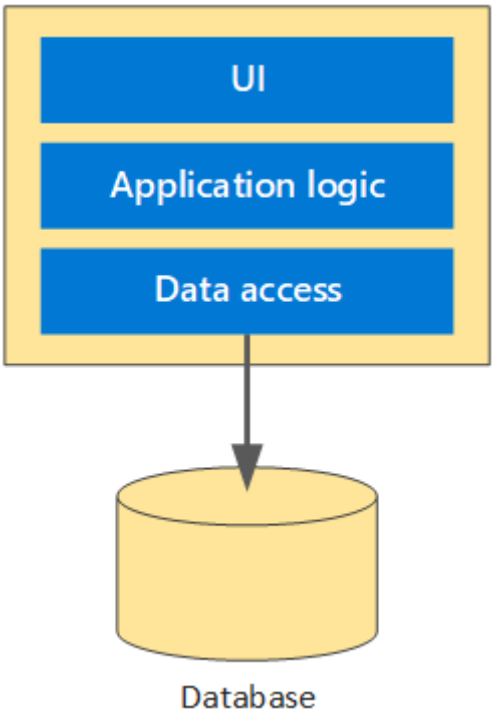
Conversión de aplicaciones monolíticas en microservicios mediante el diseño basado en dominios

Artículo de autoría de Microsoft Inc.

En este artículo se describe cómo usar el diseño basado en dominios (DDD) para migrar una aplicación monolítica a microservicios.

Normalmente, una aplicación monolítica es un sistema de aplicaciones en el que todos los módulos relevantes se empaquetan como una única unidad de ejecución que se puede implementar. Por ejemplo, se podría tratar de una aplicación web Java (WAR) que se ejecuta en Tomcat o una aplicación ASP.NET que se ejecuta en IIS. Una aplicación monolítica típica usa un diseño de capas independientes para la interfaz de usuario, la lógica de la aplicación y el acceso a los datos.



Estos sistemas empiezan con un tamaño pequeño, pero tienden a crecer con el tiempo para satisfacer las necesidades empresariales. En algún momento, a medida que se agregan nuevas características, una aplicación monolítica puede empezar a sufrir los problemas siguientes:

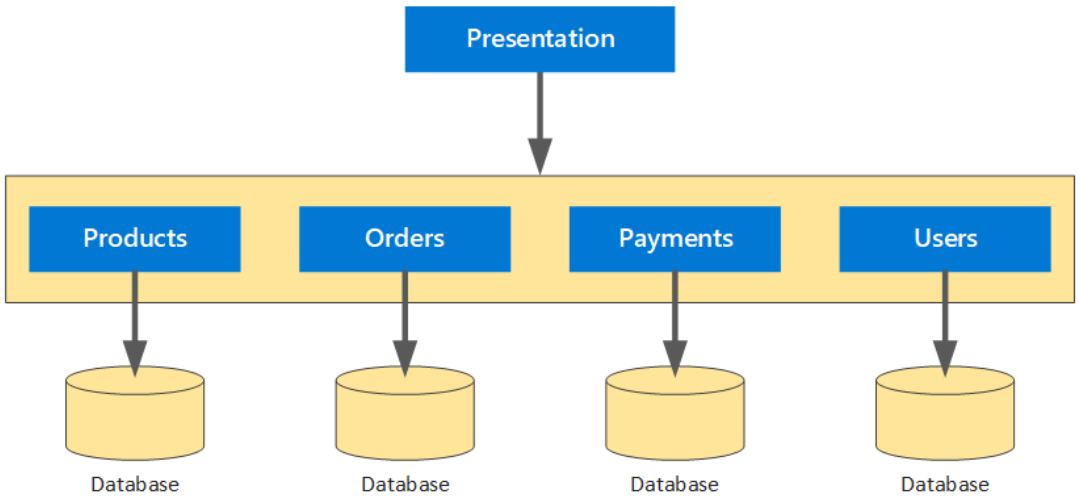
- Los elementos individuales del sistema no se pueden escalar de manera independiente, porque están estrechamente acoplados.
- Resulta complicado mantener el código, debido al acoplamiento estrecho y a las dependencias ocultas.
- Las pruebas se hacen más difíciles, lo que aumenta la probabilidad de que aparezcan vulnerabilidades.

Estos problemas se pueden convertir en un obstáculo para el crecimiento y la estabilidad futuros. Los equipos se vuelven precavidos a la hora de realizar cambios, en especial si los desarrolladores originales ya no trabajan en el proyecto y los documentos de diseño son escasos o no están actualizados.

A pesar de estas limitaciones, un diseño monolítico puede tener sentido como punto de partida para una aplicación. Las aplicaciones monolíticas suelen ser la ruta más rápida a la creación de una prueba de concepto o un producto viable mínimo. En las primeras fases del desarrollo, las aplicaciones monolíticas tienden a ser:

- Más fáciles de compilar, ya que solo hay una base de código compartido.
- Más fáciles de depurar, porque el código se ejecuta en un único proceso y espacio de memoria.
- Más fáciles de entender, porque hay menos elementos móviles.

Pero a medida que crece la complejidad de la aplicación, estas ventajas pueden desaparecer. A menudo, las grandes aplicaciones monolíticas son cada vez más difíciles de compilar, depurar y entender. Llega un momento en el que los problemas superan a las ventajas. Este es el punto en el que puede tener sentido migrar la aplicación a una arquitectura de microservicios. A diferencia de las aplicaciones monolíticas, los microservicios suelen ser las unidades de ejecución descentralizadas y de acoplamiento flexible. En el diagrama siguiente se muestra una arquitectura típica de microservicios:



Para la migración de una aplicación monolítica a un microservicio se necesita tiempo y una inversión significativa para evitar errores o sobrecostos. Para asegurarse de que cualquier migración sea correcta, es conveniente comprender tanto las ventajas como los desafíos que aportan los microservicios. Entre las ventajas se incluye lo siguiente:

- Los servicios pueden evolucionar de forma independiente en función de las necesidades de los usuarios.
- Los servicios se pueden escalar de manera independiente para satisfacer la demanda de los usuarios.
- Con el tiempo, los ciclos de desarrollo se vuelven más rápidos, ya que las características se pueden comercializar antes.
- Los servicios están aislados y son más tolerantes a los errores.
- Un único servicio que genere un error no hará que toda la aplicación deje de funcionar.
- Las pruebas son más coherentes, mediante el desarrollo basado en el comportamiento.

Para obtener más información sobre las ventajas y los desafíos de los microservicios, vea Estilo de arquitectura de microservicios.

Aplicación del diseño basado en dominios

Cualquier estrategia de migración debe permitir a los equipos refactorizar de forma incremental la aplicación en servicios más pequeños, a la vez que se sigue proporcionando la continuidad del servicio a los usuarios finales. Este es el enfoque general:

- Se deja de agregar funcionalidad al monolito.
- Se separa el front-end del back-end.
- Se descompone y desacopla la aplicación monolítica en una serie de microservicios.

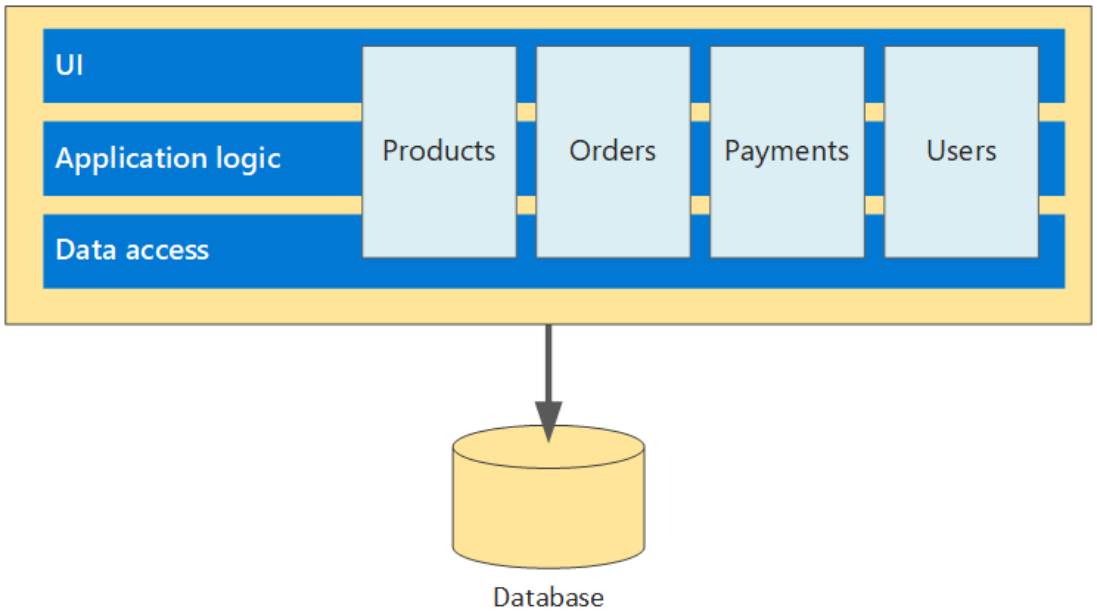
Para facilitar esta descomposición, un enfoque de desarrollo de software viable consiste en aplicar los principios del diseño basado en dominios (DDD).

El diseño basado en dominios (DDD) es un enfoque de desarrollo de software presentado por primera vez por Eric Evans. Para DDD se necesita un buen conocimiento del dominio para el que se va a escribir la aplicación. El conocimiento necesario del dominio para crear la aplicación reside en las personas que lo entienden, es decir, los expertos del dominio.

El enfoque de DDD se puede aplicar de forma retroactiva a una aplicación existente, como una manera de empezar a descomponerla.

1. Comience con un *lenguaje omnipresente* , un vocabulario común que se comparta entre todas las partes interesadas.
2. Identifique los módulos relevantes de la aplicación monolítica y aplíqueles el vocabulario común.
3. Defina los modelos de dominio de la aplicación monolítica. El modelo de dominio es un modelo abstracto del ámbito empresarial.
4. Defina *contextos delimitados* para los modelos. Un contexto delimitado es el límite dentro de un dominio donde se aplica un modelo de dominio concreto. Aplique límites explícitos con modelos y responsabilidades claramente definidos.

Los contextos delimitados identificados en el paso 4 son candidatos para la refactorización en microservicios más pequeños. En el diagrama siguiente se muestra la aplicación monolítica existente con los contextos delimitados superpuestos:

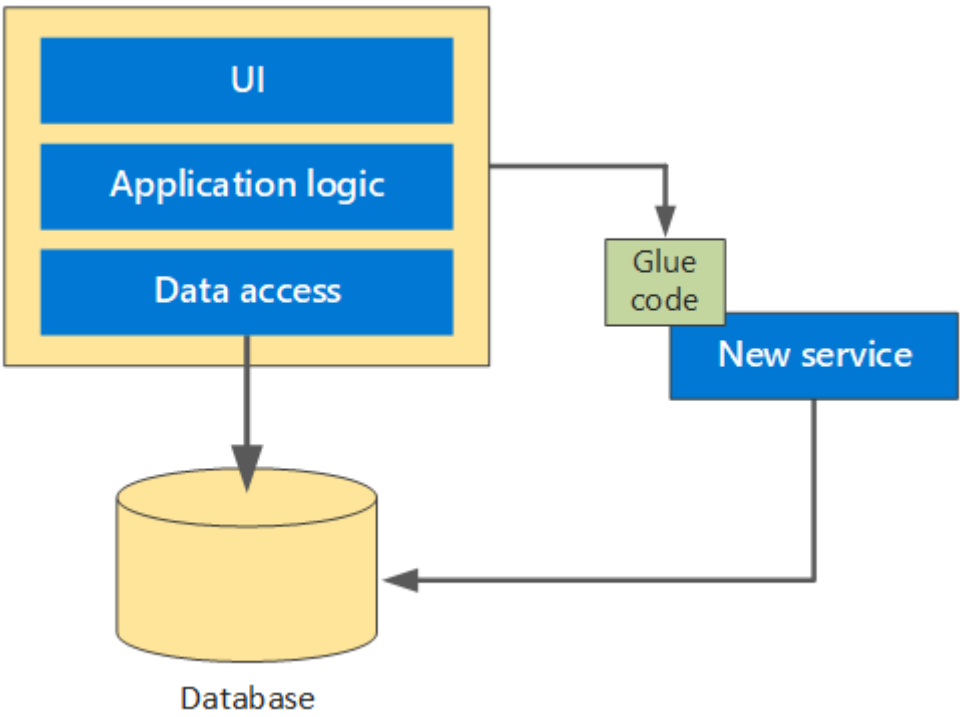


Para obtener más información sobre el uso de un enfoque de DDD para arquitecturas de microservicios, vea Uso de análisis de dominios para modelar microservicios.

Uso de código de adherencia (capa para evitar daños)

Aunque este trabajo de investigación se realiza para inventariar la aplicación monolítica, se pueden agregar funciones nuevas si se aplican los principios de DDD como servicios independientes. El "código de adherencia" permite a la

aplicación monolítica realizar llamadas del proxy al nuevo servicio para obtener funciones nuevas.



El código de adherencia (patrón de adaptador) actúa eficazmente como una capa para evitar daños, y garantiza que el nuevo servicio no se contamine por los modelos de datos que necesita la aplicación monolítica. El código de adherencia ayuda a mediar las interacciones entre los dos y garantiza que solo se pasen los datos que necesita el nuevo servicio para habilitar la compatibilidad.

A través del proceso de refactorización, los equipos pueden inventariar la aplicación monolítica e identificar candidatos para la refactorización en microservicios, al tiempo que se establece una nueva funcionalidad con nuevos servicios.

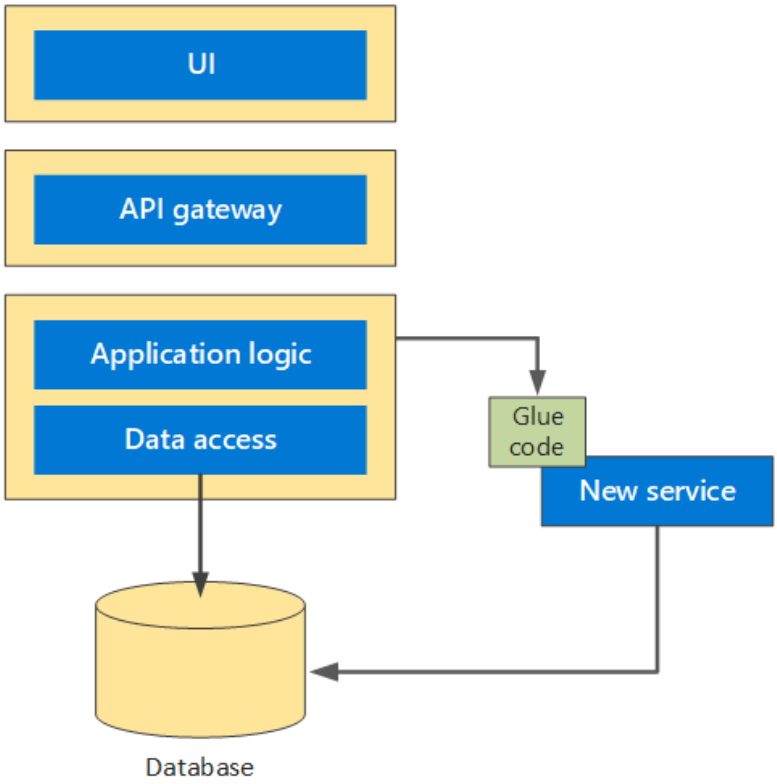
Para obtener más información sobre las capas para evitar daños, vea Patrón de capas para evitar daños.

Creación de una capa de presentación

El siguiente paso consiste en separar la capa de presentación de la capa de back-end. En una aplicación de n niveles tradicional, la capa de aplicación (empresarial) tiende a incluir los componentes básicos de la aplicación y que tienen lógica de dominio. Estas API generales interactúan con la capa de acceso a datos para recuperar datos almacenados de una base de datos. Estas API

establecen un límite natural en la capa de presentación y ayudan a desacoplarla en un espacio de aplicación independiente.

En el diagrama siguiente se muestra la división de la capa de presentación (IU) de las de lógica de la aplicación y acceso a datos.

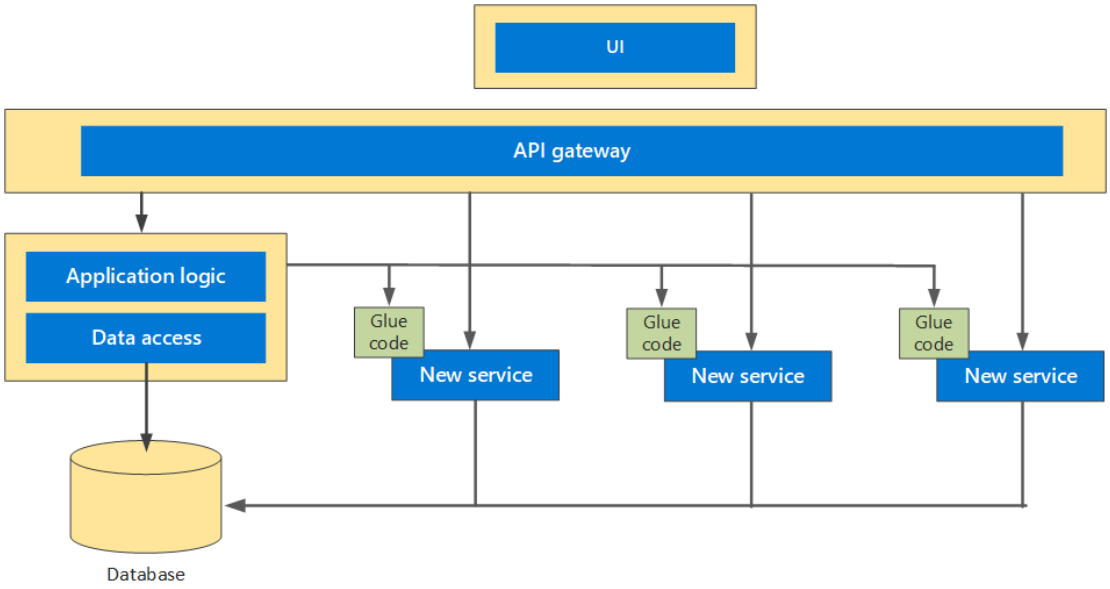


En este diagrama también se presenta otra capa, la puerta de enlace de API, que se encuentra entre la capa de presentación y la lógica de la aplicación. La puerta de enlace de API es una capa de fachada que proporciona una interfaz coherente y uniforme para que interactúe con la capa de presentación, a la vez que permite que los servicios de nivel inferior evolucionen de forma independiente, sin que ello afecte a la aplicación. La puerta de enlace de API puede usar una tecnología como Azure API Management y permite que la aplicación interactúe con un estilo RESTful.

La capa de presentación se puede desarrollar en cualquier lenguaje o marco con el que tenga experiencia el equipo, como una aplicación de página única o una aplicación MVC. Estas aplicaciones interactúan con los microservicios a través de la puerta de enlace, mediante llamadas HTTP estándar. Para obtener más información sobre las puertas de enlace de API, vea Uso de puertas de enlace de API en microservicios.

Inicio de la retirada de la aplicación monolítica

En esta fase, el equipo puede empezar a diseccionar la aplicación monolítica y extraer lentamente los servicios establecidos por sus contextos delimitados en un conjunto de microservicios propio. Los microservicios pueden exponer una interfaz RESTful para que interactúe con la capa de aplicación, a través de la puerta de enlace de API, con código de adherencia para comunicarse con la aplicación monolítica bajo determinadas circunstancias.



A medida que siga diseccionando la aplicación monolítica, llegará el momento en que ya no es necesario que exista y se hayan extraído correctamente los microservicios. En este momento, se puede quitar la capa para evitar daños (el código de adherencia) sin ningún riesgo.

Este enfoque es un ejemplo del Patrón Strangler y permite una descomposición controlada de una aplicación monolítica en un conjunto de microservicios. Con el tiempo, a medida que la funcionalidad existente se transfiera a los microservicios, la aplicación monolítica se reducirá en tamaño y complejidad, hasta el punto en que deje de existir.