Department of Digital Business
Journal of Artificial Intelligence and Digital Business (RIGGS)
Homepage: https://journal.ilmudata.co.id/index.php/RIGGS
Vol. 4 No. 4 (2026) pp: 12831-12840
P-ISSN: 2963-9298, e-ISSN: 2963-914X

Implementasi Kubernetes Cluster untuk High Availability dan Load Balancing SIAK Undiksha

Ketut Saka Pradipta[1], Gede Arna Jude Saskara[2], Bagus Gede Krishna Yudistira[3]
[13]Pendidikan Teknik Informatika, Fakultas Teknik dan Kejuruan, Universitas Pendidikan Ganesha
[2]Sistem Informasi, Fakultas Teknik dan Kejuruan, Universitas Pendidikan Ganesha
[1]saka.pradipta@undiksha.ac.id, [2]jude.saskara@undiksha.ac.id , [3]krishna.yudistira@undiksha.ac.id

Abstrak

Sistem Informasi Akademik (SIAK) merupakan layanan kritis pada lingkungan perguruan tinggi yang dituntut memiliki tingkat ketersediaan dan keandalan tinggi, khususnya pada periode lonjakan akses pengguna seperti pengisian KRS dan penginputan nilai. Pada Universitas Pendidikan Ganesha (Undiksha), arsitektur SIAK yang masih berbasis single server menimbulkan permasalahan berupa penurunan performa dan potensi downtime akibat keterbatasan mekanisme replikasi serta distribusi beban. Penelitian ini bertujuan mengevaluasi penerapan arsitektur Kubernetes Cluster untuk meningkatkan ketahanan layanan (service resilience), high availability, dan efektivitas load balancing pada SIAK Undiksha. Metode penelitian yang digunakan bersifat eksperimental teknis dengan melakukan simulasi kegagalan node dan variasi beban permintaan menggunakan Apache JMeter. Parameter yang dianalisis meliputi Mean Time to Repair (MTTR), tingkat availability, throughput, latency, dan error rate. Hasil pengujian menunjukkan bahwa Kubernetes Cluster mampu mempertahankan ketersediaan layanan di atas 99,9% pada skenario kegagalan satu node dengan waktu pemulihan rata-rata 60–70 detik. Pada pengujian performa, penerapan replikasi pod dan mekanisme load balancing internal Kubernetes mampu meningkatkan throughput secara signifikan serta menekan error rate hingga 0% pada beban menengah hingga tinggi. Temuan ini menunjukkan bahwa Kubernetes Cluster efektif dalam mengurangi risiko single point of failure serta meningkatkan resiliensi dan skalabilitas layanan akademik, meskipun performa sistem tetap dipengaruhi oleh kapasitas sumber daya dan konfigurasi cluster yang digunakan.

***Kata Kunci**: Kubernetes Cluster, High Availability, Load Balancing, Sistem Informasi Akademik.*

1. Latar Belakang

Seiring perkembangan zaman, manusia menciptakan berbagai jenis teknologi dan menggunakannya sebagai solusi dari persoalan mereka [1]. Salah satu inovasi penting yang muncul dalam perkembangan teknologi saat ini adalah penerapan Sistem Informasi Akademik (SIAK). Sistem Informasi Akademik (SIAK) merupakan layanan kritis yang menopang seluruh aktivitas akademik di perguruan tinggi [2]. Menurut [3], Sistem Informasi Akademik merupakan layanan inti pada institusi pendidikan yang berfungsi mendukung pengelolaan data akademik secara terintegrasi, meliputi pendaftaran mahasiswa, pengelolaan nilai, penjadwalan perkuliahan, serta manajemen data dosen dan mahasiswa, sesuai dengan kebutuhan masing-masing institusi. Pada Universitas Pendidikan Ganesha, SIAK melayani lebih dari 24.000 mahasiswa dan ratusan dosen, sehingga memiliki karakteristik beban akses tinggi dan fluktuatif, terutama pada periode KRS dan penginputan nilai. Kondisi ini menuntut sistem dengan tingkat ketersediaan tinggi dan kemampuan distribusi beban yang baik. Arsitektur SIAK Undiksha saat ini masih bergantung pada pendekatan single server dengan vertical scaling, sehingga sistem rentan terhadap single point of failure. Ketika terjadi lonjakan akses atau gangguan pada server utama, layanan mengalami penurunan performa hingga downtime. Upaya mitigasi yang telah dilakukan, seperti peningkatan spesifikasi server dan proteksi trafik, belum mampu mengatasi permasalahan ketersediaan layanan karena tidak didukung mekanisme replikasi dan failover otomatis.

Permasalahan tersebut menunjukkan bahwa arsitektur *vertical scaling* yang digunakan saat ini tidak cukup untuk mendukung sistem akademik dengan tingkat aktivitas tinggi. Untuk meningkatkan ketersediaan layanan, diperlukan pendekatan *horizontal scaling* [4], yaitu penambahan beberapa node server yang bekerja secara paralel. Dengan model ini, beban kerja dapat didistribusikan ke beberapa node sehingga apabila salah satu node mengalami gangguan, node lainnya tetap dapat melanjutkan layanan tanpa menyebabkan *downtime* (*high availability*). Penerapan *horizontal scaling* memerlukan sistem *clustering* yang mampu melakukan replikasi layanan, pembagian beban (*load balancing*), serta pemulihan otomatis (*failover*) antar node.

High Availability atau ketersediaan tinggi merupakan sistem yang dapat beroperasi terus menerus tanpa kegagalan dengan tingkat kinerja dan kualitas operasional yang tinggi [5]. Untuk konsep load balancing [6] adalah

mekanisme penyeimbang beban yang bertujuan untuk mengoptimalkan pemanfaatan sumber daya, mempercepat waktu respons, dan mencegah kegagalan sistem akibat overload pada satu server. Penerapan load balancing diperlukan untuk mendistribusikan beban kerja secara merata di berbagai server, sehingga meningkatkan kinerja sistem dan menghindari satu titik kegagalan [7].

Untuk mendukung dalam segi implementasi sistem, dibutuhkan teknologi seperti microservices yang mendukung sistem clustering. Dalam perkembangan jenis komputasi modern cluster computing merupakan Solusi sistem yang mendukung dalam optimalisasi dari sisi kinerja dalam konteks ketersediaan sistem dan load balancing [8]. Menurut [9], Cluster computing merupakan kumpulan dari beberapa komputer yang terhubung satu dengan yang lain melalui jaringan Local Area Network (LAN) dengan setiap komputer berjalan sendiri dalam sebuah sistem operasi untuk menjalankan layanan tertentu. Salah satu teknologi yang dapat memenuhi teknologi tersebut adalah Kubernetes.

Kubernetes merupakan platform *open-source* untuk orkestrasi kontainer yang dirancang untuk mengelola pengelompokan aplikasi dalam bentuk *pod* secara otomatis [10]. Kubernetes memiliki kemampuan dalam penjadwalan, penskalaan, pemantauan, serta *self-healing*, yang memungkinkan sistem tetap beroperasi meskipun terjadi kegagalan pada salah satu node [11]. Jika dibandingkan dengan platform lain seperti Docker Swarm atau Proxmox, Kubernetes memiliki keunggulan dalam hal skalabilitas, otomatisasi, serta dukungan komunitas yang luas. Docker Swarm memang lebih sederhana dalam hal konfigurasi, tetapi memiliki keterbatasan dalam mekanisme *failover* [12]. Sedangkan Proxmox lebih berfokus pada virtualisasi berbasis *virtual machine* [13]. Dengan memanfaatkan Kubernetes, Sistem Informasi Akademik Undiksha dapat diimplementasikan dalam lingkungan *cluster container* yang mendukung *high availability* dan *load balancing* secara terintegrasi. Melalui mekanisme replikasi *pod* dan konfigurasi *load balancing* menggunakan NGINX [14], sistem dapat menangani lonjakan beban pengguna secara efisien.

Pendekatan horizontal scaling melalui sistem terdistribusi menjadi alternatif untuk meningkatkan ketahanan layanan. Kubernetes menyediakan mekanisme orkestrasi kontainer yang mendukung replikasi layanan, load balancing, serta pemulihan otomatis. Namun, implementasi Kubernetes umumnya diasosiasikan dengan kebutuhan sumber daya yang besar, sementara banyak institusi pendidikan memiliki keterbatasan infrastruktur.

Oleh karena itu, penelitian ini difokuskan pada evaluasi penerapan Kubernetes Cluster dalam lingkungan dengan sumber daya terbatas, untuk menilai sejauh mana sistem mampu meningkatkan high availability dan performa load balancing pada layanan SIAK. Tujuan penelitian ini adalah menganalisis ketahanan layanan berdasarkan parameter MTTR dan availability, serta mengevaluasi performa sistem melalui throughput, latency, dan error rate pada berbagai skenario beban dan kegagalan node.

2. Metode Penelitian

Penelitian ini bersifat eksperimental teknis yang berfokus pada evaluasi ketahanan layanan serta distribusi beban aplikasi pada Sistem Informasi Akademik (SIAK) melalui pengujian yang terkontrol pada arsitektur terdistribusi. Pengujian dilakukan menggunakan empat server fisik yang difungsikan sebagai node layanan dan basis data, tanpa melibatkan responden manusia. Evaluasi sistem difokuskan pada parameter berbasis hasil, meliputi:

a. Waktu pemulihan layanan (MTTR) saat terjadi kegagalan node,
b. Tingkat ketersediaan layanan (availability),
c. Throughput, latency, dan error rate pada variasi permintaan,
d. Distribusi penggunaan CPU dan RAM antar node layanan.

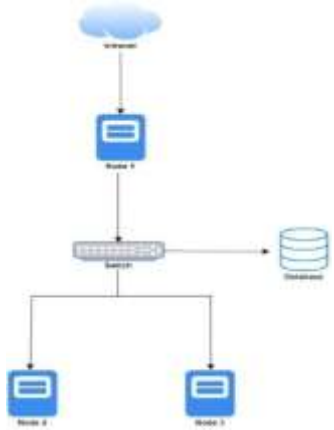
Seluruh pengujian dilakukan dengan pendekatan simulasi kegagalan dan beban akses bertahap untuk mengamati perilaku dari sistem dalam mempertahankan layanan.

2.1 Identifikasi Kebutuhan Sistem

Pada tahap ini berfokus pada analisis kebutuhan *high availability* dan *load balancing* serta asesmen kesiapan infrastruktur teknologi informasi eksisting di UPA TIK Universitas Pendidikan Ganesha. Pengumpulan data dilakukan melalui pendekatan empiris dan teoretis dengan metode observasi langsung, wawancara dengan pihak bidang infrastruktur, jaringan, dan keamanan, serta studi pustaka.

DOI: https://doi.org/10.31004/riggs.v4i4.5846

2.2 Perancangan Desain Arsitektur Sistem



Gambar 1 Rancangan Topologi Penelitian

Gambar 2 menunjukkan rancangan arsitektur sistem pengujian yang terdiri dari satu master node, dua worker node, dan satu database node terpisah. Pemisahan database dilakukan untuk memastikan bahwa pengukuran performa dan ketersediaan difokuskan pada lapisan aplikasi dan orkestrasi, bukan dipengaruhi oleh beban transaksi basis data. Topologi ini dirancang untuk mensimulasikan lingkungan produksi minimal dengan keterbatasan sumber daya, sehingga hasil pengujian dapat merepresentasikan kondisi implementasi Kubernetes pada institusi dengan infrastruktur terbatas.

2.3 Implementasi Infrastruktur Sistem

Pada tahap ini, peneliti merealisasikan rancangan sistem dalam jaringan Sistem Informasi Akademik Universitas Pendidikan Ganesha menggunakan tiga server yang terhubung ke jaringan UPA TIK Undiksha. Pengujian kinerja Kubernetes Cluster dilakukan dengan melibatkan pengguna dan memanfaatkan Apache JMeter untuk mengukur responsivitas serta performa sistem saat akses simultan. Tujuannya untuk memastikan integrasi komponen berjalan baik dan sistem mampu menangani beban pengguna secara optimal dan andal, dengan spesifikasi perangkat disesuaikan kebutuhan server SIAK.

2.4 Pengujian Ketersediaan dan Performa

Tahap ini bertujuan memastikan server berfungsi sesuai tujuan penelitian dengan bantuan tools Prometheus dan Grafana untuk mengumpulkan data layanan dan performa [15]. Pada analisis data, analisis *high availability* diukur berdasarkan jumlah "nine" pada tingkat ketersediaan, mengamati uptime dan downtime menggunakan Grafana dan Prometheus untuk menilai keandalan sistem [16]. Tabel 1 mendefinisikan skenario pengujian mengevaluasi sistem *high availability* (HA) dengan mematikan salah satu server worker (node 2 atau node 3 pada Kubernetes cluster) untuk menilai kemampuan sistem mempertahankan akses layanan dan waktu pemulihan.

Tabel 1 Tabel Skenario Uji Coba Ketersediaan Layanan

	1 Pod	2 Pod
No	Kondisi	Kondisi
1	Node 2 mati	Node 2 mati
2	Node 3 mati	Node 3 mati
3	Kedua node mati	Kedua node mati

Adapun beberapa parameter penting pada dari perhitungan availability yang harus terpenuhi. Dalam menghitung availability terdapat standar formula availability [17], pada rumus ini terdapat istilah MTBF atau mean time between failures dan MTTR atau mean time to repair. Berikut adalah bentuk rumus availability berdasarkan sumber dari Buku IBM Cloud Resiliency:

DOI: https://doi.org/10.31004/riggs.v4i4.5846

$$Availability = \frac{MTBF}{MTBF + MTTR}\ x\ 100 \quad (1)$$

Keterangan:

a) MTBF = Total waktu layanan beroperasi / jumlah kegagalan

b) MTTR = Total downtime / jumlah kegagalan

Pada uji performa, kinerja host dianalisis melalui pemanfaatan CPU, RAM, dan memori. Pengujian menggunakan load/stress client mengukur pengaruh load balancer dan tingkat error pada Kubernetes Cluster. Apache JMeter digunakan untuk mengirim sampel permintaan, mengukur responsivitas dan efektivitas sistem dalam menangani beban kerja tinggi.

Tabel 2 List Parameter testing uji performa

No	Pod	Request
1	1	1000
2	1	2500
3	1	5000
4	2	5000
5	2	10000
6	4	10000
7	4	20000

Tabel 2 mendefinisikan skenario pengujian performa berdasarkan variasi jumlah pod dan jumlah request. Parameter jumlah request merepresentasikan tingkat beban pengguna, sedangkan jumlah pod menunjukkan tingkat replikasi layanan. Korelasi antara kedua parameter ini digunakan untuk mengamati batas kapasitas sistem serta efektivitas mekanisme load balancing dalam menekan error rate dan latency pada kondisi beban menengah hingga tinggi.

Tabel 3 Korelasi parameter pada hasil penelitian

Parameter	Dampak yang Dianalisis
Request ↑	Beban sistem, error rate
Pod ↑	Distribusi beban
Pod vs Request	Efektivitas load balancing

Tabel 3 mendefinisikan korelasi parameter pada hasil penelitian. Peningkatan jumlah *request* merepresentasikan kenaikan beban sistem yang berdampak pada error rate dan waktu respons. Penambahan jumlah *pod* berfungsi untuk mendistribusikan beban ke beberapa instance layanan. Korelasi antara jumlah *pod* dan *request* digunakan untuk menilai efektivitas load balancing, yang ditunjukkan melalui penurunan error rate, latency yang lebih rendah, dan throughput yang lebih stabil. Data dan hasil pengujian *high availability* dan *load balancing* dijadikan dasar kesimpulan efektivitas sistem Kubernetes dalam meningkatkan ketersediaan dan performa Sistem Informasi Akademik Universitas Pendidikan Ganesha.

2.5 Evaluasi dan Rekomendasi

Tahap ini fokus pada kebijakan operasional, pemeliharaan, dan manajemen sistem untuk memastikan keberlanjutan dan keandalan jangka panjang. Kebijakan yang diterapkan dirancang agar sistem yang dibangun berfungsi optimal dan stabil dalam waktu lama. Peneliti menyusun pertimbangan dan rekomendasi bagi UPA TIK Undiksha berdasarkan hasil analisis sebelumnya, dengan tujuan meningkatkan efisiensi sistem serta menjamin operasional yang andal dan berkelanjutan.

DOI: https://doi.org/10.31004/riggs.v4i4.5846

3. Hasil dan Diskusi

3.1 Hasil Pengujian Ketersediaan Layanan

Tabel 4 Tabel Hasil Uji Coba Ketersediaan

Skenario	Node Mati	MTBF (menit)	MTTR (menit)	Availability (%)
HA-1	Node 2	4200	1,03	99,97
HA-2	Node 3	5640	1,20	99,97
HA-3	Node 2 & 3	–	–	–
HA-4	Node 2	6780	1,10	99,98
HA-5	Node 3	4200	1,03	99,97
HA-6	Node 2 & 3	–	–	–

Berdasarkan hasil pengujian ketersediaan layanan pada berbagai skenario kegagalan node pada tabel 3, dapat disimpulkan bahwa sistem mampu mempertahankan tingkat availability yang tinggi ketika terjadi kegagalan pada satu node worker, dengan nilai availability berada di atas 99,9%. Namun, kegagalan dua node worker secara bersamaan menyebabkan layanan tidak dapat dipulihkan, sehingga parameter availability tidak dapat dihitung pada skenario tersebut. Temuan ini menunjukkan bahwa tingkat ketersediaan layanan yang dicapai masih bergantung pada jumlah node worker yang aktif di dalam cluster. Dengan demikian, hasil penelitian ini mengindikasikan bahwa peningkatan jumlah node worker dan perencanaan redundansi infrastruktur perlu dipertimbangkan untuk mencapai tingkat *high availability* yang lebih optimal pada level node, khususnya dalam menghadapi skenario kegagalan ganda. Dari enam skenario pengujian, empat skenario digunakan dalam perhitungan rata-rata availability karena masih memenuhi kondisi operasional sistem, sedangkan dua skenario lainnya digunakan sebagai pengujian batas kegagalan sistem.

3.2 Hasil Pengujian Performa

Tabel 5 Hasil Pengujian Performa Keseluruhan

No	Pod	Request	Error Rate	Avg. Latency	Throughtput
1	1	1000	0%	1148 ms	333,8/sec
2	1	2500	0%	2921 ms	309,9/sec
3	1	5000	25,71%	4021 ms	483,5/sec
4	2	5000	0%	2530 ms	720,5/sec
5	2	10000	14,78%	2735 ms	769,6/sec
6	4	10000	0%	2037 ms	1053,7/sec
7	4	20000	0%	1927 ms	1059,0/sec

Tabel 5 merangkum hasil pengujian performa sistem berdasarkan variasi jumlah pod dan jumlah request. Pada kondisi satu pod, sistem masih mampu menangani hingga 2500 request tanpa error, namun mulai menunjukkan keterbatasan pada 5000 request dengan peningkatan error rate dan latency yang signifikan. Penambahan jumlah pod terbukti menurunkan error rate secara drastis dan meningkatkan throughput. Pada beban 5000 request, peningkatan pod dari satu menjadi dua mampu menghilangkan error dan menurunkan latency. Pola yang sama terlihat pada beban 10000 request, di mana penggunaan empat pod menghasilkan error rate 0% dengan throughput tertinggi dan latency terendah. Temuan ini menunjukkan bahwa peningkatan replikasi pod berkontribusi langsung terhadap efektivitas load balancing dan stabilitas performa sistem pada beban tinggi.

Tabel 6 Perbandingan pengujian 1000 request

	Single Server	Kubernetes Cluster		
		Master	Worker 1	Worker 2
Error Rate	0%		0%	
Throughput	334,1/sec		333,8/sec	
Avg.Latency	1143ms		1148 ms	
CPU	13,62%	5,85%	8,85%	1,54%
RAM	17,18%	30,10%	13,02%	16,28%

DOI: https://doi.org/10.31004/riggs.v4i4.5846

Tabel 6 menunjukkan bahwa pada pengujian dengan 1000 request, kinerja sistem single server dan Kubernetes Cluster relatif sebanding. Kedua arsitektur mampu memproses seluruh permintaan tanpa error, dengan nilai throughput dan rata-rata latency yang hampir sama. Kondisi ini mengindikasikan bahwa pada beban rendah, kapasitas single server masih mencukupi sehingga mekanisme load balancing belum memberikan peningkatan performa yang signifikan. Meskipun demikian, pada Kubernetes Cluster terlihat distribusi pemanfaatan CPU dan RAM yang lebih merata di antara master dan worker node. Hal ini menunjukkan bahwa Kubernetes telah melakukan pembagian beban kerja secara efektif, yang menjadi fondasi peningkatan stabilitas dan kinerja sistem pada skenario beban yang lebih tinggi.

Tabel 7 Perbandingan pengujian 2500 request

	Single Server	Kubernetes Cluster		
		Master	Worker 1	Worker 2
Error Rate	0%		0%	
Throughput	283,6/sec		309,9/sec	
Avg.Latency	3035ms		2921 ms	
CPU	19,83%	8,81%	21,52%	1,65%
RAM	16,99%	30,46%	13,32%	16,21%

Tabel 7 menunjukkan bahwa pada pengujian dengan 2500 request, kedua arsitektur masih mampu memproses seluruh permintaan tanpa error. Namun, Kubernetes Cluster mulai menunjukkan keunggulan kinerja dibandingkan single server, yang terlihat dari throughput yang lebih tinggi dan rata-rata latency yang lebih rendah. Distribusi penggunaan CPU dan RAM pada Kubernetes Cluster juga lebih tersebar pada worker node, menandakan mekanisme load balancing mulai bekerja secara efektif. Hasil ini mengindikasikan bahwa pada beban menengah, Kubernetes lebih mampu menjaga stabilitas dan efisiensi sistem dibandingkan arsitektur single server.

Tabel 8 Perbandingan pengujian 5000 request

	Single Server	Kubernetes Cluster		
		Master	Worker 1	Worker 2
Error Rate	39,51%		0%	
Throughput	305,2/sec		720,5/sec	
Avg.Latency	4592ms		2530 ms	
CPU	27,93%	11,60%	20,94%	28,83%
RAM	17,37%	30,51%	13,45%	14,47%

Tabel 8 menunjukkan perbedaan kinerja yang signifikan antara single server dan Kubernetes Cluster pada pengujian 5000 request. Arsitektur single server mengalami peningkatan error rate hingga 39,51% disertai latency yang tinggi, yang mengindikasikan keterbatasan kapasitas sistem dalam menangani beban menengah. Sebaliknya, Kubernetes Cluster mampu memproses seluruh request tanpa error dengan throughput lebih dari dua kali lipat dan latency yang jauh lebih rendah. Distribusi pemanfaatan CPU dan RAM pada worker node menunjukkan bahwa mekanisme replikasi pod dan load balancing berperan efektif dalam mendistribusikan beban kerja, sehingga sistem tetap stabil pada beban yang lebih tinggi.

Tabel 9 Perbandingan pengujian 10000 request

	Single Server	Kubernetes Cluster		
		Master	Worker 1	Worker 2
Error Rate	30,39%		0%	
Throughput	427,8/sec		1053,7/sec	
Avg.Latency	4237ms		2037 ms	
CPU	31,06%	13,07%	49,92%	45,99%
RAM	17,42%	31,25%	14,21%	15,52%

Tabel 9 menunjukkan bahwa pada pengujian dengan 10000 request, kinerja sistem single server semakin menurun, yang ditandai dengan error rate sebesar 30,39% dan latency yang tinggi. Kondisi ini mengindikasikan bahwa single server telah melampaui kapasitas optimalnya dalam menangani beban tinggi. Sebaliknya, Kubernetes Cluster mampu mempertahankan error rate 0% dengan throughput yang meningkat secara signifikan serta latency yang lebih rendah. Tingginya pemanfaatan CPU pada worker node menunjukkan bahwa beban kerja berhasil didistribusikan secara efektif, sehingga sistem tetap stabil dan responsif meskipun berada pada kondisi beban tinggi.

DOI: https://doi.org/10.31004/riggs.v4i4.5846

Tabel 10 Perbandingan pengujian 20000 request

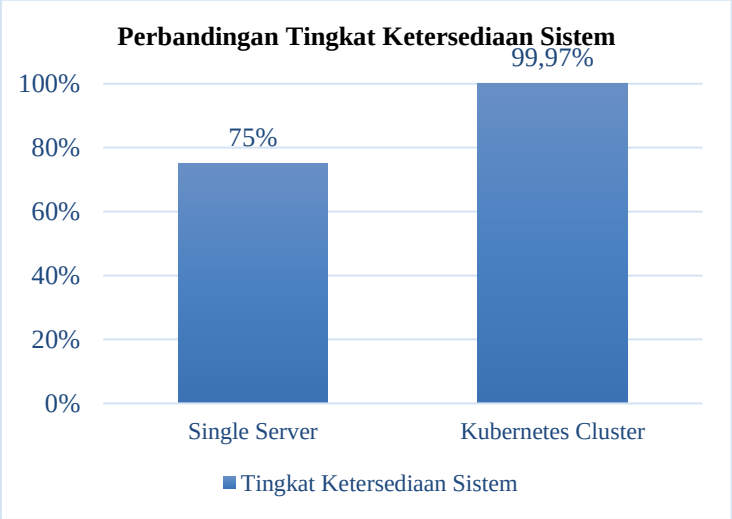
	Single Server	Kubernetes Cluster		
		Master	Worker 1	Worker 2
Error Rate	30,49%		0%	
Throughput	408,8/sec		1059,0/sec	
Avg.Latency	4429ms		1927 ms	
CPU	37,54%	17,00%	82,99%	77,56%
RAM	17,20%	30,66%	14,24%	15,47%

Tabel 10 menunjukkan bahwa pada pengujian dengan 20000 request, sistem single server mengalami degradasi kinerja yang konsisten, ditandai dengan error rate di atas 30% dan latency yang semakin tinggi. Kondisi ini menegaskan bahwa single server tidak mampu menangani beban sangat tinggi secara andal. Sebaliknya, Kubernetes Cluster tetap mampu memproses seluruh request tanpa error dengan throughput yang stabil dan latency yang lebih rendah. Tingginya pemanfaatan CPU pada worker node menunjukkan bahwa sistem bekerja mendekati kapasitas maksimum, namun mekanisme replikasi pod dan load balancing tetap mampu menjaga kontinuitas layanan. Temuan ini menegaskan keunggulan Kubernetes Cluster dalam menangani beban tinggi dibandingkan arsitektur single server.

3.3 Pembahasan

Pengujian *high availability* dilakukan dengan mensimulasikan kegagalan worker node. Hasil pengamatan menunjukkan bahwa layanan tetap dapat diakses setelah terjadi kegagalan, dengan waktu pemulihan yang relatif singkat. Berdasarkan pendekatan failover response time, sistem menunjukkan tingkat ketersediaan layanan yang tinggi. Namun perlu ditekankan bahwa nilai availability yang diperoleh bukan hasil pengukuran reliability jangka panjang, melainkan hasil observasi eksperimental dalam skenario terbatas. Oleh karena itu, angka availability yang dihasilkan tidak dapat dibandingkan secara langsung dengan standar SLA atau service level agreement pada lingkungan produksi.

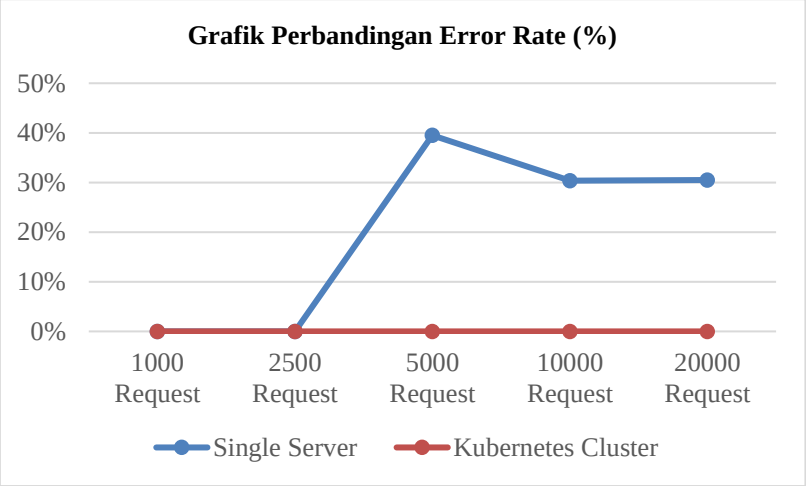
Berdasarkan hasil pengujian skenario kegagalan node pada sistem Kubernetes Cluster, diperoleh nilai *Mean Time To Repair* (MTTR) berada pada rentang 60 – 70 detik. Nilai ini menunjukkan waktu yang dibutuhkan sistem untuk memulihkan layanan hingga kembali dapat diakses oleh pengguna setelah terjadi gangguan. Berdasarkan hasil analisis yang ditemukan ada beberapa faktor kemungkinan yang memengaruhi kondisi dari rentang waktu tersebut [18]. Waktu pemulihan layanan dipengaruhi oleh beberapa faktor, yaitu waktu deteksi kegagalan node oleh *controller manager* sekitar 40 detik hingga status berubah menjadi *NotReady*, waktu toleransi pod sebesar 3 detik (*tolerationSeconds*), serta proses terminasi pod selama 3 detik (*terminationGracePeriodSeconds*). Selain itu, proses penjadwalan ulang dan inisialisasi pod baru, termasuk *pulling image* dan *startup* aplikasi Laravel, serta pembaruan objek oleh kubelet yang memerlukan waktu tambahan, turut memengaruhi durasi pemulihan. Gangguan jaringan saat proses *pull image* juga dapat menambah waktu pemulihan. Berdasarkan faktor-faktor tersebut, rentang MTTR 60–70 detik dapat dikategorikan sebagai hasil optimasi dari konfigurasi toleransi yang agresif pada sistem Kubernetes.



Gambar 2 Hasil Tingkat Ketersediaan Sistem

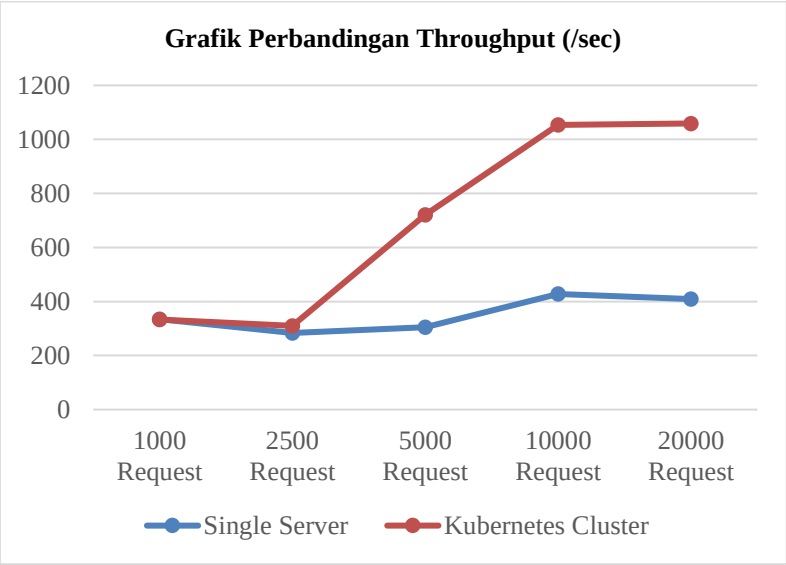
DOI: https://doi.org/10.31004/riggs.v4i4.5846

Berdasarkan Gambar 3 terlihat perbedaan yang signifikan antara tingkat ketersediaan layanan pada arsitektur single server dan Kubernetes cluster. Arsitektur single server memiliki tingkat availability sebesar 75%, yang disebabkan oleh waktu pemulihan layanan (MTTR) yang relatif lama karena sistem tidak dapat beroperasi kembali sebelum proses perbaikan oleh teknisi selesai. Sebaliknya, Kubernetes cluster menunjukkan tingkat availability rata-rata sebesar 99,97%, yang diperoleh dari beberapa skenario pengujian kegagalan node tunggal (HA-1, HA-2, HA-4, dan HA-5). Tingginya nilai availability pada Kubernetes cluster dipengaruhi oleh mekanisme self-healing dan failover, di mana pod secara otomatis dijadwalkan ulang ke node lain yang masih aktif ketika terjadi kegagalan. Hasil ini menunjukkan bahwa Kubernetes cluster mampu mempertahankan kontinuitas layanan secara lebih optimal dibandingkan arsitektur single server, khususnya dalam konteks ketersediaan Sistem Informasi Akademik.



Gambar 3 Grafik Perbandingan Error Rate

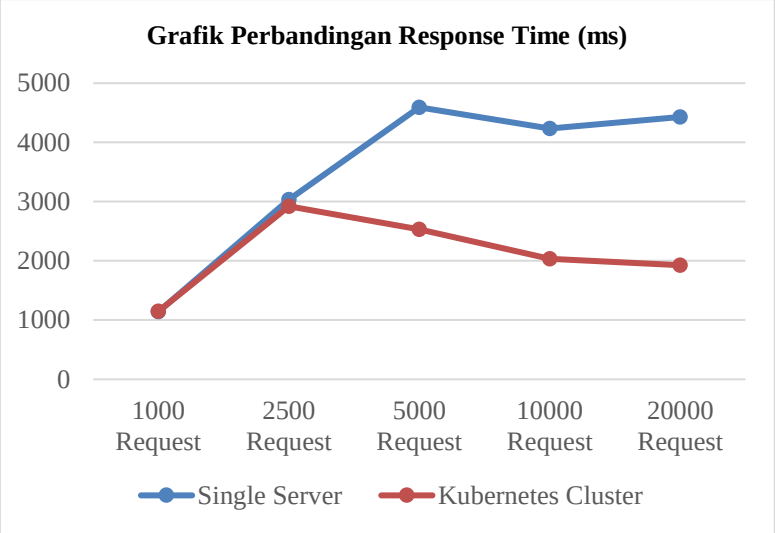
Gambar 3 menunjukkan tren error rate terhadap peningkatan jumlah request. Pada arsitektur single server, error rate mulai meningkat signifikan pada beban 5000 request dan terus bertahan di atas 30% pada beban yang lebih tinggi. Sebaliknya, Kubernetes Cluster mampu mempertahankan error rate 0% hingga 20000 request dengan penyesuaian jumlah pod. Temuan ini menunjukkan bahwa Kubernetes lebih resilien terhadap lonjakan beban dibandingkan single server.



Gambar 4 Grafik Perbandingan Throughput

Tren throughput pada gambar 4 menunjukkan bahwa throughput single server cenderung stagnan meskipun beban meningkat, yang mengindikasikan keterbatasan kapasitas sistem. Sebaliknya, Kubernetes Cluster menunjukkan peningkatan throughput yang konsisten seiring bertambahnya jumlah request dan pod, menandakan kemampuan sistem dalam melakukan scaling dan distribusi beban secara efektif.

DOI: https://doi.org/10.31004/riggs.v4i4.5846



Gambar 5 Grafik Perbandingan Response Time

Pada gambar 5 terlihat bahwa latency pada single server meningkat seiring bertambahnya beban request, sedangkan Kubernetes Cluster menunjukkan penurunan latency ketika jumlah pod ditingkatkan. Hal ini menegaskan bahwa mekanisme load balancing dan replikasi pod berkontribusi langsung terhadap peningkatan responsivitas sistem.

Secara keseluruhan, hasil analisis tren performa menunjukkan bahwa Kubernetes Cluster memiliki ketahanan dan skalabilitas yang lebih baik dibandingkan arsitektur single server, khususnya pada beban menengah hingga tinggi. Penerapan Kubernetes memungkinkan sistem mempertahankan stabilitas layanan, meningkatkan throughput, dan menekan error rate melalui mekanisme distribusi beban dan replikasi layanan, meskipun tetap dibatasi oleh kapasitas sumber daya fisik yang tersedia.

4. Kesimpulan

Berdasarkan hasil implementasi dan pengujian sistem berbasis Kubernetes Cluster pada layanan Sistem Informasi Akademik (SIAK) Universitas Pendidikan Ganesha, dapat disimpulkan bahwa penerapan arsitektur container orchestration mampu meningkatkan ketersediaan layanan dan efektivitas distribusi beban pada level aplikasi. Pengujian skenario kegagalan worker node menunjukkan bahwa Kubernetes mampu melakukan pemulihan layanan secara otomatis melalui mekanisme penjadwalan ulang pod, sehingga layanan tetap dapat diakses dengan waktu pemulihan yang relatif singkat dan nilai *Mean Time to Repair* (MTTR) yang rendah. Selain itu, penerapan replikasi pod dan mekanisme *load balancing* internal memberikan dampak positif terhadap kinerja sistem, yang ditunjukkan oleh peningkatan throughput dan distribusi beban kerja yang lebih merata pada pengujian dengan lebih dari satu pod. Meskipun demikian, pada skenario beban yang sangat tinggi, sistem masih menunjukkan keterbatasan performa yang dipengaruhi oleh kapasitas sumber daya perangkat keras dan konfigurasi cluster yang digunakan. Nilai availability yang diperoleh dalam penelitian ini merepresentasikan estimasi ketersediaan layanan berdasarkan skenario pengujian terbatas dan lebih tepat diinterpretasikan sebagai analisis ketahanan layanan (*service resilience*) pada level aplikasi, bukan sebagai ukuran reliability operasional jangka panjang. Secara keseluruhan, implementasi Kubernetes Cluster dalam penelitian ini bersifat *Pengujian terkontrol* dan berhasil menunjukkan potensi penerapan arsitektur berbasis orkestrasi kontainer untuk mengurangi risiko *single point of failure* serta meningkatkan resiliensi dan skalabilitas layanan SIAK. Namun, sistem yang dikembangkan belum mencakup *high availability* pada level *control plane*, sehingga diperlukan pengembangan dan pengujian lanjutan sebelum dapat direkomendasikan untuk penerapan pada lingkungan produksi.

Referensi

[1] Efendi, H.M, A., and Pasaribu, "Teknologi Sistem Informasi," *INNOVATIVE: Journal Of Social Science Research*, vol. Vol. 3, no. No. 2, pp. 43–53, 2023, Accessed: Jan. 08, 2026. [Online]. Available: https://j-innovative.org/index.php/Innovative/article/view/281

[2] Y. N. Febianti, E. Herawan, and A. L. Safitri, "Digital Literacy and Student Creativity Through E-Resources on the Quality of Learning in College," *Journal of Education Technology*, vol. 7, no. 1, pp. 25–33, 2023, doi: 10.23887/jet.v7i1.436.

[3] A. Saputri and E. Mulyani, "The Impact of Academic Information System to Students' Satisfaction in Higher Education," *Studies in Educational Management*, vol. 12, pp. 1–9, Sep. 2022, doi: 10.32038/sem.2022.12.01.

DOI: https://doi.org/10.31004/riggs.v4i4.5846

[4] M. P. Yadav, G. Raj, H. A. Akarte, and D. K. Yadav, "Horizontal scaling for containerized application using hybrid approach," *Ingenierie des Systemes d'Information*, vol. 25, no. 6, pp. 709–718, Dec. 2020, doi: 10.18280/isi.250601.

[5] N. Iryani, K. D. Ayatri, and R. D. Wahyuningrum, "Analisis performansi high availability cluster server menggunakan heartbeat pada private cloud," *JITEL (Jurnal Ilmiah Telekomunikasi, Elektronika, dan Listrik Tenaga)*, vol. 2, no. 2, pp. 129–138, Sep. 2022, doi: 10.35313/jitel.v2.i2.2022.129-138.

[6] R. Banerjee, "Integrating cloud load balancer with container orchestration for high availability," *International Journal of Multidisciplinary Research and Analysis*, vol. 8, no. 8, Aug. 2025, doi: 10.47191/ijmra/v8-i08-49.

[7] S. Sinaga and I. A. Sobari, "Implementasi Load Balancing pada Web Server Berbasis Container dalam Cluster Kubernetes pada PT Mandiri Utama Finance," *Jurnal Rekayasa Perangkat Lunak*, vol. 5, no. 1, 2024, [Online]. Available: http://jurnal.bsi.ac.id/index.php/reputasi

[8] S. K. Mondal, Z. Zheng, and Y. Cheng, "On the Optimization of Kubernetes toward the Enhancement of Cloud Computing," *Mathematics*, vol. 12, no. 16, Aug. 2024, doi: 10.3390/math12162476.

[9] M. A. F. Ridha and K. Gunawan, "Implementasi Cloud Computing Cluster Menggunakan Microstack Untuk Layanan Web," *Jurnal Komputer Terapan*, vol. 9, no. 2, pp. 122–133, Nov. 2023, doi: 10.35143/jkt.v9i2.6017.

[10] A. Poniszewska-Marańda and E. Czechowska, "Kubernetes cluster for automating software production environment," *Sensors*, vol. 21, no. 5, pp. 1–24, Mar. 2021, doi: 10.3390/s21051910.

[11] N. Hengky, E. Dazki, and E. Indrajit, "Proposed Implementation uses TOGAF ADM and ArchiMate - Enterprise Architecture in Retail Industry," *sinkron*, vol. 8, no. 4, pp. 2172–2184, Oct. 2024, doi: 10.33395/sinkron.v8i4.14052.

[12] M. Šimon, L. Huraj, and N. Búčik, "A Comparative Analysis of High Availability for Linux Container Infrastructures," *Future Internet*, vol. 15, no. 8, Aug. 2023, doi: 10.3390/fi15080253.

[13] H. Sturley, A. Fournier, A. Salcedo-Navarro, M. Garcia-Pineda, and J. Segura-Garcia, "Virtualization vs. Containerization, a Comparative Approach for Application Deployment in the Computing Continuum Focused on the Edge," *Future Internet*, vol. 16, no. 11, Nov. 2024, doi: 10.3390/fi16110427.

[14] C. Ma and Y. Chi, "Evaluation Test and Improvement of Load Balancing Algorithms of Nginx," *IEEE Access*, vol. 10, pp. 14311–14324, 2022, doi: 10.1109/ACCESS.2022.3146422.

[15] S. R. Dira, M. Arif, and F. Ridha, "Monitoring Kubernetes Cluster Menggunakan Prometheus dan Grafana," *Proceeding Applied Business and Engineering Conference*, vol. Vol. 10, no. 10th, pp. 17–19, 2022, Accessed: Jan. 08, 2026. [Online]. Available: https://abecindonesia.org/proceeding/index.php/abec/article/view/309/306

[16] G. Arna *et al.*, "Implementasi Proxmox Untuk High availability dan Load balancing Sistem SIAK UNDIKSHA," *Jurnal Algoritme*, vol. 6, no. 1, pp. 61–72, 2025, doi: 10.35957/algoritme.v6i1.11313.

[17] IBM, "IBM Cloud Resiliency." [Online]. Available: https://cloud.ibm.com/docs/resiliency.

[18] Kubernetes Authors, "Node Status," https://kubernetes.io/docs/.

DOI: https://doi.org/10.31004/riggs.v4i4.5846