Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

ingweb-tx-2

Aplicación Escalable

Autor:	Toral Pallás, Héctor - 798095
Grado:	Ingeniería Informática
Curso:	2022-2023

29 de noviembre de 2022

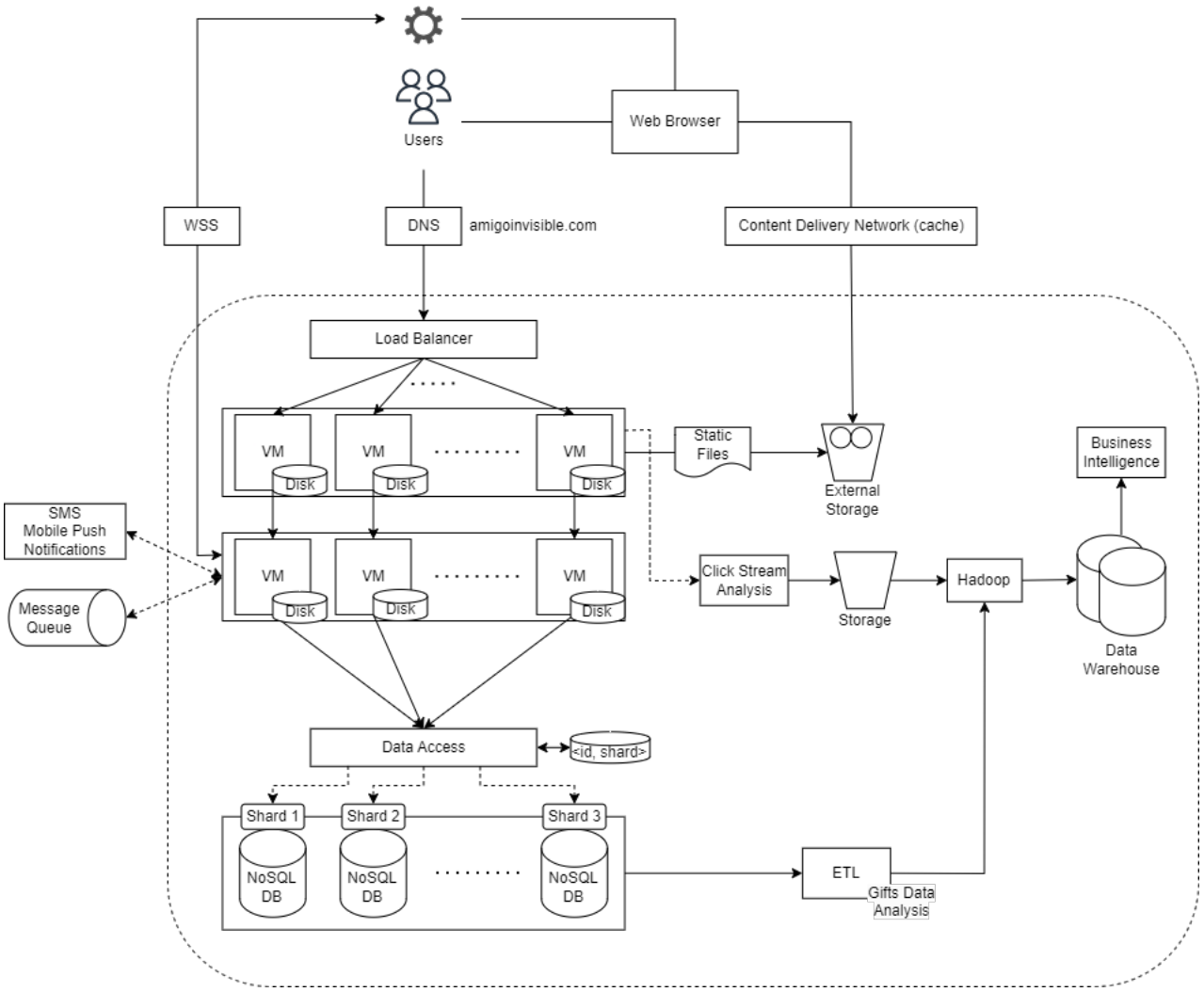
Índice

1. Introducción

La aplicación sobre la que se plantea realizar el diseño de arquitectura escalable surge de la idea de realizar amigos invisibles principalmente en épocas como la navidad debido a que las familias con miembros no tan jóvenes no suelen saber que regalarse los unos a los otros. La aplicación permitiría crear una serie de salas, una para cada evento en el que se añadirían los participantes pudiendo estos añadir sugerencias sobre posibles regalos a otros miembros del grupo. A su vez, como en el amigo invisible, a cada miembro perteneciente a un grupo se le asignaría otro integrante del mismo al cual le tocaría hacerle el regalo.

Como se puede observar en el siguiente gráfico, la palabra clave: **regalo** aumenta su popularidad para las fechas de navidad al igual que en días clave como: **San Valentin** o el **día de la madre**.

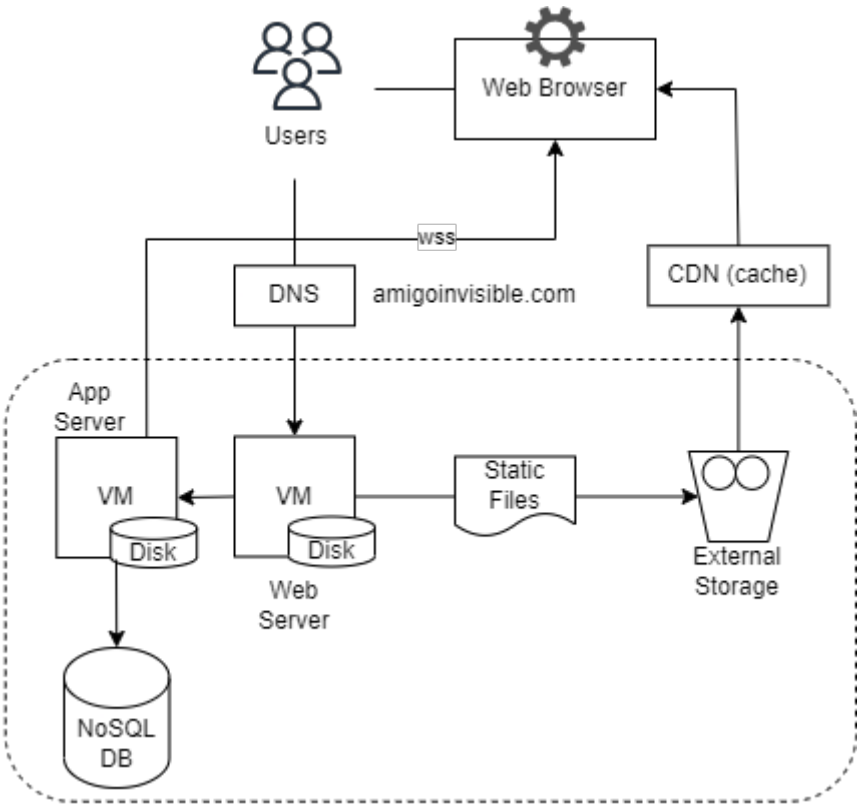
2. Arquitectura



La arquitectura que se plantea en la ilustración superior se organiza en cuatro partes bien diferencias, la de frontend que se encuentra en la sección superior de la imagen, la capa de servicios situada en la capa intermedia de la misma, la de persistencia en la zona mas inferior y la de procesamiento de datos para la parte de negocio en el lateral derecho de la imagen.

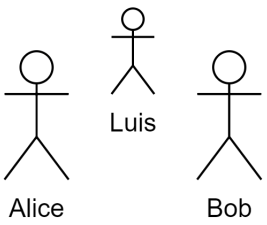
La finalidad de esta arquitectura es diversa puesto que contamos con distintas finalidades. Una de ellas, sería el minimizar el tiempo de carga de cara al usuario que utiliza la aplicación, otra los procesos de notificación, otra los procesos de batch y ETL empleados para hacer estudios de los clientes y poder generar sugerencias sobre ciertos productos y realizar un sistema de ventas cruzadas en base a los gustos especificos de cada usuario favoreciendo así el incremento del precio de impresión de un anuncio.

3. Front-End



Para la parte del frontend se ha optado por seguir una estrategia de data fetching conocida como Incremental Static Regeneration (ISR) en los web servers de tal manera que, una vez hayamos creado nuestro sitio web podamos actualizar y crear archivos estáticos para posteriormente guardarlos en el external storage. Este tipo de estrategia permite utilizar la generación estática por página, sin necesidad de reconstruir todo el sitio. La principal razón de escoger este tipo de estrategia frente a una de SSR (Server Side Rendering) es que de esta manera estaríamos liberando carga de trabajo de nuestro Web Server en cada petición de un nuevo usuario sirviendo así los archivos estáticos previamente generados.

Una vez generados y guardados los archivos estáticos en el external storage procedemos a distribuirlo mediante el CDN lo que permitirá a los usuarios realizar un primer acceso más rápido a nuestro servicio a la vez que hemos liberado a nuestro web server de una posible carga de trabajo adicional.



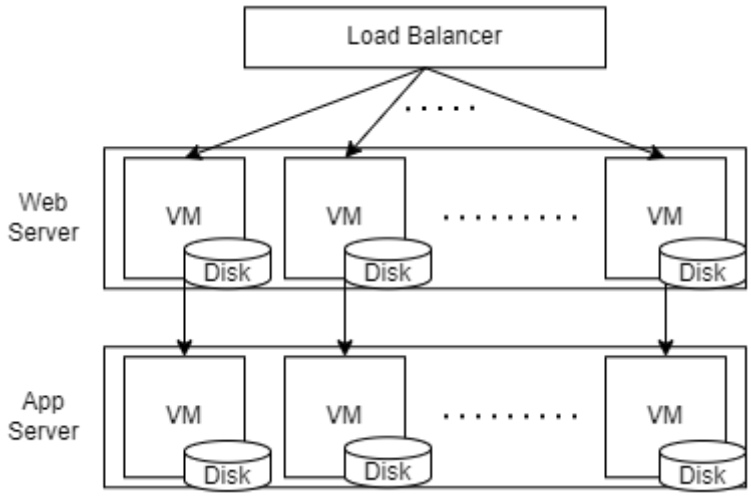
Por otro lado, se creará una conexión mediante websockets entre el cliente y nuestro app server de tal manera que si a Luis le ha tocado como amigo Bob y Alice añade un posible regalo para Bob al sistema, este deberá notificar al amigo invisible de Bob(Luis) de que hay un nuevo regalo que le podría gustar a Bob y sea el propio cliente del usuario (el navegador) el que se encargue de renderizar el nuevo dato en pantalla.

4. Servicios Web

Para la capa de servicios web se ha optado por colocar dos tipos de servidores, el web server y el app server.

El primero sería encargado de realizar la estrategia ISR comentada en la sección de frontend de tal manera que generaría un primer estático para el usuario que sería regenerado para así mantenerlo actualizado. La estrategía de actualización podría variar en base a la configuración que se le quiera dar, por ejemplo que se actualice cada 10 minutos.

El segundo, es el encargado del procesamiento de datos de los clientes es decir, si un usuario añade un posible regalo a otro, este se añade a una cola de mensajes para posteriormente hacer el dato persistente, además añade el dato al canal de websockets que corresponda para que el cliente que sea pueda actualizar su información en tiempo real. Una vez realizada la operativa sería el encargado de notificar al cliente vía SMS de que se ha añadido algo a la base de datos sobre el usuario a los que pueda estar suscrito.

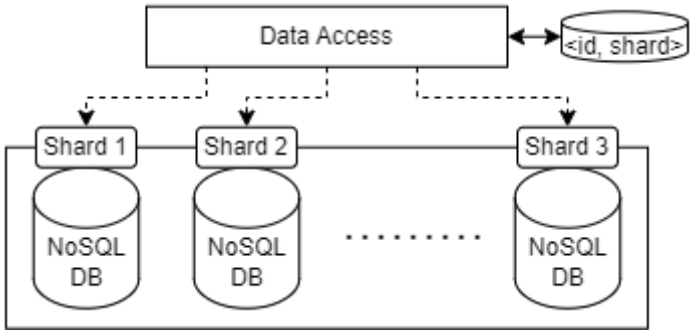


Otra de las opciones barajadas sería la de hacer una arquitectura basada en micro servicios de tal manera que hubiera un servicio de notificación, uno de actualización; como no dispongo de la experiencia ni de los datos para poder comparar el rendimiento de una vs la otra, asumiré que el balanceo de carga entre los N servidores será suficiente para un elevado número de usuarios permitiendo realizar las escrituras y notificaciones de manera rápida y segura.

De cualquier forma, si hubiera cualquier problema en la arquitectura elegida sería relativamente sencillo pasar a una de microservicios debido a que son funcionalidades muy acotadas y no supondrían a priori un gran cambio en el código que implementa dicha funcionalidad.

5. Búsqueda y Persistencia

Para la capa de persistencia se ha optado por una base de datos no sql debido a que no necesitamos características como las propiedades ACID ya que principalmente se van a realizar operaciones de lectura.



En cuanto a la arquitectura, como se puede observar en la figura, se ha optado por hacer sharding de la base de datos agrupando los datos de los usuarios por regiones, de esta manera al realizar una operación de SELECT se podrá realizar sobre un conjunto menor de datos reduciendo así el coste de reccorer una tabla de N filas a una de M filas siendo M « N.

»6. Infraestructura

»La infraestructura del sistema contaría con los siguientes componentes: Un sistema de colas en el que se encolarían las distintas sugerencias de los usuarios que a su vez estarían siendo comunicadas al cliente mediante el websocket. Un proceso ETL para la extracción, transformación y carga de datos en un hadoop que se encargaría de realizar un preprocesamiento y analisis de los datos para posteriormente cargarlos en un datacenter y poder analizar los mismos.

»7. Otros

»Otros aspectos que también serían relevantes de cara a poder escalar el sistema de la mejor forma posible serían los siguientes:

- »■ Seguir patrones dentro del código como clean architecture de tal manera que los componentes utilizados estén bien ordenados y listos para ser reutilizados.
- »■ Realizar distintos tipos de test, tanto E2E como unitarios de tal manera que si se requiere cambiar de modelo arquitectural o realizar cualquier cambio en la aplicación funcione correctamente, o en caso contrario permitan detectar los fallos con la mayor brevedad posible.
- »■ Realizar una buena documentación de los servicios desplegados para que esten disponibles con facilidad y sean entendibles por todo el equipo de desarrollo utilizando así estándares como openapi.
- »■ Intentar desplegar la aplicación dentro de contenedores para ser independientes del sistema o máquina donde se despliegue la aplicación, permitiendo hacer despliegues y escalados de forma automática y rápida.
- »■ Integrar analizadores de código estático para detección de vulnerabilidades a la vez que linters para el unificado del estilo en todo el código utilizando herramientas como detekt, eslint, etc.
- »■ Utilizar sistemas de descubrimiento de servicios para saber que nodos de X servicio se encuentran disponibles y cuales caídos así como saber cuanto tiempo han estados caídos y propiedades de los mismos.