Escuela Politécnica Superior

Monitorización y análisis de configuraciones y esquemas en motores de bases de datos relacionales

Máster Universitario en Ciberseguridad

Trabajo Fin de Máster

Autor:
Eddie Rodríguez Pastor

Tutor/es:
Juan Antonio Gil Martinez-Abarca

Septiembre 2021

Universitat d'Alacant
Universidad de Alicante

Resumen

Por mi experiencia profesional, las bases de datos relacionales son uno de los componentes más sensibles dentro de una aplicación. Un mínimo cambio no deseado en estas puede desencadenar un resultado catastrófico que puede ir desde una denegación de servicio hasta la perdida de datos. Si bien, una base de datos bien configurada no debería ser modificable de forma no deseada, esto no se cumple en la mayoría de las empresas.

En este proyecto se plantea la creación de un software cuya principal tarea será la de monitorizar instancias de bases de datos relacionales, así como su configuración y sus posibles mejoras.

Este trabajo abordará la prueba de concepto de este software que será centrada principalmente en el proveedor de bases de datos de Microsoft, SQL Server. De forma conjunta, implementará su equivalente para la base de datos MySQL, siendo también compatible con ciertas versiones de MariaDB. De todas formas, la implementación de este proyecto estará abierta a la introducción de nuevos sistemas de monitorización para otros sabores de bases de datos.

Palabras clave: ciberseguridad, integridad, bases de datos relacionales, SQL Server, MySQL, monitorización.

Motivación, justificación y objetivo general

Durante mi primera experiencia laboral, tuve la oportunidad de trabajar como desarrollador para una consultora que estaba centrada principalmente en la administración de bases de datos de SQL Server. Si bien mi puesto no estaba centrado en tratar directamente con las bases de datos de los clientes, tuve la oportunidad de descubrir algunos problemas que surgían en relación con estas.

Dado que los clientes a menudo desconocían como debían gestionar correctamente las bases de datos, delegaban esta responsabilidad en la empresa para la que trabajaba. La empresa consultora, se hacía cargo de llevar a cabo todo el mantenimiento y gestión de cambios de estas bases de datos. A pesar de esto, era muy común que el cliente modificara sin consultar con la consultora alguna configuración o esquema de la base de datos, resultando en ocasiones la causa de algún problema mayor.

Otra de las problemáticas que me he encontrado trabajando en diferentes proyectos es que a menudo existen procesos automatizados de despliegue con permisos para modificar completamente la base de datos. Si seguimos la cadena de procesos, al final podemos apreciar que dicho usuario de bases de datos es accesible para todo aquel que tenga acceso al proceso de despliegue automático, al cual normalmente se accede a través de los repositorios de código. Esto permite a los usuarios desarrolladores tener acceso a las instancias de bases de datos, algo que no debería ser posible y que puede ocasionar problemas.

Siempre que me surgía algún problema relacionado con un cambio inesperado en una base de datos, deseaba tener alguna aplicación que realizara un control de todos aquellos cambios que estaban llevándose a cambio dentro de la instancia de SQL con el fin de localizar cuanto antes el problema he incluso prevenir la inclusión de bombas lógicas.

Actualmente ya no trabajo en esa área laboral y me he movido a un rol relacionado con la arquitectura de sistemas. En este rol, las configuraciones desplegadas en las bases de datos siguen siendo un problema dado que a menudo encontramos configuraciones por defecto que deben ser evitadas.

Con el desarrollo de este proyecto, deseo poder solventar todos aquellos problemas que me he encontrado a lo largo de mi vida laboral relacionados con la configuración y cambios en una base de datos relacional.

Agradecimientos

Agradecer en primer lugar, a todos los compañeros de trabajo que he tenido, especialmente a Cristina y Guillermo, de los cuales he podido aprender lo que hoy hace posible este trabajo.

En segundo lugar, agradecer a mi tutor del Trabajo de Final de Máster, Juan Antonio Gil, por haberme permitido realizar este proyecto y su apoyo durante el desarrollo.

En tercer lugar, agradezco a todos aquellos que me han acompañado en el Máster de Ciberseguridad, tanto compañeros como profesores, por haberme aportado conocimientos y un ambiente agradable en el que desarrollarme como profesional.

En cuarto lugar, agradezco a Luis Pérez por haber estado a mi lado durante una importante etapa y por haber sido quien me convenció para estudiar este Máster.

En quinto lugar, agradecer a Adrián Benítez por haberme acompañado todas las tardes mientras desarrollaba este proyecto.

Finalmente, quiero dar las gracias a mi familia y amigos, sin ellos no habría llegado a donde estoy hoy.

Citas

“Tú, queriendo descifrar

mi empeño por poner

un cielo azul aquí entre tanto trasto.”

Robe Iniesta

Índice

Índice de ilustraciones

1. Introducción

Actualmente estamos viviendo una evolución en la forma que tenemos de crear aplicaciones. Gracias en parte al auge de las tecnologías Cloud, la forma en la que las aplicaciones están siendo desarrolladas está migrando a un tipo de desarrollo más ágil que debe de ser capaz de evolucionar a los cambios de una forma rápida y eficaz.

Debido a este cambio en el paradigma del desarrollo de las aplicaciones, se están popularizando ciertas soluciones que permiten solventar problemas en código que no estaban pensados para ello. Un ejemplo de esto es la infraestructura como código, siendo Terraform una de las soluciones más populares actualmente.

En la infraestructura como código, el usuario es capaz de definir utilizando código la arquitectura completa de un sistema. Esto habilita a los desarrolladores la posibilidad de desplegar infraestructura perfectamente configurada en cuestión de segundos, permitiendo además la reutilización de estas arquitecturas, algo muy útil dado que, en numerosas ocasiones, las aplicaciones comparten la misma arquitectura.

Una de las ventajas de la infraestructura como código es que nos permite detectar cambios en nuestra infraestructura. Pongamos un ejemplo, si hemos definido una infraestructura en código que contenía dos servidores con el puerto 443 expuesto, si realizamos un cambio en esta infraestructura, ya sea la eliminación de un servidor, la creación de uno nuevo o la configuración en los puertos expuestos, cuando se realice una comparación entre la infraestructura desplegada y la infraestructura definida en código, se generará un informe donde se señalen las diferencias existentes entre estas dos.

Si bien esta característica no está pensada para ser utilizada de cara a la monitorización de cambios de una infraestructura, es una herramienta muy útil que permite al usuario conocer rápidamente aquellos cambios que se realizaron sobre la infraestructura y que no estaban planeados. Si estos cambios hubieran sido planeados, se habrían aplicado sobre la plantilla de infraestructura como código y posteriormente sobre la infraestructura desplegada.

Otro ejemplo de cómo el código nos puede permitir automatizar procesos es Ansible, el cual es otro software de infraestructura como código, pero más centrado en la automatización de despliegue de configuraciones que en la creación de infraestructura. Si bien Terraform nos permitía desplegar por ejemplo máquinas virtuales, Ansible nos permite desplegar software sobre estas máquinas virtuales.

Mientras que muchos componentes de la infraestructura se están beneficiando de estas herramientas, hay todavía un componente que, dada su complejidad, está siendo difícil de tratar y son las bases de datos relacionales.

Si bien es cierto que mediante Terraform y Ansible somos capaces de desplegar una base de datos configurada, esta configuración se queda en una capa superficial dado que a menudo, la configuración de una base de datos dista mucho del clásico fichero de configuración que utilizan la mayoría de las aplicaciones.

Gestión de cambios de configuraciones

Una de las cuestiones en cuanto a seguridad que plantea una infraestructura, es la gestión de cambios de las configuraciones. Si bien anteriormente hemos visto que se puede automatizar el despliegue de configuraciones con softwares como Ansible, de nada sirve si después un atacante o cualquier usuario con acceso al sistema es capaz de modificar esta configuración.

Por este motivo, surgen aplicaciones cuya finalidad es realizar un control sobre las configuraciones que hay aplicadas en una infraestructura. Un ejemplo de esto es Tripwire.

Tripwire es un software que se encarga de monitorizar los cambios que se realizan en los ficheros de configuración de un servidor. Para ello, Tripwire indexa en una base de datos todas las configuraciones aplicadas actualmente en un servidor y periódicamente comprueba si estas han cambiado, dependiendo de cómo se haya configurado Tripwire, se realizarán una serie de acciones como pueden ser el forzar una configuración o lanzar una alerta.

Si bien Tripwire puede ser muy útil para gestionar los cambios de configuración, no puede ser utilizado para gestionar los cambios de configuraciones internas en las bases de datos ni, principalmente, en el esquema de estas.

2. Estado del arte

Con esta situación planteada, en este capítulo se tratará de analizar aquellas herramientas que tratan de solventar algunas de las situaciones que son útiles para este proyecto.

De esta forma lograremos juntar las características más interesantes de cada uno de ellos con tal de formalizar un sistema de control de cambios de bases de datos relacionales robusto y que pueda resultar útil en un entorno productivo.

Proyecto de base de datos en Visual Studio

Visual Studio es el IDE creado por Microsoft que es compatible con los lenguajes de programación C, C# y F#.

Uno de los posibles tipos de proyectos que se encuentran disponibles dentro del IDE es el proyecto de base de datos. Este proyecto le permite al usuario la posibilidad de definir un esquema completo de base de datos compatible con el motor de base de datos de Microsoft, SQL Server.

Dentro de este proyecto, podemos crear cualquier tipo de objeto que forme parte del esquema como pueden ser tablas, vistas, procedimientos almacenados, disparadores e incluso algunos tipos de objetos que solo existen dentro de SQL Server como son las colas de Service Broker o los contratos.

Además, el IDE nos ofrece un conjunto de herramientas para poder trabajar con el proyecto y poder interactuar con instancias de SQL. Por ejemplo, se incluye una herramienta que permite realizar una comparación entre el esquema desplegado en una instancia y el esquema definido.

Una vez finalizada la comparación, el usuario tiene dos opciones, importar los cambios al proyecto o desplegar los cambios en la instancia.

Además de esta herramienta, también permite exportar un fichero que contiene el esquema para poder desplegarlo en cualquier instancia sin necesidad de hacer uso de Visual Studio.

La principal ventaja que aporta este tipo de proyecto es que trata la definición del esquema de una base de datos como si fuera código y por tanto se puede registrar en un sistema de control de versiones como GIT.

Si juntamos GIT junto a realizar comparaciones periódicas para importar cambios del esquema al proyecto, tenemos un buen sistema que nos permite registrar los cambios que sufre el esquema a lo largo del tiempo.

Los principales problemas que plantea esta solución es que no está pensado para ser un sistema que registre cambios, está pensado como una forma de desarrollar un esquema de base de datos junto a la propia aplicación. No se aporta ninguna herramienta que automatice las comparaciones entre esquemas.

Además, este proyecto solo es compatible con SQL Server y hay ciertas ocasiones donde no es compatible. En SQL Server, se pueden definir procedimientos almacenados que hagan uso de dos bases de datos diferentes, si quisiéramos generar un proyecto de base de datos que contuviera este tipo de procedimientos almacenados, necesitaríamos tener ambas bases de datos importadas, lo cual es posible.

Existe un caso donde no es posible utilizar el proyecto de base de datos y es en el caso de utilizar recursos externos. En Azure, no se pueden desplegar procedimientos almacenados que hagan uso de dos bases de datos diferentes, pero si se puede arreglar importando la segunda base de datos como un recurso externo y utilizando una tecnología denominada Elastic Query. Esta tecnología es, a fecha de hoy, incompatible con el proyecto de base de datos de Visual Studio.

Finalmente, otra de las desventajas que podemos considerar de este proyecto es que solo trabaja con el esquema de la base de datos, pero no con la configuración de esta.

DBForge

DBForge es un conjunto de herramientas que permite el desarrollo de esquemas de base de datos relacionales de forma similar a como lo hacen los proyectos de bases de datos de Visual Studio.

DBForge es compatible con diferentes motores de bases de datos como son SQL Server, Oracle, MySQL y PostgreSQL.

DBForge además añade algunas opciones como por ejemplo la posibilidad de crear Backups de las bases de datos, pero estas características se pueden utilizar con cualquier cliente que haga uso de la base de datos así que no se considera destacable.

SentryOne Document

SentryOne Document (Wright, 2020) es una herramienta de pago desarrollada por SolarWinds que se anuncia como una solución que permite llevar un control de los cambios que han surgido en la configuración de SQL Server. Esta solución automatiza la generación de informes que contienen las configuraciones aplicadas a los servidores de bases de datos.

SentryOne también es compatible con otras bases de datos de Microsoft como son Microsoft SQL Server BI Stack o SQL Server Analysis Services.

3. Objetivos

El objetivo general de este proyecto es el de desarrollar una herramienta software que permita recopilar tanto la configuración como el esquema de una base de datos relacional, realizar recomendaciones en función a estos y realizar un historial de cambios con el fin de aumentar la seguridad, tanto desde un punto de vista de resiliencia en las configuraciones aplicadas como el de un seguimiento en los cambios aplicados para la comprobación de que no se hayan realizado cambios no autorizados.

Para lograr este objetivo, previamente se ha de realizar las siguientes tareas:

- Estudio del funcionamiento de los softwares de monitorización.
- Estudio de los elementos que componen un esquema de base de datos.
- Investigación sobre las buenas prácticas de configuración de los motores de bases de datos de SQL Server y MySQL.
- Investigación sobre las configuraciones de una instancia de base de datos.
- Desarrollo de la herramienta de monitorización.
- Desarrollo de una herramienta que permita la visualización de los datos recopilados y tratados.
- Pruebas y validación de la herramienta.

4. Planificación del proyecto

En este capítulo se explicará la metodología que se utilizará para el desarrollo de este proyecto junto a la planificación que se ha elaborado para este. Una vez definida esta metodología, se definirá el entorno de trabajo a utilizar para el desarrollo del componente software del proyecto, así como el entorno desplegado donde se han realizado las pruebas del producto.

4.1 Metodología del desarrollo

La metodología de desarrollo software (Software development process, s.f.) es un marco de trabajo que recoge las prácticas que se han de realizar para estructurar, planificar y manejar el desarrollo de un proyecto software.

Existen diversas metodologías de desarrollo software, cada una con sus propias características y con el tiempo surgen nuevas metodologías que tratan de solventar las carencias de las existentes o simplemente aportar un punto de vista diferente a la hora de abordar un proyecto de desarrollo.

La metodología utilizada para este proyecto ha sido SCRUM por ser una metodología ágil y centrada en iteraciones. El enfoque de esta metodología permite dividir el proceso de desarrollo en iteraciones de tiempo, de manera en la que al final de cada tarea se consigue un producto final. Con este modelo se busca evolucionar el producto final de cada iteración con el fin de que el producto resultante de la última iteración sea el que cumpla con los requisitos definidos para el proyecto.

Se han definido una serie de tareas que serán incluidas en cada iteración.

El desarrollo de este proyecto resultará en un producto sencillo dado que este proyecto puede abarcar muchas mejoras y ampliaciones que serán propuestas al final de este trabajo.

4.2 Entorno y tecnologías de desarrollo

Para el desarrollo de este proyecto, nos tendremos que apoyar en diferentes tecnologías que giran en torno al ecosistema de Microsoft dado que SQL Server es una tecnología de esta compañía.

Si bien Microsoft ha hecho pública una versión al mercado de SQL Server compatible con Linux, necesitamos de componentes de Active Directory que son más accesibles desde un entorno de trabajo que corra bajo el sistema operativo Windows.

Por este motivo, se ha seleccionado Windows 10 como principal sistema operativo para el desarrollo y para las pruebas realizadas sobre SQL Server.

No obstante, se utilizará una imagen de contenedor Docker Linux para las pruebas realizadas sobre MySQL.

Todos los componentes software serán desarrollados utilizando el lenguaje de programación C# y el framework .NET 5 dado que es compatible con los sistemas operativos más populares, Windows, MacOs y Linux.

4.2.1 .NET 5

.NET 5 es un framework de desarrollo multiplataforma liberado de forma open source por Microsoft en noviembre de 2020.

Este framework provee de una interfaz de programación única que permite el desarrollo de componentes software compatible tanto con sistemas operativos de escritorio como pueden ser Windows, MacOS o Linux como por sistemas operativos móviles como Android e iOS.

.NET 5 ofrece herramientas para el desarrollo de servicios, interfaces de escritorio, páginas web, APIs, aplicaciones móviles entre otros proyectos populares.

Por la gran versatilidad que ofrece .NET 5, su alto rendimiento y su compatibilidad con diferentes sistemas operativos, ha sido elegido la pieza angular para el desarrollo software de este proyecto.

4.2.2 WorkerService

El framework de .NET 5 ofrece una plantilla software denominada WorkerService en su IDE de desarrollo Visual Studio.

Esta plantilla está pensada para el desarrollo de procesos que corren en segundo plano, ejecutando tareas que no requieren de una interfaz de usuario para funcionar.

Originalmente, esta plantilla estaba pensada para el desarrollo de lo que se denomina Servicio en el ecosistema de Windows, pero desde que .NET 5 fue liberado siendo compatible también con el sistema operativo Linux, nos permite el desarrollo de demonios compatibles con ambos sistemas operativos.

De esta forma, estaremos desarrollando un demonio que es compatible con los sistemas operativos más populares, haciendo posible que nuestro software de monitorización sea desplegado en cualquier entorno, independientemente del sistema operativo elegido para su instalación.

Esta característica nos es muy útil dado que, si fuera necesario, este proceso podría ser encapsulado dentro de un contenedor Docker para agilizar su instalación.

4.2.3 API

Una API es un tipo de interfaz software que permite la conexión entre dos o más ordenadores. Una API ofrece un conjunto de funciones y procedimientos que pueden ser utilizados por otras máquinas.

Para este proyecto, se ha decidido desarrollar una API de tipo REST. Una API REST, más conocida como API RESTful (RedHat, 2020) es aquella que cumple una arquitectura concreta que cumple, entre otros, los siguientes criterios:

- Cumple una arquitectura basada en cliente-servidor cuya comunicación es llevada a cabo utilizando el protocolo HTTP.
- La comunicación entre el cliente y el servidor es sin estado, es decir, ninguna información acerca del cliente es almacenada entre peticiones, cada petición es independiente y no está conectada.
- La información intercambiada entre los componentes está estandarizada y existe una interfaz común uniforme. Esto requiere de lo siguiente:

    - Los recursos requeridos son identificables y separables de las representaciones enviadas a los clientes.
    - Los recursos pueden ser manipulados por el cliente a través de las representaciones que ellos reciben porque la representación contiene suficiente información para realizarlo.
    - Los mensajes auto descriptivos devueltos al cliente contienen suficiente información para describir como el cliente debe procesarlo.
    - El hipertexto y la hipermedia está disponible, significando que, accediendo al recurso, el cliente debe de ser capaz de encontrar enlaces para encontrar las posibles acciones disponibles.
- El sistema separado en capas debe de ser invisible al usuario.

La comunicación entre el cliente y el servidor se suele realizar haciendo uso de algún estándar. Algunos de estos estándares más comunes suelen ser XML, Protobuf o JSON. En nuestro caso, hemos escogido este último al ser un formato que reduce bastante el tamaño del mensaje y además es de fácil lectura, facilitándonos el trabajo del desarrollo de la API.

El formato JSON es una notación utilizada por el lenguaje Javascript para serializar objetos, aún que hoy en día, esta notación ha sido extendida al resto de lenguajes y puede ser utilizada por estos.

Esta API será desarrollada utilizando ASP.NET core 5, el cual está basado en la versión del framework .NET 5 y será compatible para ser desplegada tanto en sistemas Linux como en sistemas Windows.

4.2.4 Docker

Docker (What is Docker?, s.f.) es una plataforma abierta para el desarrollo, despliegue y ejecución de aplicaciones. Docker permite al usuario la separación entre la aplicación y la infraestructura que la ejecuta, agilizando los procesos de desarrollo. Docker provee la habilidad de empaquetar y ejecutar la aplicación en un entorno aislado llamado contenedor.

Un servidor que tenga instalado un gestor de contenedores, puede ejecutar uno o varios contenedores al mismo tiempo. El sistema operativo que este ejecutando los

contenedores no afecta a la forma de ejecutarlo, siendo este uno de los motivos por el cual Docker es tan versátil y ágil.

Estos contenedores, además de estar aislados, son inmutables, es decir, cada vez que se reinicia un contenedor, este vuelve a su estado original.

En este proyecto, se ha escogido Docker para el despliegue de la base de datos MySQL. El motivo por el cual se ha escogido Docker para esta tarea es porque agiliza el despliegue del entorno de pruebas, donde todos los cambios realizados en este entorno pueden ser revertidos fácilmente redesplegando la imagen del contenedor de Docker.

4.2.5 MySQL

MySQL (What is MySQL?, s.f.) es una base de datos de tipo relacional desarrollada por la compañía Oracle. MySQL será utilizada en este proyecto como base de datos a monitorizar.

4.2.6 SQL Server

SQL Server es un motor de base de datos relacional desarrollado por Microsoft compatible con el sistema operativo Windows y Linux. SQL Server utiliza una versión modificada del lenguaje de consultas SQL denominado Transact SQL.

SQL Server se vende bajo licencia, pero hay una versión gratuita denominada SQL Server Express, el cual además de limitar el número de funciones disponibles, limita el tamaño de las bases de datos en 10GB.

Cuando se instala el IDE de Microsoft SQL Server, también se instala en el sistema la versión Express de SQL Server.

SQL Server se ha utilizado en este proyecto tanto para uso como sistema de base de datos a monitorizar como pieza de la arquitectura del software de monitorización. SQL Server será el servidor encargado de almacenar toda la información recopilada de los servidores a monitorizar.

Para poder realizar consultas al servidor de SQL Server, se ha utilizado una librería denominada Dapper que facilita el mapeo de objetos entre las tablas de la base de datos y los objetos desarrollados en C#.

4.2.7 Postman

Postman es un software especializado en ayudar al desarrollo de API Rests. Este software permite al usuario realizar consultas a una API Rest de una forma sencilla e intuitiva.

Estas pruebas formarán parte de los tests que se desarrollen para la API desarrollada para el proyecto.

Postman realiza peticiones HTTP a una dirección dada. Como valores de entrada, Postman no solo acepta la URL de la API a consultar, sino que nos permite además especificar que verbo HTTP utilizar junto a los parámetros de entrada. Estos parámetros pueden ser codificados utilizando diferentes notaciones, en nuestro caso utilizaremos JSON. Además, Postman nos permite especificar un método de autenticación a utilizar, algo realmente útil dado que la mayoría de las API Rests requieren de autenticación.

Postman nos permite la creación de grupos de llamadas, así como la creación de variables de entorno. Estas variables pueden ser utilizadas para, por ejemplo, parametrizar la URL de la API, de esta manera podemos utilizar la misma llamada para dos APIs diferentes, algo que adquiere gran utilidad cuando se trabaja con diferentes entornos como pueden ser de desarrollo, preproducción y producción.

Por estas herramientas que ofrece Postman, será utilizado en el desarrollo de la API de este proyecto.

5. Desarrollo del proyecto

Antes de comenzar a desarrollar el proyecto, es necesario elegir qué arquitectura de monitorización seguirá nuestro software ya que de esta dependerá el cómo debemos realizar el desarrollo.

Push vs pull

A pesar de la que existe una gran variedad de sistemas de monitorización, todos acaban siguiendo una de estas dos arquitecturas. Estas dos arquitecturas son muy parecidas en cuanto a la forma de monitorizar, pero la forma en la que se recopilan los datos afecta a las cualidades de estas.

Actualmente, las arquitecturas de tipo pull son las más comunes de encontrar. Esto puede ser debido a la gran popularidad que ha adquirido el software de monitorización Prometheus (Team, s.f.) debido al aumento de los sistemas que utilizan Kubernetes como base para su infraestructura.

Prometheus es una herramienta de monitorización de tipo pull sobre HTTP. Los sistemas que son monitorizados exponen una URL donde publican las métricas en texto plano utilizando un formato estandarizado para Prometheus y es el servidor central de Prometheus el encargado de consultar estas métricas y almacenarlas.

Por otro lado, tenemos los sistemas de monitorización push. Un ejemplo de estos sistemas es el de Graphite (Giedrius, s.f.). En los sistemas de tipo push, son los agentes instalados en cada recurso a monitorizar, los encargados de recopilar las métricas y mandarlas al sistema central.

Hay una gran discusión sobre qué sistema es mejor y cual es peor pero lo cierto es que a rasgos generales, ambos funcionan realmente bien y pueden ser totalmente válidos. A pesar de esto, se pueden extraer las siguientes conclusiones (Peter, s.f.):

	Push	Pull
Descubrimiento	Los agentes mandan las métricas tan pronto como estén disponibles, siendo el descubrimiento inmediato.	El descubrimiento requiere que el servidor central barra periódicamente el espacio de direcciones para encontrar nuevos agentes.
Escalabilidad	Escala de forma lineal dado que la carga de trabajo se reparte entre los agentes y el servidor central.	La carga del servidor central aumenta con el número de dispositivos monitorizados.
Seguridad	Por lo general, los agentes de tipo push son seguros contra ataques remotos ya que no escuchan conexiones de red.	El servicio es potencialmente vulnerable a ataques remotos o denegaciones de servicio.
Complejidad operacional	Configuración mínima requerida por los agentes: intervalo de tiempo y dirección del servidor central. Los firewalls han de configurarse para permitir el tráfico de forma unidireccional del agente al servidor central.	El servidor central ha de ser configurado con la lista de dispositivos a monitorizar, las credenciales para acceder a dichos dispositivos y las métricas a recolectar. Los firewalls han de configurarse de forma bidireccional.
Latencia	La baja carga de trabajo del servidor central permite la disminución del intervalo de recolección.	Los problemas de escalado impiden disminuir el intervalo de recolección. La comunicación bidireccional aumenta la

		latencia en la orden de recolección.
Flexibilidad	Relativamente inflexible debido a la forma en la que se configuran los agentes.	Flexible, el servidor central puede requerir cualquier métrica en cualquier momento.

Como podemos ver, por lo general, los sistemas push ofrecen bastantes ventajas frente a los sistemas pull pero el principal motivo por el cual hemos escogido este tipo de sistema para nuestro proyecto es por el apartado de seguridad.

Por lo general, los entornos donde se encuentran las bases de datos son altamente asegurados y aislados, donde solo se permite conexiones entrantes al puerto que escucha la base de datos y en ocasiones se permite cierto tráfico saliente, además de las respuestas de la base de datos, relacionado con el almacenamiento de copias de seguridad o actualizaciones del sistema.

Si hubiéramos escogido un sistema de tipo pull, tendríamos que haber añadido una regla al firewall que permitiera el tráfico de entrada desde el servidor central y una regla de salida para que el agente responda con las métricas. Utilizando un agente push solo nos hace falta permitir su tráfico saliente del entorno.

Otro motivo por el cual hemos escogido un sistema de tipo pull es porque es común tener entornos aislados para cada aplicación, es decir, si tenemos una aplicación A y otra aplicación B, estas aplicaciones suelen estar en entornos aislados y no interconectados, es decir, desde la infraestructura A es difícil llegar a la infraestructura si no es a través de Internet. En un modelo push, podemos tener nuestro servicio de monitorización C expuesto al público para que los agentes de monitorización de A y B sean capaces de publicar sus métricas en el servidor central a través de internet sin tener que exponer ningún puerto de estas dos infraestructuras.

Finalmente, se considera la latencia un factor importante debido a que, al tratarse de un sistema de monitorización de seguridad, es posible que sea detectar cambios y lanzar alertas tan pronto como sean detectadas las anomalías.

Arquitectura de la aplicación

Una vez elegido el modelo del sistema de monitorización que vamos a implementar, ya podemos diseñar la arquitectura de la aplicación. Como se ha comentado previamente, la arquitectura constará de un cliente y un servidor.

El cliente será un WorkerService que podrá ser instalado tanto en la propia máquina donde se encuentre instalado la base de datos como en una máquina externa que tenga conectividad con el servidor donde se encuentra instalado. Esto es de gran utilidad dado que existen diversidad de soluciones PaaS que ofrecen servicios de bases de datos como pueden ser Azure SQL Server, Amazon Web Services RDS o Cloud SQL de Google Cloud Platform.

En una arquitectura PaaS, un proveedor Cloud ofrece al cliente la posibilidad de acceder a un recurso, pero no al servidor que está ejecutando este servicio. En nuestro caso, podremos acceder al SQL Server o MySQL del proveedor Cloud, pero no al servidor donde se encuentra instalado ni a sus archivos de instalación. Por este motivo, es importante que el agente pueda ser instalado en una máquina externa a la base de datos.

Otra de las ventajas que aporta el poder instalarlo en una máquina externa es que, dado que toda la información monitorizada está pensada para ser accesible a través el lenguaje de consultas SQL, el agente de monitorización solo requiere de acceso a través del protocolo de SQL a la base de datos. Este acceso es el que también disponen de las aplicaciones que hacen uso de las bases de dato, de esta forma podemos desplegar nuestro agente en la zona DMZ donde se encuentre la aplicación que tenga acceso a las bases de dato, logrando no instalar ningún agente dentro de la red privada donde se encuentran las instancias de bases de datos.

Por otro lado, tenemos el servidor central el cual será una API REST. Este servidor central será el encargado de escuchar las peticiones de los agentes de monitorización para guardar nuevas métricas, las tratará y las almacenará en la base de datos de historial de cambios.

Finalmente, también se creará un portal web donde se mostrará aquella información relevante de cara a la monitorización como puede ser el histórico de versiones de cada base de datos monitorizada, el cambio que ha habido entre cada versión, que

características de seguridad cumple, que recomendaciones existe para cada base de datos y una visión general del estado de la infraestructura.

Teniendo estos componentes en cuenta, se ha diseñado el siguiente diagrama mostrado en la Ilustración 1 que resume la arquitectura de la aplicación utilizando como ejemplo un escenario donde disponemos de dos agentes publicando métricas y uno de ellos monitorizando dos bases de datos simultáneamente.

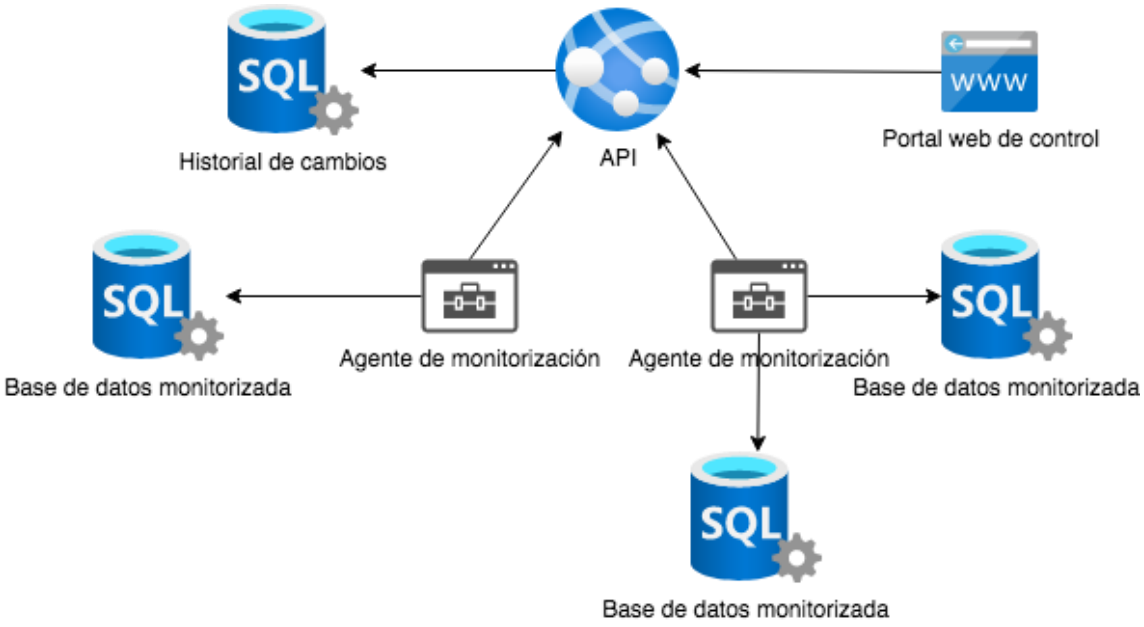

Ilustración 1. Diagrama de la arquitectura

Agente de monitorización

El primer componente que se ha desarrollado es el agente de monitorización. Este componente consistirá en un servicio de Windows o en un demonio de Linux que nada más arrancar leerá un fichero de configuración donde encontrará dos datos, en primer lugar, leerá la dirección URL donde se encuentra la API que recopilará los datos, en segundo lugar, leerá una lista de cadenas de conexión, las cuales corresponderán con todas aquellas bases de datos que se han de monitorizar.

Una vez el agente haya leído el fichero de configuración, el agente recorrerá la lista de cadenas de conexión, se conectará con cada una de las instancias de base de datos, recopilará información del esquema de base de datos, datos relacionados con las copias

de seguridad y finalmente recopilará toda la información disponible relacionada con los servicios activos y aquellos componentes que puedan ser relevantes de cara a la seguridad y las mejores prácticas que puedan ser aplicadas.

Una vez recopilada toda esta información, el agente la mandará a la API y esta se encargará de procesarla y almacenarla.

Información del esquema

Como se ha comentado en el apartado anterior, tras leer la configuración, lo primero que hará el agente será recopilar la información del esquema. La forma en la que lo hará dependerá del motor de base de datos que se esté utilizando, por este motivo es importante discernir contra qué tipo de base de datos se están consultando los datos.

Los datos que estamos tratando de recopilar acerca del esquema serán los necesarios para poder identificar cambios en las columnas, las tablas, las vistas, los procedimientos almacenados y los disparadores.

Bases de datos

En primer lugar, si no se ha especificado una base de datos en concreto, si no que se ha especificado una instancia de base de datos, lo que hará el agente es enumerar todas las bases de datos existentes en la instancia con el fin de recopilar todos los esquemas presentes.

Antes debemos matizar cual es la diferencia entre una base de datos y una instancia de base de datos. Una instancia de base de datos es aquel software que nos permite disponer de base de datos como puede ser Oracle, SQL Server o MySQL. Una base de datos es un grupo lógico que contiene un esquema, datos y metadatos. De esta forma, para disponer de una base de datos, necesitamos disponer de una instancia de base de datos.

Además, en una instancia de base de datos podemos encontrar una o más bases de datos. Por este motivo se enumeran todas las bases de datos existentes en la instancia.

Bases de datos en SQL Server

En SQL Server, es común encontrar numerosas bases de datos, sobre todo en las instancias con licencia Express dado que esta licencia solo permite hasta un máximo de 10GB por base de datos, por lo tanto, los usuarios suelen optar por dividir sus datos en diferentes bases con el fin de no alcanzar esta limitación.

Para listar estas bases de datos hemos hecho uso de la tabla Master.Sys.Databases, de la cual hemos extraído el ID de la base de datos y su nombre. Este ID nos será útil para identificar si una base de datos ha sido renombrada.

Las bases de datos en SQL Server, además, contienen unos grupos lógicos denominados esquemas (SQL Server Create Schema, s.f.) los cuales son colecciones de objetos de bases de datos que pueden ser tablas, vistas, disparadores, procedimientos almacenados o índices.

Una base de datos puede tener uno o varios esquemas y por defecto tiene un esquema denominado “dbo”. Esto es de gran utilidad dentro de una base de datos de cara a tener una mejor organización de los objetos. Por ejemplo, si tenemos una empresa dividida en departamentos, podemos crear un esquema por cada departamento y así separar los recursos de cada uno.

La convención de nombres en SQL Server sería “[esquema].[nombre]”, siendo por ejemplo el nombre de una tabla [dbo].[tabla]. Podemos disponer de objetos con nombre repetido mientras se encuentren en esquemas diferentes por ejemplo siguiendo con el caso de la empresa dividida en departamentos, podemos encontrar la tabla [Marketing].[Empleados] y [RRHH].[Empleados].

Bases de datos en MySQL

En MySql, si bien es también común encontrar numerosas bases de datos, también es muy común encontrar una única base de datos dado que muchos productos utilizan MySQL como método de almacenamiento y los usuarios que solo utilizan un único producto suelen tener esta única base de datos. Algunos ejemplos de productos software que utilizan MySQL son wordpress y prestashop.

Para enumerar las bases de datos existentes en el servidor, hemos hecho uso del comando “SHOW DATABASES;”, el cual devuelve un listado con los nombres de las bases de datos gestionadas por la instancia de MySQL. A diferencia de SQL Server, en MySQL no obtenemos un ID por base de datos, solo un nombre, por tanto, nos será imposible diferenciar si una base de datos ha sido renombrada o no.

Tablas

Las tablas son el elemento más importante dentro de una base de datos relacional. Una tabla (RDBMS Concept, s.f.) es una colección de elementos de información organizado en filas y columnas. Una tabla también puede ser considerada convenientemente como una relación aún que en el modelo relacional una tabla no puede contener duplicados mientras que en una tabla de una base de datos sí pueden encontrarse.

Una entrada en una base de datos se denomina tupla o fila. Por otro lado, tenemos una serie de atributos que son organizados en columnas. En el siguiente ejemplo podemos encontrar como sería una tabla que almacenara información de personas.

Id	**Nombre**	**Apellido**	**Edad**	**Nacionalidad**
1111111	Kaladin	Tormenta	20	Española
2222222	Brandon	Sanderson	45	Estadounidense

En el anterior ejemplo, encontramos la tabla persona con los atributos Id, Nombre, Apellido, Edad y Nacionalidad y dicha tabla contiene dos filas que corresponde a dos personas diferentes.

Dado que nuestro objetivo es adquirir el esquema y no los datos que contienen las tablas, obtendremos la información relacionada con las columnas de la tabla y no de sus filas.

Junto a la información relacionada con las tablas, recopilaremos información acerca de las columnas. Los datos que recopilaremos en relación con las columnas serán:

- Nombre

- Tipo de datos
- ¿Puede ser nulo?
- Valor por defecto

Además de esta información, podemos encontrar información extra tanto de las tablas como de las columnas pero que no hemos considerado prioritarias dado que con la información ya recopilada se podría volver a reconstruir una tabla eliminada.

Tablas en SQL Server

Como se ha comentado previamente, las tablas de SQL Server dispondrán de un esquema y de un nombre para su identificación. Hemos obtenido esta información utilizando la tabla de sistema INFORMATION_SCHEMA.TABLES para obtener un listado completo de todas las tablas filtrando para que sean del tipo 'BASE TABLE' para excluir aquellas tablas que sean de tipo vista.

Una vez obtenido el listado de las tablas existentes, necesitamos recopilar información de las columnas que de estas. Esta información podemos extraerla de la tabla INFORMATION_SCHEMA.COLUMNS.

Tablas en MySQL

A diferencia de SQL Server, las tablas en MySQL solo se identifican por un nombre y no por un esquema y nombre. Por este motivo, cuando enumeremos las tablas existentes solo obtendremos su nombre.

Para listar las tablas en MySQL hemos utilizado el comando "SHOW TABLES;" el cual devuelve una lista con los nombres de las tablas.

Para obtener información de la definición de la tabla, en MySQL podríamos haber optado por una estrategia diferente a la utilizada en SQL Server. Si bien con el anterior motor recopilábamos información de manera manual y selectiva, en MySQL existe una función que devuelve la definición completa de una tabla.

En SQL Server no estaba disponible esta funcionalidad, pero existen propuestas por la comunidad. Igualmente, estas propuestas han sido descartadas por no ser algo intrínseco

al motor de base de datos además de las razones de por qué no hemos utilizado esta función que si es nativa en MySQL.

En MySQL existe el comando “SHOW CREATE TABLE nombretabla;” el cual devuelve el script que se utilizaría para crear la tabla en el mismo estado que se encuentra actualmente incluyendo los tipos de datos, claves y todo tipo de restricciones que puedan contener las columnas.

Existen dos motivos principales por los cuales no hemos decidido utilizar esta estrategia y hemos seguido la misma utilizada en SQL Server.

En primer lugar, se ha decidido unificar la estrategia a seguir para todos los motores que se puedan monitorizar, siendo difícil adaptar la estrategia de generar el script de creación de tabla a SQL Server.

En segundo lugar, siendo esta la principal razón, es debido a que de esta forma disponemos de una forma estructurada de los datos. De la anterior forma, para una definición de tabla solo dispondríamos de una cadena de texto correspondiendo con el script, siendo difícil utilizarla para señalar los cambios que han variado entre versiones. De esta forma, disponemos de toda la información de la tabla de una forma estructurada, por ejemplo, disponiendo de una lista de columnas existentes fácilmente comparable entre versiones.

Por estos motivos, se ha decidido utilizar el comando “SHOW COLUMNS FROM nombretabla;” (MySQL Reference Manual: Show columns, s.f.). Este comando devuelve los datos que son de nuestro interés además de información extra como si la columna tiene algún tipo de restricción sobre cómo debe de comportarse cuando una referencia sea actualizada o borrada.

Una vez recopilada esta información, ya tendríamos toda la información requerida para las tablas de MySQL.

Vistas

Una vista (SQL Views, s.f.) es un tipo de tabla virtual. Al igual que una tabla, también dispone de filas y columnas, pero internamente estas filas están almacenadas en tablas y no en la propia vista. Una vista es una herramienta que nos permite abstraernos de cómo

están ordenados los datos en las tablas y nos permite crear nuestras propias tablas virtuales.

El uso de vistas no solo ayuda a simplificar la forma de consultar datos, sino que también es de gran importancia de cara a la seguridad. En el ámbito de la seguridad, las vistas se utilizan para otorgar permisos sobre ciertos datos a un usuario.

Pongamos un ejemplo, digamos que disponemos de una tabla denominada personas donde podemos encontrar información acerca de estas como puede ser su nombre, edad, nacionalidad y que, por cualquier motivo, tenemos una aplicación que solo ha de ser capaz de leer el nombre de una persona y su edad, pero no su nacionalidad.

La solución correcta a esta problemática sería crear una vista que contuviera las columnas nombre y edad y estuviera alimentada por las filas de la tabla personas y a continuación, otorgar permisos al usuario de la aplicación para que pueda realizar consultas sobre la vista creada.

Debido a que la definición de una vista puede ser compleja, se ha decidido optar por una estrategia donde se ha obtenido directamente el script que genera la vista en cuestión.

Vistas en SQL Server

Para recopilar todas las vistas creadas en la base de datos junto a su definición se ha consultado la tabla INFORMATION_SCHEMA.VIEW.

Vistas en MySQL

En MySQL, en primer lugar, se ha elaborado un listado de cada una de las vistas disponibles utilizando la orden “SHOW VIEWS;” y a continuación se ha utilizado el comando “SHOW CREATE VIEW nombrevista;” para acceder a la definición de las vistas.

Procedimientos

Los procedimientos (SQL Stored Procedures, s.f.), también llamado procedimientos almacenados en SQL Server, son un tipo de objeto en las bases de datos que nos permite almacenar código que puede ser reutilizado tantas veces como se desee.

Un procedimiento almacenado puede tanto realizar cambios en los datos existentes como realizar consultas sobre estos.

Dada la amplia libertad que existe a la hora de crear procedimientos, se va a seguir una estrategia similar a la utilizada con las vistas, se va a consultar el nombre junto a su definición.

Procedimientos en SQL Server

Para obtener la lista de procedimientos almacenados existentes junto a su definición en la base de datos se ha consultado la tabla INFORMATION_SCHEMA.ROUTINES.

Procedimientos en MySQL

En MySQL es posible acceder a la lista de procedimientos, pero para poder acceder a su definición, el motor ha de ser compilado con el flag de debugging activo.

Debido a que es muy difícil de ver y poco recomendable en un entorno productivo, estando desactivado en instancias PaaS como son RDS de Amazon Web Services o Cloud SQL de Google Cloud Platform, hemos decidido dejar esta funcionalidad excluida para MySQL.

En caso de que se deseara incluir, se puede acceder a la lista de procedimientos con el comando “SHOW PROCEDURE STATUS;” y a sus definiciones utilizando “SHOW CREATE PROCEDURE nombreprocedimiento;”.

Información de copias de seguridad

Una de las funcionalidades del software es notificar cuando ha pasado demasiado tiempo desde la última copia de seguridad, por este motivo el agente recopilará la fecha de cuando fue la última vez que se realizó una copia de seguridad.

Para obtener esta información, en SQL Server vamos a consultar las tablas MSDB.dbo.BackupSet y MSDB.dbo.BackupMediaFamily.

En MySQL no hemos encontrado ninguna tabla interna donde se almacene la fecha de la útlima vez que se realizó un Dump así que hemos deshabilitado esta función para este motor.

Información de seguridad

Finalmente, los últimos datos que recopilará el agente antes de enviarlos a la API serán aquellos que puedan ser relevantes de cara a tener una configuración robusta de cara a la seguridad.

Información de seguridad en SQL Server

Usuarios

En primer lugar, en SQL Server se recopilará la información de todos los usuarios, así como la fecha de la última vez que se cambió la contraseña, de esta forma podremos aplicar políticas de renovación de contraseña.

En segundo lugar, se comprobará el tipo de login que tiene el usuario SA (adminstrador) de la instancia, siendo un login de tipo Active Directory el recomendado para este tipo de usuarios o si fuera posible, desactivar esta cuenta.

Xp_cmdshell

Xp_cmdshell es una función de SQL Server que permite ejecutar comandos PowerShell en el servidor donde esté instalado la instancia de SQL. Si esta instancia está ejecutándose bajo una cuenta de administrador, los comandos serán ejecutados bajo ese nivel de privilegios.

Esta función rara vez es utilizada en un contexto habitual de SQL y por tanto ha de desactivarse por el alto peligro que supone, permitiendo la ejecución remota de código a través de una conexión SQL.

A pesar de que esta función se puede deshabilitar, se puede volver a habilitar y no hay forma de desactivarla de forma indefinida en SQL Server.

Transparent Data Encryption

SQL Server dispone de una funcionalidad nativa que permite encriptar tanto los ficheros de datos como de logs. Desafortunadamente, esta funcionalidad está deshabilitada en las versiones Express pero si está disponible en otras licencias de pago como puede ser la Enterprise.

Además, las copias de seguridad que se generen de estas bases de dato encriptadas también estarán encriptadas. Por este motivo se recomienda activarlo allí donde sea posible.

SQL Dinámico

SQL dinámico (SQL Server Dynamic SQL , s.f.) es una forma de realizar consultas SQL donde la consulta es construida utilizando una lógica de programación y a continuación utilizando una función de SQL que ejecuta código SQL.

Este tipo de consultas están completamente desaconsejadas porque pueden ser vulnerables a inyección SQL por la forma en la que se construyen las sentencias. Por este motivo se va a tratar de identificar aquellos procedimientos almacenados que hagan uso de esta funcionalidad.

Información de seguridad en MySQL

Usuarios

Se va a realizar un inventario de los usuarios que hay en el servidor, así como de los permisos que tienen estos.

A diferencia de SQL Server, MySQL no nos ofrece la fecha de la última vez que se cambió la contraseña así que esta funcionalidad no será añadida para este motor de base de datos.

En MySQL existe la posibilidad de que existan usuarios sin contraseña y es totalmente desaconsejable a nivel de seguridad así que se realizará un control de si existen usuarios con esta característica.

Transparent Data Encryption

MySQL también dispone de forma nativa de encriptación de los datos “at rest” (InnoDB Data Encryption, s.f.) y al igual que en SQL Server, es recomendable activarlo siempre que sea posible.

Base de datos TEST

Por defecto, algunas versiones de MySQL traen una base de datos de ejemplo que es accesible para todos los usuarios. El acceso no autorizado a este recurso podría suponer un problema ya que un atacante podría hacer uso de esta base de datos, mermando la capacidad de respuesta de la base de datos y pudiendo generar un ataque de denegación de servicio.

Por este motivo es imprescindible eliminar esta base de datos ya que carece de utilidad más allá de servir como ejemplo.

API

La API será la encargada de recibir todos los datos generados por los agentes, así como de realizar las comprobaciones pertinentes cada vez que se almacenen datos.

Histórico de esquemas

Cada vez que la API reciba una llama que incluya un esquema de base de datos, generará un resumen HASH utilizando el algoritmo SHA256. Hemos optado por este algoritmo ya que aporta suficiente longitud de bits como para que sea difícil que dos esquemas diferentes posean la misma firma.

Una vez generado el resumen, la API obtendrá la última versión del esquema de dicha base de datos que tenga almacenado internamente y comprobará las firmas, si estas son

diferentes, la API insertará este nuevo esquema como una nueva versión de la base de datos.

En caso de que las firmas coincidan, simplemente actualizará la fecha de la última vez que se recolectó dicho esquema.

Histórico de bases de datos

Cuando la API reciba una petición de almacenar una fecha de la última vez que se realizó una copia de seguridad, comprobará si esta fecha está dentro de los valores aceptados. Internamente, la API dispone de un parámetro configurable donde podemos especificar cuando queremos que salte una alerta.

Por ejemplo, si especificamos 3 meses de límite y recibimos que la última copia de seguridad fue realizada hace 4 meses, la API lanzará una alerta sobre esta base de datos.

Histórico de seguridad

Si la API recibe una petición de almacenamiento de histórico de seguridad, en primer lugar, comprobará si la configuración actual ha variado respecto a la última configuración conocida.

En caso de que haya variado, se generarán una serie de alertas en función a aquellos valores que puedan causar problemas, indicando con una breve descripción el por qué pueden suponer un problema.

En caso de que no haya variado, simplemente se actualizará la fecha de última recolección para dicha configuración.

Consulta de datos

Como se comentó previamente, la API también dispondrá de llamadas que permitirán consultar todos los datos almacenados en la base de datos de históricos.

Estas llamadas serán utilizadas por el portal web que será explicado a continuación.

Portal Web

Para poder explotar los datos recopilados por la aplicación, se ha decidido utilizar un portal web. Este tipo de estrategia también es utilizada por otras aplicaciones de monitorización como pueden ser Nagios (Nagios Core Frontends, s.f.) y Prometheus (Use the Prometheus web UI, s.f.).

El portal web estará desarrollado en Blazor y dispondrá de los siguientes apartados.

Panel de control principal

En primer lugar, al acceder al portal web se cargará un panel de control donde se mostrará información acerca de todos los recursos monitorizados.

Se mostrará un listado de todas las bases de datos para las cuales se tenga datos, la fecha de la versión de esquema que estén ejecutando, la fecha de la última copia de seguridad y el número de alertas que tenga activas la base de datos.

A través de este portal, podremos acceder a otros menús.

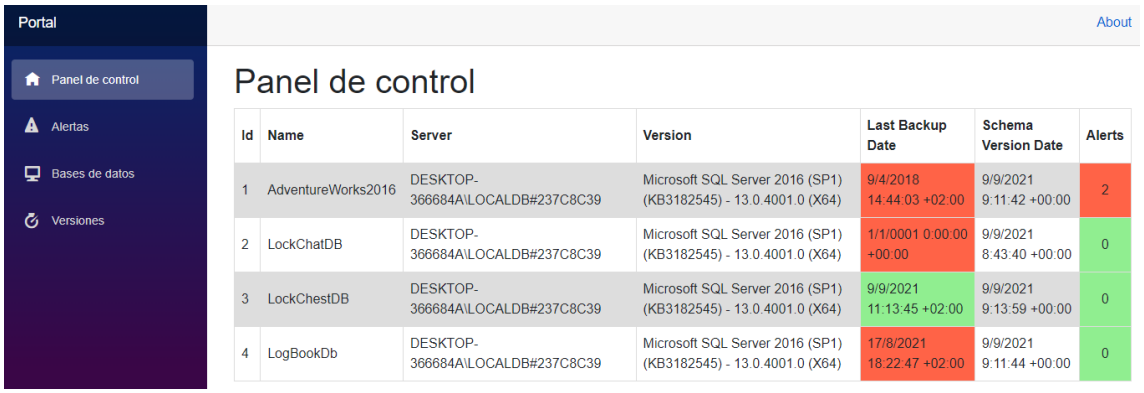

Ilustración 2 Panel de control

Listado de alertas

En este apartado del portal web, podremos consultar las alertas actuales vigentes y pasadas.

Además, podremos marcar una alerta como vista, para evitar que figure en el portal principal como una alerta presente en la actual versión de la base de datos.

Este listado estará mostrado en una tabla donde se podrá ordenar por base de datos afectada y versión.

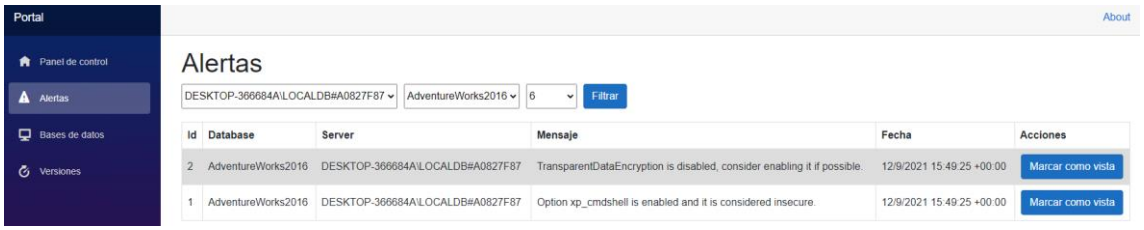

Ilustración 3 Panel de alertas

Información sobre las bases de datos

En este apartado se mostrará información en referente a la configuración de las alertas como cuales están activas o cuáles son sus parámetros aceptados, por ejemplo, en el caso de copias de seguridad, cuantos días son tolerables.

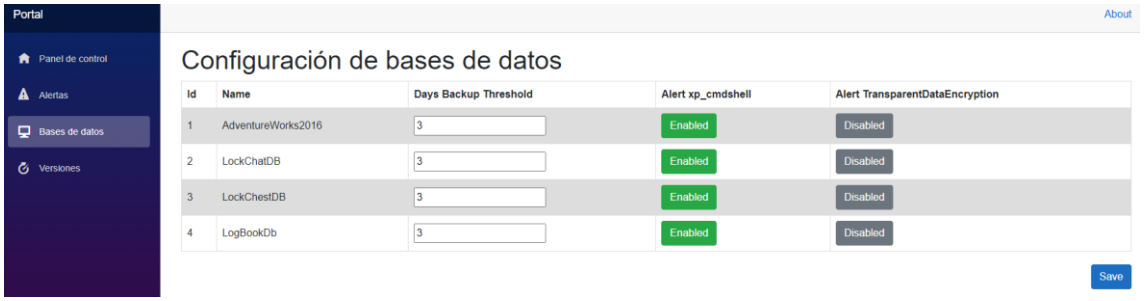

Ilustración 4 Panel de configuración de bases de datos

Comparativa entre esquemas

En este apartado, podremos seleccionar dos versiones del esquema de bases de datos y generar un informe con todos los cambios que hayan surgido entre estas.

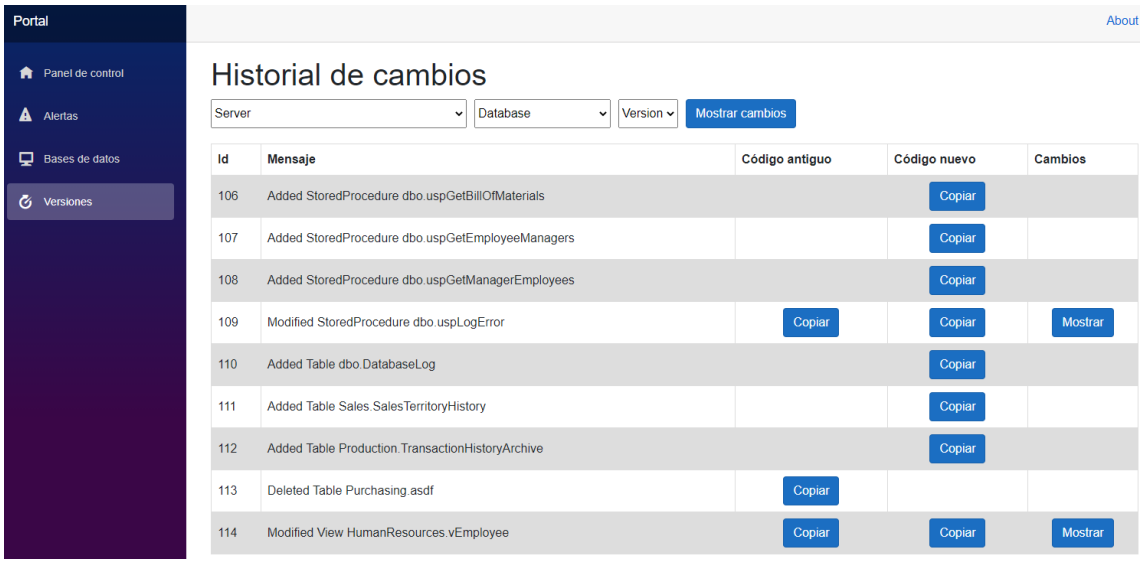

Ilustración 5 Panel de historial de cambios

Además, dependiendo del tipo de cambio, tendremos diferentes opciones disponibles. En cambio, de ser un objeto nuevo añadido, podremos ver su código al igual que si es un objeto borrado, podremos ver el código antiguo.

Finalmente, si es una modificación de un objeto, podremos ver que cambios se han realizado.

Ilustración 6 Menú de comparativa entre dos versiones de un objeto del esquema de base de datos

6. Conclusiones

Llegados a este punto, se ha conseguido desarrollar una muestra del software que se deseaba conseguir con este proyecto de monitorización de bases de datos, consiguiendo así cumplir los objetivos descritos al inicio del proyecto.

Este proyecto ha sido diseñado originalmente desde el punto de vista de la extensión, tratando de homogenizar lo máximo posible la monitorización, tratando de obviar las diferencias que existen entre los diferentes motores de bases de datos. Si bien no siempre ha sido posible, se ha tratado de utilizar aquellos elementos que eran comunes en estas con el fin de unificar toda la monitorización en un único proyecto.

El haber desarrollado un software multiplataforma y poco intrusivo, nos ha permitido diseñar un sistema que puede ser aceptable y recomendable en entornos productivos sin comprometer la seguridad de estos. Además, la elección de un servicio que consultara datos simplemente a través del lenguaje de consultas SQL, ha permitido que este sea compatible con entornos PaaS, donde esta es la única forma que se dispone de comunicarse con la plataforma.

Todo este desarrollo partía de un área donde apenas existían softwares que solventaran la problemática propuesta en este proyecto, demostrando así que es posible el diseño y la implementación de un software capaz de monitorizar y registrar cambios en los esquemas de las bases de datos y sus configuraciones de seguridad.

A pesar de todo lo conseguido con este proyecto, se han encontrado una serie de problemas a lo largo del desarrollo, así como se han identificado una serie de futuras mejoras que serán propuestas al final de este documento.

Problemas encontrados

A lo largo de este proyecto, se han encontrado una serie de problemas que, si bien no han sido limitantes para este proyecto dado su bajo alcance, si pueden llegar a limitarlo en un futuro y o dificultar su continuidad.

El principal problema encontrado son las diferencias existentes entre los motores de bases de datos. Si bien todos los motores parten de una misma premisa como es el uso

del modelo relacional y el uso del lenguaje de consultas SQL, estos acaban implementando ciertas pequeñas diferencias que en conjunto son significativas.

Esto hemos podido encontrarlo en el proyecto con algunos ejemplos como eran el uso de esquemas en SQL Server. En MySQL, cualquier objeto dentro de una base de datos puede ser identificado solo con su nombre, esto no ocurre en SQL Server, donde hace falta también un nombre de esquema, siendo posible que dentro de la misma base de datos existan objetos con el mismo nombre, pero con distinto esquema.

Otra diferencia notable encontrada ha sido las diferentes variantes del lenguaje SQL existentes (Rathee, s.f.). SQL Server utiliza su propia variante de SQL denominada Transact-SQL al igual que Oracle utiliza Oracle SQL.

Esta diferencia no ha afectado del todo al proyecto dado que las fuentes de información utilizadas para recopilar los datos necesarios dependían del motor de base de datos consultado, impidiendo así la reutilización de consultas. De todas formas, si ha significado una dificultad por el hecho de necesitar conocer las peculiaridades de cada versión del lenguaje.

La forma que tienen los motores de bases de datos de utilizar los ficheros de configuración ha sido otro limitante en este proyecto. Si bien en SQL Server, la mayoría de los parámetros de configuración eran accesibles a través del lenguaje de consultas SQL, esto no era siempre posible en MySQL, donde algunos de estos parámetros estaban configurados en un fichero en el sistema de archivos, siendo inaccesible a través de SQL.

Dada que la filosofía de este software es la de no acceder a los ficheros del sistema, ha sido imposible recopilar esta información, siendo un limitante de que elementos podrían ser monitorizados.

A pesar de estos problemas encontrados, en el siguiente apartado se ha realizado una enumeración de posibles propuestas para garantizar la continuidad del proyecto.

Trabajo futuro

En este apartado se elaborará una lista de aquellas posibles mejoras para el proyecto ya que la monitorización de seguridad de bases de datos es un terreno relativamente poco explorado y está abierto a su desarrollo.

Extensión a otros motores de bases de datos

La primera propuesta de trabajo futuro es quizá la más obvia y es extrapolar el sistema actual de monitorización a otros motores de bases de datos, principalmente a Oracle SQL y PostgreSQL por ser dos de los motores con más cuota de mercado (Most popular databases in 2020, s.f.) junto a los utilizados en este proyecto, SQL Server y MySQL.

Otra opción que podría ser interesante es tratar de no limitar este sistema de monitorización a bases de datos OLTP, las cuales son las todas mencionadas en este trabajo, y considerar las bases de datos OLAP. Por ejemplo, SQL Server es una base de datos tipo OLTP y SQL Server Analysis Services es su equivalente en OLAP.

Las bases de datos OLAP son relativamente parecidas a las bases de datos OLTP, pero estas están pensadas para el análisis de datos. Al igual que las bases de datos OLTP, las de tipo OLAP también tienen tablas y esquema de datos, al igual que es de suma importancia proteger su contenido por contener una alta cantidad de datos.

Análisis de logs

Otra opción que sería interesante valorar de cara a la monitorización con vistas a la seguridad es el análisis de logs, concretamente en búsqueda de intentos de SQL Injection.

Esta estrategia sería variable en función del motor de base de datos a monitorizar puesto que los logs funcionan diferentes. En el caso de MySQL, bastaría con activar la opción de logs, preferiblemente con salida a una tabla con el fin de garantizar que se puede monitorizar solamente utilizando SQL y realizar búsquedas de patrones utilizados en ataques de este tipo como la cadena “OR 1=1”.

En el caso de SQL Server sería más complejo puesto que la forma más optima de realizar esto es a través de los Extended Events o XEvents. Los Extended Events son

archivos de log que contienen información acerca de las consultas SQL realizadas con contra el servidor, así como los parámetros que se han utilizado, el usuario que las ejecuta, los tiempos y los planes de ejecución. A través de estos ficheros podríamos obtener los parámetros utilizados en las consultas con el fin de buscar patrones al igual que se ha propuesto con MySQL. Una desventaja de esta solución es que perdemos la característica de monitorizar solamente utilizando SQL, al tener que monitorizar también archivos de log.

Dockerizar la solución

Desde el principio del proyecto, esta solución software ha sido diseñada con la posibilidad de que fuera desplegado utilizando contenedores.

El motivo de este es porque Docker es una solución muy popular hoy en día como se ha explicado previamente, ya que permite el despliegue uniforme de aplicaciones independientemente de la infraestructura que las ejecute.

Dado que todos los componentes software son compatibles con Linux, desplegar esta solución utilizando contenedores no debería suponer un problema.

Cabe mencionar que la base de datos no es recomendable que se encuentre en un contenedor dado que estos están pensados para ser ligeros y volátiles, algo totalmente opuesto a lo que es una base de datos.

Análisis de patrones y comportamientos

Otra opción muy interesante para valorar es añadir al software la capacidad de analizar patrones y comportamientos de los usuarios que hacen uso de la base de datos.

Por un lado, se podrían detectar ataques a través de las anomalías en las consultas, si por ejemplo una tabla se consulta poco y de repente, su uso pasa a ser intensivo, puede significar que nos encontramos ante un ataque.

Por otro lado, este tipo de análisis es interesante porque pueden detectar configuraciones erróneas en los permisos configurados para cada usuario. Por ejemplo, si disponemos de un usuario que solamente realiza consultas en el servidor, lo más probable es que no

requiera de permisos para poder escribir en la base de datos. De la misma forma, es absurdo encontrar que varios usuarios dentro de una base de datos contienen permisos de administrador cuando lo habitual es que estos permisos estén limitados a pocos usuarios.

Bibliografía

Giedrius. (n.d.). *Push vs Pull in monitoring systems*. Retrieved from https://giedrius.blog/2019/05/11/push-vs-pull-in-monitoring-systems/

InnoDB Data Encryption. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/innodb-data-encryption.html

MariaDB. (n.d.). Retrieved from https://en.wikipedia.org/wiki/MariaDB

Most popular databases in 2020. (n.d.). Retrieved from https://www.eversql.com/most-popular-databases-in-2020/

MySQL Reference Manual: Show columns. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/show-columns.html

Nagios Core Frontends. (n.d.). Retrieved from https://www.nagios.org/downloads/nagios-core-frontends/

Peter. (n.d.). *sFlow*. Retrieved from https://blog.sflow.com/2012/08/push-vs-pull.html

Rathee, K. (n.d.). *The many flavours of SQL*. Retrieved from https://towardsdatascience.com/the-many-flavours-of-sql-7b7da5d56c1e

RDBMS Concept. (n.d.). Retrieved from https://www.studytonight.com/dbms/rdbms-concept.php

RedHat. (2020, Mayo 8). *What is a REST API?* Retrieved from https://www.redhat.com/en/topics/api/what-is-a-rest-api

Software development process. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Software_development_process

SQL Server Create Schema. (n.d.). Retrieved from https://www.sqlservertutorial.net/sql-server-basics/sql-server-create-schema/

SQL Server Dynamic SQL . (n.d.). Retrieved from https://www.sqlservertutorial.net/sql-server-stored-procedures/sql-server-dynamic-sql/

SQL Stored Procedures. (n.d.). Retrieved from https://www.w3schools.com/sql/sql_stored_procedures.asp

SQL Views. (n.d.). Retrieved from https://www.geeksforgeeks.org/sql-views/

Team, O. (n.d.). *Why is Prometheus monitoring the Industry Standard?* Retrieved from https://opsani.com/blog/why-is-prometheus-monitoring-the-industry-standard/

Use the Prometheus web UI. (n.d.). Retrieved from https://docs.mirantis.com/mcp/q4-18/mcp-operations-guide/lma/use-prometheus-web-ui.html

What is Docker? (n.d.). Retrieved from https://docs.docker.com/get-started/overview/

What is MySQL? (n.d.). Retrieved from https://www.talend.com/resources/what-is-mysql/

Wright, S. (2020, Mayo 4). *SentryOne*. Retrieved from How to track SQL Server configuration settings with SentryOne Document: https://www.sentryone.com/blog/how-to-track-sql-server-configuration-settings-with-sentryone-document