Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

15th February 2026. Vol. 5. No. 2; e-ISSN: 2808-4519

Comparative Analysis of Serverless Container Service Performance Between Google Cloud Run and AWS App Runner in Cross-Cloud Architecture

Muhammad Adithya Pratama[1*], Odi Nurdiawan[2], Arif Rinaldi Dikananda[3], Denni Pratama[4], Dian Ade Kurnia[5]

1,2,3,4,5 Sekolah Tinggi Manajemen Informatika dan Komputer IKMI Cirebon
adutya37@gmail.com[1], odynurdiawan@gmail.com[2], rinaldi21crb@gmail.com[3], pratamadenni@gmail.com[4], dianade2012@gmail.com[5]*

Abstract

Research on the performance of serverless container services is becoming increasingly important as the need for modern distributed and cross-cloud architectures grows. This study analyzes the performance of two leading serverless services, Google Cloud Run and AWS App Runner, in a cross-cloud architecture scenario. Testing was conducted using identical parameters, including container configuration, region, memory, vCPU, and concurrency. Performance testing included p95 latency, throughput, and error rate metrics using loads of up to 1000 virtual users. The results showed that Google Cloud Run provided more stable performance with p95 latency of 47–71 ms, throughput of 436–438 RPS, and 0% error rate. In contrast, AWS App Runner showed p95 latency of 490–651 ms with throughput variation of 388–410 RPS and an error rate of 2–4.41%. The difference in performance was due to autoscaling mechanisms, cross-cloud communication overhead, and resource contention. This study provides empirical evidence for selecting the optimal serverless service for distributed architectures.

Keywords: *App Runner, Cloud Run, Cross-Cloud, Serverless.*

1. Introduction

The development of information technology has accelerated significantly in the last two decades, especially in terms of computing, connectivity, and process automation. The digital transformation that has occurred in various sectors has encouraged the adoption of cloud-based systems as the main foundation in supporting modern computing needs. Innovations in container-based computing and serverless architecture have further strengthened the efficiency of application development in dynamic environments, enabling systems to run flexibly without physical server management. In addition, the integration of AI-based technology, container orchestration, and dynamic autoscaling capabilities expands the potential for utilizing cloud services to support various organizational needs [1][5]. Although cloud technology continues to evolve, significant challenges arise, especially in the implementation of serverless and container-based services in cross-cloud architectures. The use of services such as Google Cloud Run and AWS App Runner raises issues of performance consistency, network latency, throughput, and reliability when applications are run cross-cloud. Variations in autoscaling mechanisms, container orchestration policies, and differences in traffic management protocols further exacerbate performance instability. Differences in runtime configuration and cold start management on serverless platforms can also result in performance inconsistencies during high loads, while challenges in function validation and serverless performance monitoring are still rarely discussed in depth [3]. Previous research has contributed to the understanding of serverless technology, container management, and network optimization. Barrak et al. [1] mapped the development of serverless in the context of machine learning, but did not directly compare the performance between serverless platforms. Jeon et al. [5] explored deep learning-based container scheduling to improve cloud reliability, but their focus was not on the performance of serverless services such as Cloud Run or App Runner. Traffic management studies by Li et al. [4] have also not been directed at evaluating the performance of container-based serverless in cross-cloud architectures. This study aims to provide a comparative analysis of the performance of Google Cloud Run and AWS App Runner based on latency, throughput, and autoscaling in cross-cloud conditions, as well as to identify the influence of the internal architecture of both platforms on system stability and efficiency when running with identical workloads. The research approach was conducted through experimental testing using a series of workload scenarios applied consistently on both platforms, with measurements of latency, throughput, and autoscaling activation time to obtain an objective picture of performance.

2. Literature Review

Studies related to the domain of serverless computing show that the performance of serverless services is greatly influenced by cold start mechanisms, service provider architecture, and container orchestration methods. Barrak et al. explain the development of serverless in the context of machine learning and the technical challenges of automatic deployment [1]. The study by Golec et al. shows that cold start is one of the main sources of latency on modern serverless platforms [2], while Wen et al. introduce a more structured performance testing approach to analyze serverless behavior under high load [3].

In the context of container orchestration, Kaiser et al. benchmarked various container technologies and found that container capabilities are highly dependent on orchestration strategies and resource configurations [4]. Jeon et al. developed a hybrid deep learning approach for efficient container scheduling in cloud environments [5], while Usman et al. emphasized the importance of observability in container-based and edge systems to understand overall performance behavior [6].

Research related to distributed architecture also provides an important foundation for this study. Sofia et al. proposed adaptive cross-layer orchestration for containerized applications in dynamic environments [7]. Garbugli et al. introduced QoS middleware for FaaS that can improve latency management in edge and cloud architectures [8]. These findings provide a strong basis that the performance of serverless services in cross-cloud architectures is not only determined by the platform's internal resources, but also by the orchestration, network, and scaling dynamics used by each service.

Research on serverless computing has grown rapidly, covering topics such as autoscaling, resource optimization, and container orchestration performance. Several studies highlight challenges such as cold starts, latency variations, and scalability issues in multicloud environments. However, there has not been much research specifically comparing the performance of cross-cloud platform container serverless services using a standardized experimental approach.

Services such as Google Cloud Run and AWS App Runner are designed to automatically run containers with dynamic scaling. However, the implementation of autoscaling and internal orchestration differs between platforms, leading to potential performance variations..

3. Research Methods

This research methodology was designed to develop the *Rubbish Go* application that aligns with user needs, is feasible for implementation, and contributes meaningfully to addressing community-based waste management challenges. The study adopts an applied research approach by integrating methods of digital product design, idea validation, and systematic prototype testing (usability testing) [6][7].

3.1. Types of research

This research is categorized as quantitative research with a computational experimental approach, which aims to empirically measure the performance of two cross-cloud provider serverless container platforms, namely Google Cloud Run and AWS App Runner. This approach is suitable for producing objective and measurable findings because environmental variables, configurations, and resources can be systematically controlled. All experimental results are presented in the form of numerical data, which is then statistically analyzed to identify significant differences between platforms [6][8].

A purely experimental method is applied to test the causal relationship between independent and dependent variables under controlled conditions. The independent variable is the type of cloud platform, while the dependent variables include performance indicators: latency (p95), throughput, error rate, and autoscaling responsiveness. Control variables such as CPU configuration, memory capacity, deployment region, and container concurrency level are standardized to ensure fair and unbiased test results. Thus, any performance changes that arise can be directly attributed to platform characteristics.

The experimental environment was built through an automation pipeline so that the build, deploy, and test processes ran synchronously on both platforms. This pipeline was run using Google Cloud Build for Cloud Run and AWS CodeBuild for App Runner. The k6 testing tool was used to generate dynamic workloads of up to 1000 virtual users (VUs), enabling observation of steady-state, autoscaling, and stress-load conditions. The experimental design referred to the principles of factorial experimental design, which allowed observation of the main effects and interactions between factors.

All test results are collected in a structured manner and analyzed using statistical approaches, including mixed-effects models, which are used to handle variability between repetitions and strengthen the validity of experimental results [7]. This approach allows for a more accurate interpretation of cross-platform performance, especially when experiments involve repeated measurements under identical conditions.

Overall, this research is empirical, computational, and replicative. All configurations, test scripts, and performance logs are documented to support reproducibility. This controlled experimental design is expected to strengthen the development of a more systematic cross-cloud benchmarking methodology and can be used as a reference in future studies, particularly in the context of serverless container performance [4][6].

3.2. Research design

This study was designed as a series of four interconnected stages of experimentation, in which the output of each stage formed the basis for the next stage.

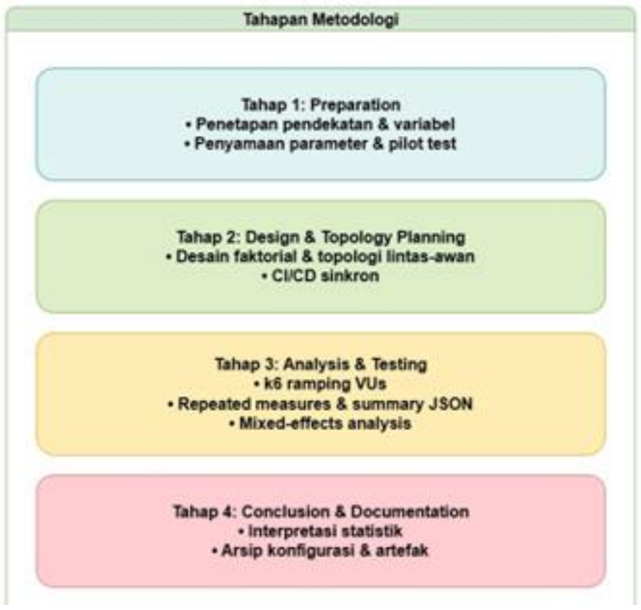

Tahapan Metodologi
Tahap 1: Preparation
• Penetapan pendekatan & variabel
• Penyamaan parameter & pilot test
Tahap 2: Design & Topology Planning
• Desain faktorial & topologi lintas-awan
• CI/CD sinkron
Tahap 3: Analysis & Testing
• k6 ramping VUs
• Repeated measures & summary JSON
• Mixed-effects analysis
Tahap 4: Conclusion & Documentation
• Interpretasi statistik
• Arsip konfigurasi & artefak

Fig. 1: Research Stages

1. Stage 1 — Preparation (Preparation and Formulation)

Table 1: Summary of Work Structure

Kategori	AWS	Cloud Run
Concurrency	80	80
vCPU & Memory	1 vCPU & 2 GB	1 vCPU & 2 GB
Minimum Size (Min Instances)	1	1
Maximum Size (Max Instances)	10	10
Protocol (Health Check)	TCP	TCP
Unhealthy Threshold	5 requests	1 request

This stage includes setting experimental objectives, formulating hypotheses, and standardizing variables. The Cloud Run and App Runner platforms serve as independent variables, while p95 latency, throughput, error rate, and autoscaling responsiveness are dependent variables. Control variables such as region, CPU–memory configuration, concurrency, timeouts, and testing tools (k6) are standardized to maintain consistency. The k6 script is prepared using a ramping scenario of up to 1000 VUs with checks such as p(95) < 1000 ms and http_req_failed < 1%. Preliminary tests are run to validate the endpoint, credentials, DNS, and database connectivity as the experiment baseline.

Stage 2 — Design & Topology Planning

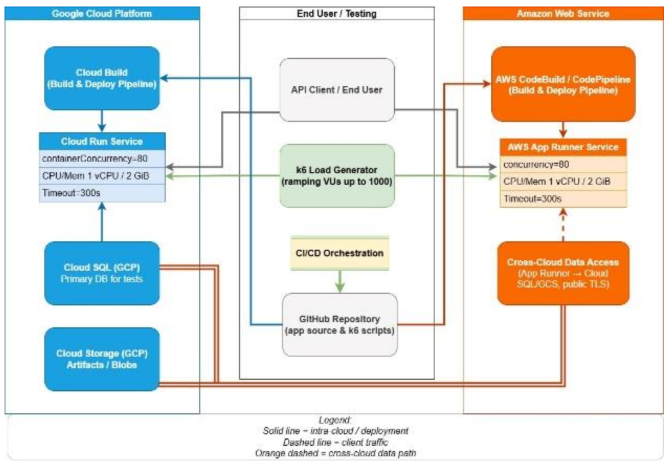
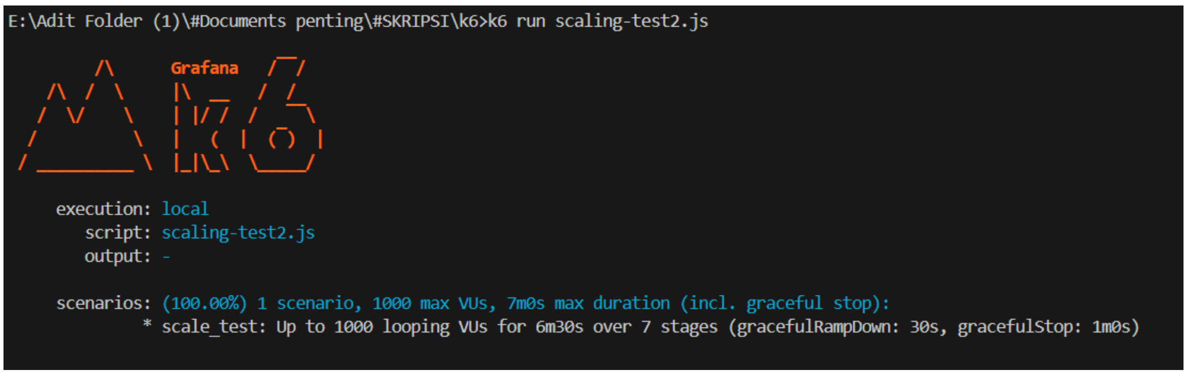

Google Cloud Platform
Cloud Build (Build & Deploy Pipeline)
Cloud Run Service
containerConcurrency=80
CPU/Mem 1 vCPU / 2 GiB
Timeout=300s
Cloud SQL (GCP) Primary DB for tests
Cloud Storage (GCP) Artifacts / Blobs
End User / Testing
API Client / End User
k6 Load Generator (ramping VUs up to 1000)
CI/CD Orchestration
GitHub Repository (app source & k6 scripts)
Amazon Web Service
AWS CodeBuild / CodePipeline (Build & Deploy Pipeline)
AWS App Runner Service
concurrency=80
CPU/Mem 1 vCPU / 2 GiB
Timeout=300s
Cross-Cloud Data Access (App Runner → Cloud SQL/GCS, public TLS)
Legend:
Solid line = intra-cloud / deployment
Dashed line = client traffic
Orange dashed = cross-cloud data path

Fig. 2: Research Topology

At this stage, a factorial design was developed that combined platform variations and mixed workloads. The cross-cloud topology was clarified by separating the application layer (Cloud Run/App Runner) and the data layer (Cloud SQL/Cloud Storage), connected via a secure TLS path. An automation pipeline (Cloud Build, CodeBuild/CodePipeline) is set up to ensure that the build–deploy–test process runs consistently. The execution sequence includes warm-up → steady-state → stress for each factor combination.

2. Stage 3 — Analysis & Testing



```
E:\Adit Folder (1)\#Documents penting\#SKRIPSI\k6>k6 run scaling-test2.js

         /\      Grafana   /‾‾/
    /\  /  \     |\  __   /  /
   /  \/    \    | |/ /  /   ‾‾\
  /          \   |   (  |  (‾)  |
 / __________ \  |_|\_\  \_____/

     execution: local
        script: scaling-test2.js
        output: -

     scenarios: (100.00%) 1 scenario, 1000 max VUs, 7m0s max duration (incl. graceful stop):
              * scale_test: Up to 1000 looping VUs for 6m30s over 7 stages (gracefulRampDown: 30s, gracefulStop: 1m0s)
```

Fig. 3: UI from k6

The experiment was conducted with a gradual load (100 → 500 → 1000 VUs) and repeated to obtain data reliability. The metrics collected included latency distribution (median, p90, p95), throughput, error rate (4xx/5xx), and connection indicators such as connect time and TLS handshake. If there was a region mismatch between the endpoint and the data source, spikes in p95 and error rate were recorded as control findings. After region alignment was performed (e.g., Asia–Singapore), the measurement results were compared to the baseline. All k6 results were exported for descriptive and inferential analysis using appropriate statistical approaches [3][6].

3. Stage 4 — Conclusion & Documentation
The final stage summarizes the experimental results based on a comparison of the performance of both platforms under equivalent conditions, an analysis of the impact of cross-cloud networking on latency and errors, and the practical implications for platform selection. All experimental artifacts—k6 scripts, service configurations, logs, and JSON summaries—are archived to ensure replication. Performance graphs, tables of key parameters, and topology diagrams are included to clarify the results. Thus, the experimental design is complete and ready to be replicated for the next workload variation.

4. Result and Discussion

4.1. Research results

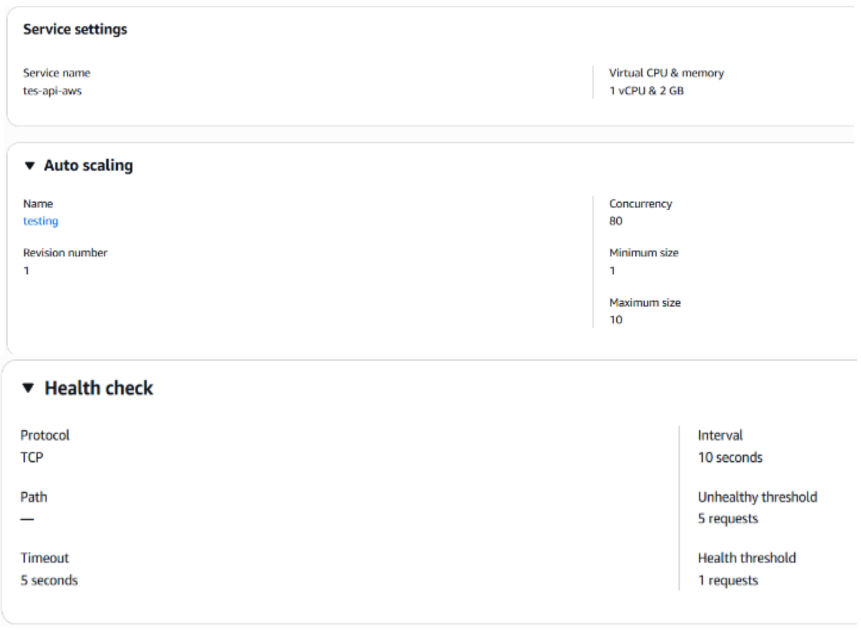
This section presents the results of experiments conducted to address the two main objectives of the study:

1. to empirically compare the performance of Google Cloud Run and AWS App Runner on cross-cloud architectures, and
2. to identify factors causing performance variations such as cold-start, autoscaling mechanisms, and inter-cloud network paths.

The tests were conducted five times using a ramp load of up to 1000 VUs on both platforms with all control variables such as region, CPU/memory, concurrency, timeout, and test scenarios kept the same.

1. Stage 1: Preparation

The preparation phase included setting independent variables (cloud platform), dependent variables (p95 latency, throughput, error rate, autoscaling responsiveness), and control variables such as region, container configuration, concurrency, and k6 testing tools. A pilot test was conducted to ensure that the endpoint, pipeline, and request structure were running stably.



Service settings

Service name	Virtual CPU & memory
tes-api-aws	1 vCPU & 2 GB

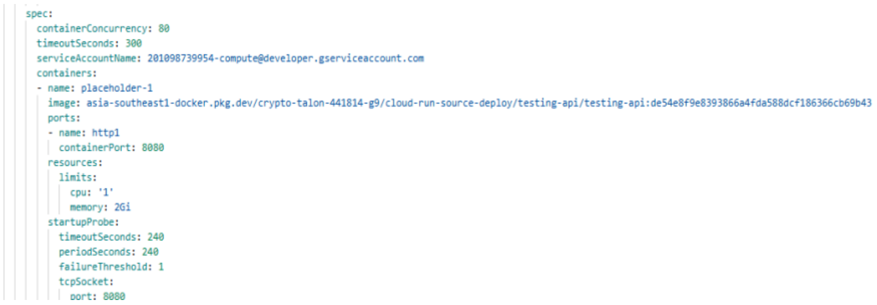
▼ Auto scaling

Name: testing	Concurrency: 80
Revision number: 1	Minimum size: 1
	Maximum size: 10

▼ Health check

Protocol: TCP	Interval: 10 seconds
Path: —	Unhealthy threshold: 5 requests
Timeout: 5 seconds	Health threshold: 1 requests

Fig. 4: AWS Apprunner Configuration



```
spec:
  containerConcurrency: 80
  timeoutSeconds: 300
  serviceAccountName: 201098739954-compute@developer.gserviceaccount.com
  containers:
  - name: placeholder-1
    image: asia-southeast1-docker.pkg.dev/crypto-talon-441814-g9/cloud-run-source-deploy/testing-api/testing-api:de54e8f9e8393866a4fda588dcf186366cb69b43
    ports:
    - name: http1
      containerPort: 8080
    resources:
      limits:
        cpu: '1'
        memory: 2Gi
    startupProbe:
      timeoutSeconds: 240
      periodSeconds: 240
      failureThreshold: 1
      tcpSocket:
        port: 8080
```

Fig. 5: Google Cloud Run Configuration

Table 2: Test Specifications

Kategori	AWS	Cloud Run
Concurrency	80	80
vCPU & Memory	1 vCPU & 2 GB	1 vCPU & 2 GB
Minimum Size (Min Instances)	1	1
Maximum Size (Max Instances)	10	10
Protocol (Health Check)	TCP	TCP
Unhealthy Threshold	5 requests	1 request

2. Stage 2 — Design & Topology Planning

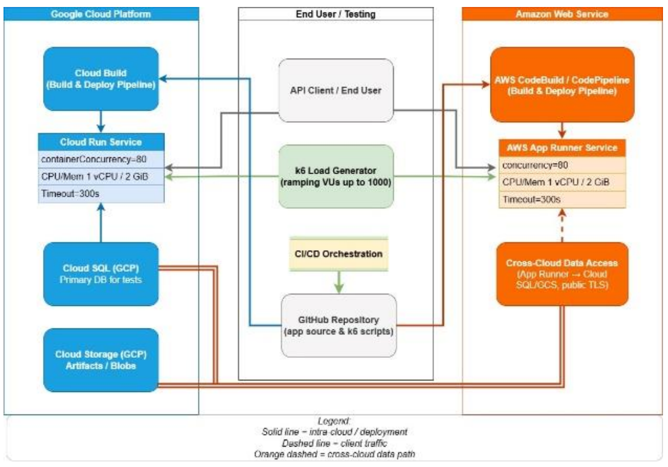

Google Cloud Platform
End User / Testing
Amazon Web Service
Cloud Build (Build & Deploy Pipeline)
API Client / End User
AWS CodeBuild / CodePipeline (Build & Deploy Pipeline)
Cloud Run Service
containerConcurrency=80
CPU/Mem 1 vCPU / 2 GiB
Timeout=300s
k6 Load Generator (ramping VUs up to 1000)
AWS App Runner Service
concurrency=80
CPU/Mem 1 vCPU / 2 GiB
Timeout=300s
Cloud SQL (GCP) Primary DB for tests
CI/CD Orchestration
Cross-Cloud Data Access (App Runner → Cloud SQL/GCS, public TLS)
GitHub Repository (app source & k6 scripts)
Cloud Storage (GCP) Artifacts / Blobs
Legend:
Solid line − intra-cloud / deployment
Dashed line − client traffic
Orange dashed = cross-cloud data path

Fig. 6: Research Topology

The experiment topology separates compute services (Cloud Run/App Runner) and databases (Cloud SQL) to simulate cross-cloud conditions. All builds and deployments are performed synchronously using GitHub Actions to ensure that both platforms receive identical container images.

3. Stage 3 — Analysis & Testing
   The test was conducted for 6 minutes and 30 seconds with a gradual increase up to 1000 VUs. The JSON summary from k6 was used as the main data source. There was a stable pattern in Cloud Run and a fluctuating pattern in App Runner, especially in the first three tests.

Table 3: Cloud Run test results table

Uji	Avg (ms)	p95 (ms)	RPS	Error	Total Req
1	40.52	71.71	436.03	0%	170,397
2	36.29	48.01	437.66	0%	171,082
3	35.80	47.04	437.67	0%	171,137
4	39.12	53.81	436.62	0%	170,595
5	36.42	47.45	437.75	0%	171,062

Cloud Run shows stable patterns across all metrics, low p95 (47–71 ms), and no errors.

Table 4: App Runner test results table

Uji	Avg (ms)	p95 (ms)	RPS	Error	Total Req
1	149.15	651.70	388.12	4.41%	151,675
2	113.10	493.54	403.50	2.16%	157,754
3	122.71	639.75	400.81	2.96%	156,624
4	99.73	490.67	409.68	0.06%	160,072
5	110.86	512.37	403.65	0.11%	157,790

App Runner experienced a high error rate in the initial three tests, fluctuating throughput, and a significantly higher p95 (490–651 ms).

4. Empirical Comparison

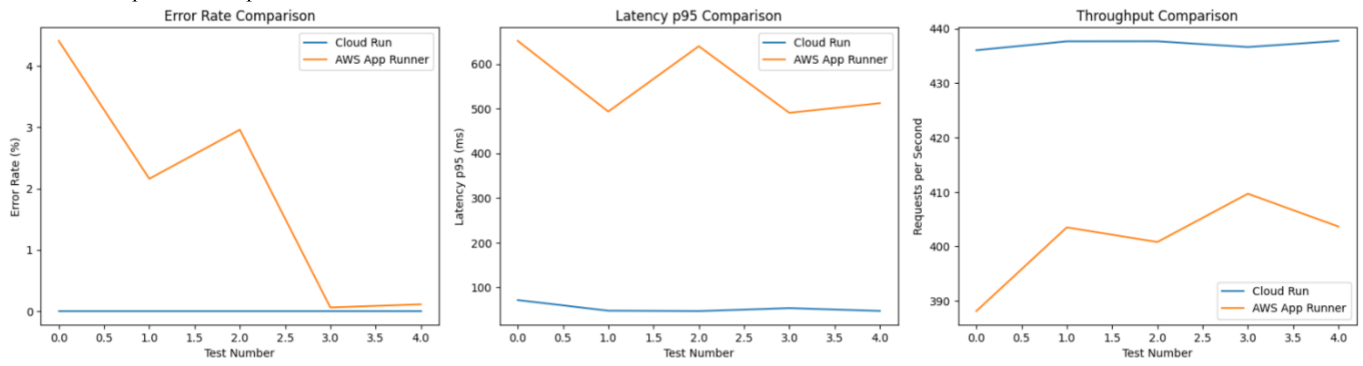


Fig. 7: Chart of Latency, Throughput and Error Rate

Cloud Run is stable at 47–71 ms. App Runner is at 490–651 ms, more than 10 times higher. The intra-cloud communication path on Cloud Run makes responses more efficient than cross-cloud on App Runner. Cloud Run throughput is stable at 436–438 RPS. App Runner is at 388–410 RPS with greater variation. And for error rate, Cloud Run: 0% in all tests. App Runner: 2–4% error in the first three tests. The dominant source of error: timeouts and backlogs due to slow automatic scaling.

5. Identify Factors Causing Variation
   The results of the experiment show that performance variations are influenced by three main factors:
   a. Autoscaling Mechanism
      1. Cloud Run uses concurrency-based scaling → faster & more stable
      2. App Runner uses request-based scaling → triggers backlog → high error & latency
      3.

   This is in line with serverless performance theory [3][6].
   b. Cross-Cloud Network Path
      1. App Runner → cross-cloud Cloud SQL → connect p95: 24–30 ms, TLS p95: 60–66 ms
      2. Cloud Run → intra-GCP → p95 connection: 0 ms
      3.

   Consistent with cross-cloud routing theory [1][8].
   c. Cold-Start and Initial Ramp-Up
      1. App Runner shows significant variation at the beginning of the test
      2. Cloud Run does not show significant cold-start

   These three factors explain the consistent performance differences across all tests.

6. Stage 4 — Conclusion & Documentation
   All experimental artifacts are stored in full: k6 scripts, pipeline logs, Cloud Run/App Runner configuration YAML, and summary JSON. The results show that Cloud Run excels significantly in all metrics and system stability.

4.2. Discussion of Results

The discussion relates the experimental results to theory and previous research findings. Cloud Run exhibits low and stable p95 due to its responsive scaling mechanism and efficient intra-cloud communication paths. This pattern is consistent with research on serverless efficiency and modern container performance [3][6][7].

In contrast, App Runner exhibits high latency, initial error rates, and throughput fluctuations due to backlog-based scaling, instance initialization delays, and cross-cloud network paths. This supports ideas related to multi-cloud architecture overhead and serverless service performance variations [1][2][8].

Overall, Cloud Run proves superior in:

1. p95 latency
2. autoscaling performance
3. stable throughput
4. zero error rate
5. network path efficiency

Meanwhile, App Runner is affected by:

1. cross-cloud communication paths
2. slower request-based scaling
3. cold-start and ramp-up delays

These results fill a gap in previous research with direct empirical evidence regarding the impact of cross-cloud topologies on serverless container performance.

5. Conclusion and Recommendations

5.1 Conclusions

This study was conducted to answer three main questions regarding the performance comparison between Google Cloud Run and AWS App Runner in a cross-cloud architecture. Based on the entire series of experiments, data analysis, and discussions that have been carried out, several important conclusions can be formulated as follows.

First, empirical test results show that Google Cloud Run has more stable, efficient, and consistent performance than AWS App Runner across all key metrics measured. Cloud Run recorded low p95 latency of 47–71 ms, stable throughput in the range of 436–438 RPS, and an error rate of 0% in five tests. In contrast, AWS App Runner recorded a much higher p95 latency of 490–651 ms, lower throughput in the range of 388–410 RPS, and an error rate of up to 4% in the first three tests. This difference in performance shows that Cloud Run is better able to handle high loads and maintain fast responses in cross-cloud scenarios.

Second, the factors causing performance variations can be clearly identified by observing connection metrics, TLS handshakes, throughput fluctuations, and error rates. AWS App Runner experienced increased connect times, longer TLS durations, and significant error rates in several tests. The contributing factors include cross-cloud communication paths, longer instance initialization times, backlog-based scaling mechanisms, and resource contention in the early stages of load increase. In contrast, Cloud Run did not exhibit the same variations because all components run within a single internal network domain, thereby maintaining communication latency, scaling stability, and throughput consistency.

Third, the evaluation methodology applied proved to be appropriate, measurable, and replicable. The use of strict variable controls (region, CPU/memory, concurrency, workload pattern), ramping load scenarios up to 1000 VUs, and structured data export mechanisms ensured that the test results were objective and directly comparable between platforms. Five test repetitions produced stable data and reinforced the validity of the research conclusions.

Overall, this research successfully answered all research questions and provided a comprehensive empirical understanding of the performance differences between containerless serverless services on cross-cloud architectures, while confirming that Cloud Run delivers better performance than App Runner under identical load conditions and equivalent test environments.

5.2 Recommendations

Based on the research findings and practical implications that have emerged, several recommendations are put forward as follows.

a. Theoretical Recommendations

To enrich the theoretical foundation regarding the performance of serverless container services, workload variations need to be expanded in future research. Additional tests such as database write operations, intensive parallel processing, payload size variations, and streaming tests can provide a more comprehensive picture of the behavior of serverless platforms in complex conditions. With this expansion in scope, theories regarding the efficiency of serverless platforms, autoscaling models, and inter-cloud network characteristics can be strengthened through more diverse and in-depth empirical findings.

b. Practical Recommendations

When implementing cross-cloud architectures in production systems, developers and practitioners need to pay attention to the proximity between compute and data components, given that this study shows that the separation of components between clouds has a significant impact on latency, tail latency, and error rates. In addition, autoscaling configurations need to be adjusted to operational load patterns:

1. Concurrency-based scaling, such as in Cloud Run, is more effective for intensive and bursty workloads.
2. Request-based scaling, such as App Runner, requires more careful capacity planning to reduce the risk of backlogs.

These architectural and configuration adjustments can improve system reliability and optimize service costs and responsiveness.

c. Academic Recommendations

The methodological design used in this study, which includes strict variable control, structured load scenarios, experiment replication, and analysis of raw metrics, can serve as a standard framework for future research on cloud computing, serverless computing, and distributed systems. This testing model enables further research to obtain more valid and directly comparable results. This methodology can also be applied in learning processes, practicums, and experimental studies related to cloud-native performance engineering.

d. Technological Recommendations

From a technology development perspective, the findings of this study indicate that improvements in several technical aspects can have a significant impact on the performance of serverless container platforms, especially in cross-cloud architectures. These recommendations include:

1. Optimizing the autoscaling algorithm to respond more quickly to load changes.
2. Improving data-plane routing to reduce inter-cloud network overhead.
3. Improving the cold-start and container initialization mechanisms.
4. Improving container orchestration efficiency to maintain stable throughput under high loads.

Progress in these areas can result in cloud services that are more adaptive, responsive, and tailored to the needs of large-scale applications in the era of cloud-native architecture.

References

[1] A. Barrak, F. Petrillo, and F. Jaafar, “Serverless on machine learning: A systematic mapping study,” IEEE Access, vol. 10, pp. 99337–99352, 2022.

[2] M. Golec et al., “Cold Start Latency in Serverless Computing: A Systematic Review, Taxonomy, and Future Directions,” arXiv preprint arXiv:2310.08437, 2023, doi: 10.48550/arXiv.2310.08437.

[3] J. Wen et al., “SuperFlow: Performance Testing for Serverless Computing,” arXiv preprint arXiv:2306.01620, 2023, doi: 10.48550/arXiv.2306.01620.

[4] S. Kaiser, A. Tosun, and T. Korkmaz, “Benchmarking container technologies on ARM-based edge devices,” IEEE Access, vol. 11, pp. 107331–107347, 2023.

[5] J. Jeon, S. Park, B. Jeong, and Y. Jeong, “Efficient container scheduling with hybrid deep learning,” IEEE Access, vol. 12, pp. 65166–65177, 2024.

[6] M. Usman, S. Ferlin, A. Brunström, and J. Taheri, “A survey on observability of distributed edge & container-based microservices,” IEEE Access, vol. 10, pp. 86904–86919, 2022.

[7] Á. Sofia, D. Dykeman, P. Urbanetz, A. Galal, and D. Dave, “Dynamic, context-aware cross-layer orchestration of containerized applications,” IEEE Access, vol. 11, pp. 93129–93150, 2023.

[8] A. Garbugli, A. Sabbioni, A. Corradi, and P. Bellavista, “TEMPOS: QoS management middleware for edge cloud computing FaaS,” IEEE Access, vol. 10, pp. 49114–49127, 202.