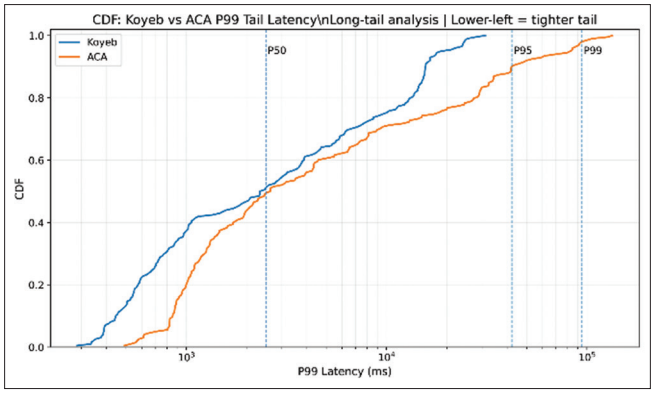


**Fig. 10.** Koyeb versus Azure Container Apps P99 cumulative distribution functions.

d = 6.40 at C = 1000), whereas Koyeb versus ACA shows large effects at low and extreme concurrency (Cohen's d ≥ 1.48 at C = 5000), with intermediate concurrency showing smaller effects consistent with the non-significant *P*-values in Table 8.

### 4.4. Cross-Topology Deployment Outcomes (Local Loopback vs Internet-hosted Platform)

Table 10 compares Apache (loopback, TCR ≈ 0) and Koyeb (Internet, TCR ≈ 0.09 at C = 10) under DTAB Rule 2,

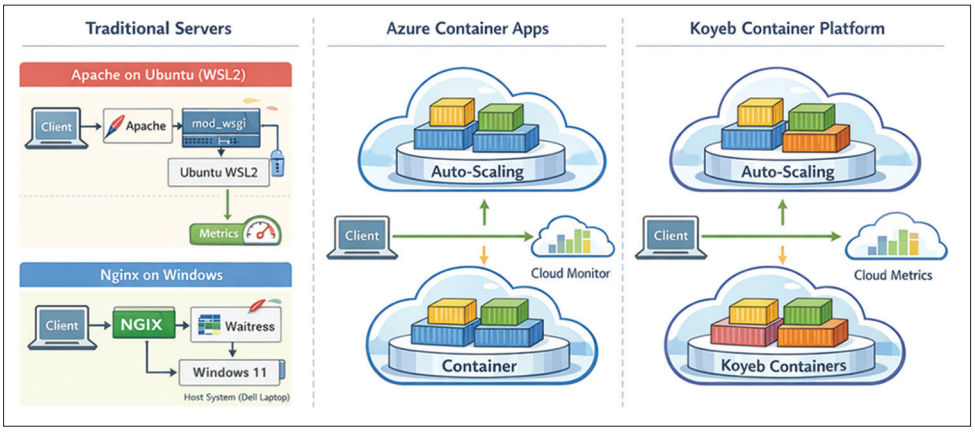


**Fig. 11.** Experimental deployment stacks and measurement points for traditional web-server stacks and container-based managed platforms (Azure Container Apps and Koyeb). (Conceptual diagram generated with artificial intelligence assistance based on the authors' implementation).

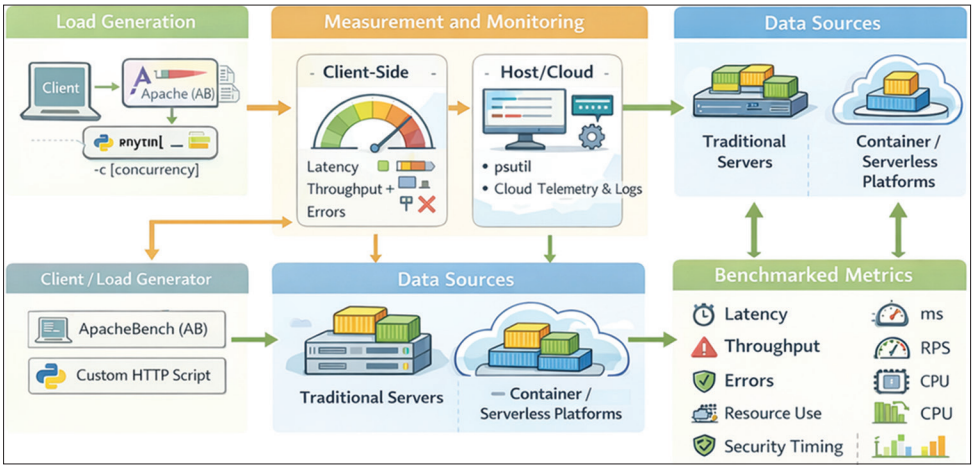


**Fig. 12.** Measurement workflow showing client-side load generation, host-level monitoring, and cloud-platform telemetry integration. (Conceptual diagram generated with artificial intelligence assistance based on the authors' methodology).

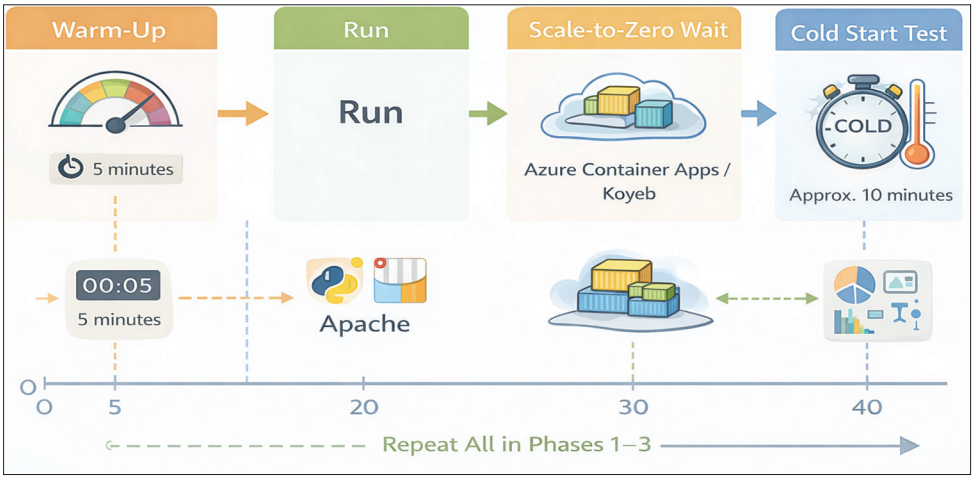


**Fig. 13.** Three-phase experimental workflow showing initialization and warm-up (Phase 1), load execution and measurement (Phase 2), and recovery with scale-to-zero and cold-start observation (Phase 3). Phases represent sequential stages within a controlled experimental session. Conceptual diagram generated with artificial intelligence assistance based on the authors' experimental procedure.

reporting client-observed end-to-end deployment outcomes rather than intrinsic server-processing performance. At C = 10, ACA's observed latency of 492.6 ms includes approximately 181 ms of network RTT (TCR ≈0.37), confirming that over one-third of its end-to-end latency is attributable to transport-layer effects; Koyeb's 269.8 ms includes only 25 ms of RTT (TCR ≈0.09), indicating that platform processing dominates its profile. Under loopback topology, Apache achieves lower client-observed latency and higher throughput; under Internet topology, Koyeb shows lower error rate, tighter tail spread, and lower EPR. These differences reflect deployment context – network path, provider routing, and orchestration policy – not server-engine superiority. Practitioners should favor traditional stacks where infrastructure is provisioned close to users and load is predictable; managed platforms are preferable when demand is variable, and robustness under overload is prioritized.

## 5. CONCLUSION

This study presented a unified experimental benchmarking evaluation comparing traditional web-server hosting with container-based serverless platforms for a Django e-commerce application, addressing five research gaps identified in prior literature.

### 5.1. Contribution 1 (Unified Framework)

A reproducible protocol applying 16 m (12 shared + 4 elasticity) across five load profiles (C = 10 to C = 10,000; $n$ = 500–50,000) to both paradigms under identical workloads and applications, enabling principled cross-paradigm comparison that reveals platform-specific trade-offs masked by isolated evaluations (Gaps 1–5). The protocol demonstrated that 12 shared metrics are sufficient to characterize steady-state performance, whereas four additional elasticity metrics are necessary to capture container-orchestration effects in managed platforms.

### 5.2. Contribution 2 (Empirical Evaluation)

Traditional stacks (within-topology, DTAB Rule 1): Apache achieved ×9 lower mean latency and ×2.5 higher throughput; Nginx achieved zero errors and tighter tail spread (117 ms vs. 610 ms P99–P95). All differences confirmed statistically ($P < 0.001$, Cohen's d ≥ 1.23). Managed platforms (within-topology, DTAB Rule 1): Koyeb outperformed ACA in steady-state performance (×3.3 lower mean latency, ×3.7 higher throughput, 0.09% vs. 8.12% error) and scaling responsiveness (auto-scaling: 978 vs. 9,198 ms); ACA provided faster cold-start activation (856 vs. 6,243 ms). Omnibus Kruskal–Wallis confirms overall distributional differences across all four platforms ($H = 357.46$, $P = 3.61 \times 10^{-77}$) (Gaps 1–4).

### 5.3. Contribution 3 (Cross-Paradigm Interpretation)

Cross-topology results (DTAB Rule 2) represent end-to-end client-observed deployment outcomes, including network-path effects; no intrinsic server-processing-time superiority is claimed for local stacks. Apache on provisioned local infrastructure is preferable for sustained, latency-critical traffic where throughput requirements exceed approximately 3000 req/s and predictable load is expected; managed platforms are preferable when demand is variable, and robustness under overload is prioritized. Elasticity comes with orchestration overhead: cold-start latencies (856 ms for ACA, 6,243 ms for Koyeb) and auto-scaling latencies demonstrate user-visible delays that must be measured explicitly. Tail behavior further reveals robustness differences: Nginx's zero error rate despite lower throughput demonstrates that robustness and tail latency stability can be decoupled from peak capacity. Resource metrics require context-aware interpretation, as CPU, memory, and EPR are collected under different scopes across environments and support only qualitative cross-environment comparison (Gaps 1, 3).

### 5.4. Contribution 4 (DTAB Model)

The DTAB model (Equations 1–2) provides a reusable framework for any cross-environment benchmarking study. TCR values (TCR_Apache ≈ 0, TCR_Koyeb ≈ 0.09, TCR_ACA ≈ 0.37 at C = 10) quantify network-path influence; three interpretation rules enable principled comparisons with appropriate caveats (Gaps 1, 3). The DTAB decomposition and TCR metric advance the methodological foundation for cross-paradigm web-system benchmarking beyond *ad hoc* disclaimers.

### 5.5. Limitations

(1) single-endpoint navigation-style workload, not transactional; (2) WSL2 virtualization overhead on Apache results; (3) configuration non-parity across stacks – OS, WSGI runtime, and container runtime differ; (4) non-comparable resource measurement scopes (psutil vs. provider telemetry); (5) single-client, single-region experiments; (6) EPR is a utilization-based proxy, not a direct power measurement. Future work will extend to native Linux hosts, multi-endpoint user journeys, multi-region experiments, and direct power instrumentation.

### 5.6. AI Tools Disclosure

Generative AI (ChatGPT, OpenAI) assisted in drafting portions of the manuscript text and generated three conceptual diagrams (Figs. 11-13) illustrating the experimental

workflow and deployment architecture; these figures contain no experimental data. Figs. 1-10 were generated directly from experimental measurements. All experimental design, data collection, statistical analysis, and result interpretation were performed by the authors, who retain full responsibility for all content.

## REFERENCES


[1] M. Basalla, J. Schneider, M. Luksik, R. Jaakonmäki and J. Vom Brocke. "On latency of e-commerce platforms". *Journal of Organizational Computing and Electronic Commerce*, vol. 31, no. 1, pp. 1-17, 2021.

[2] Z. Li, L. Pons, S. Petit, J. Sahuquillo and J. Pons. "*TailBench++: Flexible Multi-Client, Multi-Server Benchmarking for Latency-Critical Workloads*". [arXiv Preprint]; 2025.

[3] J. Dean and L. A. Barroso. "The tail at scale". *Communications of the ACM*, vol. 56, no. 2, pp. 74-80, 2013.

[4] W. Wicoksono, H. A. Mustaqhim, P. P. Anwas and L. N. L. Badratul. "Performance comparison of NGINX, apache, and lighttpd using WRK on a debian". *bit-Tech*, vol. 8, no. 1, pp. 670-680, 2025.

[5] J. Yang and A. Abraham. "Analyzing the features, usability, and performance of deploying a containerized mobile web application on serverless cloud platforms". *Future Internet*, vol. 16, no. 12, p. 475, 2024.

[6] V. Besozzi, M. Della Bartola, P. Dazzi and M. Danelutto. "High-performance serverless computing: A systematic literature review on serverless for HPC, AI, and big data". *IEEE Access*, vol. 13, pp. 195611-195656, 2025.

[7] T. Schirmer, N. Japke, S. Greten, T. Pfandzelter and D. Bermbach. "*The Night Shift: Understanding Performance Variability of Cloud Serverless Platforms*". In: *Proceedings of the 1st Workshop on Serverless Systems, Applications and Methodologies*, pp. 27-33, 2023.

[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] A. Farhadian, M. Bastam, E. Ataei and M. Babagoli. "Performance evaluation of apache and nginx web servers on docker, podman, and LXC containers". *Nashriyyah-i Muhandisi-i Barq va Muhandisi-i Kampyutar-i Iran*, vol. 120, no. 2, p. 79, 2025.

[10] M. Copik, G. Kwasniewski, M. Besta, M. Podstawski and T. Hoefler. "*Sebs: A Serverless Benchmark Suite for Function-as-a-Service Computing*. In: *Proceedings of the 22nd International Middleware Conference*, pp. 64-78, 2021.

[11] T. Yu, Q. Liu, D. Du, Y. Xia, B. Zang, Z. Lu, P. Yang, C. Qin, and H. Chen. "*Characterizing Serverless Platforms with Serverlessbench*". In: *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp. 30-44, 2020.

[12] P. Melo, G. J. Da Silva, A. B. Vanderlei, D. Beserra and J. Araujo, "Performance Evaluation of Containerized Virtualization Platforms," *SN Computer Science,* vol. 6, no. 7, p. 771, 2025.

[13] S. Werner, M. C. Borges, K. Wolf and S. Tai. "A comprehensive experimentation framework for energy-efficient design of cloud-native applications". In: *2025 IEEE 22nd International Conference on Software Architecture (ICSA)*. IEEE, United States, pp. 176-186, 2025.

[14] J. Legler, S. Werner, M. C. Borges and S. Tai. "Service-level energy modeling and experimentation for cloud-native microservices". In: *International Conference on Service-Oriented Computing*. Springer, Berlin, pp. 230-247, 2025.

[15] M. Ghattas, S. Odeh and A. M. Mora. "Predicting website performance: A systematic review of metrics, methods, and research gaps (2010-2024)". *Computers*, vol. 14, no. 10, p. 446, 2025.

[16] D. Kunda, S. Chihana and M. Sinyinda. "Web server performance of apache and nginx: A systematic literature review". *Computer Engineering and Intelligent Systems*, vol. 8, pp. 43-52, 2017.

[17] J. Scheuner, R. Deng, J. P. Steghöfer and P. Leitner. "Crossfit: Fine-grained benchmarking of serverless application performance across cloud providers". In: *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. IEEE, United States, pp. 51-60, 2022.

[18] M. A. Pratama, O. Nurdiawan, A. R. Dikananda, D. Pratama and D. A. Kurnia. "Comparative analysis of serverless container service performance between google cloud run and AWS app runner in cross-cloud architecture". *Journal of Artificial Intelligence and Engineering Applications (JAIEA)*, vol. 5, no. 2, pp. 2531-2539, 2026.

[19] S. Henning and W. Hasselbring. "Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud". *Journal of Systems and Software*, vol. 208, p. 111879, 2024.

[20] S. Henning, A. Vogel, E. Perez-Wohlfeil, O. Ertl and R. Rabiser. "*When Should I Run my Application Benchmark? Studying Cloud Performance Variability for the Case of Stream Processing Applications*". In: *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, pp. 400-410, 2025.

[21] H. Kasture and D. Sanchez. "Tailbench: A benchmark suite and evaluation methodology for latency-critical applications". In: *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, United States, pp. 1-10, 2016.

[22] D. Kliazovich, P. Bouvry and S. U. Khan. "GreenCloud: A packet-level simulator of energy-aware cloud computing data centers". *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263-1283, 2012.

[23] F. Palumbo, G. Aceto, A. Botta, D. Ciuonzo, V. Persico and A. Pescapé. "Characterization and analysis of cloud-to-user latency: The case of Azure and AWS". *Computer Networks*, vol. 184, p. 107693, 2021.

[24] D. Purohith, J. Mohan and V. Chidambaram. "*The Dangers and Complexities of Sqlite Benchmarking*." In: *Proceedings of the 8th Asia-Pacific Workshop on Systems*, pp. 1-6, 2017.

[25] Z. Li, M. Kihl, Q. Lu and J. A. Andersson. "Performance Overhead Comparison between Hypervisor and Container Based Virtualization". In: *2017 IEEE 31st International Conference on advanced information networking and applications (AINA)*. IEEE, pp. 955-962, 2017.

[26] M. Larabel. "*Ubuntu 24.04 via WSL2 on Windows 11 25H2 Performance Benchmarks*." Phoronix. Available from: https://www.phoronix.com/review/wsl2-windows11-25h2 [Last accessed on 2026 Jan 15].

[27] R. Sime, N. Sezgin and F. Ağgün. "An integrated web security application: Integration of nginx reverse proxy, Fail2ban, Waf, postgresql and laravel". *Balkan Journal of Electrical and Computer Engineering*, vol. 13, no. 1, pp. 106-111, 2025.

[28] Microsoft. "*Set Scaling Rules in Azure Container Apps (KEDA-Powered Scaling*)", 2026. Available from: https://learn.microsoft.com/en-us/azure/container-apps/scale-app [Last accessed on 2026 Jan 15].

[29] Koyeb. "*Scale-to-Zero*". *Koyeb Documentation,* 2025. Available from: https://www.koyeb.com/docs/run-and-scale/scale-to-zero [Last accessed on 2026 Jan 15].