Journal of Cyber Forensics and Security (CYFORS) *ISSN 3032-5870*
Vol.3, No.1 (2026)

IMPLEMENTATION OF HIGH AVAILABILITY WITH NGINX LOAD BALANCER AND GALERA CLUSTER

Dino Ananda Saputra[1)], Indrawan[2)], Chaerul Umam[3)], Muhammad Aditia[4)], Anas Nasrulloh[5)], Irwan Siswanto[6)]

[1,2,3,4)] *Information Technology, Institute Technology of South Tangerang*
[5,6)] *Information System, Institute Technology of South Tangerang*
email : : dino.ananda22@gmail.com[1)], indrawan.gcgarasi@gmail.com[2)], chaerulumam1989@gmail.com[3)], aditia22@gmail.com[4)], anas@itts.ac.id[5)], irwansiswanto@itts.ac.id[6)]

Abstract

High Availability (HA) systems play an important role in ensuring that critical applications continue to run despite server failures, hardware failures, or cyberattacks. This study discusses the implementation of the HA system using Nginx as the load balancer and the Galera Cluster for the MariaDB database. The main goal is to optimize system performance and minimize downtime during traffic spikes or system outages. By leveraging Nginx for traffic distribution and Galera Cluster for real-time data replication, this architecture provides high fault tolerance and reliable service availability. This study evaluates the system's ability to deal with traffic spikes, maintain service continuity, and be resistant to hardware and software interference. The system is designed to meet modern SLA standards with a minimum uptime of 99.95%. In addition, the implementation of a failover mechanism on applications and databases allows the system to continue operating normally even if there is a disruption to one of the components.

***Keywords* :**
High availbility, nginx, load balancing, galera, mariadb, cluster.

Introduction

In the ever-growing digital era, organizations rely heavily on application systems to run daily operations and serve users optimally. A system that is unable to cope with surging demand or fails to maintain service availability will lead to economic losses, reputational decline, and loss of customer trust. According to Andrew S. Tanenbaum in his book Distributed Systems: Principles and Paradigms (2007), one of the main pillars of modern systems is availability, which is the ability of the system to remain accessible despite failures in some of its components.

The main problem that is often faced is downtime, which is the time when the system is inaccessible to the user. Downtime can be caused by a variety of factors, such as:

A surge in user demand that the existing infrastructure is unable to handle. As explained by Rajkumar Buyya in Cloud Computing: Principles and Paradigms (2011), load spikes often occur in applications that are dynamic in nature and require optimal load distribution to avoid data traffic bottlenecks.

Hardware or software failure, where a physical or virtual server is damaged so that the system cannot run properly.

Cyber attacks such as Distributed Denial of Service (DDoS) or exploits of vulnerabilities that can cause the system to malfunction. In this context, the protection of servers is very important as described in Web Application Security by Bryan Sullivan and Vincent Liu (2011).

To overcome these challenges, the implementation of the High Availability (HA) System is the main need. HA refers to the system's ability to minimize downtime by supporting continuous operation despite disruptions. One of the key components in HA implementation is Load Balancing. Nginx, as one of the load balancer technologies, allows the distribution of traffic to multiple backend servers. This ensures that user requests are not centralized on a single server, which can lead to bottlenecks or complete failures. As mentioned by Clément Nedelcu in his book Nginx HTTP Server (2010), "Nginx excels in

handling large numbers of concurrent connections, making it ideal for high-traffic applications."

In addition, other critical aspects of the HA system are

database replication. Galera Cluster for MariaDB provides a multi-master replication solution that supports real-time data synchronization. This technology allows each node in the cluster to act as a master, thereby improving scalability and failure tolerance. According to Baron Schwartz in High Performance MySQL (2012), "Cluster-based database solutions provide fault tolerance and ensure data integrity during failures, making them ideal for HA deployments."

Through the combination of Nginx as a load balancer and MariaDB's Galera Cluster for database replication, the system can provide high reliability in handling demand spikes while ensuring data integrity. This implementation not only reduces the impact of downtime but also provides resilience to cyberattacks and hardware failures. With this approach, organizations can more effectively manage their technology infrastructure to support the ever-increasing operational needs.

Departing from this case, the author came up with the idea to create a system that can maintain Availability which is outlined in the form of this journal entitled ***"Implementation of High Availability System with the Utilization of Nginx Load Balancing and Galera Cluster MariaDB Database"***

Literature Review

1. High Availability (HA) System

High Availability (HA) is a concept that focuses on ensuring that the system can continue to operate despite disruptions or damage to some of its components. Tanenbaum and Van Steen (2007) in the book Distributed Systems: Principles and Paradigms explain that systems designed with HA principles must have redundancy and the ability to detect and recover from failures quickly. The system also relies on three main aspects: availability, reliability, and scalability. The combination of these three provides the basis for keeping the service running despite an outage in one of the subsystems.

2. Load Balancing with Nginx

Load balancing is a technique of distributing network traffic or user requests to multiple backend servers to prevent overloading on a single server. According to Nedelcu (2010) in Nginx HTTP Server, Nginx is one of the most efficient solutions in handling high traffic thanks to its ability to manage thousands of connections simultaneously with minimal resource consumption. Nginx works with algorithms such as Round Robin, Least Connections, and IP Hash to ensure that workloads are optimally distributed to each server. In addition, the failover feature on Nginx provides a guarantee that if one of the servers fails, traffic will be automatically redirected to another server in the pool.

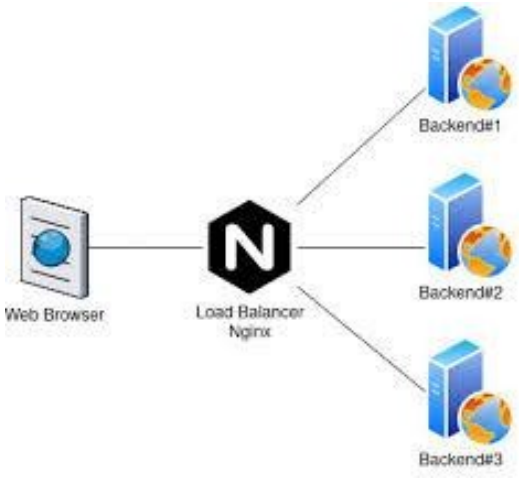


Figure 2. Nginx Load Balancing

a. Round Robin

Round Robin is the default algorithm in Nginx that distributes requests to the backend server in turn. Each server gets one request evenly in a recurring cycle, regardless of server status or load.

According to Nedelcu (2010) in Nginx HTTP Server, "Round Robin is a simple and effective load balancing method for distributing requests equally across all backend servers. It is ideal for systems with equally capable servers and uniform traffic."That is why

Advantages:

- The implementation is simple and easy to understand.
- Suitable for infrastructure with servers that have uniform specifications.

Disadvantages:

- It doesn't pay attention to the actual load on the backend server, which can cause a bottleneck if one of the servers has lower performance.

b. IP Hash IP Hash is an algorithm that distributes requests based on the hash of the client's IP address. This algorithm ensures that requests from the same IP will always be routed to the same server, unless that server is unavailable.

Nedelcu (2010) also mentions that, "The IP Hash method is particularly useful for session persistence,

where the application relies on the same backend server to handle consecutive requests from the same client."That is why

Advantages :

- It supports session stickiness, making it suitable for applications that store user sessions on specific servers.

- Consistency in handling requests from the same client.

Disadvantages :

- Load distribution is less than optimal, especially if the client's IP distribution is uneven.

c. Least Connections

Least Connections is an algorithm that directs requests to the server with the least active connections. This algorithm takes into account server workload when distributing requests, making it more efficient than Round Robin in systems with non-uniform request traffic.

Schwartz et al. (2012) in High Performance MySQL explain that, "The Least Connections method effectively balances the workload by considering the number of active connections, preventing overload and improving efficiency in environments with variable traffic patterns."

Advantages:

- Optimal for servers with different capacities or varied workloads.

- Prevents certain servers from being overloaded.

Disadvantages:

- Requires active monitoring of each server's active connection.

- It does not take into account the duration of each connection or the processing time of the request. Comparison of the Three Algorithms:

- Round Robin: Suitable for simple systems with uniform servers and even traffic.

IP Hash: Ideal for applications that require session consistency or have session data stored on a specific server. Least Connections: Best suited for systems with varying workloads or servers with different specifications.

3. Database Replication with Galera Cluster

Galera Cluster is a multi-master replication-based database clustering solution that supports consistent data synchronization between nodes. Schwartz et al. (2012) in High Performance MySQL explain that one of the main advantages of Galera Cluster is its multi-master architecture that allows all active nodes to receive read and write requests. With a synchronous replication model, data updated on one node is immediately replicated to another node, thus guaranteeing data consistency even if one of the nodes fails.

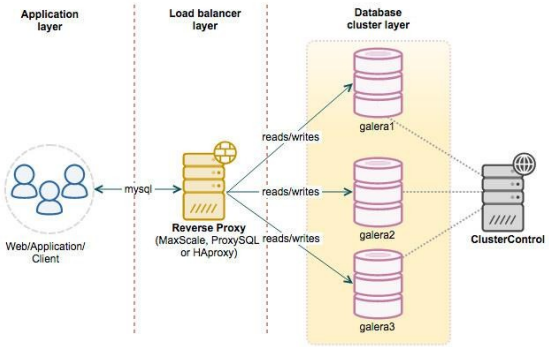


Figure 3. MariaDB Galera Cluster

4. Failure Management and Recovery

Another aspect that is no less important is failure management. Buyya et al. (2011) in Cloud Computing: Principles and Paradigms emphasize the importance of integrating monitoring and failure detection mechanisms into HA systems. Active monitoring ensures that every component in the system is monitored in real-time, so failures can be identified early to minimize their impact.

5. System Security and Resiliency

Security is also an important factor in HA systems, especially to protect infrastructure from threats such as Distributed Denial of Service (DDoS) attacks and vulnerability exploits. Sullivan and Liu (2011) in Web Application Security mentioned that the implementation of application firewalls, data encryption, and strong authentication is necessary to keep the system secure and available.

6. HA Implementation for Modern Applications

Various studies and implementations show that combining load balancing with database replication is

an effective solution for building HA systems. A Combination of Nginx and

Journal Of High Availability System

Galera Cluster has proven to deliver high efficiency and scalability, making it an ideal choice for applications with dynamic demand levels. With the right configuration, this architecture can significantly reduce downtime and provide a more reliable user experience.

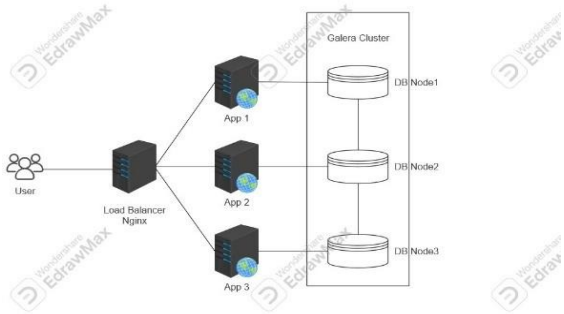


Figure 4. High Availability System

Research Methods

To build this High Availability System, it is necessary to design excess infrastructure that has been running before.

In this case, the application is set up as shown below:

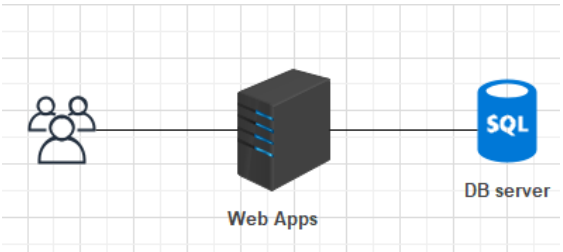


Figure 5. Stand Alone Application Architecture

Web-based applications are set up on 1 server that can be accessed directly by the user, where the database is also separate from the web application server. From this architecture, it is very prone to service downtime because the system only runs on 1 machine. When there is a hard were failure or a surge in requests from users, the web application service will be down and take time to recover.

Departing from this problem, it is necessary to design a system that can minimize service downtime by creating load-balancing-based clustering setups for web applications and database clustering that synchronizes between the nodes.

To build the design, additional servers are needed to build this cluster system. The design to be built is as follows:

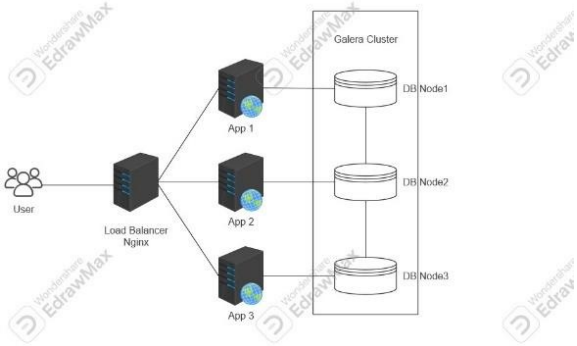


Figure 6. High Availability System

From the above design, additional system components are needed which are as follows:

1. 1 LB Server (Using Nginx Load Balancing)

On LB Server the domain and HTTPS connection are set up. However, the request will be forwarded to the application's HTTP service to the three application server nodes behind it.

2. 3 Application Server.

These 3 servers store web-based applications from system services that are the same as each other. With the following setup:

- Application Server 1 is connected to the database server 1,
- Application Server 2 is connected to database server 2,
- Application Server 3 connects to database server 3

3. 3 MariaDB database servers

3 Server databases are set up clustering and synchronize master-to-master between nodes.

SYSTEM IMPLEMENTATION

a.LB Server

Setup for the https service domain or port 443 with the load balancing configuration as follows:

```
vi /etc/nginx/sites-enabled/loadbalance.conf

upstream backend { least_conn;
server ip_address_app1:80 max_fails=3
fail_timeout=10s;
server ip_address_app2:80 max_fails=3
fail_timeout=10s;
server ip_address_app3:80 max_fails=3
fail_timeout=10s;
}
```

```
server {
server_name example.my.id;
error_log /var/log/nginx/examplemyid_error.log;
access_log /var/log/nginx/examplemyid_access.log;

location / {
proxy_redirect
                    o
ff;
proxy_set_header    X-Real-IP
$remote_addr; proxy_set_header X-
Forwarded-For
$proxy_add_x_forwarded_for;
proxy_set_header    X-Forwarded-Proto
$scheme; proxy_set_header        Host
$http_host;
proxy_pass http://backend;
}

443 SSL;
/etc/nginx/ssl/examplemyid/fullchain.pem
ssl_certificate_key
/etc/nginx/ssl/examplemyid/privkey.pem;
```

The load balancing configuration is contained in the upstream backend function. Above it can be seen that the algorithm used is the least connection and defines all the nodes of the server application. In the configuration, there is a process of checking the status of the Node whether it is active or not in the following configuration:

max_fails = 3 file_timeout=10s;

This means that the load balancer will continuously check each node, if the node does not respond 3 times and the duration is 10s, it means that the node is declared down and the request will be forwarded to another available node.

To reverse or forward requests that enter the LB server to the application server is in the following configuration:

proxy_pass http://backend;

b. Application Server

On the application server, only configuration for HTTP port or port 80 is carried out as usual as needed. The same configuration applies to all 3 active nodes and the application must be the same between the 3 nodes of the server application. If there are updates or changes to application 1, then these updates and changes must also be made to server application 2 and server application 3.

```
server {
listen      80 default  server;

server_name example.my.id;
Root        /www-data/example/public;
index index.php index.html pindex.htm;
access_log /www-data/log/example_id_access.log;
error_log /www-data/log/example_id_error.log;

# Load configuration files for the default server
block.

location / {
try_files $uri $uri/ /index.php$query_string;
}

location ~ \.php$ {
include /etc/nginx/default.d/*.conf;
include fastcgi.conf;
fastcgi_split_path_info ^(.+\.php)(/.+)$;
fastcgi_pass unix:/run/php-fpm/www.sock;
}
#error_page 404 /index.php;

location ~ /\. {
deny all;
}
}
```

The above configuration applies to all setups on the application server. Furthermore, for the configuration of applications to databases, each application 1 is connected to database 1, application 2 to database 2

and application 3 to database 3. To split the service database load.

c. Database Server

Perform cluster galera configuration on 3 server databases. The steps are as follows:

- Install the mariadb galera packet on the 3rd database server

```
# dnf install galley -y
```

- Backup and restore databases from existing databases to other databases as nodes 2 and 3 galera clusters.

- Setup the galera on node 1 of the database server by adding the following configuration on the

/etc/my.cnf.d/server.cnf

```
[mysqld]
log_error=/var/log/mariadb.log
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0
# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib64/galera-
4/libgalera_smm.so
# Galera Cluster Configuration
wsrep_cluster_name="clusterDB"
wsrep_cluster_address="gcomm://IP_AddressDB1
,IP_AddressDB2,IP_Address3"
# Synchronization Configuration
wsrep_sst_method=rsync galley
# Configuration Node Gallery
wsrep_node_address="IP_AddressDB1"
wsrep_node_name="DBNode1"
```

Run the service galera cluster on node1 by running the following command:

```
# galera_new_cluster
```

Configure node2 and node 3 by adding The following configuration on file /etc/my.cnf.d/server.dnf

Node2 DB server:

```
[mysqld]
log_error=/var/log/mariadb.log
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0
# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib64/galera-
4/libgalera_smm.so
# Cluster Configuration Gallery
wsrep_cluster_name="clusterDB"
wsrep_cluster_address="gcomm://IP_AddressDB1
,IP_AddressDB2,IP_Address3"
# Synchronization Configuration
wsrep_sst_method=rsync galley
# Configuration Node Gallery
wsrep_node_address="IP_AddressDB2"
wsrep_node_name="DBNode2"
```

Node 3 DB Server:

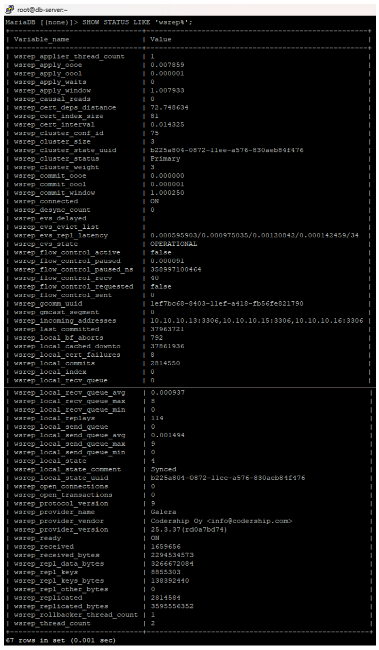
```
[mysqld]
log_error=/var/log/mariadb.log
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0
# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib64/galera-
4/libgalera_smm.so
# Cluster Configuration Gallery
wsrep_cluster_name="clusterDB"
wsrep_cluster_address="gcomm://IP_AddressDB1
,IP_AddressDB2,IP_Address3"
# Synchronization Configuration
wsrep_sst_method=rsync galley
# Configuration Node Gallery
wsrep_node_address="IP_AddressDB3"
wsrep_node_name="DBNode3"
```

- Start the MariaDB service node 2 and node 3 to activate the server's database service.

```
# systemctl start mysqld
```

- Perform a synchronize database cluster check.

```
MariaDB [(none)]> show status as 'wsrep_%';
```



```
root@db-server:~
MariaDB [(none)]> SHOW STATUS LIKE 'wsrep%';
+-------------------------------+----------------------------------------------------+
| Variable_name                 | Value                                              |
+-------------------------------+----------------------------------------------------+
| wsrep_applier_thread_count    | 1                                                  |
| wsrep_apply_oooe              | 0.007859                                           |
| wsrep_apply_oool              | 0.000001                                           |
| wsrep_apply_waits             | 0                                                  |
| wsrep_apply_window            | 1.007933                                           |
| wsrep_causal_reads            | 0                                                  |
| wsrep_cert_deps_distance      | 72.748634                                          |
| wsrep_cert_index_size         | 81                                                 |
| wsrep_cert_interval           | 0.014325                                           |
| wsrep_cluster_conf_id         | 75                                                 |
| wsrep_cluster_size            | 3                                                  |
| wsrep_cluster_state_uuid      | b225a804-0872-11ee-a576-830aeb84f476               |
| wsrep_cluster_status          | Primary                                            |
| wsrep_cluster_weight          | 3                                                  |
| wsrep_commit_oooe             | 0.000000                                           |
| wsrep_commit_oool             | 0.000001                                           |
| wsrep_commit_window           | 1.000250                                           |
| wsrep_connected               | ON                                                 |
| wsrep_desync_count            | 0                                                  |
| wsrep_evs_delayed             |                                                    |
| wsrep_evs_evict_list          |                                                    |
| wsrep_evs_repl_latency        | 0.000595903/0.000975035/0.00120842/0.000142459/34  |
| wsrep_evs_state               | OPERATIONAL                                        |
| wsrep_flow_control_active     | false                                              |
| wsrep_flow_control_paused     | 0.000091                                           |
| wsrep_flow_control_paused_ns  | 358997100464                                       |
| wsrep_flow_control_recv       | 40                                                 |
| wsrep_flow_control_requested  | false                                              |
| wsrep_flow_control_sent       | 0                                                  |
| wsrep_gcomm_uuid              | 1ef7bc68-8403-11ef-a418-fb56fe821790               |
| wsrep_gmcast_segment          | 0                                                  |
| wsrep_incoming_addresses      | 10.10.10.13:3306,10.10.10.15:3306,10.10.10.16:3306 |
| wsrep_last_committed          | 37963721                                           |
| wsrep_local_bf_aborts         | 792                                                |
| wsrep_local_cached_downto     | 37861936                                           |
| wsrep_local_cert_failures     | 8                                                  |
| wsrep_local_commits           | 2814550                                            |
| wsrep_local_index             | 0                                                  |
| wsrep_local_recv_queue        | 0                                                  |
| wsrep_local_recv_queue_avg    | 0.000937                                           |
| wsrep_local_recv_queue_max    | 8                                                  |
| wsrep_local_recv_queue_min    | 0                                                  |
| wsrep_local_replays           | 114                                                |
| wsrep_local_send_queue        | 0                                                  |
| wsrep_local_send_queue_avg    | 0.001494                                           |
| wsrep_local_send_queue_max    | 9                                                  |
| wsrep_local_send_queue_min    | 0                                                  |
| wsrep_local_state             | 4                                                  |
| wsrep_local_state_comment     | Synced                                             |
| wsrep_local_state_uuid        | b225a804-0872-11ee-a576-830aeb84f476               |
| wsrep_open_connections        | 0                                                  |
| wsrep_open_transactions       | 0                                                  |
| wsrep_protocol_version        | 9                                                  |
| wsrep_provider_name           | Galera                                             |
| wsrep_provider_vendor         | Codership Oy <info@codership.com>                  |
| wsrep_provider_version        | 25.3.37(rd0a7bd74)                                 |
| wsrep_ready                   | ON                                                 |
| wsrep_received                | 1659656                                            |
| wsrep_received_bytes          | 2294534573                                         |
| wsrep_repl_data_bytes         | 3266672084                                         |
| wsrep_repl_keys               | 8855303                                            |
| wsrep_repl_keys_bytes         | 138392440                                          |
| wsrep_repl_other_bytes        | 0                                                  |
| wsrep_replicated              | 2814584                                            |
| wsrep_replicated_bytes        | 3595556352                                         |
| wsrep_rollbacker_thread_count | 1                                                  |
| wsrep_thread_count            | 2                                                  |
+-------------------------------+----------------------------------------------------+
67 rows in set (0.001 sec)
```

Figure 7. Clustering galera status results

Results and Discussion

For this High Avalailability System test, several scenarios can be carried out and checks on the application logs can be carried out.

1. Server Load Balancing Testing

Testing by shutting down one of the application servers and testing the application's Web access via a browser.

```
[root@lb-server ~]# ping 10.10.10.11
PING 10.10.10.11 (10.10.10.11) 56(84) bytes of data.
64 bytes from 10.10.10.11: icmp_seq=1 ttl=64 time=0.414 ms
64 bytes from 10.10.10.11: icmp_seq=2 ttl=64 time=0.392 ms
^C
--- 10.10.10.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1060ms
rtt min/avg/max/mdev = 0.392/0.403/0.414/0.011 ms
[root@lb-server ~]# ping 10.10.10.12
PING 10.10.10.12 (10.10.10.12) 56(84) bytes of data.
From 10.10.10.5 icmp_seq=1 Destination Host Unreachable
From 10.10.10.5 icmp_seq=2 Destination Host Unreachable
From 10.10.10.5 icmp_seq=3 Destination Host Unreachable
^C
--- 10.10.10.12 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4088ms
pipe 3
[root@lb-server ~]# ping 10.10.10.14
PING 10.10.10.14 (10.10.10.14) 56(84) bytes of data.
64 bytes from 10.10.10.14: icmp_seq=1 ttl=64 time=0.447 ms
64 bytes from 10.10.10.14: icmp_seq=2 ttl=64 time=0.423 ms
64 bytes from 10.10.10.14: icmp_seq=3 ttl=64 time=0.479 ms
^C
--- 10.10.10.14 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2048ms
rtt min/avg/max/mdev = 0.423/0.449/0.479/0.033 ms
[root@lb-server ~]#
```

LB Server

From the ping results of the image above, it can be seen that the application server 2 is detected down or cannot be accessed by LB Server. However, in terms of application, the application can still be accessed.

Figure 9. Testing application access after Application server 2 is shut down.

Furthermore, it can also be seen from the application access log, namely from the access_log log. Where it can be seen that the request will still enter each application from the incoming request on the LB Server.

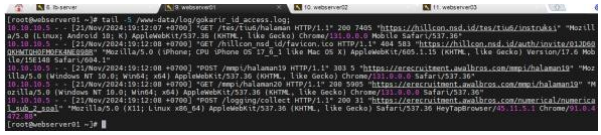


Figure 10. Access Log on Application 1

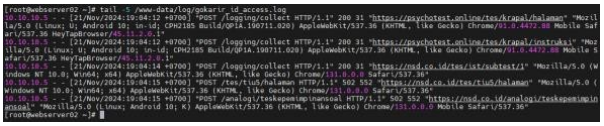


Figure 11. Access Log on Application 2

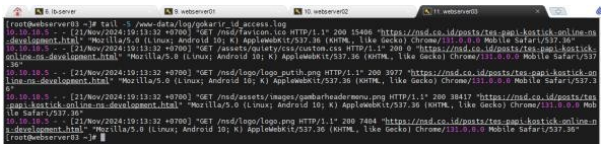


Figure 12. Access Log on Application 3

1. MariaDB Cluster Galera Testing

For testing the MariaDB cluster galera, it can be done by checking through the monitoring tools from maxscale.

```
[root@lb-server ~]# maxctrl list servers
```

Server	Address	Port	Connections	State	GTID	Monitor
DBNode1	10.10.10.13	3306	0	Master, Synced, Running		MariaDB-Monitor
DBNode2	10.10.10.15	3306	0	Slave, Synced, Running		MariaDB-Monitor
DBNode3	10.10.10.16	3306	0	Slave, Synced, Running		MariaDB-Monitor

```
[root@lb-server ~]#
```

Figure 13. Synchronized DB Node Monitor

Furthermore, it can also be powered off on one of the DB nodes and checked for monitoring status

```
[root@lb-server ~]# maxctrl list servers
```

Server	Address	Port	Connections	State	GTID	Monitor
DBNode1	10.10.10.13	3306	0	Master, Synced, Running		MariaDB-Monitor
DBNode2	10.10.10.15	3306	0	Down		MariaDB-Monitor
DBNode3	10.10.10.16	3306	0	Slave, Synced, Running		MariaDB-Monitor

```
[root@lb-server ~]#
```

Figure 14. Monitor 1 Node DB Down

After the system is down on Node DB 2, the application can still be accessed

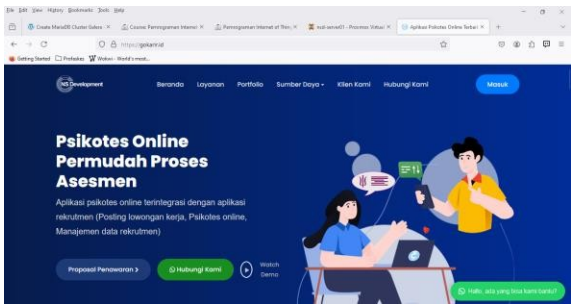


Figure 15. The application can still be accessed if Node2 DB is Down

However, this condition still requires monitoring and bypassing or turning off the LB Server configuration for clustering to Application Server 2. This is because the application service is still detected by LB but application 2 cannot access the DB Server. This is because it does not create a Load balancer system for its database as an application access to Clustering DB. Next, check the sample data from the database whether it is the same in each DB. This can be done by running Query Select to display some of the last data in each DB.

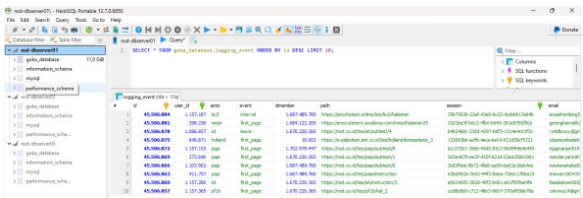


Figure 16. Query 10 The last data from one of the app's log event tables.

From the results of the query executed on each DB Node, it gets the exact same data output. And this means that the database is actually set up in the cluster and synchronized in real time.
If one of the DB nodes is dead or damaged, then it can be powered on again with the same setup and service as before. We can just start the database to get the DB synchronized again into Clustering.

Conclusions and Suggestions

The implementation of the High Availability (HA) system using Nginx as the load balancer and Galera Cluster MariaDB as the backend database proves that this technology is able to significantly increase the availability of application services. Nginx as a load balancer plays the role of efficiently distributing user requests to the backend server using algorithms such as Round Robin, Least Connections, or IP Hash, which can be adjusted to the needs of the system. Meanwhile, Galera Cluster provides synchronous data replication to ensure consistency between nodes, thus being able to maintain data integrity even if one of the nodes fails.

Tests conducted show that this combination of technology can handle traffic spikes with low response times and minimize the risk of downtime due to server failure. With this architecture, the system is able to achieve an availability level that is close to the modern SLA standard, which is a minimum uptime of 99.95%. In addition, the implementation of failover at the application and database level allows the system to continue operating normally despite an outage in one of the components.

However, there are challenges in implementation, such as complex configuration requirements and consistent monitoring to ensure optimal performance. For further development, the system can be improved with the integration of real-time monitoring tools such as Prometheus or Grafana for early detection of anomalies, as well as failover automation using devices such as Keepalived or HAProxy. And there is still a need for development to build Load Balancing in the MariaDB cluster gallery for application-to-database communication via Load Balancing, so as to also reduce manual failover activities on the LB side of the application and can further increase the SLA value of the application service.

Overall, the solutions implemented provide a solid foundation for a reliable and scalable system. This approach is particularly relevant for organizations that prioritize application service continuity and business sustainability, especially in the face of technical challenges and cyberattacks in the digital age.

References

[1] Bass, L., Clements, P., & Kazman, R. (2019). Software architecture in practice (3rd ed.). Addison-Wesley.

[2] Bhayangkara, D. S., & Ashari, W. M. (2024). The effect of load balancing on DDoS attack mitigation using NGINX. The Indonesian Journal of Computer Science, 13(4), 4118–4127. https://doi.org/10.33022/ijcs.v13i4.4118

[3] Crilly, L. (2021). Scaling MySQL with TCP load balancing and Galera Cluster. NGINX (F5). https://www.f5.com/company/blog/nginx/scaling-mysql-tcp-load-balancing-nginx-plus-galera-cluster

[4] Huang, Y., Zhang, M., & Xu, K. (2021). High availability and load balancing in distributed systems. Springer. https://doi.org/10.1007/978-3-030-69132-4

[5] Ikramsyah, M. A., Seta, H. B., Isnainiyah, I. N., & Theresiawati, T. (2025). Evaluating service quality in NGINX load balancing: A comparative study of round robin and least connection algorithms. Informatics: Journal of Computer Science, 21(3). https://doi.org/10.52958/iftk.v21i3.10796

[6] MariaDB Corporation. (2025). Load balancing in MariaDB Galera Cluster. https://mariadb.com/docs/galera-cluster/high-availability/load-balancing/load-balancing-in-mariadb-galera-cluster/

[7] NGINX, Inc. (2025). Configuring active-active high availability and additional passive HA nodes. https://docs.nginx.com/nginx/admin-guide/high-availability/ha-keepalived-nodes/

[8] Prakasa, J. E. W., Hanani, A., & Hariri, F. R. (2023). Improving Moodle performance using HAProxy and MariaDB Galera Cluster. Applied Information System and Management, 7(1), 1–10. https://doi.org/10.15408/aism.v7i1.34871

[9] Setiawan, A., & Kansha, W. M. (2021). The creation of a cluster database system using the Galera Cluster at the IPB University Vocational School. Journal of Applied Science: Information Vehicles and Agricultural Technology Transfer, 11(2), 49–59. https://doi.org/10.29244/jstsv.11.2.49-59

[10] Sibuea, S., Widodo, Y. B., & Khaliq, M. N. (2024). The use of NGINX software as load balancing web server clustering. Journal of Informatics and Computer Technology, 10(1), 45–54. https://doi.org/10.37012/jtik.v10i1.2184

[11] Sumarna, S., Nurdin, H., & Handono, F. W.

(2025). Planning N-clustering high availability web server with load balancing and failover. Journal of Engineering and Computer Science, 9(1), 22–30.

[12] Terence, J. K., & Nagaiah, M. V. (2024). High availability database infrastructure with Galera & HAProxy. International Journal for Multidisciplinary Research, 6(6), 1–8.

[13] Severalnines AB. (2020). Using NGINX as a database load balancer for Galera Cluster. https://severalnines.com/blog/using-nginx-database-load-balancer-galera-cluster/

[14] Hwang, K., Fox, G. C., & Dongarra, J. (2013). Distributed and cloud computing: From parallel processing to the Internet of Things. Morgan Kaufmann.

[15] Kartikasari, D., & Nugroho, A. (2022). Analysis of high availability system performance on clustering-based web servers. Journal of Information and Communication Technology, 14(2), 101–110.

[16] Rahman, F., & Pratama, R. (2023). Implementation of failover and load balancing systems to improve the availability of web services. Journal of RESTI (Systems Engineering and Information Technology), 7(3), 512–520. https://doi.org/10.29207/resti.v7i3.4789

[17] Slesarev, A., Mikhailov, M., & Chernishev, G. (2022). Benchmarking hashing algorithms for load balancing in distributed database environments. arXiv preprint. https://arxiv.org/abs/2211.00741

[18] Zhang, Q., Chen, M., & Li, Y. (2021). Performance evaluation of load balancing algorithms in high availability web systems. Journal of Cloud Computing, 10(1), 1–14. https://doi.org/10.1186/s13677-021-00257-4