Arquitectura sistema base de datos

Pongamos que quiero construir en mi casa una aplicación web que sirva de base de datos con información gráfica y numérica de los balances financieros de compañías del S&P500. Si todos los datos de esta app se almacenan en tablas podremos utilizar una base de datos SQL (PostgreSQL por ejemplo). Si la app requiere almacenar por ejemplo pdf`s, documentos, etc, usamos una base de datos NoSQL (MongoDB). En nuestro caso usaremos tablas y PostgreSQL como “Sistema gestor de base de datos” SGBD.

Decido implementar toda la arquitectura de forma local para testar la primera versión de la app, en una segunda etapa migraremos a la nube. Ahora en local podría crear una arquitectura monolítica o de microservicios [1], escojo la de microservicios por su escalabilidad y flexibilidad. Entonces los elementos y recursos que actuarían sobre el flujo ¿serían estos? :

- Servidores Separados: en casa, un servidor de datos para las bases de datos y un servidor separado para la aplicación (backend).
- Docker en Ambos: Instalas Docker [2] [3] en ambos servidores.
- Contenedores de Datos: En el servidor de datos, configuras contenedores Docker para cada base de datos que quieras manejar.
- Contenedores de Aplicación: En el servidor de aplicaciones, configura contenedores Docker para los microservicios y/o la aplicación principal.
- Comunicación: Los contenedores en el servidor de aplicaciones se comunican con los contenedores en el servidor de datos a través de la red (específicamente, puertos abiertos y direcciones IP).

En este escenario PostgreSQL estaría encapsulado en uno o más contenedores Docker en el servidor de datos. Este o estos contenedores serían la fuente principal de datos para los microservicios en el servidor de aplicaciones.

Despliegue en el Servidor de Datos:

- Instalación de PostgreSQL en un Contenedor: Puedes instalar PostgreSQL en un contenedor Docker que se ejecuta en tu servidor de datos físico. Este contenedor tendría todo lo necesario para ejecutar PostgreSQL, incluyendo el sistema operativo, PostgreSQL mismo, y cualquier otra dependencia.
- Configuración y Seguridad: Configurar este contenedor de PostgreSQL para que sea accesible desde tu servidor de aplicaciones, pero seguramente restringido para que solo ciertos servicios o IPs puedan acceder a él.
- Almacenamiento de Datos: Este contenedor de PostgreSQL alojaría las tablas y datos que tus microservicios necesitan. Podrías tener múltiples contenedores de PostgreSQL en el mismo servidor físico si tienes múltiples bases de datos o si quieres separar datos entre diferentes microservicios.
- Acceso a Datos: Los microservicios en tu servidor de aplicaciones accederían a este contenedor de PostgreSQL para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar), probablemente utilizando una biblioteca cliente de PostgreSQL en el lenguaje de programación que estés utilizando para tus microservicios.

- Comunicación: La comunicación entre el servidor de aplicaciones y el servidor de datos se realizaría a través de la red, utilizando los puertos y protocolos estándar de PostgreSQL (usualmente el puerto 5432). Esta comunicación deberá ser segura, posiblemente utilizando SSL/TLS o una VPN entre servidores.
- Escalabilidad y Rendimiento: Si necesitas más rendimiento o alta disponibilidad, podrías configurar contenedores adicionales de PostgreSQL para replicación o particionamiento, aunque esto añadiría complejidad.
- Backup y Recuperación: También deberías considerar cómo vas a hacer copias de seguridad de tu base de datos PostgreSQL. Esto podría implicar otro contenedor que se encargue de esta tarea o una solución externa.
- Monitoreo y Mantenimiento: Finalmente, tendrías que monitorizar el rendimiento, los registros y la salud de tu contenedor de PostgreSQL, posiblemente usando herramientas de monitoreo y registro que también podrían ejecutarse en contenedores.

Despliegue en el Servidor de Aplicaciones:

- Docker en Servidor de Aplicaciones: Despliega contenedores Docker para cada microservicio que componga tu backend. Cada microservicio debería ser capaz de comunicarse con la base de datos en el servidor de datos.
- ETL Avanzado y Lógica de Negocio: Inicia cualquier proceso ETL más complejo o lógica de negocio en este servidor. Los microservicios aquí transformarán los datos según sea necesario antes de presentarlos al cliente.
    - Opción 1. Almacenar en el Servidor de Aplicaciones : Si las transformaciones son temporales o específicas para una tarea que está manejando el servidor de la aplicación, podrías optar por mantener esos datos en el servidor de aplicaciones.
        - Rendimiento: Podría ser más rápido acceder a estos datos transformados si ya están en el mismo servidor que los necesita.
        - Desacoplamiento: Esto también podría simplificar la arquitectura al mantener los datos específicos de una operación o tarea cerca del servicio que los maneja.
    - Opción 2. Almacenar en el Servidor de Datos : En esta opción, una vez que el servidor de la aplicación ha realizado las transformaciones adicionales o más complejas en los datos, esos datos transformados se envían de vuelta al servidor de datos para su almacenamiento. Esto podría hacerse por varias razones:
        - Consolidación de Datos: Mantener todos los datos en un solo lugar facilita la administración y el mantenimiento.
        - Seguridad: Si tienes controles de seguridad más estrictos en el servidor de datos, podría tener sentido almacenar allí toda la información sensible.
    - Opción 3: Ambos. En algunos casos, podría tener sentido almacenar los datos transformados en ambos servidores.
        - Caché Local: Podrías mantener una caché de datos transformados en el servidor de la aplicación para un acceso rápido, mientras que también envías estos datos al servidor de datos para su almacenamiento a largo plazo.

    - Resiliencia: Tener datos en ambos lugares podría ser útil desde el punto de vista de la resiliencia y la recuperación de desastres.
- API RESTful: Expone una API RESTful para que los clientes puedan interactuar con tus microservicios.

Acceso Público

- Dominio y DNS: Compra un nombre de dominio y configura el DNS para que apunte a la IP pública de tu servidor de aplicaciones.
- Proxy Inverso: Instala y configura un proxy inverso (como Nginx o Apache) en tu servidor de aplicaciones para dirigir el tráfico entrante a los contenedores Docker apropiados. Asegúrate de habilitar HTTPS para la seguridad.

Acceso de los Clientes

- Frontend: Los usuarios interactuarán con tu sistema a través de un frontend, que podría ser una aplicación web o móvil. Esta aplicación consumirá la API RESTful que has expuesto en tu servidor de aplicaciones.
- Autenticación y Autorización: Implementa un sistema de autenticación y autorización para que solo los usuarios autorizados puedan acceder a tu aplicación.

Configuración en el Servidor de Datos

- Instalar PostgreSQL: Asegúrate de que PostgreSQL está instalado y configurado en tu servidor de datos.
- Configurar Seguridad: Utiliza medidas de seguridad como firewalls, y asegúrate de que PostgreSQL solo acepte conexiones de las IPs que necesitan acceso (en este caso, la IP de tu servidor de aplicaciones).
- Crear la Base de Datos y Tablas: Crea la base de datos que necesitarás, junto con cualquier tabla y esquema que sean relevantes.
- Permisos de Usuario: Configura un usuario en PostgreSQL con los permisos necesarios y asegúrate de que ese usuario pueda acceder desde el servidor de aplicaciones.

Configuración en el Servidor de Aplicaciones

- Driver o Biblioteca para PostgreSQL: Tu aplicación necesitará una forma de comunicarse con PostgreSQL. Esto generalmente se hace a través de un driver o una biblioteca que sea compatible con el lenguaje de programación que estés utilizando (por ejemplo, psycopg2 para Python).
- String de Conexión: Configura tu aplicación para usar una string de conexión que apunte a tu servidor de datos PostgreSQL. Esta string de conexión generalmente incluirá la dirección IP del servidor de datos, el puerto (generalmente 5432 para PostgreSQL), el nombre de la base de datos, y las credenciales del usuario.
- Queries y Operaciones: Usa el driver o biblioteca para enviar consultas SQL desde tu aplicación al servidor de datos.

Suponiendo que estás utilizando Python en tu servidor de aplicaciones, aquí hay un fragmento de código de ejemplo que muestra cómo se podría establecer una conexión y hacer una consulta:

Ejemplo en Python con psycopg2

```python
import psycopg2

# Conexión a la base de datos
conn = psycopg2.connect(
    host="tu_servidor_de_datos",
    port="5432",
    dbname="tu_base_de_datos",
    user="tu_usuario",
    password="tu_contraseña"
)

# Crear un objeto 'cursor' para interactuar con la base de datos
cur = conn.cursor()

# Ejecutar una consulta SQL
cur.execute("SELECT * FROM tu_tabla;")

# Obtener los resultados
resultados = cur.fetchall()

# Hacer algo con los resultados
for fila in resultados:
    print(fila)

# Cerrar el cursor y la conexión
cur.close()
conn.close()
```

DBeaver [4] es una herramienta de administración de bases de datos que ofrece una interfaz gráfica para gestionar diversas bases de datos, incluyendo PostgreSQL. Puede ser útil en varios aspectos de tu arquitectura y desarrollo:

Desarrollo y Pruebas

- Exploración de Datos: Puedes usar DBeaver para explorar los datos en tu base de datos, lo cual es especialmente útil durante la fase de desarrollo.
- Debugging: Si algo va mal con tus consultas o datos, puedes usar DBeaver para investigar el problema.
- Manipulación de Datos: DBeaver permite realizar operaciones de CRUD (Crear, Leer, Actualizar, Borrar) sin tener que escribir SQL manualmente.

Administración de la Base de Datos

- Gestión de Esquemas y Tablas: Puedes crear, modificar o eliminar tablas y esquemas fácilmente.
- Gestión de Usuarios y Permisos: Aunque esto a menudo se hace mejor a través de la línea de comandos por razones de seguridad, algunas tareas simples de gestión de usuarios y permisos también se pueden realizar.
- Exportar e Importar Datos: DBeaver ofrece funcionalidades para exportar e importar datos, lo que podría ser útil para tu proceso ETL o para realizar copias de seguridad.

Mantenimiento y Monitoreo

- Ejecución de Consultas SQL: Si necesitas realizar consultas SQL ad-hoc para el mantenimiento o monitoreo, puedes hacerlo directamente desde DBeaver.
- Visualización: DBeaver puede ayudar a visualizar la estructura de tu base de datos, las relaciones entre tablas, etc., lo cual es útil para entender el diseño y la arquitectura de tu base de datos.

Conexión Remota

- Acceso Remoto: DBeaver permite conectarse a bases de datos remotas. Así que, incluso si tu base de datos PostgreSQL está en un servidor en tu casa, puedes acceder a ella desde otro lugar (siempre y cuando los ajustes de red y seguridad lo permitan).

Dicho esto, DBeaver es generalmente más útil en las etapas de desarrollo, depuración y mantenimiento que en un entorno de producción en funcionamiento. No reemplaza a las herramientas de administración de sistemas y monitoreo más robustas que querrás tener en un entorno de producción, pero ciertamente puede ser una herramienta valiosa en tu caja de herramientas de desarrollo.

En un entorno de producción, especialmente si estás manejando datos financieros sensibles y/o una gran cantidad de tráfico de usuarios, querrás asegurarte de que tu sistema sea seguro, eficiente y fácil de monitorizar y mantener. A continuación te presento algunas categorías de herramientas que podrían ser útiles:

Monitoreo de Rendimiento y Salud del Sistema

- Prometheus: Es una herramienta de monitoreo y alerta de código abierto que maneja métricas multidimensionales muy eficientemente.
- Grafana: A menudo se usa junto con Prometheus para visualizar las métricas.
- Nagios: Es una herramienta de monitoreo de sistema ampliamente utilizada que puede alertar sobre problemas antes de que afecten a los usuarios o los servicios.

Administración de Bases de Datos

- pgAdmin: Para bases de datos PostgreSQL, pgAdmin ofrece una variedad de características avanzadas para la administración.
- Database Performance Analyzer: Herramienta que ayuda a identificar problemas de rendimiento en bases de datos.

Seguridad

- Firewalls de Aplicaciones Web (WAF): Herramientas como Cloudflare o AWS WAF pueden proteger tu aplicación de diversas amenazas.
- Herramientas de Análisis de Seguridad de Código: SonarQube, Checkmarx, etc., pueden ayudar a identificar problemas de seguridad en el código antes de que lleguen a producción.

Logs y Análisis de Datos

- ELK Stack (Elasticsearch, Logstash, Kibana): Estas herramientas trabajan juntas para permitir la búsqueda y visualización de logs.
- Graylog: Una alternativa al stack ELK, más fácil de configurar en algunos aspectos.

Automatización y Despliegue

- Jenkins: Una herramienta de CI/CD que te permite automatizar diversas partes del proceso de desarrollo.
- Kubernetes: Para la orquestación de contenedores si estás usando una arquitectura de microservicios.

Backups y Recuperación de Desastres

- Herramientas de Backup de Base de Datos: Asegúrate de que tienes backups regulares y automáticos de tu base de datos.
- Plan de Recuperación de Desastres: Esto no es una herramienta per se, pero es esencial tener un plan y posiblemente herramientas automáticas para manejar fallas catastróficas.

Análisis de Tráfico Web y User Analytics

- Google Analytics: Para análisis de tráfico web general.
- Mixpanel/Amplitude: Para análisis más detallado del comportamiento del usuario dentro de la aplicación.

Esta lista no es exhaustiva, pero debería darte una buena idea de las diferentes áreas que querrás tener en cuenta cuando muevas tu aplicación y base de datos a un entorno de producción.

PREGUNTAS :

¿dónde hago las transformaciones ETL?

Podría depender de los objetivos específicos de la aplicación y de los servicios que se planee ofrecer a los usuarios, por ejemplo:

El proceso ETL en el Servidor de Datos lo realizaría si :

- se manejan grandes volúmenes de datos que requieren transformaciones computacionalmente intensivas
- Al realizar la transformación de datos donde se almacenan los datos minimiza la necesidad de transferencias de datos adicionales.
- etc

El proceso ETL en el Servidor de la aplicación lo realizaría si :

- Tengo algoritmos complejos para analizar o enriquecer datos (por ejemplo, recomendaciones de inversión), podrían implementarse más fácilmente en el servidor de la aplicación.
- Cuando la lógica de negocio requiere interactuar con otros servicios web o APIs en tiempo real.
- la lógica de negocio que afecta los datos cambia con frecuencia. Con “lógica del negocio” me refiero al objetivo de la app (Selección de Datos, Histórico de Datos, Comparación Entre Empresas, Análisis Fundamental, Notificaciones en Tiempo Real, Integración con Otras Plataformas, Gestión de Usuarios, Informes y Análisis).

También podríamos aplicar un enfoque híbrido, donde se puede realizar transformaciones básicas y filtrados en el servidor de datos, y luego realizar transformaciones más complejas y específicas en el servidor de la aplicación.

¿Cuánto varía si lo que quiero es utilizar una base de datos NoSql?

Utilizar una base de datos NoSQL como MongoDB en lugar de una base de datos relacional como PostgreSQL puede cambiar varios aspectos de tu proyecto, desde la estructura de los datos hasta cómo se realizan las operaciones de ETL (Extracción, Transformación, Carga). A continuación, te muestro algunas de las variaciones que podrías enfrentar:

Diseño de la Base de Datos

- Estructura de Datos: En MongoDB, los datos generalmente se almacenan en documentos en formato BSON (similar a JSON). Esto puede facilitar el almacenamiento de datos estructurados y semi-estructurados, como PDFs en campos binarios.
- Esquema: MongoDB es esquema-less, lo que significa que no tienes que definir una estructura rígida de antemano. Esto ofrece flexibilidad pero puede complicar la integridad de los datos si no se maneja cuidadosamente.

Proceso ETL

- Transformación de Datos: Las operaciones de transformación pueden requerir una lógica diferente debido al cambio de formato de los datos.
- Consultas y Agregaciones: MongoDB tiene un conjunto diferente de operadores para consultas y agregaciones, lo que puede cambiar la forma en que realizas ciertas operaciones en el proceso de ETL.

Rendimiento y Escalabilidad

- Escalabilidad Horizontal: MongoDB está diseñado para escalar horizontalmente, lo que podría hacer que sea más fácil de escalar en comparación con soluciones SQL, que suelen centrarse en la escalabilidad vertical.

Integración con el Backend

- Drivers y Conectores: Deberás utilizar drivers y bibliotecas compatibles con MongoDB en tu lenguaje de programación backend.
- APIs: Las consultas que realiza tu API podrían tener que ser reescritas para ajustarse a cómo MongoDB maneja las operaciones de datos.

Monitoreo y Mantenimiento

- Herramientas de Monitoreo: Necesitarás herramientas que sean compatibles con MongoDB para el monitoreo del rendimiento, logs, etc.

Backup y Recuperación

- Operaciones de Backup: Las herramientas y procedimientos para backup y recuperación podrían cambiar, y tendrás que utilizar herramientas que son específicas para MongoDB.

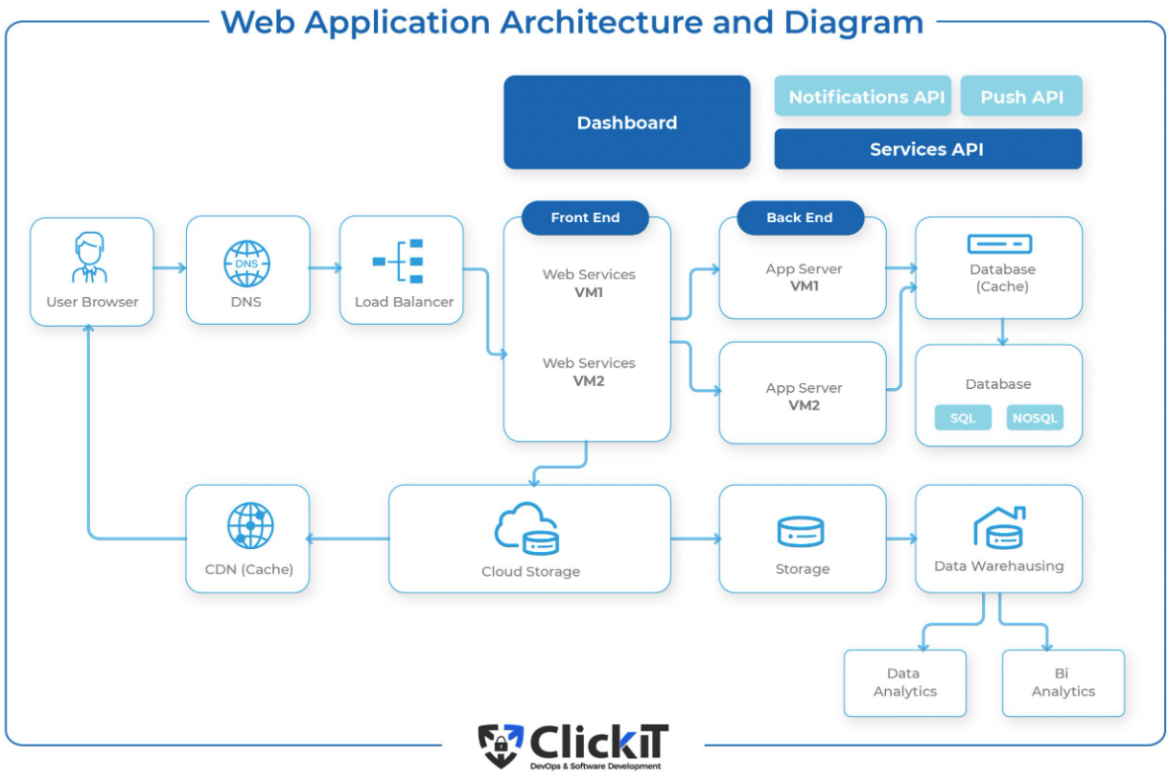
Seguridad

- Configuración de Seguridad: Al igual que con cualquier base de datos, tendrás que asegurarte de que tu instalación de MongoDB esté debidamente asegurada, pero las prácticas exactas podrían variar respecto a una base de datos SQL.

Dicho todo esto, la elección entre MongoDB y PostgreSQL [5] dependerá de tus necesidades específicas, la naturaleza de tus datos, y tus preferencias en términos de

flexibilidad frente a la estructura. Ambas son tecnologías robustas y ampliamente utilizadas, pero atienden a diferentes necesidades y casos de uso.

Diagrama de arquitectura de aplicaciones web [6]



RESEÑAS

[1] A. Valdes, «Microservices vs Monolith: The Ultimate Comparison 2022», *ClickIT*, 10 de marzo de 2021. https://www.clickittech.com/devops/microservices-vs-monolith/ (accedido 16 de septiembre de 2023).

[2] «Docker overview», *Docker Documentation*, 11:06 + +0100 de 100d. C. https://docs.docker.com/get-started/overview/ (accedido 16 de septiembre de 2023).

[3] G. Velez, «Kubernetes vs Docker: Best open source technologies for Financial Services», *ClickIT*, 26 de agosto de 2019. https://www.clickittech.com/fintech/open-source-fintech/ (accedido 16 de septiembre de 2023).

[4] «DBeaver Community | Free Universal Database Tool». https://dbeaver.io/ (accedido 16 de septiembre de 2023).

[5] «MongoDB Vs PostgreSQL», *MongoDB*. https://www.mongodb.com/compare/mongodb-postgresql (accedido 16 de septiembre de 2023).

[6] William, «Web Application Architecture: The Latest Guide 2022», *ClickIT*, 10 de marzo de 2022. https://www.clickittech.com/devops/web-application-architecture/ (accedido 16 de septiembre de 2023).