Jurnal Teknologi dan Komputasi
Vol. 08, No. 3, September 2026

https://journalversa.com/s/index.php/jtk

ANALISIS DAN IMPLEMENTASI FAILOVER DAN AUTOSCALING MENGGUNAKAN KUBERNETES UNTUK SISTEM E-COMMERCE

Mochamad Taufik Romdony[1], Budi Tjahjono[2], Hermansyah[3], Ari Pambudi[4]
[1,2,3,4]Universitas Esa Unggul
Email: taufik.romdony@student.esaunggul.ac.id[1], budi.tjahjono@esaunggul.ac.id[2], hermansyah@esaunggul.ac.id[3], ari.pambudi@esaunggul.ac.id[4]

Abstrak

PT. XYZ merupakan perusahaan ritel yang mengembangkan kanal e-commerce, sehingga ketersediaan layanan dan kemampuan menangani lonjakan beban terutama saat flash sale bulanan menjadi kebutuhan utama. Infrastruktur existing berbasis virtualisasi VMware belum memiliki mekanisme elastis yang menyesuaikan kapasitas secara otomatis terhadap perubahan beban maupun kegagalan komponen, sementara biaya lisensi VMware per core meningkat signifikan pasca-akuisisi Broadcom. Penelitian ini menganalisis dan mengimplementasikan mekanisme failover dan autoscaling menggunakan cluster Kubernetes pada aplikasi web e-commerce berbasis microservices yang di-deploy pada Google Kubernetes Engine (GKE). Metode penelitian bersifat eksperimental, meliputi perancangan cluster, deployment microservices, pengujian failover pada tingkat node dan service, serta pengujian autoscaling menggunakan Horizontal Pod Autoscaler (HPA) dengan target utilisasi CPU 50%; beban uji dibangkitkan menggunakan k6.io. Hasil menunjukkan kegagalan pada tingkat service dipulihkan rata-rata 12,61 detik, sedangkan pada tingkat node 164,71 detik. Waktu respons scale up 12 detik dan scale down 120 detik (stabilization window), dengan utilisasi CPU per pod yang menurun setelah HPA menambah replika (mis. frontend 65% menjadi ~35% pada 2 pod). Penerapan Kubernetes terbukti meningkatkan ketahanan dan efisiensi sumber daya sistem e-commerce sehingga dapat menjadi acuan pengambilan keputusan migrasi ke arsitektur cloud-native.
Kata Kunci: Kubernetes, Failover, Autoscaling, Layanan Mikro, Cloud-Native.

Abstract

PT. XYZ is a retail company that has developed an e-commerce channel, making service availability and the ability to handle traffic spikes—especially during monthly flash sales—critical requirements. The existing VMware-based virtualization infrastructure lacks an elastic mechanism to automatically adjust capacity to workload changes or component failures, while VMware per-core licensing costs have risen significantly following the Broadcom acquisition. This study analyzes and implements failover and autoscaling mechanisms using a Kubernetes cluster on a microservices-based e-commerce application deployed on Google Kubernetes Engine (GKE). An experimental method was used, covering cluster design, microservices deployment, failover testing at the node and service levels, and autoscaling testing using the Horizontal Pod Autoscaler (HPA) with a target CPU utilization of 50%; test load was generated using k6.io. The results show that service-level failures recovered in an average of 12.61 seconds, while node-level failures required 164.71 seconds. The scale-up response time

was 12 seconds and scale-down 120 seconds (stabilization window), with per-pod CPU utilization dropping after the HPA added replicas (e.g., frontend from 65% to ~35% across two pods). Implementing Kubernetes improves the resilience and resource efficiency of the e-commerce system, providing a reference for migration decisions toward a cloud-native architecture.
Keywords: *Kubernetes, Failover, Autoscaling, Microservices, Cloud-Native.*

1. PENDAHULUAN

Ekonomi digital dan transaksi e-commerce di Indonesia tumbuh pesat, meningkat dari sekitar US$41 miliar pada 2019 menjadi US$77 miliar pada 2022 dengan e-commerce sebagai kontributor terbesar [1]. Salah satu pemicu lonjakan permintaan paling nyata adalah program flash sale yang diselenggarakan hampir setiap bulan, seperti promo tanggal kembar (9.9, 10.10, 11.11, 12.12), payday sale, dan Hari Belanja Online Nasional (Harbolnas) [2]. Pada periode tersebut jumlah pengunjung dan transaksi dapat melonjak beberapa kali lipat dalam waktu singkat, sehingga menimbulkan beban puncak yang sulit diprediksi dan berisiko menurunkan waktu respons hingga kegagalan layanan apabila kapasitas server bersifat statis. Untuk memitigasi kondisi tersebut diperlukan mekanisme autoscaling yang menambah dan mengurangi kapasitas layanan secara otomatis [3].

PT. XYZ menjalankan sistem e-commerce di atas infrastruktur virtualisasi VMware yang bersifat statis dan belum memiliki mekanisme elastis untuk menghubungkan perubahan beban dengan penyesuaian kapasitas maupun pemulihan otomatis saat terjadi kegagalan. Selain kendala teknis, terdapat pertimbangan ekonomis: pasca-akuisisi oleh Broadcom, model lisensi VMware berubah menjadi berlangganan per core CPU dengan biaya yang meningkat signifikan [4]. Kondisi ini mendorong evaluasi alternatif berbasis container orchestration yang tidak bergantung pada lisensi hypervisor proprietary.

Kubernetes telah menjadi standar de facto orkestrasi container [5], dan pendekatan self-adaptive berbasis Kubernetes dapat meningkatkan resiliensi serta skalabilitas layanan microservices [6]. Namun kemampuan self-healing perlu diuji secara empiris karena efektivitas deteksi berbeda menurut jenis kegagalan [7], sementara rancangan failover yang tepat dapat menekan downtime secara signifikan [8]. Pada sisi elastisitas, Horizontal Pod Autoscaler (HPA) bawaan Kubernetes bersifat reaktif [9]; tuning parameter HPA dapat mengurangi konsumsi sumber daya [10] dan penyesuaian threshold adaptif menekan

overprovisioning [11], sedangkan koordinasi autoscaling container dan virtual machine berpengaruh pada efisiensi biaya [12]. Ketersediaan tinggi pada arsitektur microservices bergantung pada desain redundansi dan mekanisme pemulihan [13].

Landasan konsep cloud computing dan model layanannya (IaaS, PaaS, SaaS) beserta model deployment dibahas pada [14] dan [15]; perbandingan container terhadap virtual machine pada [16]; peran web server dan load balancing pada [17]; serta konsep jaringan komputer dan arsitektur WAN modern pada [18] dan [19]. Penelitian terdahulu di Indonesia menunjukkan autoscaling berbasis Kubernetes mampu menyesuaikan kapasitas layanan terhadap beban [20].

Kebaruan penelitian ini terletak pada penggabungan pengujian failover (tingkat node dan service) dan autoscaling (scale up dan scale down) sekaligus pengukuran efisiensi penggunaan CPU dalam satu studi kasus e-commerce berbasis microservices nyata yang dijalankan pada Google Kubernetes Engine (GKE). Sebagian besar penelitian terdahulu hanya berfokus pada salah satu aspek. Tujuan penelitian adalah mengimplementasikan Kubernetes pada sistem e-commerce, menerapkan failover dan autoscaling, serta menganalisis pengaruh autoscaling terhadap efisiensi penggunaan CPU, sehingga menghasilkan baseline teknis yang relevan bagi pengambilan keputusan migrasi infrastruktur menuju arsitektur cloud-native.

2. METODE PENELITIAN

Penelitian menggunakan metode eksperimental yang meliputi perancangan cluster Kubernetes, deployment aplikasi microservices, serta pengujian failover dan autoscaling. Data dikumpulkan melalui observasi dan wawancara terhadap tim IT PT. XYZ untuk memetakan kondisi infrastruktur existing, serta studi pustaka dari artikel jurnal terkini. Cluster dibangun pada Google Kubernetes Engine (GKE) bernama demo-online-shop di zona asia-southeast2-b (Jakarta) dengan konfigurasi pada Tabel 1. GKE dipilih sebagai lingkungan uji yang representatif karena pengadaan server fisik khusus tidak memungkinkan dari sisi biaya dan waktu, sementara mekanisme *failover* dan *autoscaling* Kubernetes bersifat *platform-agnostic* sehingga pola hasilnya tetap dapat menjadi acuan bagi lingkungan *on-premises* PT. XYZ.

Tabel 1. Konfigurasi Cluster Kubernetes

Komponen	Konfigurasi
Nama Cluster	demo-online-shop
Zona	asia-southeast2-b (Jakarta)
Jumlah Node	1 control plane + 2 worker
Versi Kubernetes	v1.28 (default GKE)
Tipe Node	e2-standard-2 (2 vCPU, 8 GB)
Sistem Operasi	Container-Optimized OS
Autoscaling	Horizontal Pod Autoscaler
Failover	Scheduler & node health

Aplikasi web e-commerce yang digunakan adalah aplikasi microservices berbasis repositori open-source microservices-demo dari Google, terdiri atas 11 layanan yang saling terhubung melalui protokol gRPC. Deployment dilakukan dengan perintah kubectl apply terhadap berkas manifest Kubernetes. Autoscaling diaktifkan melalui HPA dengan target utilisasi CPU 50% (scale up saat di atas 50%, scale down saat di bawah 50%).

Pengujian failover dilakukan pada dua skenario: (1) node failure, yaitu mematikan salah satu worker node sehingga pod harus dijadwalkan ulang ke node sehat; dan (2) service failure, yaitu menghapus salah satu pod menggunakan kubectl delete pod sehingga Kubernetes memicu self-healing. Parameter yang diukur adalah waktu pemulihan (recovery time). Pengujian autoscaling dilakukan dengan membangkitkan beban tinggi secara terus-menerus menggunakan k6.io hingga penggunaan CPU melewati threshold, kemudian mengukur waktu scale up dan scale down. Pengaruh autoscaling terhadap penggunaan CPU dievaluasi dengan

membandingkan konsumsi CPU pada konfigurasi replika statis dan konfigurasi autoscaling, dipantau melalui kubectl top pods dan Google Cloud Monitoring.

3. HASIL DAN PEMBAHASAN

A. Implementasi dan Deployment Sistem

Seluruh 11 pod microservices berhasil dijalankan dan berada dalam status Running dengan kolom READY bernilai 1/1 tanpa restart. Layanan frontend di-expose ke internet melalui Service bertipe LoadBalancer, dan validasi transaksi (menambah produk, memilih mata uang, mengisi alamat, hingga pembayaran) menunjukkan seluruh microservices terintegrasi dengan baik. Hal ini menjawab kebutuhan implementasi Kubernetes pada prototipe sistem e-commerce.

B. Pengujian Failover

Tabel 2. Hasil Pengujian Failover

Jenis Kegagalan	Rata-rata Waktu Pemulihan
Node Failure	164,71 detik
Service Failure	12,61 detik

Tabel 2 menunjukkan perbedaan yang jelas antara service failure dan node failure. Pemulihan pada tingkat service jauh lebih cepat karena control plane tetap aktif dan langsung menjadwalkan ulang pod pengganti, sedangkan pada node failure Kubernetes harus mendeteksi node yang tidak responsif, menandainya NotReady, lalu merelokasi pod ke node sehat sehingga menimbulkan jeda tambahan. Pola ini konsisten dengan konsep self-healing yang dibahas pada [7] dan pentingnya rancangan failover untuk menekan downtime [8]. Karena itu, konfigurasi default untuk node failure masih perlu dioptimalkan (health check, readiness/liveness probe, dan pengaturan node) agar target ketersediaan tercapai konsisten.

C. Pengujian Autoscaling

Tabel 3. Rata-rata Waktu Autoscaling

Proses	Rata-rata Waktu
Scale Up	12 detik (respons terukur)
Scale Down	120 detik (stabilization window)

Sistem lebih cepat menambah kapasitas (scale up) daripada menguranginya (scale down). Waktu respons scale up 12 detik merupakan gabungan waktu observasi metrik, keputusan autoscaler, pembuatan pod, dan kesiapan pod menerima request, sesuai karakter HPA yang reaktif [9]. Scale down yang lebih lama menunjukkan kehati-hatian sistem menghindari under-provisioning saat beban masih berfluktuasi, sejalan dengan prinsip graceful degradation. Hasil ini memperkuat pentingnya tuning parameter autoscaler [10] dan penyesuaian threshold [11].

D. Efisiensi Penggunaan CPU

Tabel 4. Perbandingan Penggunaan CPU

Kondisi	Utilisasi CPU per Pod (frontend)
Sebelum auto-scaling (1 pod)	65%
Sesudah auto-scaling (2 pod)	~35%

Penggunaan CPU per pod pada kondisi autoscaling menurun setelah beban terdistribusi ke lebih banyak replika saat scale up (mis. frontend 65% menjadi ~35% pada 2 pod, currencyservice 64% menjadi ~14% pada 3 pod). Namun, interpretasi efisiensi tidak berhenti pada rendahnya persentase per pod; konsumsi sumber daya agregat seluruh replika perlu diperhitungkan karena penambahan pod dapat menurunkan utilisasi per pod tetapi menambah

konsumsi total. Optimasi yang lebih utuh perlu mengoordinasikan jumlah container dan kapasitas node [12].

E. Perbandingan dengan Penelitian Terdahulu

Berbeda dengan penelitian terdahulu yang umumnya membahas satu aspek—failover saja [7][8] atau autoscaling saja [9][10][11][20]—penelitian ini menggabungkan failover (node dan service), autoscaling (scale up dan scale down), dan pengukuran efisiensi CPU dalam satu studi kasus e-commerce berbasis microservices nyata pada GKE. Temuan menguatkan posisi Kubernetes sebagai platform orkestrasi yang relevan untuk aplikasi dengan kebutuhan availability dan workload dinamis [5][6], sekaligus menunjukkan parameter yang masih perlu dioptimalkan (penanganan node failure, pilihan metrik, dan perhitungan resource agregat) sebelum penerapan pada lingkungan produksi.

4. KESIMPULAN DAN SARAN

Implementasi Kubernetes pada prototipe sistem e-commerce PT. XYZ berhasil dilakukan; aplikasi microservices dengan 11 layanan berjalan stabil pada cluster GKE. Mekanisme failover dan autoscaling berjalan otomatis: failover pada tingkat service dipulihkan rata-rata 12,61 detik dan node 164,71 detik, sedangkan autoscaling menghasilkan waktu respons scale up 12 detik dan scale down 120 detik. Utilisasi CPU per pod menurun setelah autoscaling menambah replika (mis. frontend 65% menjadi ~35%). Alhasil, penerapan Kubernetes meningkatkan ketahanan dan efisiensi pengelolaan sumber daya sistem e-commerce dan dapat menjadi acuan teknis migrasi dari virtualisasi statis menuju arsitektur cloud-native. Pengembangan selanjutnya diarahkan pada optimasi penanganan node failure, penggunaan custom metrics (memori, latency, RPS), pengukuran resource agregat dan biaya, serta penambahan aspek keamanan dan Cluster Autoscaler.

DAFTAR PUSTAKA

Google, Temasek, and Bain & Company, "e-Conomy SEA 2022: Through the Waves, Towards a Sea of Opportunity," 2022.

Kredivo and Katadata Insight Center, "Laporan Perilaku Konsumen E-Commerce Indonesia 2023," Jakarta, 2023.

Nerella, "Intelligent Autoscaling for E-Commerce Applications in Public Cloud Kubernetes Environment," Lect. Notes Netw. Syst., vol. 1356, pp. 143–155, 2025, doi: 10.1007/978-981-96-5238-9_14.

Broadcom, "Counting Cores for VMware Cloud Foundation and vSphere Foundation," Broadcom Knowledge Base, 2024.

C. Carrión, "Kubernetes as a Standard Container Orchestrator - A Bibliometric Analysis," J. Grid Comput., vol. 20, p. 42, 2022, doi: 10.1007/s10723-022-09629-8.

J. Figueira and C. Coutinho, "Developing Self-Adaptive Microservices," Procedia Comput. Sci., vol. 232, pp. 264–273, 2024, doi: 10.1016/j.procs.2024.01.026.

J. Flora, P. Gonçalves, M. Teixeira, and N. Antunes, "A Study on the Aging and Fault Tolerance of Microservices in Kubernetes," IEEE Access, vol. 10, pp. 132786–132799, 2022, doi: 10.1109/ACCESS.2022.3231191.

K. Pakrijauskas and D. Mažeika, "A Method of Transparent Graceful Failover in Low Latency Stateful Microservices," Electronics, vol. 11, no. 23, p. 3936, 2022, doi: 10.3390/electronics11233936.

S. K. Mondal et al., "Toward Optimal Load Prediction and Customizable Autoscaling Scheme for Kubernetes," Mathematics, vol. 11, no. 12, p. 2675, 2023, doi: 10.3390/math11122675.

D. R. Augustyn, Ł. Wyciślik, and M. Sojka, "Tuning a Kubernetes Horizontal Pod Autoscaler for Meeting Performance and Load Demands in Cloud Deployments," Appl. Sci., vol. 14, no. 2, p. 646, 2024, doi: 10.3390/app14020646.

O. Pozdniakova, D. Mažeika, and A. Cholomskis, "SLA-Adaptive Threshold Adjustment for a Kubernetes Horizontal Pod Autoscaler," Electronics, vol. 13, no. 7, p. 1242, 2024, doi: 10.3390/electronics13071242.

J. Entrialgo et al., "Joint Autoscaling of Containers and Virtual Machines for Cost Optimization in Container Clusters," J. Grid Comput., vol. 22, p. 17, 2024, doi: 10.1007/s10723-023-09732-4.

M. Mena, J. Criado, L. Iribarne, and A. Corral, "Towards High-Availability Cyber-Physical Systems Using a Microservice Architecture," Computing, vol. 105, pp. 1745–1768, 2023, doi: 10.1007/s00607-023-01165-x.

R. Younis, M. Iqbal, K. Munir, M. A. Javed, M. Haris, and S. Alahmari, "A Comprehensive Analysis of Cloud Service Models: IaaS, PaaS, and SaaS in the Context of Emerging Technologies and Trends," in 2024 Int. Conf. on Electrical, Communication and Computer Engineering (ICECCE), IEEE, 2024.

E. Yumami, Irfansyah, M. K. Anam, and Hamdani, "Implementation of Cloud Computing Based on Infrastructure as a Service (IaaS) to Improve Transaction Quality," J. Teknol. dan Open Source, vol. 6, no. 1, pp. 86–97, 2023, doi: 10.36378/jtos.v6i1.3127.

H. Aqasizade, E. Ataie, and M. Bastam, "Experimental Assessment of Containers Running on Top of Virtual Machines," IET Networks, 2025, doi: 10.1049/ntw2.12138.

C. Ma and Y. Chi, "Evaluation Test and Improvement of Load Balancing Algorithms of Nginx," IEEE Access, vol. 10, pp. 14311–14324, 2022.

J. Dudczyk, M. Sergiel, and J. Krygiel, "Analysis of SD-WAN Architectures and Techniques for Efficient Traffic Control Under Transmission Constraints," Sensors, vol. 25, no. 20, p. 6317, 2025, doi: 10.3390/s25206317.

M. G. S. Wirya, Wawan, and Mahmudin, "Perkembangan Teknologi Jaringan Komputer Dari PAN, LAN, MAN, WAN Hingga 5G," JRIIN J. Ris. Inform. dan Inov., vol. 3, no. 7, pp. 1544–1548, 2025.

A. R. Nasution, F. Dewanta, and B. Aditya, "Auto Scaling Database Service with Micro Kubernetes Cluster," J. Tek. Inform. (JUTIF), vol. 3, no. 4, pp. 1075–1082, 2022