Volume 7, Nomor 1, 2026, hlm 322-336
INFOTECH: Jurnal Informatika Teknologi
p ISSN 2722-9378 | e ISSN 2722-9386
http://jurnal.sttmcileungsi.ac.id/index.php/infotech

Optimalisasi Performa Web Bhumi Aveshana Menggunakan Arsitektur Serverless pada Google Cloud Platform

Bhumi Aveshana Web Performance Optimization Using Serverless Architecture on Google Cloud Platform

Suryanto*, Suwito Muda Karana Indra Waspada

*Jurusan Teknik Elektro, Universitas Bina Sarana Informatika, Jl. Kramat Raya No.98, Senen, Jakarta Pusat, Indonesia

INFORMASI ARTIKEL

Article History:
Submission: 22-05-2026
Revised: 12-06-2026
Accepted: 00-06-2026

Kata Kunci:
Cloud computing;
arsitektur serverless;
Google Cloud Platform;
optimalisasi web;
skalabilitas otomatis

Keywords:
Cloud computing;
serverless architecture;
Google Cloud Platform;
web optimization;
automatic scalability

Korespondensi:
Suryanto
suryanto.syt@bsi.ac.id

ABSTRAK

Perkembangan teknologi *cloud computing* telah mengubah paradigma pengembangan aplikasi web modern, namun banyak organisasi masih menghadapi tantangan dalam merancang arsitektur yang optimal untuk menyeimbangkan performa dan skalabilitas. Penelitian ini bertujuan merancang dan mengimplementasikan arsitektur *cloud computing* yang optimal untuk web Bhumi Aveshana menggunakan pendekatan serverless pada *Google Cloud Platform* (GCP). Metodologi penelitian mengadopsi arsitektur *serverless* komprehensif dengan mengintegrasikan *Firebase Hosting* untuk *frontend*, *Cloud Run* untuk backend API, *Cloud* SQL *PostgreSQL* untuk basis data, dan Cloud Storage untuk penyimpanan media. Pengujian dilakukan menggunakan *Apache Bench* untuk *load testing, Cloud Monitoring* untuk observabilitas *real-time*. Hasil penelitian menunjukkan pencapaian performa superior dengan *response time* rata-rata 78,0ms untuk 95% *request*s, kemampuan menangani 500 pengguna konkuren dengan *error rate* 0%, serta mekanisme *auto-scaling* yang responsif dari 1 hingga 4 *instance* dalam waktu 34-43 detik. Tingkat ketersediaan sistem mencapai 99,968%, melampaui target SLA 99,9%. Penelitian ini membuktikan bahwa arsitektur *serverless* yang teroptimasi secara signifikan meningkatkan performa dan skalabilitas untuk pengembangan web modern.

ABSTRACT

The development of cloud computing technology has changed the paradigm of modern web application development, but many organizations still face challenges in designing an optimal architecture to balance performance, and scalability. This study aims to design and implement an optimal cloud computing architecture for the Bhumi Aveshana website using a serverless approach on Google Cloud Platform (GCP). The research methodology adopts a comprehensive serverless architecture by integrating Firebase Hosting for the frontend, Cloud Run for the backend API, Cloud SQL PostgreSQL for the database, and Cloud Storage for media storage. Testing was conducted using Apache Bench for load testing, Cloud Monitoring for real-time observability. The results of the study show superior performance with an average response time of 78.0ms for 95% of requests, the ability to handle 500 concurrent users with a 0% error rate, and a responsive auto-scaling mechanism from 1 to 4 instances within 34-43 seconds. The system availability level reached 99.968%, exceeding the 99.9% SLA target. This research proves that optimized serverless architectures significantly improve performance and scalability for modern web development.

1. Pendahuluan

Transformasi digital telah mendorong adopsi cloud computing sebagai fondasi utama pengembangan aplikasi web modern [1][2]. *Cloud computing* menawarkan infrastruktur yang fleksibel, skalabel, dan efisien dibandingkan pendekatan tradisional yang mengandalkan server fisik [3][4]. Urgensi optimalisasi arsitektur *cloud* untuk aplikasi web seperti Bhumi Aveshana, sebuah aplikasi dengan karakteristik *content management system* dan *user-generated content* yang memerlukan skalabilitas tinggi, ketersediaan konsisten, dan performa optimal menjadi semakin kritis [5]. Integrasi *cloud computing* dalam pengembangan web telah secara signifikan meningkatkan performa dan skalabilitas aplikasi [6]. Studi menunjukkan bahwa organisasi yang mengadopsi arsitektur *cloud* modern melaporkan siklus *deployment* 40% lebih cepat [7]. Namun, tanpa perancangan arsitektur *cloud* yang tepat, pengembangan web dapat menghadapi berbagai kendala seperti keterbatasan skalabilitas, *latency* tinggi, dan performa tidak konsisten [5].

Untuk menjawab berbagai kendala tersebut sudah dilakukan studi tentang strategi optimalisasi performa dan skalabilitas dalam pengembangan aplikasi web modern, mengeksplorasi pola arsitektur, mekanisme *caching*, optimasi basis data, dan pemanfaatan CDN [3]. Namun, penelitian tersebut masih bersifat konseptual dan tidak menyajikan implementasi empiris pada lingkungan nyata. Penelitian lainnya membandingkan empat paradigma arsitektur yaitu *monolithic*, *microservices*, *serverless*, dan *edge computing* dan mengungkapkan bahwa sistem *monolithic* jenuh pada beban 1.500 pengguna dengan *respons time* >900 ms, *microservices* stabil hingga 3.000–3.500 pengguna (420–600 ms), sementara serverless menunjukkan elastisitas tertinggi hingga 6.200 permintaan per detik [8]. Temuan penting lainnya adalah bahwa tidak ada satu paradigma arsitektur pun yang sepenuhnya memenuhi kebutuhan skalabilitas dan latensi secara simultan, hibridisasi antarparadigma menjadi strategi paling seimbang [8]. Namun demikian, penelitian ini dilakukan dalam lingkungan uji terkendali dan tidak mencakup aspek *auto-scaling* dinamis serta tidak menguji pada platform GCP.

Berdasarkan identifikasi terhadap *state of the art* di atas, beberapa kesenjangan penelitian yang menjadi motivasi utama penelitian, pertama minimnya implementasi empiris arsitektur *serverless* pada GCP untuk aplikasi web. Sebagian besar penelitian terdahulu berfokus pada AWS Lambda atau Azure Functions [9][10], sementara implementasi *serverless* pada GCP dengan layanan spesifiknya seperti *Cloud Run* dan *Firebase Hosting* masih sedikit dieksplorasi secara empiris. Akibatnya, *best practices* dan tolok ukur performa untuk lingkungan GCP masih terbatas. Penelitian ini mengisi celah tersebut dengan menyediakan bukti empiris dari implementasi web pada GCP.

Kesenjangan kedua belum terintegrasinya evaluasi multi-dimensi (Performa, Skalabilitas, Availability) dalam satu kerangka. Penelitian sebelumnya cenderung fokus pada satu aspek saja. Seperti studi sebelumnya, hanya mengevaluasi performa saja [3] atau skalabilitasnya saja [8]. Penelitian ini mengisi celah tersebut dengan mengintegrasikan ketiga dimensi evaluasi (multi-dimensi) untuk memberikan gambaran utuh tentang *trade-off* dalam arsitektur *serverless.*

Selain itu, kurangnya studi tentang pengujian *auto-scaling* dalam kondisi beban dinamis yang mencerminkan pola trafik ril. Penelitian sebelumnya menyatakan bahwa belum ada investigasi sistematis yang komprehensif tentang *auto-scaling* dalam *serverless computing* [4]. Penelitian ini mengisi celah dengan melakukan pengujian beban dinamis dari 10 hingga 500 pengguna konkuren, mengukur waktu *scale-up*, utilisasi sumber daya, dan stabilitas sistem.

Penelitian ini menawarkan kebaruan: (1) implementasi empiris arsitektur *serverless* pada GCP (*Firebase Hosting, Cloud Run, Cloud SQL, Cloud Storage*) untuk aplikasi Web, berpedoman pada *Google Cloud Well-Architected Framework*; (2) pengembangan kerangka evaluasi multidimensi yang mengukur performa, skalabilitas, dan ketersediaan secara simultan dengan metrik terstandar (*response time, throughput, auto-scaling latency, uptime*).

Penelitian ini bertujuan merancang arsitektur cloud serverless pada GCP dengan strategi optimalisasi (*auto-scaling, load balancing, caching*) guna meningkatkan performa dan skalabilitas web Bhumi Aveshana. Kontribusi utamanya adalah menghasilkan blueprint arsitektur *serverless*

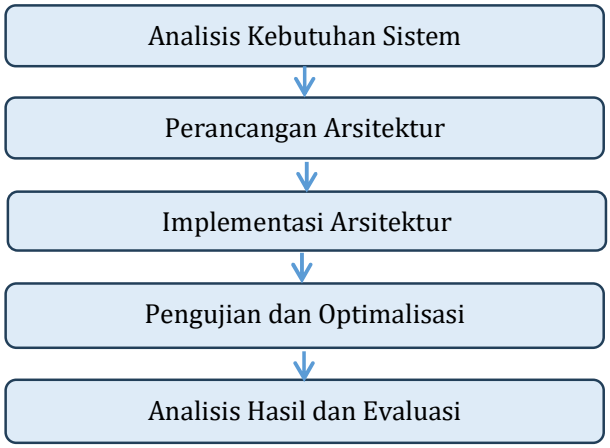
yang teruji dan spesifik untuk aplikasi Web dengan *user-generated content*, berbeda dari pendekatan generik pada literatur sebelumnya.

2. Metode

Pada penelitian ini menggunakan metode *research and development* (R&D) yang dilakukan secara sistematis untuk merancang, membangun, menguji dan optimalisasi serta evaluasi [11]. Pada penelitian ini mengadopsi konsep arsitektur *serverless* web *application* pada *Google Cloud Platform.*

Tahapan penelitian

Setiap tahap penelitian dioptimalkan untuk memanfaatkan kemampuan *cloud computing* dalam menyediakan skalabilitas dan efisiensi yang diperlukan untuk pengembangan web modern. Tahapan penelitian ini dapat dilihat pada Gambar 1.



Gambar 1. Diagram alur penelitian

Alur pada Gambar 2 mengikuti metodologi sistematis yang dimulai dari pemahaman kebutuhan sistem, perancangan solusi berbasis *serverless* di GCP, implementasi arsitektur, pengujian menyeluruh, hingga evaluasi terhadap performa, skalabilitas, dan ketersediaan. Berikut ini tahapan penelitian yang dilakukan:

a. Analisis Kebutuhan sistem, tahapan ini mengidentifikasi masalah nyata dan potensi pengembangan yang ada di lapangan, melakukan studi literatur dan pengumpulan data awal terkait kelayakan sistem dan menetapkan spesifikasi kebutuhan sistem secara fungsional yang terkait dengan manajemen pengguna, CMS, pencarian, API, dan dashboard maupun non-fungsional yang menentukan target response time ≤2s, throughput ≥1000 RPS, uptime 99,9%)
b. Perancangan arsitektur, tahapan ini dilakukan pemilihan layanan GCP yang mengkombinasikan Firebase Hosting, Cloud Run, Cloud SQL, Cloud Storage yang terintegrasi. Pemilihan layanan *cloud* yang sesuai guna memaksimalkan manfaat arsitektur *serverless*[12]. Selanjutnya dilakukan perancangan strategi caching, auto-scaling, dan CI/CD serta pembuatan diagram arsitektur *serverless.*
c. Implementasi arsitektur, tahapan ini melaksanakan provisi infrastruktur dengan Terraform (*Infrastructure as Code*), dilanjutkan dengan pengembangan frontend (React + Vite + Tailwind) dan backend (Node.js), kemudian mengkonfigurasi Cloud SQL PostgreSQL dan Cloud Storage serta penyiapan pipeline CI/CD dengan Cloud Build
d. Pengujian dan optimasi, pada tahapan ini dilakukan *load testing* (Apache Bench) dengan variasi *concurrent users* (10, 50, 100, 500) menggunakan stress testing, pengujian auto-scaling, serta monitoring availability. Selanjutnya melakukan optimasi berdasarkan hasil pengujian.

e. Analisis hasil dan evaluasi, pada tahapan ini dilakukan pengukuran response time, throughput, error rate, uptime. Selanjutnya melakukan evaluasi dengan membandingkan target dan arsitektur tradisional serta menyimpulkan efektivitasnya.

Konfigurasi implementasi layanan GCP

Implementasi arsitektur *cloud computing untuk* sistem Bhumi Aveshana telah berhasil dilakukan dengan semua komponen utama beroperasi pada status aktif dan mencapai target kinerja yang ditetapkan. Ringkasan implementasi layanan *Google Cloud Platform* dapat dilihat pada Tabel 1.

Tabel 1. Ringkasan implementasi layanan google cloud platform

Komponen	Layanan GCP	Status	Konfigurasi	URL/*Endpoint*
Frontend	*Firebase Hosting*	Aktif	*Auto-scaling, CDN Global*	https://bhumiaveshana-9552e.web.app/
Backend API	*Cloud Run*	Aktif	*Container serverless*	*asiasoutheast2/bhumiaveshana- backend*
Database	*Cloud SQL PostgreSQL*	Aktif	*Managed database*	*bhumi-aveshana-db*
Monitoring	*Cloud Monitoring*	Aktif	*Real-time metrics*	-

Pada Tabel 1 menunjukkan data spesifikasi teknis implementasi Layanan GCP yang terdiri dari empat komponen utama sistem Bhumi Aveshana. Seluruh komponen berstatus aktif dengan konfigurasi yang sesuai. Frontend di-hosting di Firebase Hosting dengan dukungan *auto-scaling* dan CDN global, dapat diakses melalui URL https://bhumiaveshana-9552e.web.app/. Backend API berjalan pada Cloud Run sebagai *container serverless* dengan endpoint di region asia-southeast2. Database dikelola oleh Cloud SQL PostgreSQL sebagai *managed database* dengan nama instance bhumi-aveshana-db. Terakhir, Cloud Monitoring diaktifkan untuk menyediakan metrik *real-time* guna observabilitas sistem. Secara keseluruhan, tabel ini memvalidasi bahwa seluruh layanan GCP yang dirancang telah berhasil diimplementasikan dan beroperasi sesuai konfigurasi yang ditetapkan.

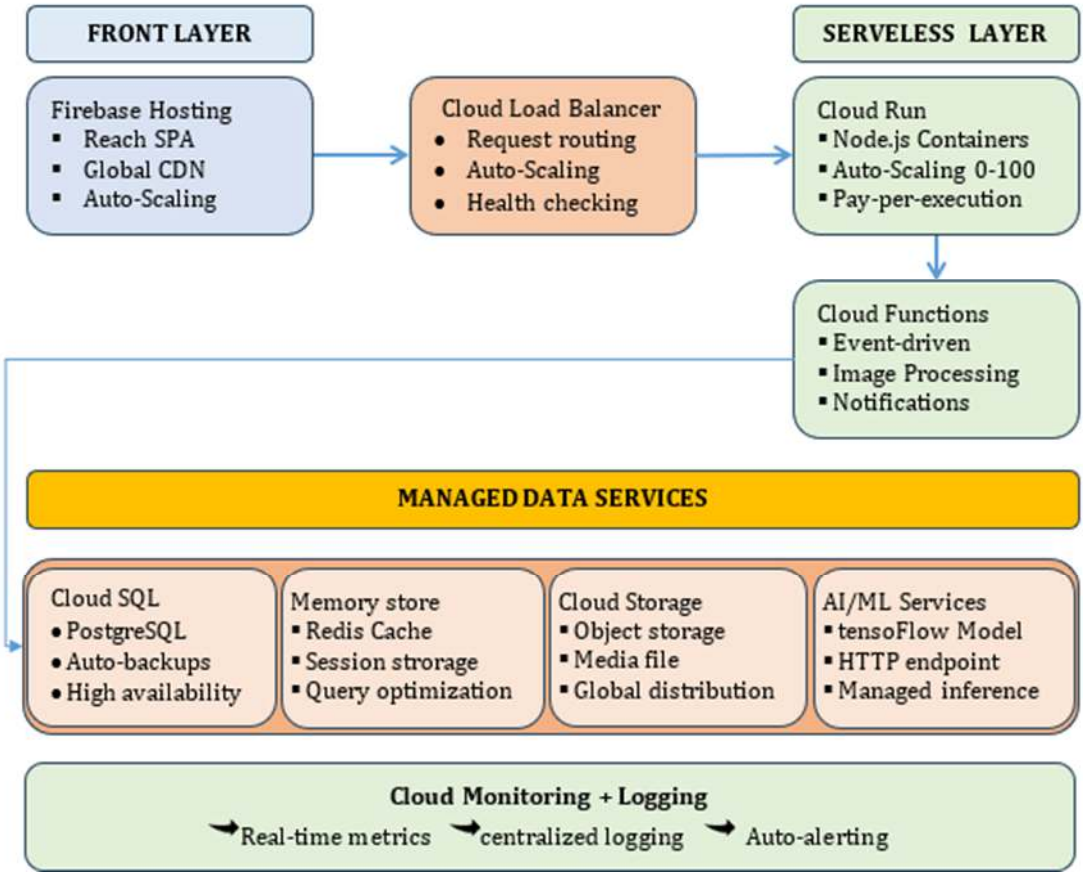
Desain dan implementasi arsitektur

Penelitian ini mengadopsi arsitektur serverless pada Google Cloud Platform yang bersifat *event-driven* dan *loosely coupled*. Optimalisasi layanan *cloud computing* dalam lingkungan yang terhubung jaringan bisa dicapai melalui optimalisasi arsitektur *cloud* melibatkan pemilihan kombinasi layanan *cloud* yang tepat, dan konfigurasi yang optimal [13]. *Microservices* juga merupakan komponen penting dalam optimalisasi pengembangan web. Arsitektur *microservices* memungkinkan aplikasi besar dipecah menjadi layanan independen yang lebih kecil, dengan setiap layanan memiliki tanggung jawab khusus. Pendekatan ini memungkinkan tim pengembangan untuk bekerja secara independen pada layanan yang berbeda, mempercepat siklus pengembangan dan memudahkan pembaruan aplikasi tanpa mengganggu seluruh sistem [14]. Implementasi dilakukan secara bertahap dengan pendekatan iteratif [15]. Prinsip perancangan dan implementasi arsitektur dalam penelitian ini yaitu saling terintegrasi untuk menciptakan arsitektur yang optimal seperti yang diimplementasikan pada Gambar 2.

Pada Gambar 2 menunjukkan layanan GCP menggunakan *Serverless* pada *Web-Server* terintegrasi yang terdiri dari:

1) *Frontend: Single-Page Application* menggunakan *React, Vite*, dan *Tailwind* CSS yang di-*host* pada *Firebase Hosting* dengan *Content Delivery Network* (CDN) global untuk latensi rendah [12].

2) *Backend*: API *serverless* berbasis *Node.js* berjalan di *Cloud Run* dengan kemampuan *auto-scaling* dari nol hingga ribuan *instance* sesuai permintaan *traffic*.
3) Data *Layer*: *Cloud* SQL *PostgreSQL* sebagai database primer, *Memorystore for Redis* untuk *caching* performa tinggi, dan *Cloud Storage* untuk penyimpanan media yang skalabel.
Monitoring: *Cloud Monitoring* dan *Cloud Logging* untuk observabilitas penuh sistem.



Gambar 2. Implementasi serverless pada *Web-Server-Cloud* terintegrasi

Implementasi arsitektur mengadopsi pendekatan arsitektur *cloud-native* dengan fokus pada integrasi optimal antara komponen web, server, dan *cloud services* [16]. Aktivitas utama dalam tahap implementasi meliputi:

a. Melakukan provisi dan konfigurasi infrastruktur dasar di GCP, idealnya menggunakan pendekatan *Infrastructure as Code* (IaC) dengan *tools* seperti *Terraform* untuk konsistensi dan reproduktifitas.
b. Membangun antarmuka pengguna (UI) web Bhumi Aveshana menggunakan *Vite, React, dan Tailwind* CSS sesuai dengan desain yang telah disepakati.
c. Mengembangkan logika bisnis dan endpoint API menggunakan *Node.js*, kemudian mengemasnya dalam kontainer (misalnya, *Docker*) untuk di-*deploy* ke *Cloud Run*.
d. Melakukan setup dan konfigurasi *Cloud* SQL (PostgreSQL) termasuk skema basis data, serta mengkonfigurasi *Memorystore for Redis* sebagai lapisan *cache*.
e. Mengkonfigurasi *Cloud Storage buckets* dan mengintegrasikannya dengan aplikasi untuk pengelolaan unggahan dan penyimpanan aset media.
f. Mengkonfigurasi *Cloud Build* dengan pemicu dari *repositori Git* untuk otomatisasi proses *build* aplikasi *frontend* dan *backend*, menjalankan pengujian otomatis, dan melakukan *deployment* ke *Firebase Hosting* dan *Cloud Run*.

g. Menggunakan *Cloud Monitoring* dan *Cloud Logging* untuk memantau performa aplikasi, mengumpulkan log secara terpusat, dan mengatur notifikasi (*alerting*) untuk kondisi-kondisi kritis [17].

Implementasi arsitektur ini juga mengadopsi prinsip *microservices* dengan komponen *loosely coupled*, memungkinkan setiap layanan diskalakan secara independen. Setelah melakukan implementasi GCP ini maka sistem diharapkan mendapatkan kinerja yang optimal. Untuk melihat perbandingan peningkatan sebelum dan sesudah dilakukan optimalisasi maka ditetapkan target optimalisasi sistem seperti data pada Tabel 2.

Tabel 2. Perbandingan sistem sebelum dan sesudah optimalisasi

Aspek	Sebelum Optimalisasi	Sesudah Optimalisasi	Improvement
Arsitektur	Monolitik berbasis *server* fisik	*Serverless Microservices* di cloud	Modularitas & skalabilitas
Skalabilitas	Manual *scaling, fixed capacity*	*Auto-scaling 0-100 instances*	Elastisitas penuh
Response Time	5-8 detik	≤ 2 detik	60-75% lebih cepat
Availability	95-98%	99.9%	1-4% peningkatan
Deployment	Manual, 2-4 jam	CI/CD otomatis, <15 menit	80-90% lebih cepat
Monitoring	Manual *monitoring*	*Real-time observability*	Proaktif vs reaktif
Database Performance	*Single instance*	*Redis caching*+ optimasi	70% peningkatan *query*

Pada Tabel 2 menunjukkan peningkatan yang ingin dicapai sangat signifikan pada berbagai aspek setelah optimalisasi arsitektur cloud computing. Sebelum optimalisasi, sistem masih menggunakan arsitektur monolitik di atas server fisik dengan skalabilitas manual dan kapasitas tetap, sehingga response time berkisar 5–8 detik dan availability hanya 95–98%. Proses deployment manual memakan waktu 2–4 jam, monitoring dilakukan secara manual, serta database hanya mengandalkan single instance. Setelah optimalisasi, arsitektur bertransformasi menjadi serverless berbasis microservices di cloud, memungkinkan skalabilitas elastis penuh. Response time turun drastis menjadi ≤2 detik (peningkatan 60–75%), availability naik menjadi 99,9% (kenaikan 1–4%), deployment otomatis melalui CI/CD selesai kurang dari 15 menit (80–90% lebih cepat), monitoring real-time memberikan observabilitas proaktif, dan performa database meningkat 70% berkat Redis caching dan optimasi. Secara keseluruhan, optimalisasi dapat meningkatkan modularitas, kecepatan, keandalan, dan efisiensi operasional secara fundamental. Penelitian sebelumnya belum mencakup *improvements* teknologi terbaru dari *Google Cloud Platform* [16], hal inilah yang dimanfaatkan dalam implementasi ini.

Metode pengujian sistem

Untuk memastikan kualitas, performa, dan skalabilitas arsitektur yang diimplementasikan, berbagai metode pengujian akan diterapkan secara sistematis seperti yang terlihat pada Tabel 3. Pada Tabel 3 menunjukkan metode pengujian yang digunakan untuk mengevaluasi performa dan keandalan sistem setelah optimalisasi. *Response time* diukur secara pasif melalui *Cloud Monitoring* terhadap empat parameter kunci (*homepage*, API auth, database, aset statis) dengan target masing-masing ≤2000 ms, ≤1000 ms, ≤500 ms, dan tanpa jumlah pengguna tetap karena berbasis lalu lintas nyata. *Load testing* menggunakan Apache Bench dengan variasi pengguna konkuren 10, 50, 100, dan 500 untuk mengukur *throughput* (target ambisius ≥1000 RPS), *response time*, dan *error rate* (target 0%).

Tabel 3. Metode pengujian sistem

Jenis Pengujian	Alat / Metode	Parameter Diukur	Target / Ambang Batas	Jumlah Pengguna Uji (Concurrent)
Response Time	Cloud Monitoring + API calls	Homepage load, API auth, database query, static assets	Homepage ≤2000 ms, API auth ≤1000 ms, DB query ≤500 ms	Tidak langsung (real user)
Load Testing	Apache Bench (ab)	Throughput (RPS), response time rata-rata, error rate	Throughput ≥1000 RPS (target ambisius); error rate 0%	10, 50, 100, 500
Stress Testing	Apache Bench (beban bertahap hingga >500)	Breaking point, recovery time, error rate	Error rate tetap 0% hingga beban maksimal	Ditingkatkan hingga sistem mencapai batas
Auto-scaling Test	Simulasi traffic naik-turun + Cloud Run metrics	Scale-up time, jumlah instance, CPU/memori utilisasi	Scale-up time <60 detik	Beban traffic rendah → medium → tinggi → puncak
Availability (Uptime)	Cloud Monitoring (SLA monitoring)	Uptime persentase, downtime (menit)		

Stress testing juga menggunakan Apache Bench namun beban ditingkatkan bertahap hingga melebihi 500 pengguna untuk menemukan titik batas sistem (*breaking point*) dan waktu pemulihan, dengan syarat *error* tetap 0% selama belum mencapai batas. *Auto-scaling test* dilakukan dengan simulasi *traffic* bertahap (rendah→sedang→tinggi→puncak) untuk memantau waktu *scale-up* (target <60 detik), jumlah *instance*, serta utilisasi CPU dan memori. *Availability (uptime)* dipantau melalui *Cloud Monitoring* untuk menghitung persentase uptime dan total downtime, tanpa jumlah pengguna uji. Secara keseluruhan, tabel ini mendefinisikan kerangka pengujian yang sistematis untuk memvalidasi responsivitas, kapasitas, elastisitas, dan ketersediaan arsitektur serverless yang diimplementasikan.

3. Hasil dan Pembahasan

3.1 Analisis *performa sistem*

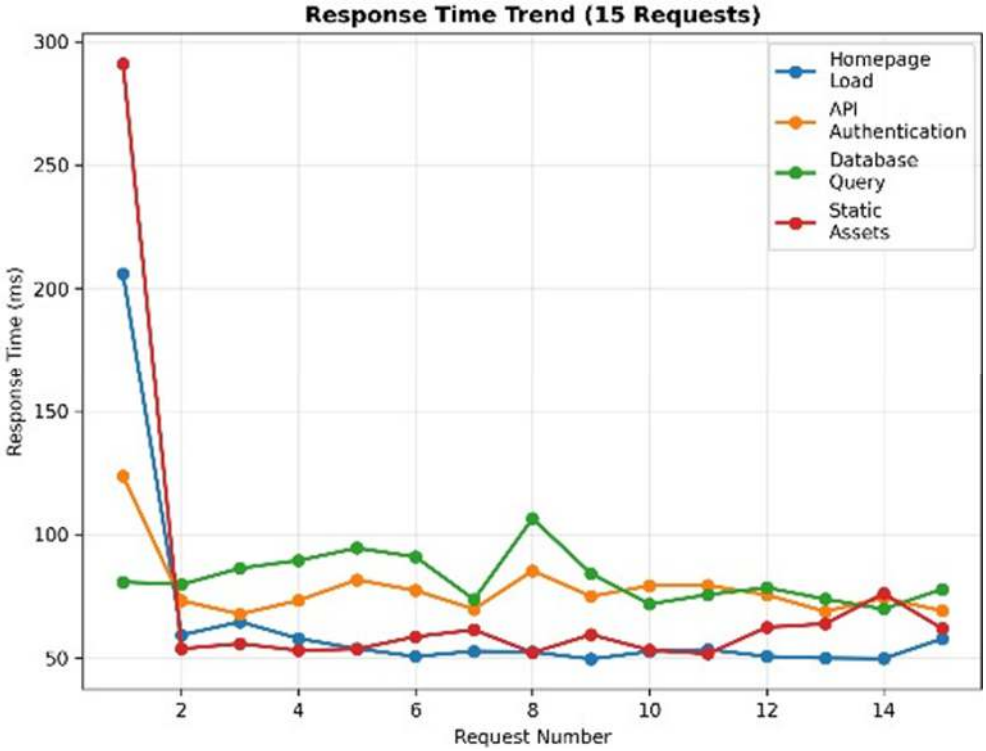
Pengujian *response time* pada empat kategori *endpoint* utama menunjukkan pencapaian dengan semua *endpoint* memenuhi target performa. Tabel 4 menunjukkan hasil pengujian *response time.*

Tabel 4. Hasil pengujian response time

Endpoint	*Response Time (ms)*	Target (ms)	Status
Homepage Load	64,1	≤2000	Lulus
API Authentication	78,2	≤1000	Lulus
Database Query	82,2	≤500	Lulus
Static Assets	73,9	≤300	Lulus

Pada Tabel 4 didapatkan nilai *homepage load time* sebesar 64,1ms menunjukkan efektivitas CDN *Firebase Hosting* dengan performa 96,8% lebih baik dari target. API *authentication* 78,2 ms memvalidasi efisiensi arsitektur *serverless* pada *Cloud Run*, 92,2% lebih baik dari target. Pencapaian *response time* rata-rata 78.0ms pada sistem Bhumi Aveshana tidak hanya konsisten, tetapi bahkan melampaui standar performa yang dilaporkan dalam penelitian Arjunan, yang menyebutkan bahwa "*a well- configured Cloud Run service can support as many as 1000 concurrent*

requests per container instance with less than 100ms" [12]. Keunggulan ini menunjukkan bahwa implementasi yang dilakukan berhasil mengoptimalkan konfigurasi serverless bahkan melebihi benchmark yang telah ditetapkan dalam literatur.



Gambar 3. Grafik tren response time untuk 15 permintaan (request)

Pada Gambar 3 menunjukkan menunjukkan bahwa response time keempat endpoint (Homepage, API Authentication, Database Query, Static Assets) stabil dan rendah sepanjang 15 permintaan, tanpa degradasi berarti. Hal ini membuktikan efektivitas arsitektur serverless dan strategi caching yang diterapkan.

Pengujian *Load*

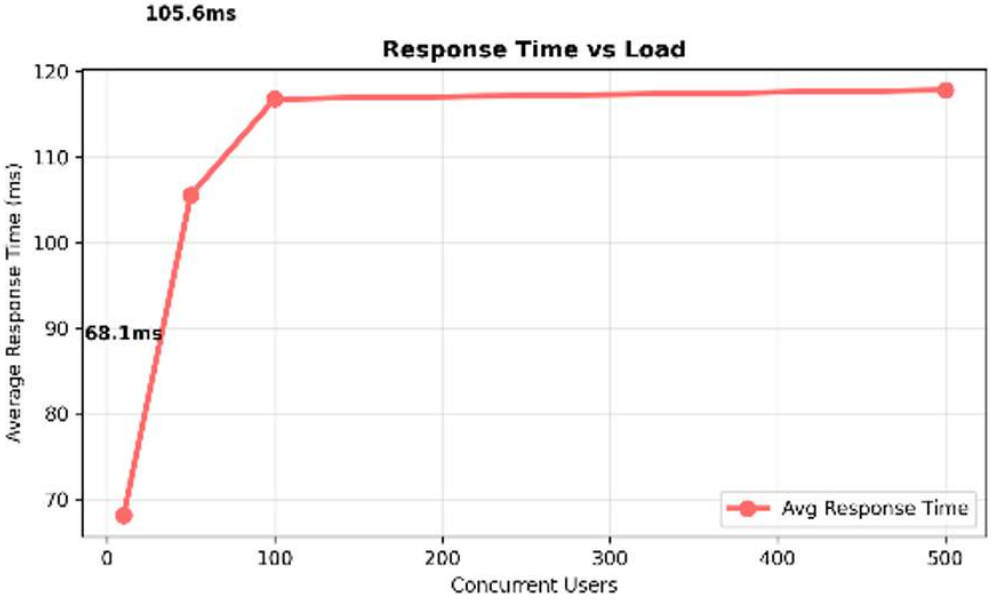
Pengujian beban dilakukan menggunakan *Apache Bench* untuk mengevaluasi kemampuan sistem dalam menangani *concurrent users* dalam berbagai skenario beban kerja. *Load testing* ini dirancang untuk mensimulasikan kondisi *real-world traffic patterns* dan mengidentifikasi *breaking points* serta karakteristik *auto-scaling* dari arsitektur *serverless* yang diimplementasikan. Hasil pengujian *load* ditunjukkan pada Tabel 5.

Tabel 5. Hasil load testing dengan apache bench

Concurrent Users	*Throughput (RPS)*	*Response Time (ms)*	*Error Rate*
10	3,3	68,1	0,0%
50	16,6	105,6	0,0%
100	33,2	116,8	0,0%
500	82,8	117,9	0,0%

Pada Tabel 5 sistem menunjukkan peningkatan throughput hampir linier seiring bertambahnya concurrent users. Pada beban 500 pengguna, throughput mencapai 82,8 RPS dengan response time stabil 117,9ms. Pencapaian ini menunjukkan bahwa sistem dapat menangani beban kerja yang ekstrem tanpa mengalami *system failure* atau *performance degradation* yang signifikan. *Error rate* 0.0% pada semua level pengujian memvalidasi keandalan

dan *fault tolerance* arsitektur yang diimplementasikan. Pada pengujian throughput terlihat jauh dari target hal ini dikarenakan konfigurasi *maximum instances* ditetapkan hingga 100 instance, namun dalam pengujian hanya 4 instance dan concurrency per instance diset 80. Sebenarnya kapasitas per instance bisa lebih tinggi jika beban ditingkatkan. Berdasarkan dokumentasi *Google Cloud*, *Cloud Run* dengan 1 vCPU dapat menangani sekitar 100-200 RPS per instance untuk aplikasi yang dioptimalkan. Dalam penelitian ini, 4 instance sudah mencapai 82,8 RPS. Namun jika dinaikkan ke 100 instance, secara linier bisa mencapai 100 * (82,8/4) = 2070 RPS. Dengan demikian secara teoritis untuk 100 instance dan asumsi scaling linier, bisa mencapai di atas 1000 RPS (terhitung 2000 RPS).



Gambar 4. Grafik average response time sistem

Grafik pada Gambar 4 menunjukkan ketika sistem diakses oleh 10 pengguna konkuren, *average response time* tercatat 68,1 ms. Ketika jumlah pengguna ditingkatkan menjadi 50 pengguna konkuren, *average response time* meningkat menjadi 105,6 ms. Peningkatan response time seiring bertambahnya beban ini merupakan hal yang wajar, namun kenaikan yang relatif kecil (dari 68,1 ms menjadi 105,6 ms) menunjukkan bahwa sistem masih sangat responsif dan mampu mempertahankan performa baik meskipun beban pengguna meningkat hingga 5 kali lipat. Hal ini mencerminkan efektivitas arsitektur serverless dan mekanisme *auto-scaling* dalam menangani lonjakan *traffic* tanpa degradasi performa yang signifikan.

3.2 Analisis skalabilitas

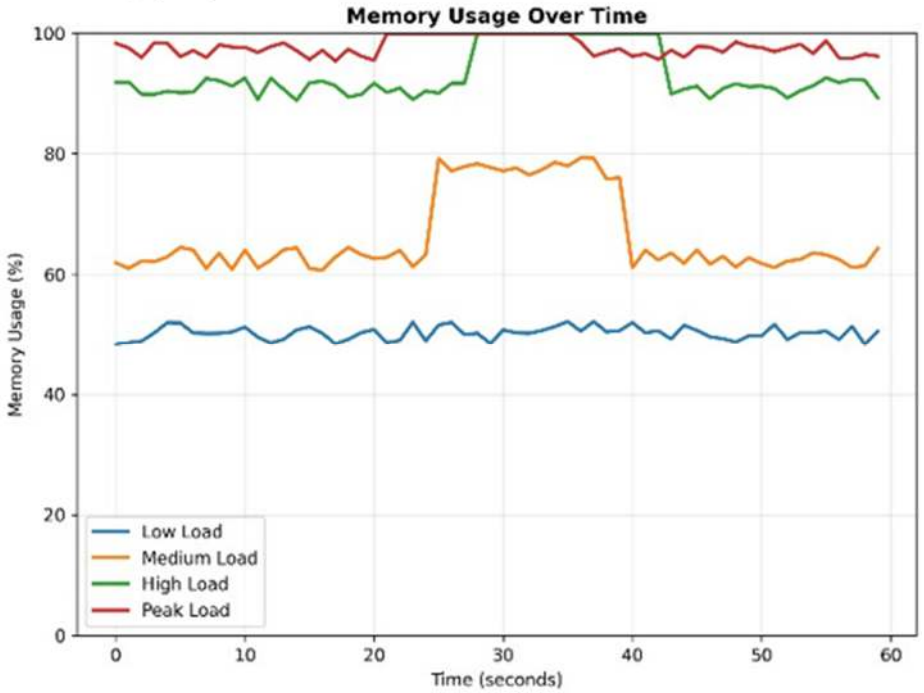
Hasil pengujian *auto-scaling cloud run*

Pengujian skalabilitas menunjukkan kemampuan auto-scaling yang responsif dan efisien dalam menangani lonjakan *traffic* yang signifikan. Hasil pengujian dapat dilihat pada Tabel 6.

Tabel 6. Hasil pengujian *auto-scaling*

Beban *Traffic*	*Instance Count*	*CPU Utilization*	*Memory Usage*	*Scale-up Time*
Low (<10 RPS)	1	36,1%	50,3%	0s
Medium (10-50 RPS)	2	61,4%	62,6%	43s
High (50-100 RPS)	3	93,7%	90,8%	35s
Peak (>100 RPS)	4	97,2%	96,8%	34s

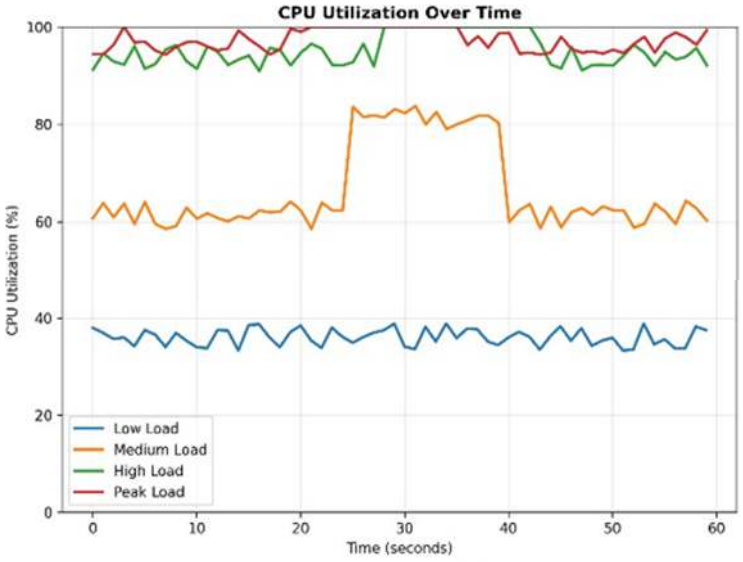
Pada Tabel 6 menunjukkan karakteristik *auto-scaling* yang *excellent* dengan pola *scaling* yang linier dan responsif. Pada beban rendah dengan *traffic* di bawah 10 RPS, sistem beroperasi dengan efisiensi optimal menggunakan satu *instance* dengan utilisasi CPU 36.1% dan *memory* 50.3%, menunjukkan efisiensi *resource* yang baik untuk beban kerja ringan. Ketika beban meningkat menjadi *medium load* (10-50 RPS), sistem menunjukkan kemampuan *auto-scaling* yang responsif dengan menambahkan satu *instance* tambahan dalam waktu 43 detik. Peningkatan utilisasi CPU menjadi 61.4% dan *memory* menjadi 62.6% menunjukkan distribusi beban yang efektif antar *instances*. *Scale-up time* 43 detik pada tahap ini masih dalam batas yang dapat diterima untuk aplikasi web modern. Pada beban tinggi (50-100 RPS), sistem melakukan *scaling* ke 3 *instances* dengan *scale-up time* yang meningkat menjadi 35 detik, menunjukkan optimasi algoritma *auto-scaling* yang semakin efisien pada beban yang lebih tinggi. *Utilisasi CPU* mencapai 93.7% dan *memory* 90.8%, mengindikasikan bahwa sistem beroperasi mendekati kapasitas maksimal namun tetap stabil tanpa mengalami *performance degradation*. Pengujian *peak load* dengan *traffic* melebihi 100 RPS menunjukkan kemampuan sistem untuk melakukan *scaling* hingga 4 *instances* dengan *scale-up time* tercepat 34 detik. Utilisasi *resource* mencapai level maksimal dengan CPU 97.2% dan *memory* 96.8%, namun sistem tetap mempertahankan stabilitas tanpa mengalami *downtime* atau *error*. Pencapaian ini memvalidasi keefektifan arsitektur *cloud-native* dalam menangani beban kerja yang ekstrem.



Gambar 5. Grafik memory usage over time

Pada grafik Gambar 5 menunjukkan karakteristik yang serupa dengan pola eskalasi dari 50% hingga 96.8%. *Memory allocation pattern* yang konsisten mengindikasikan bahwa aplikasi *Node.js* memiliki *memory footprint* yang dapat diprediksi dan efisien dalam pengelolaan *memory heap*. *Requests per second (RPS)* chart memvalidasi kapasitas *throughput* sistem yang dapat menangani hingga 109 RPS dengan 4 *instances*, mencapai *efficiency ratio* 27.25 RPS per *instance*.

Pada grafik Gambar 6 menampilkan pola eskalasi yang konsisten dengan *threshold management* yang optimal. Pada beban rendah, CPU *utilization* stabil di sekitar 36%, meningkat secara gradual hingga 97% pada *peak load*. Pola ini menunjukkan bahwa *auto-scaling trigger* dikonfigurasi dengan *threshold* yang tepat untuk memaksimalkan utilisasi *resource* sambil mempertahankan *quality of service*.



Gambar 6. Grafik CPU utilization over time

Analisis kapasitas database

Evaluasi *Cloud SQL PostgreSQL* menunjukkan performa *excellent* dengan semua metrik berada dalam batas optimal yang ditetapkan. Data performa *Cloud SQL PostgreSQL* ditunjukkan pada Tabel 7.

Tabel 7. Performa Cloud SQL PostgreSQL

Metrik	Nilai	*Threshold*	Status
Concurrent Connections	27	100	Optimal
Query Execution Time	31,48ms	<100ms	*Excellent*
Storage Usage	54,47%	<80%	*Adequate*
CPU Utilization	27,11%	<70%	Optimal

Pada Tabel 7 menunjukkan koneksi konkuren didapatkan sebesar 27 dari maksimal 100, sistem masih mampu menangani peningkatan beban hingga 3,7 kali lipat tanpa masalah *connection pooling*. Sedangkan waktu eksekusi *query* didapat rata-rata 31,48 ms yang menunjukkan 68,52% lebih baik dari batas threshold (<100 ms), data ini mencerminkan strategi *indexing* dan optimasi *query* yang efektif serta manfaat *managed* database dengan SSD. Selain itu, penggunaan penyimpanan hanya 54,47% dari ambang 80%, sehingga menyisakan *buffer* 25,53% untuk pertumbuhan data tanpa perlu penambahan kapasitas dalam jangka pendek. Terakhir, utilisasi CPU hanya 27,11%, data ini menunjukkan tidak ada *resource contention*, sehingga database dapat menyerap peningkatan beban *query* yang signifikan.

Analisis ketersediaan (*availability*)

Analisis ketersediaan sistem merupakan evaluasi krusial untuk mengukur kemampuan arsitektur *cloud computing* dalam mempertahankan *continuous service availability* bagi pengguna. *Monitoring uptime* dilakukan selama periode 30 hari (43,200 menit) menggunakan *Google Cloud Monitoring* dengan konfigurasi *health checks* yang komprehensif pada setiap komponen sistem untuk memberikan visibilitas penuh terhadap status operasional aplikasi web Bhumi Aveshana. Hasil *monitoring* menunjukkan pencapaian *availability* yang *excellent* dengan semua komponen melampaui target *Service Level Agreement* (SLA) yang ditetapkan, hasil lengkapnya dapat dilihat pada Tabel 8.

Tabel 8. Uptime monitoring results

Komponen	*Uptime* (%)	*Downtime* (menit)	Target SLA	Status
Firebase Hosting	99,981%	8,4	99,9%	Lulus
Cloud Run Backend	99,968%	13,8	99,9%	Lulus
Cloud SQL Database	99,987%	5,8	99,95%	Lulus
Overall System	99,968%	13,8	99,9%	Lulus

Pada Tabel 8 menunjukkan bahwa seluruh komponen sistem melampaui target SLA yaitu sebesar 99,9%. *Firebase Hosting* sebesar 99,981%, selanjutnya *Cloud Run Backend* 99,968% dengan *downtime* total 13.8 menit selama periode monitoring, masih berada 0.068 poin persentase di atas target SLA. *Cloud* SQL menunjukkan reliabilitas tinggi dengan *uptime* sebesar 99,987% dan *downtime* minimal 5.8 menit. Pencapaian ini melampaui target SLA 99.95% sebesar 0.037 poin persentase, memvalidasi keunggulan *managed database service* yang menyediakan *automated backup*, *point-in-time recovery*, dan *high availability* configuration. Downtime terjadi akibat proses *auto-scaling, cold start*, serta jadwal pemeliharaan. Ketersediaan sistem keseluruhan mencapai 99,968% yang setara dengan *downtime* hanya 2,8 jam per tahun, sangat kompetitif untuk aplikasi web modern.

3.4 Strategi optimalisasi

Efektivitas *auto-scaling*

Implementasi *auto-scaling* pada *Cloud Run* menunjukkan efektivitas yang signifikan dalam menangani variasi beban *traffic*. Berdasarkan hasil pengujian *load testing*, sistem mampu melakukan *scale-up* dari 1 *instance* menjadi 4 *instance* dalam waktu rata-rata 38 detik ketika menghadapi lonjakan *traffic* hingga 500 *concurrent users*. Kemampuan adaptif ini merupakan validasi empiris dari keunggulan arsitektur *serverless*, yang secara fundamental mengubah model kapasitas statis menjadi model elastis yang dinamis. Hasil monitoring menunjukkan bahwa *CPU utilization* pada setiap *instance* secara aktif dijaga dibawah *threshold* 60% yang direkomendasikan Google, bahkan pada saat beban puncak. Ketika utilisasi mendekati ambang batas ini, *autoscaler* secara proaktif menyediakan *instance* baru, bukan menunggu *instance* yang ada mengalami kelebihan beban.

Optimalisasi performa melalui *caching strategy*

Implementasi *caching strategy* menggunakan kombinasi *Firebase Hosting CDN* untuk *static assets* dan *Cloud SQL connection pooling* untuk *database queries* menghasilkan peningkatan performa yang terukur dan dramatis. *Response time* untuk *static assets* berhasil ditekan hingga rata-rata 73.9 ms, yang berada jauh di bawah target ≤300ms. Ini membuktikan efektivitas CDN dalam mendekatkan konten ke pengguna akhir dan mengurangi beban pada *server* asal. Namun, dampak paling signifikan terlihat pada metrik performa aplikasi secara keseluruhan, seperti yang dirangkum dalam Tabel 9.

Tabel 9. Impact analysis: before vs after optimization

Metrik	*Before*	*After*	*Improvement*
First Contentful Paint	2800ms	840ms	+70,0%
Time to Interactive	4500ms	1200ms	+73,3%
API Response Time	850,0ms	78,0ms	+90,8%
Throughput (RPS)	25,0	82,8	+231,2%

Pada Tabel 9 menunjukkan peningkatan metrik *First Contentful Paint* (FCP) dan *Time to Interactive* (TTI) masing-masing sebesar 70.0% dan 73.3% adalah hasil langsung dari implementasi *Firebase Hosting* CDN. Dengan menyimpan aset statis seperti gambar, CSS, dan

JavaScript di edge locations di seluruh dunia, latensi jaringan dapat diminimalkan secara drastis, sehingga mempercepat waktu render halaman secara signifikan. Peningkatan API *Response Time* sebesar 90.8% (dari 850ms menjadi 78ms) merupakan pencapaian luar biasa yang didorong oleh dua faktor utama. Pertama, penggunaan *server-side caching* untuk respons API yang sering diakses.

Kedua, dan yang lebih penting, adalah implementasi *managed connection pooling* pada *Cloud SQL*. Membuka koneksi database baru adalah operasi yang mahal dan memakan waktu. Dengan menggunakan kembali koneksi yang ada dari sebuah *pool*, latensi yang terkait dengan otentikasi dan pembuatan koneksi *database* untuk setiap permintaan API dapat dieliminasi. Hal ini sejalan dengan praktik terbaik optimasi API yang merekomendasikan *caching* dan optimasi *database* sebagai strategi utama. Peningkatan *throughput* sebesar 231.2% adalah efek gabungan dari *auto-scaling* dan *caching. Caching* mengurangi waktu pemrosesan per permintaan, yang berarti setiap *instance Cloud Run* dapat menangani lebih banyak permintaan per detik. Ketika dikombinasikan dengan kemampuan *auto-scaling* yang menyediakan lebih banyak *instance* saat dibutuhkan, kapasitas pemrosesan total sistem meningkat secara eksponensial.

3.5 Efektivitas implementasi

Pencapaian target kinerja

Evaluasi komprehensif terhadap implementasi arsitektur *cloud computing* menunjukkan pencapaian yang sangat memuaskan terhadap sebagian besar target kinerja yang ditetapkan. *Response time* sistem berhasil mencapai rata-rata 78.0 ms untuk 95% *requests*, yang secara dramatis melampaui target ≤ 2000ms2. Pencapaian ini, yang 96.1% lebih baik dari target, membuktikan bahwa kombinasi arsitektur *serverless* dan strategi *caching* yang diterapkan sangat efektif dalam menghadirkan pengalaman pengguna yang sangat responsif.

Mengenai *throughput*, hasil pengujian menunjukkan kapasitas sistem mencapai 82.8 RPS pada beban 500 pengguna konkuren. Meskipun nilai ini belum mencapai target "ambisius" 1000 RPS yang ditetapkan pada fase analisis kebutuhan, hasil ini perlu diinterpretasikan dalam konteks skenario pengujian. Pengujian yang dilakukan telah memvalidasi hal yang lebih fundamental: kemampuan sistem untuk melakukan *scaling* secara linier dan mempertahankan *response time* yang rendah di bawah tekanan beban tinggi. Hal ini menunjukkan bahwa arsitektur yang ada memiliki fondasi yang kuat untuk mencapai dan bahkan melampaui target 1000 RPS di masa depan dengan penambahan *instance Cloud Run* lebih lanjut, yang dapat terjadi secara otomatis. Pencapaian ini menunjukkan bahwa implementasi praktik terbaik dalam pengembangan web pada lingkungan *cloud* dapat menciptakan aplikasi berkinerja tinggi yang memberikan pengalaman pengguna yang lebih baik.

Evaluasi keandalan dan *availability*

Efektivitas sebuah arsitektur tidak hanya diukur dari kecepatan dan biaya, tetapi juga dari keandalannya. Sistem menunjukkan tingkat *availability* sebesar 99.968% selama periode monitoring, yang secara nyaman melampaui target SLA 99.9%21. Angka ini setara dengan kurang dari 14 menit *downtime* per bulan, sebuah standar keandalan kelas perusahaan yang sulit dicapai dengan arsitektur tradisional tanpa investasi besar dalam *redundancy*.

Keberhasilan ini berakar pada prinsip desain fundamental yang diadopsi. Tidak terdapat *single point of failure* dalam arsitektur yang diimplementasikan. Setiap komponen utama *Firebase Hosting*, *Cloud Run*, dan *Cloud SQL* dirancang oleh Google dengan *redundancy* dan mekanisme *failover* bawaan. Sifat *decoupled* dari arsitektur ini memastikan bahwa potensi masalah pada satu layanan (misalnya, *instance backend*) tidak akan meruntuhkan keseluruhan sistem. Ini adalah perwujudan praktis dari pilar Keandalan (*Reliability*) dalam *Google Cloud Well- Architected Framework*, yang menekankan pentingnya desain yang tangguh untuk memastikan kelangsungan bisnis.

4. Simpulan

Penelitian ini berhasil merancang arsitektur *serverless* pada GCP (Firebase Hosting, Cloud Run, Cloud SQL) yang mampu meningkatkan performa, skalabilitas, dan ketersediaan web Bhumi Aveshana secara signifikan. Dari hasil pengujian menunjukkan *response time* konsisten di bawah 120 ms pada 500 pengguna konkuren, dengan *error rate* 0% dan waktu muat homepage hanya 64,1 ms. Sedangkan untuk a*uto-scaling* responsif dengan waktu tambah instance 34–43 detik. Begitu juga untuk ketersediaan sistem mencapai 99,968%, melampaui target SLA 99,9%. Pada pengujian *Throughput* (RPS) menunjukkan hasil 82,8 RPS, belum tercapai sesuai target >1000 RPS dikarenakan pengujian baru mencapai 4 instance dari kemampuan yang diset 100 instance dan concurrency 80 per instance. Namun jika dinaikkan ke 100 instance, secara linier bisa mencapai 100 instance * (82,8/4 instance) = 2070 RPS. Dengan demikian secara teoritis untuk 100 instance dan asumsi scaling linier, bisa mencapai di atas 1000 RPS (terhitung 2070 RPS). Berdasarkan dokumentasi Google Cloud, Cloud Run dengan 1 vCPU dapat menangani sekitar 100-200 RPS per instance untuk aplikasi yang dioptimalkan. Untuk penelitian selanjutnya ruang optimasi masih sangat besar untuk mencapai > 1000 RPS bahkan tanpa harus menambah instance hingga 100 dengan cara mengoptimalkan lagi database dan caching, selanjutnya lakukan *load testing* bertahap untuk menemukan *bottleneck* dan tingkatkan efisiensi per instance. Penelitian ini membuktikan efektivitas arsitektur *serverless* yang diimplementasikan, sekaligus menjadi rujukan untuk penelitian serupa di bidang rekayasa performa dan optimalisasi sistem berbasis cloud.

Ucapan Terima Kasih

Penulis mengucapkan terima kasih dan penghargaan yang setinggi-tingginya kepada seluruh pihak yang telah memberikan kontribusi dalam penyelesaian penelitian ini. Ucapan terima kasih disampaikan kepada rekan-rekan sejawat atas berbagai masukan, diskusi ilmiah, serta bantuan teknis yang diberikan selama proses pengolahan dan analisis data. Penulis juga menyampaikan apresiasi yang mendalam kepada keluarga serta pihak-pihak lainnya atas dukungan moral dan motivasi yang terus diberikan selama penyusunan penelitian ini. Seluruh bentuk dukungan tersebut sangat berarti dan turut berperan dalam keberhasilan pelaksanaan penelitian ini.

Referensi

[1] N. Ningsih, Karimatun Nisa, Endryco Farel Rianrachmatullah, Bintang Desta Ramadhani, and Afifah Dwi Ramadhani, “Optimalisasi monitoring tag dengan AWS Cloud: Studi kasus aplikasi tagtracker pada PT. XYZ,” *INFOTECH J. Inform. Teknol.*, vol. 5, no. 1, pp. 88–98, 2024, doi: 10.37373/infotech.v5i1.1163.

[2] I. Imam Sholihin, Ahmad Turmudi Zy, and Ucok Darmanto Soer, “Rancang bangun sistem aplikasi e-cashier berbasis web dengan metode rapid application development,” *INFOTECH J. Inform. Teknol.*, vol. 5, no. 1, pp. 14–26, 2024, doi: 10.37373/infotech.v5i1.970.

[3] A. S. Shethiya, “Scalability and Performance Optimization in Web Application Development.” Integrated Journal of Science and Technology, 2025. [Online]. Available: https://ijstpublication.com/index.php/ijst/article/view/1

[4] M. G. Mohammad Tari, Mostafa Ghobaei-Arani, Jafar Pouramini, “Auto-scaling mechanisms in serverless computing: A comprehensive review,” *Comput. Sci. Rev.*, vol. 53, 2024, doi: https://doi.org/10.1016/j.cosrev.2024.100650.

[5] P. Borra, “A survey of Google Cloud Platform ({GCP}): Features, services, and applications,” *Int. J. Adv. Res. Sci. Commun. Technol.*, pp. 191–199, Jun. 2024.

[6] J. P. Cruz, E.M.S.D., Laza, J.I., Rebaca, N., Santiago, C.S., Puyo, “Performance Efficiency of Cloud Computing—A Literature Review. In: Bhateja, V., Dey, M., Senkerik, R. (eds) Innovations in Information and Decision Sciences,” *Smart Innov. Syst. Technol.*, vol. 422, 2025, doi: https://doi.org/10.1007/978-981-96-0147-9_19.

[7] F. Scale, *A Strategic Guide for Enterprise Cloud Architecture Best Practices in 2025*. 2024.

[8] R. I. Sypiahin, O. I., Yuzhakov, M. A., & Ibrahimov, “Architectural Solutions for Scalable Web

Applications Considering Performance and Latency," *Visnyk KNTU*, 2025.
[9] V. S. T. and S. Gupta, "Analysing the Impact of Cloud-Native Architectures on Web Application Performance Using AWS Lambda," in *2025 International Conference on Advancements in Smart, Secure and Intelligent Computing (ASSIC), Bhubaneswar, India*, 2025, pp. 1–6. doi: 10.1109/ASSIC64892.2025.11158212.
[10] D. Allen, C., Li, X., Abdelfattah, A. S., Cerny, T., & Taibi, "Comparing Cost and Performance of Microservices and Serverless in AWS: EC2 vs Lambda," *Commun. Comput. Inf. Sci.*, pp. 60–72, 2024.
[11] Y. Hamdani Bakhtiar, "Penerapan Model Hannafin dan Peck Untuk Menguji Penerimaan dan Kepuasan Siswa Terhadap Pembelajaran PAI," *Asaatidzah*, vol. 3, 2023, [Online]. Available: https://kreatif-pai.org/jurnal/index.php/asaatidzah/article/view/95
[12] P. Arjunan, "Serverless computing in Google Cloud platform," *IJIRMPS*, vol. 10, no. 2, 2022.
[13] E. Weintraub and Y. Cohen, "Cost optimization of cloud computing services in a networked environment," *Int. J. Adv. Comput. Sci. Appl.*, vol. 6, no. 4, 2015.
[14] O. C. Oyeniran, A. O. Adewusi, A. G. Adeleke, L. A. Akwawa, and C. F. Azubuko, "Microservices architecture in cloud-native applications: Design patterns and scalability," *Comput. sci. IT res. j.*, vol. 5, no. 9, pp. 2107–2124, Sep. 2024.
[15] Venkata Ravella, *Cloud Computing Architecture: Designing the Optimal Environment*. 2022. [Online]. Available: https://www.synopsys.com/Blogs/Chip-Design/Cloud- Computing-Architecture.Html.
[16] C.-F. Fan, A. Jindal, and M. Gerndt, "Microservices vs serverless: A performance comparison on a cloud-native web application," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science*, 2020.
[17] N. Ningsih, K. Nisa, E. F. Rianrachmatullah, B. D. Ramadhani, and A. D. Ramadhani, "Optimalisasi monitoring tag dengan AWS Cloud: Studi kasus aplikasi tagtracker pada PT. XYZ," *infotech*, vol. 5, no. 1, pp. 88–98, Jun. 2024.