Rancang Bangun dan Evaluasi Kinerja *Proof of Concept* (PoC) Infrastruktur *High Availability* Berbasis Proxmox VE (Studi Kasus: PT Global Intermedia Nusantara)

Fitra Maulana[1], Ardhi Wicaksono Santoso[1]*
[1] Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada;
fitramaulana@mail.ugm.ac.id
*Korespondensi: ardhi.wicaksono.s@ugm.ac.id

Abstract – Single-server infrastructure is vulnerable to Single Point of Failure (SPOF) which risks causing downtime and data loss. This research aims to design, implement, and evaluate the performance of a High Availability (HA) infrastructure Proof of Concept (PoC) based on Proxmox VE in an on-premise environment, with a case study at PT Global Intermedia Nusantara. The system integrates Nginx as a Load Balancer, Node.js application cluster, PostgreSQL Streaming Replication, Apache Kafka for event-driven communication, and Prometheus-Grafana for observability. Testing results demonstrate that the infrastructure successfully achieved high availability and end-to-end data consistency. Nginx effectively distributed traffic with 100% availability during service failure. PostgreSQL replication operated in real-time with a peak CPU processing spike of 2.32 milliseconds and executed fail-safe protection to prevent split-brain. Kafka integration handled idempotency logic and guaranteed Zero Data Loss despite an approximate 30-second rebalancing delay and a maximum transient lag of 2 messages during network instability. Observability demonstrated system efficiency with minimal overhead through scrape durations under 360 ms for JMX and under 50 ms for other exporters. The PoC proved feasible and reliable for enhancing system fault tolerance on limited-specification hardware.

Keywords – High Availability, Proxmox VE, PostgreSQL Streaming Replication, Apache Kafka, Observability

Intisari – Infrastruktur *server* tunggal rentan terhadap *Single Point of Failure* (SPOF) yang berisiko menyebabkan *downtime* dan kehilangan data. Penelitian ini bertujuan merancang bangun dan mengevaluasi kinerja *Proof of Concept* (PoC) infrastruktur *High Availability* (HA) berbasis Proxmox VE pada lingkungan *on-premise* dengan studi kasus di PT Global Intermedia Nusantara. Sistem dibangun dengan mengintegrasikan Nginx sebagai *Load Balancer*, klaster aplikasi Node.js, PostgreSQL *Streaming Replication*, Apache Kafka untuk komunikasi *event-driven*, serta Prometheus-Grafana untuk observabilitas. Hasil pengujian menunjukkan infrastruktur ini berhasil mencapai ketersediaan tinggi dan konsistensi data secara *end-to-end*. Nginx mendistribusikan beban trafik dengan ketersediaan 100% saat kegagalan layanan. Replikasi PostgreSQL berjalan *real-time* dengan lonjakan CPU rata-rata 2,32 milidetik serta mampu menjalankan proteksi *fail-safe* untuk mencegah *split-brain*. Kafka menjamin *Zero Data Loss* meski terjadi jeda *rebalancing* sekitar 30 detik dengan *transient lag* maksimal 2 pesan. Observabilitas membuktikan efisiensi sistem dengan *overhead* minimal melalui durasi scrape di bawah 360 ms untuk JMX dan di bawah 50 ms untuk *exporter* lainnya. PoC ini terbukti layak dan andal untuk meningkatkan toleransi kesalahan pada perangkat keras berspesifikasi terbatas.

***Kata kunci – High Availability*, Proxmox VE, PostgreSQL *Streaming Replication*, Apache Kafka, Observabilitas**

I. PENDAHULUAN

Perkembangan teknologi informasi yang pesat telah menuntut sistem layanan digital untuk memiliki standar ketersediaan tinggi *(High Availability)* dan konsistensi data yang handal. Kebutuhan akan layanan yang mampu beroperasi tanpa henti menjadi semakin krusial, terutama bagi perusahaan yang menggantungkan operasional bisnisnya pada aplikasi berbasis *web* dan sistem basis data terintegrasi [1]. Salah satu pendekatan strategis yang kini banyak diterapkan adalah pemanfaatan teknologi virtualisasi *server* yang menawarkan efisiensi perangkat keras sekaligus menjadi fondasi dalam membangun infrastruktur yang redundan dan toleran terhadap kegagalan [2].

Dalam konteks operasional di PT Global Intermedia Nusantara, sebuah perusahaan di bidang solusi teknologi informasi, tuntutan akan keandalan sistem semakin mendesak seiring bertambahnya volume transaksi klien. Meskipun kebutuhan arsitektur *High Availability* (HA) sangat mendesak, penerapan perubahan arsitektur secara langsung pada *server* produksi memiliki risiko operasional yang tinggi. Oleh karena itu, diperlukan pendekatan berbasis *Proof of Concept* (PoC) untuk merancang, menguji, dan mengevaluasi kinerja arsitektur baru dalam lingkungan yang terisolasi sebelum diimplementasikan pada skala produksi.

Platform virtualisasi Proxmox VE memainkan peran vital dalam manajemen infrastruktur modern melalui mekanisme *containerization* (LXC) dan *full virtualization* (KVM). Teknologi ini menyediakan kapabilitas membangun arsitektur redundansi pada level aplikasi, seperti penerapan *load balancer* pada lapisan *web* dan replikasi data pada lapisan basis data [3]. Penelitian terdahulu menunjukkan bahwa implementasi klaster HA berbasis Proxmox mampu menurunkan waktu *downtime* secara signifikan serta meningkatkan *throughput* sistem [4].

Tantangan tidak hanya pada ketersediaan *server*, tetapi juga pada jaminan konsistensi data antar-layanan. Penerapan PostgreSQL *streaming replication* dan Apache Kafka telah menjadi tren utama untuk menangani isu ini. Apache Kafka berfungsi sebagai *backbone* komunikasi data yang menjamin pengiriman pesan secara *real-time* dengan latensi rendah. Penelitian menunjukkan integrasi teknologi ini mampu meningkatkan *throughput* hingga 30% sekaligus meminimalisir latensi [5], serta memperkuat ketahanan sistem melalui mekanisme *failover* [6]. Selain itu, aspek

observabilitas melalui integrasi Prometheus dan Grafana memungkinkan administrator memvisualisasikan metrik kinerja secara dinamis untuk mitigasi gangguan yang lebih cepat [7].

Meskipun teknologi-teknologi tersebut menawarkan solusi yang menjanjikan secara terpisah, integrasi menyeluruh dalam lingkungan *on-premise* dengan sumber daya terbatas menghadirkan tantangan tersendiri. Penelitian ini bertujuan untuk: (1) merancang dan mengimplementasikan PoC arsitektur HA menggunakan Proxmox VE, Nginx, PostgreSQL *Streaming Replication*, dan Apache Kafka; serta (2) mengevaluasi kinerja sistem berdasarkan parameter efektivitas *failover*, distribusi trafik, *consumer lag*, dan efisiensi penggunaan sumber daya.

II. DASAR TEORI

A. Proxmox *Virtual Environment* (Proxmox VE)

Proxmox VE merupakan platform virtualisasi *open-source* yang mendukung dua teknologi virtualisasi utama, yaitu *Kernel-based Virtual Machine* (KVM) untuk *full virtualization* dan *Linux Containers* (LXC) untuk *container-based virtualization*. Platform ini menyediakan antarmuka *web* terpusat untuk manajemen mesin virtual, *container*, penyimpanan, dan jaringan. Dalam konteks HA, Proxmox VE memungkinkan pembangunan arsitektur redundansi pada level aplikasi melalui pengelolaan *multi-node* secara efisien [1,4].

B. Konsep *High Availability* (HA)

High Availability mengacu pada kemampuan sistem untuk tetap beroperasi secara kontinu meskipun terjadi kegagalan pada salah satu komponen. Prinsip utama HA meliputi redundansi komponen, mekanisme *failover* otomatis, dan eliminasi *Single Point of Failure* (SPOF). Pada lingkungan virtual, HA diimplementasikan melalui replikasi layanan pada beberapa *node* sehingga ketika satu *node* gagal, *node* lain dapat mengambil alih beban kerja secara otomatis [1-2].

C. *Load Balancing* dengan Nginx

Load balancing merupakan teknik distribusi beban trafik jaringan ke beberapa *server* secara merata untuk mengoptimalkan penggunaan sumber daya dan menjaga ketersediaan layanan. Nginx mendukung beberapa algoritma distribusi seperti *Round Robin*, *Least Connections*, dan IP Hash. Dalam arsitektur HA, Nginx berperan ganda sebagai *reverse proxy* dan *load balancer* yang mengarahkan permintaan pengguna ke *node* aplikasi yang tersedia [5].

D. PostgreSQL *Streaming Replication*

PostgreSQL *Streaming Replication* adalah mekanisme replikasi data yang memungkinkan *server standby* menerima dan menerapkan perubahan *Write-Ahead Log* (WAL) dari *server primary* secara kontinu. Mekanisme ini mendukung mode sinkron dan asinkron, di mana mode asinkron menawarkan latensi penulisan yang lebih rendah dengan *trade-off* berupa potensi kehilangan data minimal saat *failover* [10].

E. Apache Kafka

Apache Kafka adalah platform *distributed streaming* yang dirancang untuk menangani aliran data *real-time* dengan *throughput* tinggi dan latensi rendah. Kafka menggunakan arsitektur *publish-subscribe* dengan konsep topik, partisi, dan grup *consumer*. Fitur replikasi antar-*broker* menjamin ketersediaan data meskipun terjadi kegagalan pada salah satu *node*. Mekanisme *at-least-once delivery* memastikan setiap pesan terkirim minimal satu kali, sementara logika idempotensi pada sisi *consumer* mencegah duplikasi data [7-9].

F. Observabilitas dengan Prometheus dan Grafana

Prometheus adalah sistem *monitoring* dan *alerting open-source* yang mengumpulkan metrik dari target yang dikonfigurasi melalui mekanisme *pull (scraping)* pada interval waktu tertentu. Grafana berfungsi sebagai platform visualisasi yang mengubah data metrik menjadi *dashboard* interaktif. Integrasi keduanya memungkinkan pemantauan kesehatan sistem secara *real-time*, deteksi anomali, dan analisis kinerja infrastruktur terdistribusi [3].

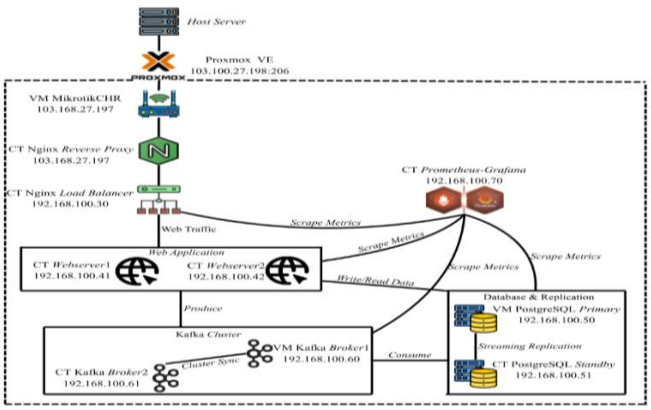
III. METODOLOGI

A. Lingkungan Penelitian

Penelitian dilakukan pada lingkungan simulasi *single-host deployment* menggunakan perangkat keras dengan spesifikasi Intel *Core* i7-7567U (2 *core*, 4 *thread*), RAM 16 GB, dan *storage* 1 TB SSD yang menjalankan Proxmox VE sebagai *hypervisor*. Seluruh komponen sistem dibangun di atas mesin virtual dan *container* yang dikelola oleh Proxmox VE.

B. Arsitektur Sistem

Arsitektur PoC terdiri dari beberapa komponen utama: (1) Mikrotik CHR sebagai router virtual untuk simulasi jaringan; (2) Nginx sebagai *reverse proxy* dan *load balancer* yang mendistribusikan trafik ke dua *node* aplikasi; (3) dua *container* Node.js (*webserver1* dan *webserver2*) sebagai lapisan aplikasi; (4) PostgreSQL dengan konfigurasi *Primary-Standby Streaming Replication* untuk redundansi data; (5) klaster Apache Kafka dengan dua *broker* untuk komunikasi *event-driven*; serta (6) Prometheus dan Grafana untuk observabilitas sistem. Topologi sistem secara umum ditunjukkan pada Gambar 1. Diagram Alir Sistem.



Gambar 1. Diagram Alir Sistem

C. Tahapan Implementasi

Implementasi dilakukan secara bertahap dimulai dari persiapan lingkungan virtualisasi Proxmox VE, konfigurasi jaringan internal menggunakan Mikrotik CHR, implementasi *reverse proxy* dan *load balancer* Nginx dengan algoritma *Round Robin*, *deployment* aplikasi Node.js pada dua *container*, konfigurasi PostgreSQL *Streaming Replication* dengan mode asinkron, implementasi klaster Kafka dengan dua *broker* dan faktor replikasi dua, hingga konfigurasi *stack observabilitas* Prometheus-Grafana dengan berbagai *exporter* (`node_exporter`, `postgres_exporter`, `kafka_exporter`, `kafka_jmx`).

D. Skenario Pengujian

Pengujian dirancang untuk mengevaluasi enam aspek utama: (1) distribusi trafik pada Nginx *Load Balancer* melalui verifikasi pola *Round Robin*; (2) fungsionalitas *web server* dalam memproses dan menyimpan data; (3) konsistensi PostgreSQL *Streaming Replication* melalui perbandingan data antar-*node*; (4) integritas alur *end-to-end* Kafka dari *Producer* hingga *Consumer* termasuk validasi idempotensi; (5) akurasi dan overhead sistem observabilitas Prometheus-Grafana; serta (6) ketahanan failover melalui simulasi kegagalan pada masing-masing lapisan arsitektur (aplikasi, basis data, dan messaging).

IV. HASIL DAN PEMBAHASAN

A. Pengujian *Load Balancer* Nginx

Pengujian distribusi trafik dilakukan menggunakan utilitas *curl* untuk mengirimkan permintaan HTTP secara berurutan ke Nginx. Hasil menunjukkan bahwa identitas *server* yang merespons berubah secara bergantian dengan pola seling-seling (*webserver1-webserver2*), mengonfirmasi bahwa algoritma *Round Robin* berfungsi efektif. Sebagaimana disajikan pada Tabel 1, sistem secara konsisten membagi beban trafik secara merata (rasio 50:50) dengan tingkat keberhasilan 100% tanpa satu pun *request* yang gagal. Hasil pengujian disajikan pada Tabel 1. Hasil Pengujian Distribusi Trafik Nginx *Load Balancer*.

Tabel 1. Hasil Pengujian Distribusi Trafik Nginx *Load Balancer*

Percobaan	**Total *Request***	***Node* 1**	***Node* 2**	**Keberhasilan**
1	4	2	2	100%
2	6	3	3	100%
3	10	5	5	100%

Evaluasi pada lapisan basis data memperlihatkan bahwa HAProxy berhasil mengelola koneksi ke PostgreSQL dalam mode *Active-Standby*. *Node Primary* berstatus *UP* dengan volume trafik yang terus meningkat, sementara *node Replica* menunjukkan statistik trafik nol. Kondisi ini bukan indikasi kegagalan, melainkan konfirmasi bahwa *Replica* diposisikan murni sebagai *failover* unit yang hanya menerima koneksi apabila *Primary* gagal.

B. Pengujian *Web Server*

Pengujian fungsionalitas *web server* memvalidasi bahwa layanan Node.js pada kedua *container* mampu menerima permintaan HTTP, memproses *input* pengguna, dan menyimpan data ke PostgreSQL secara andal. Data pesan yang dikirimkan melalui formulir *input* berhasil tersimpan dan ditampilkan kembali pada *dashboard* admin. Hasil ini mengonfirmasi dua aspek: konektivitas basis data berfungsi optimal pada kedua *node*, dan sistem menunjukkan konsistensi data yang tinggi dengan data tersentralisasi pada klaster PostgreSQL yang sama.

C. Pengujian PostgreSQL *Streaming Replication*

Pemeriksaan status konektivitas melalui *view* sistem `pg_stat_replication` menunjukkan *node Standby* terdeteksi dalam status *streaming* dengan `sync_state` bernilai *async*. Konfigurasi asinkron diterapkan untuk mengoptimalkan latensi penulisan, di mana *server Primary* dapat menyelesaikan transaksi tanpa menunggu konfirmasi dari *Standby*.

Validasi integritas dilakukan dengan membandingkan isi basis data pada kedua *server* setelah pengiriman pesan baru. Sebagaimana ditunjukkan pada Tabel 2, data yang baru masuk ke *database* utama terduplikasi secara identik di *database* replika dengan latensi mendekati real-time dan lonjakan CPU yang stabil di kisaran 2,32 milidetik. Hasil pengujian konsistensi dan performa PostgreSQL *Streaming Replication* disajikan pada Tabel 2. Hasil Pengujian PostgreSQL *Streaming Replication*.

Tabel 2. Hasil Pengujian PostgreSQL *Streaming Replication*

Percobaan	**Beban**	**Lonjakan CPU**	**Latensi**	**Integritas**
1	*Insert* pesan	2,32 ms	< 1 detik	100%
2	*Insert* pesan	2,35 ms	< 1 detik	100%
3	*Insert* pesan	2,30 ms	< 1 detik	100%
4	*Insert* pesan	2,33 ms	< 1 detik	100%
5	*Insert* pesan	2,31 ms	< 1 detik	100%
Rata-rata	2,32 ms		Real-time	Konsisten

D. Pengujian *Kafka End-to-End*

Verifikasi metadata topik menunjukkan konfigurasi HA yang berjalan normal dengan `PartitionCount: 1` untuk menjamin pengurutan total *(Total Ordering)* dan `ReplicationFactor: 2` untuk redundansi data. Kedua broker terdaftar lengkap dalam kolom *In-Sync Replicas* (ISR), membuktikan sinkronisasi berjalan optimal.

Pemantauan *log Producer* menunjukkan respons konsisten dengan *payload* JSON yang mencakup `message_uid`, `timestamp`, dan `hostname`. Pada sisi `Consumer`, `log` memperlihatkan siklus pemrosesan yang dimulai dengan penerimaan pesan dan diikuti penyimpanan ke PostgreSQL. Sistem juga berhasil menjalankan logika deduplikasi dengan mendeteksi dan melewati pesan dengan `message_uid` yang sudah diproses sebelumnya.

Selama pengujian teramati jeda penerimaan pesan oleh *Consumer* sekitar 30 detik akibat mekanisme *Consumer Group Rebalancing*. Seperti disajikan pada Tabel 3, meskipun

terjadi *transient lag* maksimal 2 pesan saat *rebalancing*, tidak ada data yang hilang *(Zero Data Loss)*. Seluruh pesan yang tertunda tetap tersimpan aman dalam antrean *broker* dan diproses seketika setelah klaster stabil. Hasil observasi dicatat pada Tabel 3. Hasil Pengujian Ketahanan *Failover* dan *Zero Data Loss* Apache Kafka.

Tabel 3. Hasil Pengujian Ketahanan Failover dan Zero Data Loss Apache Kafka

Percobaan	Pemicu	Lag Maks.	Waktu Pulih	*Data Loss*
1	Beban normal	0 pesan	0 detik	Nihil
2	Keterlambatan *heartbeat*	2 pesan	30 detik	Nihil
3	Keterlambatan *heartbeat*	2 pesan	28 detik	Nihil
4	Keterlambatan *heartbeat*	2 pesan	32 detik	Nihil

E. Pengujian Observabilitas

Verifikasi pada antarmuka Prometheus mengonfirmasi seluruh *exporter* (`node_exporter`, `postgres_exporter`, dan `kafka_exporter`, `kafka_jmx`) berstatus *UP* pada seluruh *node*. Durasi *scrape* terpantau stabil dalam orde milidetik dengan rata-rata di bawah 360 ms untuk JMX dan di bawah 50 ms untuk *exporter* lainnya, menunjukkan *overhead* pengumpulan data yang minimal.

Visualisasi Grafana pada *dashboard Node Exporter* merekam CPU *Busy* sebesar 27,8% dan *System Load* 49,5%. Kondisi *System Load* yang melampaui CPU menandakan karakteristik beban I/O *Bound*, di mana antrean sistem didominasi oleh aktivitas tunggu I/O jaringan saat aplikasi menunggu *acknowledgment* dari *broker* Kafka. Pada lapisan basis data, grafik CPU menunjukkan pola *sawtooth* yang merupakan refleksi normal dari beban kerja diskrit berbasis *event*, dengan CPU memiliki waktu istirahat yang cukup di antara transaksi.

F. Pengujian *Failover*

Simulasi kegagalan pada lapisan aplikasi dilakukan dengan mematikan *webserver1*. Nginx secara otomatis mendeteksi kegagalan dan mengalihkan seluruh trafik ke *webserver2* tanpa menghasilkan HTTP *error*, mencapai ketersediaan 100% *(zero downtime)*. Pengguna akhir tetap dapat mengirimkan data tanpa mengalami putus koneksi.

Pada lapisan basis data, sistem menerapkan proteksi *fail-safe* dengan menolak perintah penulisan saat *Primary* mati untuk mencegah *split-brain*. *Node Replica* tetap menyajikan data secara *read-only*. Saat *Primary* dihidupkan kembali, layanan basis data secara mandiri melakukan sinkronisasi ulang *(catch-up)* terhadap *log* yang tertinggal tanpa intervensi manual, dan sistem kembali menerima penulisan pesan baru.

Pada lapisan *messaging*, mematikan *broker* kafka1 tidak mengganggu integritas antrean pesan berkat `ReplicationFactor: 2`. Beban kerja dan kepemilikan partisi secara otomatis diambil alih oleh kafka2. Saat kafka1 dihidupkan kembali, *broker* berhasil melakukan *bootstrap* dan bergabung ke klaster dengan sinkronisasi *log* secara mandiri.

G. Analisis Efisiensi Sumber Daya

Evaluasi menyeluruh terhadap efisiensi sumber daya, sebagaimana dirangkum pada Tabel 4, menunjukkan bahwa seluruh komponen beroperasi di bawah ambang batas kritis (<70%). CPU *webserver* tercatat 27,8% dengan karakteristik I/O *Bound* yang wajar. Sistem pemantauan Prometheus terbukti ringan dengan *overhead* minimal, dan *Consumer Lag* Kafka selalu pulih ke angka nol setelah lonjakan *transient*. Pemetaan posisi setiap metrik pengujian terhadap rentang indikator efisiensi dirangkum pada Tabel 4. Evaluasi Efisiensi Sumber Daya.

Tabel 4. Evaluasi Efisiensi Sumber Daya

Komponen	Metrik	Nilai	Kategori
Webserver	CPU *Usage*	27,8%	Efisien
Webserver	*System Load*	49,5%	Efisien
PostgreSQL	CPU *Spike*	2,32 ms	Efisien
Prometheus	*Scrape Overhead*	< 50 ms	Efisien
Prometheus	*Scrape* JMX	< 360 ms	Efisien
Kafka	*Consumer Lag*	Maksimal 2 pesan	Efisien

V. SIMPULAN

Berdasarkan hasil implementasi dan pengujian, dapat disimpulkan bahwa arsitektur *High Availability* berbasis Proxmox VE *single-host* yang mengintegrasikan Nginx, Node.js, PostgreSQL, Kafka, dan Prometheus-Grafana terbukti dapat beroperasi stabil secara *end-to-end* pada perangkat keras berspesifikasi terbatas. Nginx andal mendistribusikan beban secara merata dengan keberhasilan 100% dan mencapai *seamless failover* saat *node* mengalami kegagalan. PostgreSQL *Streaming Replication* menjaga konsistensi data dengan latensi rata-rata 2,32 milidetik serta mencegah *split-brain* melalui mekanisme *fail-safe*. Klaster Kafka menjamin *Zero Data Loss* dengan pemulihan *lag* yang selalu kembali ke nol meskipun terjadi *rebalancing*. Seluruh arsitektur beroperasi efisien di bawah ambang batas kritis dengan *overhead monitoring* yang minimal. Untuk pengembangan selanjutnya, disarankan implementasi Proxmox *Multi-Node Cluster* dengan *Ceph Storage*, penggunaan Patroni atau Repmgr untuk *auto-promotion* PostgreSQL, peningkatan Kafka menjadi tiga *node* untuk quorum yang stabil, serta penerapan otomasi *deployment* menggunakan Ansible atau CI/CD *pipeline*.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Bapak Ardhi Wicaksono Santoso, S.Kom., M.Cs. selaku dosen pembimbing, serta jajaran pimpinan dan staf PT Global Intermedia Nusantara yang telah memberikan fasilitas dan dukungan bagi terlaksananya penelitian ini.

REFERENSI

[1] A. Idrus, "Designing High Availability Cluster Using Server Virtualization," International Journal of Information System & Technology, vol. 5, no. 2, pp. 93-98, 2021.

[2] A. Heryanto and A. Gunanta, "High Availability in Server Clusters by Using Backpropagation Neural Network Method," Journal of Technology and Open Source, vol. 4, no. 1, pp. 08-18, 2021.

[3] A. T. Irmawan, "Implementation of Proxmox Server Monitoring System," Journal of Mantik, vol. 7, no. 4, pp. 2685-4236, 2024.

[4] A. Isa, "Network Server Management Based on Virtualization Technology Using Proxmox at Diskominfo Bengkayang Regency," Journal of Community Service, vol. 5, no. 3, pp. 219-228, 2024.

[5] V. S. Ramdoss, "Optimizing System Performance: Load Balancers and High Availability," The Eastasouth Journal of Information System and Computer Science, vol. 1, no. 02, pp. 113-117, 2023.

[6] D. Kardha, A. R. Pamungkas, and H. Setiawan, "Pengembangan Virtual Server dengan Proxmox VE 6.2 sebagai Cloud Computing berbasis Free/Open Source Software," Scientific Journal of STMIK AUB, vol. 26, no. 1, pp. 1693, 2020.

[7] Shubneet, N. Chatterjee, A. R. Yadav, and N. S. Talwandi, "Temporal Resilience in Stream Processing: Mitigating Late Data and Lag in Apache Kafka 4.0," International Journal on Science and Technology, vol. 16, issue 2, pp. 2229-7677, 2025.

[8] V. K. Tambi, "Multi-Cloud Data Synchronization Using Kafka Stream Processing," The Research Journal, vol. 7, issue 6, pp. 2454-4930, 2021.

[9] T. P. Raptis, C. Cicconetti, and A. Passarella, "Efficient Topic Partitioning of Apache Kafka for High-Reliability Real-Time Data Streaming Applications," Journal of Future Generation Computer System, vol. 154, pp. 173-188, 2024.

[10] P. M. Dhulavvagol and S. G. Totad, "Performance Enhancement of Distributed System Using HDFS Federation and Sharding," Journal of Procedia Computer Science, vol. 218, pp. 2830-2841, 2023.