

ALTA DISPONIBILIDAD EN APLICACIONES WEB



TFG 2ºASIR 2023 / 2024

MIGUEL ÁNGEL RINCÓN BUENO



CC BY-NC 4.0 DEED

Attribution-NonCommercial 4.0 International

ESTA OBRA ESTA SUJETA A UNA LICENCIA DE

Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)

Atribución - No Comercial 4.0 Internacional (CC BY-NC 4.0)

Derivada: Licencia.



FICHA DE PROYECTO

Título del trabajo:	Alta Disponibilidad en Aplicaciones Web
Nombre del autor:	Miguel Ángel Rincón Bueno
Fecha de entrega (mm/aaaa):	06/2024
Área del trabajo final:	Redes
Ciclo Grado Superior:	Administración de Sistemas Informáticos en Red
Resumen del trabajo (máximo 250 palabras):	
<p>El proyecto consiste en crear una estructura de Alta Disponibilidad orientada a microservicios en un entorno de Virtualización como Proxmox. Esto permitirá que las PYMES, que dependen de una aplicación web para generar ingresos, puedan adquirir una infraestructura robusta y asegurar una entrega continua, sin cortes en el servicio.</p> <p>La arquitectura de microservicios divide una aplicación en servicios pequeños e independientes que se despliegan y gestionan de forma aislada, mejorando la escalabilidad y flexibilidad. Cada microservicio puede ser desarrollado, desplegado y escalado de manera autónoma.</p> <p>Para asegurar la Alta Disponibilidad, se implementarán mecanismos de distribución de carga que equilibren el tráfico entre múltiples instancias de microservicios, evitando puntos únicos de fallo y manteniendo el servicio accesible incluso en caso de fallos individuales. La distribución de carga también optimizará el rendimiento, manejando grandes volúmenes de solicitudes eficientemente y mejorando la experiencia del usuario.</p> <p>Una vez implementado el esquema de Alta Disponibilidad en Aplicaciones Web, se realizarán pruebas de rendimiento y consumo para comprobar la viabilidad del sistema. Estas pruebas evaluarán el comportamiento de la infraestructura bajo diferentes cargas de trabajo y condiciones, asegurando que puede manejar el tráfico esperado sin degradar la calidad del servicio.</p>	



1. Introducción	03
1.1 Contexto	04
a) Marco legal	05
1.2 Justificación	07
1.3 Limitaciones	09
1.4 Marco tecnológico-competencial	10
2. Objetivos	12
3. Propuesta de solución	14
4. Entorno de despliegue	23
5. Recursos	25
5.1 Requisitos mínimos	25
5.2 Presupuesto	25
5.3 Seguimiento y control	27
5.3.1 Mantenimiento del proyecto	30
6. Documentación Técnica	31
6.1 Procedimientos	31
6.2 Desarrollo de Actividades	34
7. Cronograma	67
8. Conclusiones	68
9. Referencias	73



Introducción

Alta disponibilidad en Aplicaciones Web, procederemos a diseñar una estructura basada en microservicios para garantizar la alta disponibilidad en las aplicaciones web.

Esto implica diseñar una arquitectura de servicios en red, robusta y segura que nos permita garantizar la entrega de un servicio web continuo, optimizado y sin interrupciones.

Se continuará explicando paso a paso, la estructura implementada, los servidores y entornos utilizados, servicios de red que los componen, temporización y control del proyecto, requisitos de software, hardware y humanos, además de incluir el presupuesto.

Finalmente veremos como realizar pruebas de control y verificar que todo este funcionando correctamente, además de incluir instrucciones para su correcto mantenimiento.



Contexto

Las Pequeñas y Medianas Empresas (PYMES) que sus ingresos dependen de una aplicación web, necesitan una estructura web robusta para poder ofrecerles a sus clientes el mejor servicio existente.

Para estas empresas un corte de servicio en su aplicación web durante horas ,podrían significar miles y miles de euros. Por eso vamos a asegurarnos de que esto no ocurra.

Este proyecto esta dirigido principalmente a PYMES que sus ingresos dependen de una aplicación web, es decir:

Empresas con un alto volumen de ventas online diarias (+1.000 pedidos en un día) y que necesitan una solida estructura web para conseguir las ventas.

Empresas que ofrecen servicios B2B a través de una aplicación web (Ejemplos: CRM online, ticketing online, Software en la nube...)

La solución que propondremos será crear una estructura de alta disponibilidad basada en microservicios fiable y escalable que le permita a estas empresas despreocuparse de la parte informatica.



Vamos a definir el marco legal Español que aplica para las PYMES empresas relacionadas con este proyecto:

1- Obligaciones Fiscales:

- **Impuesto sobre Sociedades:** Las empresas en España deben pagar este impuesto sobre los beneficios obtenidos. La tasa general es del 25%.
- **IVA (Impuesto sobre el Valor Añadido):** Las ventas de servicios, como la implementación de soluciones tecnológicas, están sujetas al IVA. La tasa estándar es del 21%.
- **Retenciones e Ingresos a Cuenta:** Se deben realizar retenciones sobre los pagos a empleados, autónomos y otros proveedores de servicios.

2- Obligaciones Laborales:

- **Contratos de Trabajo:** Deben ser adecuados a la normativa laboral española, asegurando derechos y beneficios adecuados a los empleados.
- **Seguridad Social:** La empresa debe inscribir a sus empleados en la Seguridad Social y realizar las cotizaciones correspondientes.
- **Convenios Colectivos:** Dependiendo del sector, puede haber convenios colectivos que regulen las condiciones laborales.



3- Prevención de Riesgos Laborales:

- **Ley de Prevención de Riesgos Laborales (Ley 31/1995):** Esta ley obliga a las empresas a garantizar la seguridad y salud de los trabajadores, implementando medidas preventivas adecuadas.
- **Evaluación de Riesgos:** Identificación de riesgos asociados con el entorno laboral y adopción de medidas para mitigarlos.
- **Formación y Equipos de Protección:** Proveer formación continua en seguridad y equipos de protección personal cuando sea necesario.

Respecto a las ayudas y subvenciones:

Las subvenciones nacionales de España suelen estar orientadas a PYMES, en nuestro caso no aplicarían este tipo de ayudas.

Las subvenciones autonómicas: Cada comunidad autónoma tiene sus propias ayudas y subvenciones para fomentar la innovación tecnológica. Por ejemplo, en Cataluña, la agencia ACCIÓ ofrece diversas ayudas.

A nivel Europeo tenemos a FEDER (Fondo Europeo de Desarrollo Regional): Apoya proyectos de innovación y desarrollo tecnológico.



Razones para la Realización del Proyecto:

El presente proyecto se justifica por la necesidad creciente de garantizar la disponibilidad, seguridad y rendimiento de los servicios web ofrecidos por organizaciones en un entorno cada vez más digitalizado y competitivo. La importancia de la alta disponibilidad en servidores web radica en su capacidad para asegurar que los servicios estén siempre accesibles para los usuarios, incluso en situaciones de alta demanda o fallos en los sistemas.

- **Garantía de Servicio Continuo:** Las interrupciones en los servicios web pueden tener un impacto significativo en la reputación y la satisfacción del cliente, es imperativo implementar soluciones que aseguren la disponibilidad continua de los servicios.
- **Mejora de la Experiencia del Usuario:** La alta disponibilidad no solo garantiza que los servicios estén siempre disponibles, sino que también contribuye a una experiencia del usuario fluida y satisfactoria al evitar tiempos de inactividad y retrasos en la carga de páginas.
- **Seguridad de los Datos:** La separación de los servicios en diferentes equipos y la implementación de medidas de seguridad, como la ubicación de los servidores de bases de datos en entornos protegidos, ayudan a mitigar riesgos de seguridad y proteger la integridad de los datos de los usuarios.
- **Optimización del Rendimiento:** La distribución de carga, el uso de caché SQL y el almacenamiento distribuido contribuyen a optimizar el rendimiento de los servicios web al reducir la carga en los servidores y mejorar los tiempos de respuesta.



Beneficiarios del Proyecto:

Los beneficiarios directos de este proyecto incluyen a las organizaciones que dependen de servicios web para sus operaciones comerciales, así como a los usuarios finales que acceden a estos servicios. Entre los beneficiarios se encuentran:

- **Empresas y Organizaciones (PYMES):** Obtendrán una infraestructura de servicios web más robusta y confiable, lo que les permitirá mantener la continuidad operativa y brindar una experiencia de usuario de alta calidad.
- **Usuarios Finales:** Experimentarán una mayor disponibilidad y velocidad en el acceso a los servicios web, lo que mejorará su satisfacción y fidelidad hacia las plataformas y aplicaciones ofrecidas.



Limitaciones:

A pesar de los esfuerzos por implementar una infraestructura de alta disponibilidad en servidores web. Es importante tener en cuenta estas limitaciones durante la planificación y ejecución del proyecto, y tomar medidas proactivas para mitigar su impacto en la efectividad y el éxito del proyecto.

- **Recursos Financieros:** El presupuesto asignado para el proyecto puede limitar la adquisición de hardware, software y servicios profesionales necesarios para implementar una infraestructura de alta disponibilidad completa y robusta.
- **Recursos Humanos:** La disponibilidad de personal capacitado y experimentado en la configuración, gestión y mantenimiento de una infraestructura de alta disponibilidad puede ser limitada. La falta de personal capacitado puede afectar la eficiencia y efectividad del proyecto.
- **Escalabilidad:** La capacidad de la infraestructura para escalar y adaptarse a cambios en la demanda y el tráfico de usuarios puede verse limitada por factores como la capacidad de los servidores, la capacidad de almacenamiento y la capacidad de la red.
- **Tiempo de Implementación:** El tiempo necesario para diseñar, implementar y probar la infraestructura de alta disponibilidad puede ser limitado por restricciones de tiempo y plazos de entrega, lo que puede afectar la calidad y la efectividad del proyecto.



Conceptos Fundamentales:

- **Alta Disponibilidad:** Se refiere a la capacidad de un sistema o servicio para permanecer accesible y operativo durante períodos de tiempo prolongados.
- **Balanceo de Carga:** Es una técnica utilizada para distribuir el tráfico de red entre varios servidores, con el fin de mejorar la capacidad de respuesta, la disponibilidad y la escalabilidad de los servicios web.
- **Caché SQL:** Es una técnica utilizada para almacenar en memoria caché los resultados de consultas SQL frecuentes, con el fin de mejorar el rendimiento y reducir la carga en los servidores de bases de datos.
- **Web Caché:** Almacenan temporalmente el contenido estático y dinámico de las respuestas web, de modo que las solicitudes repetidas se pueden servir rápidamente sin necesidad de consultar a los servidores de aplicaciones cada vez.

Conceptos Fundamentales:

- **Nginx:** Es un servidor web y proxy inverso ligero, de alto rendimiento y código abierto, ampliamente utilizado para el balanceo de carga y la gestión de tráfico web.
- **Apache HTTP Server:** Es un servidor web de código abierto ampliamente utilizado, conocido por su estabilidad, flexibilidad y capacidad para alojar una amplia variedad de sitios web y aplicaciones.



Conceptos Fundamentales:

- **Memcached:** Es un sistema de almacenamiento en caché de alto rendimiento, utilizado para almacenar en memoria caché datos de acceso frecuente, como consultas SQL y objetos de aplicación, con el fin de mejorar el rendimiento y la escalabilidad de las aplicaciones web.
- **MySQL:** Es un sistema de gestión de bases de datos relacional de código abierto, utilizado para almacenar y gestionar datos estructurados en aplicaciones web y sistemas de información.
- **Infraestructura de Red:** Se establecerá una infraestructura de red adecuada para garantizar la conectividad y comunicación entre los diferentes componentes del sistema. Esto incluye la configuración de switches, routers y servidores para crear una red local confiable y segura.
- **Arquitectura de Servidores Virtuales:** Se definirá la arquitectura de servidores virtuales en VirtualBox, determinando la distribución y configuración de los diferentes servicios y aplicaciones en los servidores virtuales. Se establecerán políticas de asignación de recursos y configuración de redes para optimizar el rendimiento y la disponibilidad de los servicios web.



Objetivo General:

Implementar una infraestructura de alta disponibilidad orientada a microservicios en aplicaciones web que asegure la continuidad operativa y optimice la experiencia del usuario.

Objetivos Específicos:

- **Configurar un servidor web con balanceo de carga:** Esto permitirá distribuir el tráfico de manera eficiente entre los servidores web, asegurando una distribución equitativa de las solicitudes y mejorando la disponibilidad del servicio.
- **Implementar servidores para alojar las aplicaciones web:** Utilizaremos servidores Apache2 para servir el contenido de las aplicaciones web de manera rápida y segura, garantizando un acceso eficiente a los usuarios.
- **Utilizar sistemas de caché Web:** Para mejorar aún más la velocidad y la eficiencia del sistema, se implementan servidores de caché web Nginx delante de los servidores de aplicaciones. Estos servidores de caché almacenan temporalmente el contenido estático y dinámico de las respuestas web, de modo que las solicitudes repetidas se pueden servir rápidamente sin necesidad de consultar a los servidores de aplicaciones cada vez.



Objetivos Específicos:

- **Implementar un sistema de caché SQL:** Se configurará un servidor caché SQL: Memcached, para almacenar en caché las consultas SQL más frecuentes, reduciendo el tiempo de respuesta y la carga en los servidores de bases de datos.
- **Desplegar un servidor de base de datos:** Se configurará un servidor MySQL para gestionar y almacenar los datos de las aplicaciones web de manera segura y eficiente.

Resultados Esperados:

Al finalizar el proyecto, se espera obtener una infraestructura de alta disponibilidad orientada a microservicios en aplicaciones web que cumpla con los requisitos de disponibilidad, seguridad y rendimiento establecidos, garantizando una experiencia del usuario óptima y segura.



Definición del problema

Punto de partida:

Supongamos que una organización proporciona un sitio web (Tienda Online) a través de la URL:

<https://rincon.2asir.net/shop>

Se aproximan tiempos de facturación y ventas (Black Friday) y la empresa nos contacto por que quieren asegurarse de que todo funcionara correctamente.

- Quieren evitar caidas en red (Saturación de peticiones).
- Quieren evitar tiempos de espera en la página web (Latencia).
- Quieren evitar descordinación de datos (Carritos de compra, cuentas de usuario...).

Para garantizar una entrega de servicio correcta y profesional nos aseguraremos de implantar balanceadores de Carga y servidores Web Caché y SQL Caché para garantizar una entrega rápida de datos.

Para garantizar la actividad continua de los servidores, implantaremos un balanceador de carga para que no sature un único servidor, si no que distribuya el trafico conforme el servidor tenga menos carga de trabajo.

Para evitar una descordinación de datos, nos aseguraremos de crear una base de datos robusta y configurarla para que los servidores de aplicaciones web (Apache2) puedan acceder y manejar esos datos.

Por ultimo, pero no menos importante, la instalación de la propia tienda Online en los servidores Apache2.

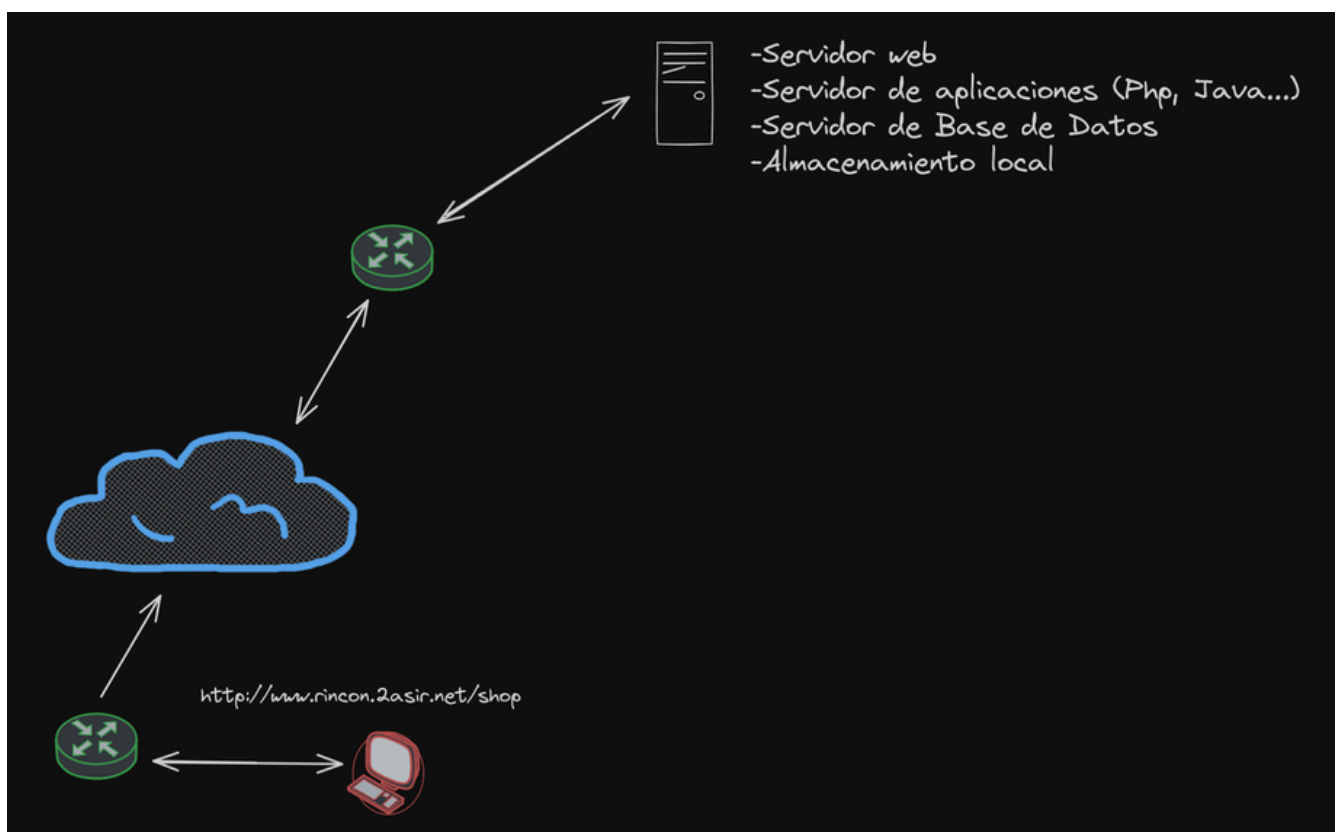
3. PROPUESTA DE SOLUCIÓN



Resolución del problema

Vamos a ir planteando soluciones de menos a mas, hasta crear una estructura solida de Alta disponibilidad.

En un esquema inicial, pensaríamos que proporcionar estos cuatro servicios de red, en un único servidor, satisfacería las necesidades del problema. Pero, desde un punto de la seguridad y la optimización, lo correcto, no sería tenerlo todo, en un mismo servidor, por eso mismo vamos a ir desglosando el problema, junto a sus soluciones.



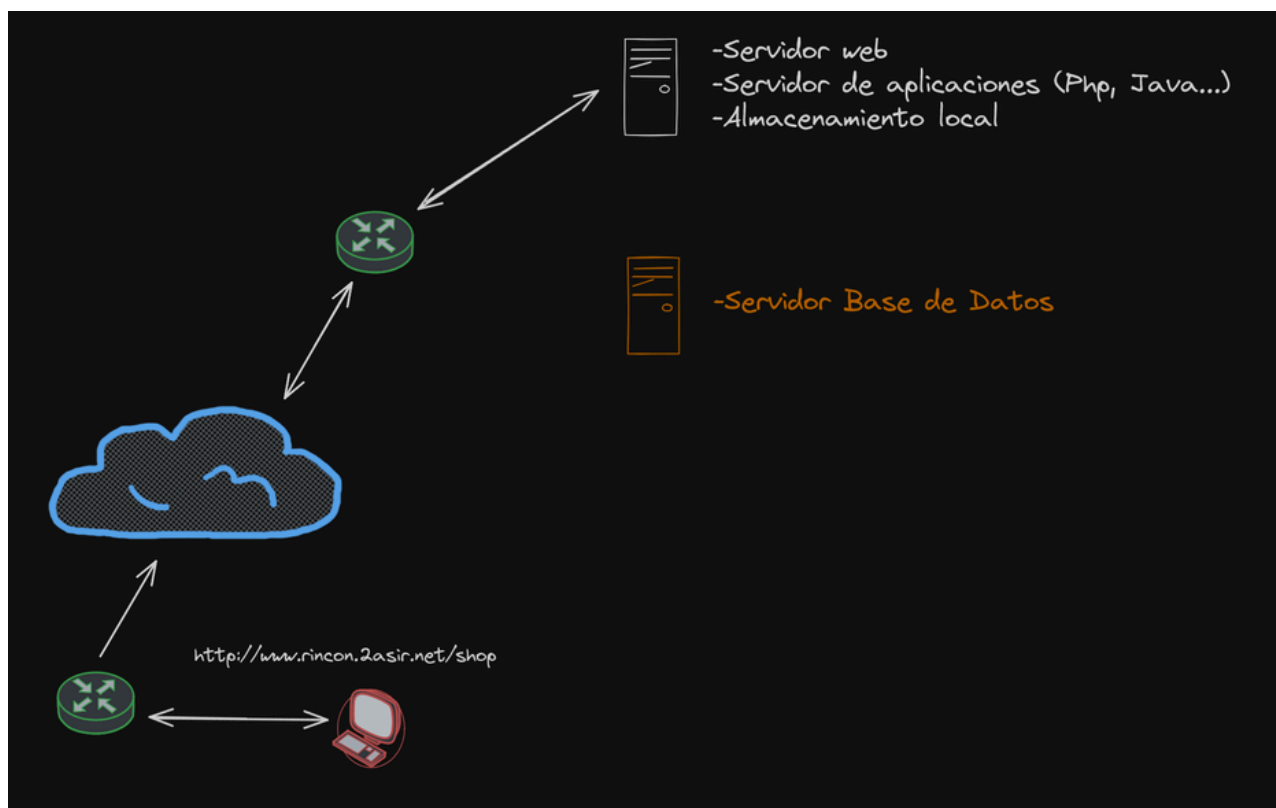
3. PROPUESTA DE SOLUCIÓN



Servidor de bases de datos separado:

Desde un punto de vista de seguridad, ubicar el servidor de bases de datos en el mismo equipo que el servidor web es totalmente inadecuado, ya que el servidor web, por su propia naturaleza debe permitir que cualquier usuario acceda desde Internet y una vulnerabilidad en este equipo podría exponer los datos que se ubican en las bases de datos a un potencial atacante. Además, desde el punto de vista del rendimiento y la disponibilidad, separar los servicios en diferentes equipos hace que no haya interacciones entre ellos y no compitan por los mismos recursos.

Utilizaremos un servidor MySQL como servidor de base de datos, podríamos haber utilizado alternativas como MariaDB, PostgreSQL... Nos decantamos por MySQL, es el más expandido, liviano, sencillo de implantar y con la configuración adecuada es robusto y útil.



3. PROPUESTA DE SOLUCIÓN

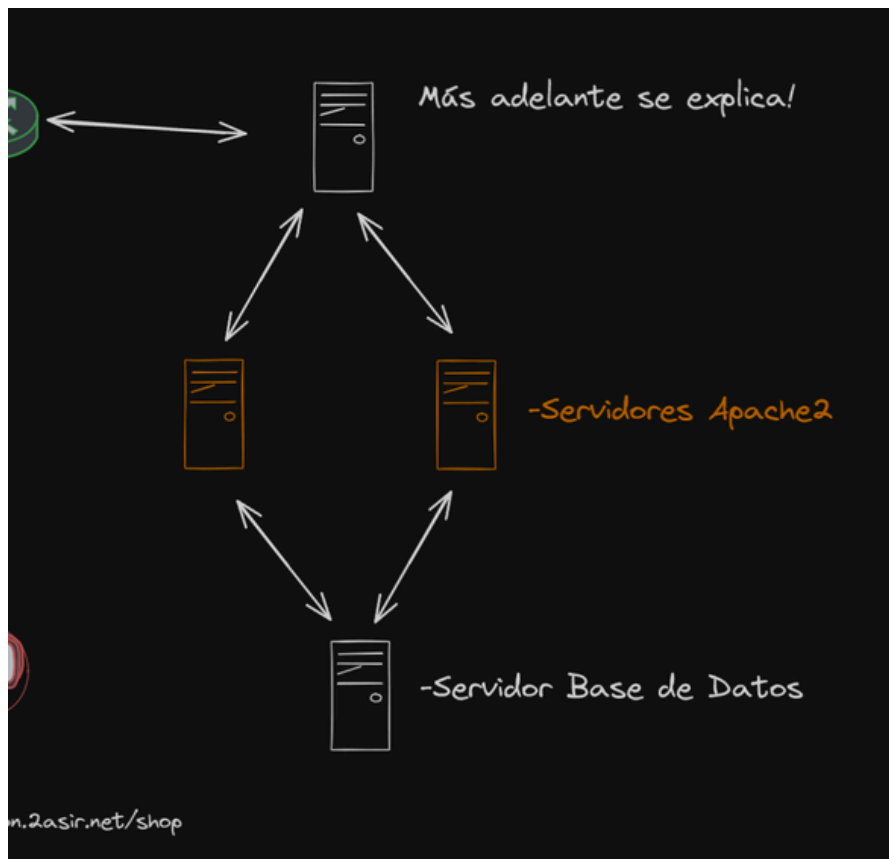


Servidores de aplicaciones en equipos separados:

El coste computacional de una tienda online con un elevado número de solicitudes puede ser considerablemente alto. Para evitar caídas de red y la pérdida de datos, es fundamental tomar medidas adecuadas que aseguren la estabilidad y eficiencia del sistema. Una estrategia eficaz para abordar este problema es la instalación de dos servidores Apache2. Al implementar múltiples servidores, se obtiene una mayor capacidad y recursos, lo que permite distribuir la carga de trabajo de manera equilibrada.

De esta forma, se reduce el riesgo de sobrecargar un solo servidor, lo que a su vez minimiza la posibilidad de interrupciones en el servicio.

Utilizaremos Apache2 para alojar la aplicación web y así probar la compatibilidad entre diferentes tecnologías, es un servicio web robusto y funcional para el caso. Podríamos haber utilizado Nginx perfectamente, pero en este caso y para probar diferentes tecnologías, me decanto por Apache2.



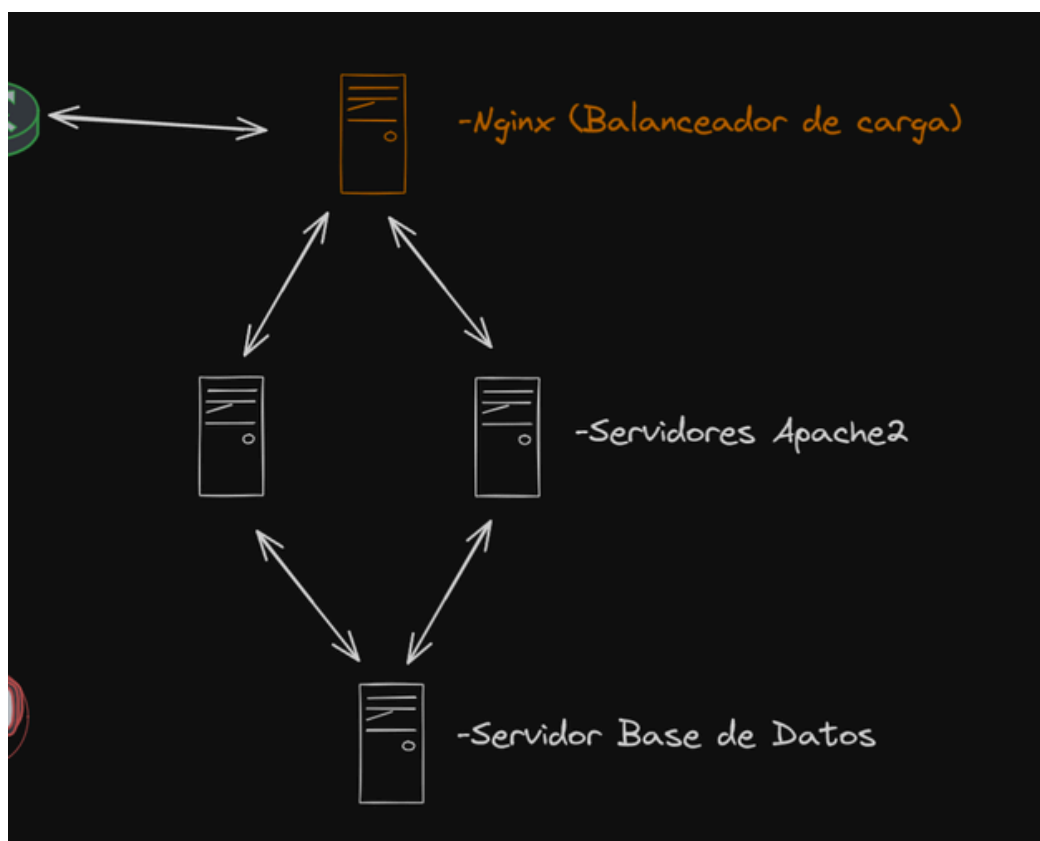
3. PROPUESTA DE SOLUCIÓN



Balanceador de carga:

Para manejar un elevado número de solicitudes y asegurar la disponibilidad y eficiencia del sistema, es fundamental utilizar un balanceador de carga. El balanceador de carga Nginx distribuye las solicitudes entrantes entre los diferentes servidores de aplicaciones (Apache2), asegurando que ninguna máquina se sobrecargue y optimizando el uso de los recursos disponibles. Esto no solo mejora el rendimiento general del sistema, sino que también proporciona redundancia; si uno de los servidores falla, el balanceador puede redirigir el tráfico a los servidores restantes, garantizando así la continuidad del servicio.

Nginx nos permite utilizar un balanceador de carga, simple y fácil de configurar. Para el proyecto, lo único que necesitamos es: No saturar un servidor, y eso Nginx lo permite. Si quisieramos utilizar Balanceadores de carga más potentes podríamos usar: HaProxy o SquidProxy.



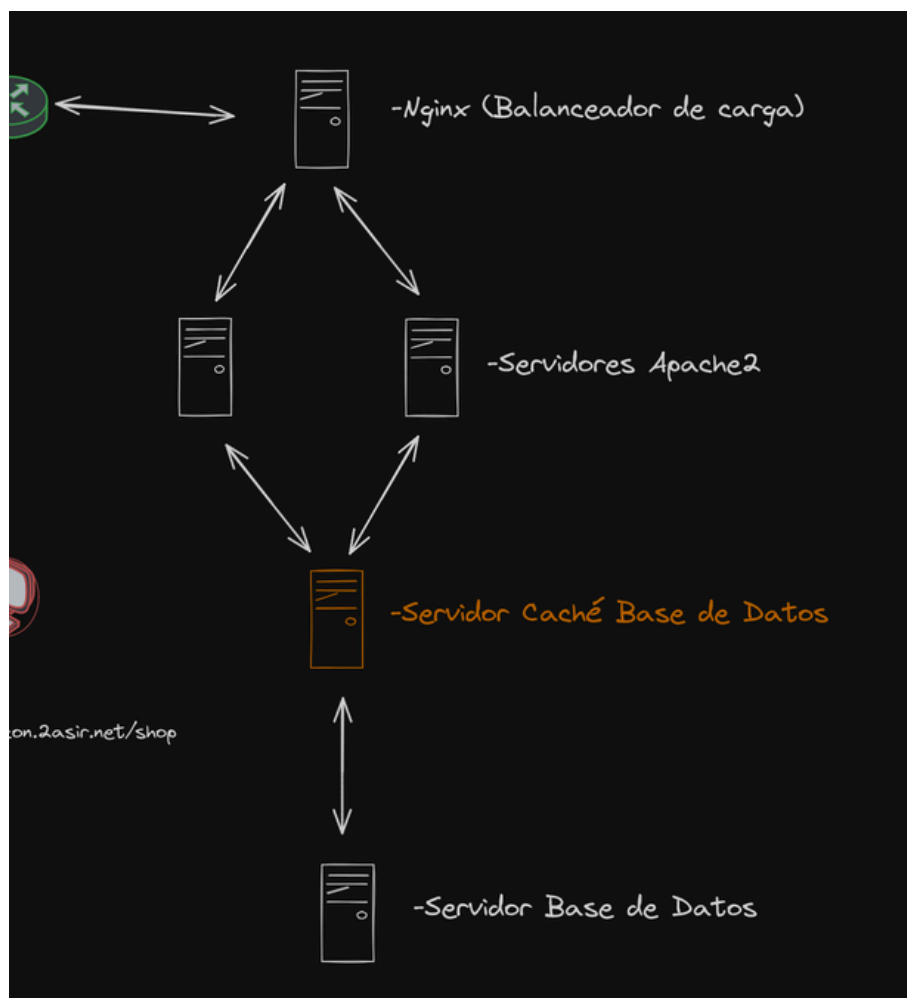
3. PROPUESTA DE SOLUCIÓN



Caché SQL:

Los servidores de aplicaciones consultan continuamente a los servidores de bases de datos y cada consulta conlleva un importante coste computacional y una ralentización de la respuesta. Si la misma consulta ya se ha realizado antes, se puede acelerar mucho la velocidad de respuesta con menor coste computacional utilizando un servicio de caché SQL, de manera que los servidores de aplicaciones se configuran para consultar al servidor caché, que servirá directamente la respuesta si ya lo ha hecho anteriormente, o consultará al servidor de bases de datos en caso necesario.

Utilizaremos Memcached como caché SQL, simplemente por que es compatible con el servidor MySQL, es bastante rápido y sencillo de configurar.



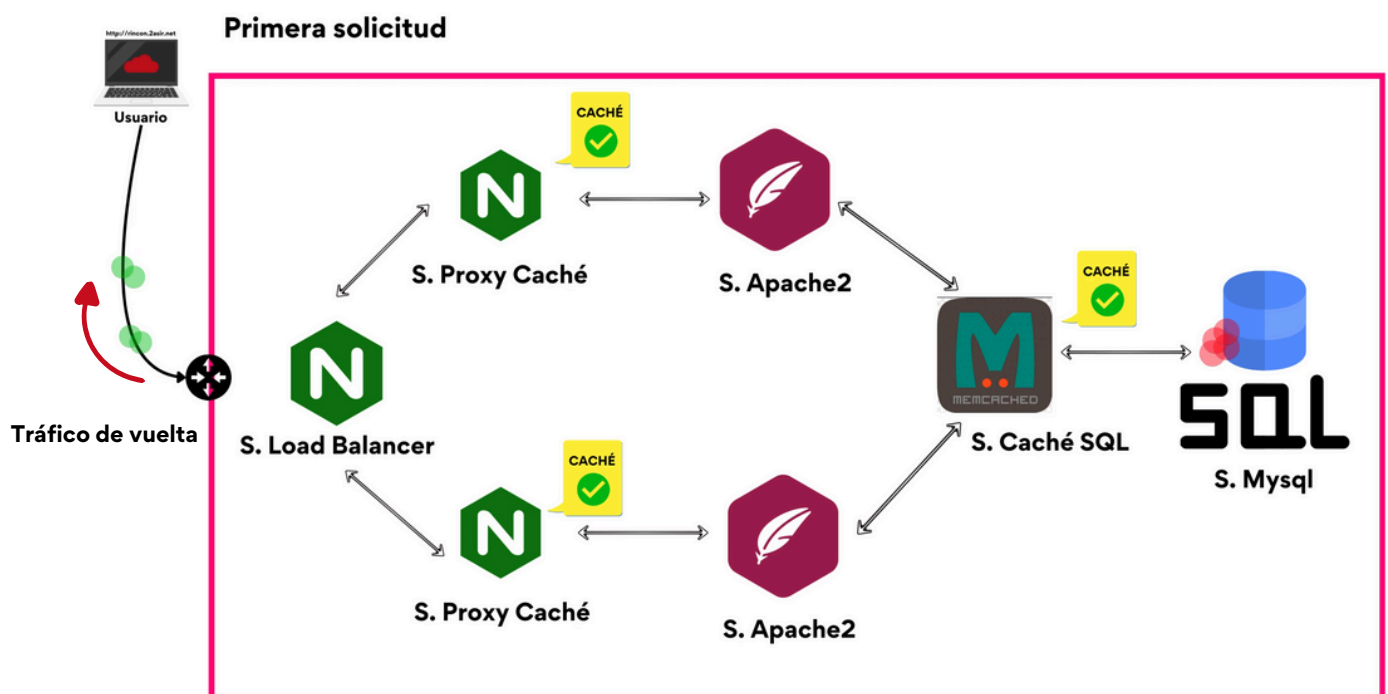
3. PROPUESTA DE SOLUCIÓN



Servidor Web Caché:

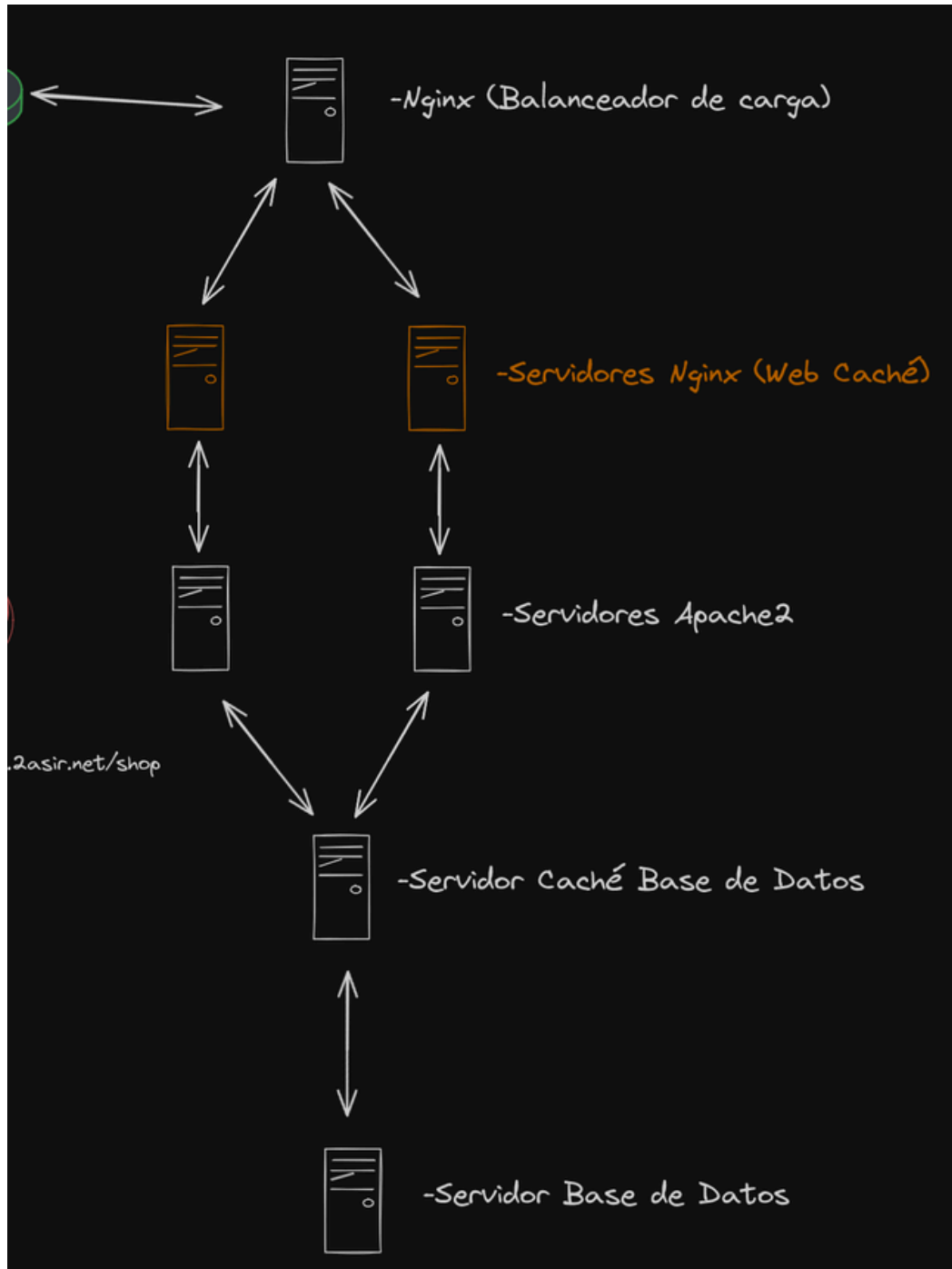
Para mejorar aún más la velocidad y la eficiencia del sistema, se implementan servidores de caché web Nginx delante de los servidores de aplicaciones. Estos servidores de caché almacenan temporalmente el contenido estático y dinámico de las respuestas web, de modo que las solicitudes repetidas se pueden servir rápidamente sin necesidad de consultar a los servidores de aplicaciones cada vez. Esto reduce significativamente la carga en los servidores de aplicaciones y acelera el tiempo de respuesta para los usuarios finales. Al tener servidores de caché dedicados, se optimiza el uso de los recursos y se mejora la escalabilidad del sistema, lo que permite manejar un mayor volumen de tráfico sin degradar el rendimiento.

Nginx, configurado como un Proxy Caché es la solución perfecta para nuestro esquema, seguimos manteniendo una estructura liviana de servidores y además incluye todas las configuraciones necesarias de caché. Otra alternativa sería Varnish Caché.



*Esta imagen también es válida para explicar el caso de Caché SQL (Paso 5)

3. PROPUESTA DE SOLUCIÓN

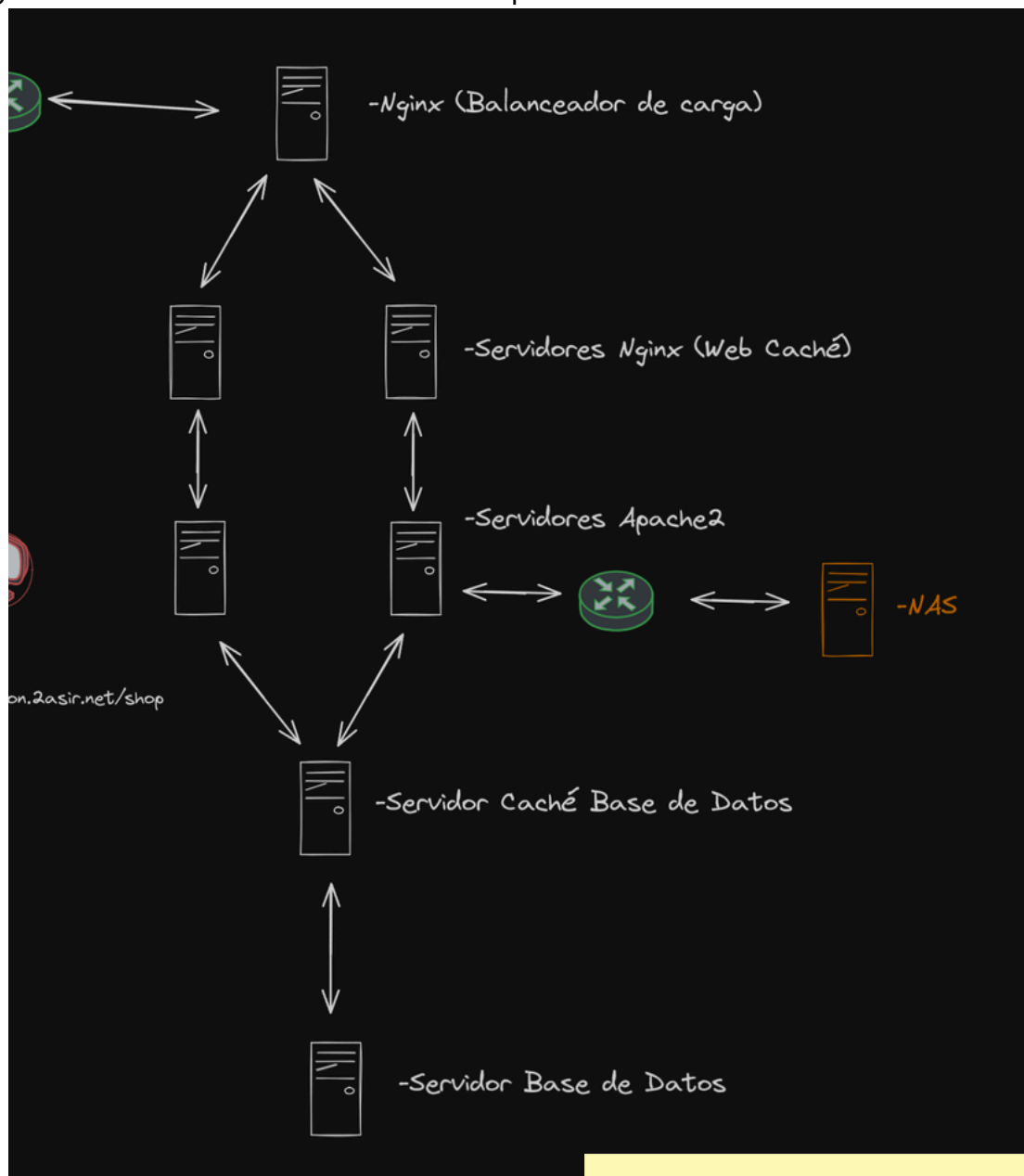


3. PROPUESTA DE SOLUCIÓN



Almacenamiento Distribuido:

El almacenamiento entre los servidores de aplicación de la misma aplicación tiene que estar distribuido de forma que garantice el uso concurrente y se deben repartir las peticiones a los diferentes servidores de aplicación a través de un balanceador de carga. Por eso mismo implantaremos un servidor NAS (Network Attached Storage) para asegurar el almacenamiento de las aplicaciones web.



Aclaración importante:

Almacenamiento Distribuido, se entiende como una propuesta de mejora, por eso mismo no estará incluida en las explicaciones e implementaciones posteriores.

Rincon

4. ENTORNO DE DESPLIEGUE

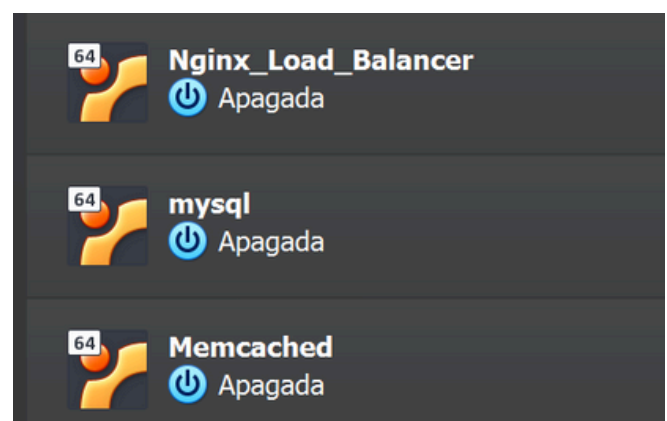


Entorno de desarrollo (Virtualbox):

Para instalar los servicios de red y realizar las pruebas de rendimiento, crearemos un entorno de desarrollo con Virtualbox, simulando cada máquina virtual como un contenedor de proxmox.

Además utilizaremos este entorno de desarrollo para desplegar y probar el proyecto.

Para desplegar el proyecto en un entorno local, utilizamos VirtualBox simulando cada contenedor de proxmox como una máquina virtual, conectandolos todos a la misma red para que tengan conexión y poder realizar las comprobaciones desde un cliente.



Entorno de producción (Proxmox):

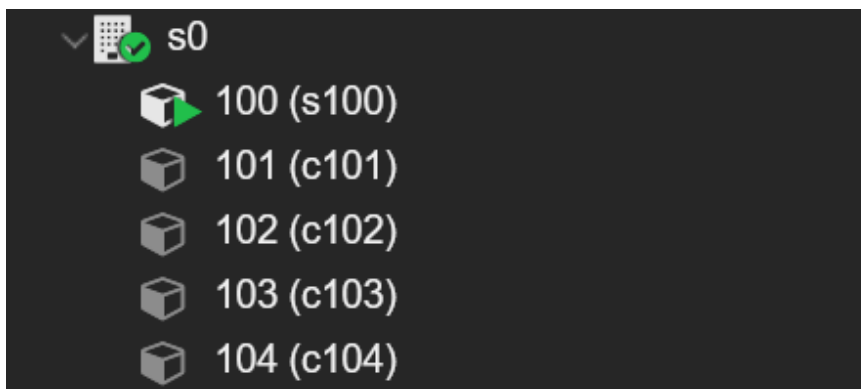
Este proyecto está pensado para desplegarlo en un entorno de virtualización como Proxmox.

Proxmox, es un software de virtualización que nos permite tener un cluster de servidores y contenedores dentro de ellos para asegurar un sistema de Alta disponibilidad.



Este es un ejemplo de como utilizar proxmox para desplegar el proyecto, simulando cada servidor con un contenedor. Todos los contenedores estarán en la misma red local asegurando la conexión entre ellos.

Un entorno semejante a este:



Despliegue del proyecto en una empresa:

En una empresa sería prácticamente igual, instalaríamos nuestro servidor proxmox y comenzariamos a configurarlo.

Lo más optimo sería tener el servidor en una sala adecuada para el mismo, donde exista un control de la temperatura y limpieza. Tambien sería conveniente instalar SAls para evitar cortes de luz.

Acceso al entorno:

Los administradores de red accederán al entorno de producción y equipos a través de una VPN configurada en el router de la compañía de internet.



5.1 Requisitos mínimos

Describiremos los requisitos optimos que necesita un servidor proxmox para asegurar una entrega de servicio sin cortes ni interrupciones y así realizar un pedido de hardware, acorde con lo que necesitamos y ajustar el presupuesto lo máximo posible.

Servidor Proxmox: Procesador 8 nucleos, 12 hilos. 64GB RAM. 500GB SSD m.2. (Componentes adicionales: Caja, fuente de alimentación, placa base... Simplemente buscar la compatibilidad).

5.2 Presupuesto:

A continuación, se presenta una estimación detallada de los costos asociados con hardware, software, servicios profesionales y mantenimiento.

Hardware:

- **Servidor Proxmox:**
 - Procesador 8 nucleos, 12 hilos -> 460€
 - 64GB RAM -> 150€
 - SSD M.2 500GB -> 155€
 - Componentes adicionales como: Placa base, caja, fuente de alimentación, cables, ventiladores, disipador. -> 320€
 - Costo estimado: 1.085€



5.2 Presupuesto:

Software:

- **Licencias de Software:**

- Se necesitarán licencias de software para el sistema operativo (Linux) de los servidores, así como para las aplicaciones y herramientas utilizadas, como Nginx, Apache, MySQL y Memcached.
- Costo estimado: Open Source ;)

Servicios Profesionales:

La contratación de servicios profesionales de consultoría para el diseño e implementación de la infraestructura, así como la capacitación del personal, son aspectos críticos para el éxito del proyecto. Se estima que los costos totales de servicios profesionales ascienden a aproximadamente 2.000€.

Mantenimiento:

El costo de mantener la infraestructura operativa y garantizar su disponibilidad a largo plazo mediante contratos de soporte técnico también debe ser considerado. Se estima que los costos totales de mantenimiento ascienden a aproximadamente 1.000€ anuales.

Presupuesto Total Aproximado:

Sumando los costos estimados de hardware, software, servicios profesionales y mantenimiento, se estima que el presupuesto total aproximado para la implementación de la infraestructura de alta disponibilidad en servidores web es de alrededor de 4.085€.



5.3 Seguimiento y control

El seguimiento y control del proyecto es una de las fases más importantes del proyecto, consiste en establecer unos objetivos o banderas (flags) a corto, medio y largo plazo. Para conocer el estado del proyecto y saber si esta avanzando como esperaba, poder realizar optimizaciones de tiempo, cambios en el proyecto...

Para este proyecto de Alta disponibilidad en Aplicaciones Web en un entorno real y empresarial, daría un tiempo de 4 meses para completar el proyecto.

Verificando cada apartado, podemos asegurarnos que el proyecto va avanzando bien en una relación tiempo - producción.

MES 1: Planificación y pedido.

Durante el primer mes planificaremos todo lo necesario para luego ponernos manos a la obra.

Planificación de red: Crear una topología de red acorde a las direcciones IP necesarias.

Planificación de servicios en red : Decidir que servicios en red vamos a utilizar para cumplir con las necesidades del proyecto y por qué. Incluye la evaluación de requisitos de hardware.

Pedido Hardware: Una vez evaluado todo lo anterior, vamos a necesitar pedir el equipo de hardware por internet para poder realizar el proyecto.



MES 2: Instalación y configuración básica:

Instalación física: Una vez recibido el pedido, necesitaremos comenzar a instalar el equipo físico. Revisando el flujo de ventilación.

Continuaremos con el cableado, conectando los equipos a tomas de corriente y los cables de Ethernet al switch.

Una vez todo instalado realizaremos pruebas de “luz” para asegurarnos que todo este instalado correctamente a la corriente.

Configuración de servidor proxmox: Ya instalado el equipo, procederemos a instalar proxmox y a crear todos los contenedores necesarios para instalar posteriormente los servicios de red. Una vez creados los contenedores, vamos a configurar los nombres de los equipos y direcciones IP estáticas a todos ellos.

Pruebas de conexión de red: Una vez todo conectado y pre configurado, podemos realizar las primeras pruebas de conexión ping entre toda la estructura de red para asegurarnos que haya conectividad en todos los puntos.

MES 3: Instalación y configuración de servicios:

Instalación y configuración de servicios: Comenzaremos a instalar los servicios en el orden que se indican en los procedimientos.

Durante todo el mes la única actividad será: Aseguraremos de dejar todos los servicios bien instalados y configurados.

Es complejo realizarlo todo a la primera, por eso dejaremos margen suficiente para completarlo.



MES 4: Pruebas y rendimiento.

Pruebas y evaluación de rendimiento: Se realizarán todo tipo de pruebas entre los servicios de red para asegurar el correcto funcionamiento entre toda la estructura, entre ellas:

1- Asegurarnos del correcto funcionamiento del balanceador de carga.

(Comprobando que realmente distribuye el tráfico entre los dos servidores caché web)

2- Asegurarnos del correcto funcionamiento del caché web.

(Comprobando que cachea la respuesta)

3- Asegurarnos del correcto funcionamiento del servidor Apache2

(Comprobando que muestre los resultados al visitar la página web)

4- Asegurarnos del correcto funcionamiento del caché Memcached.

(Comprobando que cachea la respuesta)

5- Asegurarnos del correcto funcionamiento de la base de datos

MySQL. (Comprobando que podemos realizar consultas y modificaciones de datos)

Evaluación de rendimiento: Una vez comprobado que todo está OK. Solo revisar algunos parámetros como:

1- Temperaturas y exigencia que reciben los servidores. (Comprobar que todo está en niveles normales, en caso contrario evaluar un cambio de ventiladores, cambiar algún componente...)

2- Latencia en tiempos de entrega: Comprobar cuánto tarda la estructura en devolver la página web. ¿Hay puntos de mejora?, ¿Se pueden optimizar tiempos en servidores caché web o Memcached, Balanceadores de carga, servidores Apache2)

Hay que verificar muy bien estas métricas y compararlas con estructuras y entregas similares.



5.3.1 Mantenimiento del proyecto: No solo es necesario instalar toda la arquitectura, si no mantenerla durante el tiempo.

Para asegurarnos se un correcto funcionamiento de los equipos realizaremos los siguientes requerimientos almenos una vez al mes.

Mantenimiento físico: Mantenimiento del hardware, pasar físicamente a limpiar los equipos solo por el exterior, principalmente conectores y rendijas de ventilación para evitar el máximo polvo posible.

Cada 6 meses, realizar una limpieza a fondo (Incluido interior)

Cada 2 años, cambiar pasta termica del procesador.

Mantenimiento Software: Actualizar continuamente el S.O Linux, además de los propios servicios.

Seguir realizando comprobaciones:

1- Temperaturas y exigencia que reciben los servidores. (Comprobar que todo este en niveles normales, en caso contrario evaluar un cambio de ventiladores, cambiar algún componente...)

2- Latencia en tiempos de entrega: Comprobar cuanto tarda la estructura en devolver la página web. ¿Hay puntos de mejora?, ¿Se pueden optimizar tiempos en servidores caché web o Memcached, Balanceadores de carga, servidores Apache2)

Hay que verificar muy bien estás metricas y compararlas con estructuras y entregas similares.



6.1 PROCEDIMIENTOS:

Procedimiento 1: Planificación de la Topología de Red y Configuración del entorno de producción.

- **Planificación de la Topología de Red Local:** Identificar los requisitos de direccionamiento IP y diseñar una topología de red que garantice la disponibilidad, seguridad y rendimiento de los servicios web.
- **Configuración de VirtualBox (Instalación local):** Instalar y configurar el software de virtualización VirtualBox en los servidores físicos designados para alojar los servidores virtuales. Configurar la red en VirtualBox para reflejar la topología de red planificada, incluida la asignación de direcciones IP.
- **Configuración de entorno empresarial:** Instalaremos todos los equipos físicos en una sala de servidores, dentro de un rack apropiado para ellos. Conectaremos las tomas de luz y cableado correctamente. Posteriormente nos aseguraremos de que el rack este ubicado dentro de la sala de servidores en una posición que favorezca la ventilación del mismo, además como recomendación sería conveniente instalar sistemas SAI para evitar cortes de luz en el servicio.

Procedimiento 2: Despliegue de Servidores Apache2 para Alojar las Aplicaciones Web.

- Instalar y configurar los servidores Apache2 en servidores dedicados.
- Configurar los virtual hosts para alojar las aplicaciones web en cada servidor Apache2.
- Programar código PHP que va a contener el servidor de aplicaciones.



Procedimiento 3: Despliegue de un Servidor MySQL para Almacenamiento de Bases de Datos.

- Instalar y configurar el servidor MySQL en un servidor dedicado.
- Crear y configurar las bases de datos necesarias para las aplicaciones web en el servidor MySQL.
- Importante, asignar permisos a los servidores Apache para acceder a las Bases de Datos.

Procedimiento 4: Configuración del Servidor Nginx con Balanceo de Carga.

- Instalar y configurar el servidor Nginx en el servidor principal.
- Configurar el balanceo de carga para distribuir el tráfico entre los servidores Apache.

Procedimiento 5: Implementación de un Servidor Memcached para Caché SQL.

- Instalar y configurar el servidor Memcached en un servidor dedicado.
- Configurar los servidores Apache2 para utilizar el servidor Memcached como caché SQL para consultas frecuentes a la base de datos.



Procedimiento 6: Configuración de servidores Caché Web.

- Instalar servidor Nginx y configurarlo como caché de manera individual para cada servidor Apache.
- Replicar el servidor para cada Servidor Apache individual.

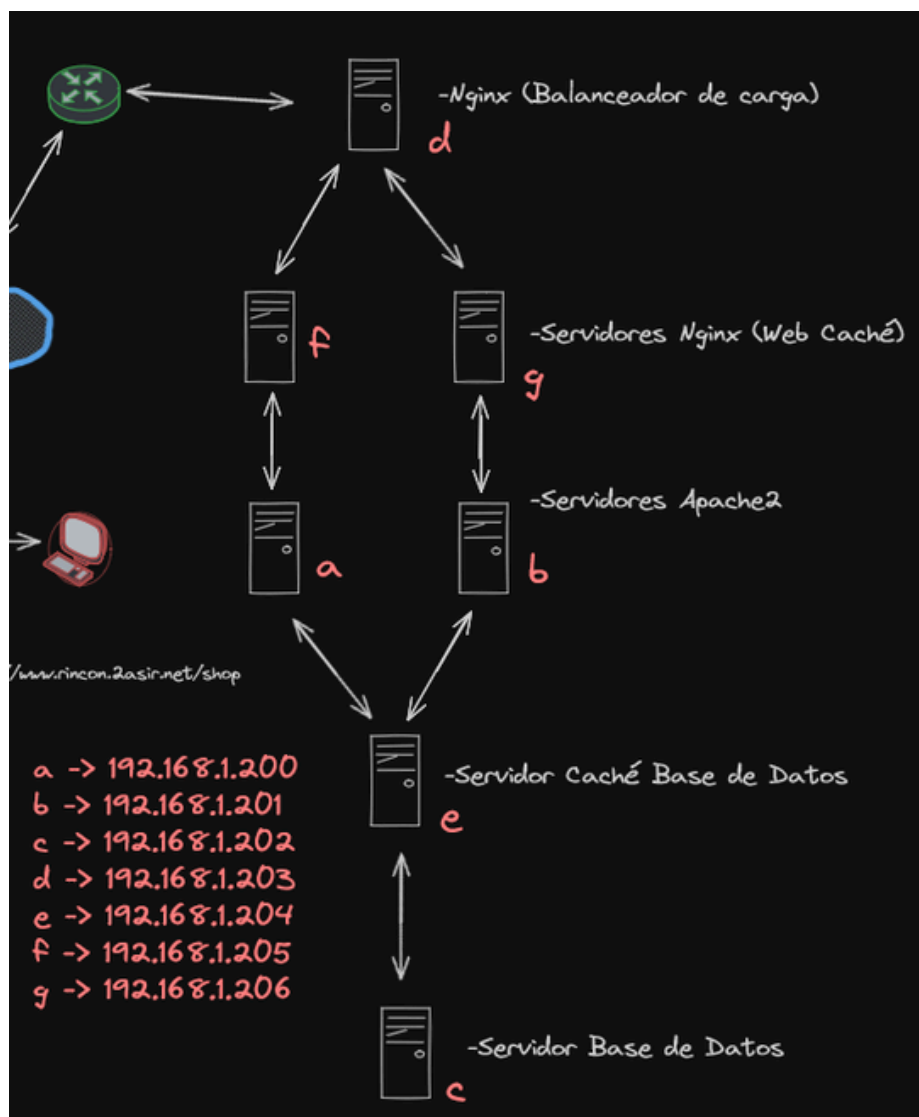
Procedimiento 7: Pruebas y Ajustes de la Infraestructura:

- Realizar pruebas de rendimiento para evaluar la capacidad de la infraestructura para manejar cargas de trabajo y tráfico de usuarios.
- Identificar y corregir posibles problemas de configuración o rendimiento mediante ajustes en la infraestructura.
- Documentar los resultados de las pruebas y los ajustes realizados para futuras referencias.



Planificación de la Topología de Red y Configuración de Virtual Box.

Para planificar la topología de Red, simplemente vamos a asignar direcciones IP a nuestros servidores:



En un entorno real, deberíamos utilizar una máscara de red acorde con el número de direcciones IP que vamos a utilizar: En este caso la máscara de red ideal para 7 servidores y dejando espacio para algún servidor de ampliación:

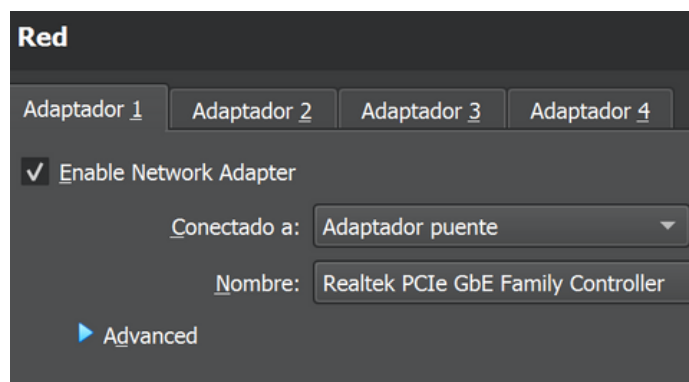
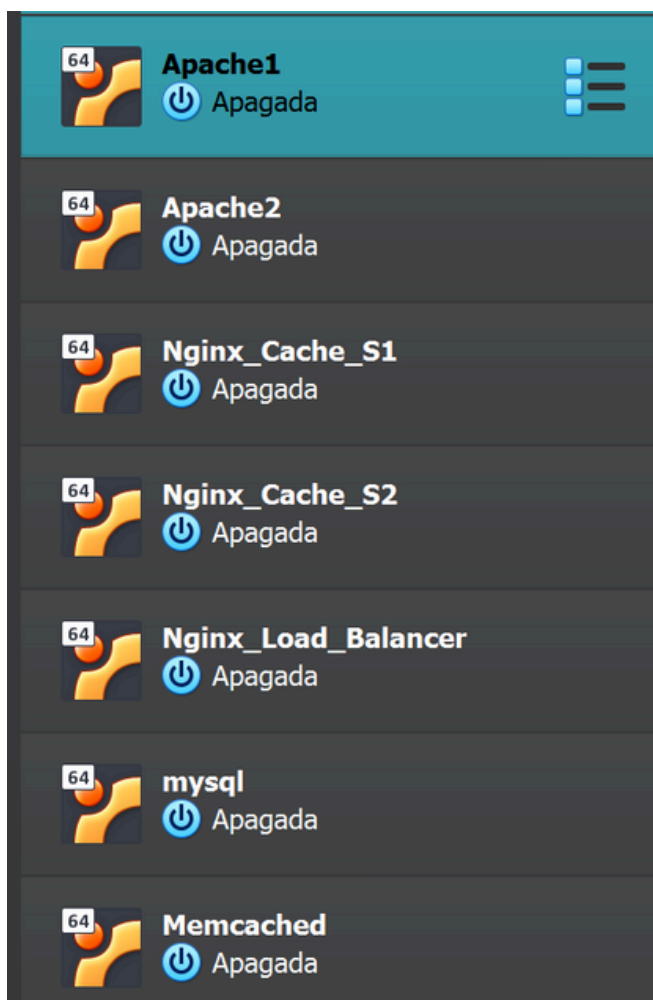
Máscara de red: /28 -> 14 hosts.



Configurar VirtualBox.

Configuración de VirtualBox: Instalar y configurar el software de virtualización VirtualBox en los servidores físicos designados para alojar los servidores virtuales. Configurar la red en VirtualBox para reflejar la topología de red planificada, incluida la asignación de direcciones IP.

En mi caso, por comodidad y para poder acceder desde la maquina anfitrion como “cliente”, utilizare la configuración de red: Adaptador puente. Igualmente VirtualBox permite muchisimas y diferentes configuraciones.





Cambio de nombre y dirección IP de cada servidor:

A cada equipo se le cambiará su nombre modificando el archivo: /etc/hosts y /etc/hostname.

```
GNU nano 6.2 /etc/hosts
127.0.0.1 localhost
127.0.1.1 apache1

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0  ip6-localnet
ff00::0  ip6-mcastprefix
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters
```

```
GNU nano 6.2 /etc/hostname
apache1
```

Además se le asignará la configuración de una dirección IP privada.
/etc/netplan/00-installer-config.yaml

```
GNU nano 6.2 /etc/netplan/00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      addresses:
        - 192.168.1.200/24
      nameservers:
        addresses:
          - 8.8.8.8
        search:
          - 8.8.8.8
      routes:
        - to: default
          via: 192.168.1.1
  version: 2
```



Instalación servidor Apache2:

Instalamos el servidor Apache2 con todas sus dependencias: `sudo apt install apache2 php libapache2-mod-php php-mysql`

Habilitamos el sitio web: `sudo a2ensite /etc/apache2/sites-available/000-default.conf`

Configuramos el sitio web en `/var/www/html/index.php`

```
rincon@apache1:~$ sudo apt install apache2 php libapache2-mod-php php-mysql
[sudo] password for rincon:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
libapache2-mod-php ya está en su versión más reciente (2:8.1+92ubuntu1).
php ya está en su versión más reciente (2:8.1+92ubuntu1).
php-mysql ya está en su versión más reciente (2:8.1+92ubuntu1).
apache2 ya está en su versión más reciente (2.4.52-1ubuntu4.9).
```

```
rincon@apache1:/etc/apache2/sites-available$ sudo a2ensite 000-default.conf
Site 000-default already enabled
rincon@apache1:/etc/apache2/sites-available$ |
```

```
rincon@apache1:/etc/apache2/sites-available$ ls -lia /var/www/html/
total 20
 35419 drwxr-xr-x  3 root root 4096 may 20 17:19 .
 35418 drwxr-xr-x  3 root root 4096 mar 22 15:51 ..
291252 drwxr-xr-x  2 root root 4096 abr  9 10:53 backEnd
  1251 -rw-r--r--  1 root root 2774 may 13 16:47 index.php
 34868 -rw-r--r--  1 root root 2318 may 13 12:20 index.php.bak
rincon@apache1:/etc/apache2/sites-available$ |
```

```
rincon@apache1:/etc/apache2/sites-available$ sudo systemctl restart apache2.service
rincon@apache1:/etc/apache2/sites-available$ sudo systemctl status apache2.service
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2024-05-30 15:25:07 UTC; 4s ago
     Docs: https://httpd.apache.org/docs/2.4/
```

Instalación segundo servidor Apache2:

Simplemente tendríamos que clonar la máquina de VirtualBox, Asignar un nuevo nombre y dirección IP al equipo y verificar que todo este OK.

6.2 DESARROLLO ACTIVIDADES



Instalación servidor MySQL:

Instalamos el servidor MySQL con: `sudo apt install mysql-server -y`

Entramos dentro de Mysql con: `sudo mysql -u root` y configuramos la base de datos.

(Aquí como configuremos la base de datos es irrelevante, lo importante es asignar permisos a los servidores Apache, para que tengan acceso y puedan modificar, solicitar datos de la base de datos)

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| rincon |
| sys |
+-----+
8 rows in set (0,01 sec)
```

```
Database changed
mysql> show tables;
+-----+
| Tables_in_rincon |
+-----+
| asignaturas |
| notas |
+-----+
2 rows in set (0,00 sec)

mysql> |
```

```
mysql> select * from asignaturas;
+-----+-----+
| id_asignatura | asignatura |
+-----+-----+
| 1 | Base de Datos |
| 2 | Sistemas |
| 3 | Seguridad Informática |
| 4 | Servicios en Red |
| 5 | Aplicaciones Web |
| 6 | Programación Python |
+-----+-----+
6 rows in set (0,00 sec)

mysql> select * from notas;
+-----+-----+-----+
| id_nota | id_asignatura | nota |
+-----+-----+-----+
| 1 | 1 | 6.00 |
| 2 | 2 | 6.00 |
| 3 | 3 | 6.00 |
| 4 | 4 | 7.00 |
| 5 | 5 | 6.00 |
| 6 | 6 | 6.00 |
+-----+-----+-----+
6 rows in set (0,00 sec)
```

```
rincon@mysql: /etc/mysql/co  x  +  v

mysql> CREATE USER 'remoto'@'192.168.1.200' IDENTIFIED BY 'Rincon2003-';
Query OK, 0 rows affected (0,02 sec)

mysql> GRANT ALL PRIVILEGES ON *.* TO 'remoto'@'192.168.1.200' WITH GRANT OPTION;
Query OK, 0 rows affected (0,01 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0,01 sec)

mysql>

mysql> UPDATE mysql.user SET Host='%' WHERE User='remoto' AND Host='192.168.1.200';
Query OK, 1 row affected (0,01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```



IIIMPORTANTE hacerlo dos veces una con la IP 200 y otra con la 201.

Así asignaremos permisos a ambos servidores Apache2.

Por ultimo en el archivo de configuración de mysql.conf , necesitamos comentar esta línea de blind para permitir conexiones todas conexiones exteriores.

```
rincon@mysql: /etc/mysql/mi × + v
GNU nano 6.2
# * Basic Settings
#
user                = mysql
# pid-file           = /var/run/mysqld/mysq
# socket             = /var/run/mysqld/mysq
# port               = 3306
# datadir            = /var/lib/mysql

# If MySQL is running as a replication
# changed. Ref https://dev.mysql.com/d
# tmpdir             = /tmp
#
# Instead of skip-networking the defau
# localhost which is more compatible a
# bind-address      = 127.0.0.1
mysqlx-bind-address = 127.0.0.1
#
# * Fine Tuning
#
key_buffer_size     = 16M
# max allowed packet = 64M
```

```
rincon@mysql:~$ sudo systemctl status mysql.service
[sudo] password for rincon:
● mysql.service - MySQL Community Server
   Loaded: loaded (/lib/systemd/system/mysql.service;
   Active: active (running) since Thu 2024-05-30 15:3
   Process: 643 ExecStartPre=/usr/share/mysql/mysql-sv
```



Instalación servidor Nginx, Balanceador de carga:

Instalamos nginx: `sudo apt install nginx`

Configuramos el archivo de configuración: `/etc/nginx/conf.d/load-balancing.conf`

Voy a explicar el código:

Upstream Backend → Definir grupo de servidores. UPSTREAM (Directiva) ,
BACKEND (Nombre del conjunto de servidores)

Directiva server: Definir un servidor virtual que escuchará por el puerto 80 y que aceptará las peticiones a este nombre de servidor (En mi caso he puesto una ip, como podría haber puesto `rincon.net`)

Directiva Location: Todas las solicitudes que vayan a esa URL (En este caso la BARRA HORIZONTAL indican todas las solicitudes)

La regla de proxy redirect → Parece insignificativa, pero es muy importante. Se asegura de que la solicitud entre del cliente y la respuesta del servidor cache sea la misma. Por que puede ocurrir que el cliente pida A. El servidor devuelva A y el servidor caché devuelva B. Entonces para evitar una mala intención del usuario o un error entre servidores. Pues utilizamos esto.

Las reglas de Proxy header → Se encargan de transmitir toda la información de las cabeceras al servidor final, Una se encarga de la IP, Otra de las ips aunque utilice un proxy, Otro para almacenar la información del host.

Por ultimo el **proxy pass** se utiliza para enviar las solicitudes al grupo de servidores backend que son los de arriba.

El balanceo de carga por defecto que utiliza es conocido como “**Round-Robin**” que al final lo que hace es. La primera solicitud al servidor A, la segunda solicitud al servidor B, la tercera al A, la cuarta al B...



```
GNU nano 6.2 /etc/nginx/conf.d/load-balancing.conf
upstream backend {
    server 192.168.1.205;
    server 192.168.1.206;
}

server {
    listen 80;
    server_name 192.168.1.203;

    location / {
        proxy_redirect off;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_pass http://backend;
    }
}
```

Posible mejora: Utilizando el metodo de balanceo de carga "least-connection" lograríamos que el trafico se distribuyera al servidor con menos carga en ese momento. Por lo cual para este proyecto valdría bastante la pena. Lo adjunto como posible mejora ;) La configuración sería algo semejante a esto:

Rincon

```
upstream backend {
    least_conn;
    server backend1.example.com;
    server backend2.example.com;
}
```

6.2 DESARROLLO ACTIVIDADES



Configuración servidor Memcached: Para instalar memcached utilizaremos el comando: `sudo apt install libmemcached11 -y` , junto a sus dependencias: `sudo apt install php-memcached -y`

```
rincon@memcached:~$ sudo apt install libmemcached11 -y; sudo apt install php-memcached -y
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
libmemcached11 ya está en su versión más reciente (1.0.18-4.2ubuntu4).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 15 no actualizados.
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
php-memcached ya está en su versión más reciente (3.1.5+2.2.0-14.1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 15 no actualizados.
rincon@memcached:~$ |
```

Ahora vamos a configurar el único archivo que necesitamos para que funcione Memcached: `sudo nano /etc/memcached.conf`

El archivo de memcached es simple, solo tenemos que asegurarnos de cambiar la dirección IP y poner la del servidor que utilizará memcached. (opción -l) , las demás opciones las tendremos que dejar por defecto.

```
# Start with a cap of 64 megs of memory. It's reasonable, and the daemon default
# Note that the daemon will grow to this size, but does not start out holding this much
# memory
-m 64

# Default connection port is 11211
-p 11211

# Run the daemon as root. The start-memcached will default to running as root if no
# -u command is present in this config file
-u memcache

# Specify which IP address to listen on. The default is to listen on all IP addresses
# This parameter is one of the only security measures that memcached has, so make sure
# it's listening on a firewalled interface.
-l 192.168.1.204
```

```
27 sudo iptables -A INPUT -p tcp --dport 11211 -j ACCEPT
28 sudo iptables-save > /etc/iptables/rules.v4
29 sudo systemctl restart iptables
30 sudo systemctl restart memcached.service
```




Configuración de Iptables: Vamos a configurar ip tables para que acepte las solicitudes entrantes y así que los servidores externos como Apache2 y MySQL puedan acceder al servicio Memcached.

Con iptables, agregamos una nueva línea de regla (-A INPUT), por el protocolo tcp (-p) , por el puerto 11211 (--dport) , aceptaremos todo tipo de trafico entrante si cumple los requisitos (-j)

```
27 sudo iptables -A INPUT -p tcp --dport 11211 -j ACCEPT
28 sudo iptables-save > /etc/iptables/rules.v4
29 sudo systemctl restart iptables
30 sudo systemctl restart memcached.service
```

Por ultimo, reiniciar servicio y aplicar los cambios.

```
rincon@memcached:~$ sudo systemctl status memcached.service
[sudo] password for rincon:
● memcached.service - memcached daemon
   Loaded: loaded (/lib/systemd/system/memcached.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2024-05-31 18:26:42 UTC; 42min ago
     Docs: man:memcached(1)
   Main PID: 643 (memcached)
   CGroup: /systemd/system/memcached.service
           └─ 643 /usr/sbin/memcached
```



Configuración de servidor web caché: El servidor web caché lo instalaremos en Nginx, por lo cual lo primero será instalar el servicio: `sudo apt install nginx -y`

Abriremos el archivo `/etc/nginx/nginx.conf` y Añadimos únicamente la línea de `proxy_cache_path`: Para indicar algunas configuraciones como la ruta donde se va a almacenar la caché y algunos ajustes básicos como el tiempo de caché.

Las demás configuraciones de dejan por defecto.

```
GNU nano 6.2 /etc/nginx/nginx.conf
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
}

http {
    proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=my_cache:10m max_size=10g inactive=60m;

    sendfile on;
    tcp_nopush on;
    types_hash_max_size 2048;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: P00DLE
    ssl_prefer_server_ciphers on;

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    gzip on;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```



Configuración de servidor web caché: Ahora vamos al archivo de /etc/nginx/sites-availables/default

Aquí Definimos un servidor virtual con el puerto 80 y el nombre del servidor.

Con **location** definimos que todo lo que pase por esa URL que en este caso son todas, el **Proxy pass** reditige todo el trafijo a un servidor original apache2.

Abajo tenemos todas las directivas y configuraciones de caché.

→ Nombre que va a recibir la configuración caché

→ Tiempo de vida de caché

La ultima linea es opcional, pero no sirve a nosotros para comprobar que realmente este cacheando las solicitudes

```
GNU nano 6.2
server {
    listen 80;
    server_name rincon.2asir.net;

    location / {
        proxy_pass http://192.168.1.200;
        proxy_cache my_cache;
        proxy_cache_valid 200 302 10m;
        proxy_cache_valid 404 1m;
        proxy_cache_use_stale error timeout updating http_500 http_502 http_503 http_504;
        proxy_cache_background_update on;
        proxy_cache_lock on;

        # Agrega esta línea para incluir el encabezado X-Cache-Status
        add_header X-Cache-Status $upstream_cache_status;
    }
}
```

Por ultimo recordar aplicar el enlace simbolico para habilitar el sitio (default) -> sudo ln -s /etc/nginx/sites-available/default /etc/nginx/sites-enabled/

***IMPORTANTE:** Recordar que como son dos servidores Nginx_Web_Caché , necesitamos clonar la máquina y crear otra cambiando, dirección IP y nombre de la máquina Caché, además de sustituir la dirección IP de destino 192.168.1.200 a 192.168.1.201 (Redirigir el trafico al Segundo servidor Apache2)



Por ultimo, reiniciar servicio y aplicar cambios:

```
rincon@cache:~$ sudo systemctl status nginx.service
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2024-05-31 18:58:05 UTC; 9min ago
     Docs: man:nginx(8)
```



Una vez con todos los servicios instalados, vamos a programar el código PHP para que la estructura de alta disponibilidad tenga sentido.

Aclarar que la estructura del código sigue el patron (MVC) Modelo Vista Controlador.

El MVC lo utilizamos para no crear un archivo index.php gigante, sin embargo así conseguimos optimizar al máximo los archivos y ganar en tiempo de carga.



La lógica de programación de estos archivos PHP es la siguiente:

index.php -> Programa principal y muestrario de datos. Es decir, aquí controlaremos el flujo principal de datos, conectando con los archivos de BackEnd y mostrando los resultados de los mismos.

Control_MySQL.php -> Conectamos la sesión con MySQL y realizamos la consulta que posteriormente mostraremos en index.php

Control_Memcached.php -> Configuramos la sesión con Memcached (Dirección IP y puertos).

6.2 DESARROLLO ACTIVIDADES



Archivo index.php (parte 1)

```
GNU nano 6.2 /var/www/html/index.php
<?php
// Ejecutar el comando curl -I a nivel de consola y capturar la salida
$curl_command = 'curl -I http://192.168.1.205';
$curl_output = shell_exec($curl_command);

require_once 'backEnd/Control_MySQL.php';
require_once 'backEnd/Control_Memcached.php';

// Conexión a MySQL
$controlador_mysql = new ControladorMySQL();
$conexion_mysql = $controlador_mysql ? "Conexión con MySQL establecida correctamente." : "Error al establecer conexión con MySQL.";

// Conexión a Memcached
$controlador_memcached = new ControladorMemcached();
$conexion_memcached = $controlador_memcached ? "Conexión con Memcached establecida correctamente." : "Error al establecer conexión con Memcached.";

$data_memcached = $controlador_memcached->obtenerAsignaturasYNotas();

if (!empty($data_memcached)) {
    $data = $data_memcached;
    $mensaje_datos = "Datos obtenidos de Memcached";
} else {
    $data = $controlador_mysql->obtenerAsignaturasYNotas();
    if (!empty($data)) {
        $controlador_memcached->almacenarAsignaturasYNotas($data);
        $mensaje_datos = "Datos obtenidos de MySQL y almacenados en Memcached para futuras consultas.";
    } else {
        $mensaje_datos = "No se encontraron datos.";
    }
}

function imprimirTabla($data) {
    if (!empty($data)) {
        foreach ($data as $row) {
            printf("<tr>
                <td>%s</td>
                <td>%s</td>
            </tr>", htmlspecialchars($row["asignatura"]), htmlspecialchars($row["nota"]));
        }
    } else {
        echo "<p>No se encontraron resultados.</p>";
    }
}

?>
```



Archivo index.php (parte 2)

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Asignaturas y Notas de Miguel Ángel Rincón Bueno</title>
  <style>
    table {
      border-collapse: collapse;
      width: 100%;
    }
    th, td {
      border: 1px solid black;
      padding: 8px;
      text-align: left;
    }
    th {
      background-color: #f2f2f2;
    }
  </style>
</head>
<body>
  <h1>SERVIDOR 1</h1>
  <h3>Asignaturas y notas de "Miguel Ángel Rincón"</h3>

  <?php
  if ($curl_output) {
    echo "<p>Headers del servidor en 192.168.1.205:</p><pre>$curl_output</pre>";
  } else {
    echo "<p>Error al ejecutar el comando cURL.</p>";
  }
  ?>

  <p><?php echo $conexion_mysql; ?></p>
  <p><?php echo $conexion_memcached; ?></p>
  <p><?php echo $mensaje_datos; ?></p>

  <table>
    <tr>
      <th>Asignatura</th>
      <th>Nota</th>
    </tr>
    <?php imprimirTabla($data); ?>
  </table>
</body>
</html>
```



Vamos a explicar el código PHP parte por parte.

index.php

```
GNU nano 6.2 /var/www/html/index.php
<?php
// Ejecutar el comando curl -I a nivel de consola y capturar la salida
$curl_command = 'curl -I http://192.168.1.205';
$curl_output = shell_exec($curl_command);

require_once 'backEnd/Control_MySQL.php';
require_once 'backEnd/Control_Memcached.php';

// Conexión a MySQL
$controlador_mysql = new ControladorMySQL();
$conexion_mysql = $controlador_mysql ? "Conexión con MySQL establecida correctamente." : "Error al establecer conexión con MySQL.";

// Conexión a Memcached
$controlador_memcached = new ControladorMemcached();
$conexion_memcached = $controlador_memcached ? "Conexión con Memcached establecida correctamente." : "Error al establecer conexión con Memcached.";

$data_memcached = $controlador_memcached->obtenerAsignaturasYNotas();
```

En este primer bloque de código tenemos las dos primeras líneas la ejecución y guardado del comando curl.

En la primera línea almacenamos el comando curl (\$curl_command) y en la segunda lo ejecutamos y almacenamos el resultado en la variable \$curl_output

Con “Require_once” incluimos dos archivos de BackEnd. ControlSQL y Memcached.

\$controlador_mysql = new ControladorMySQL(); Crea una instancia de la clase ControladorMySQL.

\$conexion_mysql = \$controlador_mysql ? 'Conexión con MySQL establecida correctamente.' : 'Error al establecer conexión con MySQL.'; Verifica si la instancia se creó correctamente y asigna un mensaje correspondiente a \$conexion_mysql.

\$controlador_memcached = new ControladorMemcached(); Crea una instancia de la clase ControladorMemcached.

\$conexion_memcached = \$controlador_memcached ? 'Conexión con Memcached establecida correctamente.' : 'Error al establecer conexión con Memcached.'; Verifica si la instancia se creó correctamente y asigna un mensaje correspondiente a \$conexion_memcached.

\$data_memcached = \$controlador_memcached->obtenerAsignaturasYNotas(); Llama al método obtener Asignaturas Y Notas de la instancia Controlador Memcached y guarda los datos en \$data_memcached.



index.php

```
if (!empty($data_memcached)) {
    $data = $data_memcached;
    $mensaje_datos = "Datos obtenidos de Memcached";
} else {
    $data = $controlador_mysql->obtenerAsignaturasYNotas();
    if (!empty($data)) {
        $controlador_memcached->almacenarAsignaturasYNotas($data);
        $mensaje_datos = "Datos obtenidos de MySQL y almacenados en Memcached para futuras consultas.";
    } else {
        $mensaje_datos = "No se encontraron datos.";
    }
}

function imprimirTabla($data) {
    if (!empty($data)) {
        foreach ($data as $row) {
            printf("<tr>
                <td>%s</td>
                <td>%s</td>
            </tr>", htmlspecialchars($row["asignatura"]), htmlspecialchars($row["nota"]));
        }
    } else {
        echo "<p>No se encontraron resultados.</p>";
    }
}

?>
```

if (!empty(\$data_memcached)) {: Verifica si \$data_memcached no está vacío.

\$data = \$data_memcached;: Asigna los datos obtenidos de Memcached a la variable \$data.

\$mensaje_datos = 'Datos obtenidos de Memcached';: Asigna un mensaje indicando que los datos se obtuvieron de Memcached.

} else {: Si \$data_memcached está vacío.

\$data = \$controlador_mysql->obtenerAsignaturasYNotas();: Llama al método obtenerAsignaturasYNotas de la instancia ControladorMySQL y guarda los datos en \$data.

if (!empty(\$data)) {: Verifica si \$data no está vacío.



\$controlador_memcached->almacenarAsignaturasYNotas(\$data);:

Almacena los datos obtenidos de MySQL en Memcached para futuras consultas.

\$mensaje_datos = 'Datos obtenidos de MySQL y almacenados en Memcached para futuras consultas.'; Asigna un mensaje indicando que los datos se obtuvieron de MySQL y se almacenaron en Memcached.

} else {: Si \$data está vacío.

\$mensaje_datos = 'No se encontraron datos.'; Asigna un mensaje indicando que no se encontraron datos.

function imprimirTabla(\$data) {: Define una función llamada imprimirTabla que recibe un parámetro \$data.

if (!empty(\$data)) {: Verifica si \$data no está vacío.

foreach (\$data as \$row) {: Itera sobre cada fila de datos.

**printf("<tr><td>%s</td><td>%s</td></tr>",
htmlspecialchars(\$row['asignatura']), htmlspecialchars(\$row['nota']));:**
Imprime una fila de tabla con la asignatura y la nota, escapando caracteres especiales para evitar inyección de código.

} else {: Si \$data está vacío.

echo '<p>No se encontraron resultados.</p>'; Imprime un mensaje indicando que no se encontraron resultados.



index.php

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Asignaturas y Notas de Miguel Ángel Rincón Bueno</title>
  <style>
    table {
      border-collapse: collapse;
      width: 100%;
    }
    th, td {
      border: 1px solid black;
      padding: 8px;
      text-align: left;
    }
    th {
      background-color: #f2f2f2;
    }
  </style>
</head>
<body>
  <h1>SERVIDOR 1</h1>
  <h3>Asignaturas y notas de "Miguel Ángel Rincón"</h3>

  <?php
  if ($curl_output) {
    echo "<p>Headers del servidor en 192.168.1.205:</p><pre>$curl_output</pre>";
  } else {
    echo "<p>Error al ejecutar el comando cURL.</p>";
  }
  ?>

  <p><?php echo $conexion_mysql; ?></p>
  <p><?php echo $conexion_memcached; ?></p>
  <p><?php echo $mensaje_datos; ?></p>

  <table>
    <tr>
      <th>Asignatura</th>
      <th>Nota</th>
    </tr>
    <?php imprimirTabla($data); ?>
  </table>
</body>
</html>
```



<!DOCTYPE html>: Indica el tipo de documento.

<html lang="es">: Define el inicio del documento HTML y especifica el idioma español.

<head>: Comienza la sección de la cabecera del documento HTML.

<meta charset="UTF-8">: Especifica la codificación de caracteres como UTF-8.

<meta name="viewport" content="width=device-width, initial-scale=1.0">: Configura la vista para que sea adaptable a diferentes dispositivos.

<title>Asignaturas y Notas de Miguel Ángel Rincón Bueno</title>: Establece el título de la página.

<style>: Inicia una sección de estilos CSS.

table { ... }: Define estilos para la tabla.

th, td { ... }: Define estilos para las celdas de la tabla.

th { ... }: Define estilos para los encabezados de la tabla.

</style>: Finaliza la sección de estilos CSS.

</head>: Finaliza la sección de la cabecera.

<body>: Comienza el cuerpo del documento HTML.

<h1>SERVIDOR 1</h1>: Encabezado de nivel 1.

<h3>Asignaturas y notas de "Miguel Ángel Rincón"</h3>: Encabezado de nivel 3.



```
<?php
if ($curl_output) {
    echo "<p>Headers del servidor en 192.168.1.205:</p><pre>$curl_output</pre>";
} else {
    echo "<p>Error al ejecutar el comando cURL.</p>";
}
?>

<p><?php echo $conexion_mysql; ?></p>
<p><?php echo $conexion_memcached; ?></p>
<p><?php echo $mensaje_datos; ?></p>

<table>
    <tr>
        <th>Asignatura</th>
        <th>Nota</th>
    </tr>
    <?php imprimirTabla($data); ?>
</table>
</body>
</html>
```

<?php: Inicia el bloque de código PHP dentro del HTML.

if (\$curl_output) {: Verifica si \$curl_output no está vacío, es decir, si el comando cURL se ejecutó correctamente y devolvió alguna salida.

echo "<p>Headers del servidor en 192.168.1.205:</p> , mostrar por pantalla

<pre>\$curl_output</pre>;: Si \$curl_output contiene datos, imprime un mensaje indicando que los encabezados del servidor en 192.168.1.205 fueron obtenidos, seguido de los encabezados en formato preformateado (<pre>).

} else {: Si \$curl_output está vacío, es decir, si el comando cURL no se ejecutó correctamente.

echo "<p>Error al ejecutar el comando cURL.</p>;: Imprime un mensaje indicando que hubo un error al ejecutar el comando cURL.

?>: Finaliza el bloque de código PHP.



<p><?php echo \$conexion_mysql; ?></p>: Dentro de un párrafo HTML, imprime el mensaje almacenado en \$conexion_mysql, que indica el estado de la conexión a MySQL (ya sea "Conexión con MySQL establecida correctamente." o "Error al establecer conexión con MySQL.").

<p><?php echo \$conexion_memcached; ?></p>: Dentro de un párrafo HTML, imprime el mensaje almacenado en \$conexion_memcached, que indica el estado de la conexión a Memcached (ya sea "Conexión con Memcached establecida correctamente." o "Error al establecer conexión con Memcached.").

<p><?php echo \$mensaje_datos; ?></p>: Dentro de un párrafo HTML, imprime el mensaje almacenado en \$mensaje_datos, que indica si los datos fueron obtenidos de Memcached, de MySQL, o si no se encontraron datos.

<table>: Inicia una tabla HTML para mostrar los datos de asignaturas y notas.

<tr>: Inicia una fila de la tabla.

<th>Asignatura</th>: Define una celda de encabezado de la tabla para "Asignatura".

<th>Nota</th>: Define una celda de encabezado de la tabla para "Nota".
</tr>: Finaliza la fila de encabezado.

<?php imprimirTabla(\$data); ?>: Llama a la función imprimirTabla definida anteriormente, pasando los datos obtenidos en la variable \$data. Esta función genera las filas de la tabla para cada asignatura y nota.

</table>: Finaliza la tabla HTML.

</body>: Finaliza el cuerpo del documento HTML.

</html>: Finaliza el documento HTML.



Archivo Control_MySQL.php

```
GNU nano 6.2 /var
<?php
class ConexionMySQL {
    private $link;

    public function __construct($server, $user, $password, $dbname) {
        $this->link = mysqli_connect($server, $user, $password, $dbname);
        if (!$this->link) {
            die("Error de conexión: " . mysqli_connect_error());
        }
    }

    public function query($query) {
        return mysqli_query($this->link, $query);
    }

    public function numRows($result) {
        return mysqli_num_rows($result);
    }

    public function fetchAssoc($result) {
        return mysqli_fetch_assoc($result);
    }

    public function close() {
        mysqli_close($this->link);
    }
}

class ControladorMySQL {
    public function obtenerAsignaturasYNotas() {
        $mysql_server = '192.168.1.202';
        $mysql_user = 'remoto';
        $mysql_password = 'Rincon2003-';
        $mysql_dbname = 'rincon';

        $conexion_mysql = new ConexionMySQL($mysql_server, $mysql_user, $mysql_password, $mysql_dbname);

        $query = "SELECT a.asignatura, n.nota
                  FROM asignaturas AS a
                  INNER JOIN notas AS n ON a.id_asignatura = n.id_asignatura";
        $result = $conexion_mysql->query($query);

        $data = array();
        if ($result && $conexion_mysql->numRows($result) > 0) {
            while ($row = $conexion_mysql->fetchAssoc($result)) {
                $data[] = $row;
            }
        }

        $conexion_mysql->close();

        return $data;
    }
}
?>
```



Control_MySQL.php

```
GNU nano 6.2 /var
<?php
class ConexionMySQL {
    private $link;

    public function __construct($server, $user, $password, $dbname) {
        $this->link = mysqli_connect($server, $user, $password, $dbname);
        if (!$this->link) {
            die("Error de conexión: " . mysqli_connect_error());
        }
    }

    public function query($query) {
        return mysqli_query($this->link, $query);
    }

    public function numRows($result) {
        return mysqli_num_rows($result);
    }

    public function fetchAssoc($result) {
        return mysqli_fetch_assoc($result);
    }

    public function close() {
        mysqli_close($this->link);
    }
}
```

<?php: Indica el inicio del código PHP.

class ConexionMySQL {: Define una clase llamada ConexionMySQL que manejará la conexión a la base de datos MySQL.

private \$link;: Declara una propiedad privada \$link que almacenará el recurso de conexión a MySQL.

public function __construct(\$server, \$user, \$password, \$dbname) {: Define el constructor de la clase, que se ejecuta cuando se crea una instancia de ConexionMySQL. Este constructor toma cuatro parámetros: \$server, \$user, \$password, y \$dbname.

\$this->link = mysqli_connect(\$server, \$user, \$password, \$dbname);: Intenta establecer una conexión a MySQL usando los parámetros proporcionados y almacena el recurso de conexión en \$this->link.

if (!\$this->link) {: Verifica si la conexión falló.

die("Error de conexión: " . mysqli_connect_error());: Si la conexión falla, termina el script y muestra un mensaje de error.



public function query(\$query) {: Define un método público query que toma un parámetro \$query.

return mysqli_query(\$this->link, \$query);: Ejecuta la consulta SQL \$query usando la conexión almacenada en \$this->link y devuelve el resultado.

public function numRows(\$result) {: Define un método público numRows que toma un parámetro \$result.

return mysqli_num_rows(\$result);: Devuelve el número de filas en el resultado \$result.

public function fetchAssoc(\$result) {: Define un método público fetchAssoc que toma un parámetro \$result.

return mysqli_fetch_assoc(\$result);: Devuelve una fila del resultado \$result como un array asociativo.

public function close() {: Define un método público close que cierra la conexión a MySQL.

mysqli_close(\$this->link);: Cierra la conexión a MySQL almacenada en \$this->link.

}; Finaliza la definición de la clase ConexionMySQL.



Control_MySQL.php

```
class ControladorMySQL {
    public function obtenerAsignaturasYNotas() {
        $mysql_server = '192.168.1.202';
        $mysql_user = 'remoto';
        $mysql_password = 'Rincon2003-';
        $mysql_dbname = 'rincon';

        $conexion_mysql = new ConexionMySQL($mysql_server, $mysql_user, $mysql_password, $mysql_dbname);

        $query = "SELECT a.asignatura, n.nota
                  FROM asignaturas AS a
                  INNER JOIN notas AS n ON a.id_asignatura = n.id_asignatura";
        $result = $conexion_mysql->query($query);

        $data = array();
        if ($result && $conexion_mysql->numRows($result) > 0) {
            while ($row = $conexion_mysql->fetchAssoc($result)) {
                $data[] = $row;
            }
        }

        $conexion_mysql->close();

        return $data;
    }
}
```

class ControladorMySQL {: Define una clase llamada ControladorMySQL que manejará las operaciones específicas de la aplicación usando la clase ConexionMySQL.

public function obtenerAsignaturasYNotas() {: Define un método público obtenerAsignaturasYNotas que obtendrá las asignaturas y notas de la base de datos.

\$mysql_server = '192.168.1.202';: Declara una variable \$mysql_server y le asigna la dirección IP del servidor MySQL.

\$mysql_user = 'remoto';: Declara una variable \$mysql_user y le asigna el nombre de usuario para la conexión a MySQL.

\$mysql_password='Rincon2003-';: Declara una variable \$mysql_password y le asigna la contraseña para la conexión a MySQL.

\$mysql_dbname = 'rincon';: Declara una variable \$mysql_dbname y le asigna el nombre de la base de datos.



`$conexion_mysql = new ConexionMySQL($mysql_server, $mysql_user, $mysql_password, $mysql_dbname);` Crea una nueva instancia de la clase `ConexionMySQL` usando los parámetros de conexión definidos anteriormente y la almacena en `$conexion_mysql`.

`$query = "SELECT a.asignatura, n.nota FROM asignaturas AS a INNER JOIN notas AS n ON a.id_asignatura = n.id_asignatura";` Define una consulta SQL que selecciona las asignaturas y sus notas asociadas, uniendo las tablas `asignaturas` y `notas`.

`$result = $conexion_mysql->query($query);` Ejecuta la consulta SQL usando el método `query` de la instancia `ConexionMySQL` y almacena el resultado en `$result`.

`$data = array();` Declara un array vacío `$data` para almacenar los resultados.

`if ($result && $conexion_mysql->numRows($result) > 0) {` Verifica si la consulta se ejecutó correctamente y si el número de filas en el resultado es mayor que cero.

`while ($row = $conexion_mysql->fetchAssoc($result)) {` Itera sobre cada fila del resultado.

`$data[] = $row;` Añade cada fila del resultado al array `$data`.

`$conexion_mysql->close();` Cierra la conexión a MySQL usando el método `close` de la instancia `ConexionMySQL`.

`return $data;` Devuelve el array `$data` que contiene las asignaturas y notas.

`}` Finaliza la definición del método `obtenerAsignaturasYNotas`.

`}` Finaliza la definición de la clase `ControladorMySQL`.

`?>` Finaliza el bloque de código PHP.



Archivo Control_Memcached.php

```
GNU nano 6.2 /var/wi
<?php
class ConexionMemcached {
    private $memcached;

    public function __construct($server, $port) {
        $this->memcached = new Memcached();
        $this->memcached->addServer($server, $port);
    }

    public function get($key) {
        return $this->memcached->get($key);
    }

    public function set($key, $value, $expiration = 3600) {
        return $this->memcached->set($key, $value, $expiration);
    }
}

class ControladorMemcached {
    public function obtenerAsignaturasYNotas() {
        $memcached_server = '192.168.1.204';
        $memcached_port = 11211;

        $conexion_memcached = new ConexionMemcached($memcached_server, $memcached_port);
        $data = $conexion_memcached->get('asignaturas_y_notas');

        return $data;
    }

    public function almacenarAsignaturasYNotas($data) {
        $memcached_server = '192.168.1.204';
        $memcached_port = 11211;

        $conexion_memcached = new ConexionMemcached($memcached_server, $memcached_port);
        $conexion_memcached->set('asignaturas_y_notas', $data, 3600);
    }
}
?>
```



Control_Memcached.php

```
GNU nano 6.2 /var/w
<?php
class ConexionMemcached {
    private $memcached;

    public function __construct($server, $port) {
        $this->memcached = new Memcached();
        $this->memcached->addServer($server, $port);
    }

    public function get($key) {
        return $this->memcached->get($key);
    }

    public function set($key, $value, $expiration = 3600) {
        return $this->memcached->set($key, $value, $expiration);
    }
}
```

<?php: Indica el inicio del código PHP.

class ConexionMemcached {: Define una clase llamada ConexionMemcached que manejará la conexión a Memcached.

private \$memcached;: Declara una propiedad privada \$memcached que almacenará la instancia de Memcached.

public function __construct(\$server, \$port) {: Define el constructor de la clase, que se ejecuta cuando se crea una instancia de ConexionMemcached. Este constructor toma dos parámetros: \$server y \$port.

\$this->memcached = new Memcached();: Crea una nueva instancia de la clase Memcached y la asigna a la propiedad \$this->memcached.

\$this->memcached->addServer(\$server, \$port);: Añade un servidor Memcached utilizando la dirección IP y el puerto proporcionados.

public function get(\$key) {: Define un método público get que toma un parámetro \$key.



return \$this->memcached->get(\$key);: Recupera el valor almacenado en Memcached correspondiente a la clave \$key.

public function set(\$key, \$value, \$expiration = 3600) {: Define un método público set que toma tres parámetros: \$key, \$value, y \$expiration con un valor predeterminado de 3600 segundos (1 hora).

return \$this->memcached->set(\$key, \$value, \$expiration);: Almacena el valor \$value en Memcached con la clave \$key y una duración de \$expiration segundos.



Control_Memcached.php

```
class ControladorMemcached {
    public function obtenerAsignaturasYNotas() {
        $memcached_server = '192.168.1.204';
        $memcached_port = 11211;

        $conexion_memcached = new ConexionMemcached($memcached_server, $memcached_port);
        $data = $conexion_memcached->get('asignaturas_y_notas');

        return $data;
    }

    public function almacenarAsignaturasYNotas($data) {
        $memcached_server = '192.168.1.204';
        $memcached_port = 11211;

        $conexion_memcached = new ConexionMemcached($memcached_server, $memcached_port);
        $conexion_memcached->set('asignaturas_y_notas', $data, 3600);
    }
}
?>
```

class ControladorMemcached {}: Define una clase llamada ControladorMemcached que manejará operaciones específicas usando la clase ConexionMemcached.

public function obtenerAsignaturasYNotas() {}: Define un método público obtenerAsignaturasYNotas que recuperará las asignaturas y notas de Memcached.

\$memcached_server = '192.168.1.204';: Declara una variable \$memcached_server y le asigna la dirección IP del servidor Memcached.

\$memcached_port = 11211;: Declara una variable \$memcached_port y le asigna el puerto del servidor Memcached.

\$conexion_memcached = new ConexionMemcached(\$memcached_server, \$memcached_port);: Crea una nueva instancia de la clase ConexionMemcached usando los parámetros de conexión definidos anteriormente y la almacena en \$conexion_memcached.



\$data = \$conexion_memcached->get('asignaturas_y_notas'); Recupera los datos almacenados en Memcached con la clave 'asignaturas_y_notas' y los almacena en \$data.

return \$data; Devuelve los datos obtenidos de Memcached.
}: Finaliza la definición del método obtenerAsignaturasYNotas.

public function almacenarAsignaturasYNotas(\$data) { Define un método público almacenarAsignaturasYNotas que almacena las asignaturas y notas en Memcached. Este método toma un parámetro \$data.

\$memcached_server = '192.168.1.204'; Declara una variable \$memcached_server y le asigna la dirección IP del servidor Memcached.

\$memcached_port = 11211; Declara una variable \$memcached_port y le asigna el puerto del servidor Memcached.

**\$conexion_memcached = new
ConexionMemcached(\$memcached_server, \$memcached_port);** Crea una nueva instancia de la clase ConexionMemcached usando los parámetros de conexión definidos anteriormente y la almacena en \$conexion_memcached.

\$conexion_memcached->set('asignaturas_y_notas', \$data, 3600); Almacena los datos \$data en Memcached con la clave 'asignaturas_y_notas' y una duración de 3600 segundos (1 hora).

}; Finaliza la definición del método almacenarAsignaturasYNotas.

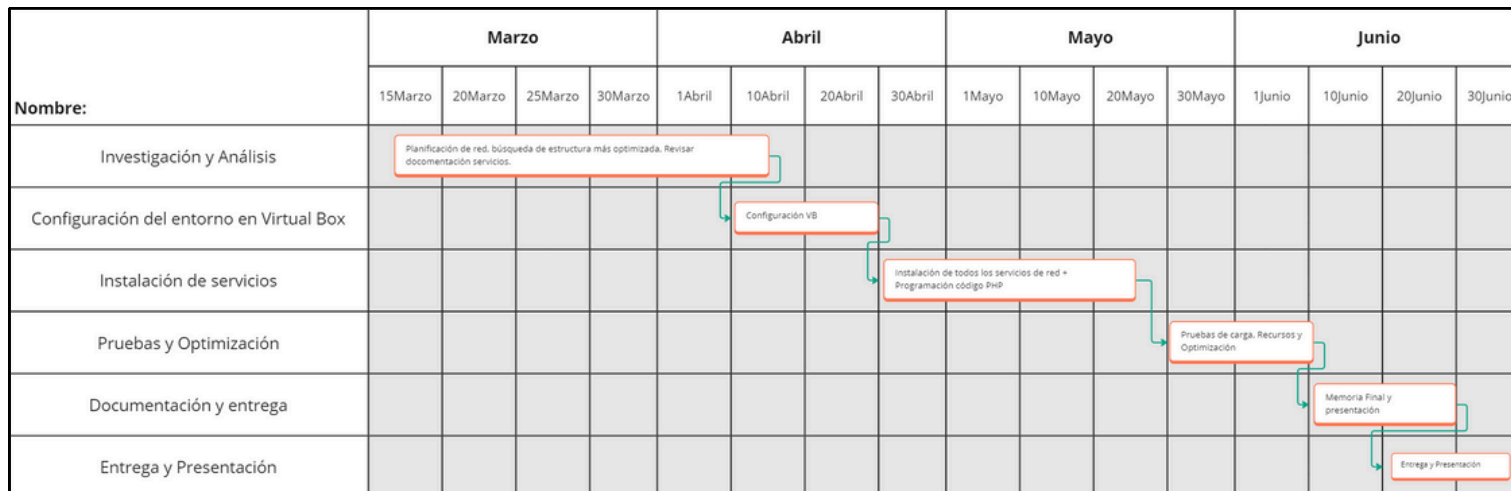
}; Finaliza la definición de la clase ControladorMemcached.

?> Finaliza el bloque de código PHP.

7. CRONOGRAMA



***Es necesario hacer zoom al documento PDF para visualizar correctamente el diagrama de GANT**



De lo contrario, si no es posible su correcta visualización accediendo a este enlace podréis descargar la imagen: [Enlace](#).



Conclusiones del Proyecto

La implementación de una infraestructura de servidores para una aplicación web compleja, como una tienda online, requiere un enfoque cuidadoso y bien planificado para garantizar la seguridad, el rendimiento y la disponibilidad. A través de este proyecto, se ha demostrado cómo la segmentación de servicios y el uso de diversas tecnologías pueden resolver los problemas comunes que enfrentan las aplicaciones web de gran escala.

- **Seguridad Mejorada:** Separar el servidor de bases de datos del servidor web es una práctica fundamental para mejorar la seguridad. Al aislar los datos sensibles en un servidor dedicado y no expuesto directamente a Internet, se reduce significativamente el riesgo de brechas de seguridad que podrían comprometer información crítica de los usuarios.
- **Optimización del Rendimiento:** La instalación de múltiples servidores Apache2 y el uso de un balanceador de carga Nginx han demostrado ser efectivos para distribuir la carga de trabajo de manera equilibrada. Esta configuración no solo mejora el rendimiento al evitar la sobrecarga de un solo servidor, sino que también proporciona redundancia, asegurando que el sistema pueda manejar fallos sin interrupciones significativas en el servicio.
- **Eficiencia a Través de la Caché:** La utilización de servidores de caché web Nginx y Memcached como caché SQL ha permitido una optimización considerable del tiempo de respuesta y la reducción de la carga en los servidores de aplicaciones y bases de datos. Al almacenar en caché las respuestas a consultas repetitivas, el sistema puede servir contenido rápidamente y con menor costo computacional.



- **Escalabilidad y Flexibilidad:** La arquitectura propuesta es altamente escalable. La posibilidad de añadir más servidores Apache2 detrás del balanceador de carga Nginx y de incrementar los recursos de los servidores de caché permite que la infraestructura pueda crecer según las necesidades de la tienda online. Esta flexibilidad asegura que el sistema pueda adaptarse a un aumento en el tráfico y en la demanda sin comprometer la calidad del servicio.
- **Disponibilidad y Redundancia:** La configuración de balanceo de carga y los servidores de caché contribuyen significativamente a la alta disponibilidad del sistema. En caso de fallo de uno de los servidores de aplicaciones, el balanceador de carga puede redirigir las solicitudes a los servidores que siguen operativos, minimizando el impacto en los usuarios. Además, la caché web reduce la dependencia de los servidores backend, aumentando la resiliencia del sistema.

Reflexión Final

Este proyecto ha demostrado que una arquitectura bien diseñada puede abordar efectivamente los desafíos de rendimiento, seguridad y escalabilidad que enfrenta una tienda online moderna. La segmentación de servicios y la implementación de tecnologías de caché y balanceo de carga no solo mejoran la experiencia del usuario, sino que también aseguran la sostenibilidad y el crecimiento del negocio en el largo plazo. Los principios y prácticas desarrollados en este proyecto pueden servir como base para futuras implementaciones y mejoras en infraestructuras similares.

8. CONCLUSIONES



Resultado del proyecto: Finalmente, cuando nos conectamos al balanceador de carga. Este hace su función correctamente redirigiendonos a ambos servidores web.

Destacar la solicitud CURL, en el encabezado X-Caché Status: HIT , indica que el servidor web caché (Nginx) Esta funcionando correctamente.

Conexión MySQL correctamente.

Conexión con Memcached correctamente.

Por ultimo la aplicación evalua si los datos los ha recibido de Memcached o de Mysql. Ya que en caso de que deje de funcionar Memcached, realizaria la conexión directamente con el servidor MySQL y avisaria de que Memcached no esta activo.



SERVIDOR 1

Asignaturas y notas de "Miguel Ángel Rincón"

Headers del servidor en 192.168.1.205:

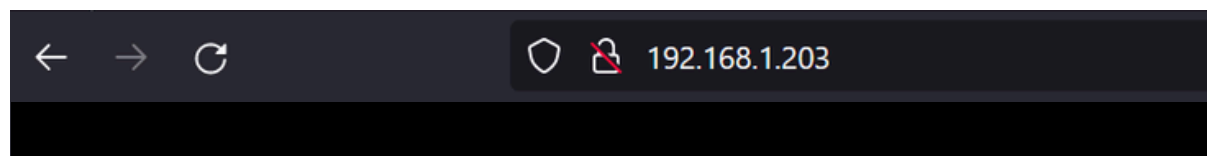
```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 03 Jun 2024 16:30:29 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1862
Connection: keep-alive
Vary: Accept-Encoding
X-Cache-Status: HIT
```

Conexión con MySQL establecida correctamente.

Conexión con Memcached establecida correctamente.

Datos obtenidos de Memcached

Asignatura	Nota
Base de Datos	9.00
Sistemas	10.00
Seguridad Informática	10.00
Servicios en Red	9.00
Aplicaciones Web	9.50
Programación Python	9.00



SERVIDOR 2

Asignaturas y notas de "Miguel Ángel Rincón"

Headers del servidor en 192.168.1.206:

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 03 Jun 2024 16:43:19 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1836
Connection: keep-alive
Vary: Accept-Encoding
X-Cache-Status: HIT
```

Conexión con MySQL establecida correctamente.

Conexión con Memcached establecida correctamente.

Datos obtenidos de MySQL y almacenados en Memcached para futuras consultas.

Asignatura	Nota
Base de Datos	9.00
Sistemas	10.00
Seguridad Informática	10.00
Servicios en Red	9.00
Aplicaciones Web	9.50
Programación Python	9.00



Idea principal del proyecto: Seguir la estructura de microservicios en Kubernetes y orientarlo a servidores web. Josejuansanchez. (s. f.). curso_kubernetes_cep/modulo1/implantacion-aplic-web.md at main · josejuansanchez/cursos_kubernetes_cep. GitHub.
https://github.com/josejuansanchez/cursos_kubernetes_cep/blob/main/modulo1/implantacion-aplic-web.md

Servicio Apache: Install and Configure Apache | Ubuntu. (s. f.). Ubuntu.
<https://ubuntu.com/tutorials/install-and-configure-apache>

Nginx Balanceo de carga:
<https://help.clouding.io/hc/es/articles/360019908839-C%C3%B3mo-configurar-un-servidor-de-balanceo-de-carga-Nginx-en-Ubuntu-20-04>

Nginx Web Caché: NGINX Content Caching. (s. f.). NGINX Documentation. <https://docs.nginx.com/nginx/admin-guide/content-cache/content-caching/>

Mysql: Recursos de clase + Conectar con un servidor de base de datos MySQL remoto. (s. f.). DesarrolloWeb.com.
<https://desarrolloweb.com/articulos/conectar-servidor-mysql-remoto>

Memcached: Camisso, J. (2021, 27 septiembre). How To Install and Secure Memcached on Ubuntu 20.04. DigitalOcean.
<https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-memcached-on-ubuntu-20-04>

Código php: Funciones MySQL de Víctor: Zamora Lorente, V. Z. (s. f.). z0: Centro de recursos y biblioteca digital. Repositorio Digital.
<https://s00.ddns.net/>

Licencia Creative Commons: CC BY-NC 4.0 Deed | Attribution-NonCommercial 4.0 International | Creative Commons. (s. f.).
<https://creativecommons.org/licenses/by-nc/4.0/?ref=chooser-v1>