Journal of Applied Informatics and Computing (JAIC)
Vol.x, No.x, Juli 201x, pp. xx~xx
e-ISSN: 2548-6861

Implementasi Keamanan dan *High availability Database* PostgreSQL Berbasis Docker Melalui Replikasi dan Load Balancing

Feby Widialoka[1*], Nur Cahyono Kushardianto[2]**
**Rekayasa Keamanan Siber, Jurusan Teknik Informatika, Politeknik Negeri Batam*
***Teknik Komputer, Jurusan Teknik Informatika, Politeknik Negeri Batam*
[1]febywidialoka@gmail.com, [2] anung@polibatam.ac.id

Article Info

Article history:

Received …
Revised …
Accepted …

Keyword:

High availability, PostgreSQL, Docker Swarm, Streaming Replication, HAProxy, Keepalived, Wazuh.

ABSTRACT

Ketersediaan layanan basis data menjadi faktor penting dalam mendukung operasional sistem informasi, terutama pada lingkungan yang membutuhkan akses data secara terus-menerus. Gangguan pada server *database* dapat menyebabkan terhentinya layanan dan menghambat proses bisnis. Penelitian ini bertujuan untuk mengimplementasikan sistem *High availability* PostgreSQL berbasis Docker Swarm menggunakan mekanisme streaming replication, HAProxy, dan Keepalived guna meningkatkan ketersediaan layanan *database*. Selain itu, sistem dilengkapi dengan monitoring keamanan menggunakan Wazuh dan Suricata untuk mendeteksi aktivitas yang berpotensi mengganggu layanan. Metode penelitian yang digunakan adalah metode eksperimental dengan melakukan implementasi sistem dan serangkaian pengujian meliputi replikasi *database*, konsistensi data, failover, recovery node, serta monitoring keamanan. Hasil pengujian menunjukkan bahwa replikasi data berhasil menjaga sinkronisasi antara *node primary* dan replica secara *near real-time* menggunakan mekanisme *asynchronous replication*. Mekanisme failover berhasil mengalihkan Virtual IP (VIP) ke node cadangan dalam waktu 12 detik, sedangkan layanan *database* berhasil dipulihkan dalam waktu 197 detik. Pengujian keamanan juga menunjukkan bahwa Wazuh dan Suricata mampu mendeteksi aktivitas jaringan abnormal, perubahan konfigurasi user *database*, serta kesalahan eksekusi query. Berdasarkan hasil tersebut, sistem yang dibangun berhasil meningkatkan ketersediaan layanan *database* serta menyediakan monitoring keamanan yang mendukung keandalan sistem.

I. PENDAHULUAN

Perkembangan teknologi informasi telah mendorong perusahaan untuk semakin bergantung pada sistem informasi dalam mendukung proses bisnis yang kompleks dan terdistribusi. Pada sektor energi dan industri berat, sistem basis data berperan penting sebagai pusat penyimpanan, pengelolaan, dan pertukaran informasi yang digunakan dalam operasional perusahaan. Ketersediaan (*availability*) layanan basis data menjadi faktor yang sangat penting karena gangguan atau *downtime* dapat menghambat proses bisnis, menurunkan produktivitas, serta berpotensi menimbulkan kerugian operasional . Namun demikian, masih banyak sistem basis data yang bergantung pada satu server sehingga berisiko mengalami *single point of failure* ketika terjadi kegagalan perangkat keras, kesalahan konfigurasi, maupun gangguan lainnya [1].

Salah satu pendekatan yang dapat digunakan untuk meningkatkan ketersediaan layanan adalah penerapan *High availability* (HA). Konsep HA memungkinkan sistem tetap beroperasi meskipun terjadi kegagalan pada salah satu komponen melalui mekanisme replikasi data dan *failover* ke server cadangan [2]. Dalam lingkungan basis data, penerapan replikasi memungkinkan data pada server utama tersinkronisasi dengan server cadangan sehingga risiko kehilangan data dapat diminimalkan. Pada mekanisme *asynchronous replication*, sinkronisasi data dilakukan secara *near real-time* dengan jeda yang relatif kecil, meskipun masih

e-ISSN: 2548-6861

terdapat potensi kehilangan data dalam jumlah minimal apabila terjadi kegagalan mendadak pada server utama sebelum proses replikasi selesai dilakukan [3].

Seiring perkembangan teknologi containerization, Docker menjadi salah satu platform yang banyak digunakan untuk membangun layanan yang fleksibel dan mudah dikelola. Docker memungkinkan aplikasi dijalankan dalam lingkungan yang terisolasi sehingga mempermudah proses deployment, skalabilitas, serta pengelolaan infrastruktur dibandingkan pendekatan konvensional [4]. Pada sisi basis data, PostgreSQL merupakan salah satu sistem manajemen basis data relasional (Relational *Database* Management System) yang mendukung fitur streaming replication untuk membangun sistem dengan tingkat ketersediaan yang lebih tinggi [5].

Selain aspek ketersediaan layanan, keamanan juga menjadi faktor penting dalam pengelolaan sistem informasi. Dalam prinsip CIA Triad, availability merupakan salah satu komponen utama selain confidentiality dan integrity [6]. Oleh karena itu, penerapan sistem *High availability* perlu didukung dengan mekanisme monitoring keamanan untuk mendeteksi aktivitas yang berpotensi mengganggu operasional layanan. Wazuh merupakan platform Security Information and Event Management (SIEM) dan Host Intrusion Detection System (HIDS) yang mampu melakukan pengumpulan log, deteksi anomali, serta pemberian peringatan terhadap aktivitas yang mencurigakan pada sistem [7].

Penelitian mengenai teknologi container dan *High availability* telah banyak dilakukan untuk meningkatkan ketersediaan layanan dan efisiensi pengelolaan sistem. Penelitian oleh Aldiwidianto dan Prismana (2023) [8] membahas implementasi *High availability* berbasis Docker Swarm dengan memanfaatkan algoritma Raft untuk menjaga ketersediaan layanan pada lingkungan container. Hasil penelitian menunjukkan bahwa layanan tetap dapat berjalan meskipun salah satu node mengalami kegagalan karena Docker Swarm mampu melakukan penjadwalan ulang (rescheduling) service secara otomatis ke node yang masih aktif. Namun, penelitian tersebut lebih berfokus pada ketersediaan layanan container dan belum membahas implementasi replikasi *database* serta pengelolaan akses menggunakan Virtual IP.

Penelitian oleh Abdillah dan Prismana (2022) [9] mengkaji penerapan load balancing menggunakan HAProxy pada lingkungan server terdistribusi. Hasil penelitian menunjukkan bahwa HAProxy mampu mendistribusikan beban akses ke beberapa server backend sehingga meningkatkan performa dan menjaga stabilitas layanan. Selain itu, sistem tetap dapat melayani permintaan pengguna ketika salah satu server backend mengalami gangguan. Meskipun demikian, penelitian ini lebih berfokus pada aspek load balancing dan belum membahas integrasi dengan mekanisme *High availability database* maupun monitoring keamanan sistem.

Penelitian yang dilakukan oleh Prabowo et al. (2024) [10] berfokus pada keamanan lingkungan Docker melalui pengujian serangan Distributed Denial of Service (DDoS) dan Docker Daemon Attack. Hasil penelitian menunjukkan bahwa penerapan sistem monitoring mampu membantu administrator dalam mendeteksi aktivitas mencurigakan dan serangan yang terjadi pada lingkungan container. Namun, penelitian tersebut belum mengintegrasikan aspek keamanan dengan mekanisme *High availability* maupun replikasi *database*.

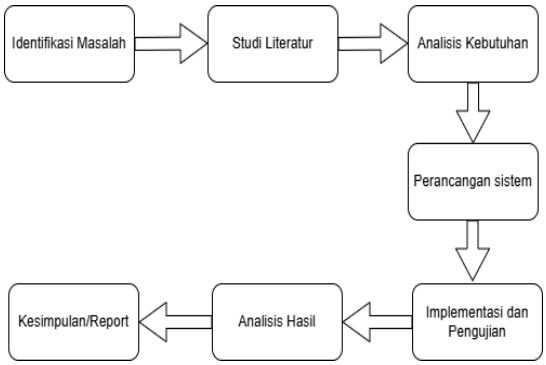
Penelitian oleh Mukti dan Nugroho (2023) [1] mengusulkan implementasi *High availability* server menggunakan metode load balancing berbasis klaster dengan memanfaatkan HAProxy sebagai load balancer dan Galera sebagai mekanisme sinkronisasi *database*. Hasil penelitian menunjukkan bahwa mekanisme failover dan load balancing mampu menjaga ketersediaan layanan ketika salah satu node *database* mengalami kegagalan. Selain itu, sinkronisasi data antar node dapat berjalan dengan baik sehingga konsistensi data tetap terjaga. HAProxy juga mampu mengalihkan koneksi secara otomatis ke node yang masih aktif ketika terjadi gangguan pada server utama. Penelitian tersebut juga menunjukkan peningkatan performa akses data sebesar 196,38% dan penurunan waktu eksekusi transaksi sebesar 66,04% dibandingkan server tunggal. Namun, penelitian ini belum membahas penerapan containerisasi maupun monitoring keamanan secara terintegrasi.

Berdasarkan penelitian-penelitian sebelumnya, masih terdapat keterbatasan pada integrasi antara *High availability* basis data, *load balancing*, dan monitoring keamanan dalam satu lingkungan terkontainerisasi. Oleh karena itu, penelitian ini mengimplementasikan sistem *High availability* PostgreSQL berbasis Docker Swarm menggunakan mekanisme *streaming replication* dan HAProxy untuk mendukung ketersediaan layanan, serta mengintegrasikan Wazuh sebagai solusi monitoring keamanan. Penelitian ini bertujuan untuk mengevaluasi kemampuan sistem dalam menjaga ketersediaan layanan basis data serta mendeteksi aktivitas yang berpotensi menimbulkan risiko terhadap operasional sistem.

II. METODE

A. Metode Penelitian

Penelitian ini menggunakan metode eksperimental untuk mengevaluasi implementasi sistem *High availability* PostgreSQL berbasis Docker Swarm [11]. Metode ini dilakukan dengan membangun lingkungan pengujian yang terdiri dari dua node *database* PostgreSQL, HAProxy sebagai load balancer, Keepalived sebagai penyedia Virtual IP (VIP) [12], serta Wazuh dan Suricata sebagai sistem monitoring keamanan [7]. Tahapan penelitian meliputi identifikasi masalah, studi literatur, analisis kebutuhan, perancangan sistem, implementasi dan pengujian, analisis hasil, serta penyusunan kesimpulan. Alur penelitian ditunjukkan pada Gambar 1.



Gambar 1. Desain Eksperimental

B. Analisis Kebutuhan Sistem

a) Kebutuhan Fungsional

Kebutuhan fungsional menjelaskan kemampuan yang harus dimiliki sistem agar dapat memenuhi tujuan penelitian. Pada penelitian ini, yang dibangun antara lain:

1. Menyediakan layanan *database* PostgreSQL yang dapat diakses melalui satu alamat Virtual IP (VIP).
2. Melakukan sinkronisasi data secara otomatis antara *node primary* dan *node replica* menggunakan mekanisme *streaming replication*.
3. Melakukan failover ketika *node primary* mengalami kegagalan sehingga layanan dapat dialihkan ke node cadangan.
4. Mengembalikan node yang gagal ke dalam sistem replikasi setelah proses recovery selesai dilakukan.

b) Kebutuhan Non-Fungsional

Kebutuhan non-fungsional meliputi kebutuhan perangkat keras, kebutuhan perangkat lunak dan kebutuhan server virtual machine masing-masing ditampilkan pada Tabel I dan Tabel II.

Tabel I. Kebutuhan Perangkat Keras

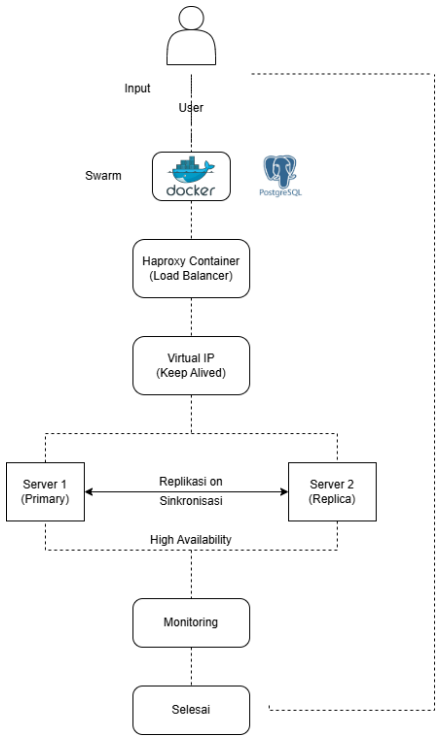
Komponen	Spesifikasi
Prosesor	Intel® Core™ i5-6200U CPU @ 2.30 GHz
Memori (RAM)	12 GB
Storage	500 GB HDD
Hypervisor	VMware Workstation

Tabel II. Kebutuhan Perangkat Lunak

Komponen	Versi
VMware Workstation	17.
Ubuntu Server	22.04 LTS
Docker Engine	28.1.1
PostgreSQL	16
HAProxy	2.0.33
Keepalived	V2.0.19
Wazuh	4.
DBeaver	28

C. Perancangan Arsitektur Sistem

Arsitektur sistem terdiri dari dua node PostgreSQL yang dikonfigurasi menggunakan mekanisme *streaming replication* [2]. HAProxy digunakan sebagai pengarah koneksi menuju node aktif, sedangkan *Keepalived* menyediakan Virtual IP (VIP) sehingga client dapat mengakses *database* melalui satu alamat IP yang tetap [12]. Untuk mendukung monitoring keamanan, Wazuh digunakan sebagai platform SIEM yang terintegrasi dengan Suricata sebagai *Intrusion Detection System* (IDS) [7]. Implementasi sistem dilakukan menggunakan beberapa *virtual machine* sehingga setiap node *database* dapat berjalan secara independen dan berkomunikasi melalui jaringan virtual. Pendekatan ini dipilih untuk mensimulasikan kondisi implementasi yang lebih mendekati lingkungan nyata, sehingga mekanisme *streaming replication*, *failover*, dan perpindahan *Virtual IP* (VIP) dapat dievaluasi secara lebih realistis dibandingkan apabila seluruh layanan dijalankan pada satu host. Arsitektur sistem ditunjukkan pada Gambar 2.



Gambar 2. Perancangan Arsitektur

Tabel III. Deskripsi Komponen Sistem

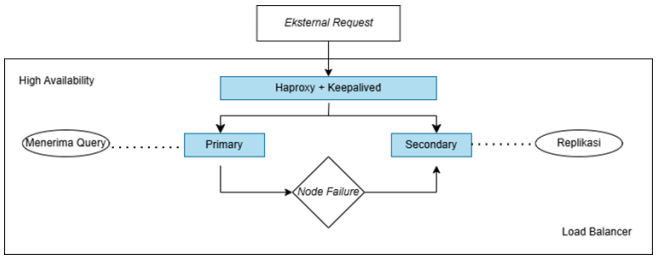
Komponen	Fungsi
User	Mengakses layanan *database*
HAProxy	Mengelola koneksi client
Keepalived	Menyediakan Virtual IP

e-ISSN: 2548-6861

PostgreSQL Primary	Menerima transaksi
PostgreSQL Replica	Menyimpan salinan data
Wazuh	Monitoring keamanan

D. Skenario Pengujian

Pengujian dilakukan untuk mengevaluasi aspek *high availability*, load balancing, dan monitoring keamanan. Skenario pengujian *high availability* dan load balancing ditunjukkan pada Gambar 3.



Gambar 3. Pengujian *High availability* dan Load Balancing

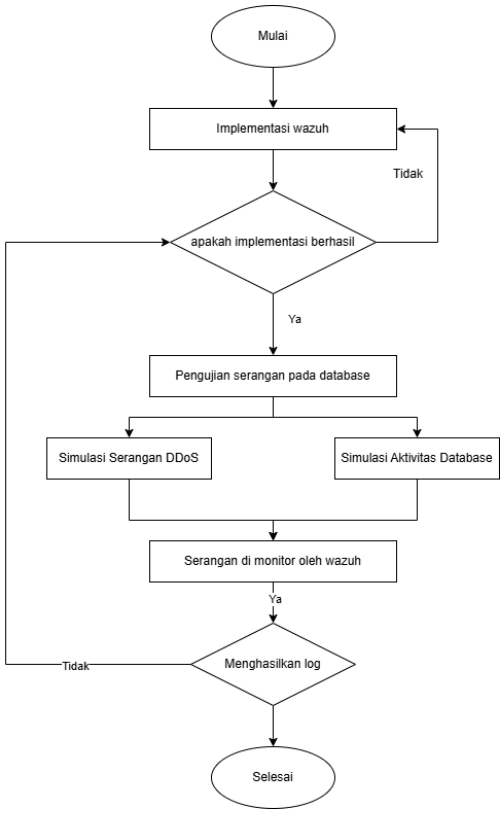
Parameter pengujian meliputi replikasi *database*, konsistensi data, failover, recovery node, recovery time objective (RTO), pengalihan koneksi, dan Waktu Eksekusi Query [13]. Indikator keberhasilan pengujian ditunjukkan pada Tabel IV.

Tabel IV. Indikator Keberhasilan Pengujian

Aspek	Parameter	Indikator
High availability	Replikasi *Database*	Data tersinkronisasi
High availability	Konsistensi Data	Tidak ada data hilang
High availability	Failover	VIP berpindah dan layanan pulih
High availability	Recovery Node	Node kembali bergabung
High availability	RTO	≤ 30 menit
Performa Sistem	Waktu Eksekusi Query	Waktu eksekusi relatif stabil

E. Pengujian Keamanan

Pengujian keamanan dilakukan menggunakan Wazuh dan Suricata untuk memonitor aktivitas sistem serta mendeteksi aktivitas yang berpotensi mengganggu layanan *database* [7]. Pengujian dilakukan melalui simulasi trafik abnormal menggunakan metode SYN Flood (DDoS) dan aktivitas *database* yang menghasilkan alert keamanan [10]. Alur pengujian keamanan ditunjukkan pada Gambar 4.



Gambar 4. Flowchart Skenario Pengujian Keamanan

Keberhasilan pengujian keamanan dievaluasi berdasarkan kemampuan sistem dalam mendeteksi aktivitas mencurigakan, menghasilkan alert keamanan, dan mencatat log aktivitas yang terjadi pada sistem.

Tabel V. Indikator Pengujian Keamanan

Komponen	Fungsi
Security Monitoring	Deteksi Aktivitas
Security Monitoring	Alert
Security Monitoring	Log Activity

III. HASIL DAN PEMBAHASAN

A. Implementasi Sistem

Pada tahap implementasi, sistem *High availability* PostgreSQL dibangun menggunakan Docker Swarm yang terdiri dari dua node *database*, yaitu *node primary* dan *node replica*. Mekanisme streaming replication digunakan untuk melakukan sinkronisasi data secara *near real-time* menggunakan mode *asynchronous replication*, sedangkan HAProxy dan Keepalived digunakan untuk menjaga ketersediaan layanan melalui pengalihan koneksi dan penyediaan Virtual IP (VIP). Selain itu, Wazuh diimplementasikan sebagai sistem monitoring keamanan untuk memantau aktivitas *database* dan layanan container.

1. Implementasi PostgreSQL Primary-Replica

Implementasi *database* dilakukan menggunakan PostgreSQL versi 16 yang dijalankan pada lingkungan Docker. Sistem terdiri dari dua node *database*, yaitu Server 1 sebagai *primary database* dan Server 2 sebagai *Replica Database*. Replikasi data diterapkan menggunakan mekanisme streaming replication dengan mode *asynchronous replication* sehingga perubahan data pada *node primary* dapat disinkronisasikan ke *node replica* dengan jeda yang relatif kecil.

Mode *asynchronous replication* dipilih karena mampu memberikan performa penulisan (*write*) yang lebih baik dibandingkan *synchronous replication*. Pada mekanisme ini, *node primary* tidak perlu menunggu konfirmasi dari *node replica* sebelum transaksi dinyatakan berhasil sehingga latensi transaksi menjadi lebih rendah. Namun, konsekuensinya adalah terdapat potensi kehilangan sebagian transaksi terakhir apabila *node primary* mengalami kegagalan secara mendadak sebelum proses replikasi selesai. Oleh karena itu, sinkronisasi data pada penelitian ini dikategorikan sebagai *near real-time*.

Untuk memastikan status masing-masing node, dilakukan pengujian menggunakan fungsi *pg_is_in_recovery()*. Hasil pengujian menunjukkan bahwa Server 1 memiliki nilai *false* yang menandakan node berada pada kondisi *primary*, sedangkan Server 2 memiliki nilai *true* yang menandakan node berfungsi sebagai replica atau *standby* server.

```
feby@feby1:~$ docker exec -it postgres-primary.1.tp4hrjfffufnb661wgz2z8wsw bash
root@a9b94280a8da:/# psql -U postgres
psql (16.14 (Debian 16.14-1.pgdg13+1))
Type "help" for help.

postgres=# select pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 f
(1 row)

postgres=#
```

Gambar 5. Status Primary *Database*

```
feby@feby2:~$ docker exec -it postgres-replica psql -U postgres
psql (16.14 (Debian 16.14-1.pgdg13+1))
Type "help" for help.

postgres=# SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 t
(1 row)

postgres=#
```

Gambar 6. Status *Replica Database*

2. Implementasi Streaming Replication

Streaming replication digunakan untuk melakukan sinkronisasi data dari *node primary* ke *node replica* secara *near real-time* menggunakan mekanisme *asynchronous replication*, dengan memastikan data tetap tersedia pada node cadangan ketika terjadi kegagalan pada node utama.

```
feby@feby1:~$ docker exec -it postgres-primary psql -U postgres
psql (16.14 (Debian 16.14-1.pgdg13+1))
Type "help" for help.

postgres=# SELECT client_addr, state, sync_state FROM pg_stat_replication;
   client_addr    |   state   | sync_state
------------------+-----------+------------
 192.168.118.143 | streaming | async
(1 row)
```

Gambar 7. Sinkronisasi data replica

3. Implementasi Docker Swarm

Implementasi sistem dilakukan menggunakan Docker Swarm sebagai platform orkestrasi container. Docker Swarm digunakan untuk mengelola layanan yang berjalan pada beberapa host secara terdistribusi dan menyediakan komunikasi antar container melalui jaringan overlay network.

Penggunaan Docker Swarm memudahkan proses deployment, pengelolaan layanan, serta mendukung penerapan arsitektur *high availability* yang digunakan pada penelitian ini. Setiap layanan yang berjalan pada sistem dapat saling berkomunikasi melalui jaringan yang sama meskipun berada pada host yang berbeda.

```
feby@feby1:~$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
1jbitl4jhfyrafvvgs06s1tor *   feby1      Ready     Active         Leader           28.1.1
qxo9fzozch5m6evs1797n6aob     feby2      Ready     Active                          28.1.1
feby@feby1:~$
```

Gambar 8. Status server 1 & server 2 aktif

4. Implementasi HAProxy

HAProxy diimplementasikan sebagai load balancer dan pengarah koneksi menuju node *database* yang aktif. Seluruh koneksi client diarahkan melalui satu *endpoint* sehingga pengguna tidak perlu mengetahui lokasi server *database* yang sedang aktif.

Selain berfungsi sebagai load balancer, HAProxy juga digunakan untuk melakukan health check terhadap layanan *database* guna memastikan hanya node yang tersedia yang menerima koneksi dari client.

```
feby@feby1:~$ systemctl status haproxy.service
● haproxy.service - HAProxy Load Balancer
     Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2026-06-08 15:57:57 UTC; 1h 23min ago
       Docs: man:haproxy(1)
             file:/usr/share/doc/haproxy/configuration.txt.gz
    Process: 1139 ExecStartPre=/usr/sbin/haproxy -Ws -f $CONFIG -c -q $EXTRAOPTS (code=exited, status=0/SUCCESS)
   Main PID: 1204 (haproxy)
      Tasks: 3 (limit: 4550)
     Memory: 37.3M
     CGroup: /system.slice/haproxy.service
             ├─1204 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -S /run/haproxy-master.sock
             └─1208 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -S /run/haproxy-master.sock
```

Gambar 9. Status HAProxy

5. Implementasi Keepalived

Keepalived digunakan untuk menyediakan Virtual IP (VIP) sehingga client tetap menggunakan satu alamat IP meskipun terjadi perpindahan node aktif [12].

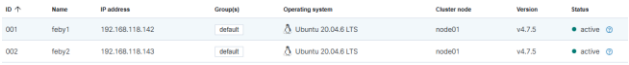
```
       valid_lft forever preferred_lft forever
feby@feby1:~$ ip -br a
lo               UNKNOWN        127.0.0.1/8 ::1/128
ens33            UP             192.168.118.142/24 metric 100 192.168.118.145/32 fe80::20c:29ff:feda:7b2b/64
```

Gambar 10. Virtual IP pada Server Primary

e-ISSN: 2548-6861

6. Implementasi Monitoring Wazuh

Wazuh digunakan untuk melakukan monitoring keamanan terhadap aktivitas sistem dan layanan *database* PostgreSQL.



Gambar 11. *Database* server 1 & 2 yang termonitoring.

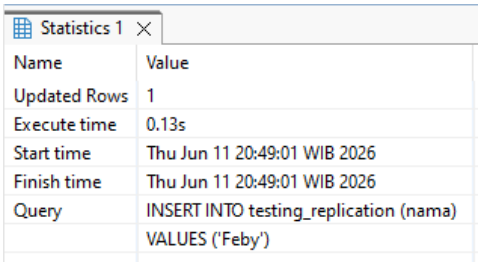
B. Pengujian dan Analisis Sistem

1. Pengujian Replikasi *Database*

Pengujian replikasi *database* dilakukan dengan menambahkan data pada tabel testing_replication di *node primary* menggunakan perintah INSERT. Hasil pengujian menunjukkan bahwa data berhasil direplikasi secara otomatis ke *node replica* tanpa memerlukan sinkronisasi tambahan. Hasil query SELECT pada kedua node menunjukkan data yang identik, sehingga mekanisme streaming replication berhasil menjaga konsistensi data antar node dan mendukung kebutuhan *High availability* pada sistem yang dibangun [14].

Hasil tersebut menunjukkan bahwa seluruh data pada skenario pengujian berhasil direplikasi ke node *replica*. Namun, karena penelitian ini menggunakan *asynchronous replication*, hasil tersebut diperoleh pada kondisi ketika proses replikasi telah selesai. Apabila node *primary* mengalami kegagalan sebelum proses replikasi selesai, terdapat potensi sebagian transaksi terakhir belum diterapkan pada node *replica*.

Selain itu, hasil pengujian ini diperoleh pada lingkungan virtual dengan konfigurasi perangkat keras dan jaringan tertentu. Oleh karena itu, waktu sinkronisasi maupun performa replikasi dapat berbeda apabila diterapkan pada lingkungan yang memiliki spesifikasi atau kondisi jaringan yang berbeda. Meskipun demikian, selama arsitektur sistem dan mekanisme komunikasi yang digunakan tetap sama, mekanisme replikasi akan bekerja sesuai dengan rancangan sistem yang diterapkan pada penelitian ini.



Gambar 12. Buat table uji di DBeaver

Gambar 13. Sinkronisasi data ke replica

Tabel VI. Menunjukkan hasil pengujian replikasi *database* yang dilakukan pada sistem

Pengujian	Hasil
Pembuatan tabel pada dbeaver	Berhasil
Insert data pada primary	Berhasil
Sinkronisasi data ke replica	Berhasil
Status replikasi	Aktif

2. Pengujian Konsistensi Data

Pengujian konsistensi data dilakukan dengan membandingkan hasil *query SELECT* pada *node primary* dan *replica* setelah proses replikasi berlangsung. Hasil pengujian menunjukkan bahwa jumlah *record* dan isi data pada kedua node identik tanpa ditemukan kehilangan maupun duplikasi data. Hasil tersebut menunjukkan bahwa mekanisme replikasi mampu menjaga integritas dan konsistensi data sehingga layanan tetap dapat berjalan dengan data yang sama ketika terjadi perpindahan ke *node replica*.

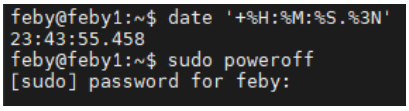
3. Pengujian Failover

Pengujian *failover* dilakukan dengan mematikan *node primary* (Server 1) menggunakan perintah *poweroff*. Hasil pengujian menunjukkan bahwa Keepalived pada Server 2 berhasil mendeteksi kegagalan node utama dan secara otomatis mengambil alih Virtual IP 192.168.118.145. Server 2 memasuki status *MASTER* pada pukul 23:44:07 setelah Server 1 dimatikan pada pukul 23:43:55, sehingga waktu perpindahan layanan (*failover time*) yang diperoleh adalah 12 detik.

Hasil tersebut menunjukkan bahwa mekanisme *failover* berjalan sesuai dengan rancangan sistem. Waktu *failover* sebesar 12 detik masih berada jauh di bawah target *Recovery Time Objective* (RTO) pada skenario pengujian, yaitu 30 menit. Apabila dikaitkan dengan target SLA 99,99% yang dihitung dalam periode satu bulan, waktu gangguan selama 12 detik masih berada dalam batas toleransi waktu henti layanan. Namun, nilai tersebut bukan merupakan waktu *failover* tercepat yang dapat dicapai oleh sistem. Pada penelitian ini, konfigurasi parameter deteksi kegagalan dan proses perpindahan layanan belum dioptimalkan sepenuhnya, serta masih terdapat tahapan yang dilakukan secara manual. Kondisi tersebut dapat menambah waktu sebelum layanan kembali aktif.

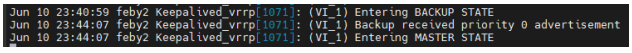
Selain itu, penggunaan beberapa *virtual machine* menyebabkan komunikasi antar node berlangsung melalui jaringan virtual. Akan tetapi, hal tersebut tidak dapat dinyatakan sebagai satu-satunya penyebab waktu *failover* sebesar 12 detik karena pengaruh *network hop* tidak diukur secara khusus dalam penelitian ini. Waktu *failover* masih dapat dioptimalkan melalui penyesuaian interval pemeriksaan layanan, parameter VRRP pada Keepalived, otomatisasi proses promosi node *replica*, dan pengurangan tahapan manual. Dengan demikian, hasil 12 detik merupakan waktu yang diperoleh berdasarkan konfigurasi dan skenario pengujian yang digunakan, bukan batas minimum kemampuan sistem.

Perpindahan *Virtual IP* memungkinkan klien tetap mengakses layanan database menggunakan alamat IP yang sama tanpa perlu melakukan perubahan konfigurasi koneksi. Dengan demikian, implementasi Keepalived berhasil mendukung ketersediaan layanan database pada lingkungan *High Availability* yang dibangun.



Gambar 14. Proses simulasi kegagalan pada *node primary* dengan mematikan Server 1 menggunakan perintah *poweroff*

Gambar 15. Virtual IP berhasil berpindah ke Server 2 setelah Server 1 mengalami kegagalan



Gambar 16. Keepalived mendeteksi kegagalan *node primary* dan mengubah status Server 2 menjadi MASTER sehingga Virtual IP dapat diambil alih

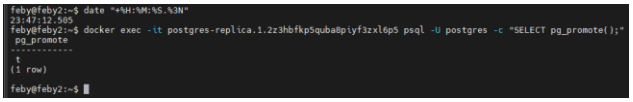
Tabel VII. Hasil Pengujian *Failover*

Pengujian	Hasil
Waktu Primary Down	23:43:55
Waktu Server 2 Menjadi MASTER	23:44:07
Waktu Perpindahan VIP	12 detik
Status Perpindahan VIP	Berhasil

4. Pengujian *recovery node*

Pengujian *recovery node* dilakukan dengan mengaktifkan kembali Server 1 setelah proses *failover* berhasil dijalankan. Server 1 kemudian dikonfigurasi ulang sebagai *node replica* dan dihubungkan kembali ke Server 2 yang telah berperan sebagai *primary*. Hasil pengujian menunjukkan bahwa Server 1 berhasil bergabung kembali ke dalam sistem replikasi dan menerima sinkronisasi data dari *node primary* tanpa mengalami kehilangan data.

Keberhasilan proses *rejoin* menunjukkan bahwa mekanisme *High availability* yang diterapkan tidak hanya mampu mempertahankan layanan saat terjadi kegagalan, tetapi juga memungkinkan node yang gagal untuk kembali beroperasi sebagai bagian dari sistem replikasi. Dengan demikian, ketersediaan layanan dan redundansi data tetap terjaga setelah proses pemulihan sistem.



Gambar 17. Proses replica dipromosikan menjadi *primary*

Tabel VIII. Hasil Recovery Node

Pengujian	Hasil
Recovery time	3 menit 17 detik
Recovery time dalam detik	197 detik
Batas toleransi pemulihan	30 menit / 1800 detik

5. Pengujian Availability

Pada penelitian ini, target *Recovery Time Objective* (RTO) ditetapkan maksimal 30 menit sebagai batas toleransi waktu pemulihan layanan. Hasil pengujian menunjukkan bahwa Server 1 mengalami kegagalan pada pukul 23:43:55. Setelah proses *failover* dan promosi *node replica* menjadi *primary*, layanan *database* kembali aktif pada pukul 23:47:12. Dengan demikian, waktu pemulihan layanan yang diperoleh adalah 3 menit 17 detik atau 197 detik.

Berdasarkan hasil tersebut, waktu pemulihan yang dicapai berada di bawah target RTO yang telah ditetapkan, yaitu 30 menit. Hal ini menunjukkan bahwa mekanisme *High availability* yang diterapkan mampu mempertahankan ketersediaan layanan *database* dan memulihkan layanan dalam waktu yang dapat diterima ketika terjadi kegagalan pada node utama.

Tabel IX. Hasil Availability

Parameter	Nilai
Total Time	1800 detik (30 menit)
Downtime	197 detik
Availability	89,05%
Status Pemulihan	Berhasil

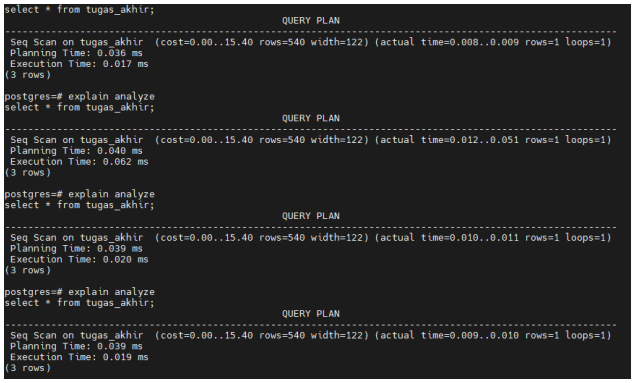
6. Pengujian Waktu Eksekusi *Query*

Pengujian waktu eksekusi query dilakukan menggunakan perintah *EXPLAIN ANALYZE* pada query *SELECT* terhadap

e-ISSN: 2548-6861

tabel tugas_akhir sebanyak empat kali percobaan. Hasil pengujian menunjukkan waktu eksekusi berturut-turut sebesar 0,017 ms, 0,062 ms, 0,020 ms, dan 0,019 ms dengan rata-rata waktu eksekusi sebesar 0,030 ms.

Perbedaan waktu eksekusi antar percobaan relatif kecil sehingga menunjukkan hasil yang konsisten selama pengujian. Seluruh hasil pengujian berada di bawah 1 ms yang menunjukkan bahwa PostgreSQL mampu memproses query dengan cepat pada lingkungan yang telah menerapkan mekanisme *High availability*. Pengujian ini mengukur waktu eksekusi query pada sisi *database* menggunakan EXPLAIN ANALYZE dan tidak mencerminkan waktu respons end-to-end yang dialami client melalui jaringan maupun HAProxy.



Gambar 18. Hasil Pengujian Waktu Eksekusi Query Menggunakan *Explain Analyze*

Tabel X. Waktu Eksekusi Query

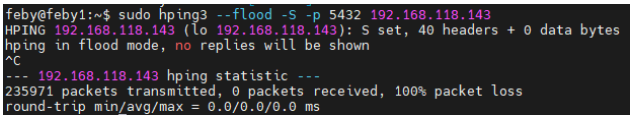
Percobaan	Waktu Eksekusi Query (ms)
1	0.017
2	0.062
3	0.020
4	0.019

7. Pengujian Keamanan Sistem

Pengujian keamanan dilakukan untuk mengevaluasi kemampuan sistem monitoring berbasis Wazuh dalam mendeteksi, mencatat, dan menghasilkan alert terhadap aktivitas yang berpotensi mengganggu layanan *database* PostgreSQL. Pengujian dilakukan melalui simulasi aktivitas jaringan abnormal menggunakan Suricata serta aktivitas mencurigakan pada *database* PostgreSQL yang dipantau oleh Wazuh.

A. Pengujian DDoS

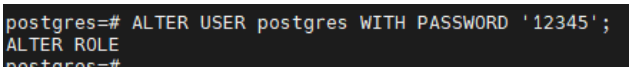
Pengujian DDoS dilakukan menggunakan *hping3* dengan metode *SYN Flood* yang ditujukan ke layanan PostgreSQL pada port 5432. Hasil pengujian menunjukkan bahwa Suricata berhasil mendeteksi aktivitas jaringan abnormal dan menghasilkan alert seperti *SURICATA STREAM SHUTDOWN RST invalid ack* dan *SURICATA STREAM Packet with invalid ack*. Alert tersebut berhasil diteruskan dan ditampilkan pada Wazuh Dashboard. Hasil ini menunjukkan bahwa integrasi Suricata dan Wazuh mampu mendeteksi serta mencatat aktivitas jaringan yang berpotensi mengganggu ketersediaan layanan *database*.



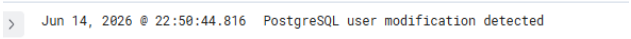
Gambar 19. Simulasi menggunakan hping3

B. Pengujian Modifikasi User *Database*

Pengujian dilakukan dengan melakukan perubahan password pada user PostgreSQL. Hasil pengujian menunjukkan bahwa aktivitas tersebut berhasil dideteksi oleh Wazuh dan menghasilkan alert *PostgreSQL user modification detected*. Alert yang muncul menunjukkan bahwa sistem monitoring mampu mengidentifikasi perubahan konfigurasi akun *database* yang termasuk aktivitas sensitif dari sisi keamanan.



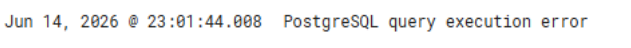
Gambar 20. Mengubah password



Gambar 21. Alert masuk ke wazuh

C. Pengujian Query Error

Pengujian dilakukan dengan menjalankan *query* yang tidak valid sehingga menghasilkan kesalahan eksekusi pada PostgreSQL. Hasil pengujian menunjukkan bahwa Wazuh berhasil mendeteksi aktivitas tersebut dan menghasilkan alert *PostgreSQL query execution error*. Alert yang dihasilkan membuktikan bahwa sistem monitoring mampu mencatat aktivitas kesalahan pada *database* yang dapat digunakan sebagai bahan analisis dan investigasi oleh administrator sistem.



Gambar 22. Menampilkan log error

Berdasarkan seluruh skenario pengujian keamanan yang dilakukan, Wazuh berhasil mendeteksi dan mencatat aktivitas yang berkaitan dengan anomali jaringan, perubahan

konfigurasi akun *database*, serta kesalahan eksekusi query. Hasil tersebut menunjukkan bahwa sistem monitoring yang diterapkan mampu memenuhi indikator pengujian keamanan berupa deteksi aktivitas, pembentukan alert, dan pencatatan log keamanan pada lingkungan PostgreSQL berbasis Docker.

IV. KESIMPULAN

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, sistem *High availability* PostgreSQL berbasis Docker Swarm berhasil dibangun menggunakan mekanisme *streaming replication*, HAProxy, dan Keepalived. Hasil pengujian menunjukkan bahwa replikasi *database* mampu menjaga konsistensi data antara *node primary* dan *replica*, serta mekanisme *failover* berhasil mengalihkan layanan ke node cadangan ketika terjadi kegagalan pada node utama. Perpindahan *Virtual IP* (VIP) berhasil dilakukan dalam waktu 12 detik dan layanan *database* dapat dipulihkan dalam waktu 3 menit 17 detik, sehingga memenuhi target *Recovery Time Objective* (RTO) yang telah ditetapkan.

Dari aspek performa, sistem menunjukkan waktu eksekusi *query* yang relatif stabil dengan rata-rata sebesar 0,030 ms. Selain itu, implementasi Wazuh berhasil mendeteksi dan mencatat berbagai aktivitas keamanan, termasuk anomali jaringan, modifikasi pengguna *database*, dan kesalahan eksekusi *query*. Secara keseluruhan, sistem yang dibangun berhasil memenuhi tujuan penelitian dalam meningkatkan ketersediaan layanan *database*, menjaga konsistensi data, serta menyediakan monitoring keamanan untuk mendukung keandalan operasional sistem.

Penelitian ini masih memiliki keterbatasan karena menggunakan mekanisme *asynchronous streaming replication*, sehingga masih terdapat potensi sebagian transaksi terakhir belum direplikasi apabila node *primary* mengalami kegagalan secara mendadak sebelum proses sinkronisasi selesai. Selain itu, hasil pengujian diperoleh pada lingkungan virtual dengan konfigurasi perangkat keras dan jaringan tertentu, sehingga waktu *failover* maupun performa sistem dapat berbeda apabila diterapkan pada lingkungan yang berbeda. Oleh karena itu, penelitian selanjutnya dapat mengembangkan implementasi menggunakan *synchronous replication* atau melakukan optimasi konfigurasi sistem untuk memperoleh waktu *failover* yang lebih cepat dan meningkatkan keandalan layanan database.

DAFTAR PUSTAKA

[1] Mukti, F. S., & Nugroho, F. E. (2023). Upaya Pencapaian Status *High availability* Server Menggunakan Metode Load Balancing Berbasis Klaster pada *Database* Server PT. XYZ.

[2] Nahak, Y. J. S., & Dwi Purnomo, H. (2023). Perancangan Sistem Replikasi Dan Sistem Backup *Database* Postgresql Menggunakan Repmgr Dan Barman. Progresif: Jurnal Ilmiah Komputer, 19(2), 867. https://doi.org/10.35889/progresif.v19i2.1319

[3] Putra, M. A. A., Fitri, I., & Iskandar, A. (2020). Implementasi *High availability* Cluster Web Server Menggunakan Virtualisasi Container Docker. JURNAL MEDIA INFORMATIKA BUDIDARMA, 4, 9. https://doi.org/10.30865/mib.v4i1.1729

[4] Khaeruddin, Alamin, Z., Arisandi, & Cholis, M. N. (2025). Implementasi Algoritma Raft untuk Menjamin *High availability* pada Docker Swarm Clustering. Jurnal Pengembangan Sains dan Teknologi, 1(2), 101–115. https://doi.org/10.63866/jpst.v1i2.79

[5] Praba, A. D., & Safitri, M. (2020). STUDI PERBANDINGAN PERFORMANSI ANTARA MYSQL DAN POSTGRESQL. Jurnal Khatulistiwa Informatika, 8(2). https://doi.org/10.31294/jki.v8i2.8851

[6] Vansuri, R., Fauzi, A., Teguh Prasetyo, E., Negara, R., Ramadhan, R., Mada Restu, A., & Raffi Firmansyah, R. (2023). Peran CIA (Confidentiality, Integrity, Availability) Terhadap Manajemen Keamanan Informasi. Jurnal Ilmu Multidisplin, 2, 106–113. https://doi.org/10.38035/jim.v2i1.234

[7] Pratama, M. D., Nova, F., & Prayama, D. (2022). Wazuh sebagai Log Event Management dan Deteksi Celah Keamanan pada Server dari Serangan Dos.

[8] Aldiwidianto, W., & Prismana, I. G. L. P. E. (2023). Analisis Perbandingan High Availibility Pada Web Server Menggunakan Orchestration Tool Kubernetes Dan Docker Swarm. Journal of Informatics and Computer Science (JINACS), 5(02), 138–148. https://doi.org/10.26740/jinacs.v5n02.p138-148

[9] Abdillah, T., & Prismana, I. G. L. P. E. (2022). Analisis Performansi Web Server Menggunakan Load Balancing pada Virtualisasi Docker Container. *Journal of Informatics and Computer Science (JINACS)*, *3*(04), 526–533. https://doi.org/10.26740/jinacs.v3n04.p526-533

[10] Prabowo, C. R., Rohadi, E., & Irmanto, D. (2024). Study and Analysis of Docker Internal System Security From Docker Daemon Attack and DDOS on The Open Journal System. JPUA: Jurnal Perpustakaan Universitas Airlangga: Media Informasi dan Komunikasi Kepustakawanan, 14, 61–68. https://doi.org/10.20473/jpua.v14i1.2024.61-68

[11] Afrizal, H., & Prihanto, A. (2022). Analisis Kebutuhan Resource Dan Independensi Antara Teknologi Single Server, Virtualisasi Dan Container. Journal of Informatics and Computer Science (JINACS), 4(01), 26–33. https://doi.org/10.26740/jinacs.v4n01.p26-33

[12] Adhiwibowo, W., & Daru, A. F. (2020). *High availability* Web and *Database* Service Model with Redundancy Servers. AITI: Jurnal Teknologi Informasi, 17(1), 33–41.

[13] Yuliono, W. A., & Prihanto, A. (2021). Sinergi Replikasi Server dan Sistem Failover pada *Database* Server untuk Mereduksi Downtime Disaster Recovery Planing (DRP). Journal of Informatics and Computer Science (JINACS), 3(01), 29–38. https://doi.org/10.26740/jinacs.v3n01.p29-38

[14] Thaariq, M. R. A., Hamzah, A., & Raharjo, S. (2024). Analisis Perbandingan Aspek Keamanan Basis Data Menggunakan Skema dan Tanpa Skema di PostgreSQL. JNANALOKA, 51–61. https://doi.org/10.36802/jnanaloka.2024.v5-no2-51-61

[15] Nurdin, H., & Handono, F. W. (2019). PERANCANGAN N-CLUSTERING *HIGH AVAILABILITY* WEB SERVER DENGAN LOAD BALANCING DAN FAILOVER.