Balanceo de Carga de Bases de Datos con mysql y nginx

Andres J. urbano, Alejandro Monsalve, Camilo A. Velásquez, Mauricio Cortes
Universidad autónoma de occidente
Facultad de ingeniería departamento de Sistemas y procesos
Servicios telemáticos – cod 533246 – grupo 51
andres_jose.urbano@uao.edu.co
alejandro.monsalve@uao.edu.co
camilo_alb.velasuqez@uao.edu.co
mauricio.cortes@uao.edu.co

Resumen— En el presente proyecto, presentamos el diseño, las posibles soluciones, una propuesta de implementación y las pruebas correspondientes para abordar el problema de alta disponibilidad de servicios con MySQL. Se discutirán los resultados de las pruebas para evaluar la efectividad de la solución propuesta.

Abstract— In this project, we present the design, possible solutions, a proposed implementation, and the corresponding tests to address the problem of high availability of services with MySQL. The test results will be discussed to evaluate the effectiveness of the proposed solution..

I. Introducción

En los sistemas de información modernos, la disponibilidad y la resiliencia son aspectos críticos para garantizar la continuidad del negocio y la satisfacción del usuario. Una de las principales preocupaciones en este contexto es la centralización de la base de datos, ya que puede representar un único punto de falla, comprometiendo la disponibilidad del servicio en caso de interrupciones.

Para mitigar estos riesgos, se implementan arquitecturas de balanceo de carga, como la configuración maestro-esclavo en MySQL, que distribuyen la carga de trabajo y proporcionan redundancia.

Esta documentación describe los riesgos asociados con la centralización de bases de datos y presenta una solución de balanceo de carga maestro-esclavo para mejorar la disponibilidad y la resiliencia del sistema.

Los márgenes para la hoja en formato A4 serán: superior = 19 mm, inferior = 30 mm, lado = 13 mm. El ancho de la columna es de 88 mm. El espacio entre la dos columnas es de 4 mm. La sangría de los párrafos es de 3,5 mm.

El texto en las columnas debe estar justificado. La longitud de las figuras y tablas debe ajustarse al de la columna.

II. contexto

*La centralización de bases de datos sin el uso de clusters**(Un cluster de bases de datos es una arquitectura donde múltiples servidores de bases de datos trabajan juntos para su disponibilidad, escalabilidad, redundancia y balanceo)** presenta varios problemas que pueden afectar significativamente la disponibilidad y el rendimiento del sistema. A continuación, se detallan los principales problemas asociados con esta configuración:*

Punto Único de Falla: Una base de datos centralizada se convierte en un único punto de falla. Si el servidor falla, todo el sistema que depende de esa base de datos quedará inoperativo, llevando a tiempos de inactividad prolongados.

Sobrecarga y Rendimiento: Todas las solicitudes de lectura y escritura se dirigen a un solo servidor. A medida que aumenta el número de usuarios y transacciones, el servidor puede sobrecargarse, resultando en un rendimiento degradado y mayores tiempos de respuesta.

Escalabilidad Limitada: Las bases de datos centralizadas tienen limitaciones significativas en términos de escalabilidad. Escalar verticalmente (mejorar el hardware del servidor) es difícil y costoso, y no hay opciones de escalado horizontal (agregar más servidores).

Recuperación y Redundancia: La capacidad de recuperación ante desastres es limitada. La ausencia de redundancia significa que, si el servidor principal falla, no hay servidores secundarios o de respaldo que puedan asumir la carga automáticamente, aumentando el riesgo de pérdida de datos y tiempos de inactividad prolongados.

*la centralización de bases de datos sin clusters introduce **riesgos** significativos para la disponibilidad, escalabilidad y recuperación del sistema. Estos problemas resaltan la necesidad de arquitecturas más resilientes y distribuidas, como las configuraciones de balanceo de carga y replicación maestro-esclavo en MySQL, para mitigar estos riesgos y mejorar la continuidad del negocio.*

III. Alternativas de solucion

Para la generación de clusters de bases de datos tenemos varias opciones tanto de implementación manual, como servicios que tenemos en nube o en otras tecnologías que nos permiten configurar y administrar estos clusters

- **Amazon Aurora:** es un motor de base de datos relacional de alta disponibilidad y rendimiento, compatible con MySQL y PostgreSQL, que utiliza clusters para distribuir datos y mejorar la resiliencia. Los clusters de Aurora consisten en una instancia principal para operaciones de lectura y escritura, y hasta 15 réplicas para operaciones de lectura y redundancia.
- **C4C7US**:es una plataforma de gestión de infraestructura en la nube que automatiza procesos de DevSecOps y DataOps, permitiendo la creación y gestión eficiente de recursos como bases de datos, almacenamiento, y redes privadas virtuales. Ofrece escalabilidad automática, alta disponibilidad, monitoreo integral y una solución de CI/CD, lo que facilita la implementación y gestión de clusters, asegurando un rendimiento óptimo y reduciendo costos.
- **ProxySQL:** es una solución avanzada para gestionar la conectividad a bases de datos MySQL/MariaDB, proporcionando funcionalidades de balanceo de carga y alta disponibilidad. Configurar clusters con ProxySQL implica varios pasos que incluyen la instalación de ProxySQL, configuración de nodos, y ajustes de reglas de enrutamiento.
- **MySQL Router:** Un componente oficial de MySQL que proporciona enrutamiento de solicitudes de bases de datos y puede ser utilizado en conjunto con MySQL InnoDB Cluster para alta disponibilidad.

.

IV. Diseño de la solucion

Por los requerimientos dados decidimos utilizar Mysql Router para la sincronziacion y nginx para el balanceo de carga

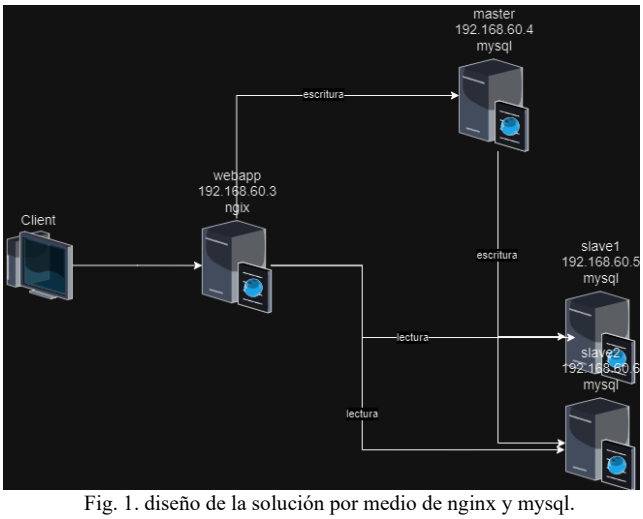

Fig. 1. diseño de la solución por medio de nginx y mysql.

Se va a utilizar vagrant para levantar los servidores en la demostración que haremos, y se utilizaran archivos .sh para poder correr todos los comandos y así la solución sea mas generalizada y automatizada.

Las tecnologías que se utilizaran serán

- Flask
- Mysql
- Python
- Ubuntu
- Vagrant
- Virtualbox

V. Implementación

El archivo de configuración de Vagrant define una infraestructura virtual compuesta por cuatro máquinas virtuales, todas basadas en la imagen "bento/ubuntu-22.04". A continuación se detalla la configuración de cada máquina:

1. Master
   Hostname: master
   IP: 192.168.60.4
   Provisiones:
   Copia un archivo SQL de inicialización (init.sql) desde el directorio local ./utils/mysql/ al directorio /home/vagrant/ en la máquina virtual.
   Copia un archivo de configuración de MySQL (my.cnf) desde ./utils/mysql/master/ al mismo destino.
   Ejecuta dos scripts de shell (script_mysq.sh y script_master.sh) que probablemente configuran y arrancan el servicio de MySQL como nodo maestro.
2. Slave1
   Hostname: slave1
   IP: 192.168.60.5
   Provisiones:
   Similar al nodo master, copia init.sql y my.cnf desde ./utils/mysql/slave/slave_1/ al directorio /home/vagrant/.
   Ejecuta dos scripts de shell (script_mysq.sh y script_slave.sh) que configuran y arrancan el servicio de MySQL como nodo esclavo.
3. Slave2
   Hostname: slave2
   IP: 192.168.60.6
   Provisiones:
   Copia init.sql y my.cnf desde ./utils/mysql/slave/slave_2/ al directorio /home/vagrant/.
   Ejecuta script_mysq.sh y script_slave.sh para configurar y arrancar MySQL como nodo esclavo.
4. Webserver
   Hostname: webserver
   IP: 192.168.60.3
   Provisiones:
   Copia el contenido de una aplicación en Flask web desde ./utils/api/ al directorio /home/vagrant/webapp/.
   Ejecuta dos scripts de shell (script_nginx.sh y script_webapp.sh) que probablemente configuran y arrancan el servidor web Nginx y despliegan la aplicación web.

Para resumir la arquitectura de las redes

Tabla 1.
Arquitectura de los las maquinas virtuales

Nombre	Ip	Tecnologías	uso
Master	192.168.60.4	Mysql	Administrador del cluster
Slave1	192.168.60.5	Mysql	Respaldo nodo
Slave2	192.168.60.6	Mysql	Respaldo o nodo
Webserver	192.168.60.3	Nginx, Python, Flask	Api, y balanceador de carga

A. Resumen de los scripts

- **script_mysq.sh:** Este script se encarga de instalar MySQL en la máquina virtual. Define la contraseña de root para MySQL y realiza la instalación del servidor de MySQL. Posteriormente, copia un archivo de configuración personalizado (`my.cnf`) al directorio de configuración de MySQL y reinicia el servicio para aplicar los cambios. Finalmente, ejecuta un script SQL de inicialización (`init.sql`) que se encuentra en el directorio del usuario `vagrant`, asegurando que la base de datos esté configurada adecuadamente desde el inicio.

- **script_master.sh:** Este script configura el nodo maestro de MySQL. Primero, crea un usuario de replicación llamado 'repl' con permisos de replicación y acceso total a la base de datos `myflaskapp`. Luego, obtiene el estado actual del registro binario (binlog) del maestro y lo guarda en un archivo llamado `master_status.txt` ubicado en el directorio compartido `/vagrant`. Esto es crucial para configurar la replicación en los nodos esclavos, asegurando que comiencen a replicar desde el punto correcto en el binlog.

- **script_slave.sh:** Este script configura los nodos esclavos de MySQL para la replicación. Verifica si el archivo `master_status.txt` existe y extrae los valores de `MASTER_LOG_FILE` y `MASTER_LOG_POS` necesarios para establecer la replicación. Configura el esclavo para conectarse al maestro utilizando estos valores, y luego inicia la replicación. Además, crea el usuario de replicación en el esclavo si no existe, asegurando que el nodo esté listo para replicar datos desde el maestro. Finalmente, guarda el estado de la replicación en un archivo `slave_status.txt` para su verificación.

- **script_nginx.sh:** Este script instala y configura Nginx en la máquina virtual para balancear la carga entre los servidores MySQL y servir una aplicación Flask. Primero, actualiza los repositorios e instala Nginx. Luego, crea un archivo de configuración para Nginx que define dos grupos de upstream: uno para el maestro de MySQL y otro para los esclavos, permitiendo balancear las consultas de lectura y escritura. También configura Nginx para servir la aplicación Flask en el puerto 80, redirigiendo el tráfico HTTP a la aplicación Flask que se ejecuta localmente en el puerto 5000. Finalmente, reinicia el servicio de Nginx para aplicar las nuevas configuraciones.

- **script_webapp.sh:** Este script instala Python y las dependencias necesarias para ejecutar una aplicación Flask. Instala varios paquetes como `python3-dev`, `default-libmysqlclient-dev`, y `build-essential` junto con `mysql-client` y `python3-pip`. Luego, utiliza pip para instalar Flask, `flask-cors`, `Flask-MySQLdb` y `Flask-SQLAlchemy`, preparando el entorno para la aplicación web. Finalmente, incluye comentarios sobre cómo ejecutar la aplicación Flask, aunque no los ejecuta directamente, dejando esta tarea para que sea realizada manualmente según sea necesario.

B. Esquema

El esquema “init.sql” SQL crea una base de datos llamada myflaskapp y define dos tablas: users y product. La tabla users contiene columnas para almacenar información de los usuarios, incluyendo un ID autoincrementado como clave primaria, nombre, correo electrónico, nombre de usuario y contraseña. La tabla product está diseñada para almacenar detalles de productos, incluyendo un ID autoincrementado, nombre del producto, descripción y precio. Además, el esquema inserta datos iniciales en ambas tablas, con dos usuarios (Juan y Maria) y cinco productos (como una Laptop y un Teléfono). Estos datos iniciales proporcionan una base para probar la funcionalidad de la aplicación

VI. DISEÑO DE PRUEBAS

- Pruebas de Replicación
  Para verificar la replicación en el servidor web, se realizarán solicitudes HTTP con los métodos POST, PUT y DELETE. Estas operaciones modificarán los datos en la base de datos, permitiendo observar cómo los cambios se replican en los nodos de la base de datos. Por ejemplo, una solicitud POST añadirá un nuevo usuario, una solicitud PUT actualizará la información de este usuario y una solicitud DELETE lo eliminará. Tras cada operación, se verificará en los nodos de replicación que los cambios se han propagado correctamente a través del clúster.

- Pruebas de Tolerancia a Errores
  Para evaluar la tolerancia a errores, se apagarán las máquinas virtuales que alojan los nodos de la base de datos y el servidor web en diferentes escenarios. Primero, se apagará el nodo maestro para observar cómo los nodos esclavos manejan la ausencia del maestro y si uno de ellos puede asumir su rol. Luego, se apagarán uno o más nodos esclavos para verificar la continuidad del sistema y la replicación en los restantes nodos esclavos. Finalmente, se apagará el

servidor web para ver cómo el sistema maneja la caída del servidor front-end y luego se reiniciará para verificar la recuperación del servicio sin pérdida de datos.

VII. DISCUSIÓN DE LAS PRUEBAS

Al hacer un POST, PUT o DELETE, dirigido al webserver, podemos comprobar que dichos cambios realizados a la base de datos se replican exitosamente en las bases de datos alojadas en los slaves.

Adicionalmente, al apagar una de las maquinas, y mandar un SELECT podemos ver como la aplicación sigue corriendo con normalidad, demostrando asi la alta disponibilidad de nuestro sistema.

VIII. CONCLUCIONES

El desarrollo y despliegue de este proyecto han proporcionado un conjunto de habilidades valiosas y ampliamente aplicables en múltiples áreas del la gestión de infraestructura. En primer lugar, la configuración de Nginx como servidor web y balanceador de carga ha sido fundamental para comprender cómo distribuir el tráfico de manera que sea persistente a errores entre diferentes bases de datos, mejorando así la disponibilidad y capacidad de respuesta del sistema. Este conocimiento es crucial para cualquier aplicación que deba manejar un alto volumen de tráfico y requiere una infraestructura robusta y escalable.

La implementación de balanceo de carga y alta disponibilidad es otro aspecto crítico que se ha abordado. Estas prácticas aseguran que la aplicación permanezca operativa y accesible incluso en caso de fallos del servidor, distribuyendo la carga de trabajo y minimizando el tiempo de inactividad. Este conocimiento es indispensable en el desarrollo de aplicaciones que deben ofrecer un servicio ininterrumpido y fiable, especialmente en entornos de producción donde la disponibilidad es una prioridad.

El manejo de configuración del servidor MYSQL nos permitió aprender la administración de las bases de datos y skills que nos pueden ayudar a la hora de mantener la disponibilidad de la informacion

La automatización del despliegue nos sirvió para comprender como manejar mejor valgrant, con uso del -c para cambiar archivos desde línea de comandos, hasta usar bien los providers, lo que nos permite hacer una automatización del despliegue aunque sea en el entorno de desarrollo

Por último, El desarrollo de APIs con Flask ha demostrado ser una habilidad esencial para la creación de aplicaciones web modernas. Flask, con su simplicidad y flexibilidad, permite construir rápidamente endpoints API que pueden ser consumidos por diferentes clientes, desde aplicaciones móviles hasta sistemas empresariales complejos. Junto con SQLAlchemy, que actúa como un ORM (Object-Relational Mapping), hemos aprendido a interactuar de manera eficiente con bases de datos relacionales, facilitando la gestión y manipulación de datos de una manera más intuitiva y coherente con la lógica de la aplicación.

este proyecto no solo ha desarrollado competencias técnicas esenciales, sino que también ha reforzado prácticas de desarrollo y gestión que son extremadamente valiosas para cualquier equipo que persona en el desarrollo y mantenimiento de aplicaciones web modernas y robustas. La combinación de estas habilidades y conocimientos proporciona una base sólida para enfrentar los desafíos de la industria tecnológica actual y futura basándonos en uno de las características de la calidad de software como es la alta disponibilidad.

REFERENCIAS

[1] Amazon Web Services, “What is Amazon Aurora?” docs.aws.amazon.com. [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.Overview.html. [Accessed: May 20, 2024].

[2] Oracle Corporation, “MySQL Router with InnoDB Cluster,” dev.mysql.com. [Online]. Available: https://dev.mysql.com/doc/mysql-router/8.4/en/mysql-router-innodb-cluster.html. [Accessed: May 13, 2024].

[3] DigitalOcean, “Cómo configurar un clúster Galera con MySQL en servidores Ubuntu 18.04,” digitalocean.com. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-configure-a-galera-cluster-with-mysql-on-ubuntu-18-04-servers-es. [Accessed: May 17, 2024].

[4] C4C7US, “Productos,” c4c7us.com. [Online]. Available: https://c4c7us.com/es/product/. [Accessed: May 21, 2024].